

Failure Detection and Propagation in HPC systems

George Bosilca¹, Aurélien Bouteiller¹, Amina Guermouche¹, Thomas Hérault¹, Yves Robert^{1,2}, Pierre Sens³ and Jack Dongarra^{1,4}

1. University Tennessee Knoxville
2. ENS Lyon, France
3. LIP6 Paris, France
4. Manchester University, UK

CCDSC – October 4, 2016

Failure detection: why?

- Nodes do crash at scale (you've heard the story before)
- Current solution:
 - ① Detection: TCP time-out ($\approx 20mn$)
 - ② Knowledge propagation: Admin network
- Work on **fail-stop** errors assumes *instantaneous* failure detection
- Seems we put the cart before the horse 😞

Resilient applications

- Continue execution after crash of one node

Resilient applications

- Continue execution after crash of **several** nodes

Resilient applications

- Continue execution after crash of **several** nodes
- Need *rapid* and *global* knowledge of group members
 - ① **Rapid**: failure detection
 - ② **Global**: failure knowledge propagation

Resilient applications

- Continue execution after crash of **several** nodes
- Need *rapid* and *global* knowledge of group members
 - ① **Rapid**: failure detection
 - ② **Global**: failure knowledge propagation
- Resilience mechanism should **come for free**

Resilient applications

- Continue execution after crash of **several** nodes
- Need *rapid* and *global* knowledge of group members
 - ① **Rapid**: failure detection
 - ② **Global**: failure knowledge propagation
- Resilience mechanism should **have minimal impact**

Contribution

- Failure-free overhead constant per node (memory, communications)
- Failure detection with minimal overhead
- Knowledge propagation based on fault-tolerant broadcast overlay
- Tolerate an **arbitrary** number of failures
(but bounded number within threshold interval)

Outline

- 1 Model
- 2 Failure detector
- 3 Worst-case analysis
- 4 Implementation and experiments

Outline

- 1 Model
- 2 Failure detector
- 3 Worst-case analysis
- 4 Implementation and experiments

Framework

- Large-scale platform with (dense) interconnection graph (physical links)
- **One-port** message passing model
- Reliable links (messages not lost/duplicated/modified)
- Communication time on each link:
randomly distributed but bounded by τ
- Permanent node crashes

Failure detector

Definition

Failure detector: distributed service able to return the state of any node, alive or dead. **Perfect** if:

- 1 any failure is eventually detected by all living nodes and
- 2 no living node suspects another living node

Definition

Stable configuration: all failed nodes are known to all processes (nodes may not be aware that they are in a stable configuration).

Vocabulary

- Node = physical resource
- Process = program running on node
- Thread = part of a process that can run on a single core
- Failure detector will detect both process and node failures
- Failure detector mandatory to detect some node failures

Outline

- 1 Model
- 2 Failure detector**
- 3 Worst-case analysis
- 4 Implementation and experiments

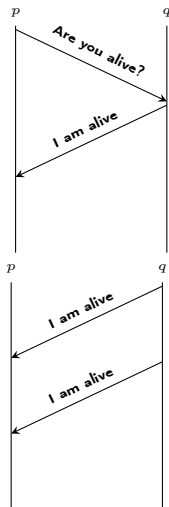
Timeout techniques: p observes q

- Pull technique

- Observer p requests a live message from q
- ☹ More messages
- ☹ Long timeout

- Push technique [1]

- Observed q periodically sends heartbeats to p
- 😊 Less messages
- 😊 Faster detection (shorter timeout)



[1]: W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. IEEE Transactions on Computers, 2002

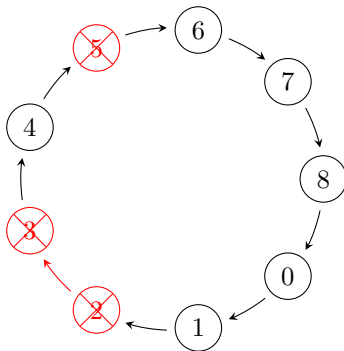
Timeout techniques: platform-wide

- All-to-all:
 - 😊 Immediate knowledge propagation
 - 😞 Dramatic overhead

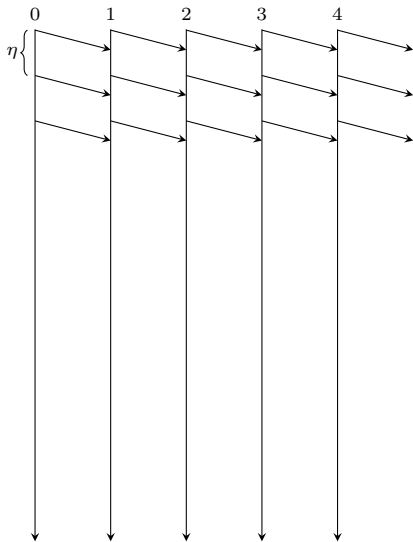
- Random nodes and gossip:
 - 😊 Quick knowledge propagation
 - 😞 Redundant/partial failure information
(observation round with n nodes selecting random target
⇒ expect $\frac{n}{e}$ nodes ignored)
 - 😞 Difficult to define timeout
 - 😞 Difficult to bound detection latency

Algorithm for failure detection

- Processes arranged as a ring
- Periodic heartbeats from a node to its successor
- **Maintain ring of live nodes**
 - Reconnect ring after a failure
 - Inform all processes



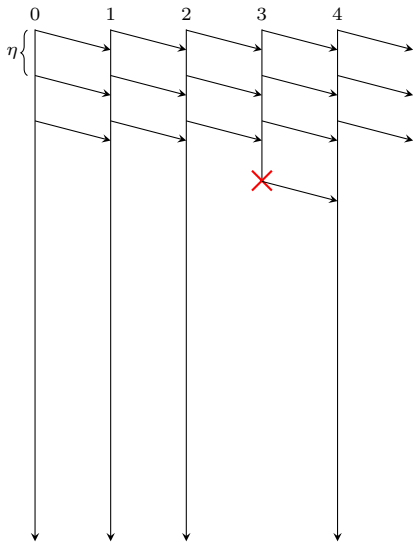
Reconnecting the ring



η : Heartbeat interval

→ Heartbeat

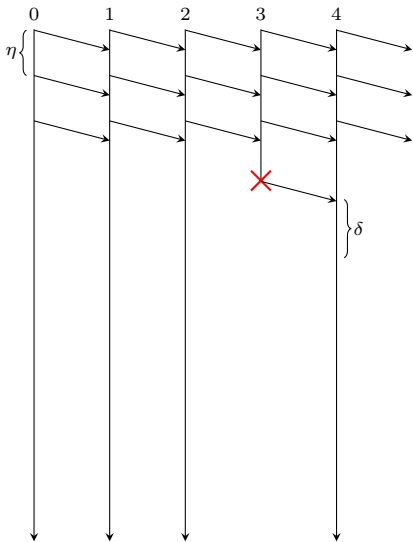
Reconnecting the ring



η : Heartbeat interval

→ Heartbeat

Reconnecting the ring

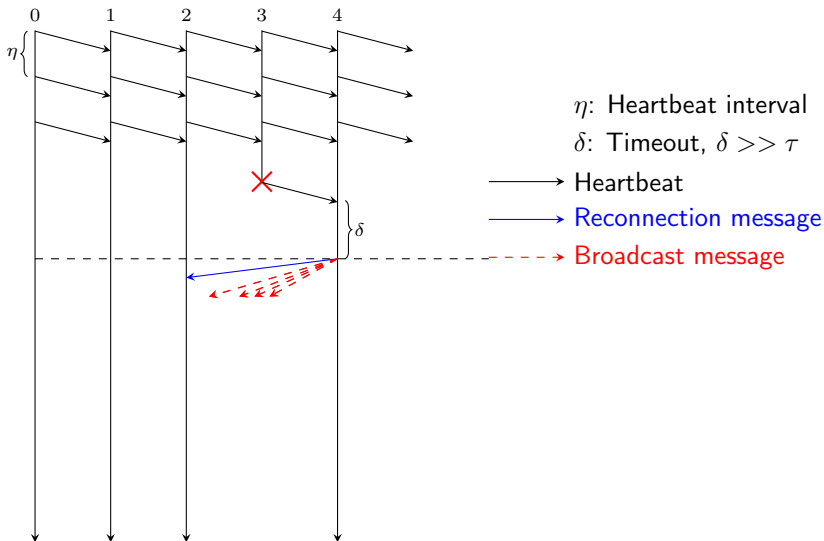


η : Heartbeat interval

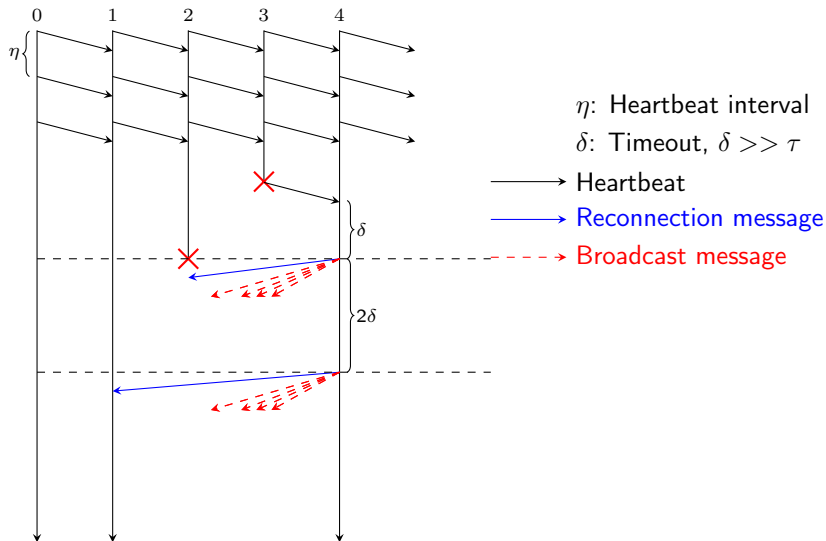
δ : Timeout, $\delta \gg \tau$

→ Heartbeat

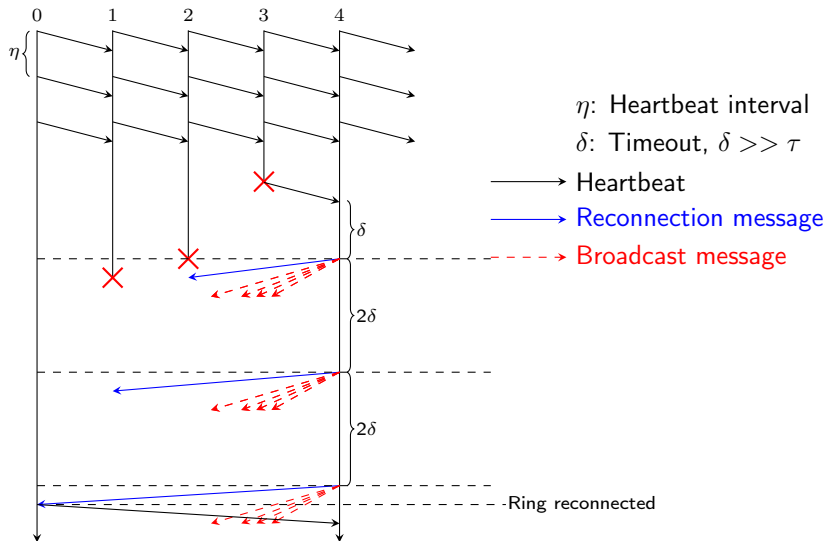
Reconnecting the ring



Reconnecting the ring



Reconnecting the ring



Algorithm

task Initialization

```

emitteri ← (i - 1) mod N
observeri ← (i + 1) mod N
HB-Timeout ← η
Susp-Timeout ← δ
Di ← ∅

```

end task

task T1: When HB-Timeout expires

```

HB-Timeout ← η
Send heartbeat(i) to observeri

```

end task

task T2: upon reception of heartbeat(emitter_i)

```

Susp-Timeout ← δ

```

end task

task T3: When Susp-Timeout expires

```

Susp-Timeout ← 2δ
Di ← Di ∪ emitteri
dead ← emitteri
emitteri ← FindEmitter(Di)
Send NewObserver(i) to emitteri
Send BcastMsg(dead, i, Di) to Neighbors(i, Di)

```

end task

task T4: upon reception of NewObserver(j)

```

observeri ← j
HB-Timeout ← 0

```

end task

task T5: upon reception of BcastMsg(dead, s, D)

```

Di ← Di ∪ {dead}
Send BcastMsg(dead, s, D) to Neighbors(s, D)

```

end task

function FindEmitter(D_i)

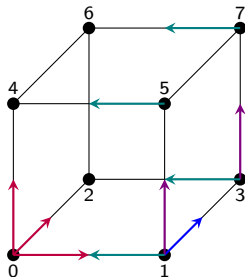
```

k ← emitteri
while k ∈ Di do
    k ← (k - 1) mod N
return k
end function

```


Broadcast algorithm

- Hypercube Broadcast Algorithm [1]
 - Disjoint paths to deliver multiple broadcast message copies
 - Recursive doubling broadcast algorithm by each node
 - Completes if $f \leq \lfloor \log(n) \rfloor - 1$
(f : number of failures,
 n : number of live processes)



Node	Node1	Node2	Node4
1	0	0-2-3	0-4-5
2	0-1-3	0	0-4-6
3	0-1	0-2	0-4-5-7
4	0-1-5	0-2-6	0
5	0-1	0-2-6-7	0-4
6	0-1-3-7	0-2	0-4
7	0-1-3	0-2-6	0-4-5

[1] P. Ramanathan and Kang G. Shin, 'Reliable Broadcast Algorithm', IEEE transaction on computers, 1998

Failure propagation

- Hypercube Broadcast Algorithm
 - Completes if $f \leq \lfloor \log(n) \rfloor - 1$ (f : number of failures, n : number of living processes)
 - Completes after $2\tau \log(n)$
- Application to failure detector
 - If $n \neq 2^l$
 - $k = \lfloor \log(n) \rfloor$
 - $2^k \leq n \leq 2^{k+1}$
 - Initiate two successive broadcast operations
 - Source s of broadcast sends its current list D of dead processes
 - No update of D during broadcast initiated by s
(do NOT change broadcast topology on the fly)

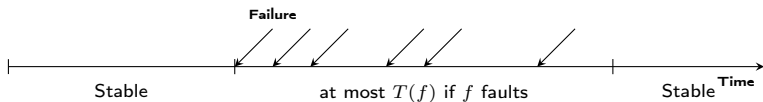
Quick digression

- Need a fault-tolerant overlay with small **fault-tolerant diameter** and easy routing
- Known only for specific values of n :
 - Hypercubes: $n = 2^k$
 - Binomial graphs: $n = 2^k$
 - Circulant networks: $n = cd^k$
 - ...

Outline

- 1 Model
- 2 Failure detector
- 3 Worst-case analysis**
- 4 Implementation and experiments

Worst-case analysis



Theorem

With $n \leq N$ alive nodes, and for any $f \leq \lfloor \log n \rfloor - 1$, we have

$$T(f) \leq f(f+1)\delta + f\tau + \frac{f(f+1)}{2}B(n)$$

where $B(n) = 8\tau \log n$.

- 2 sequential broadcasts: $4\tau \log(n)$
- One-port model: broadcast messages and heartbeats interleaved

Worst-case scenario

$$T(f) \leq f(f+1)\delta + f\tau + \frac{f(f+1)}{2}B(n)$$

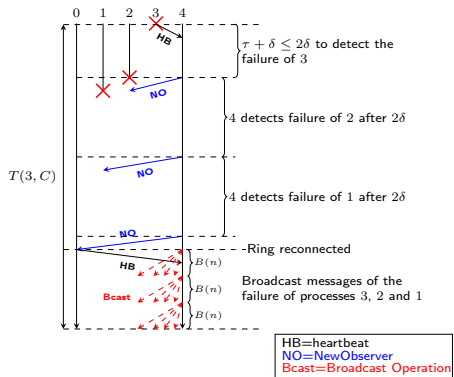
- $R(f)$ ring reconstruction time
- $T(f) \leq R(f) + \text{broadcasts (for the proof)}$
- Process p discovers the death of q at most **once**
 $\Rightarrow i$ -th failed process discovered dead by at most $f - i + 1$ processes
 \Rightarrow at most $\frac{f(f+1)}{2}$ broadcasts
- For $1 \leq f \leq \lfloor \log n \rfloor - 1$,

$$R(f) \leq R(f-1) + 2f\delta + \tau$$

Ring reconnection

$$R(f) \leq R(f - 1) + 2f\delta + \tau$$

- $R(1) \leq 2\tau + \delta \leq 2\delta + \tau$
- $R(f) \leq R(f - 1) + R(1)$
if next failure *non-adjacent* to previous ones
- Worst-case when failing nodes consecutive in the ring
- Build the ring by “jumping” over platform to avoid correlated failures



Worst-case scenario

$$T(f) \leq f(f + 1)\delta + f\tau + \frac{f(f + 1)}{2}B(n)$$

Worst-case scenario

$$T(f) \leq (f + 1)\delta + f\tau + \frac{f(f+1)}{2}B(n)$$

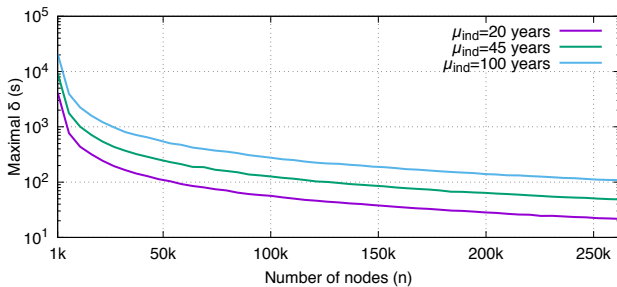
Too pessimistic!?

Worst-case scenario

- 1 If time between two consecutive faults is larger than $T(1)$, then average stabilization time is $T(1) = O(\log n)$
- 2 If f quickly overlapping faults hit non-consecutive nodes, $T(f) = O(\log^2 n)$
- 3 If f quickly overlapping faults hit f consecutive nodes in the ring, $T(f) = O(\log^3 n)$

Large platforms: two successive faults strike consecutive nodes with probability $2/n$

Risk assessment with $\tau = 1\mu s$



$$\mathbb{P}(\leq \lfloor \log_2(n) \rfloor \text{ failures in } T(\lfloor \log_2(n) \rfloor)) < 0.000000001$$

- With $\mu_{ind} = 45$ years, $\delta \leq 60s \Rightarrow$ timely convergence
- Detector generates negligible noise to applications (e.g., $\eta = \delta/10$)

Simulations

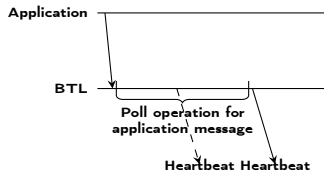
Average stabilization time \Rightarrow see paper!

Outline

- 1 Model
- 2 Failure detector
- 3 Worst-case analysis
- 4 Implementation and experiments**

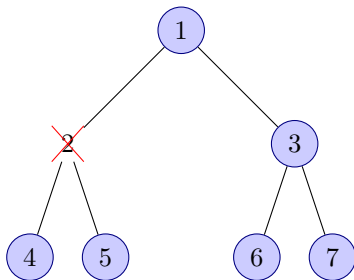
Implementation

- Observation ring and propagation topology implemented in Byte Transport Layer (BTL)
- No missing heartbeat period:
 - Implemented in MPI internal thread independently from application communications
 - RDMA put channel to directly raise a flag at receiver memory
- No allocated memory, no message wait queue
- Implementation in ULFM / OpenMPI



Case study: ULFM

- Extension to the MPI library allowing the user to provide its own fault tolerance technique
 - Failure notification in MPI calls that involve a failed process
 - ULFM requires an agreement (broadcast succeeded)
- All **live** processes need to participate
- Examples: `MPI_COMM_AGREE` and `MPI_COMM_SHRINK`

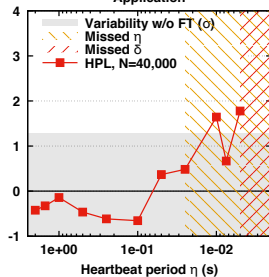
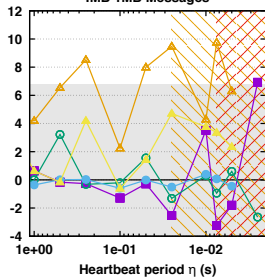
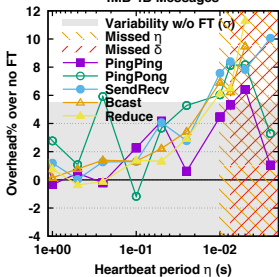


Experimental setup

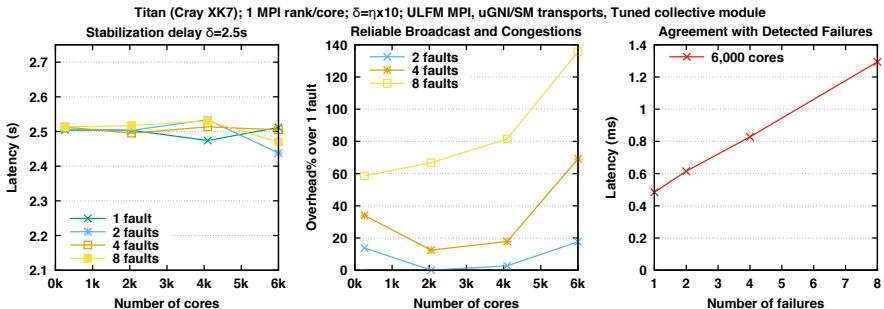
- Titan ORNL Supercomputer
 - 16-core AMD Opteron processors
 - Cray Gemini interconnect
- ULFM
 - OpenMPI 2.x
 - Compiled with `MPI_THREAD_MULTIPLE`
- One MPI rank per core
- Average of 30 times

Noise

Titan (Cray XK7); 256 MPI ranks on 256 cores; $\delta = \eta \times 10$; ULMF MPI, uGNI/SM transports, Tuned collective module
IMB 4B Messages **IMB 1MB Messages** **Application**

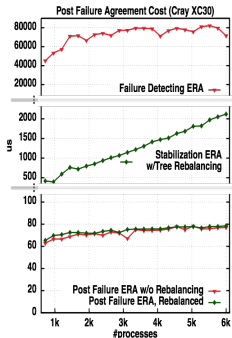


Detection and propagation delay



Consensus in ULFM without fault detector

- Provided by the system
 - 1 Timeout: Large to avoid false positive 😞
 - 2 Failures detected by ORTE, which informs mpirun, which then broadcasts
 - 😞 Non resilient binary tree structure
 - 😞 Delays on the mpirun level to start the propagation



50X improvement with failure detector 😊😊😊

Conclusion and future work

- Conclusion
 - Failure detector based on timeout and heartbeats
 - Tolerate arbitrary number of failures (but not too frequent)
 - Complicated trade off between noise, detection and risks (of not detecting failures)
 - Implementation in ULFM
 - Negligible noise
 - Quick failure information dissemination
- Future work
 - System-level implementation
 - Address trade-off between detection time and risk