TORSTEN HOEFLER, AMNON BARAK, AMNON SHILOH, ZVI DREZNER

Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems
Failures in large-scale computing system

- The number of components grows
  - More and more transistors used
  - But also more racks, cabinets, cables, power supplies, etc.
  - Everything at a nearly constant reliability per part
- Things will fail!
  - Wang et al., 2010: “Peta-scale systems: MTBF 1.25 hours”
  - Brightwell et al., 2011: “Next generation systems must be designed to handle failures without interrupting the workloads on the system or crippling the efficiency of the resource.”
  
  Checkpoint/restart will take longer MTBF!
- We need to enable applications to survive failures
  - ... to reach Petascale Exascale!
  - Like they did for decades in distributed systems!
Distributed systems scenarios

- **Loosely consistent systems based on gossip**
  - Not all nodes always up to date
  - Sometimes eventual consistency
  - Weak ordering guarantees
  - Hard to control in general but may work well (e.g., load balancing)

- **Strongly consistent systems based on atomic broadcast/consensus**
  - Ordering guaranteed
  - Can survive up to k node failures, latency of k
  - Very limited in scalability
  
  *Check our work on AllConcur at HPDC’17 though!*

- **High-performance systems are specialized**
  - FARM – Fast Remote Memory (consistent FT database)
  - Corrected Gossip for group communications (this paper)
Specialized to HPC? Let’s start with the simplest operation - broadcast

- **Gossip?**
  - If root or message received: send to random other node until some global time expires
  - Proven to be very effective
  - Not strongly consistent 😊
  - Nice theory
    - needs $1.64 \log_2 n$ rounds to reach all \( w.h.p. \)
  - But for \( N=1000 \)
    - 17 rounds only color all nodes 95% of the time

- **Very problematic for BSP-style applications**
But how does FT-MPICH do this? Buntinas’ FT broadcast

- Uses a dynamic tree, each message contains information about children at next levels
- Children propagate back to root, relying on local failure-detectors

- Complex tree rebuild protocol
- Root failure results in bcast never delivered
- At least $2 \log_2 n$ depth!
But how does FT-OpenMPI do this? Binomial graph broadcast

- Use fixed graph, send along redundant edges
- Binomial graphs: each node sends to and receives from $\log_2 n$ neighbors

- Can survive up to $\log_2 n$ worst-case node failures
  - In practice much more (not worst-case)
How to beat these algorithms?

- The power of randomness: gossip but **not just** gossip!
- Combine the probabilistic gossip protocol with a **deterministic correction protocol**

Corrected gossip turns Monte Carlo style gossiping algorithms into Las Vegas style deterministic algorithms!

- **But what is a fault-tolerant broadcast? Root failures, arbitrary failures?**
  - Assuming fail-stop, four criteria need to be fulfilled:
    1. Integrity (all received messages have been sent)
    2. No duplicates (each sent message is received only once)
    3. Nonfaulty liveness (messages from a live node are received by all live nodes)
    4. Faulty liveness (messages sent from a failed node are either received by all or none live nodes)
- **We relax 3+4 a bit: three levels of consistency**
  1. Not consistent (we provide an improvement over normal gossiping)
  2. Nearly consistent (assuming no nodes fail during the correction phase, practical assumption)
  3. Fully consistent (any failures allowed)
First algorithm: OCG (Opportunistic Corrected Gossip)

- Not consistent, works w.h.p. --- let’s first consider just gossiping

Are all these redundant messages efficient?
First algorithm: OCG (Opportunistic Corrected Gossip)

Number of reached nodes

Optimal deterministic Fibonacci tree

Number of reached nodes becomes inefficient

Optimal deterministic Fibonacci tree

Optimal deterministic Fibonacci tree
First algorithm: OCG (Opportunistic Corrected Gossip)

- OCG main idea: run gossip for a while and then switch to a ring-correction protocol
  - Every node that received a message sends it to \((\text{rank} + 1) \% \text{nranks}\)

- Each message may be received twice
  - But this depends on when we switch! But what is the longest uncolored chain?
The longest uncolored chain!

99% probable longest uncolored chain

gossip becomes inefficient

Time

Nodes

K

opt

c(t)
First algorithm: OCG (Opportunistic Corrected Gossip)

- When to switch from gossip to correction?
  - Well, when the expected number of correction steps is small and gossip is inefficient
- We can bound the probability of a longest chain of length k
  - In terms of the LogP parameters, T (gossip time), and N (nranks)

\[ T_{opt}^{OCG} = \arg\min_{T} (T + 2L + (2 + \overline{K})O) \]
OCG Consistency

- OCG is more efficient than gossip but does not guarantee that all nodes are reached (even w/o failures)

- So we need to check that they were actually reached!
Second algorithm: CCG (Checked Corrected Gossip)

- CCG sends to the next node until it sent to a node it received from (i.e., knows that node was alive!)
  - Since the node it received from also sent, it “knows” that all other nodes have been covered!

- CCG guarantees that all nodes are reached unless a node dies in the middle of the correction phase!
  - And another node assumes it finished its job!
Second algorithm: CCG (Checked Corrected Gossip)

- When to switch from gossip to correction?

- A bit later than OCG
Third algorithm: FCG (Failure-proof Corrected Gossip)

- FCG can protect from $f$ failures – similar to CCG but instead of aborting to send when heard from one, it waits to hear from $f+1$ other nodes!
- So any $f$ nodes can fail and it will still succeed (keep sending)

- Wait, what if there are less than $f+1$ nodes reached during gossip and they somehow die in the middle of the protocol?
  - So we need to involve the non-gossip-colored nodes
  - They will wait to hear from a gossip-colored nodes to exit
  - If no such exit signal comes within a timeout period, panic!
  - In panic mode, send to every other node
  - Every node that receives panic messages also panics
  - This guarantees consistency (at a high cost)

- Panic mode is extremely unlikely in practice (much less likely than the failing of binomial graphs)
  - Likelihood can be reduced arbitrarily with gossiping time!
  - So panic is just a theoretical concern (to proof correctness)
Observations and Optimizations

- Why the ring topology?
  - One could choose different topologies (e.g., broadcast trees), we did not find a better practical one
  - This seems to be an interesting research topic

- Optimization: bidirectional
  - In fact, all our algorithms send backwards and forward along the ring
    
  *We skipped it to simplify the explanation*
  - Buys a factor of two, very practical (very impactful for CCG/FCG)

- Does the principle generalize
  - We believe so, more algorithms to come!

- Both the binomial graphs and FCG require to pick an $f$, is there a total consistency?
  - Only if $f=N-2$, which is not practical
  - Yet, both algorithms can be configured for an arbitrarily high success probability!
Case study: TSUBAME 2.0

- **TiTech machine, published failure logs**
- **MTBF = 18304 hours**
  - Assume 12 hour run on 4096 nodes = 2.69 failures
- **We compare all algorithms and report**
  1. Expected latency
  2. Expected work
  3. Expected inconsistency

For CCG/OCG/FCG, we simulate until the nonparametric CI was within 2% of the median

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\hat{\tau}$</th>
<th>$T$</th>
<th>lat</th>
<th>work</th>
<th>incon.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOS [12]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOS [12]</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCG</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCG</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCG</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCG</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCG</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCG</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIG [2]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIG [2]</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFB [8]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFB [8]</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scaling – Without failures

![Scaling Graph](image-url)

- BFB
- BIG
- FCG
- CCG
- OCG
- opt
Scaling – With failures (expected for 12 hours on TSUBAME 2.0)
Summary and Conclusions

- **New principle to implement fault-tolerant group communications**
  - Combines randomness and determinism – Las Vegas style algorithms

- **Three versions with growing consistency**
  - Opportunistic Corrected Gossip
  - Checked Corrected Gossip
  - Failure-proof Corrected Gossip

- **Analytic models to selecting parameters**
  - Fast to compute gossiping time

- **Now we need to see if it’s practical**
  - May need some hardware support

*In a trivial implementation, wasted o dominate!*