Byzantine Generals Problem, CSI 524

Prof. William A. Maniatty
maniatty@cs.albany.edu

http://www.cs.albany.edu/~maniatty/teaching/security/
Introduction to Fault Tolerance

Failures - Cause a machine to give the wrong result for some inputs

- Persistent or Intermittent
- Node Failure vs. Communication Failure
- Security intrusions can be modeled as failures

A formal model of a distributed system

- Modeled as a graph $G = (V, E)$
  - $|V| = N$, i.e. there are $N$ nodes.
  - $|E| \leq \frac{N^2 - N}{2}$, where $E$ is the number of communication channels (links).

A fault tolerant system can continue to operate properly in the presence of a reasonable number of failures.

- **Fail Stop** - Failed nodes/links shut down
- **Byzantine** - Failed links/nodes give incorrect values
- Note: undetected faults cannot be tolerated
Fault Tolerance in Distributed Systems

By definition distributed systems don’t have a centralized controller.

Thus distributed solution methods require reaching consensus (voting).

Distributed systems can be characterized as:

- **Asynchronous** - Makes no assumption about timing, no time outs.
- **Synchronous** - Permits time outs.
Fault Tolerance in Asynchronous Systems

Fisher, et al. proved \[3\] Cannot be guaranteed even under ideal conditions

- Fail stop model.
- Only one failure in \(N\) nodes

Why?

- Remember no timing assumptions allowed in Asynchronous Model
- Hence can’t time out
- During a long wait for a message or is the node/link just really slow?
- However, G. Bracha and S. Toueg \[1\] demonstrated that probabilistic consensus is possible
  \[
  \text{the probability of indefinite delay can be made negligible (have probability 0).}
  \]

Asynchronous systems are of a more theoretical interest.

- Probabilistic consensus is possible
  \[
  \text{the probability of indefinite delay can be made negligible (have probability 0).}
  \]
• Adding failure detectors (so that you know if a node or link is dead) can help.
• Relaxing asynchrony (by allowing atomic operations) helps.
Lamport et al. [4] defined the Byzantine Generals Problem (BGP) as:

- Consider a city under siege by $N$ divisions of the Byzantine Army
- Each division has a General.
  - There is one commanding general.
  - The commander has $N - 1$ lieutenant generals
- Generals communicate by messengers
- Have to agree on a common strategy (or globally fail)
- What if some generals are traitors? Our goals are:
  - All loyal generals should agree on the same strategy
  - A small number of traitors should not be able to trick the loyal generals into using a bad strategy.
BGP Formalized

One possibility the commander is traitor.

This gives rise to Lamport et al’s formalization using Interactive Consistency Conditions

- IC1) All loyal lieutenants obey the same order
- IC2) If the commander is loyal, all loyal lieutenants obey the order he sends.
A question

Consider a case where there is 1 traitor and 3 generals, can we guarantee a correct outcome?

- (HINT) Lieutenants can relay the commander’s order.
An Answer

Given: 1 traitor and 3 generals.

To Prove: A correct outcome is not guaranteed

The idea: Prove One Lieutenant Gets Conflicting Reports And Doesn’t know what to do
In both cases Loyal Lieutenant 1 receives:

- Attack order directly from Commander
- Retreat order directly from Lieutenant 2

Case 1: Lieutenant 2 defects

- IC2) implies Lieutenant 1 should attack
- Suggests a (faulty) rule: Listen only to the commander

Case 2: Commander defects

- If Lieutenant 1 obeys commander he must attack
- If Lieutenant 2 obeys commander he must retreat
- But this violates IC1)

Thus, lieutenants need to listen to each other to detect a traitorous commander
Generalizing the Result

What if we have $N > 3$ generals and $m < N$ traitors?

To distinguish this from the 3 general Byzantine General Problem we call these generals *Albanian Generals*.

In general if $N < 3m + 1$, there is no solution

- Suppose $N = 3m$
- Without loss of generality we can model this by partitioning the Albanians
  - 2 Byzantine Lieutenants, each representing $m$ Albanian Lieutenants
  - 1 Byzantine Commander, representing 1 Albanian commander and $m - 1$ Albanian Lieutenants
- But this representation is exactly the unsolvable Byzantine Generals Problem
Approach to Conflicting Messages

So what should a node do if it gets conflicting messages? Explode in a fiery cataclysm of doom? No...

Each node picks a ‘‘representative’’ message value using a voting method.

- Majority
- Median value
- Mean value (for continuous values)

Picking a voting method depends on application and message type
Approximate Agreement in the BGP 1 of 2

If we have \( N \) generals and \( m \geq \frac{N}{3} \) approximate agreement is impossible.

Consider a scenario with 3 Generals and one traitor where they

- Have synchronized clocks
- All loyal lieutenants must attack within 10 minutes of each other

This gives rise to modified versions of IC1) and IC2)

- IC1)′ All loyal lieutenants must attack within 10 minutes of each other
- IC2)′ If the commander is loyal, all loyal lieutenants must attack within 10 minutes of the time given in his order.
The commander sends a message with a time

- 1:00 means attack at 1:00
- 2:00 means retreat

Lamport Suggests Each Lieutenant does the following:

- Step 1) If the commander’s message is
  - (a) 1:10 or earlier, attack
  - (b) 1:50 or later, retreat
  - (c) Otherwise do step 2

- Step 2) Ask other lieutenant what they decided
  - If the other lieutenant decided, do the same action
  - Otherwise retreat

It can be shown that this approach fails if the commander is a traitor.
Oral messages use a reliable channel where:

- Every sent message is correctly delivered
- The receiver of a message knows who sent it
- The absence of a message can be detected

Lamport et al. developed an Oral Message Algorithm $\text{OM}(m)$, where

- There are $N$ generals with
  - 1 Commander
  - $N - 1$ Lieutenants
  - $m$ of the generals are loyal
- Each pair of generals has a channel for oral messages
- Can’t have too many traitors, requires $N \geq 3m + 1$
- Use a function to obtain representative value $\text{majority}(v_1, v_2, \ldots, v_{N-1})$
  - Can use simple majority, median for ordered sets or average for continuous values
The Oral Message tolerating \( m \) traitors, \( \text{OM}(m) \) algorithm

1. \( \text{OM}(0) \) (\( m = 0 \) case, i.e. there are no traitors)
   
   (a) The commander sends his value to every lieutenant
   (b) Each lieutenant receiving a command uses the value received, if a message does not arrive, uses the value RETREAT

2. \( \text{OM}(m) \) (\( m > 0 \) case, i.e. there are \( m \) traitors)'

   (a) The commander sends his value to every lieutenant
   (b) For each Lieutenant \( i, 1 \leq i \leq N \) let \( v_i \) be the value \( i \) receives from the commander or RETREAT if no such value was received.
      In the next stage, Lieutenant \( i \) will act as a commander of the remaining \( n - 2 \) Lieutenants in \( \text{OM}(m - 1) \) with order \( v_i \).
   (c) For each node \( i \), let \( j \neq i, 1 \leq j \leq N \), be some other Lieutenant. Let \( v_j \) be the value \( j \) sends to \( i \) in Step 2b (using \( \text{OM}(m - 1) \)) or else retreat if he receives no such value.
      Lieutenant \( i \) uses \( \text{majority}(v_1, v_2, \ldots, v_{N-1}) \).
Examples of $\text{OM}(1)$ for $N = 4$

Lieutenant 1 (loyal)  
Lieutenant 2 (loyal)  
Commander (loyal)  
Lieutenant 3 (traitor)

Lieutenant 3 Defects

Lieutenant 1 (loyal)  
Lieutenant 2 (loyal)  
Commander (traitor)  
Lieutenant 3 (loyal)

Commander Defects
Remarks on Correctness of $\text{OM}(m)$

Theorem: Algorithm $\text{OM}(m)$ satisfies IC1 and IC2 if there are no more than $m$ traitors and at least $3m$ generals (i.e. $n > 3m$).

Proof by induction on $m$.

- Base Case: $m = 0$ means there are no traitors, so $\text{OM}(0)$ satisfies IC1 and IC2.
- Induction Step: Show that theorem holds for $\text{OM}(m)$ case if the theorem holds for $\text{OM}(m - 1)$ where $m > 0$.
- Case 1: The Commander is loyal.
  - Lemma: For any $m$ and $k$, $\text{OM}(0)$ satisfies IC2 if there are at least $2k + m$ generals and no more than $k$ traitors.
  - If $k = m$ then $\text{OM}(m)$ satisfies IC2 and since the commander is loyal IC1 holds.
- Case 2: The commander is a traitor.
  - Then there are at most $m - 1$ traitorous lieutenants and 1 traitorous commander.
  - From our hypothesis are $n - 1 > 3m - 1$ lieutenants, and $m - 1$ traitors. $\text{OM}(m - 1)$ on the lieutenants obeys our constraint since $n - 1 > 3m - 1 > 3(m - 1)$.  

Some Cost Measures in Distributed/Parallel Algorithms

Common measures of parallel algorithm resource efficiency are:

- Run Time - when the last processor finishes
- Number of rounds (for algorithms that synchronize on iterations).
- Number of messages transmitted
- Operations performed by a single processor
- Work = Operations per processor $\times$ num processors.
- Memory needed (per node or global memory required).
Remarks on Cost/Complexity of \( OM(m) \)

- Time: The algorithm runs for \( m + 1 \) rounds.
  - Work per round is proportional to the number of messages

- Message Count: \( O(N^{(m+1)}) \).
  - **Round 1**: Commander sends \( N - 1 \) messages
  - **Round 2**: \( N - 1 \) lieutenants act as commanders for \( N - 2 \) of their peers for a total of \( (N - 1)(N - 2) \) messages.
  - **By induction Round \( k \), \( 1 \leq k \leq m + 1 \) requires**
    \[
    \prod_{i=1}^{k} (N - i) = (N - 1)(N - 2) \ldots (N - k) \tag{1}
    \]

- So the total number of messages is:
  \[
  \text{Number of Messages} = \sum_{i=1}^{m+1} \prod_{j=1}^{i} (N - j) = O(N^{(m+1)}) \tag{2}
  \]
Concluding Remarks and Alternatives

Number of rounds is inherently \( m + 1 \) for this class of problem.

Even if the faults happen to be fail stop instead of Byzantine faults, message count is large, since generals must check for altered messages:

- If faults are fail stop, the message count can be reduced to (I think to \( O(mN^2) \) but I’m not sure).
- Lamport et al [4] developed a written message protocol (assumes Byzantine Faults)
  - The messages exchanged have tamper resistant signatures appended
  - Forging signatures is hard (correctly guessing has negligible probability)
  - Readers of messages can use the signature to detect tampering.
  - Increases message size
  - For \( N \) generals tolerates up to \( m < \frac{N}{3} \) traitors.
  - Still takes \( O(m + 1) \) rounds and \( O(N^{m+1}) \) total messages.
  - Can append signatures to message
  - In 3 general case, can now detect 1 traitor.
Dolev and Strong [2] were able to reduce the number of messages to $O(N^2)$ messages by avoiding retransmitting messages that were already sent.
Signed Messages Allow Byzantine Agreement with 
\[ N = 3m \] Generals
Review and Conclusions

Conclusions on Fault Tolerance

- Byzantine Generals Problem is a very strong result
- However, reaching consensus is expensive
- Especially for large systems
- Or systems with expensive data communication
- But some applications need it . . . .
Distributed file systems (DFS) are characterized by managing a large number of seldom shared files in a distributed systems environment.

Oceanstore applies Byzantine Agreement to ensure data integrity.

Oceanstore’s goal is to modernize DFS, goals include:

- Persistant storage
- Incremental Scalability
- Secure Sharing
- Universal Availability
- Long Term Durability
- Cooperatively operate at the internet scale

Employs a two tier design

- Top tier is powerful well connected hosts
  - **Serialize Changes**
• **Archive Results**

  - Lower tier: work stations
    - *Where users actually perform updates*
    - *Provide storage resources to the system*
Oceanstore Architectural Features

The unit of storage is the *data object*

- Data objects management should support following requirements
  - *Universal accessibility of information (access anywhere, any time)*
  - *Find the right balance between sharing and privacy of information*
  - *Have an easily understood and usable consistency model*
  - *Guarantee data integrity.*

- Flexible, can support e-mail or a Unix File system

Design assumptions are

- Infrastructure untrusted except in aggregation
- Infrastructure is constantly changing
  - *Variable congestion*
  - *Changing connectivity*
  - *So system must be self organizing and self-repairing*

Challenge: To design a system that:

- Provides an expressive storage interface to users
- While guaranteeing high durability
• Atop an untrusted and constantly changing infrastructure
Oceanstore Data Model

A data object is similar to a traditional file
Data objects are an ordered sequence of read only versions
Versions are persistent

- Simplifies caching and replication
- Permits *time travel* (rollback and version comparison)
- What about Hippocratic databases?
Oceanstore Version Support Internals

Copy on write used to reduce storage overhead

- Files stored in trees with pointers to leaves which contain data
- Versions made by tracking only updated blocks

GUID - Globally-Unique Identifier

- AGUID - Active GUID - handle for a sequence
- BGUID - Block GUID
- VGUID - Version GUID - BGUID of top of structure
- GUID’s Heirarchically cryptographically hashed for security reasons
Oceanstore Application-Specific Consistency

Updates add a new head to the version stream of one or more data objects

- Updates are atomic
- Each action guarded by a predicate (much like Bayou)
  - Example Actions: Append, replacing data, truncation
  - Example Predicates: checking latest version number
- Supports application defined consistency semantics.
Oceanstore System Architecture

Changes to a single object must be coordinated via shared resources

Changes to different objects are independent (and can be parallelized)

Virtualization through Tapestry

- Virtual resources are mapped to physical resources
- The mapping is dynamic
- Applications access virtual resources
- Virtual resources have a GUID to get the state information needed to access that resource
- Tapestry supports decentralized object location and routing (DOLR)
  - Built on a scalable overlay network (e.g. TCP/IP)
  - Messages sent via Tapestry routed using GUID, not IP address
  - Tapestry is locality aware (picks the nearest instance with high probability)
  - Physical hosts join tapestry by supplying Resource GUIDs to register themselves
- Hosts publish GUIDs of their resources in Tapestry
- Hosts may unpublish or leave the network at any time.
Oceanstore Replication and Consistency

Replication and Consistency

- Each data object’s AGUID mapping to its current state changes over time
- Each object has a primary replica designated to
  - Serialize and Apply updates
  - Enforce Access Control
- A heartbeat digital certificate is created to map AGUID to the VGUID of the most recent version.
  - a tuple $<\text{AGUID, VGUID, Time Stamp, Version Number}>$
- To securely identify the heartbeat a client may include a nonce in its request
  - A nonce is a special one time message (avoids replay and guessing)
  - Response contains client id and nonce and is signed by the parent
- Primary replica is implemented on a small inner ring of servers
  - Inner ring servers run Byzantine Generals Algorithm to agree on updates and sign result
  - Primary replica is a virtual resource, not bound to a particular server.
- Metadata and secondary replicas also need to be maintained
Oceanstore Ensuring Archival Integrity

How can we prevent data loss in the event of the failure of a small number of nodes?

- For disks we might use replication (mirroring)
- Replication is expensive (100 % overhead per replica)
- Alternatively we can use an erasure code (a superset of RAID)
  - Parity Codes
  - Hamming Codes
  - Pond uses Cauchy Reed-Solomon Codes (beyond lecture scope)
- Erasure Codes partitions a block into $m$ equal sized fragments
- The $m$ fragments are encoded into $n$, $n > m$ fragments
- $r = \frac{m}{n} < 1$ is the rate of encoding
  - Storage overhead increases by a factor of $\frac{1}{r}$.
  - The original object can be reconstructed from any $m$ fragments
  - This is much more fault tolerant than replication (if bad fragments are detected and discarded)
  - Pond uses a Cauchy Reed-Solomon Code with $m = 16$, $n = 32$.
- Updates to primary replicas are erasure coded and fragments are distributed across servers.
Oceanstore Data Caching

Assembling blocks from erasure code fragments is expensive

- $m$ distinct blocks needed stored on $m$ machines mean $m$ messages

For frequently read blocks, we will use block caching

- When a host requests a block
  - *If the host does not have the block cached, request fragments and reassemble from $m$ fragments*
  - *If the host has the block cached, use the local copy*

- Caching host publishes ownership to Tapestry
- Subsequent reads satisfied from the cached block.
  - *Amortizes the cost of reassembly over all the readers*

- Pond uses a soft state approach (i.e. old state is discarded)
- What if the most recent copy is needed
  - *Block cache keeps the heartbeat used to construct the block*
  - *Host needing the block can query Tapestry for just the heartbeat*
  - *If heartbeats match use the block cache, otherwise reassemble the block*

- Write-Invalid (Push) based approach uses a *dissemination tree* for each object
When a host initiates an update in Oceanstore

- Update propagates from client to target object’s primary replica
- The update is serialized with other updates and applied
- The following are done concurrently
  - A heartbeat is generated and propagated out to secondary storage and is multicast along with update to the dissemination tree
A new version is erasure encoded and sent to the archival storage servers.
Employs a Byzantine Generals approach (Liskov-Castro Based Approach)

- Recall Byzantine failures can model nodes with compromised security.
- If we have $N = 3f + 1$ servers we can tolerate $f$ faults.
- Employs signed messages (hence $O(N^2)$ messages need to be sent).
- Hence the inner ring is kept small to reduce overhead.
- Liskov-Castro approach has 2 key management approaches
  - Public Key - Permits 3rd party authentication but slow.
  - Symmetric Key (MAC) - Authentication possible only at end points, but several orders of magnitude faster.
Departures from Castro-Liskov Approach include

- Pond uses a mixed cryptographic approach
  - MAC used in the inner ring
  - Public key used elsewhere (supports aggressive replication, since verification can be done without contacting inner ring).
  - Public key signature cost can be amortized over number of copies distributed.

- Traditional Byzantine Generals Problem Solution tolerate $f$ faults during the life of the system
  - For long term usage, that is too restrictive
  - So Liskov-Castro reboot nodes periodically using a Trusted O/S
  - Liskov-Castro Approach assumes hardware support for key management and fixed membership
  - Want to vary inner ring membership AND not alter public keys
  - Use Rabin’s Proactive Threshold Signatures - Secret Sharing
    - Partition a key into $l$ (overlapping) shares, with $k$ shares needed to reconstruct the key
    - Pond sets $l = 3f + 1$ and $k = f$. 
To change the inner ring membership, the key is repartitioned into a different \( l \) shares (having less than \( k \) old shares doesn’t help).

Assuming no more than \( k - 1 \) faults (by Byzantine Generals Assumption), the inner ring nodes will delete the old keys in favor of the new keys.
So Who gets to be in the Inner Ring?

- Pond designates an arbiter called the **responsible party**
  - Responsible party a single node, perhaps susceptible to compromise
  - If compromised, the makeup of the inner circle might change
  - But compromising data security requires breaking $f$ nodes in the inner circle, not the responsible party
  - However (not mentioned in the paper) changing inner circle involves recomputing shares (expensive). Could this expose Pond to risk of denial of service attacks?

- Traditional Byzantine Generals Problem Solution tolerate $f$ faults during the life of the system
  - For long term usage, that is too restrictive
  - So Liskov-Castro reboot nodes periodically using a Trusted O/S
  - Liskov-Castro Approach assumes hardware support for key management and fixed membership
  - Want to vary inner ring membership AND not alter public keys
  - Use Rabin’s Proactive Threshold Signatures - Secret Sharing
    - Partition a key into $l$ (overlapping) shares, with $k$ shares needed to reconstruct the key
Pond sets $l = 3f + 1$ and $k = f$.

To change the inner ring membership, the key is repartitioned into a different $l$ shares (having less than $k$ old shares doesn’t help).

Assuming no more than $k - 1$ faults (by Byzantine Generals Assumption), the inner ring nodes will delete the old keys in favor of the new keys.
References


