All-to-All Gradecast using Coding with Byzantine Failures

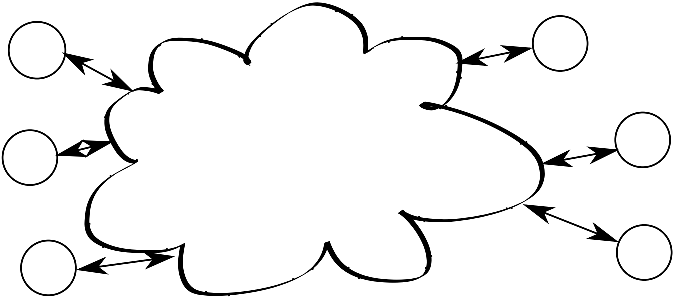
John Bridgman Vijay Garg

Parallel and Distributed Systems Lab (PDSL)
at The University of Texas at Austin
email: johnfbiii@utexas.edu

Presented at SSS 2012

October 4, 2012

Motivating Scenario

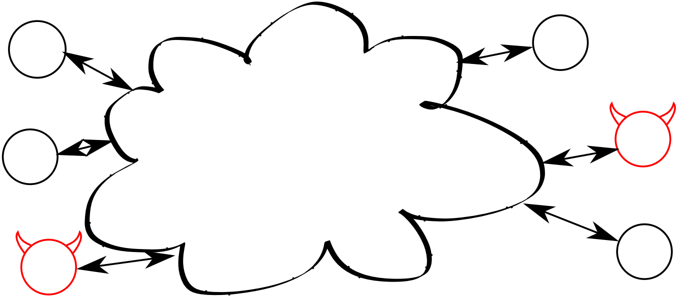


- Want to compute some global function
- For example, the average of the values at each process

- No faults: everyone transmit and compute

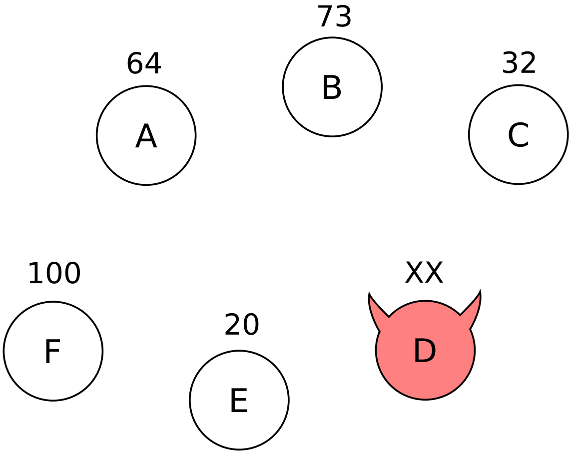
  $\Rightarrow$ No problems

With Faulty Processes

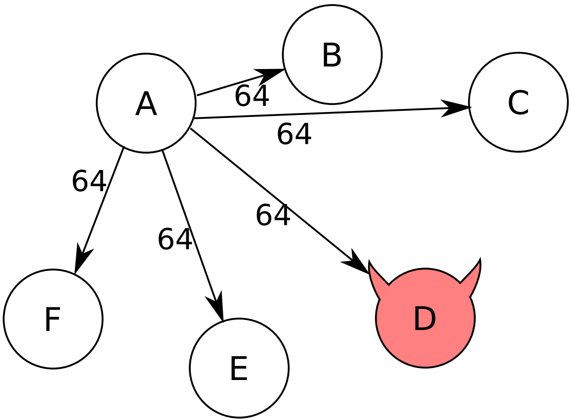


- n processes
- At most t faulty processes

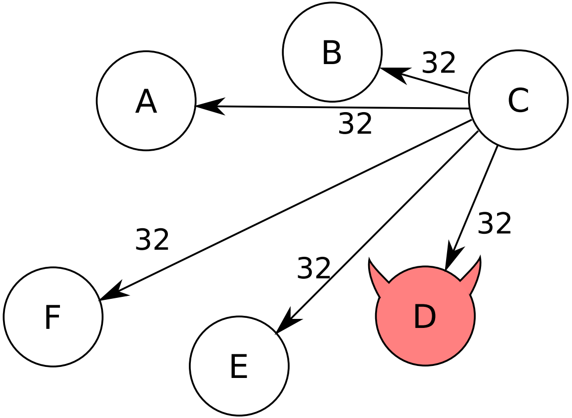
Example



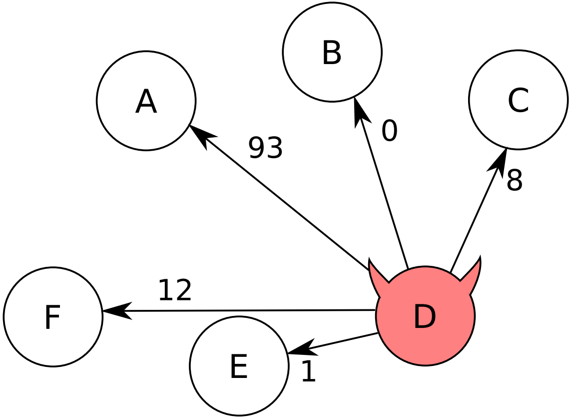
Example



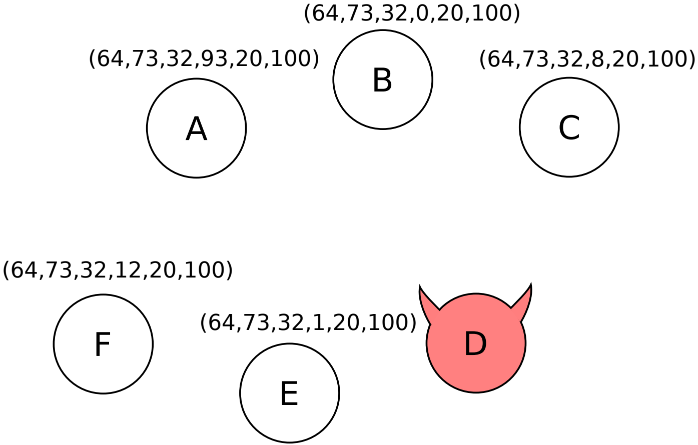
Example



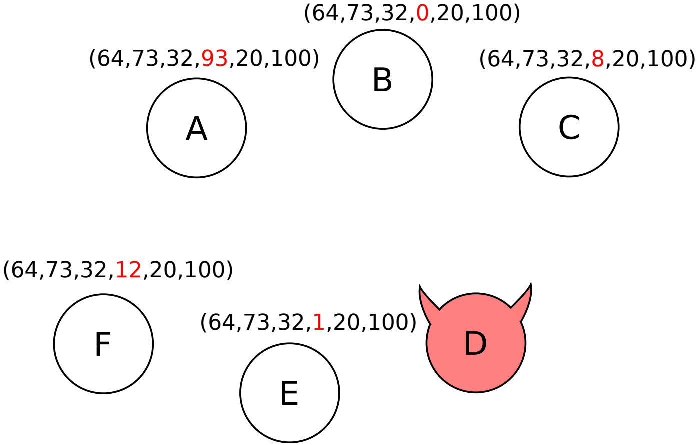
Example



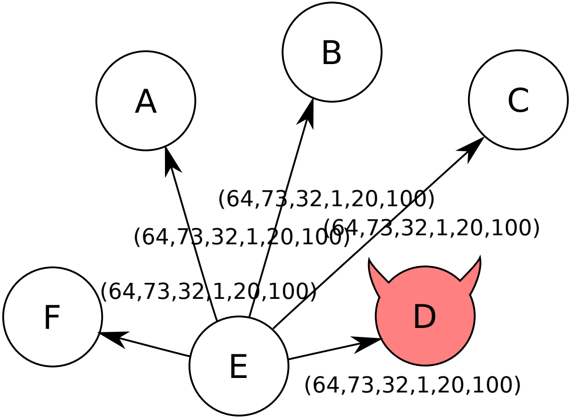
Example


(64,73,32,0,20,100)
B
(64,73,32,93,20,100)
A
(64,73,32,8,20,100)
C
(64,73,32,12,20,100)
F
(64,73,32,1,20,100)
E
D

Example

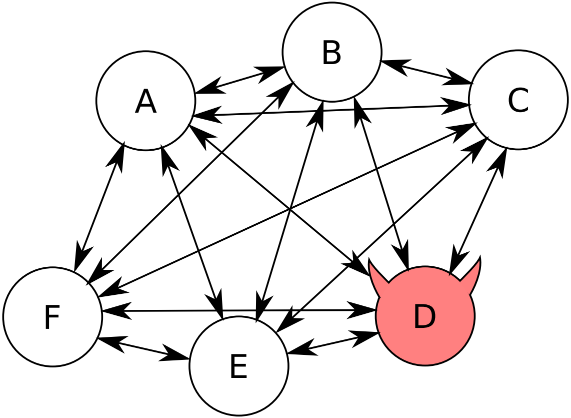

(64,73,32,0,20,100)
B
(64,73,32,93,20,100)
A
(64,73,32,8,20,100)
C
(64,73,32,12,20,100)
F
(64,73,32,1,20,100)
E
D

Example

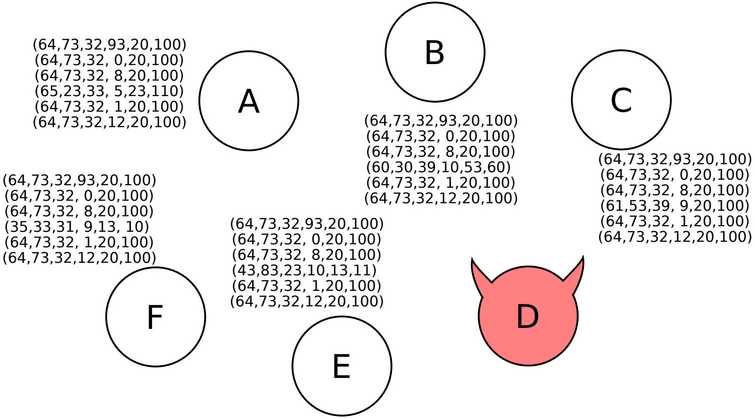


A
B
C
(64,73,32,1,20,100)
(64,73,32,1,20,100)
(64,73,32,1,20,100)
F
(64,73,32,1,20,100)
D
E
(64,73,32,1,20,100)

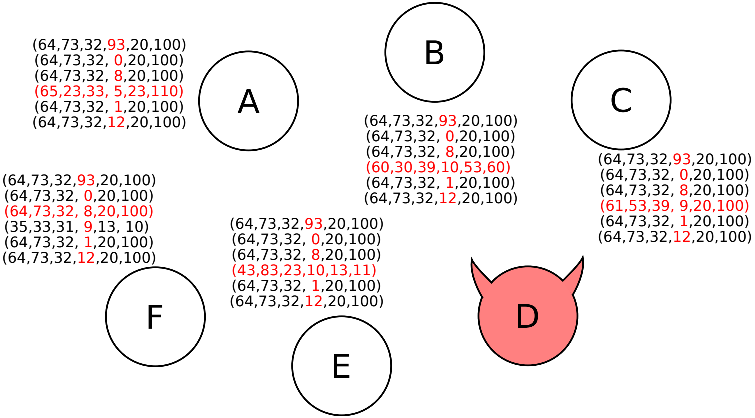
Example



Example


(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(65,23,33, 5,23,110)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
A
B
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(60,30,39,10,53,60)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
C
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(61,53,39, 9,20,100)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(35,33,31, 9,13, 10)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(43,83,23,10,13,11)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
F
E
D

Example



A
B
C
D
E
F
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(65,23,33, 5,23,110)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(60,30,39,10,53,60)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(61,53,39, 9,20,100)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(35,33,31, 9,13, 10)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)
(64,73,32,93,20,100)
(64,73,32, 0,20,100)
(64,73,32, 8,20,100)
(43,83,23,10,13,11)
(64,73,32, 1,20,100)
(64,73,32,12,20,100)

Complexity

- $2n^2$ messages
- $mn^2 + mn^3$ message bits $\equiv O(mn^3)$
- Can we do better when $t << n$?

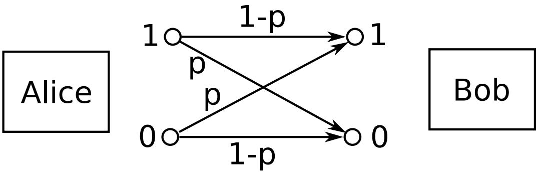
n : number of nodes

m : message size

t : maximum faulty nodes

Error Correcting Codes

- Send message over some channel with probability p to change symbols



- Alice wants to send message M
- Pick a code based on p
- Encode M to get S
- Send S
- Bob decodes S to get M

Systematic Codes

- Called systematic if the first part of S is M
- Encoding gives: $(M) \rightarrow (M, parity)$
- There are many such codes
- Example: Reed-Solomon codes [Reed and Solomon 1960]

- How is this useful here?
- Second broadcast sends redundant information
- At most t of the values different between processes
- Faulty channel indistinguishable from faulty process

Method

Alice

[241, 86, 35, 35]

Bob

[241, 86, 35, 40]

Method

Alice

[241, 86, 35, 35][39, 78]

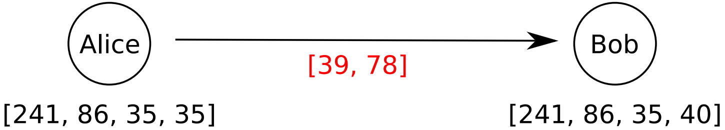
Encode

Bob

[241, 86, 35, 40]

- Reed-Solomon code over Galois field size 256: $(35x^{251} + 35x^{252} + 86x^{253} + 241x^{254})\%(102 + 164x + x^2) = 78 + 39x$

Method



Method

Alice

[241, 86, 35, 35]

Bob

[241, 86, 35, 40]

[241, 86, 35, 40][39, 78]

Method

Alice

[241, 86, 35, 35]

Bob

[241, 86, 35, 40]

[241, 86, 35, 40][39, 78]

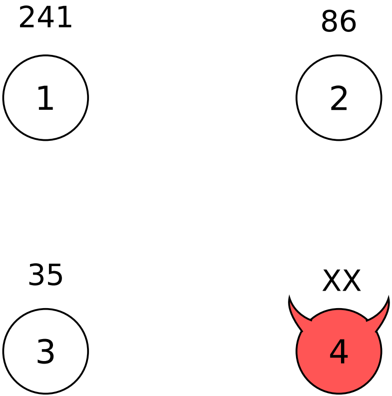
Decode

[241, 86, 35, 35]

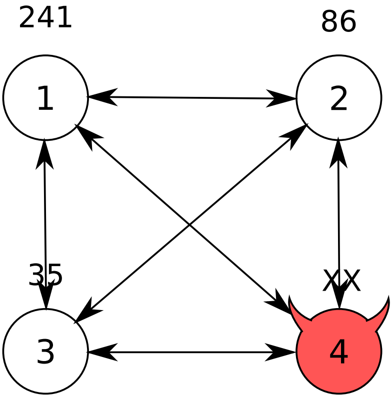
Basic Method

- At most t differences between processes
- Encode to tolerate t corruptions
- Only send parity
- Reduces message size

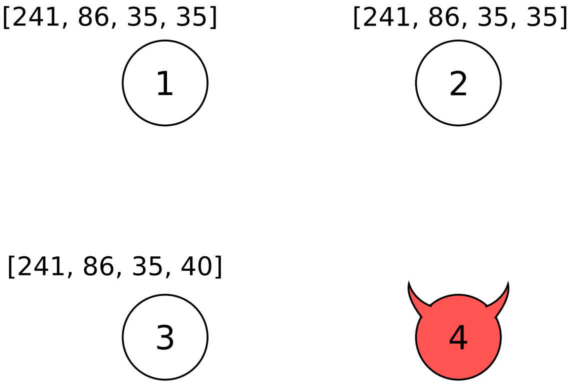
Example



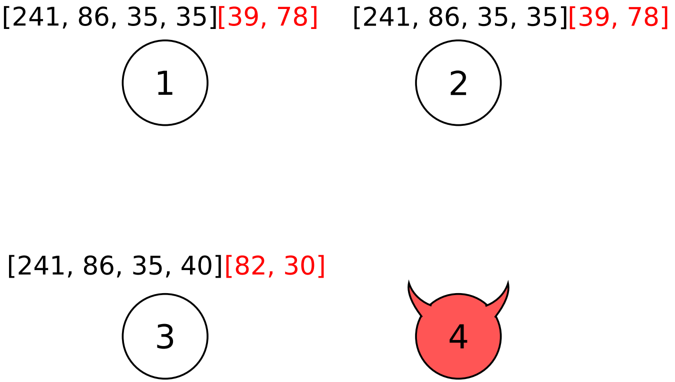
Example



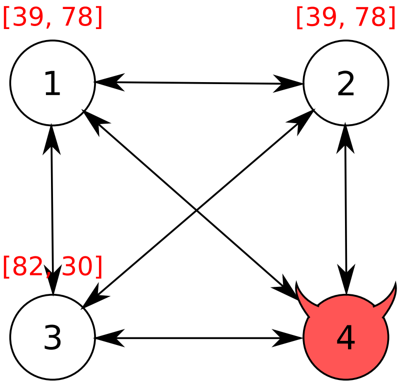
Example



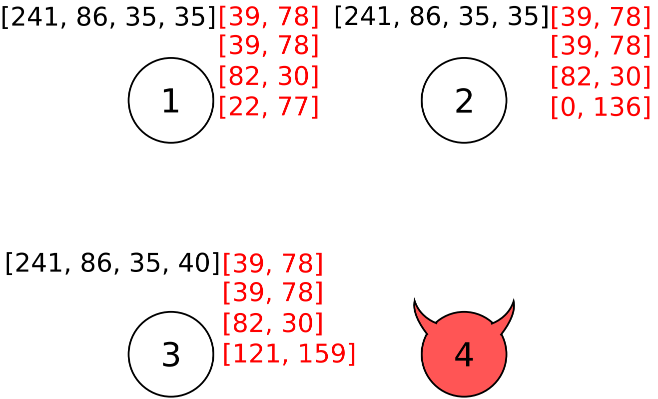
Example



Example



Example


[241, 86, 35, 35][39, 78]
[39, 78]
[82, 30]
[22, 77]
1
[241, 86, 35, 35][39, 78]
[39, 78]
[82, 30]
[0, 136]
2
[241, 86, 35, 40][39, 78]
[39, 78]
[82, 30]
[121, 159]
3
4

Example

[241, 86, 35, 35]
[241, 86, 35, 35]
[241, 86, 35, 40]
[241, 49, 35, 35]

1

2

[241, 86, 35, 35]
[241, 86, 35, 35]
[241, 86, 35, 40]
[241, 86, 129, 35]

[241, 86, 35, 35]
[241, 86, 35, 35]
[241, 86, 35, 40]
[157, 86, 35, 40]

3

4

Method Application: Gradecast

- Gradecast is a broadcast algorithm that can tolerate Byzantine faults [Feldman, Micali 1988]
- Requires $t < n/3$
- When P_i gradecasts a value, every process P_j receives $v_j[i]$ and a confidence level $confidence_j[i] \in \{0, 1, 2\}$

Gradecast Properties

- Confidence is greater than zero implies same value

$\forall P_i, P_j$ non-faulty and $\forall P_k$: $confidence_j[k] > 0$ and $confidence_i[k] > 0 \implies value_j[k] = value_i[k]$

Process	v_i	$confidence_i$	$value_i$
P_1	20	$(2,2,2,1)$	$(20,38,10,5)$
P_2	38	$(2,2,2,1)$	$(20,38,10,5)$
P_3	10	$(2,2,2,0)$	$(20,38,10,\perp)$
P_4	XX	XX	XX

Gradecast Properties

- Confidence differs by at most one

$\forall P_i, P_j$ non-faulty and $\forall P_k$:
$|confidence_i[k] - confidence_j[k]| \leq 1$

Process	v_i	$confidence_i$	$value_i$
P_1	20	$(2, 2, 2, 1)$	$(20, 38, 10, 5)$
P_2	38	$(2, 2, 2, 1)$	$(20, 38, 10, 5)$
P_3	10	$(2, 2, 2, 0)$	$(20, 38, 10, \perp)$
P_4	XX	XX	XX

Gradecast Properties

- Non-faulty source gets maximum confidence

If P_k is non-faulty, $\forall P_i$ non-faulty: $confidence_i[k] = 2$ and $value_i[k] = P_k$'s initial value

Process	v_i	$confidence_i$	$value_i$
P_1	20	$(2, 2, 2, 1)$	$(20, 38, 10, 5)$
P_2	38	$(2, 2, 2, 1)$	$(20, 38, 10, 5)$
P_3	10	$(2, 2, 2, 0)$	$(20, 38, 10, \perp)$
P_4	XX	XX	XX

Algorithm Setup

P_i::

Inputs:

v_i : Input value for P_i

Common knowledge:

n : The number of processes

t : Maximum number of faulty processes

Variables:

$V_i[1..n]$: Vector received in Step 2, initially $\perp$

$Vecc_i[1..2t+1]$: error correction vector for V_i

$X_i[1..n][1..n]$: Matrix of decoded values in Step 3

$Y_i[1..n]$: Vector of values computed in Step 3

$Yecc_i[1..2t+1]$: Error correction vector for Y_i

$Z_i[1..n][1..n]$: Matrix of decoded values in Step 4

$value_i[1..n]$: Vector of output values

$confidence_i[1..n]$: Vector of confidence levels

Algorithm Step One Though Three

// Step 1: Initial Broadcast

for j : 1 to n **do** $P_i.send(P_j, v_i)$; **end**

// Step 2: Receive, Encode, Broadcast encoding

for j : 1 to n **do** $V_i[j] = P_i.receive(P_j)$; **end**

$Vecc_i = encode(V_i)$;

for j : 1 to n **do** $P_i.send(P_j, Vecc_i)$; **end**

// Step 3: Receive encoding and process, encode result

// and broadcast encoding

for j : 1 to n **do** $X_i[j] = decode(V_i, P_i.receive(P_j))$; **end**

$\forall j$ let $Y_i[j] = x$ if $\exists x$ s.t. $|\{k : X_i[k][j] = x\}| \geq n - t$

otherwise $Y_i[j] = \perp$

$Yecc_i = encode(Y_i)$;

for j : 1 to n **do** $P_i.send(P_j, Yecc_i)$; **end**

Algorithm Step Four

// Step 4: Receive encoding, decode and compute final value

for j : 1 to n **do** $Z_i[j] = decode(Y_i, P_i.receive(P_j))$; **end**

for j : 1 to n **do**

  if $\max_x |\{k : Z_i[k][j] = x\}| \geq 2t + 1$ **then**

    $value_i[j] = \arg\max_x |\{k : Z_i[k][j] = x\}|$;

    $confidence_i[j] = 2$;

  elseif $\max_x |\{k : Z_i[k][j] = x\}| > t$ **then**

    $value_i[j] = \arg\max_x |\{k : Z_i[k][j] = x\}|$;

    $confidence_i[j] = 1$;

  else $value_i[j] = \perp$; $confidence_i[j] = 0$;

  end

end

Output $value_i$ and $confidence_i$.

Property (1)

Theorem

All non-faulty processes with positive confidence about process k have identical value[k]. Formally,

$$\forall P_i, P_j \in G, \forall k : confidence_i[k] > 0 \wedge confidence_j[k] > 0$$

implies

$$value_i[k] = value_j[k].$$

Property (2)

Theorem

For any two non-faulty processes, the difference in their confidence levels for any process P_k can differ by at most 1. Formally, $\forall P_i, P_j \in G, \forall k : |confidence_i[k] - confidence_j[k]| \leq 1.$

Property (3)

Theorem

If P_k is non-faulty, then, all non-faulty processes P_i have the value sent by process P_k and their confidence level on this value is 2. Formally,

$\forall P_i, P_k \in G : (confidence_i[k] = 2) \wedge (value_i[k] = v_k)$.

Gradecast Application: Byzantine Agreement

- Simple Byzantine Agreement algorithm by [Ben-Or, Dolev, Hoch 2010] that uses gradecast with $O(mn^3)$ message bit complexity
- Can use our version of gradecast to get $O(mtn^2)$ message bit complexity

n : number of nodes
m : message size
t : maximum faulty nodes

Gradecast Application: Approximate Agreement

- [Ben-Or, Dolev, Hoch 2010] also give an approximate agreement algorithm using gradecast
- Can use our modified algorithm
- $O(mkn^3)$ to $O(mktn^2)$ reduction in message bit complexity

n : number of nodes
m : message size
t : maximum faulty nodes
k : rounds in approximate agreement

Related Work

- [Liang and Vaidya 2011]
  $O(mn)$ bits for broadcast if $m = \Omega(n^6)$ otherwise
  $O(nm + n^4 m^{1/2} + n^6)$
- [Friedman, Mostéfaoui, Rajsbaum and Raynal 2007]
  A mapping from a distributed agreement problem to a coding problem

What about crash faults?

- Processes can only crash
- Translates to erasures
- Needs one half the parity length

Conclusions

Used Forward Error Correcting Codes to reduce message bit complexity of fault tolerant broadcast

Future Work

- Apply to other distributed algorithms like simulated authentication [Srikanth and Toueg 1987] and interactive consistency [Hélary, Hurfin, Mostéfaoui, Raynal, and Tronel 2000]
- Derive lower bounds using coding theory results

Thank you