From an Intermittent Rotating Star to a LEADER

Antonio FERNANDEZ† Michel RAYNAL⋆

† LADyR, Universidad Rey Juan Carlos, Móstoles, Spain

⋆ IRISA, Université de Rennes, France

anto@gsyc.es raynal@irisa.fr

Content

- System model
- The eventual leader oracle Ω
- Assumptions $\mathcal{A}^+$, $\mathcal{A}$, etc.
- Ω algorithms and their properties
- What do we really need?

Part I

BASE ASYNCHRONOUS MODEL

Asynchronous process model

- A set Π of n processes $p_1, \ldots, p_n$
- Timing model: Asynchrony: No upper bound on the time required to execute a computation step
- Failure model: Process crash: a process behaves according to its specification until it possibly crashes, i.e., halts prematurely (after it has crashed a process is definitely stopped)
- At most t processes may crash ($1 \leq t < n$)
  A Correct process is a p_i that never crashes
  A Faulty process is a p_i that crashes

Asynchronous communication model

Reliable message-passing communication system

- Every pair of processes is connected by a link
- Timing model: Asynchrony: No upper bound on transfer delays
- Failure model: Reliable link:
  - ⋆ No creation, no duplication of messages
  - ⋆ Every message sent by p_i to p_j is eventually received by p_j (if p_j is correct)

Asynchronous system model

- $AS_{n,t}[\emptyset]$ computation model: Time-free (asynchronous) model with process crashes and reliable links
- Motivate the time-free distributed system model:
  - Outside the process control domain, most distributed algorithms use time only to detect failures
  - Most general model wrt timing assumptions (hence, it favors modularity, generality, portability, etc. wrt design, implementation and proof)
  - Most constraining model wrt failures (only process crashes)

Part I

EVENTUAL LEADER SERVICE

Eventual leader service: definition

A "black box" (object) that provides the processes with a primitive denoted leader() that satisfies the following properties:

- Validity: leader() returns a process (node) id each time it is called
- Termination: Until a process possibly fails, all its leader() invocations return
- Eventual leadership: There is a time after which, all the leader() invocations return the same id, that is the id of an alive process

Eventual leader service: discussion

- The time from which the leadership property becomes forever satisfied is finite but unknown
- It is possible that, before that time, there is an anarchy period during which there are plenty of leaders (some of them being faulty!)
- The eventual leadership problem is consequently defined by pretty weak properties

The Ω service is a well-defined Object

- Ω is not defined in terms of a particular implementation (involving hardware clocks, network topology, message delays, etc.)
- Ω defined in terms of abstract properties: (abstract data type allowing "plenty" of different implementations)
- This allows a modular decomposition when solving a problem
  - ⋆ First design and prove a protocol based on the abstract properties of Ω
  - ⋆ and then address independently the implementation of Ω
    - ∗ Investigate sufficient assumptions that permit to implement Ω (thereby, getting an "augmented" underlying asynchronous system)
    - ∗ Finally, given a particular "augmented" asynchronous system, implement Ω on top of it

Ω in action (1)

State-machine replication paradigm

- A (deterministic) server is replicated
- Clients issue commands (operations)
- The replicas have to process the commands in the same order
  - ⋆ This problem is a particular instance of the famous consensus problem
  - ⋆ The eventual leader service represents the weakest information on failures needed to solve the consensus problem

Ω in action (2)

A sensor network problem

- Some sensor may fail initially, some may fail later
- They sense the environment and send data to a base station
- To prevent interference and save energy, eventually only one sensor has to send data to the base station
- So, they need to elect one of them that becomes leader until it crashes, and then another leader is elected, etc.

Ω in action (3)

Software transactional memory

- Obstruction-free STM
- Non-blocking STM
- The weakest contention manager for going from Obstruction-free to Non-blocking is ... Ω

Part III

The ASSUMPTIONS $\mathcal{A}^+$ and $\mathcal{A}$

Round-based computation model

- Processes execute asynchronous rounds
- *Regularly* each p_i broadcasts $alive(rn_i)$
- Meaning of *regularly*:
  - ⋆ $send\_time(i, rn)$ = time at which p_i broadcasts alive(rn)
  - ⋆ $\exists\ \beta_i$: $0 < send\_time(i, rn_i + 1) - send\_time(i, rn_i) \leq \beta_i$
    The bound β_i is not necessarily known by p_i
- Only one correct process has to behave synchronously

Messages: definitions

- A message alive(rn) is *δ-timely* if it is received by its destination process at most δ time units after it has been sent
- A message alive(rn) is *winning* if it belongs to the first $(n - t)$ alive(rn) messages received by its destination process

System model $AS_{n,t}[\mathcal{A}^+]$

- There are a correct process p, a bound δ, and a finite round number RN_0, such that for any $rn \geq RN_0$, there is a set of processes $Q(rn)$ satisfying the following properties:
  - ⋆ A1: $p \notin Q(rn)$ and $|Q(rn)| \geq t$
    ($\{p\} \cup Q(rn)$ is a t-star centered at p)
  - ⋆ A2: For any $q \in Q(rn)$ (any point of the star), one of the following properties is satisfied
    - ∗ (1) the process q has crashed,
    - ∗ (2) the message alive(rn) is δ-timely,
    - ∗ (3) the message alive(rn) is winning

Illustration: system

$n = 6 \qquad t = 2 \qquad (n-t)+(t+1) > n$

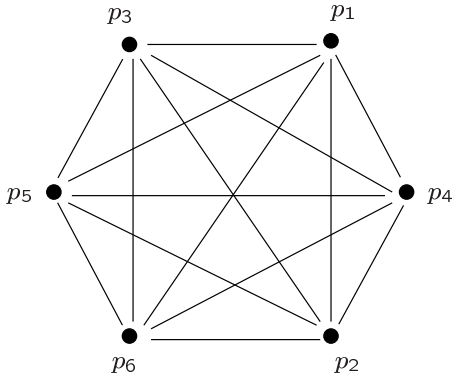
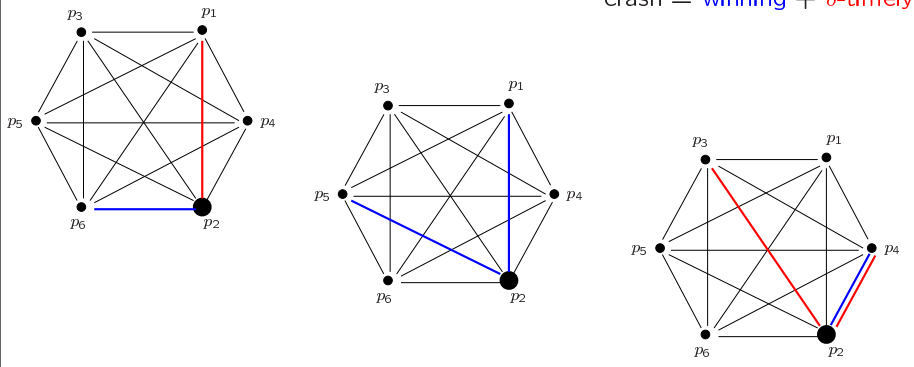


Illustration: a rotating star star

$n = 6 \qquad t = 2$

winning
δ-timely

crash = winning + δ-timely



Dynamicity notions in $AS_{n,t}[\mathcal{A}^+]$

- Wrt time: $Q(rn_1)$ and $Q(rn_2)$ can be different for $rn_1 \neq rn_2$. This is the *rotating* notion introduced in
  Malkhi D., Oprea F. and Zhou L., Ω Meets Paxos: Leader Election and Stability without Eventual Timely Links. *DISC'05*, LNCS #3724, pp. 199-213, 2005
- At any time rn, two points of the star $\{p\} \cup Q(rn)$ ($p \to q1$ and $p \to q2$), are allowed to satisfy different properties, one satisfying the "δ-timely" property, while the other satisfying the "winning" property
- If the point q appears in $Q(rn_1)$ and $Q(rn_2)$ it can satisfy the "δ-timely" property in $Q(rn_1)$ and the "winning" property in $Q(rn_2)$

Particular cases

When $\forall \;\; rn \geq RN_0$, $Q(rn)$ cannot vary

- Only A2.(1) or A2.(2) are satisfied: $\mathcal{A}^+$ boils down to the eventual t-source assumption introduced in
  Aguilera M.K., Delporte-Gallet C., Fauconnier H. and Toueg S., Communication Efficient Leader Election and Consensus with Limited Link Synchrony. *23th ACM PODC*, pp. 328-337, 2004
- Only A2.(1) or A2.(3) are satisfied, $\mathcal{A}^+$ boils down to the message pattern assumption introduced in
  Mostéfaoui A., Mourgaya E., and Raynal M., Asynchronous Implementation of Failure Detectors. *Proc. Int'l IEEE Conf. on Dependable Systems and Networks*, pp. 351-360, 2003
- Only A2.(1), A2.(2) or A2.(3) are satisfied, $\mathcal{A}^+$ boils to the assumption used in
  Mostéfaoui A., Raynal M. and Travers C., Time-free and timer-based assumptions can be combined to get eventual leadership. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):656-666, 2006

Particular cases, cont'd

If $Q(rn)$ can vary with the round numbers

- Only A2.(1) or A2.(2) are satisfied, $\mathcal{A}^+$ boils down to the eventual t-moving source assumption introduced in
  Hutle M., Malkhi D., Schmid U. and Zhou L., Chasing the weakest system model for implementing Ω and consensus. *8th Symposium on Stabilization, Safety and Security in Distributed Systems)*, LNCS #4280, pp. 576-577, 2006
- Only A2.(1) or A2.(3) are satisfied, $\mathcal{A}^+$ provides a t-moving message pattern assumption generalizing the assumption introduced in
  Mostéfaoui A., Raynal M. and Travers C., Time-free and timer-based assumptions can be combined to get eventual leadership. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):656-666, 2006

Weakening $\mathcal{A}^+$ to obtain $\mathcal{A}$

- There are a correct process p, two bounds, bound δ and bound D, and a finite round number RN_0, such that for any $rn \geq RN_0$, there is a set of processes $Q(rn)$ satisfying the following properties:
  - ⋆ There is an infinite sequence S of increasing round numbers $s_1 = RN_0, s_2, \ldots, s_k, s_{k+1}, \ldots$, such that $s_{k+1} - s_k \leq D$, (so, the round numbers in S are not necessarily consecutive)
  - ⋆ For any $s_k \in S$ there is a set of processes $Q(s_k)$ satisfying the properties A1 and A2
    - ∗ A1: $p \notin Q(rn)$ and $|Q(rn)| \geq t$
      ($\{p\} \cup Q(rn)$ is a t-star centered at p)
    - ∗ A2: For any $q \in Q(rn)$ (any point of the star), one of the following properties is satisfied
      - · (1) the process q has crashed,
      - · (2) the message alive(rn) is δ-timely,
      - · (3) the message alive(rn) is winning

$\mathcal{A}$ adds dynamicity to $\mathcal{A}^+$

- When $D = 1$, $\mathcal{A}$ boils down to $\mathcal{A}^+$
- $\mathcal{A}$ weakens $\mathcal{A}^+$ by adding another dynamicity dimension related to time: it is sufficient that the rotating t-star centered at p appears from time to time. This is why $\mathcal{A}$ defines an *intermittent rotating t-star*
- The limitation on this dynamicity dimension is expressed by the bound D

Part III

AN $\mathcal{A}^+$-based ALGORITHM

Classical basic principle: client side of p_i

- Elect the (locally) least suspected process
- $susp\_level_i[j]$ = nb of suspicions of p_i (as know by p_i)

when leader() **is invoked by the upper layer:**
let ℓ **such that**
$(susp\_level_i[\ell], \ell) = \min(\{(susp\_level_i[j], j)\}_{1 \leq j \leq n})$;
return (ℓ)

Sender side of p_i

- s_rn_i = local round number for sending $alive()$ messages
- Gossiping of $susp\_level_i[1..n]$
- Consecutive repeats separated by at most β time units

repeat regularly:
$s\_rn_i \leftarrow s\_rn_i + 1$;
for_each $j \neq i$ **do**
send alive$(s_rn_i, susp_level_i)$ *to* p_j **end_do**

- Sending rounds are not synchronized by the algorithm

Receiver side of p_i

- r_rn_i = local round number for receiving $alive()$ msgs
- $rec\_from_i[rn]$ keeps the ids of the processes from which p_i has received an alive(rn) message while $rn \geq r\_rn_i$

upon reception alive(rn, sl) **from** p_j:
for_each k **do**
$susp\_level_i[k] \leftarrow \max(susp\_level_i[k], sl[k])$ **end_do**;
if $rn \geq r\_rn_i$
then $rec\_from_i[rn] \leftarrow rec\_from_i[rn] \cup \{j\}$ **end_if**

Background task of p_i (part 1)

- $timer_i$ = local timer of p_i

when ($timer_i$ has expired) $\wedge$ ($|rec\_from_i[r\_rn_i]| \geq n - t$):
  let $suspects = \Pi \setminus rec\_from_i[r\_rn_i]$;
  for_each j **do**
    send suspicion($r\_rn_i, suspects$) *to* p_j **end_do**;
  set $timer_i$ **to** max($\{susp\_level_i[j]\}_{1 \leq j \leq n}$);
  $r\_rn_i \leftarrow r\_rn_i + 1$

- The receiving rounds are partially synchronized wrt sending rounds

Background task of p_i (part 2)

- $suspicions_i[rn, k]$ counts, as far as the receiving round rn is concerned, how many processes suspect p_k to have crashed

upon reception suspicion($rn, suspects$) **from** p_j:
  for_each $k \in suspects$ **do**
    $suspicions_i[rn, k] \leftarrow suspicions_i[rn, k] + 1$;
    if ($suspicions_i[rn, k] = n - t$)
      then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$ **end_if**
  end_do

From $\mathcal{A}^+$ to $\mathcal{A}$

- The existence of a star structure (A1) and its flashing properties (A2) can now be satisfied only on a subsequence of the rounds numbers $S = s_1, s_2, \ldots, s_k, s_{k+1}, \ldots$
  - ⋆ starting at some $s_1 = RN_0$
  - ⋆ and $\forall k$, $s_{k+1} - s_k \leq D$, D: (unknown) constant
  - ⋆ Idea: an Ω $\mathcal{A}$-based algorithm has to filter the round numbers in order to skip the irrelevant ones, i.e., the round numbers that do not belong to S

From $\mathcal{A}^+$ to $\mathcal{A}$, cont'd

- Increase $susp\_level_i[k]$ only it p_k is continuously suspected during a long enough period (measured in rounds)
- "Long enough" = during D consecutive rounds
- Approximate D with the current value of ... $susp\_level_i[k]$!

upon reception suspicion($rn, suspects$) **from** p_j:
  for_each $k \in suspects$ **do**
    $suspicions_i[rn, k] \leftarrow suspicions_i[rn, k] + 1$;
    if ($suspicions_i[rn, k] = n - t$)
  ⋆ $\wedge$ $\forall x : rn - susp\_level_i[k] < x < rn :$
  ⋆ $(suspicions_i[x, k] \geq n - t)$
      then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$ **end_if**
  end_do

Bounding all the variables (but the round numbers)

- Aim: bound each $susp\_level_i[k]$ variable
- Answer: suspects p_k only if it is the current leader!
- Even the timers are now bounded

upon reception suspicion$(rn, suspects)$ **from** p_j:
  for_each $k \in suspects$ **do**
    $suspicions_i[rn,k] \leftarrow suspicions_i[rn,k] + 1$;
    if $(suspicions_i[rn,k] = n - t)$
* $\wedge$ $\forall x : rn - susp\_level_i[k] < x < rn :$
* $(suspicions_i[x,k] \geq n - t)$
** $\wedge$ $(susp\_level_i[k] = \min(\{susp\_level_i[j]\}_{1 \leq j \leq n}))$
    then $susp\_level_i[k] \leftarrow susp\_level_i[k] + 1$ **end_if**
  end_do

Conclusion

- An assumption to implement Ω "weaker" than the previous ones
- A simple algorithm
- A deeper insight into the problem
- Possibility to define the sequence of "useful" round number according to some predefined function

What do we really need to implement Ω?

- Enough processes ($t+1$) have to be involved (star structure)
- One of them has to be correct and give a promise on its future behavior to the t other processes
- As soon as the t other processes recognize the current behavior of a process agrees with its announced promise, they stop suspecting it
- ...