The Fail-Heterogeneous Architectural Model

Marco Serafini & Neeraj Suri

TU Darmstadt
Fault Models

- Distributed system
  - Node failures
- Impact on:
  - Fault tolerance
  - Performance
- Classic models
  - Fail-Crash
  - Fail-Byzantine
Homogeneous Fault Models

- Fail-Crash
  - Best-case
  - Processes simply stop operating
    + Performance
    - Fault tolerance

© Marco Serafini
Homogeneous Fault Models

- **Fail-Crash**
  - Best-case
  + Performance
  - Fault tolerance

- **Fail-Byzantine**
  - Worst-case
  - Processes will do their best to damage the system
  + Fault tolerance
  - Performance
Homogeneous Fault Models

- **Fail-Crash**
  - Best-case
    + *Performance*
    - *Fault tolerance*

- **Fail-Byzantine**
  - Worst-case
    + *Fault tolerance*
    - *Performance*

- **Better tradeoff?**
Outline

- Limitations of homogeneous models
- The Fail-Heterogeneous Architectural Model
- Case study: HeterTrust
  - Practical state machine replication architecture
  - Provides confidentiality
  - More efficient than Fail-Byzantine protocols
- Other applications
  - DoS defense
  - Group membership
Limitations of Homogeneous Models

- **Fail-Crash allows better performance**
  - Should be used when possible

- **In some cases can be inadequate**
  1. Severe accidental faults
  2. Security-related faults (e.g. intrusions)
Limitations of Homogeneous Models

- Fail-Crash allows better performance
  - Should be used when possible
- In some cases can be inadequate
  1. Severe accidental faults
  2. Security-related faults (e.g. intrusions)
- Homogeneous model
  - One node Fail-Byzantine → All nodes Fail-Byzantine
Limitations of Homogeneous Models

- **Fail-Crash allows better performance**
  - Should be used when possible
- **In some cases can be inadequate**
  1. Severe accidental faults
  2. Security-related faults (e.g. intrusions)
- **Homogeneous model**
  - One node Fail-Byzantine → All nodes Fail-Byzantine
- **What if only SOME nodes Fail-Byzantine?**
  1. In practice, most Byzantine problems are security-related
  2. In practical security, trusted design is common
Accidental Faults

- **Theory:** Severe accidental faults → Fail-Byzantine
- **Practice:** Good error detection ~ Fail-Crash
- **Example:** Chubby
  - Paxos-based replicated naming service @ Google
  - Secure environment
  - Fail-Crash tolerant
  - 100 years/machine
  - 1 reported failure due to accidental faults
  - Reliability ~ 0.999999 → 6 nines!!

*T. Chandra et al., “Paxos Made Live – An Engineering Perspective.” PODC’07*
“Security [in an OS] is easy to provide. The only problem about security is that all the code implementing it must be bug-free”

*Andy Goldstein, HP, co-designer of VMS*

- **Practical secure systems design**
  - Correct trusted subsystem
  - Restrict interface
  - Monitor interactions

- **Security by trusted design**

Example: OS

![Diagram](image-url)
Trusted design

- **Trusted distributed subsystems = ... ?**
  - Trusted nodes!
  - Strong fault isolation, generic component...

- **Impractical trusted design**
  - Complex
  - Evolvable
  - No trusted and/or multiple admins

- **Practical trusted design**
  - Simple, dedicated nodes
  - Generic functionalities

Examples:
- Univ. File-Server
- Firewall
Fail-Heterogeneous Arch. Model

- **Dedicated Coordinators**
  - Simple, static devices
  - Generic coordination operations
  - Trust $\rightarrow$ Fail-Crash

- **Execution nodes**
  - Unconstrained, unreliable
  - Provide the services of interest
  - No trust $\rightarrow$ Fail-Byzantine
Comparison with Other Models (1)

- **Hybrid fault models**
  - A Byzantine tolerant system also tolerates crashes
  - Homogeneous: each node can still fail-Byzantine

- **Fail-Heterogeneous Arch. Model**
  - Heterogeneous: some *specific* nodes are fail-crash
  - Others are fail-Byzantine
Comparison with Other Models (2)

- **Hybrid Architectures**
  - Delta 4, Wormhole (TTCB)
  - Multiple subsystems within a node
  - Heterogeneous fault & *synchrony* models

- **Fail-Heterogeneous Arch. Model**
  - No additional assumptions on synchrony
  - Trusted design and implementation
  - No deployment aspects
Outline

- Limitations of homogeneous models
- The Fail-Heterogeneous Architectural Model
  - Case study: HeterTrust
    - Practical state machine replication architecture
    - Provides confidentiality
    - More efficient than Fail-Byzantine protocols
- Other applications
  - DoS defense
  - Group membership
State Machine Replication (SMR)

- Clients ↔ Replicated server
- **SMR** = Keep replicas *consistent*
  1. *Agreement* on the next step
  2. *Execution* of the request and reply

- Asynchronous systems
  - Eventual synchrony for liveness
Properties:

1. Termination
   - All client requests are eventually executed

2. Consistency
   - Replicas have a consistent state

3. Confidentiality
   - Replica-internal data cannot be sent to clients by faulty servers
Confidentiality

- Generally overlooked
  - Hard to guarantee under the fail-Byzantine model
- Replication is detrimental
  - Higher likelihood of intrusion
- Often more important than consistency!!
Architecture for confidentiality
- $2g+1$ dedicated coordinators
- $2f+1$ execution servers
Advantages of the Model

Comparison of SMR protocols *with confidentiality*

<table>
<thead>
<tr>
<th></th>
<th>Fail-Byzantine</th>
<th>HeterTrust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement and filter nodes</td>
<td>$f^2+4f+1 \ (*)$</td>
<td>$2g+1$</td>
</tr>
<tr>
<td>Execution servers</td>
<td>$2f+1 \ (*)$</td>
<td>$2f+1 \ (*)$</td>
</tr>
<tr>
<td>Communication steps</td>
<td>$2f+7$</td>
<td>$4$</td>
</tr>
<tr>
<td>Cryptography</td>
<td>Threshold signatures</td>
<td>Symmetric crypto</td>
</tr>
</tbody>
</table>

*J. Yin *et al.*, “Separating Agreement From Execution for Byzantine Fault-Tolerant Services.” *SOSP*’03

\(*)\) Nodes with diversified design
Classic Solution

1) Agreement (Paxos)

2) Execution

3) Filtering (confidentiality)
Classic Solution

6 communication steps in the critical path...

1) Agreement (Paxos)
2) Execution
3) Filtering (confidentiality)
Classic Solution

IDEA: Combine the 3 phases
Reduce latency
HeterTrust Protocol

ORDER requests and send them to servers

Phase 1

TENTATIVE EXECUTION based on leader proposal
HeterTrust Protocol

Phase 1

client

coordinators

servers

ACCEPT

a reply only if receive f+1 equal msgs from untrusted servers, else filter it out
HeterTrust Protocol

client

coordinators

servers

Phase 1

Phase 2

DELIVER
if receive equal msgs from
majority of coordinators

deliver

DELIVER
if receive equal msgs from
majority of coordinators

commit

commit

commit

commit

COMMIT
if receive equal msgs from
majority of coordinators

© Marco Serafini
HeterTrust Protocol

REQUEST

PROPOSE

EXECUTED

ACCEPTED

deliver

client

coordinators

servers

Phase 1

Phase 2

1

2

3

4

ONLY 4 COMMUNICATION STEPS IN THE CRITICAL PATH
HeterTrust Protocol

**Phase 1**
- Request
- Propose
- Executed

**Phase 2**
- Accepted
- Learnt
- Commit

**Phase 3**
- Commit

**Terminate**
If g+1 coordinators have learnt a request
Recovery

- Similar to Paxos
  - Executed among Fail-Crash coordinators

- Works well with
  - Byzantine servers
  - Tentative executions
Advantages of Trusted Coordinators

- **Latency**
  - Trusted leader → Consistent broadcast
  - Trusted acceptors → Smaller quorums
  - Less communication steps

- **Confidentiality**
  - No threshold signatures → Computational overhead
  - No multiple tiers of filters → Communication overhead
Outline

- Limitations of homogeneous models
- The Fail-Heterogeneous Architectural Model
- Case study: HeterTrust
  - Practical state machine replication architecture
  - Provides confidentiality
  - More efficient than Fail-Byzantine protocols

- Other applications
  - DoS defense
  - Group membership
DoS Attacks

- Network-level defenses
  - How to discriminate correct/incorrect traffic?
  - Can benefit of protocol-level information

- Fail-Heterogeneous Arch. Model
  - Dedicated coordinators → SMR-level filtering
Group Membership

- **Fail-Byzantine** → Fault location is difficult
- **Fail-Heterogeneous Arch. Model**
  - Coordinators keep membership views
  - Trusted determination of faults

![Diagram of Coordinators, Partition, Relay messages, and Agree on new view]
Conclusions

- Fail-Heterogeneous Architectural Model
  - More flexible than homogeneous models
  - Allows better tradeoffs
  - Is practical
Conclusions

HeterTrust: Efficient trustworthy SMR
- Provides confidentiality

Perspectives
- This paper: Trusted design
- Future work: Dynamic trust, adjusted at runtime
Thank you for your attention!