The Weakest Failure Detector for Consensus

References and Thanks

- Gärtner, Guerraoui and Kouznetsov, The CHT Play
  - A highly recommended informal and very accessible overview of the CHT proof
- Some of the following slides were provided by Rachid Guerraoui
What we need to prove

- To prove that a failure detector class $C$ is the weakest for a problem $P$, one needs to show that
  - If: $P$ can be solved with $D \in C$, and
  - Only if: For all $C'$ that can be used to solve $P$, $C' \geq C$
- $\diamond S = \diamond W$ is sufficient for Consensus with $n > 2t$
- Need to prove that for all $D$ that can be used to implement Consensus, $D \geq \diamond W$ with $n > 2t$

The Outline

- Define a failure detector $\Omega$ (*leader oracle*):
  - Output: process id
  - Eventually all correct processes permanently output the same process id $p$, and $p$ is correct
- Lemma 1: For any failure environment: $\Omega \geq \diamond W$
  - Proof: ?
- Lemma 2: For any failure environment:
  - If $D$ solves Consensus, then $D \geq \Omega$
  - Proof: ? 😊

A question: Is it possible that $\Omega > \diamond W$?
The Outline

• Theorem: For any failure environment:
  If D can be used to solve Consensus, then $D \geq \diamond W$

Proof:
• If D solves Consensus, then $D \geq \Omega$
  (Lemma 2). $\Omega \geq \diamond W$ (Lemma 1).
  Transitivity: If D solves Consensus, then $D \geq \diamond W$

D solves Consensus $\Rightarrow D \geq \Omega$

• Let A be a consensus algorithm using D
• Construct an algorithm T that emulates $\Omega$
  on top of D
Overview of the emulation

1. The exchange
2. The simulation
3. The tagging
4. The stabilization
5. The extraction

(1) The Exchange

• Every process periodically queries its failure detector module (D) and sends all outputs it has seen to all
• A process builds a growing DAG using the outputs provided by other processes
• A vertex of the DAG is a triple:
  – (process, f. d. value, f. d. query#)
• An arrow \((p_1,d_1,k_1) \rightarrow (p_2,d_2,k_2)\) means that \(p_1\) saw \(d_1\) before \(p_2\) saw \(d_2\)
(1) The Exchange Algorithm

DAG := empty graph;
k := 0;
Forever do:
• k := k+1;
• p receives (q,DAG_q)  // maybe null
• d := output of p’s failure detector
• DAG := DAG ∪ DAG_q;
  Add [p,d,k] to DAG and edges from all vertices of DAG to to [p,d,k];
  Send (p,DAG) to all processes
(1) The Exchange Algorithm

Properties of Local DAGs

• For any correct process $p$ and time $t$
  (1) $DAG_p(t)$ is transitively closed
    An easy induction
  (2) There is a time $t' \geq t$, $d$ and $k$ such that $\forall v \in Vertices(DAG_p(t))$, $v \rightarrow (p, d, k)$ is an edge of $DAG_p(t')$
Properties of Local DAGs

• The DAG of each correct process is ever increasing finite approximation of the same infinite limit graph
  – The common portion of correct process DAGs grows without limit

(2) The Simulation

• Every process $p_i$ uses its DAG to simulate runs of A in the system, i.e., every process locally plays the role of all other processes
• Whenever $p_i$ updates its DAG, $p_i$ triggers runs of A for:
  – All paths in the DAG
  – All input vectors $I_0, I_2, \ldots, I_n$, where $I_i$ makes processes $p_1$-$p_i$ propose 1 and the rest propose 0
(2) The Simulation

$p_i$ simulates runs of $A$ for each
(0000), (1000), (1100), (1110), (1111)

Forever do:
• $p$ receives $(q,DAG_q)$  // maybe null
• $d :=$ output of $p$’s failure detector
• $DAG := DAG \cup DAG_q$;
  Add $[p,d]$ to $DAG$ and edges from all
  vertices of $DAG$ to to $[p,d]$;
  $\textbf{Simulate}(A,DAG)$;
Send $(p,DAG)$ to all processes
The Simulation Algorithm

For each \( I = I_j, \ 0 \leq j \leq n \) do

\[ Y_I := \emptyset; \]

For each path \( g \) in \( \text{DAG}_p \):

\[ R_g := \text{a run of A from } I \text{ with the sequence of failure detector events induced by } g; \]

\[ Y_I := Y_I \cup R_g; \]

Simulation output is a collection of trees \( Y_{ij} \)

---

(2) The Simulation

Forever do:

- \( p \) receives \((q, \text{DAG}_q)\)  // maybe null
- \( d := \) output of \( p \)'s failure detector
- \( \text{DAG} := \text{DAG} \cup \text{DAG}_q; \)
  Add \([p,d] \) to \( \text{DAG} \) and edges from all vertices of \( \text{DAG} \) to \([p,d];\)

\( \{Y_{I_0}, \ldots, Y_{I_n}\} := \text{Simulate}(A, \text{DAG}); \)

Send \((p, \text{DAG})\) to all processes
Properties of the simulation at correct processes

Property 1: For any vertex $S$ of $Y_I$, there exists a finite trace $E$ containing only the steps of correct process such that $S \cdot E$ is in $Y_I$ and all correct process decide in $S \cdot E$

Intuition Behind Property 1
(3) Tagging

Forever do:
- p receives (q,DAG_q)  // maybe null
- d := output of p’s failure detector
- DAG := DAG ∪ DAG_q;
  Add [p,d] to DAG and edges from all vertices of DAG to [p,d];
  \{Y_{i_0},...,Y_{i_n}\} := Simulate(A,DAG);
  TAG(\{Y_{i_0},...,Y_{i_n}\});
  Send (p,DAG) to all processes
The Tagging Algorithm

- For every vector $Y_{ij}$: Tag $I_j$ as
  - **0-valent** if only 0 are decided in $Y_{ij}$
  - **1-valent** if only 1 are decided in $Y_{ij}$
  - **Bivalent** if both 0 and 1 are decided in $Y_{ij}$

(3) Tagging

Forever do:
- $p$ receives $(q, \text{DAG}_q)$  // maybe null
- $d := \text{output of } p$'s failure detector
- $\text{DAG} := \text{DAG} \cup \text{DAG}_q$;
  Add $[p,d]$ to $\text{DAG}$ and edges from all vertices of
  $\text{DAG}$ to to $[p,d]$;
- $\{Y_{i0}, \ldots, Y_{in}\} := \text{Simulate}(A, \text{DAG})$;
- $\text{Tagged\_forest} := \text{TAG}(\{Y_{i0}, \ldots, Y_{in}\})$;
- Send $(p, \text{DAG})$ to all processes
Tagging Properties

• By validity of consensus, $I_0$ is always tagged as 0-valent and $I_n$ as 1-valent
• Other 0 or 1-valent input vector can only get tagged bivalent
• A bivalent input vector stays bivalent forever

Critical Index

• There is some index $k$ in the sequence of vectors such that $I_{k-1}$ is 0-valent and $I_k$ is not: $k$ is called the critical index

• If $I_k$ is 1-valent, then $p_i$ trusts $p_k$

• (we do not consider here the more complicated case when $I_k$ is bivalent)
(4) The Stabilization

- Eventually, the critical index at a given process does not change anymore: this is because the index can only decrease and cannot go lower than 1

- All DAGs converge to the same infinite DAG and the same critical index k is eventually computed at all processes

(5) The Extraction

Forever do:
- p receives (q,DAG_q) // maybe null
- d := output of p’s failure detector
- DAG := DAG ∪ DAG_q;
  Add [p,d] to DAG and edges from all vertices of DAG to to [p,d];
  \{Y_{i0},...,Y_{i\in}\} := Simulate(A,DAG);
  Tagged_forest := TAG(\{Y_{i0},...,Y_{i\in}\});
  p := Extract_Leader(Tagged_forest);
  Output p;
  Send (p,DAG) to all processes
The Extraction Algorithm

If $k$ is critical then
  If $I_{k-1}$ is 0-valent and $I_k$ is 1-valent then
    return $p_k$;
  else // $I_{k-1}$ is 0-valent and $I_k$ is bivalent then
    Look for decision gadgets;
    choose a process based on a deterministically chosen decision gadget;

Correctness of Extraction

Claim: Eventually, (1) all correct processes permanently return the same process $p_k$ and (2) $p_k$ correct

Proof:
  (1) At each correct process, the critical index eventually stabilizes at $k$.
  It is eventually the same at all processes.
  All correct processes return $p_k$
Correctness of Extraction

(2) Assume $p_k$ crashes

\[ I_{k-1}, \text{tag}(I_{k-1})=0 \quad I_k, \text{tag}(I_k)=1 \]

Run without $p_k$ where all processes decide

A contradiction

Decision Gadgets: Fork

\[ I_k, \text{tag}(I_k)=\text{Bivalent} \]

$p$ is correct

Run without $p$ where all processes decide 0
Decision Gadgets: Hook

\[ l_k, \text{tag}(l_k) = \text{Bivalent} \]

\[ S (\text{bivalent}) \]

\[ S_0 (0\text{-valent}) \]

\[ S_1 (1\text{-valent}) \]

Run without \( p \) where all processes decide 0

\( p \) is correct

What people think 😊

- Actual replies I've got when enquiring about an instructional material on CHT
  - My advice is: don't do it!
  - I tried to understand it for a while and gave up
  - It's a terrible proof
  - It’s mind-boggling
  - No way I'm going to try and teach it in a class