Dependable and Distributed Systems

Professor Matthew Leeke
School of Computer Science
University of Birmingham

Topic 6 - Consensus

Consensus Problems

Problems in which a collection of distributed processors are required to reach a common decision, even if there are initial differences of opinion about what that decision ought to be

Many different consensus problems arise in practice

> Processors monitoring altimeters on board an aircraft must reach agreement on altitude
>
> Processors carrying out fault diagnosis procedures for a system component must combine diagnoses decide whether or not to replace the component

First we'll look at algorithms that can account for **link failures**, before considering approached that account for **process failures** and **byzantine failures** - then things will get really interesting!

Consensus Problems

A decision problem, meaning that all correct processes decide on a value or value vector

For example, leader election amounts to deciding on a vector of values

Usually easy to solve under assumptions of reliability but much more interesting to solve in fault-prone environments

Reaching Consensus In A Distributed System

Each of the processes in the network begins with an initial value of a particular type and will eventually output a value of that same type

The outputs are required to be the same, i.e., the processes must agree, even though the inputs can be arbitrary

> There is generally a validity condition describing the output values that are permitted for each pattern of inputs

When there are no failures of system components, consensus problems are mostly easy to solve using a simple exchange of messages

Distributed Consensus With Link Failures

The Coordinated Attack Problem

We will start by studying the **coordinated attack problem** - a fundamental problem of reaching consensus in a setting where messages may be lost

Imagine a number of generals are a coordinated attack from different directions

The only way for the attack to succeed is for all the generals to commit to attack (if only some of the generals advance then the attack fails)

Each general has an initial opinion about whether their army is ready to attack

Generals can communicate via messengers, who maybe get lost or captured

The Coordinated Attack Problem - Reliable Messages

If all messengers are reliable then the generals can send messengers to all the other generals (in several hops), saying whether or not they are ready to attack

After a number of rounds, all the generals will have a information

Number of rounds required is $diam$ (diameter of the communication graph)

Can all apply a commonly agreed rule to make the same decision about attacking

Usually rule is to attack if all the generals are ready to do so

In a model where messages may be lost, we can prove that no algorithm can correctly solve the coordinated attack problem

The Problem

Also arising in distributed database systems

A collection of processes that have participated in a database transaction

Each process arrives at an initial view on whether the transaction should be committed or aborted

> A process will want to commit the transaction if all its local involvement with that transaction is successfully completed, and will want to abort otherwise

Processes communicate and eventually agree on an outcome, commit or abort

The Problem - Formally

Assume N processes indexed $1,2, \ldots, n$, arranged in an arbitrarily undirected graph

Each process knows the entire graph, including process indices

Each process starts with an input in $\{1,0\}$

We can assume (without loss of generality) the synchronous network model

Allowing for the loss of messages in the channel

Aim: All processes to eventually output decisions in $\{1,0\}$ by setting their $decision$ state components to 0 or 1

The Problem - Formally

Three conditions are imposed on the decisions made by the processes:

Agreement:

No two processes decide on different values

Validity:

If all processes start with 0 then 0 is the only possible decision value

If all process start with 1 and all messages are delivered then 1 is the only possible decision value

Termination:

All processes eventually decide

The Problem - Formally

The agreement and termination requirements follow from the problem statement

The validity condition is just an example of one possible such statement

> Purpose is to express “reasonableness” of outcomes e.g., to guard against the trivial protocol that always decides 0
>
> We adopt the weakest statement of validity yet no algorithm can correctly solve the coordinated attack problem when messages can be lost

How Do We Prove The Impossibility?

We show for of two node graph, extending to N

Need to understand the definition of **execution** and **indistinguishable** in the synchronous network model

A state assignment of a system is defined to be an assignment of a state to each process

A message assignment is an assignment of a (possibly null) message to each channel

An execution of the system is an infinite sequence: $C_0, M_1, N_1, C_1, M_2, N_2, C_2 \ldots$, where each C_r is a state assignment and each M_r and N_r is a message assignment

> C_r represents the state after round r, whilst M_r and N_r represent to messages send and received in round r (may be different from each other if messages are lost)

How Do We Prove The Impossibility?

If α and α' are two executions, we say that α is indistinguishable from α' with respect to process i if i has the same sequences of states, the same sequence of outgoing messages and the same sequence of incoming messages in α and α'

Denoted α is indistinguishable from α' by $\alpha \sim^i \alpha'$

Can be applied to a subset of rounds

The concepts of **execution** and **indistinguishable** can be used to compare

The Commit Problem - Impossibility for Two Nodes

What we need to show: If G is the graph consisting of nodes 1 and 2 connected by a single edge, there is no algorithm that solves the coordinated attack problem on G

Proof: Suppose algorithm A solves the coordinated attack problem on G

Assume that for each process there is exactly one start state containing each input value, hence the system has exactly one execution for a fixed assignment of inputs and fixed pattern of successful messages

Let α be the execution when both processes start with value 1 and all messages are delivered

By termination and validity, both eventually decide on value 1 within within r rounds

The Commit Problem - Impossibility for Two Nodes

Let α_1 be the same as α, excepts that all messages after the first r rounds are lost

In α_1, both processes also decide on 1 within r rounds

Starting from α_1 we construct a series of executions, each of indistinguishable from its predecessor in the with respect to one of the processes

Let α_2 be the same as α_1, excepts that the last message from process 1 to process 2 is not delivered

Although process 2 may go to different states after round r in executions α_1 and α_2, this is never communicated to process 1, hence $\alpha_1 \sim^1 \alpha_2$

Since process 1 decides 1 in α_1 it also decide 1 in α_2 and process 2 (by agreement) decides 1 in α_2

The Commit Problem - Impossibility for Two Nodes

Now let α_3 be the same as α_2, excepts the last message from process 2 to process 1 is lost

Since $\alpha_2 \sim^2 \alpha_3$, process 2 decides 1 in α_3 and (by agreement) so does process 1

We alternate between removing the last message from process 1 and process 2, until we reach α' in which both processes start with 1 and no messages are delivered, meaning that each processes would decide 1

Consider the execution α'' in which process 1 starts with 1 but process 2 starts with 0 and no messages are delivered

We have $\alpha'' \sim^1 \alpha'$, hence process 1 decided 1 in α'' and (by agreement) so does process 2

However, where α''' is the execution where both processes start with 0 and no messages are received, we have $\alpha'' \sim^2 \alpha'''$

Process 2 decides 0 in α''', which gives us a contradiction by a violation of the validity condition

Implications Of The Impossibility

In a model where messages may be lost, we can prove that no algorithm can correctly solve the coordinated attack problem

Possible to solve variations of the problem where messages may be lost

- Relax the problem definition to tolerate some (hopefully small) probability of disagreement
- Possible to obtain upper and lower bounds on the probability of disagreement in term of the number of rounds but these aren’t marginal

Distributed Consensus With Process Failures

The Problem - Formally

Assume N processes indexed $1,2, \ldots, n$, arranged in a connected, undirected graph

Each process knows the entire graph, including process indices

Each process value starts from a fixed value set V

We can assume (without loss of generality) the synchronous network model

Allowing for the failure of at most f processes

All messages sent are assumed to be delivered

Aim: All processes to eventually output decisions from V by setting their $decision$ state components to values from V

The Problem - Stop Failure - Formally

Three conditions are imposed on the decisions made by the processes:

Agreement:

No two processes decide on different values

Validity:

If all processes start with the same initial value $v \in V$, then v is the only possible decision value

Termination:

All non-faulty processes eventually decide

The Problem - Byzantine Failure - Formally

Three conditions are imposed on the decisions made by the processes:

Agreement:

No two non-faulty processes decide on different values

Validity:

If all non-faulty processes start with the same initial value $v \in V$, then v is the only possible decision value for a non-faulty process

Termination:

All non-faulty processes eventually decide

Why The Changes For Byzantine Failure?

We can not impose limitations on what a faulty processes might start with or decide

We refer to the agreement problem under a Byzantine failure model as the **Byzantine agreement problem**

Relationship Between Stop and Byzantine Agreement

It is not quite the case that an algorithm that solves the Byzantine agreement automatically solves the agreement problem for stopping failures

In the stopping case, we require that all the processes decide

> Even those that subsequently fail must decide
>
> If the agreement condition for the stopping failure case is replaced by the one for the Byzantine failure case, the implication does hold

If all the non-faulty processes in the Byzantine algorithm always decide at the same round, then the algorithm also works for stopping failures

Measuring Complexity Under Byzantine Failure

Time complexity is unaffected

We can consider the number of rounds under which all non-faulty processes decide

For the communication complexity, we can count both the number of messages and number of bits of communication

Fine for stop failures

We can only use messages sent by non-faulty processes in the case of Byzantine failure, since **we can not establish non-trivial bound on the number of messages send by a faulty process**

Distributed Consensus - Stop Failure Algorithms

Algorithms Accounting For Stop Failures

The FloodSet algorithm repeatedly broadcasts the set of all values it has ever seen

We will consider how we can address the complexity issues of the FloodSet algorithm before considering an alternative approach

Exponential Information Gathering (EIG) algorithms operate by having processes relay values for rounds, recording the values they receive specific communication paths in a structure called an EIG tree - we'll see the benefit of this

FloodSet Algorithm for Consensus

Each process in a complete graph maintains a variable W containing a subset of V

The W belonging to process i initially contains i's initial value

For each of $f + 1$ rounds, each process broadcasts W, adding all the elements of the received sets to their W

After $f + 1$ rounds, process i applies a decision rule

> If W is a singleton set then i decides on the unique element of W, otherwise element i decides on v_0 (a prescribed values of V)

FloodSet Algorithm - Formally

M: message alphabet consists of subsets of V

For each i, the state in $states_i$ are:

$rounds$, a natural number (initially 0)
$decision$, values in $V \cup \{unknown\}$ (initially $unknown$)
W, subset of V (initially singleton consisting of i's initial value)

For each i, the start states $start_i$ is:

Defined by W

FloodSet Algorithm - Formally

For each i, the message generation function $msgs_i$ is:

If $rounds \leq f$ then send W to all other processes

For each i, the transition function $trans_i$ is:

$rounds := rounds + 1$
let X_j be the message from j, for each j from which a message arrives
$W := W \cup \bigcup_j X_j$
if $rounds = f + 1$ then
    if $|W| = 1$ then $decision := v$, where $W = \{v\}$
    else $decision := v_0$

FloodSet Algorithm - Correctness

We use $W_i(r)$ to denote the value of variable W at process i after r rounds

> For example, we can observe that, if no process fails during a particular round r, $1 \leq r \leq f+1$ then $W_i(r) = W_j(r)$ for all i and j that are active after r rounds

Remember that we have conditions to satisfy after we've considered the properties of the algorithm

Agreement: No two processes decide on different values

Validity: If all processes start with the same initial value $v \in V$, then v is the only possible decision value

Termination: All non-faulty processes eventually decide

FloodSet Algorithm - Correctness

The example we picked for notation is actually useful starting point

> If no process fails during a particular round r, $1 \leq r \leq f+1$ then $W_i(r) = W_j(r)$ for all i and j that are active after r rounds

Proof: Suppose that no process fails at round r and let I be the set of processes that are active after r rounds. At the end of round r, the W set of each process in I is exactly the set of value held by processes in I immediately before round r

> As $W_i(r) = W_j(r)$ for all i and j that are active after r rounds, for any round r', $r \leq r' \leq f+1$ we know $W_i(r') = W_j(r')$ for all i and j that are active after r' rounds
>
> Assuming the initial values of i and j is $v \in V$, both i and j decide v

FloodSet Algorithm - Correctness

Proof: If processes i and j are active after $f+1$ rounds then $W_i = W_j$ at the end of round $f+1$

Since there are at most f faulty processes there must be some round r, $1 \leq r \leq f+1$ at which no process fails

We know that if no process fails during a particular round r, $1 \leq r \leq f+1$ then $W_i(r) = W_j(r)$ for all i and j that are active after r rounds, and that for any round r', $r \leq r' \leq f+1$, we have $W_i(r') = W_j(r')$ for all i and j that are active after r' rounds

Knowing there are at most f faulty processes, the composition of these gives $W_i(f+1) = W_j(f+1)$ for all i and j are active after $f+1$ rounds

FloodSet Algorithm - Correctness

Termination: Termination is obvious, by the decision rule

Validity: If all the initial values are equal to v then v is the only value ever sent

> Each $W_i(f+1)$ is nonempty, because it contains i's initial value, hence $W_i(f+1)$ must be $\{v\}$ so the decision rule says that v is the only possible decision

Agreement: For agreement, let i and j be any two processes that decide

> Since decisions only occur at the end of round $f+1$, it means that i and j were active after $f+1$ rounds, hence the decision rule ensures that i and j agree

How Important Is The Decision Rule?

The decision rule we adopted works well but it is arbitrary

FloodSet guarantees that all active processes obtain the same W after $f + 1$ rounds

Other decisions rules could be devised, provided that all processes use the same rule

The rule can have implication for the conditions imposed on the algorithm

> If we, for example, make every process will choose the maximum value in W, then we guarantee a stronger notion of validity (required for k-agreement algorithms)
>
> The current rule guarantees a weaker notion of validity because the default value we select may not be the initial value of any process

FloodSet Algorithm - Complexity Analysis

FloodSet requires exactly $f + 1$ rounds until all active processes decide

The total number of messages is $O(n^2(f + 1))$

The number of bits per message is $O(nb)$, where b is an upper bound on the number of bits needed to represent any single value in V

The total number of communication bit is $O(n^3b(f + 1))$

FloodSet Algorithm - Optimisation

Possible to improve on $O(n^2(f+1))$ messages and $O(n^3b(f+1))$ bits

Observe that in round $f+1$, each process i only needs to know the exact elements of its set W_i if $|W_i| = 1$, otherwise, i needs to know only the fact that $|W_i| > 2$

Each process might only need to broadcast the first two values it encounters!

FloodSet Algorithm - Optimisation

Perform FloodSet, except:

Each process i is only allowed to broadcast two values altogether

The first broadcast is at round 1, when i shares its initial value

The second broadcast is at the first round r, $2 \leq r \leq f+1$, such that at the start of round r it is the case that i knows about some value v that is different from its initial value (if it exists), when i shares v

(if there are multiple values received at any point, we can broadcast any)

FloodSet Algorithm - Optimisation

The number of rounds required by the optimised version remains $f + 1$

The number of messages is at most $2n^2$

Every process sends at most two non-null messages to each other process

The number of bit of communication is $O(bn^2)$

Exponential Information Gathering (EIG)

Exponential Information Gathering (EIG) Algorithms

Operate by processes relaying values for rounds, recording the values they receive on specific communication paths in a structure called an EIG tree

Poor time and communication complexity for solving consensus with stop failures but the same approach has advantages under Byzantine failures

We use a labelled EIG tree $T = T_{n,f}$

> Paths from the root represent chains of distinct processes along which initial values are propagated
>
> Each node at level $k, 0 \leq k \leq f$ has exactly $n - k$ children

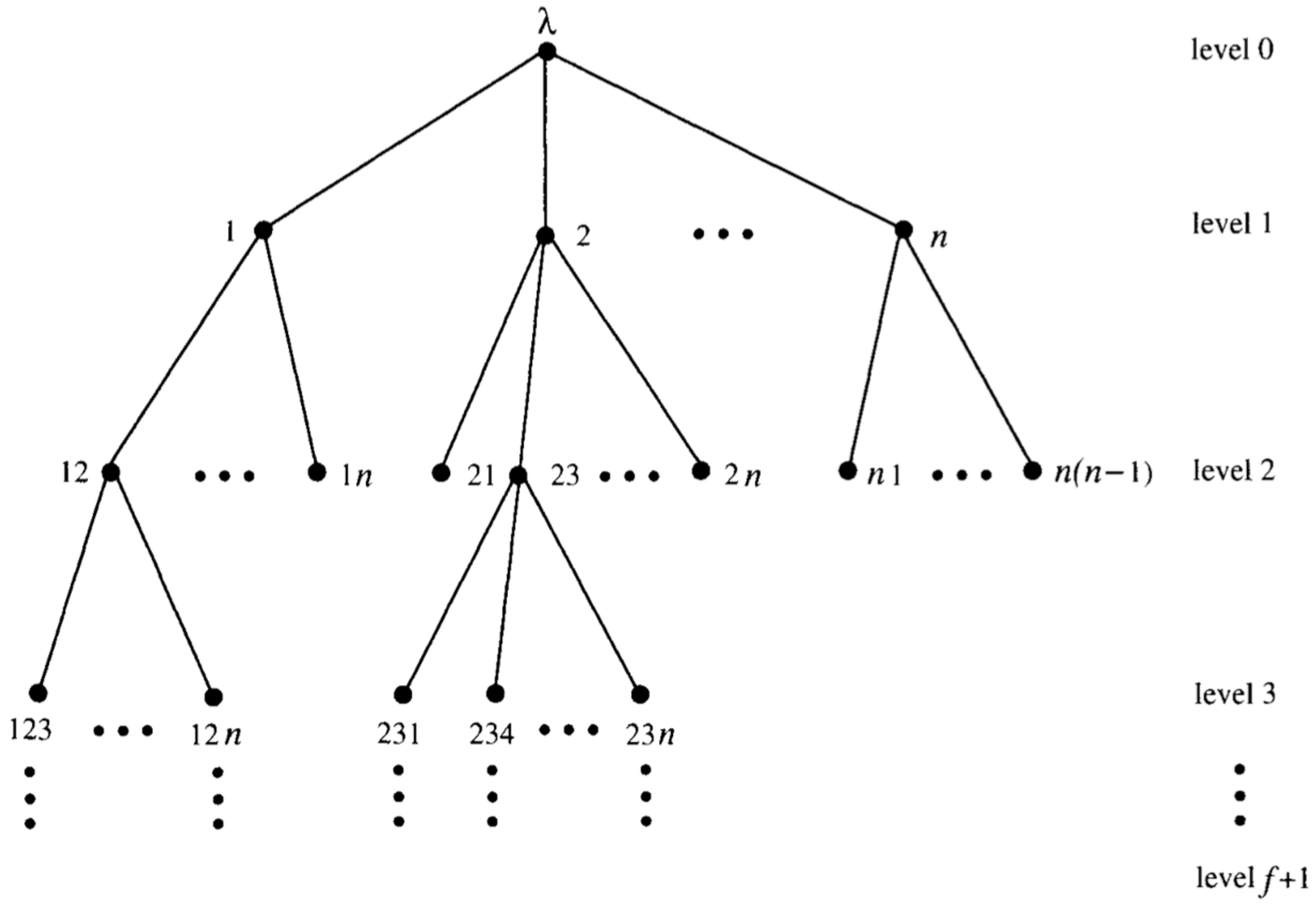
Exponential Information Gathering (EIG) Algorithms

Each node in T is labelled by a string of process indices

Root is labelled by empty string

Node with label $i_1, \ldots, i_k$ has exactly $n - k$ children with labels $i_1, \ldots, i_k j$

Here j ranges over all elements of $\{1, \ldots, n\} - \{i_1, \ldots, i_k\}$



EIG Algorithm For Stop Failure

Processes send values on all possible paths

Each process maintains a local copy of the EIG tree

Computation performed over exactly $f + 1$ rounds

Processes decorate the nodes of their trees with values in $V \cup \{null\}$

Decorating all those at level k at the end of round k

Root of process i's tree decorated with i's input value

EIG Algorithm For Stop Failure

If the node labelled by the string $i_i \ldots i_k$, $1 \leq k \leq f+1$ is decorated by a value $v \in V$ then i_k has told i at round k that i_{k-1} has told i_k at round $k-1$ that … that i_1 has told i_2 at round 1 that i's initial value is v

If the node labelled by the string $i_i \ldots i_k$ is decorated by $null$ then the chain of communication $i_1, i_2, \ldots i_k, i$ has been broken by failure

After $f+1$ rounds each process uses their local tree to decide on a value in V based on a commonly agreed decision rule

EIG Algorithm For Stop Failure

To help us define the algorithm more precisely we need a little notation:

For every string x that is a label of a node of T, each process has a variable $val(x)$

Variable hold the value with which the process decorates the node labelled x

Each process initially decorates the root of its T with its own initial value, i.e., it sets $val(\lambda)$ to its initial value

Finally, we can set out the algorithm…

EIG Algorithm For Stop Failure

Round 1: Process i broadcasts $val(\lambda)$ to all processes before recording incoming information:

1. If a message with value $v \in V$ arrives at i from j then i sets its $val(j)$ to v

2. If no message with value in V arrives at i from j then i sets its $val(j)$ to $null$

Round k, $2 \leq k \leq f+1$: Process i broadcasts all pairs (x, v), where x is a level $k-1$ label in T that does not contain index i, $v \in V$ and $v = val(x)$ before recording incoming information:

1. If x_j is a level k label, where x is a string of process indices and j is a single index, and a messages saying that $val(x) = v \in V$ arrives at i from j then i sets $val(x_j)$ to v

2. If xj is a level k label and no message with a value in V arrives at i from j then i sets its $val(x_j)$ to $null$

EIG Algorithm For Stop Failure

What is the decision rule?

Let W be the set of non-null values that decorate notes of i's tree

If W is a singleton set then i decides on the unique element of W, otherwise i decides on v_0

It looks similar to FloodSet, can we change the rule?

Since the EIG algorithm for stop failure guarantees the same W for active processes after $f + 1$ rounds, we can change the rule

For example, we could take the minimum value in W, which would guarantee that stronger notion of validity, just like we say with FloodSet

EIG Algorithm For Stop Failure - Complexity Analysis

The EIG algorithm for stop failures requires exactly $f + 1$ rounds until all active processes decide

The total number of messages is $O(n^2(f + 1))$

The number of bits per message is $O(bn^{f+1})$

Exponential in the number of failures make it unattractive in fault-prone environments - why not use optimised FloodSet?!

EIG Algorithm For Stop Failure - Optimisation

Despite them not looking particularly similar on the surface, it is possible to optimise the EIG algorithms for stop failure using the same approach as FloodSet

Perform FloodSet, except:

> Each process i is only allowed to broadcast two values altogether
>
> The first broadcast is at round 1, when i shares its initial value
>
> The second broadcast is at the first round $r, 2 \leq r \leq f + 1$, such that at the start of round r it is the case that i knows about some value v that is different from its initial value (if it exists), when i shares v and any $r - 1$ label decorated with v

Next Time…

Byzantine Generals Problem

What is it?

Can we solve it?

If so, can we prove what conditions can we solve it under?

Can we improve on our solution by introducing technology that wasn't invented in Byzantium?