Distributed Error- Confinement

Shay Kutten (Technion)

with

Boaz Patt-Shamir (Tel Aviv U.)
Yossi Azar (Tel Aviv U.)
(1) What is error confinement?
(2) The new “agility” measure for fault tolerance
(3) The new “core- bootstrapping” idea for algorithm.
(4) Optimization question and answer for “core” construction.
Motivation: “error propagation” (example)

(1) Assume no fault:

My distance to C via S: 7+4=11

Message from S to A:

Internet routing: Node A compute shortest path to C based on messages from S.
Motivation: “error propagation” (example)

(2) with fault (at B):

My distance to C via S: $7+4=11$

State corrupting fault (adversary modifies data memory)
State corrupting fault (self stabilization): Not malicious! Just a one time change of memory content.
Motivation: “error propagation” (example)

(2) With fault (at B):

My distance to C via S: $7+4=11$

Message from S to A:
- Distance 7 to C

Traffic to C

Fault
Motivation: “error propagation” (example)

(3) B’s fault propagated to A

Message from S to A:

My distance to C via B: 
2 + 0 = 2

Traffic to C

distance 7 to C

distance 0 to C
Motivation: “error propagation” (example)

B’s fault propagated to A

My distance to C via B: 2 + 0 = 2

Message from S to A:

(4) Traffic to C is sent the wrong way as a result of the fault propagation
This is, actually, how the Internet (than Called “ARPANET”) crashed in 1980

I have distance 0 to everybody

fault
“Error confinement”: non faulty node A outputs only correct output (or no output at all)

Sounds impossible?

I do not believe you!

My distance to C via S:
\[7 + 4 = 11\]
Error Confinement (Formally)

- $\Pi$: problem specification, $P$: protocol.
- $P$ solves $\Pi$ with **error confinement** if for any execution of $P$ with behavior $\beta$ (possibly containing a state corrupting fault), there exists a behavior $\beta' \in \Pi$ & for all non-faulty nodes $v$: $\beta'_v = \beta_v$

("stabilization" deals also with faulty nodes) (behavior- ignoring time)
Talk Overview

(1) What is error confinement?
(2) The new “agility” measure for fault tolerance.
(3) The core-bootstrapping idea for algorithm
(4) Optimization question and answer for “core” construction.
Introducing a new measure of fault resilience:
The resilience of a protocol is smaller at first.

Environment (e.g. user)

Input is given to $S$ at time $t_0$
The resilience of a protocol is smaller at first (cont.)

Environment (e.g. user) gives Input is to S at time $t_0$

If adversary changes the state of S at time $t_f$ shortly after the input
The resilience of a protocol is smaller at first (cont.)

Environment (e.g. user) gives Input to S at time $t_0$

If adversary changes the state of S at time $t_f$ shortly after the input

then the input is lost forever
The resilience of a protocol grows with time.

However, a fault, even in S, can be tolerated if it waits until after S distributed the input value.
The resilience of a protocol grows with time (cont.)

However, a fault, even in S, can be tolerated if it waits until after S distributed the input value.
The resilience of a protocol grows with time.

A fault even in S can be tolerated if it “waits” until after S distributed the input value.
The resilience of a protocol grows with time.

A fault even in S can be tolerated if it “waits” until after S distributed the input value.
The resilience of a protocol grows with time.

To destroy the replicated value the adversary needs to hit more nodes at $t_f > t_1 > t_0$. 

[$\text{time}$ $t_0$ $t_1$ $t_f$ $t_0$]
The resilience continues to **grow** with time.

If no faults occurred by some later $t_3$, then the input is replicated even further.
The resilience continues to **grow** with time.

The later the faults, the more faults can be tolerated.
The later the faults, the more faults can be tolerated if the protocol is designed to be robust.
“Narrow” cone $\rightarrow$ a LESS fault tolerant algorithm

Slower replication $\rightarrow$ less nodes offer help
A “Wider” cone \(\rightarrow\) a more fault tolerant algorithm

Replication to more nodes faster
So, a recovery of corrupted values is theoretically possible, for an adversary that is constrained according to a space-time-cone, but what is the algorithm that does the recovery?
Constraining faults: Agility

- **$c$-constrained environment**: environment generating faults $t_f$ time units after the input, ($c \leq 1$), only in: minority of $\cdot |Ball_S(c \cdot t_f)|$ nodes.

- **Algorithm** with agility $c$: Broadcast algorithm that guarantees error confinement against $c$-constrained environments.
Algorithm’s “agility” measures the rate the constraint on the adversary can be lifted.
Talk Overview

(1) What is error confinement?
(2) The new “agility” measure for fault tolerance.
(3) The new “core-bootstrapping” idea for algorithm.
(4) Optimization question and answer for “core” construction.
The message resides at some nodes we term “core”
A node can join the core when it “made sure” it heard the votes of all *core* nodes.
A node can join the core when it “made sure” it heard the votes of all core nodes.
A node can join the core when it “made sure” it heard the votes of all core nodes.
A node can join the core when it “made sure” it heard the votes of all core nodes.
and even the fault can be corrected
and even the fault can be corrected
Let us view again the join of one node
If \textit{core} is such that adversary’s constraint allows hit of only a minority of the \textit{core}...

Then the message passes to the new node correctly.
If $core$ is such that adversary’s constraint allows hit of only a minority of the $core$...

Then the message passes to the new node correctly.

Disclaimer: any connection to Actual historical rivalry is coincidental.
If *core* is such that adversary’s constraint allows hit of only a minority of the *core*...

Then the message passes to the new node correctly.

Disclaimer: any connection to Actual historical rivalry is coincidental
“Error confinement”: non faulty node A outputs Only correct output (or no output at all)
and even the fault can be corrected
When the core grow, the algorithm can withstand more faults.
Talk Overview

(1) What is error confinement?
(2) The new “agility” measure for fault tolerance.
(3) The new “core- bootstrapping” idea for algorithm.

(4) Optimization question and answer for “core” construction.
Dilemma- should we add a node to core ASAP?
Dilemma- should we add a node to core ASAP?

Advantage- enlarges the core *now*.
Dilemma- should we add a node to core ASAP?

Disadvantage: slows future core growth
Dilemma - should we add a node to core ASAP?

Disadvantage: slows future core growth
Dilemma- should we add a node to core ASAP?

Disadvantage: slows *future* core growth
Stages of core growth
(example: bad greedy policy)

Greedy core growth $\rightarrow O(D^2)$ time to reach diameter $D$ $\rightarrow$ Agility $= \frac{D}{D^2} = \sqrt{D}$
Optimize agility subject to feasibility: Core at time $T_i$ as a function of $\text{Core}(T_{i-1})$. 

$\text{Core}(T_{i-1})$
Agility at time $T_i$: Radius of Core at time $T_i$ Divided by time ($T_i: i'th$ time Core grows)

Core radius

$R_1$, $R_2$
Agility at time $T_i$ : Radius of Core at time $T_i$ Divided by time ($T_i : i^{th}$ time Core grows)

We calculated optimal $T$’s, $R$’s.
Agility at time $T_i$: Radius of Core at time $T_i$ divided by time ($T_i$: $i$-th time Core grows)

We calculated optimal $T$'s, $R$'s. Constant Agility.
Agility at time $T_i$ : Radius of Core at time $T_i$ Divided by time ($T_i : i$-th time Core grows)

Core radius

We calculated optimal $T$’s, $R$’s. Constant Agility. Optimal number of core increases (logarithmic!).
Additional results

- An error confined protocol to compute distances from a node $S$ (Bellman-Ford’s algorithm is self stabilizing but is not error-confined).
- An error confined protocol for broadcast with correct source.
- Lower bound (mandatory slow down): Consider an error-confined algorithm for BROADCAST, even with correct source. Then no correct node $v$ outputs before time $2 \cdot \text{dist}(\text{source}, v)$, even in the absence of faults.
- Generalizations, practical considerations.
Some related notions

• *Self stabilization* [Dijkstra, Lamport] would bring the system to a consistent global state eventually.

If the input is re-injected repeatedly (as in the routing information example) this corrects the states eventually.

If the input is not reintroduced by the environment, then the input may be lost forever.

• *Local checking* [Afek-K-Yung90], or *local detection* [Awerbuch-P-Varghese91], together with *self stabilizing reset* [KatzPerry90, AKY90, AroraGouda90, APV91] would yield agility $1/|V|$. 
Some related notions

• fault local algorithms [K-Peleg95], or Time adaptive [KP97], or fault containment [GhoshGuptaHermanPemmaraju96] or local stabilization [AroraZhang03] would allow a “small” (relative to the number of faults) error propagation.

• Snap stabilization [BuiDattaPetitVillain99] considers only the case that some nodes performed a special “purifying” “initiation” action after the faults. A fault may propagate to a non-initiator node until it communicate with initiators.

• Local stabilization of [AfekDolev97] assumes a node can detect its own faults (since a fault puts a node in a random state).
Open problems

Many!