

Introduction

What is *distributed computing*?

A distributed system is a collection of processes communicating to solve a problem.

Why do they need to *communicate*?

They communicate to:

- Share Data
- Share workload (split problem to save time).
- Share resources (for example, printer).
- Implement high-level communication application (e.g. email).
- Controlling distributed robots (for example, nuclear power plant).

What kinds of systems?

There are many different aspects of distributed computing models we have to consider:

- I. Communication Mechanisms
 - Message passing.
 - Shared Memory.
 - Remote function calls (usually message calling in disguise).
- II. Processes
 - Multiple processes on same processor (time-sharing).
 - Different processors in same box.
 - Different processors geographically distributed.
- III. Tolerating Failures
 - Crash failures (unpredictable death of a process).
 - Messages can be lost, corrupted, or duplicated.
 - Byzantine failures - faulty processes behave arbitrarily badly. (They can deviate from their algorithms).
- IV. Timing assumptions
 - Process speeds (and also message transmission times).
 - Asynchronous (system processes run at arbitrarily different speeds, no bounds on message transmission times)
 - vs.
 - Synchronous (processes run at same speed, known message transmission time)
 - vs.
 - Partially synchronous (for example, bounds on speeds of processes).

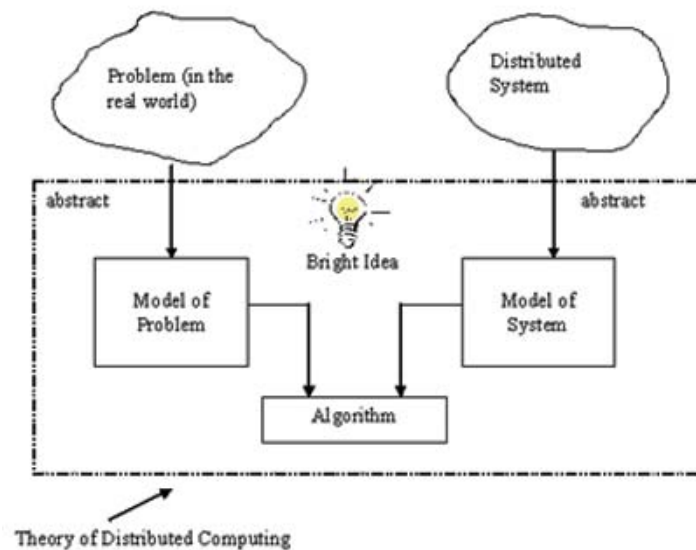


Figure 1: The big picture

- V. Other Aspects of Modelling Systems

- Deterministic vs. Randomized algorithms.
- Bounds on capacity of processes (for example, in sensor networks with tiny, weak components).
- Degree of knowledge about system. For example, do processes know whole network graph or small neighbourhood or nothing?

The area on which *Distributed Computing Theory* focuses is shown in figure 1.

Example: Two Generals

Let us consider an example with the following goals and assumptions as shown below in figure 2.

Goal

Generals leading the two red armies should attack the blue army at the same time to avoid losing the battle.

Assumptions

- Generals have synchronized clocks.
- Communication is by messages through the valley. Messenger might get captured. Message delivery takes ≤ 30 minutes if it is successful.

Conditions

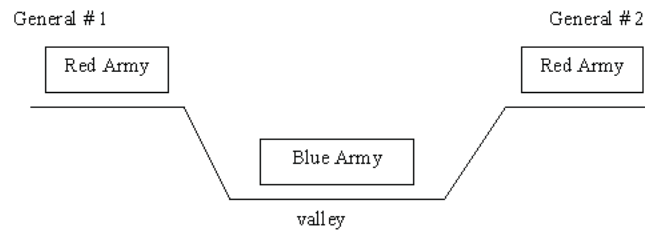


Figure 2: The valley

1. The two red generals must each eventually choose a time to attack.
2. Times chosen must be equal (in order to be successful).

Trivial Solution for conditions 1 and 2:

Attack at dawn.

To make the problem more interesting, assume each general starts with a preference of what time to attack.

Thus there is one more condition:

3. Time chosen should be one of the preferences if no messengers are captured. (Note that this still allows them, to switch to default time of dawn if all messengers are captured, or even if a single messenger is captured).

Attempted Algorithms

General #1 sends his preferred time t to General #2. Problem: what if messenger is captured? General 1 doesn't know this.

So General 2 should send back an acknowledgement.

When acknowledgment (ack) is received by #1, attack at time t . If no ack, after an hour, attack at dawn.

But there is a problem. If #2 sends an ack, and #1 never receives it, then #1 attacks at dawn and #2 attacks at time t according to the algorithm.

So let us add another acknowledgment (ack) of the acknowledgement. We also add the following to the algorithm:

When ack is received by #2, attack at t , else attack at dawn. This is shown below in figure 3.

But here is still a problem, that is, if the messenger bringing the second ack is captured, then #2 attacks at dawn and #1 attacks at t .

Eventually, we start to suspect that this problem cannot be solved. Let us prove that it cannot be solved.

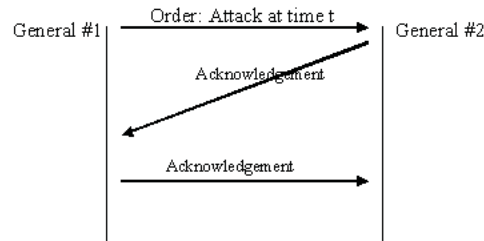


Figure 3: Attempted algorithm

Claim

It is impossible to design an algorithm that satisfies conditions 1 , 2 and 3.

Proof

Suppose \exists an algorithm, Derive a contradiction.

Let α be the execution where both generals prefer noon and no messages are lost.

By condition 3, outcome is noon.

Let β be an execution with no messages lost, where both generals prefer 1:00.

By 3, outcome of β must be 1:00.

β is the left execution in figure 4.

-Let β_1 be same as β except last message is lost.

-Sender of the last message cannot distinguish β from β_1 and must output 1:00 in β_1 . Also, receiver must output 1:00 in β_1 by condition 2.

-Let β_2 be same as β_1 except last message delivered in β_1 is lost.

By same argument, both processes output 1:00.

In general, let β_k be same as β with last k messages removed.

β_k and β_{k+1} are indistinguishable to one process (namely, the sender of the message removed from β_k to form β_{k+1}). So output of both processes is same in β_k and β_{k+1} .

If β has n messages, then by *Induction*, β_n has output 1:00.

Note: β_n is the execution where both processes start with preference 1:00 and NO messages get through.

Let α' be execution where both have preference noon, and NO messages are delivered.

By same argument, outcome of α' is noon.

This is shown in figure 5 (leftmost execution).

The rightmost execution in figure 5 is β_n .

The middle execution of figure 5 is an execution where no messages get through, and General 1 starts with the preference of noon and General 2 starts with the preference of 1:00.

Note that the left two executions in the figure are indistinguishable to General 1, so General 1 must output noon in the middle execution.

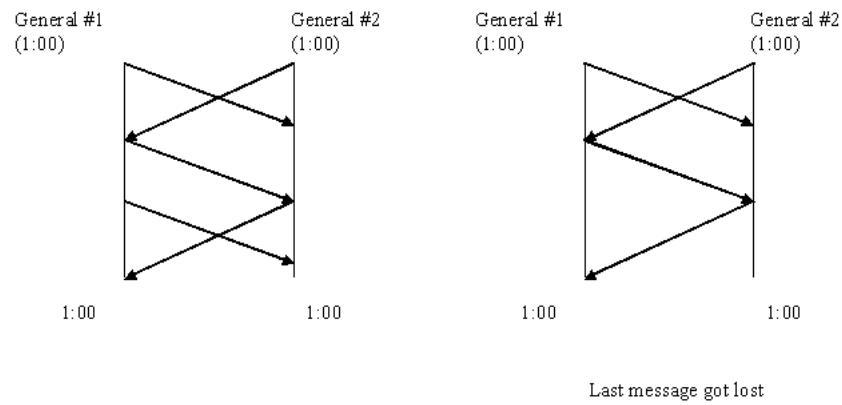


Figure 4: Losing a message

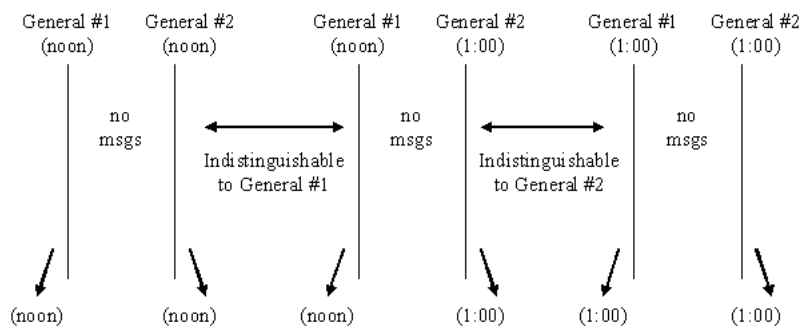


Figure 5: Three executions

Similarly, the right two executions are indistinguishable to General 2, so General 2 must output 1:00 in the middle execution.

Condition 2 is violated in the middle execution.

This contradicts the assumption that we have a correct algorithm.

Thus, no algorithm exists to solve this problem.