Today’s plan

• Fault-tolerant consensus in synchronous systems
• Link failures:
  – The Two Generals problem
• Process failures:
  – Stopping and Byzantine failure models
  – Algorithms for agreement with stopping and Byzantine failures
  – Exponential information gathering
• Reading: Section 5.1, 6.1-6.3
• Next:
  – Lower bounds for Byzantine agreement:
    • Number of processors
    • Number of rounds
  – Reading:
    • Sections 6.4-6.7
    • [Aguilera, Toueg]
    • (Optional) [Keidar-Rajsbaum]
Distributed consensus

- Abstract problem of reaching agreement among processes in a distributed system, all of which start with their own “opinions”.
- Complications: Failures (process, link); timing uncertainties.
- Motivation:
  - Database transactions: Commit or abort
  - Aircraft control:
    - Agree on value of altimeter reading (SIFT)
    - Agree on which plane should go up/down, in resolving encounters (TCAS)
  - Resource allocation: Agree on who gets priority for obtaining a resource, doing the next database update, etc.
  - Replicated state machines: To emulate a virtual machine consistently, agree on next step.
- Fundamental problem
- We’ll revisit it several times:
  - In synchronous, asynchronous, and partially synchronous settings.
  - With link failures, processor failures.
  - Algorithms, impossibility results.
Consensus with link failures

• Informal scenario:
  – Several generals plan a coordinated attack.
  – All should agree to attack:
    • Absolutely must agree.
    • Should attack if possible.
  – Each has an initial opinion about his army’s readiness.
  – Nearby generals can communicate using foot messengers:
    • Unreliable, can get lost or captured
    • Connected, undirected communication graph, known to all generals, known bound on time for successful messenger to deliver message.

• Motivation: Transaction commit
• Can show no algorithm exists!
Formal problem statement

- $G = (V,E)$, undirected graph (bidirected edges)
- Synchronous model, $n$ processes
- Each process has input 1 (attack) or 0 (don’t attack).
- Any subset of the messages can be lost.
- All should eventually set decision output variables to 0 or 1.
  - In practice, would need this by some deadline.
- Correctness conditions:
  - Agreement:
    - No two processes decide differently.
  - Validity:
    - If all start with 0, then 0 is the only allowed decision.
    - If all start with 1 and all messages are successfully delivered, then 1 is the only allowed decision.
Alternatively:

• **Stronger validity condition:**
  - If anyone starts with 0 then 0 is the only allowed decision.
  - If all start with 1 and all messages are successfully delivered, then 1 is the only allowed decision.
  - Typical for transaction commit (1 = commit, 0 = abort).

• **Guidelines:**
  - For designing algorithms, try to use stronger correctness conditions (better algorithm).
  - For impossibility results, use weaker conditions (better impossibility result).
Impossibility for 2 Generals [Gray]

- Other cases similar, LTTR.
- Proof: By contradiction.
  - Suppose we have a solution---a process (states, transitions) for each index 1, 2.
  - Assume WLOG that both processes send messages at every round.
    - Could add dummy messages.
  - Proof based on limitations of local knowledge.
  - Start with $\alpha$, the execution where both start with 1 and all messages are received.
    - By the termination condition, both eventually decide.
    - Say, by the end of $r$ rounds.
    - By the validity condition, both decide on 1.
2-Generals Impossibility

- $\alpha_1$: Same as $\alpha$, but lose all messages after round $r$.
  - Doesn’t matter, since they’ve already decided by round $r$.
  - So, both decide 1 in $\alpha_1$.
- $\alpha_2$: Same as $\alpha_1$, but lose the last message from process 1 to process 2.
  - Claim $\alpha_1$ is indistinguishable from $\alpha_2$ by process 1, $\alpha_1 \sim^1 \alpha_2$.
  - Formally, 1 sees the same sequence of states, incoming and outgoing messages.
  - So process 1 also decides 1 in $\alpha_2$.
  - By termination, process 2 decides in $\alpha_2$.
  - By agreement, process 2 decides 1 in $\alpha_2$. 
A fine point:

• In $\alpha_2$, process 2 must decide 1 at some point, not necessarily by round $r$. 
Continuing…

- $\alpha_3$: Same as $\alpha_2$, but lose the last message from process 2 to process 1.
  - Then $\alpha_2 \sim^2 \alpha_3$.
  - So process 2 decides 1 in $\alpha_3$.
  - By termination, process 1 decides in $\alpha_3$.
  - By agreement, process 1 decides 1 in $\alpha_3$.

- $\alpha_4$: Same as $\alpha_3$, but lose the last message from process 1 to process 2.
  - Then $\alpha_3 \sim^1 \alpha_4$.
  - So process 1 decides 1 in $\alpha_4$.
  - So process 2 decides 1 in $\alpha_4$.

- Keep removing edges, get to:
The contradiction

- $\alpha_{2r+1}$: Both start with 1, no messages received.
  - Still both must eventually decide 1.
- $\alpha_{2r+2}$: process 1 starts with 1, process 2 starts with 0, no messages received.
  - Then $\alpha_{2r+1} \sim^1 \alpha_{2r+2}$.
  - So process 1 decides 1 in $\alpha_{2r+2}$.
  - So process 2 decides 1 in $\alpha_{2r+2}$.
- $\alpha_{2r+3}$: Both start with 0, no messages received.
  - Then $\alpha_{2r+2} \sim^2 \alpha_{2r+3}$.
  - So process 2 decides 1 in $\alpha_{2r+3}$.
  - So process 1 decides 1 in $\alpha_{2r+3}$.

- But $\alpha_{2r+3}$ contradicts weak validity!
Consensus with process failures

• Stopping failures (crashes) and Byzantine failures (arbitrary processor malfunction, possibly malicious)

• Agreement problem:
  – n-node connected, undirected graph, known to all processes.
  – Input v from a set V, in some state variable.
  – Output v from V, by setting decision := v.
  – Bounded number ≤ f of processors may fail.

• Bounded number of failures:
  – A typical way of describing limited amounts of failure.
  – Alternatives: Bounded rate of failure; probabilistic bounds on failure.
Stopping agreement

• Assume process may stop at any point:
  – Between rounds.
  – While sending messages at a round; any subset of intended messages may be delivered.
  – After sending, before changing state.

• Correctness conditions:
  – Agreement: No two processes (failing or not) decide on different values.
    • “Uniform agreement”
  – Validity: If all processes start with the same v, then v is the only allowable decision.
  – Termination: All nonfaulty processes eventually decide.

• Alternatively:
  – Stronger validity condition: Every decision value must be some process’ initial value.
  – Use this later, for k-agreement.
Byzantine agreement

• “Byzantine Generals Problem” [Lamport, Pease, Shostak]
  – Originally “Albanian Generals”

• Faulty processes may exhibit “arbitrary behavior”:
  – Can start in arbitrary states, send arbitrary messages, perform arbitrary transitions.
  – But can’t affect anyone else’s state or outgoing messages.
  – Often called “malicious” (but they aren’t necessarily).

• Correctness conditions:
  – Agreement: No two nonfaulty processes decide on different values.
  – Validity: If all nonfaulty processes start with the same v, then v is the only allowable decision for nonfaulty processes.
  – Termination: All nonfaulty processes eventually decide.
Technicality about stopping vs. Byzantine agreement

- A Byzantine agreement algorithm doesn’t necessarily solve stopping agreement:
- For stopping, all processes that decide, even ones that later fail, must agree (uniformity condition).
- Too strong for Byzantine setting.
- Implication holds in some special cases, e.g., when all decisions must happen at the end.
Complexity measures

- **Time**: Number of rounds until all nonfaulty processes decide.
- **Communication**: Number of messages, or number of bits.
  - For Byzantine case, just count those sent by nonfaulty processes.
Simple algorithm for stopping agreement

• Assume complete n-node graph.
• Idea:
  – Processes keep sending all V values they’ve ever seen.
  – Use simple decision rule at the end.
• In more detail:
  – Process i maintains $W \subseteq V$, initially containing just i’s initial value.
  – Repeatedly: Broadcast $W$, add received elements to $W$.
  – After $k$ rounds:
    • If $|W| = 1$ then decide on the unique value.
    • Else decide on a default value $v_0 \in V$.

• Q: How large should $k$ be?
How many rounds?

- Depends on number \( f \) of failures to be tolerated.
- \( f = 0 \):
  - \( k = 1 \) is enough.
  - All get same \( W \).
- \( f = 1 \):
  - \( k = 1 \) doesn’t work:
    - Say process 1 has initial value \( u \), others have initial value \( v \).
    - Process 1 fails during round 1, sends to some and not others.
    - So some have \( W = \{v\} \), others \( \{u,v\} \), may decide differently.
  - \( k = 2 \) does work:
    - If someone fails in round 1, then no one fails in round 2.
- General \( f \):
  - \( k = f + 1 \)
Correctness proof (for \( k = f+1 \))

- **Claim 1**: Suppose \( 1 \leq r \leq f+1 \) and no process fails during round \( r \). Let \( i \) and \( j \) be two processes that haven’t failed by the end of round \( r \). Then \( W_i = W_j \) right after round \( r \).
- **Proof**: Each gets exactly the union of all the \( W \)'s of the processes that have not failed by the beginning of round \( r \).
- “Clean round”---allows everyone to resolve their differences.

- **Claim 2**: Suppose all the \( W \) sets are identical just after round \( r \), for all processes that are still non-failed. Then the same is true for any \( r' > r \).
- **Proof**: Obvious.
Check correctness conditions

- **Agreement:**
  - ∃ round r, $1 \leq r \leq f+1$, at which no process fails (since $\leq f$ failures)---a clean round.
  - Claim 1 says all that haven’t yet failed have same $W$ after round $r$.
  - Claim 2 implies that all have same $W$ after round $f + 1$.
  - So nonfaulty processes pick the same value.

- **Validity:**
  - If everyone starts with $v$, then $v$ is the only value that anyone ever gets, so $|W| = 1$ and $v$ is chosen.

- **Termination:**
  - Obvious from decision rule.
Complexity bounds

- **Time**: $f+1$ rounds
- **Communication**:
  - Messages: $\leq (f + 1) n^2$
  - Message bits: Multiply by $n \cdot b$
- Can improve communication:
  - Messages: $\leq 2 \cdot n^2$
  - Message bits: Multiply by $b$

A fixed bound on number of bits to represent a value in $V$. 
Number of values sent in a message
Improved algorithm (Opt)

• Each process broadcasts its own value in round 1.
• May broadcast at one other round, just after it first learns about some value different from its own.
• In that case, it chooses just one such value to rebroadcast.
• After $f + 1$ rounds, use same rule as before:
  – If $|W| = 1$ then decide on the unique value.
  – Else decide on default value $v_0$. 
Correctness

• Relate behavior of Opt to that of the original algorithm.
• Specifically, relate executions of both algorithms with the same inputs and same failure pattern.
• Let OW denote the W set in the optimized algorithm.
• Relation between states of the two algorithms:
  – For every i:
    • OW\(_i\) \subseteq W\(_i\).
    • If |W\(_i\)| = 1 then OW\(_i\) = W\(_i\).
    • If |W\(_i\)| > 1 then |OW\(_i\)| > 1.

  Not necessarily the same set, but both > 1.

• Relation after f+1 rounds implies same decisions.
Proof of correspondence

- Induction on number of rounds (p. 107)
- Key ideas:
  - $O W_i \subseteq W_i$
    - Obvious, since Opt just suppresses sending of some messages from Unopt.
  - If $|W_i| = 1$ then $O W_i = W_i$.
    - Nothing suppressed in this case.
    - Actually, follows from the first property and the fact that $O W_i$ is always nonempty.
  - If $|W_i| > 1$ then $|O W_i| > 1$.
    - Inductive step, for some round $r$:
      - If in Unopt, $i$ receives messages only from processes with $|W| = 1$, then in Opt, it receives the same sets. So after round $r$, $O W_i = W_i$.
      - Otherwise, in Unopt, $i$ receives a message from some process $j$ with $|W_j| > 1$, and so (by induction), $|O W_j| > 1$. Then after round $r$, $|W_i| > 1$ and $|O W_i| > 1$. 
Exponential Information Gathering (EIG)

- A strategy for consensus algorithms, which works for Byzantine agreement as well as stopping agreement.
- Based on EIG tree data structure.
- EIG tree $T_{n,f}$, for $n$ processes, $f$ failures:
  - $f+2$ levels
  - Paths from root to leaf correspond to strings of $f+1$ distinct process names.
- Example: $T_{4,2}$
EIG Stopping agreement algorithm

- Each process $i$ uses the same EIG tree, $T_{n,f}$.
- Decorates nodes of the tree with values in $V$, level by level.
- **Initially:** Decorate root with $i$’s input value.
- **Round $r \geq 1$:**
  - Send all level $r-1$ decorations for nodes whose labels don’t include $i$, to everyone.
    - Including yourself---simulate locally.
  - Use received messages to decorate level $r$ nodes---to determine label, append sender’s id at the end.
  - If no message received, use $\bot$.
- The decoration for node $(i_1, i_2, i_3, \ldots, i_k)$ in $i$’s tree is the value $v$ such that ($i_k$ told $i$) that ($i_{k-1}$ told $i_k$) that $\ldots$ that ($i_1$ told $i_2$) that $i_1$’s initial value was $v$.
- Decision rule for stopping case:
  - Trivial
  - Let $W =$ set of all values decorating the local EIG tree.
  - If $|W| = 1$ decide that value, else default $v_0$. 
Example

- 3 processes, 1 failure
- Use $T_{3,1}$:

Initial values:

Process 1

Process 2

Process 3
Example

• Process 2 is faulty, fails after sending to process 1 at round 1.
• After round 1:
Example

• After round 2:

p3 discovers that p2’s value is 0 after round 2, by hearing it from p1.
Correctness and complexity

• Correctness similar to previous algorithms.
• Time: \( f+1 \) rounds, as before.
• Messages: \( \leq (f + 1) n^2 \)
• Bits: Exponential in number of failures, \( O(n^{f+1} b) \)
• Can improve as before by only relaying the first two messages with distinct values.
• Extension:
  – The simple EIG stopping algorithm, and its optimized variant, can be used to tolerate worse types of failures.
  – Not full Byzantine model---that will require more work…
  – Rather, a restricted version of the Byzantine model, in which processes can authenticate messages.
  – Removes ability of process to relay false information about what other processes said.
Byzantine agreement algorithm

• Recall correctness conditions:
  – Agreement: No two nonfaulty processes decide on different values.
  – Validity: If all nonfaulty processes start with the same v, then v is the only allowable decision for nonfaulty processes.
  – Termination: All nonfaulty processes eventually decide.

• Present EIG algorithm for Byzantine agreement, using:
  – Exponential communication (in f)
  – f+1 rounds
  – n > 3f

• Expensive!
  – Time bound: Inherent. (Lower bound)
  – Number-of-processors bound: Inherent. (Lower bound)
  – Communication: Can be improved to polynomial.
Bad example: $n = 3, f = 1$

- Consider three executions of an EIG algorithm, with any decision rule.
- $\alpha_1$: p1 and p2 nonfaulty, initial value 1, p3 faulty, initial value 0
  - Round 1: All truthful
  - Round 2: p3 lies, telling p1 that “p2 said 0”; all other communications are truthful.
  - Validity requires that p1 and p2 decide 1.
- $\alpha_2$: p2 and p3 nonfaulty, initial value 0, p1 faulty, initial value 1
  - Round 1: All truthful
  - Round 2: p1 lies, telling p3 that “p2 said 1”; all other communications are truthful.
  - Validity requires that p2 and p3 decide 0.
- $\alpha_3$: p1 nonfaulty, initial value 1, p3 nonfaulty, initial value 0, p2 faulty, initial value doesn’t matter.
  - Round 1: p2 tells p1 its initial value is 1, tells p3 its initial value is 0 (inconsistent).
  - Round 2: All truthful.
- $\alpha_3 \sim^1 \alpha_1$, so p1 behaves the same in both, decides 1 in $\alpha_3$.
- $\alpha_3 \sim^3 \alpha_2$, so p3 behaves the same in both, decides 0 in $\alpha_3$.
- Contradicts agreement!
Bad example

- $\alpha_1$: p1 and p2 nonfaulty, initial value 1, p3 faulty, initial value 0
  - Round 1: All truthful
  - Round 2: p3 lies, telling p1 that “p2 said 0”; all other communications are truthful.
  - Validity requires that p1 and p2 decide 1.
Bad example

- $\alpha_2$: p2 and p3 nonfaulty, initial value 0, p1 faulty, initial value 1
  - Round 1: All truthful
  - Round 2: p1 lies, telling p3 that “p2 said 1”; all other communications are truthful.
  - Validity requires that p2 and p3 decide 0.
Bad example

- $\alpha_3$: p1 nonfaulty, initial value 1, p3 nonfaulty, initial value 0, p2 faulty, initial value doesn’t matter.
  - Round 1: p2 tells p1 its initial value is 1, tells p3 its initial value is 0 (inconsistent).
  - Round 2: All truthful.
Notes on the example

• The correct processes can tell something is wrong, but that doesn’t help:
  – E.g., in $\alpha_1$, p1 sees that p2 sends 1, but p3 said that p2 said 0.
  – So p1 knows that either p2 or p3 is faulty, but doesn’t know which.
  – By termination, p1 has to decide something, but neither value works right in all cases.

• Impossibility of solving Byzantine agreement with 3 processes, 1 failure:
  – This is not a proof--- maybe there’s a non-EIG algorithm, or one that takes more rounds,…
  – Come back to this later.
EIG algorithm for Byzantine agreement

- Assume $n > 3f$.
- Same EIG tree as before.
- Relay messages for $f+1$ rounds, as before.
- Decorate the tree with values from $V$, replacing any garbage messages with default value $v_0$.
- New decision rule:
  - Call the decorations $\text{val}(x)$, where $x$ is a node label.
  - Redecorate the tree, defining $\text{newval}(x)$.
    - Proceed bottom-up.
    - Leaf: $\text{newval}(x) = \text{val}(x)$
    - Non-leaf: $\text{newval}(x) =$
      - $\text{newval}$ of strict majority of children in the tree, if majority exists,
      - $v_0$ otherwise.
  - Final decision: $\text{newval}(\lambda)$ (newval at root)
Example: $n = 4$, $f = 1$

- $T_{4,1}$:
- Consider a possible execution in which $p_3$ is faulty.
- Initial values $1 1 0 0$
- Round 1
- Round 2
Example: $n = 4, f = 1$

- Now calculate newvals, bottom-up, choosing majority values, $v_0 = 0$ if no majority.
Correctness proof

- **Lemma 1**: If \(i, j, k\) are nonfaulty, then \(\text{val}(x)_i = \text{val}(x)_j\) for every node label \(x\) ending with \(k\).
- In example, such **nodes are**:

\[
\begin{array}{c}
\lambda \\
1 \\
12 13 14 \\
2 \\
21 23 24 \\
3 \\
31 32 34 \\
4 \\
41 42 43
\end{array}
\]

- **Proof**: \(k\) sends same message to \(i\) and \(j\) and they decorate accordingly.
Proof, cont’d

- **Lemma 2:** If $x$ ends with nonfaulty process index then $\exists v \in V$ such that $val(x)_i = newval(x)_i = v$ for every nonfaulty $i$.

- **Proof:** Induction on lengths of labels, bottom up.
  
  - **Basis:** Leaf.
    
    - Lemma 1 implies that all nonfaulty processes have same $val(x)$.
    - $newval = val$ for each leaf.
  
  - **Inductive step:** $|x| = r \leq f$ ($|x| = f+1$ at leaves)
    
    - Lemma 1 implies that all nonfaulty processes have same $val(x)$, say $v$.
    - We need $newval(x) = v$ everywhere also.
    - Every nonfaulty process $j$ broadcasts same $v$ for $x$ at round $r+1$, so $val(x_j)_i = v$ for every nonfaulty $j$ and $i$.
    - By inductive hypothesis, also $newval(x_j)_i = v$ for every nonfaulty $j$ and $i$.
    - A majority of labels of $x$’s children end with nonfaulty process indices:
      
      - Number of children of node $x$ is $\geq n - f > 3f - f = 2f$.
      - At most $f$ are faulty.
    
    - So, majority rule applied by $i$ leads to $newval(x)_i = v$, for all nonfaulty $i$. 
Main correctness conditions

• **Validity:**
  – If all nonfaulty processes begin with \( v \), then all nonfaulty processes broadcast \( v \) at round 1, so \( \text{val}(j)_i = v \) for all nonfaulty \( i, j \).
  – By Lemma 2, also \( \text{newval}(j)_i = v \) for all nonfaulty \( i,j \).
  – Majority rule implies \( \text{newval}(\lambda)_i = v \) for all nonfaulty \( i \).
  – So all nonfaulty \( i \) decide \( v \).

• **Termination:**
  – Obvious.

• **Agreement:**
  – Requires a bit more work:
Agreement

- **Path covering:** Subset of nodes containing at least one node on each path from root to leaf:

- **Common node:** One for which all nonfaulty processes have the same newval.
  - If a node’s label ends in nonfaulty process index, Lemma 2 implies it’s common.
  - Others might be common too.
Agreement

- **Lemma 3:** There exists a path covering all of whose nodes are common.

- **Proof:**
  - Let \( C = \) nodes with labels of the form \( x_i, \) \( i \) nonfaulty.
  - By Lemma 2, all of these are common.
  - Claim these form a path covering:
    - There are at most \( f \) faulty processes.
    - Each path contains \( f+1 \) labels ending with \( f+1 \) distinct indices.
    - So at least one of these labels ends with a nonfaulty process index.
Agreement

• **Lemma 4:** If there’s a common path covering of the subtree rooted at any node \(x\), then \(x\) is common

• **Proof:**
  – By induction, from the leaves up.
  – “Common-ness” propagates upward.

• **Lemma 5:** The root is common.
• **Proof:** By Lemmas 3 and 4.

• Thus, all nonfaulty processes get the same newval(\(\lambda\)).
• Yields Agreement.
Complexity bounds

- As for EIG for stopping agreement:
  - Time: $f+1$
  - Communication: $O(n^{f+1})$

- Number of processes: $n > 3f$
Next time…

• Lower bounds for Byzantine agreement:
  – Number of processors
  – Bounds for connectivity, weak Byzantine agreement.
  – Number of rounds

• Reading:
  – Sections 6.4-6.7
  – [Aguilera, Toueg]
  – (Optional) [Keidar-Rajsbaum]
6.852J / 18.437J Distributed Algorithms
Fall 2009

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.