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
e-mail: 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

⇒ No problems
With Faulty Processes

- $n$ processes
- At most $t$ faulty processes
Example

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 4/28
Example

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 4/28
Example

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 4/28
Example

A
B
C
D
E
F

(64,73,32,0,20,100)
(64,73,32,93,20,100)
(64,73,32,8,20,100)
(64,73,32,12,20,100)
(64,73,32,1,20,100)

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 4/28
Example

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 4/28
Complexity

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

$n$ : 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 \[ \begin{array}{c} 1 \\ p \\ 0 \end{array} \] \quad \text{1-p} \quad \begin{array}{c} 1 \\ p \\ 0 \end{array} \quad \text{Bob}

- 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: $(M) \rightarrow (M, \text{parity})$
- 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

Reed-Solomon code over Galois field size 256:

\[(35 \times 251 + 35 \times 252 + 86 \times 253 + 241 \times 254) \mod (102 + 164 \times x^2) = 78 + 39 \times x\]

JohnFBIII@utexas.edu, garg@ece.utexas.edu
Reed-Solomon code over Galois field size 256:

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

\[(35 \times 251 + 35 \times 252 + 86 \times 253 + 241 \times 254) \mod (102 + 164 \times 2^2) = 78 + 39 \times 2\]

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 9/28
Method

Reed-Solomon code over Galois field size 256:

(35 \times 2^{251} + 35 \times 2^{252} + 86 \times 2^{253} + 241 \times 2^{254}) \mod (102 + 164 \times 2 + 2^{2}) = 78 + 39

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 9/28
Method

Alice
[241, 86, 35, 35]

Bob
[241, 86, 35, 40]

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

Decode
[241, 86, 35, 35]

Reed-Solomon code over Galois field size 256:

(35 x 251 + 35 x 252 + 86 x 253 + 241 x 254)%(102 + 164 x 2 x 256) = 78 + 39 x 3
Basic Method

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

[241, 86, 35, 35]  [241, 86, 35, 35]

1 2

[241, 86, 35, 40]

3 4

johnfbiii@utexas.edu, garg@ece.utexas.edu

All-to-All Gradecast using Coding with Byzantine Failures 11/2
Example

[241, 86, 35, 35] [39, 78] [241, 86, 35, 35] [39, 78]
[39, 78] [82, 30] [39, 78] [82, 30]
[22, 77] [0, 136]

1

2

[241, 86, 35, 40] [39, 78] [241, 86, 35, 40] [39, 78]
[39, 78] [82, 30] [39, 78] [82, 30]
[82, 30] [121, 159]

3

4
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 $confidencel_j[i] \in \{0, 1, 2\}$.
Confidence is greater than zero implies same value

\[ \forall P_i, P_j \text{ non-faulty and } \forall P_k : \text{confidence}_j[k] > 0 \text{ and } \text{confidence}_i[k] > 0 \implies \text{value}_j[k] = \text{value}_i[k] \]

<table>
<thead>
<tr>
<th>Process</th>
<th>( v_i )</th>
<th>( \text{confidence}_i )</th>
<th>( \text{value}_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>20</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>38</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>10</td>
<td>(2, 2, 2, 0)</td>
<td>(20, 38, 10, ⊥)</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
</tr>
</tbody>
</table>
• Confidence differs by at most one

∀P_i, P_j non-faulty and ∀P_k :
|confidence_i[k] − confidence_j[k]| ≤ 1

<table>
<thead>
<tr>
<th>Process</th>
<th>v_i</th>
<th>confidence_i</th>
<th>value_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>20</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>P_2</td>
<td>38</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>P_3</td>
<td>10</td>
<td>(2, 2, 2, 0)</td>
<td>(20, 38, 10, ⊥)</td>
</tr>
<tr>
<td>P_4</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Process</th>
<th>$v_i$</th>
<th>$confidence_i$</th>
<th>$value_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>20</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>$P_2$</td>
<td>38</td>
<td>(2, 2, 2, 1)</td>
<td>(20, 38, 10, 5)</td>
</tr>
<tr>
<td>$P_3$</td>
<td>10</td>
<td>(2, 2, 2, 0)</td>
<td>(20, 38, 10, ⊥)</td>
</tr>
<tr>
<td>$P_4$</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
</tr>
</tbody>
</table>
$P_i::$

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 ⊥
$Vecc_i[1..2t + 1]$ : 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
// Step 1: Initial Broadcast
for j : 1 to n do Pi.send(Pj, vi); end
// Step 2: Receive, Encode, Broadcast encoding
for j : 1 to n do Vi[j] = Pi.receive(Pj); end
Vecc_i = encode(Vi);
for j : 1 to n do Pi.send(Pj, Vecc_i); end
// Step 3: Receive encoding and process, encode result
// and broadcast encoding
for j : 1 to n do Xi[j] = decode(Vi, Pi.receive(Pj)); end
∀j let Y_i[j] = x if ∃x s.t. |{k : Xi[k][j] = x}| ≥ n − t
otherwise Y_i[j] = ⊥
Yecc_i = encode(Y_i);
for j : 1 to n do Pi.send(Pj, Yecc_i); end
// 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,

$$\forall P_i, P_j \in G, \forall k : \text{confidence}_i[k] > 0 \land \text{confidence}_j[k] > 0$$

implies

$$\text{value}_i[k] = \text{value}_j[k].$$
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, $orall P_i, P_j \in G, \forall k : |\text{confidence}_i[k] - \text{confidence}_j[k]| \leq 1$. 
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,

$$\forall P_i, P_k \in G : (\text{confidence}_i[k] = 2) \land (\text{value}_i[k] = v_k).$$
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
[Ben-Or, Dolev, Hoch 2010] also give an approximate agreement algorithm using gradecast. Can use our modified algorithm $O(mkn^3)$ to $O(mkt\cdot n^2)$ 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]
  \[ O(mn) \text{ bits for broadcast if } m = \Omega(n^6) \text{ otherwise } O(nm + n^4 m^{1/2} + n^6) \]

- [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
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