

Last update: 17/06/2005, 19:10

**28/2/05**

### **Coordinated attack:**

Requirements:

1. Agreement:  $y_1 = y_2$
2. Tkefut 1: If  $x_1 = x_2 = 0$  then  $y_1 = y_2 = 0$
3. Tkefut 2: If  $x_1 = x_2 = 1$ , and all msgs arrived, then  $y_1 = y_2 = 1$
4. Halting: The sides agree in a finite time.

### **Deterministic**

A deterministic algorithm for the problem does not exist

### **Probabilistic Model**

If each msg arrives w/prob.  $\frac{1}{2}$ , seen algo. which achieves agreement w/prob.  $1 - \frac{1}{2^{r-1}}$ , for some parameter  $r$ :

Send  $r$  msgs, and decide 1 iff you got at least one msg and both  $x$ s are 1

### **Probabilistic Algorithm in Worst Case Model**

An algorithm which achieves agreement w/high prob. in the worst case model:

1. A chooses  $T \in_R [1, r - 1]$ . B chooses  $t \leftarrow 0$
2.  $c_i \leftarrow 0$
3. For  $r$  rounds, send  $\{T, c_i, x_i\}$ . When B gets the first msg, it adopts its  $T$ . When you get a msg with the correct  $T$ , assign  $c_i \leftarrow$  recieved  $c + 1$
4. At the end: decide 1 iff  $c_i \geq T$  and  $x_1 = x_2 = 1$

**7/3/05**

## **Consensus Problems**

Assumptions:

1.  $n$  processors, full network
2. Synchronous system
3. Processor-crash failures
4. No more than  $t$  failures per run

Requirements:

1. Halting: Every intact processor halts and decides
2. Agreement:  $\forall i, j : y_i = y_j$
3. Tkefut: If  $\forall i, x_i = v$  then  $y_i = v$  for all intact processors

### **Lower Bound**

Every consensus algorithm which can handle  $t$  failures requires  $t + 1$  rounds.

## Summary of Algorithms Seen

	Crash Faults	Byzantine Faults
Sync Model, Deterministic Algo's	* When Solvable ?!  * GO (7/3) * Set (7/3) * Needs $t + 1$ rounds	* Solvable iff: $n \geq 3t + 1, c(G) \geq 2t + 1$ (21/3) * EIG Algo. (14/3) * Algo. with small msgs (2/5)
Async Model Deterministic Algo's	Not solvable (4/4)	$\Rightarrow$ Not solvable
Async Model Random Algo's	Use Voting and GCF (11-18/4)	
Async with FD	(18/4)	

### Algorithm GO

(assumption:  $t \leq n - 2$ )

1. At time 0: if  $x_i = 0$ , send GO to everyone, decide 0 and halt.
2. At time  $l = 1, \dots, t$ : If you've heard GO, send GO to everyone, decide 0 and halt.
3. At time  $t + 1$ , if you've heard GO, decide 0 and halt. Otherwise, decide 1 and halt.

Complexity: Time:  $t + 1$ , msgs:  $\Theta(n^2)$

### Set algorithm

Every  $p_i$  sets:  $V_i \leftarrow \{x_i\}$ . For  $t + 1$  rounds, send  $V_i$  to everyone, and assign  $V_i \leftarrow \cup V_j$ . At the end, decide  $y_i \leftarrow \min V_i$

(For binary inputs, the algorithms are equivalent)

### Definitions (from lower bound proof)

- Run  $\alpha$ :  $\alpha = (c_0, \phi_1, c_1, \phi_2, \dots)$ , where:
  - $c_l$  is a configuration:  $c_l^i$ =State in processor  $i$ . In async models, will use also  $c_l^e$ =State of edge  $e$ .
  - $\phi_l$  = Local computation, send msgs, rcv msgs, etc. In an async model, a single event per  $\phi$ . In a sync model  $\phi_l$  describes all msgs sent and rcvd, which processors failed, etc. at time  $l$
- Image of  $p$  from run  $\alpha$ :  $\alpha|_p$
- Runs  $\alpha_1, \alpha_2$  are similar (domot) for  $p$  (denoted:  $\alpha_1 \stackrel{p}{\sim} \alpha_2$ ) if  $\alpha_1|_p = \alpha_2|_p$
- $\alpha$  is a  $t$ -execution of protocol  $\pi$  if during  $\alpha$  at most  $t$  processors failed
- $\text{dec}(\alpha)$ =agreement value decided in run  $\alpha$
- Closure of similarity:  $\alpha_1 \approx \alpha_2$  if  $\exists \beta_1, \beta_2, \dots, \beta_{k+1}, p_{i_1}, p_{i_2}, \dots, p_{i_k}$  s.t.  $\alpha_1 = \beta_1, \alpha_2 = \beta_{k+1}$ , and  $\beta_1 \stackrel{p_{i_1}}{\sim} \beta_2 \stackrel{p_{i_2}}{\sim} \beta_3 \dots \stackrel{p_{i_k}}{\sim} \beta_{k+1}$  where  $p_{i_j}$  is intact in runs  $\beta_j, \beta_{j+1}$
- For  $t$ -runs:  $\alpha_1 \stackrel{[t]}{\approx} \alpha_2$  if exist  $t$ -runs  $\beta_1, \dots, \beta_{k+1} \dots$

14/3/05

### Consensus in a byzantine model

#### Theorem

There is no deterministic algorithm in this model if  $t \geq \lceil \frac{n}{3} \rceil$

#### EIG Algorithm for $n \geq 3t + 1$

Data structure: each processor holds an EIG tree Code for processor  $i$ :

- Round 0:
  - Assign  $\text{val}(\lambda) \leftarrow x_i$

- Send  $\text{val}(\lambda)$  to all processors (including  $i$ )
- If you get a msg from processor  $j$  with  $\text{val}(\lambda) = a$ , mark node  $\sigma = (j)$  in the tree with  $\text{val}(\sigma) \leftarrow a$ . Otherwise, assign  $\text{val}(\sigma) \leftarrow 0$
- Round  $k + 1$ :
  - For each  $k \leq t$ , send to everyone all the pairs  $(\sigma, \text{val}(\sigma))$  for all  $\sigma$  in level  $k$  in your tree, s.t.  $i \notin \sigma$
  - Save msgs you get: for each msg  $(\sigma, a)$  which you get from processor  $j$ , mark  $\text{val}(\sigma') = a$ , where  $\sigma' = (\sigma, j)$ . If the msg is corrupted, mark  $\text{val}(\sigma') = 0$  for all  $\sigma'$  ending with  $j$ .
- At end of round  $t+1$ :
  - Go over your tree bottom-up: For a leaf  $\sigma$ , assign  $\text{res}(\sigma) = \text{val}(\sigma)$ . For an inner node, assign  $\text{res}(\sigma) = 1$  iff there is a real majority (rov mamash) of ones in  $\text{res}(\sigma')$  of  $\sigma$ 's sons
  - $y_i \leftarrow \text{res}(\lambda)$

Complexity: Time:  $t + 1$ , msgs:  $(t + 1)n^2$  msgs of size  $O(n^{t+1})$

## 21/3/05

### Definitions from proof of EIG algo.

- A node  $\sigma$  (in an EIG tree) is Achid-Val if for all intact processors  $i$ ,  $\text{val}_i(\sigma) = a$
- A node  $\sigma$  (in an EIG tree) is Achid-Res if for all intact processors  $i$ ,  $\text{res}_i(\sigma) = a$
- A node  $\sigma$  (in an EIG tree) is Yatziv if  $\exists a$  s.t. for all intact processor  $i$ ,  $\text{val}_i(\sigma) = \text{res}_i(\sigma) = a$  (it is Achid-Val and Achid-Res, and their value is identical)
- A node  $\sigma = \langle j_1, j_2, \dots, j_k \rangle$  is Amin if  $p_{j_k}$  is intact.

### Theorem

definition: Connectivity of a graph:  $c(G)$  = minimal # nodes to remove in order to make the graph disconnected

Note: A graph has connectivity  $c(G)$  iff each two nodes  $u, w$  have  $\geq c(G)$  vertex-disjoint paths between them.

Theorem: The byzantine consensus problem (with a known topology) is solvable on a graph  $G$  with  $n$  nodes and  $t$  failures iff: (1)  $n \geq 3t + 1$ . (2)  $c(G) \geq 2t + 1$

Theorem: In the crash failures model, (2) is replaced by  $c(G) \geq t + 1$   
????????????????

## 4/4/05

### Consensus in an async system

#### Theorem

In an async system, there is no deterministic consensus algorithm, even for the simplest case - one crash.

#### Definitions from the proof

- Event  $\phi$ :
  - $(i, m)$  -  $p_i$  gets message  $m$
  - $p_i$  performs a local computation
  - $p_i$  sends a msg to some of its nbrs
- Run  $\alpha = \langle c_0, \phi_1, c_1, \dots \rangle$
- $c_t = \langle c_t^1, \dots, c_t^n, B_t^1, \dots, B_t^n \rangle$  where  $c_t^i$  is the state of processor  $i$ , and  $B_t^i$  is the buffer of processor  $i$ .
- $(i, m)$  is “Bar-Hafala” in configuration  $c_t$  if  $m \in B_t^i$

- $p_i$  is “Bar-Hafala” in configuration  $c_t$  if  $B_t^i \neq \emptyset$
- for an infinite run  $\sigma$ ,  $p_i$  crashed if:
  - There is a finite number of events  $(i, m)$
  - There are infinitely many configurations  $c_t$  in the run where  $p_i$  is “Bar-Hafala”
- Timing: A series  $\sigma = (i_1, m_1), (i_2, m_2), \dots$
- A timing  $\sigma$  is “Bar-Hafala” in configuration  $c$  if the first event in  $\sigma$  is “Bar-Hafala” in  $c$ , and the rest of  $\sigma$ , denoted  $\sigma'$ , is “Bar-Hafala” in  $c'$ , reached after performing  $(i, m)$  in  $c$
- $\sigma(c) =$  The configuration reached by performing  $\sigma$  on configuration  $c$ .
- $\sigma$  is  $P$ -less if no processor from  $P$  appears in it.
- $\sigma$  is  $P$ -only if only processors from  $P$  appear in it.

**11/4/05**

## Random Algorithm for Consensus in Async model

The algorithm uses three sub-routines:

### Broadcast (“shidur”)

Requirements (BCast of msg  $m$  from sender  $S$ )

1. If the sender  $S$  is intact, then every intact processor will halt and have the message  $m$
2. If some intact processor got  $m$ , then every intact processor will halt and have the message  $m$

Algorithm: Flooding

Complexity: If  $S$  is intact, then time= $O(1)$

## Voting

Requirements:

1. Halting
2. “Close”-voting: All intact processors halt with one of two neighboring values out of:  $(0, H)$ ,  $(0, L)$ ,  $\perp$ ,  $(1, L)$ ,  $(1, H)$
3. Tkefut: If all the inputs are  $a$ , all intact processors decide  $(a, H)$

Algorithm:

1. Broadcast  $x_i$  to everyone
2. Save values which you receive in an array  $A_i$  (of size  $n - t$ )
3. Once you get  $n - t$  values,  $y_i \leftarrow \text{majority}(A_i)$
4. Bcast  $y_i$  to everyone
5. Save values which you receive in an array  $B_i$
6. Once you get  $n - t$  values,  $z_i \leftarrow \text{majority}(B_i)$
7. Bcast  $z_i$  to everyone
8. Save values which you receive in an array  $C_i$
9. Once you get  $n - t$  values, decide:  
If  $B_i$  includes a single value  $a$ , decide  $(a, H)$   
Otherwise, if  $C_i$  includes a single value  $a$ , decide  $(a, L)$   
Otherwise, decide  $\perp$

## Global Coin Flip (GCF)

Requirements:

- $\text{Prob}(\text{All processors decide } 0) \geq \frac{1}{8}$

- $\text{Prob}(\text{All processors decide 1}) \geq \frac{1}{8}$

Algorithm:

1. Each processor chooses  $r_i \leftarrow_R \{0, \dots, n-1\}$
2. BCast  $(i, r_i)$
3. Keep every incoming pair in an array  $R$ .
4. Once you have  $n-t$  msgs, create a snapshot of  $R$ , denoted  $\hat{R}_i$ , and BCast it.
5. For each  $\hat{R}_j$  you get: add to  $R_i$  any new pairs  $(k, r_k)$  you see
6. After getting  $n-t$  vectors  $\hat{R}_j$ , decide:  
If  $R_i$  includes a 0, decide  $c_i \leftarrow 0$ . Otherwise, decide  $c_i \leftarrow 1$

Assumption: Oblivious adversary, i.e. all timings are decided upon in advance, regardless of the coin tosses.

## 18/4/05

### Random Algorithm for Consensus in Async model - cont.

The algorithm (code for processor  $i$ ):

1.  $\alpha_i \leftarrow x_i$
2. Repeat forever:
  - (a)  $\beta_i \leftarrow \text{vote}(\alpha_i)$
  - (b)  $\gamma_i \leftarrow \text{GCF}$

(c) case( $\beta_i$ ):

If  $\beta_i = (a, H)$ , decide  $a$  and halt (besides maybe helping one more voting)

If  $\beta_i = (a, L)$ , then  $\alpha_i \leftarrow a$

If  $\beta_i = \perp$ , then  $\alpha_i \leftarrow \gamma_i$

Complexity: Time expectation:  $O(1)$  (assuming BCast takes  $O(1)$ )

## Failure Detectors

Assumptions: Whenever a processor fails, the FD will detect it in a finite time and notify all the rest. FDs never fail or make mistakes.

### Algorithm for Consensus with FD

Data structures:

$X^i$ : Array of known inputs. Initialized to  $\perp$ 's, except for cell  $i$ .

$F^i$ : Processors which are known to be failed. Initialized:  $\emptyset$

$A^i$ : Processors for which  $F^j = F^i$  and  $X^j = X^i$ . Initialized:  $\emptyset$

Algorithm (code for processor  $i$ ):

1. Bcast ( $X^i, F^i$ )
2. upon getting a msg from FD about a failed  $j$ , s.t.  $j \notin F^i$ :
  - (a) Add  $p_j$  to  $F^i$
  - (b) Bcast ( $X^i, F^i$ )
  - (c)  $A^i \leftarrow \{i\}$
  - (d) If  $|A^i \cup F^i| = n$ , decide  $\min\{a \in X^i | a \neq \perp\}$
3. upon getting a msg ( $X^j, F^j$ ) which contains new info:
  - (a) if  $j \in F^i$  - ignore
  - (b) Update  $X^i, F^i$

- (c) Bcast  $(X^i, F^i)$
  - (d)  $A^i \leftarrow \{i\}$
  - (e) If  $(X^i, F^i) = (X^j, F^j)$ , then  $A^i \leftarrow A^i \cup \{j\}$
  - (f) If  $|A^i \cup F^i| = n$ , decide  $\min\{a \in X^i | a \neq \perp\}$
4. upon getting a msg  $(X^j, F^j)$  s.t.  $(X^j, F^j) = (X^i, F^i)$ :
- (a)  $A^i \leftarrow A^i \cup \{j\}$
  - (b) If  $|A^i \cup F^i| = n$ , decide  $\min\{a \in X^i | a \neq \perp\}$

Time complexity:  $O(t)$

## 2/5/05

### Byzantine Synchronous Consensus - Alg 2

(Have already seen EIG)

**Algorithm with Small Messages**  $O(1)$

Assumption:  $n \geq 4t + 1$

Data structure: Array *pref* of size  $n$ .

Algorithm: (processor  $p_i$ )

1.  $pref[i] \leftarrow x_i$ , and  $\forall j \neq i, pref[j] \leftarrow \perp$
2. For  $t + 1$  phases  $k = 1, \dots, t + 1$ , do:
  - Round 1:
    - (a) Bcast  $pref[i]$
    - (b) Gather msgs from everybody, and save in *pref* (use  $\perp$  for missing or corrupt msgs)
    - (c)  $M \leftarrow \text{majority}(pref)$  (in case of equality,  $m \leftarrow \perp$ )
    - (d)  $c \leftarrow$  number of cells with value  $M$  in *pref*

Round 2:

(a) If  $i = k$  /\*  $i$  is the king of this phase \*/

Then Bcast  $M$

(b) Get value  $\hat{M}$  from the king

(c)  $pref[i] \leftarrow \begin{cases} M & c > \frac{n}{2} + t \\ \hat{M} & \text{o.w.} \end{cases}$

3.  $y_i \leftarrow pref[i]$

## Faulty channels - Omission

### Single Message passing

Requirements:

1. Halting
2.  $y_B = x_A$

Assumptions:

1. Synchronous model
2. No Byzantine failures
3. No crash failures

Omission Model: for every time  $t$ , exists a time  $t' > t$  s.t. in round  $t'$  the channel is OK

Theorem: The problem has no solution

### Message Streams

Requirements:  $A$  has a stream of inputs -  $m_1, m_2, \dots$

1. At any given moment,  $B$ 's output buffer  $y_B$  should hold a prefix of the inputs:  $m_1, \dots, m_k$ .

2.  $\forall k, \exists t_k$  s.t.  $m_k \in y_B$  in time  $t_k$

Algorithm (sender):

1.  $i \leftarrow 1$
2. Repeat forever:
  - (a) While no  $ack_i$  received, send  $\langle m_i, i \rangle$
  - (b) Upon receiving  $ack_i$ ,  $i \leftarrow i + 1$

Algorithm (receiver):

1.  $j \leftarrow 0$
2. Repeat forever:
  - If  $j \geq 1$ , send  $ack_j$
  - Upon receiving  $\langle m_j, j \rangle$ : Save  $m_j$ , and  $j \leftarrow i$

## Message Streams in a Network

Assumption: For each  $A - B$  cut  $c$ , for each time  $t$ , there exists a time  $t' > t$  in which at least one edge in  $c$  is OK.

Algorithm: To each neighbour, send all the messages you have seen so far, according to the single channel algorithm.

## 9/5/05

## Probabilistic Channel Failures

Assumptions:

- Omission channel failures
- Synchronous network

- Independent failures in different channels or times
- Failure probability:  $p$
- Network topology:  $v_1 - v_2 - \dots - v_n$

Algorithm: For  $c = \frac{2}{\log(p)}$ ,  $B = \lceil c \cdot \log n \rceil$ :  $v_1$  transmits  $M$   $B$  times and halts.

Each node  $v_i$ , upon seeing  $M$  for the first time, sends it  $B$  times and halts.

Analysis: Will succeed w/prob.  $\geq 1 - \frac{1}{n}$ .

Time complexity =  $Bn$  ( $=O(n \log n)$  for constant  $p$ ).

Time Expectance =  $\frac{n}{q}$  ( $=O(n)$  for constant  $p$ ).

## Byzantine Channel Failures

**Message sending on a “path”,  $p < \frac{1}{2}$**

Assumptions (can get rid of them): Simultaneous wakeup, distance from source known, adversary cannot create a msg when none was sent.

Algorithm:  $v_{i-1}$  sends  $M$   $B$  times.  $v_i$  adopts the message which was seen most.

**Message sending on a “path”,  $p \geq \frac{1}{2}$**

There is no solution in this case, not even a probabilistic algorithm.

## 16/5/05

## Broadcast in the Whispering/Telephone Model

**The model:**

In each round, every node can send a msg to a single neighbour.

**Definitions:**

- $T(G, V, S)$  = Time for Broadcast in network  $G$  with timing  $S$

- $T(G, V) = \min_S \{T(G, V, S)\}$
- $T(G) = \max_V \{T(G, V)\}$
- $T(n) = \max_{|V(G)=n} \{T(G)\}$

### Lower Bounds

$$\forall G, \forall V, |G| = n, T(G, V) \geq \lceil \log n \rceil$$

$$T(G, V) \geq \text{rad}(G, V)$$

$$T(G, V) \geq \text{diam}(G)$$

Example for achieving the bound: a  $d$ -dimensional hypercube.

## Channel Crash Failures in Whispering Model

### Definitions

$T(G, S, F)$  = Maximal time for Bcast from  $v_0$  given timing  $S$  and failures  $F$

$$T(n, S, t) = \max_{F \subseteq E, |F| \leq t} \{T(K_n, S, F)\} \quad (K_n = \text{full graph of size } n)$$

$$T(n, t) = \min_S \{T(K_n, T, S)\}$$

### Bounds

Lower bound:  $T(n, t) \geq t + \log n$

Upper bound:  $T(n, t) = O(t + \log n)$

### Deterministic Algorithm

Procedure a: Inside a set of nodes, Bcast on a “cube-tree”.

Procedure b: Between sets of nodes, choose a league on the sets, i.e., s.t. each set talks with every other set. When two sets talk, match between nodes in them.

Algorithm: Set  $s = \lfloor \frac{n}{t+2} \rfloor$ , and cut  $V$  into  $t+2$  sets of size  $s$ :  $c_1, c_2, \dots, c_{t+2}$ , and one more set of size  $r$  (assume  $r \leq s$ ).

Step a:  $v_1(c_1)$  sends  $M$  to  $v_1(c_i) \forall i$

Step b: Inside each  $c_i$ ,  $v_1(c_i)$  sends  $M$  using procedure a.

Step c: Procedure B between the sets

### Probabilistic Algorithm

Given a parameter  $B$ , each node which gets the msg, for  $B$  rounds: choose a random neighbour and send  $M$  to it.

Choosing  $B$ : for  $K_n$ ,  $t < \frac{n}{3}$ , use  $B = O(\log n)$

## 30/5/05

### Local Faults Model

A set  $W$  is  $t$ -local if  $\forall v |\Gamma(v) \cap W| \leq t$

$t$ -local Fault Model: The set of faulty nodes is  $t$ -local

### BCast in a Local-Byzantine-Fault Model

Definitions:

- $x(v, s) =$  The number of nodes in  $v$ 's neighborhood which are closer to  $s$  than  $v = |\{w \in \Gamma(v) : \text{dist}(w, s) \leq \text{dist}(v, s)\}|$
- $X(G) = \min_{v \notin \Gamma(s)} \{x(v, s)\}$

Lemma:  $\forall G$ , and  $t < \frac{X(G)}{2}$ , then there exists a Bcast algorithm tolerant to  $t$ -local byzantine faults.

Proof: Approved Msgs Algorithm:

1.  $s$  sends  $M$  to all its nbrs and halts.
2. A nbr of  $s$  which receives  $M$ :
  - Approves it
  - Sends it to its nbrs and halts

3. A different node  $v$ : Upon receiving  $t + 1$  identical copies of  $M$  from different nbrs:

- Approves it
- Sends it to its nbrs and halts

Requierments:

1. Safety: A node  $v$  never approves a wrong msg
2. Acceptance:  $M$  is approved by all intact processors

### **$t$ -local Couple-cut**

$C = C_1 \cup C_2$  is a  $t$ -local couple cut if it is a cut,  $C_1, C_2$  are disjoint and  $C_1, C_2$  are both  $t$ -local

$LPC(G) =$  The smallest  $t$  s.t.  $G$  has a  $t$ -local couple-cut.

Lemma: For a graph  $G$ , and  $t \geq LPC(G)$ , there is no Bcast algorithm tolerant to  $t$ -local faults.

## **Self Stabilization Model**

The whole system is shaken at  $t = 0$ , and we want to show that within a finite time, it stabilizes itself.

Requirements from the set  $\mathcal{L}$  of “legal” configurations:

1. Correctness: Every run  $\alpha$  which starts from a configuration from  $\mathcal{L}$  meets  $P(\alpha)$  ( $P(\alpha)$  being a predicate on runs)
2. Hitkansut: Every run includes a configuration from  $\mathcal{L}$
3. Closure of  $\mathcal{L}$  (from each configuration in  $\mathcal{L}$ , as long as there are no failures, we stay in configurations from  $\mathcal{L}$ )

## Token Finding

Requirements (i.e.,  $P(\alpha)$ ):

1. In each configuration a single processor holds the token
2. Each processor holds the token infinitely many times

Algorithm (uses a known  $k > n$ ) :

Node  $v_i, i \neq 0$ :

Holding the token:  $s_i \neq s_{i-1}$

Releasing the token:  $s_i \leftarrow s_{i-1}$

Node  $v_o$ :

Holding the token:  $s_0 = s_{n-1}$

Releasing the token:  $s_0 \leftarrow (s_{n-1} + 1) \bmod k$

## 6/6/05

## Maximal Coupling

Algorithm:

- Match rule: If  $P(v) = \perp$ , and  $v$  has a nbr  $w$  s.t.  $P(w) = v$  then  $P(v) \leftarrow w$
- Select rule: If  $P(v) = P(w) = \perp$ , and there is no  $r$  s.t.  $P(r) = v$ , then  $P(v) \leftarrow w$
- Unchain rule: If  $P(v) = w, P(w) = u$ , then  $P(v) \leftarrow \perp$