The Next 700 BFT protocols

EuroSys 2010

Rachid Guerraoui, Nikola Knežević
EPFL, Switzerland

Vivien Quéma
CNRS, France

Marko Vukolić
IBM Research - Zurich
Outline

• Context
  – What is BFT?

• Problem
  – BFT protocols are difficult and complex. Why?

• Contribution A:
  – Abstraction for simpler design & implementation of BFT protocols

• Contribution B:
  – BFT protocols developed using the abstraction

• Conclusion
What is BFT?

- BFT: acronym for Byzantine Fault Tolerant protocols

- The term Byzantine dates back to the seminal paper Lamport, Shostak, Pease: The Byzantine Generals Problem, ACM TPLS, 1982.

- Byzantine failure = arbitrary failure
(Distributed) Systems community:

BFT ~ BFT State Machine Replication
State machine replication
System model

- Message-passing with unreliable communication links

- Byzantine faults
  - Any number of clients
  - Less than 1/3 of replicas (3f+1 servers to tolerate f faults, optimal)

- Cryptographic techniques cannot be violated

- Eventual synchrony
BFT evolution

- Lamport, Shostak, Pease: The Byzantine generals problem, 1982
- Castro, Liskov: Practical BFT, 1999

- BFT in 2010 (a decade+ later)
  - BFT research is flourishing (SOSP, OSDI, NSDI, TOCS, EuroSys, ...)
  - But... it is not yet instantiated in practice

- Problem: BFT protocols are notoriously difficult to
  - Design
  - Implement
  - Prove correct
  - Test
BFT evolution

- Step 1: PBFT (Castro, Liskov OSDI99)
BFT evolution

• Step 1: PBFT (Castro, Liskov OSDI99)

Invoke(req)
BFT evolution

- Step 1: PBFT (Castro, Liskov OSDI99)
  - 20000+ lines of complex C++ code (research prototype)
  - 35 pages I/O automata proof
BFT evolution

- **Step 1:** PBFT (Castro, Liskov OSDI99)
  - 20000+ lines of complex C++ code
  - 35 pages I/O automata proof

- **Step 2:** Zyzzyva (Kotla et al. SOSP07)
  - Modify PBFT common-case path by introducing speculation

---

**Zyzzyva common case**

**Zyzzyva faulty case**
### BFT evolution

- **Step 1:** PBFT (Castro, Liskov OSDI99)
  - 20000+ lines of complex C++ code
  - 35 pages I/O automata proof

- **Step 2:** Zyzzyva (Kotla et al. SOSP07)
  - Modify PBFT best-case path by introducing speculation
  - “Speculative execution has profound effects on Zyzzyva’s view change sub-protocol”
  - Another 20000+ lines of code to be written and proved correct

- **Step 3:** Zeno (Singh et al. NSDI09)
  - Adapt Zyzzyva to relax consistency guarantees to improve availability
  - “Adaptation was challenging for several reasons, such as…adapting the view change protocols…”
  - Had to be proved from scratch

Too monolithic → difficult to design, implement, prove, test
A “view change” perspective

- Observation: A typical BFT protocol proceeds in a sequence of *views*
  - Interfaced with “view-changes” (when “something goes wrong”)
  - The **very same** subprotocol is repeated over and over again
    (only the leader is changed)

  **Why?**

- Can we modularly combine Zyzzyva speculation with PBFT?
  - Existing protocol combinations non modular (e.g., Cowling et al. [OSDI06])

- Can we change **entire** sub-protocols with a “view-change”
  - No one-size-fits-all protocol exists (e.g., Singh et al. [NSDI08])
Contribution A: ABSTRACT

• ABSTRACT
  – ABortable STate mACHine replicaTion
  – Simplifies design/implementation/proofs/testing of BFT protocols

• ABSTRACT looks like SMR
  – Except that it may sometimes *abort* a client’s request
  – Returns *abort history*: information on committed requests used to bootstrap the next ABSTRACT sub-protocol

• When is *abort* allowed?
  – *Generic parameter*
  – Models progress semantics
  – Abort when “something goes wrong”
ABSTRACT

client + request 3 + request 4

ticket for A2

+ ticket for A3

ticket for A3

client
Building BFT using ABSTRACT

• BFT is a special ABSTRACT
  – One that never aborts

• BFT safety (total order among committed reqs)
  – Preserved by any ABSTRACT

• ABSTRACT + ABSTRACT = ABSTRACT
  – Safety extends to composition of any number of instances!

• BFT designers should ensure liveness of the composition
  – NB: reuse of any existing BFT yields a powerful ABSTRACT
Summary: ABSTRACT benefits

• Simplifies BFT design/proof/implementation
  – BFT = modular combination of many ABSTRACTs

• Each ABSTRACT instance
  – Developed, proved and tested independently
  – Focus on specific progress goals
  – Abort when something goes wrong

• Switch among instances using common interface
  – ABSTRACT API “standardizes” view-changes
Contribution B: BFT protocols using ABSTRACT

- **AZyzzyva**: building a Zyzzyva-like protocol using ABSTRACT
  - Leveraging any existing BFT protocol
  - With a fraction of additional code

- **Aliph**: a BFT protocol improving State-of-the-Art:
  - By up to 360% in throughput
  - By up to 30% in latency
AZyzzyva

- Uses 2 ABSTRACT instances
  - Zlight
    - **Progress:** no server failures, no malicious clients, synchrony
  - Backup
    - **Progress:** will commit exactly $k$ requests (where $k$ can be tuned)

- Alternates between ZLight and Backup instances
AZyzzyva: ZLight

Invoke(req)  commit(req,reply)  abort (history)

wait for 2f+1 ABORT messages

replicas send ABORT messages with signed local histories
AZyzzyva: plugging-in Backup

global BFT commit history: r1 r3 r2 r4

client c1

Invoke (r2)

AH2=⟨r1,r2⟩

AH4=⟨r1,r4, AH2=⟨r1,r3, r2, r4⟩

client c2

AH3=⟨r1,r3⟩

Invoke (r3)

ZLight 1st instance

r1

Backup 1st inst. (k=2)

rAH3 r2

ABORT (r3,AH3)

COMMIT (r2,AH2)

Invoke (r2, AH2)

Invoke (r3, AH3)

COMMIT

ABORT (r4, AH4)

COMMIT

Invoking client c1:

Invoke (r1)

ABORT (r1)

COMMIT

Invoking client c2:

Invoke (r3)

ABORT (r3)

COMMIT

invocation: (r1,AH1)
AZyzyva: assessment

- Qualitative assessment

<table>
<thead>
<tr>
<th></th>
<th>Zyzzyva</th>
<th>ZLight</th>
<th>Backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages of pseudo-code</td>
<td>4,5</td>
<td>0,5</td>
<td>0,5</td>
</tr>
<tr>
<td>Pages of proofs</td>
<td>&gt; 4</td>
<td>1</td>
<td>0,5</td>
</tr>
<tr>
<td>Lines of code</td>
<td>14339</td>
<td>3358</td>
<td>600</td>
</tr>
</tbody>
</table>

- Performance
  - Common case: exact same performance
  - Faulty case: switching time ~ 20ms
Contribution B: BFT protocols using ABSTRACT

- **AZyzzyva**: building a Zyzzyva-like protocol using ABSTRACT
- Leveraging any existing BFT protocol
- With a fraction of additional code

- **Aliph**: a BFT protocol improving State of the Art:
  - By up to 360% in throughput
  - By up to 30% in latency
Consists of 3 ABSTRACT instances

1) Quorum
   Progress: no server failures, no malicious clients, synchrony, \textit{no contention}
2) Chain
   Progress: no server failures, no malicious clients, synchrony
3) Backup
   Progress: commit exactly $k$ requests

Alternates between Quorum, Chain and Backup instances
Aliph: design intuition

- Low load (no contention): use Quorum
  - Very low latency (-30%)

- High load: use Chain
  - Very high throughput (+360%)

- Failures, asynchrony: use Backup

<table>
<thead>
<tr>
<th></th>
<th>PBFT</th>
<th>Q/U</th>
<th>HQ</th>
<th>Zyzzyva</th>
<th>Aliph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of replicas</td>
<td>3f+1</td>
<td>5f+1</td>
<td>3f+1</td>
<td>3f+1</td>
<td>3f+1</td>
</tr>
<tr>
<td>Throughput (MAC ops at bottleneck server)</td>
<td>$2 + \frac{8f}{b}$</td>
<td>$2 + 4f$</td>
<td>$2 + 4f$</td>
<td>$2 + \frac{3f}{b}$</td>
<td>$1 + \frac{f+1}{b}$</td>
</tr>
<tr>
<td>Latency (1-way messages in the critical path)</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Aliph: Quorum

Invoke(req)  commit(req, reply)  abort (history)

wait for 2f+1 ABORT messages

replicas send ABORT messages with signed local histories
**Chain challenges**: make sure that

- The content of messages is not modified by a malicious replica
- No replica in the chain is bypassed
- Replies sent by the tail are correct
Aliph: evaluation

- x/y Microbenchmarks [Castro, Liskov99]
  - x kB request size
  - y kB response size

- Clients invoke requests in a closed loop
  - A client can only have 1 pending request

- Cluster: 17 identical machines, 1.66GHz dual-core, 2GB RAM, Linux 2.6.18 kernel, Gigabit ethernet switch

- Note: Backup uses PBFT
Latency (0/0 benchmark)

% of overhead wrt. Aliph (Quorum)

Q/U (5f+1)  Zyzzyva  PBFT

f=1  f=2  f=3
Throughput (0/0 benchmark)
Throughput (4/0 benchmark)
Dynamic workload (fluctuating contention)

- 30 clients
- Requests of different sizes: 0k, 0.5k, 1k, 2k, and 4k

Experiment:
- No failures
- Start with a single client issuing requests
- Progressively increase the number of clients until it reaches 10
- Simulate a load spike with 30 clients simultaneously sending requests
- Progressively decrease the number of clients, until there is only 1 client

- Low load detection to abort from Chain
Dynamic workload
Behavior under faults (Backup with k=1)
Behavior under faults (Backup with $k=2^i$)
Conclusion

• ABSTRACT
  – Abstraction for simpler BFT protocol design/proof/implementaiton

• New and efficient protocols
  – Aliph (1st of 700)
    • 30% lower latency
    • 360% higher throughput
    • With a fraction of State-of-the-Art protocol code size
Future work

699 protocols remain to be designed!

- Dynamic switching
- Build *robust* BFT protocols using Abstract
- Leverage multicore architectures (i.e. multi-threaded replication)
- …
Thank you!

Questions?
Peak throughput

![Graph showing throughput vs request size for different protocols]

- PBFT
- Zyzzyva
- Aliph

Throughput (Kops/sec) vs Request size (B)