Round model for dist. algo.
From verification to implementation

FRIDA, 24.07.2014, Vienna

Cezara Drăgoi
Thomas A. Henzinger
Josef Widder
Damien Zufferey
Our journey starts on the island of Paxos …

… where archeologists made an interesting discovery about a parliament system …
The Paxos Algorithm [Lamport 98]

Proposer

Accepter

Accepter

Prepare Promise Accept Accepted

Used at Google (Chubby), Yahoo/Apache (Zookeeper), Microsoft (Autopilot)
What verification can do?

• Hard to implement and get right
  • “The fault-tolerance computing community has not developed the tools to make it easy to implement their algorithms.” [Chandra et al. 07]
  • “The fault-tolerance computing community has not paid enough attention to testing, a key ingredient for building fault-tolerant systems.” [Chandra et al. 07]
  • ⇒ use for formal verification

• Gap between the theory community and the system community
  • “In order to build a real-world system, an expert needs to use numerous ideas scattered in the literature and make several relatively small protocol extensions. The cumulative effort will be substantial and the final system will be based on an unproven protocol.” [Chandra et al. 07]
  • ⇒ use for automated verification
Our goals

source code + specifications

verifier

proof or counterexample

runtime

executable
Our goals

• Language for verified fault tolerant distributed algorithms implementation
  • Algorithms in isolation (as published)
  • Algorithms as part of a bigger system (modified to fit purpose)

• Use the HO model to simplify the reasoning
  • User provides an inductive invariants, then push button
  • Need a very expressive logic for automation (Cezara’s talk)

• Provide an implementation that performs well enough
  • Show that the overhead of rounds is acceptable
Verification Challenges

- Parametric systems
- Asynchrony (Interleaving, delays)
- Channels
- Faults
Communication-closed Rounds

[Elrad & Francez 82]: decomposition of algorithm in communication-closed rounds.

[Dwork & Lynch & Stockmeyer, 88] defines round model for non-synchronous models: partial synchrony
Mapping asynchrony to faults

Reasoning in round-based model (partial synchrony)

Implementation in an asynchronous world
The Heard-Of model [Charron-Bost & Schiper 09]

- \( p \in HO(q, r) \): message send by \( p \) to \( q \) at round \( r \) is delivered

- Maps every faults to message faults
  - A crashed process is the same as a process whose messages are dropped.
  - Byzantine faults can be simulated altering messages
  - Simplify the proofs: does not need to case split on (in)correct processes
  - Handling transient/permanent faults is transparent at the algorithm level

- However, in practice, ...
Last Voting Algorithm

7: Round $r = 4\phi - 3$
8: $S_p^r$
9: send $(x_p, ts_p)$ to Coord($p, \phi$)
10: $T_p^r$
11: if $p = $ Coord($p, \phi$) and number of $(v, \theta)$ received $> n/2$ then
12: let $\theta$ be the largest $\theta$ from $(v, \theta)$ received
13: $vote_p :=$ one $v$ such that $(v, \theta)$ is received
14: $commit_p :=$ true

22: Round $r = 4\phi - 1$
23: $S_p^r$
24: if $ts_p = \phi$ then
25: send $(ack)$ to Coord($p, \phi$)
26: $T_p^r$
27: if $p = $ Coord($p, \phi$) and number of $(ack)$ received $> n/2$ then
28: $ready_p :=$ true

15: Round $r = 4\phi - 2$
16: $S_p^r$
17: if $p = $ Coord($p, \phi$) and $commit_p$ then
18: send $(vote_p)$ to all processes
19: $T_p^r$
20: if received $(v)$ from Coord($p, \phi$) then
21: $x_p := v; ts_p := \phi$

29: Round $r = 4\phi$
30: $S_p^r$
31: if $p = $ Coord($p, \phi$) and $ready_p$ then
32: send $(vote_p)$ to all processes
33: $T_p^r$
34: if received $(v)$ from Coord($p, \phi$) then
35: $DECIDE(v)$
36: if $p = $ Coord($p, \phi$) then
37: $ready_p :=$ false
38: $commit_p :=$ false
Verification
Goals for the verification

• Safety and liveness properties
  • Agreement, Validity, Irrevocability
  • Termination

• User provided invariants
  • Quite simple
    • No channels
    • All the processes have the same PC

• Flexibility
  • Being able to handle a large class of algorithms
Invariant for the Last Voting example

\[
\forall i. \neg decided(i) \land \neg ready(i)
\]

\[
\lor \exists v, t, A. \ A = \{ i. \ ts(i) > t \} \land |A| > n/2
\]

\[
\land \forall i. i \in A \Rightarrow x(i) = v
\]

\[
\land \forall i. decided(i) \Rightarrow x(i) = v
\]

\[
\land \forall i. commit(i) \lor ready(i) \Rightarrow vote(i) = v
\]

\[
\land \ t \leq \Phi
\]

\[
\land \forall i. ts(i) = \Phi \Rightarrow commit(coord(i)) = v
\]
Implementing a system
Architecture

User application

Algorithm

Round abstraction (Predicate)

Network
Algorithms

• Does not terminate!
  • Related to recovery procedures

• How does it start?
  • Praise the benevolent sysadmin

• Integration in the user program
  • Relating the guarantees of the algorithms to the rest of the system.

7: Round $r = 4\phi - 3$:
8: $S_p$:
9: send $(x_p, ts_p)$ to Coord($p, \phi$)
10: $T_p$:
11: if $p = Coord(p, \phi)$ and
12: number of $(v, \theta)$ received $> n/2$ then
13: let $\theta$ be the largest $\theta$ from $(v, \theta)$ received
14: vote$_p :=$ one $v$ such that $(v, \theta)$ is received
15: commit$_p :=$ true
16: Round $r = 4\phi - 2$:
17: $S_p$:
18: if $p = Coord(p, \phi)$ and commit$_p$ then
19: send $(vote_p)$ to all processes
20: $T_p$:
21: if received $(v)$ from Coord$(p, \phi)$ then
22: $x_p := v$; $ts_p := \phi$
23: Round $r = 4\phi - 1$:
24: $S_p$:
25: if $ts_p = \phi$ then
26: send $(ack)$ to Coord$(p, \phi)$
27: $T_p$:
28: if $p = Coord(p, \phi)$ and
29: number of $(ack)$ received $> n/2$ then
30: $r$eady$_p :=$ true
31: Round $r = 4\phi$:
32: $S_p$:
33: if $p = Coord(p, \phi)$ and ready$_p$ then
34: send $(vote_p)$ to all processes
35: $T_p$:
36: if received $(v)$ from Coord$(p, \phi)$ then
37: $r$eady$_p :=$ false
38: commit$_p :=$ false
Boundary conditions

• Assumes that every replicas runs forever

• Starting consensus among some replicas is not so different from a consensus problem itself?
  • Difference between not deciding and not knowing we were supposed to decide
  • What are the assumption you need to get to the point where you can run consensus?

• Sometime you don’t even know how many replicas there are.
  • ... coming from industry people who are running data centers ...
Consensus vs Atomic broadcast

• In theory, we can solve atomic broadcast by reduction to consensus.
  • Reverse is also true
• In practice, we need another algorithm.

• Paxos vs Zab
  • Zab [Junqueira et al. 11]: atomic broadcast algorithm used in Zookeeper
  • For performance reason
  • Share many ideas but Zab is designed to enable quick recovery
  • Also customized properties (stronger than atomic broadcast ?)
Same principle, different results?

Paxos

Last Voting
Predicate

- Interface between the “synchronous” rounds and “asynchronous” world.
  - Relative speed of replicas ($\delta$)
  - Network delay ($\phi$)
- example [Hutle & Schiper 07]
Related work

• TLA+ [Lamport 91] that comes with a model checker and proof assistant [Chaudhuri et al. 08]

• Isabelle formalization of algorithms in the HO model [Charron-Bost & Merz 09]

• Mace [Killian et al. 07] a DSL for distributed systems, comes with a model-checker.

• Declarative networking: (logic) programming [Loo et al. 06] and verification [Wang et al. 09]
Summary

Implementing a correct fault-tolerant system is hard:

• We have the tool to help:
  • ATPs, SMT-solvers, model-checkers, ...

• Where to position ourselves?
  • Verification/programming abstraction: more tractable vs closer to real systems
  • Formalization of the boundaries of the systems
    • Start/stop
    • Interactions between main algorithms and recovery procedures
    • Customization of algorithms and properties of the system which uses the algorithm