An Adaptive Algorithm for Efficient Message Diffusion in Unreliable Environments

Benoît Garbinato
Fernando Pedone
Rodrigo Schmidt

Proceedings of the 2004 IEEE International Conference on Dependable Systems and Networks (DSN 2004)

Our goal

We want to solve the reliable broadcast problem in large-scale distributed systems, using as few messages as possible

We want our solution to be able to adapt to changes in the system configuration

Outline

- Distributed systems
- Large scale systems
- Reliable broadcast
- Probabilistic approach
- Adaptation & Optimality

Distributed systems

"A distributed system is one that stops you from getting any work done when a machine you've never even heard of crashes."

Leslie Lamport quoted by Sape Müllender in Distributed Systems, 2nd edition. Addison-Wesley, 1993.

- As slow as the slowest
- As weak as the weakest

$\Rightarrow$ In large-scale systems, this is even worse...

Large scale

- Many nodes (processes)

  and/or

- Long-distances (high-latency)

  ⇒ Changes occur frequently

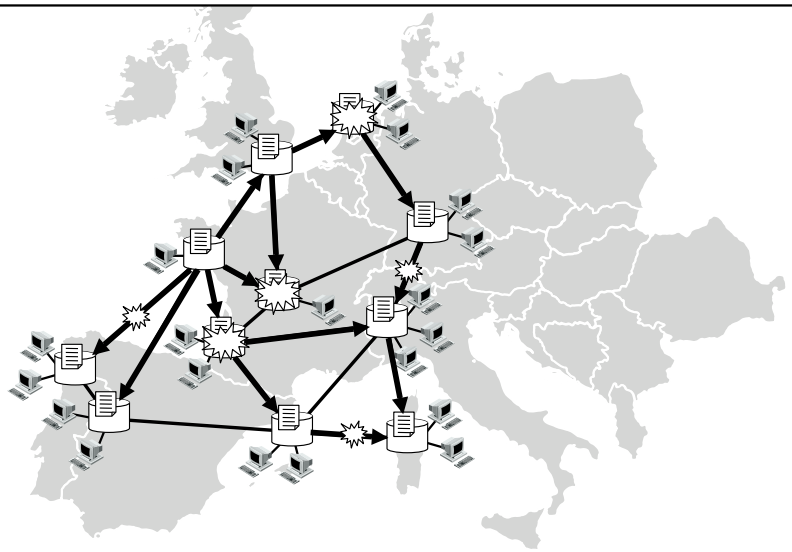
  ⇒ Difficult to act globally

Reliability

- Try to solve typical problems, e.g., consensus, reliable broadcast, atomic commitment, etc., despite the occurrence of failures
- A solution depends on the actual failure model, e.g., crash-stop, crash-recovery, Byzantine, etc.

Gossiping algorithms

Principle: only talk to some of your neighbors



Probabilistic model

Model definition

- The system is defined as a graph $G = (\Pi, \Lambda)$
  $\Pi = \{p_1, p_2, .. p_n\}$ is a set processes (vertices)
  $\Lambda = \{l_1, l_2, .. l_m\} \subseteq \Pi \times \Pi$ is a set of links (edges)
- Processes communicate by message passing
- A failure probability P_i is associated with each process p_i and a message-loss probability L_j is associated with each link l_j. This probabilities set defines a configuration.

Probabilistic reliability

Problem statement

Validity. If a process broadcasts a message m, then it eventually delivers m.

Agreement. If a process delivers a message m, then every process eventually delivers m **with probability K**.

Integrity. For any message m, every process delivers m at most once, and only if m was previously broadcast by some process.

Our goal revisited

- We want to provide a solution the probabilistic reliable broadcast in large-scale systems
- We want our solution to adapt to changes in the unreliability probabilities of processes & links
- We want to measure the effectiveness of our adaptive strategy with respect an algorithm that minimizes the number of messages

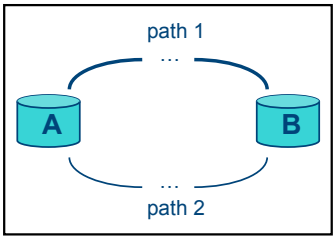
Optimality & adaptation

Optimality. A probabilistic reliable broadcast algorithm O is optimal to some configuration C w.r.t. the number of messages if there is no algorithm X such that processes executing X in C exchange fewer messages than processes executing O in C.

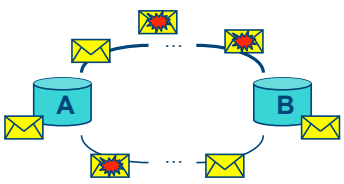
Adaptation. A probabilistic reliable broadcast algorithm A is adaptive to some configuration C if and only if the number of messages exchanged by processes executing A in C in response to a reliable broadcast is **eventually** equal to the number of messages exchanged by processes executing O in C

Optimal algorithm (1)

- Use **adaptive** gossiping, not **random** gossiping (traditional)
- **Example:**
  - Failure probability of path 1 = L
  - Failure probability of path 2 = αL, with α > 1 (path 1 more reliable than 2)
- Probability **K** of reaching B from A

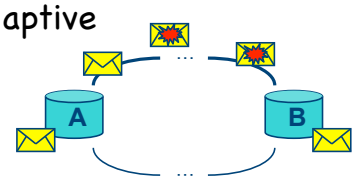


Random



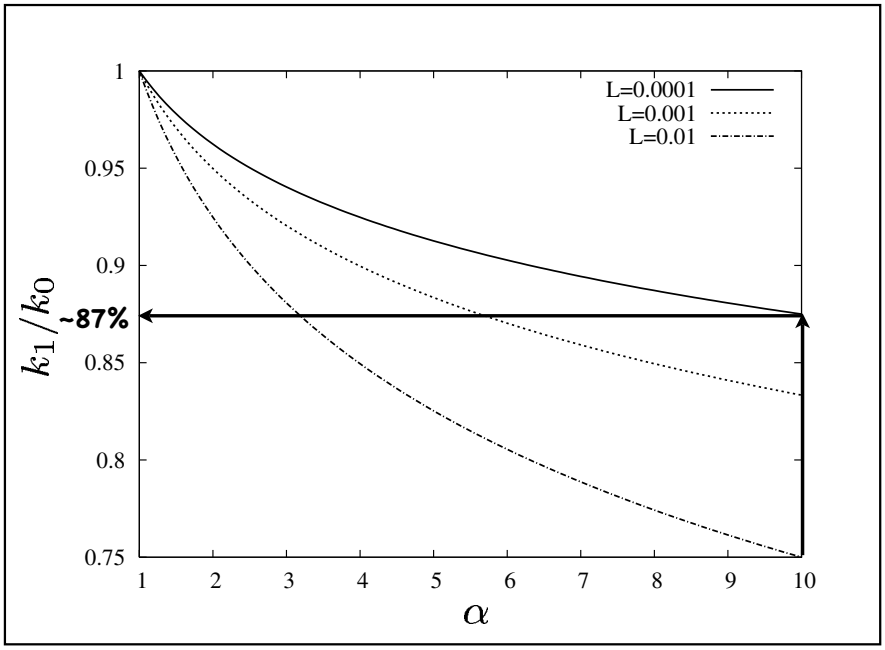
$$K = 1 - L^{k_0/2}(\alpha L)^{k_0/2} = 1 - (\sqrt{\alpha}\, L)^{k_0}$$

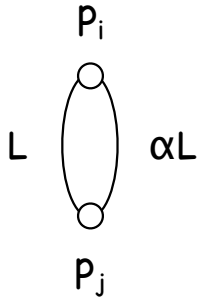
Adaptive



$$K = 1 - L^{k_1}$$

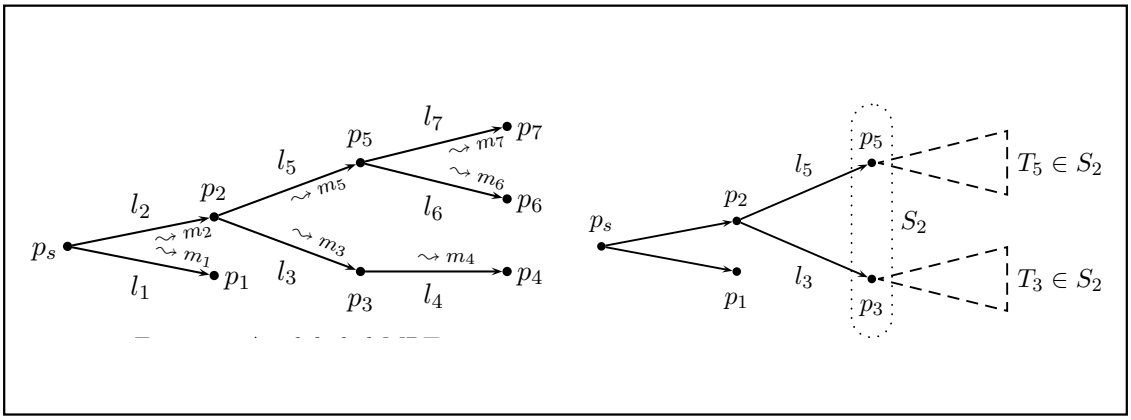
Optimal algorithm (2)





Maximum reliability tree

The maximum reliability tree is a spanning tree containing the most reliable path in G connecting all processes in Π



The reach function

Given a tree T_i and a vector $\vec{m}_i$, the reach function computes the probability that all processes in T_i are reached by at least one message

$$reach(T_i, \vec{m}_i) = \begin{cases} 1 & \text{if } T_i = \perp \\ \prod_{T_j \in S_i} (1 - [1 - (1 - P_i) \times (1 - L_j) \times (1 - P_j)]^{\vec{m}_i[j]}) \times reach(T_j, \vec{m}_j) & \text{otherwise} \end{cases}$$

$$reach(T_i, \vec{m}_i) = \prod_{j=1}^{n-1} 1 - [1 - (1 - P_{pred(j)}) \times (1 - L_j) \times (1 - P_j)]^{\vec{m}_i[j]}$$

Optimization problem

With $\lambda_j = 1 - (1 - P_{pred(j)}) \times (1 - L_j) \times (1 - P_j)$, we end up with the following optimization problem:

$$\begin{aligned} \textit{minimize} \quad & c(\vec{m}) = \sum_{j=1}^{|\vec{m}|} m[j] \\ \textit{subject to} \quad & r(\vec{m}) = \prod_{j=1}^{|\vec{m}|} 1 - \lambda_j^{m[j]} \geq K \end{aligned}$$

The optimize function

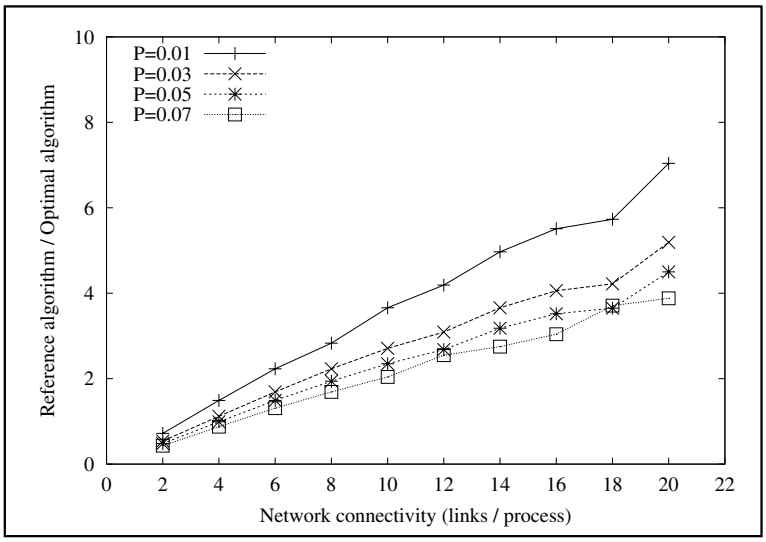
We use a greedy algorithm, since our optimization problem is itself greedy

function $optimize(mrt, K)$
  $\vec{m} \leftarrow (1, 1, 1, \cdots, 1)$
  while $r(\vec{m}) < K$ **do**
    let $\vec{u}_j$ be such that $\frac{r(\vec{m}+\vec{u}_j)}{r(\vec{m})}$ is maximum
    $\vec{m} \leftarrow \vec{m} + \vec{u}_j$
  return $\vec{m}$

Optimal algorithm

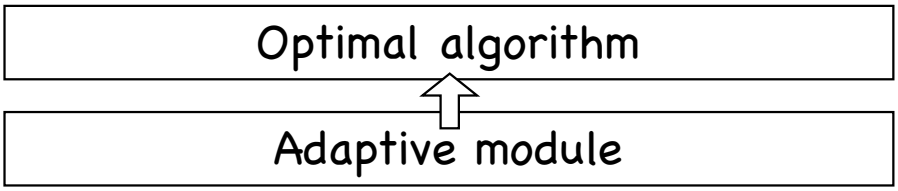
```
1:  To execute broadcast(m) do
2:      mrt_k ← mrt_k(G, C)
3:      propagate(m, mrt_k, p_k)
4:      deliver(m)

5:  when receive (m, mrt_j) for the first time
6:      propagate(m, mrt_j, p_k)
7:      deliver(m)

8:  function propagate(m, mrt_j, p_k)
9:      m_j ← optimize(mrt_j, K)
10:     for all subtree T_i ∈ S_{j,k} do
11:         repeat m_j[i] times
12:             send (m, mrt_j) to p_i
```

Performance of the optimal algorithm



From optimal to adaptive

- Our algorithm is proven to be optimal when it has exact knowledge of the system configuration
- By replacing such (unrealistic) exact knowledge with a module that approximates the changing configurations, we get an adaptive algorithm
- As soon as the approximation module converged towards the actual system configuration, we are again optimal



Adaptive algorithm

```
1:  INITIALIZATION:
2:    for all p_i ∈ Π do
3:      initializeReliability(C_k[p_i])
4:      C_k[p_i].d ← ∞
5:      C_k[p_i].seq ← 0
6:      C_k[p_i].suspected ← 0
7:      Δ_k[p_i] ← δ
8:    C_k[p_k].d ← 0

9:    Λ_k ← {l_{k,i} | p_i ∈ neighbors(p_k)}
10:   for all l_i ∈ Λ_k do
11:     initializeReliability(C_k[l_i])
12:     C_k[l_i].d ← 0

13: TO UPDATE (Λ_k, C_k):

14:   every Δ_k[p_k] do :
15:     C_k[p_k].seq ← C_k[p_k].seq + 1
16:     for all p_i ∈ neighbors(p_k) do
17:       send (Λ_k, C_k) to p_i

18:   when received (Λ_j, C_j) from p_j do                {Event 1}
19:     adjust ← C_k[p_j].suspected −
                          (C_j[p_j].seq − C_k[p_j].seq)
20:     C_k[p_j].suspected ← 0
21:     if adjust > 0 then
22:       increaseReliability(C_k[l_{k,j}], adjust)
23:       if adjust > 1 then Δ_k[p_j] ← Δ_k[p_j] + δ
24:     if adjust < 0 then
25:       decreaseReliability(C_k[l_{k,j}], |adjust|)
26:     for all p_i ∈ Π do
27:       selectBestEstimate(C_k[p_i], C_j[p_i])
28:     for all l_i ∈ (Λ_k ∩ Λ_j) do
29:       selectBestEstimate(C_k[l_i], C_j[l_i])
30:     for all l_i ∈ Λ_j − (Λ_k ∩ Λ_j) do
31:       C_k[l_i] ← C_j[l_i]
32:       C_k[l_i].d ← C_k[l_i].d + 1
33:     Λ_k ← Λ_k ∪ Λ_j
                                                          {Event 2}
34:   when not[updated C_k[p_j], p_j ≠ p_k, in the last Δ_k[p_j]] do
35:     C_k[p_j].d ← C_k[p_j].d + 1
36:     if p_j ∈ neighbors(p_k) then
37:       C_k[p_j].suspected ← C_k[p_j].suspected + 1
38:       decreaseReliability(C_k[p_j], 1)
39:       decreaseReliability(C_k[l_{k,j}], 1)

40:   every Δ_tick do                                     {Event 3}
41:     increaseReliability(C_k[p_k], 1)
                                                          {Event 4}
42:   when recovering from a crash lasting n × Δ_tick do
43:     decreaseReliability(C_k[p_k], n)
```

```
1:  Initialization
2:    U ← 100                  {precision of probabilistic intervals}

3:  function initializeReliability(estimate)
4:    with estimate do
5:      for all u = 1..U do
6:        P_{F|B}[u] ← (2u−1)/(2U)          {probabilistic intervals}
7:        P_B[u] ← 1/U                      {with equal initial beliefs}

8:  function decreaseReliability(estimate, factor)
9:    with estimate repeat factor times
10:     for all u = 1..U do
11:       P_B[u] ← P_B[u]×P_{F|B}[u] / Σ_{v=1}^{U} P_B[v]×P_{F|B}[v]

12: function increaseReliability(estimate, factor)
13:   with estimate repeat factor times
14:     for all u = 1..U do
15:       P_B[u] ← P_B[u]×(1−P_{F|B}[u]) / Σ_{v=1}^{U} P_B[v]×(1−P_{F|B}[v])
```


Bayesian inference (1)

$$P_{B|F}[u] = \frac{P_{F|B}[u] \times P_B[u]}{\sum_{v=1}^{U} P_{F|B}[v] \times P_B[v]}$$

Bayesian inference (2)

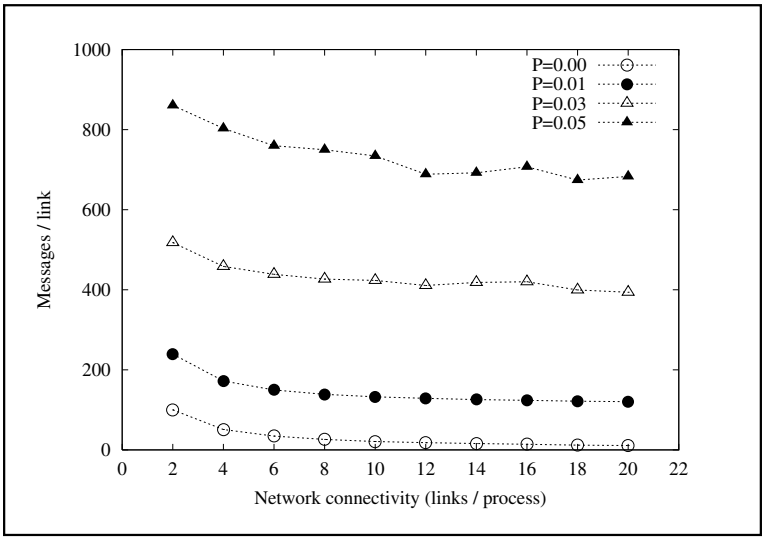
u	$C_k[p_i].P_{F\|B}[u]$	$C_k[p_i].P_B[u]$
1	[0.0 , 0.2)	0.2
2	[0.2 , 0.4)	0.2
3	[0.4 , 0.6)	0.2
4	[0.6 , 0.8)	0.2
5	[0.8 , 1.0]	0.2

(a) Initial configuration

u	$C_k[p_i].P_{F\|B}[u]$	$C_k[p_i].P_B[u]$
1	[0.0 , 0.2)	0.04
2	[0.2 , 0.4)	0.12
3	[0.4 , 0.6)	0.20
4	[0.6 , 0.8)	0.28
5	[0.8 , 1.0]	0.36

(b) After a failure suspicion

Adaptation convergence



Question?