Intrusion-Tolerant Parsimonious State Machine Replication

HariGovind V. Ramasamy
Adnan Agbaria
William H. Sanders

Workshop on Secure Multiparty Protocols (SMP 2004), Amsterdam, Oct 7-8, 2004

PERFORM Research Group
http://www.perform.csl.uiuc.edu
University of Illinois at Urbana-Champaign, USA

© 2004, HariGovind V. Ramasamy
Intrusion Tolerance (IT)

Reality
- Systems will have vulnerabilities
- Subset of vulnerabilities will be exploited eventually

Goals
- Provide acceptable service despite intrusions
- Survivability in the face of attacks

Defense-in-depth
- Intrusion prevention
- Intrusion tolerance
- Intrusion detection
- Isolation & rejuvenation
Byzantine Fault Tolerant (BFT) Replication Protocols

- One of the potential building blocks for IT
- Caveat
  - Replicas shouldn’t fall prey to same attacks
    - Diversity (N-version programming, OS heterogeneity..)
- Two categories
  - Quorum Replication (Malkhi, Reiter)
  - State-machine replication (our focus)
Byzantine Fault Tolerant (BFT) State Machine Replication (SMR)

Yin et. al [SOSP’03]
Related Work

- Long line of work on BFT
- Focus of recent work
  - Making BFT practical and relevant to *real* services
  - e.g., Rampart, Fleet, SecureRing, PBFT, COCA, SINTRA, Yin et al., etc.
- Our work: similar focus, but different means
  - Focus: BFT SMR efficient, relevant to *real* services
  - Means: Efficient BFT SMR execution phase
Parsimonious Execution

- Execution phase
  - Likely to be expensive, perhaps more than agreement
  - Often overlooked: agreement is the *harder* problem
  - Quite application dependent

- Goal
  - Improve efficiency in execution phase
  - Mechanism used must be applicable for many applications

- Key Idea – Parsimony
  - Use only a primary committee to actually execute requests
  - Reconfiguration, if committee seems to be not doing job properly
    - All replicas become temporarily active
    - Reselect primary committee
Parsimonious Execution

- Premise: Faults are exception rather than norm
- Fault-free-runs overhead
  - Minimize redundant processing & computation
- Faulty runs overhead
  - May be temporarily higher, but not unacceptable
- Uses timing assumptions not for safety/liveness, but for efficiency
Outline

- System Model
- Base Protocol
  - Fault-free operation
  - Reconfiguration
  - Analysis
  - Properties
- Lazy Updates
- MAC-Extended Protocol
- Ongoing work, Conclusion
Gateway

- Logical substitute for agreement phase
- Serializing mechanism
  - Orders client requests
  - Sends requests in same order to all execution replicas
- Trusted
  - Made trustworthy by distributed implementation (agreement replicas) using atomic broadcast protocol (e.g., PBFT)
- Not aware and doesn’t care about the primary committee

© 2004, HariGovind V. Ramasamy
System Model

- Asynchronous processing & communication
- Gateway and a set $S$ of $n$ execution replicas
  - $n \geq 2t+1$ :– total # execution replicas
  - $t$ :– max # replicas simultaneously faulty
  - $t+1$ :- size of primary committee
- FIFO, quasi-reliable channels
- Public-key cryptography + MACs
- Computationally bounded adversary
Notations

- \((t+1)\)-subset :- subset of \(S\); exactly \(t+1\) replicas
- \(PC\) :- Ordered list, contains all \((t+1)\)-subsets that could be primary committees (identical and constant across all replicas)
  - e.g., Set of all possible \((t+1)\)-subsets of \(S\)
- \(c_i\) :- committee number at replica \(i\)
  - Initialized to 1, at all replicas
- \(P_i\) :- primary committee at replica \(i\)
  - \(P_i\) is the \(c_i^{th}\) element of \(PC\)
- \(OFL_i\) :- omission fault list at replica \(i\)
- \(CFL_i\) :- commission fault list at replica \(i\)
Base Protocol

- Signed msg exchanges among replicas
- Gateway sends request (say seq# x) to all replicas
- Each committee member executes request; determines result $r$ & update $u$
  - Sends reply msg (with $r$) to gateway
  - Sends reply-update msg (with $r$, $u$) to all replicas
  - Starts timer, expects other committee members to send reply-update message before timeout
- Each Backup
  - Starts timer, expects committee members to send reply-update message before timeout
Base Protocol: Fault-free operation

- **Gateway**
  - receives $t+1$ replies with identical $r$ values
  - accepts $r$ as result for request $x$

- **Replicas**
  - receive $t+1$ reply-updates with identical $r,u$ values
  - if backup, apply state update $u$
  - move request $x$ from *input-queue* to *handled-queue*
Base Protocol: Reconfiguration

- Committee members could be corrupted/slow
- Reconfiguration
  - Needed to ensure progress
  - Must not be possible for faulty backups to initiate, when committee indeed is performing correctly
  - Initiated only with sufficient proof
    - Replica $i$ is added to fault-list of a correct replica $\rightarrow$ all correct replicas will eventually add $i$ to their fault-lists
- Progress guarantee
  - Gateway WILL accept response for request($x$) at end of reconfiguration
Base Protocol: Reconfiguration Triggers (1/2)

if reply-update from committee member \(i\) not received in time
send suspect-omission(\(i\))

if badly formatted but correctly signed msg \(m\) received from \(i\)
send suspect-commission(type-1, \(i, m\))

if reply-updates with differing \(r, u\) values received from \(\geq 2\) committee members
send suspect-commission(type-2, \(I, \Psi\))

\[I = \text{set of those replicas}\]
\[\Psi = \text{set of their different reply-update messages}\]

if any replica \(i\) had sent reply-update with wrong \(r\) or \(u\) value
send suspect-commission(type-3, \(i, \text{proof}\))

\[\text{Proof} = \text{\(i\)'s reply-update msg} \& \text{\(t+1\) correct reply-update msgs}\]
Base Protocol: 
Reconfiguration Triggers (2/2)

received suspect-omission (i) from k for request x
  if reply-update (i, x) received
    forward reply-update (i, x) to k
    if suspect-omission (i) received from n-t replicas
      add i to OFL
      if (i is committee member)
        reconfiguration()

received suspect-commission(type-1, i, m)
  add i to CFL
  if (i is committee member)
    reconfiguration()

received suspect-commission (type-3, i, proof)
  add i to CFL
  if (i is committee member)
    reconfiguration()

received suspect-commission (type-2, I, proof)
  if (all elements in set I are committee members)
    reconfiguration ()
Base Protocol: Reconfiguration Protocol

- Step 1:
  - All replicas temporarily become active
  - New commission-faulty replicas may be identified

- Step 2:
  - Reselect new primary committee skipping over elements of $PC$ containing replicas in fault-lists
Base Protocol:
Reconfiguration Protocol (Details)

reconfiguration()
send reselect (j, c, Γ)
if !(executed request x)
  execute request x
  send reply-update(j, x) to replicas, reply(j, x) to gateway
if (request(x) ∉ handled-set)
  wait for reply-update msgs with identical r,u values from t+1 replicas
  move request(x) to handled-set
if any replica i had sent reply-update with wrong r or u value
  send suspect-commission(type-3, j, i, proof)
  add i to CFL
set c to the smallest integer s.t. c\(^{th}\) element of PC doesn’t contain any element in OFL or CFL
set P to the c\(^{th}\) element of PC
Reconfiguration Protocol: Analysis

- Committees at correct replicas may temporarily differ, but will eventually concur.
  - Replica $i$ is added to fault-list of a correct replica → all correct replicas will eventually add $i$ to their fault-lists.

- Eventual goal of reconfiguration
  - “Settle” on an all-correct $(t+1)$-subset as committee
  - $O(f)$, where $f \leq t$ is #replicas in combined fault-lists of all correct replicas
  - Note: gateway doesn’t have to wait till settling to accept response
Reconfiguration Protocol: Denial-of-service attacks

- Uncorrupted replicas may be added to OFL
- \( \text{Maybe } f > t \)
- If \( |\text{OFL} \cup \text{CFL}| > t \)
  - Refresh OFL entries (CFL entries – permanent)
  - Increment \textit{reconfiguration cycle}
  - Committee reselection done using same rule as before
  - Suspect-omission msgs carry reconfiguration cycle
    - Valid only for that reconfiguration cycle
  - No longer \( O(f) \), but progress still guaranteed
    - Subject to quasi-reliable channel assumption
    - Subject to no more than \( t \) replicas actually \textit{corrupted}
Base Protocol: Properties

Termination Gateway sends request → it eventually accepts response

Total Order Updates for $x^{th}$ gateway request same at all correct replicas

Update Integrity Updates for $x^{th}$ gateway request happens exactly once, and only if gateway actually sent that request.

Response Integrity Response accepted by gateway → at least one correct replica sent that response

Parsimony Unless faulty behavior of some committee member has been observed by some replica in a manner that is provable to any other correct replica, gateway requests will be executed only by $t+1$ replicas constituting primary committee
Base Protocol: Drawbacks and Extensions

- **Drawbacks**
  - Backups receive reply-update msgs
  - Backups apply state updates after every request
  - Uses public-key signatures for all inter-replica messages

- **Extensions**
  - Lazy Updates at backups using checkpointing
  - MAC-Extended Protocol
Checkpointing for Lazy Updates at Backups

- Update backup states only when needed (reconfiguration)
- Periodic checkpoint requests
  - Processed just like a regular gateway request
  - $t+1$ identical replies received $\rightarrow$ stable checkpoint
- Stable checkpoint $\rightarrow$ state update, if requested, will be eventually received
- Trade-off: Higher reconfiguration latency vs. reduced processing overhead
MAC-Extended Protocol

- MAC-mode
  - Replicas don’t receive reply-updates from others
  - Lazy backup updates
  - Only gateway-to-replica, replica-to-gateway msgs (except for infrequent checkpoint requests)
  - As long as gateway accepts response in time

- Base-mode
  - Caused by gateway’s req-retransmission msg
Conclusion

- Parsimony applied to Byzantine fault model
- Reduced overhead in fault-free case
- Additional overhead if faults in committee
  - Detection + reconfiguration latency
- Pronounced benefits expected for
  - larger group sizes
  - Computation/communication intensive apps
  - Infrequent/inexpensive/localized state updates
- Ongoing work – experimental evaluation