Weighted Byzantine Agreement

Vijay K. Garg    John Bridgman

Parallel and Distributed Systems Lab at The University of Texas at Austin

IPDPS 2011
Byzantine Agreement

- Introduced by Lamport, Shostak and Pease 1980
- Model:
  - $n$ processes
  - $f$ byzantine faults
  - Synchronous system
Byzantine Agreement Requirements

- **Agreement**: Two correct processes cannot decide on different values.
Byzantine Agreement Requirements

- **Agreement**: Two correct processes cannot decide on different values.

<table>
<thead>
<tr>
<th></th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>...</th>
<th>$P_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td><strong>Good Output</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td><strong>Bad Output</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>
Byzantine Agreement Requirements

- **Agreement**: Two correct processes cannot decide on different values.
- **Validity**: The value decided must be proposed by some correct process.
Byzantine Agreement Requirements

- **Agreement**: Two correct processes cannot decide on different values.
- **Validity**: The value decided must be proposed by some correct process.

<table>
<thead>
<tr>
<th></th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>...</th>
<th>$P_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td><strong>Good Output</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td><strong>Bad Output</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>
Byzantine Agreement Requirements

- **Agreement**: Two correct processes cannot decide on different values.
- **Validity**: The value decided must be proposed by some correct process.
- **Termination**: All correct processes decide in finite number of steps.
Byzantine Agreement Lower Bounds

- $n \geq 3f + 1$
- Given by Lamport, Shostak, Pease 1980
Byzantine Agreement Lower Bounds

- $n \geq 3f + 1$
- Given by Lamport, Shostak, Pease 1980
- What if we have 30 processes where 15 of them can fail?
Byzantine Agreement Lower Bounds

- $n \geq 3f + 1$
- Given by Lamport, Shostak, Pease 1980
- What if we have 30 processes where 15 of them can fail?
- $f + 1$ rounds worst case
- Given by Fischer and Lynch 1982
Byzantine Agreement Lower Bounds

- $n \geq 3f + 1$
- Given by Lamport, Shostak, Pease 1980
- What if we have 30 processes where 15 of them can fail?

- $f + 1$ rounds worst case
- Given by Fischer and Lynch 1982

- Can we design a protocol that under certain assumptions can beat these?
Weight Motivation

1. Abstract notion of trust
2. Support multiple classes of processes
3. Beat bounds under certain conditions
WBA Problem Specification

- Common weight vector, $w$
- Weight of failed no more than $\rho$
- Must satisfy:
  - Agreement
  - Validity
  - Termination
WBA Lower Bounds

Let $\alpha_\rho$ be the minimum number of processes whose weight exceeds $\rho$ then

- $\alpha_\rho$ rounds
- $\rho < 1/3$
Outline

Introduction

Algorithms
  Weighted-Queen Algorithm
  Weighted-King Algorithm

Initial Weight Assignment

Updating Weights

Related Work

Conclusions
Weighted Byzantine Algorithm Examples

- Two algorithms: Weighted Queen and Weighted King
- These have good properties
  - $\leq f + 1$ phases
  - Any failure combination so long as weight $< \rho$
The Weighted-Queen Algorithm

- Based on Phase Queen given by Berman and Garay 1989

<table>
<thead>
<tr>
<th></th>
<th>Phase Queen (original)</th>
<th>Weighted Queen (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>$f &lt; \frac{n}{4}$</td>
<td>$\rho &lt; \frac{1}{4}$</td>
</tr>
<tr>
<td>Rounds</td>
<td>$2(f + 1)$</td>
<td>$2\alpha\rho$</td>
</tr>
</tbody>
</table>
The Weighted-Queen Algorithm

- Based on Phase Queen given by Berman and Garay 1989

<table>
<thead>
<tr>
<th></th>
<th>Phase Queen (original)</th>
<th>Weighted Queen (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>$f &lt; n/4$</td>
<td>$\rho &lt; 1/4$</td>
</tr>
<tr>
<td>Rounds</td>
<td>$2(f + 1)$</td>
<td>$2\alpha_{\rho}$</td>
</tr>
</tbody>
</table>

$\alpha_{\rho} \leq f + 1$
The Weighted-Queen Algorithm

For $\alpha \rho$ phases iterating over the processes starting with highest weight to lowest do:

- First round
  - Exchange own value, $v$, with everyone
  - Set $v$ to the value with the highest weight
  - Set $\text{supp}$ to the weight of $v$

- Second round
  - Queen broadcasts its value
  - If $\text{supp} \leq 3/4$, set $v$ to the queen’s value

Output own value
Weighted-Queen Example

- Example: 7 processes with weight assignment
  \[0.2, 0.2, 0.12, 0.12, 0.4, 0.12, 0.12]\n- Standard algorithm: 1 fault only, Weighted: some 2 faults
- For example 0 and 4 together
Weighted-Queen Example

- Phase 1, Round 1:

```
0
w[0]: 0.20
v: 1

1
w[1]: 0.20
v: 1

2
w[2]: 0.12
v: 0
w[3]: 0.12
v: 1

3
w[4]: 0.04
v: 1
w[5]: 0.12
v: 1
w[6]: 0.12
v: 0
```
Weighted-Queen Example

Phase 1, Round 1:

0: 0.12
1: 0.88

0: 0.52
1: 0.48

0: 0.52
1: 0.48

0: 0.12
1: 0.88
Weighted-Queen Example

Phase 1, Round 1:

0 1 2 3
4 5 6

v: 1 supp: 0.88
v: 0 supp: 0.52
v: 0 supp: 0.52
v: 1 supp: 0.88
v: 0 supp: 0.52
Weighted-Queen Example

- Phase 1, Round 2:
Weighted-Queen Example

- Phase 1, Round 2:
Weighted-Queen Example

- Phase 2, Round 1:
Weighted-Queen Example

- Phase 2, Round 1:

```
0 1 2 3
4 5 6
v: 1
supp: 0.88
v: 0
supp: 0.52
v: 0
supp: 0.52
v: 1
supp: 0.88
v: 0
supp: 0.52
```
Weighted-Queen Example

- Phase 2, Round 2:
Weighted-Queen Example

- Phase 2, Round 2:
Persistence of Agreement

Lemma (Persistence of Agreement)

Assuming $\rho < 1/4$, if all correct processes prefer a value $v$ at the beginning of a round; then, they continue to do so at the end of the round.
At Least One Correct Queen

Lemma

There is at least one in the first $\alpha \rho$ rounds in which the queen is correct.
Weighted-Queen Satisfies the WBA Problem

**Theorem**

The Weighted-Queen Algorithm solves the agreement problem for all $\rho < 1/4$.

To prove this we have to prove that this algorithm satisfies the three properties listed previously: validity, termination, and agreement.
Weighted-King Algorithm

- Three round algorithm based on algorithm given by Berman, Garay and Perry 1989

<table>
<thead>
<tr>
<th></th>
<th>Phase King (orig.)</th>
<th>Weighted King (ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>$f &lt; n/3$</td>
<td>$\rho &lt; 1/3$</td>
</tr>
<tr>
<td>Rounds</td>
<td>$3(f + 1)$</td>
<td>$3\alpha\rho$</td>
</tr>
</tbody>
</table>
Weight assignment dramatically changes the nature of these algorithms.

Simple examples:

- \([1/n, 1/n, \ldots, 1/n]\)
- \([1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 0, \ldots, 0]\)
- \([1, 0, 0, \ldots, 0]\)
Initial Weight Assignment

- Weight assignment dramatically changes the nature of these algorithms.
- Simple examples:
  - \([1/n, 1/n, \ldots, 1/n]\)
  - \([1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 0, \ldots, 0]\)
  - \([1, 0, 0, \ldots, 0]\)
- A more involved example with two sets of processes:
  - Set \(A\) is a collection of six highly reliable processes with probability of failure \(f_a = 0.1\).
  - Set \(B\) is a collection of unreliable processes with probability of failure \(f_b = 0.3\).
Initial Weight Assignment Policies

- **Uniform** (Same as regular Byzantine Agreement)
- All weight to set $A$
- $w[i] \propto 1 - Pr\{P_i \text{ fails}\}$
- $w[i] \propto \frac{1}{Pr\{P_i \text{ fails}\}}$
Weight Assignment Example Probabilities

Number of processes in set B

Probability of the weight of failed processes exceeding 1/3

- Only A non-zero
- Uniform (standard BA)
- Proportional to the inverse probability of failure
- Proportional to the probability of not failing

Number of processes in set B
Can we update weights?

Some issues with updating weights:

- Weight vector at each process must be the same
- Each process may see different views of what others have sent
Weight Update Algorithm

- A simple solution of agreeing on weights
- Process can detect a faulty process $j$ if:
  - $j$ sends a no message or corrupted message
  - $j$ is queen, queen value is different from $v$ and $supp > 3/4$
- After detect can reduce the weight of the process
- Have to be careful, faulty process can claim good process faulty
Weight Update Algorithm

- **Round one**
  - Broadcast $faultySet$
  - For each process $j$ that is suspected by some process if the weight of all processes that suspect is greater than $\rho$ then add $j$ to $faultySet$

- **Round two**
  - Use WBA to agree upon $faultySet$
  - Add to $consensusFaulty$ each one agreed to be faulty

- **Round three**
  - Set the weight of processes in $consensusFaulty$ to 0 and renormalise
Related work: Adversarial Structure

- Adversarial structure by Hirt and Maurer 1997
- The adversarial structure is the set of all processes whose failure should be tolerated
- Adversarial structure is exponential in $n$
- Many adversarial structures can be converted to a weight assignment
Weighted Versus Unweighted

- **Pros:**
  - Simple
  - Can tolerate more than $n/3$ faults in certain circumstances
  - Always $\leq f + 1$ rounds

- **Cons:**
  - Even with fewer than $n/3$ faulty processes the algorithm may not work in some cases
Future Work

- Better update methods
- Apply weights to other algorithms
- Approximately equal weight vectors
Conclusion

- Weighted Byzantine Agreement
- Weighted Algorithms
- Initial Weight Assignment
- Update Method
Adversarial Structure Example

Let

\[ P = \{ d, e, f, g, h, i \} \]

and

\[ \bar{A} = \{ \{ d, e, f \}, \{ d, g \}, \{ e, h \}, \{ e, i \}, \{ f, g \} \} \]

Then a weight assignment that satisfies this adversarial structure is:

<table>
<thead>
<tr>
<th>Process</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>1/9</td>
<td>1/18</td>
<td>8/57</td>
<td>1/16</td>
<td>5/19</td>
<td>5/19</td>
</tr>
</tbody>
</table>
Weighted-King Algorithm

For $\alpha_\rho$ rounds where the king iterates over the processes from highest weight to lowest weight:

- **Phase one:**
  - Broadcast own value
  - If 1 or 0 has weight over 2/3 then set own value to that value
  - else set own value to undecided

- **Phase two:**
  - Broadcast own value
  - If the weight of some value received is above 1/3 (giving priority to 0, then 1, then undecided) set own value to that value

- **Phase three:**
  - King broadcasts its value
  - if own value is undecided or the supporting weight from phase two of own value is under 2/3 then set value to kings value
  - if own value is undecided set value to one
Artificial Neural Networks

- Artificial Neural Networks deal with weighted sums of inputs
- Are not used the say way as the way we are using weights.