Practical Byzantine Fault Tolerance

Prof. Phillip Gibbons

Spring 2020, Lecture 15
“The Byzantine Generals Problem”
Leslie Lamport, Robert Shostak, Marshall Pease 1982

- **Leslie Lamport** (SRI International)
  - Turing Award, NAE
  - Blah, blah

- **Robert Shostak** (SRI International)
  - Founder, CTO Vocera Communications

- **Marshall Pease** (SRI International)
  - “Matrix Inversion using Parallel Processing” 1967
  - d. 2001

Based on “Reaching Agreement in the Presence of Faults”
- JACM-1980 paper. Won Dijkstra Prize in 2005
Lamport’s Comments

“I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves...I believed that the problem introduced in [41] was very important and deserved the attention of computer scientists.

The popularity of the dining philosophers problem taught me that the best way to attract attention to a problem is to present it in terms of a story.”

...I wanted to assign the generals a nationality that would not offend any readers. At the time, Albania was a completely closed society, and I felt it unlikely that there would be any Albanians around to object, so the original title of this paper was *The Albanian Generals Problem*. Jack Goldberg was smart enough to realize that there were Albanians in the world outside Albania, and Albania might not always be a black hole, so he suggested that I find another name. The obviously more appropriate Byzantine generals then occurred to me.

The main reason for writing this paper was to assign the new name to the problem. But a new paper needed new results as well..."
Before this paper, it was generally assumed that a three-processor system could tolerate one faulty processor. This paper shows that "Byzantine" faults, in which a faulty processor sends inconsistent information to the other processors, can defeat any traditional three-processor algorithm. In general, 3n+1 processors are needed to tolerate n faults. However, if digital signatures are used, 2n+1 processors are enough. This paper introduced the problem of handling Byzantine faults. I think it also contains the first precise statement of the consensus problem.

...My other contribution to this paper was getting it written. Writing is hard work, and without the threat of perishing, researchers outside academia generally do less publishing than their colleagues at universities. I wrote an initial draft, which displeased Shostak so much that he completely rewrote it to produce the final version.”
Byzantine Generals

- Generals communicate only by messenger

- Some of the generals may be traitors

- **Goals:**
  - All loyal generals decide upon the same plan of action
  - A small number of traitors cannot cause the loyal generals to adopt a bad plan

- **Approach:**
  - Every loyal general obtains the same values $v_1 \ldots v_n$ & uses same method of combining values into plan of action
  - If loyal general $i$ sends $v_i$, then all loyal generals obtain $v_i$
Byzantine Generals Problem

A commanding general must send an order to his $n-1$ lieutenant generals such that

- **IC1**: All loyal lieutenants obey the same order

- **IC2**: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends

(Can solve original problem by having each general act as commanding general in order to send its value.)
3 Generals Can’t Tolerate 1 Traitor

Case A

L1: attack
L2: attack
He said retreat
He said attack

By IC2, L1 must attack

Case B

L1: attack
L2: retreat
He said retreat
He said attack

Looks the same to L1, so L1 will attack

(IC2: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends)
3 Generals Can’t Tolerate 1 Traitor

Symmetric Case for L2:
By IC2, L2 must retreat

Looks the same to L2, so L2 will retreat

Thus, in Case B, L1 will attack & L2 will retreat, violating IC1

(IC1: All loyal lieutenants obey the same order)
Generalization to $m$ Traitors

• No solution with $< 3m + 1$ generals can tolerate $m$ traitors
  – If were to exist, solution could be used to solve 3-general/1-traitor problem (by having each general simulate $m$ generals), a contradiction
Oral Messages

A1: Every message that is sent is delivered correctly

A2: The receiver of a message knows who sent it

A3: The absence of a message can be detected

• A1 & A2 prevent a traitor from interfering with communication between two other generals

• A3 foils a traitor who doesn’t send messages (RETREAT is the default order if commander fails to send an order)

If majority of values $v_i$ equal $v$, then $\text{majority}(v_1, \ldots, v_n) = v$
Else, return RETREAT (alternative: return median)
OM\((m)\) for \(\geq 3m + 1\) Generals

Algorithm \(OM(0)\).

1. The commander sends his value to every lieutenant.
2. Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm \(OM(m)\), \(m > 0\).

1. The commander sends his value to every lieutenant.
2. For each \(i\), let \(v_i\) be the value Lieutenant \(i\) receives from the commander, or else be RETREAT if he receives no value. Lieutenant \(i\) acts as the commander in Algorithm \(OM(m - 1)\) to send the value \(v_i\) to each of the \(n - 2\) other lieutenants.
3. For each \(i\), and each \(j \neq i\), let \(v_j\) be the value Lieutenant \(i\) received from Lieutenant \(j\) in step (2) (using Algorithm \(OM(m - 1)\)), or else RETREAT if he received no such value. Lieutenant \(i\) uses the value \(majority(v_1, \ldots, v_{n-1})\).
OM(1) with Loyal Commander

\[
\text{majority}(v, v, y) = v
\]

\[
\text{majority}(v, v, x) = v
\]
OM(1) with Traitor Commander

\[
\text{majority}(x, y, z)
\]

\[
x \quad z
\]

\[
\text{majority}(x, y, z)
\]

\[
\text{majority}(x, y, z)
\]

\[
\text{majority}(x, y, z)
\]

\[
\text{majority}(x, y, z)
\]
“Practical Byzantine Fault Tolerance”
Miguel Castro, Barbara Liskov 1999

• Miguel Castro (MIT PhD, MSR)
  – Mark Weiser Award 2011
  – 27,000 citations, 200+ papers

• Barbara Liskov (MIT)
  – NAE, NAS, AAAS
  – John von Neumann Medal
  – National Inventors Hall of Fame
  – ACM Turing Award 2008
Advantages over Prior Work

- **Does not rely on synchrony (i.e., known bounds on message delay)**
  - Avoids Denial-of-Service attacks that plague synchrony
  - Assumes asynchronous distributed system with message failures
    (But: For liveness, Adversary can’t delay correct node indefinitely)

- **Much better performance**
  - One message round-trip for read-only operations
  - Two message round-trips for read-write operations
  - Uses (cheap) MACs when no faults
    Uses (expensive) public-key crypto only when faults
  - Only 3% overhead for distributed file system when no faults
Discussion: Summary Question #1

• **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
System Model & Guarantees

• **Asynchronous Distributed System**
  – Messages may be lost, delayed, duplicated, delivered out of order

• **Byzantine Failures**
  – Assume independent node failures [Why?]
  – Can’t (i) break crypto or (ii) delay correct node indefinitely
    (hence, repeatedly retried message eventually gets delivered)

• **Guarantees when $\geq 3f + 1$ nodes for $f$ faults**
  – Safety: As if 1 server processing 1 op at a time [linearizability]
  – Liveness
  – Optimal: Even with signed msgs, need $3f+1$, due to asynchrony
Byzantine Quorums & Certificates

• $3f + 1$ nodes in all

• Any group of $2f + 1$ nodes is a quorum

• Intersection of any two quorums is $\geq f + 1$
  – Implies share at least one non-faulty node

• Certificate of “P is true” = A quorum with all saying “P is true”
  – Any other quorum would agree (since share a non-faulty node)
Practical Byzantine Fault Tolerance

- Sequence of configurations called views
  - Each view has one primary. Rest are backups

- Client waits for $f+1$ replies from different replicas with same answer
Normal-Case Operation: Pre-Prepare

- After Client C sends request $m$ to Primary [for view# $v$], Primary assigns Seq# $n$ & multicasts to all backups:
  \[ \langle \text{sign0}\{\text{Pre-prepare}, v, n, \text{hash}(m)\}, m \rangle \]

- Backup accepts pre-prepare if (i) signature & hash are ok, (ii) view is correct, (iii) not already accepted a Pre-prepare,$v,n,\text{hash}(m')$

- **Purpose**: Backups learn msg & proposed sequence number
• If accepts a pre-prepare, backup $i$ multicasts to all nodes:
  \[ \text{<signi\{Prepare,$v,n,hash(m),i$}\>} \]

• Node \textbf{accepts} any prepare if (i) signature ok \& (ii) view is correct

• Node collects certificate that “$m$ is $n^{th}$ request in view $v$”:
  pre-prepare \& $2f+1$ accepted matching prepares

• \textbf{Purpose}: Nodes agree on a total order for requests in view $v$
Once collect certificate \((m,n,v)\), node \(i\) multicasts to all nodes:
\[
<\text{sign}_{i}\{\text{Commit}, v, n, \text{hash}(m), i\}>
\]

Node accepts any commit if (i) signature ok & (ii) view is correct

Node collects a certificate: 2f+1 accepted matching commits

Purpose: \(\geq f+1\) non-faulty nodes have certificate & will apply \(m\)
Once collect “commit” certificate for \((m,n,v)\) & all requests with Seq# < \(n\) have been applied:

Apply \(m\) to local replica & send reply to the client

**Purpose**: Safely apply \(m\) & ensure client gets a reply
Discussion: Summary Question #2

- Describe the paper's single most glaring deficiency. Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
View Changes

• **Provide liveness when primary fails:**
  – timeouts trigger view changes
  – select new primary (= view number mod 3 \( f + 1 \))

• Replicas send VIEW-CHANGE message along with the requests they prepared since last “stable checkpoint”

• Once new primary collects 2f+1 VIEW-CHANGE messages, it multicasts to all a NEW-VIEW message
  – includes above requests to be redone in the current view

• Once current view is up-to-date, can start receiving new requests
Optimizations

• Client request designates a node to send the result; other nodes send only digest

• Don’t wait for “commit” certificate: Once collect certificate \((m,n,v)\) at end of Prepare & all requests with Seq\# < \(n\) have been applied:
  
  Apply \(m\) to local replica & send tentative-reply to the client (will revert to last checkpoint if view-change invalidates operation)

• Client multicasts read-only request to all nodes. Node waits for all prior requests to be committed, then replies

• Use (cheap) MACs instead of (expensive) RSA in most cases
Performance on Andrew Benchmark

<table>
<thead>
<tr>
<th>phase</th>
<th>BFS</th>
<th></th>
<th></th>
<th>BFS-nr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>strict</td>
<td>r/o lookup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.55 (57%)</td>
<td>0.47 (34%)</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9.24 (82%)</td>
<td>7.91 (56%)</td>
<td>5.08</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7.24 (18%)</td>
<td>6.45 (6%)</td>
<td>6.11</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8.77 (18%)</td>
<td>7.87 (6%)</td>
<td>7.41</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>38.68 (20%)</td>
<td>38.38 (19%)</td>
<td>32.12</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>64.48 (26%)</td>
<td>61.07 (20%)</td>
<td>51.07</td>
<td></td>
</tr>
</tbody>
</table>

For synchronous writes:
- NFS writes to disk
- BFS achieves stability via replication
Approximate Agreement? [BGP]

- **Variant: Agree on approximate time of attack**
  - **IC1’**: All loyal lieutenants attack within 10 minutes of one another
  - **IC2’**: If the commander is loyal, every loyal lieutenant attacks within 10 minutes of the time given in the commander’s order

- **If were to exist, solution could be used to solve BGP**
  - Attack: use attack time 1:00; Retreat: use attack time 2:00
  - Lieutenant: if order time $\leq 1:10$ then attack; $\geq 1:50$ retreat
  - Else if other lieutenant reached a decision, make same decision; Otherwise Retreat.

  - If commander is loyal: IC2’ implies IC2, and IC2 implies IC1
  - If commander is traitor: lieutenants are loyal. By IC1’, decide the same in step (1) or in step (2), implying IC1
Signed Messages [BGP]

A4: Loyal general’s signature cannot be forged & any content alteration of his signed message can be detected. Anyone can verify the authenticity of a general’s signature.

\[
\text{choice}(\text{singleton set } v) = v; \quad \text{choice}(\text{empty set}) = \text{RETREAT}
\]

(e.g. choice returns median)

- \(v : j : i\) denotes the value \(v\) signed by \(j\), and then that value \(v : j\) signed by \(i\)
Algorithm $SM(m)$.

Initially $V_i = \emptyset$.

1. The commander signs and sends his value to every lieutenant.
2. For each $i$:
   
   (A) If Lieutenant $i$ receives a message of the form $v:0$ from the commander and he has not yet received any order, then
      
      (i) he lets $V_i$ equal $\{v\}$;
      (ii) he sends the message $v:0:i$ to every other lieutenant.

   (B) If Lieutenant $i$ receives a message of the form $v:0:j_1: \ldots :j_k$ and $v$ is not in the set $V_i$, then
      
      (i) he adds $v$ to $V_i$;
      (ii) if $k < m$, then he sends the message $v:0:j_1: \ldots :j_k:i$ to every lieutenant other than $j_1, \ldots , j_k$.

3. For each $i$: When Lieutenant $i$ will receive no more messages, he obeys the order $\text{choice}(V_i)$.
SM(1) for 3 Generals [BGP]

L1 will attack

L1 & L2 each obey choice(attack, retreat)
Message Guarantees in Practice [BGP]

A1: Every message that is sent is delivered correctly
• Communication failure is viewed as faulty sender

A2: The receiver of a message knows who sent it
• Need: Faulty processor cannot impersonate nonfaulty one
• Communicate over fixed lines or consider faulty network nodes

A3: The absence of a message can be detected
• Use time-outs based on clocks synchronized within a known maximum error (But: How to sync clocks in Byzantine setting)

A4: Loyal general’s signature cannot be forged & any content alteration of his signed message can be detected.
Anyone can verify the authenticity of a general’s signature
• Standard crypto (simpler if only random malfunction case)
Friday

Project Proposals due 11:59 pm

Monday

Kernels & Virtual Machines (I)

“On μ-Kernel Construction”
Jochen Liedtke 1995

“Safe Kernel Extensions Without Run-Time Checking”
George C. Necula, Peter Lee 1996