Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live

Petr Kouznetsov, Max Planck Institute for Software Systems

Join work (in progress!) with:

Bobby Bhattacharjee, Univ. Maryland
Rodrigo Rodrigues, INESC-ID and Tech. Univ. Lisbon

FuDiCo III, June 2007

Big picture

Choosing an adequate model to implement a system is crucial

- **Optimistic**: the system is very efficient but likely to fail
- **Conservative**: the system is very robust but inefficient (or impossible to implement)

How to find a good balance?
Prepare for the worst and hope for the best

- How good is the best and how bad is the worst?
  ✓ Best case – failures are few
  ✓ Worst case – almost everything can be faulty

- What do we mean by “prepare” and “hope”?
  ✓ The system is very efficient in the best case
  ✓ The system never produces inconsistent output (even in the worst case), but …

✓ May become unavailable in the (rare) “intermediate” case

The context: clients and services

- A client issues a request to a service
- The service executes the request and returns a response to the client
The fault-tolerant computing challenge

- Even if some system components (clients or service units) fail, the correct clients still get something useful from the service.
- Failures can be Byzantine: a component can arbitrarily deviate from its expected behavior.

![Diagram of a request and response between a client and a server.]

The replication approach

[Lamport, 1990; Schneider, 1990]

- Replicate the service.
- Correct clients treat the distributed service as one correct server:
  - Requests are totally ordered, respecting the precedence relation (safety).
  - Every request issued by a correct client is served (liveness).
- Byzantine fault-tolerance (BFT) [Castro and Liskov, 1999]

![Diagram of three servers and a client interacting through requests and responses.]
BFT: costs and optimistic assumptions

- A request (a batch of requests) involves a three-phase agreement protocol to be executed
- A large fraction (more than 2/3) of the service replicas (servers) must be correct
  - Ok if faults are independent (hardware failures)
  - Questionable for software bugs or security attacks
  - An obstacle for scalability (unlikely to hold for large number of replica groups)

Why 2/3?

- Safety: every two requests should involve at least one common correct server
Why 2/3?

n – number of servers
x – quorum size (number of servers involved in processing a request)
f – upper bound on the number of faulty servers

\[2x-n \geq f+1 \quad \text{or} \quad x \geq \frac{n+f+1}{2} \quad \text{(safety)}\]

\[n - f \geq x \quad \text{(liveness)}\]

\[\Rightarrow \quad n \geq 3f+1\]

Trading off liveness for safety

- Every request involves at least \(\frac{n+f+1}{2}\) servers
  \(\Rightarrow\) safety is ensured as long as \(f\) or less servers fail
- Liveness will be provided if not more than \(n-(n+f+1)/2=\frac{n-f-1}{2}\) servers fail

\- n=10, f=7: liveness tolerates at most one failure
Trading off liveness for safety

- $f < n/3$
  - Both safety and liveness are ensured with quorums of size $2/3n+1$
- $f = n-1$
  - Safety: $n-1$ or less faulty servers
  - Liveness: no fault-tolerance at all

Unexpected benefits!

- Large quorums may make things faster!

- Very fast in the good case
- Very slow (unavailable) in the (rare) intermediate case
- But always correct

- Holds only for the special case $f=n-1$?
Using the trade-off

- A “bimodal” failure model?
  - Few failures is the common case
  - Many failures is a possible (but rare) case ($f >> n/3$)
    - Software bugs and security attacks?

- Modified BFT looks like a perfect fit!

Challenge: scalable BFT

- Farsite, Rosebud, OceanStore,…
  - All of them use multiple BFT groups
  - A group is responsible for a part of the system state (an object)
  - Each group is supposed to be safe and live (the 2/3 assumption is not violated)

- The more groups we have - the more likely one of them fails: the system safety is in danger

- Going beyond 2/3 per group?
Using the trade-off: scalable BFT

- The (large) bound on the number of faulty servers per group is never exceeded
- Each group runs the modified BFT: can be seen as a crash-fault processor

Addressing liveness

- Primary-backup: from $p$ to $p^2$
  - Every object is associated with a pair of groups
- Speculative executions [Nightingale et al., 2005]
  - Primary group produces tentative results
  - Backup group assist in committing them
Normal case

Client
- Run operations on the primary group tentatively
- Check whether the tentative results turned into definitive (the state was successfully transferred to the backup group)

Backup-primary
- Periodically transfer the system state from primary to backup

Liveness checks and recovery

Takeover protocol: when the primary fails the backup takes over the speculative execution
- Primary fails: backup takes over in speculative executions
- Backup fails: select a new backup
- Configuration changes: elect new primary and backup (at least one of the old ones must remain live until the state is transferred)
Properties

- Safety: always
- Liveness: as long as at least one group is available

Related work

- BFT, Castro and Liskov, 1999
- “Scalable” BFT: OceanStore, 2000; Farsite, 2002; Rosebud, 2003,…
- Safety-liveness trade-offs, Lamport, 2003
- Fork consistency, Li and Mazieres, 2007
- Singh et al., 2007
- Speculative executions, Nightingale et al., 2005
- Fault isolation, Douceur et al., 2007
Conclusions and Future

- Safety at the expense of liveness [HotDep07]
  - Security and tolerance to software errors
  - Scalability

- Safety + conditional liveness
  - Crash fault computing: safe algorithms + failure detectors
  - Software transactional memory: optimistic STMs + contention managers

- Does this stuff work?
  - Fault model analysis
  - Multiple backups: from $p^2$ to $p^k$
  - Paxos?
  - Implementation