

Randomized Distributed Algorithm



Rajiv Misra

Dept. of Computer Science &
Engineering

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Introduction

- This lecture focused on a specific type of distributed algorithms, which employ **randomization**.
- Randomization has **proved to be a very powerful tool** for designing distributed algorithms (as for many other areas).
- It often simplifies algorithms and more importantly **allows us to solve problems in situations where they cannot be solved by deterministic algorithms**, or with fewer resources than the best deterministic algorithm.

Contd...

- This lecture extends the formal model to include randomization and describes randomized algorithm for **Leader Election Problem**.
- Randomization allows us to **overcome impossibility results and lower bounds**, by relaxing the termination conditions or the individual liveness properties (in the case of mutual exclusion).

Distributed Leader Election Algorithm: A Case Study

Weakening the Problem Definition

- A **randomized** algorithm is an algorithm that has access to some source of random information, such as that provided by flipping a coin or rolling dice.

More formally:

- **Extend the transition function** of a processor to take as an additional input a random number, drawn from a bounded range under some fixed distribution.

Why is it important?

The Bad News:

1. The addition of random information alone typically will not affect the existence of **impossibility results or worst-case bounds**.
2. For instance, even if processors have access to random numbers, they will not be able to elect a leader in an anonymous ring or solve consensus in fewer than **$f+1$ rounds** in all (admissible) executions.

Contd...

The Good News:

- **randomization + *weakening*** of problem statement is a powerful tool for overcoming limitations.
- Usually the **weakening** involves the termination condition (for instance, a leader must be elected with a certain probability) while the other conditions are not changed (for instance, it should never be the case that two leaders are elected).

Differs from a average case analysis of a deterministic algorithm

- **In average case analysis**, there are several choices as to what is being averaged over. One natural choice is the inputs. There are two difficulties with this approach:

- (i) Determining an accurate probability distribution on the inputs is often not practical.
- (ii) Another drawback is even if such distributions can be chosen with some degree of confidence, very little is guaranteed about the behavior of the algorithm on a particular input. For instance, even if the **average** running time over all inputs is determined to be small, there still could be some inputs for which the running time is enormous.

Contd...

- In the randomized approach, more stringent guarantees can be made. Because the random numbers introduce another dimension of variability even for the same inputs there are many different executions for the same input.
- A good randomized algorithm will guarantee good performance with some probability for each individual input.
- Typically the performance of a randomized algorithm is defined to be the **worst-case probability over all inputs.**

Randomized Leader Election

Randomized Leader Election

- The simplest use of randomization is **to create initial asymmetry in situations that are inherently symmetric.**
- One such situation is anonymous rings, where processors do not have distinct identifiers, it is **impossible to elect a unique leader.**
(From Theorem: There is no nonuniform anonymous algorithm for leader election in synchronous rings)
- This impossibility results holds even for randomized algorithms. However a randomized algorithm formulation of LE “leader is elected **with some probability**”.
- A variant of the leader election problem that relaxes the condition that eventually a leader must be elected in every admissible execution. Relaxed version of the leader election problem requires:

Safety: In every configuration of every admissible execution, at most one processor is in an elected state

Liveness: At least one processor is elected with some non-zero probability

Contd...

- **Safety property has to hold with certainty:** that is, the algorithm should never elect two leaders.
- **Liveness condition is relaxed,** and the algorithm **need not always terminate with a leader**, rather, it is required to do so ***with nonzero probability.***
- An algorithm that satisfies this weakened liveness condition can fail to elect a leader either by not terminating at all or by terminating without a leader.

1. Synchronous One-Shot Algorithm

- First, let us consider **synchronous rings**.
- There is only one admissible execution on an anonymous ring for a deterministic algorithm.
- For a randomized algorithm, however, there can be many different executions, depending on the random choices.
- The approach that is used to devising a randomized leader election algorithm is to use randomization **to create asymmetry by having processors choose *random pseudo-identifiers***, drawn from some range, and then execute a deterministic leader election algorithm.

Properties

- A simple deterministic leader election algorithm with these properties is the following:
 1. Each processor sends a message around the ring to collect all pseudo-identifiers.
 2. When the message returns (after collecting ***n pseudo-identifiers***), a processor knows whether it is a unique maximum or not.

Algorithm: Overview

- In this algorithm, the **pseudo-identifier is chosen to be 2 with probability $1/n$ and 1 with probability $(1 - 1/n)$** , where n is the number of processors on the ring.
- Thus each processor makes use of its source of randomness exactly once, and the random numbers are drawn from **the range $[1...2]$** .
- The set of all possible admissible executions of this algorithm for fixed ring size n contains exactly one execution for each element of the **set $\mathfrak{R} = \{1,2\}^n$**
- That is, by specifying which random number, 1 or 2, is obtained by each of the n processors in its first step, we have completely determined the execution. Given an element R of $\mathfrak{R}$ then the corresponding execution will be denoted by ***exec(R)***

Algorithm1

Algorithm 1 Randomized leader election in an anonymous ring:

code for processor p_i , $0 \leq i \leq n - 1$.

- 1: initially // spontaneously or upon receiving the first message
- 2: $id_i := \begin{cases} 1 & \text{with probability } 1 - \frac{1}{n} \\ 2 & \text{with probability } \frac{1}{n} \end{cases}$ // choose pseudo-identifier
- 3: send $\langle id_i \rangle$ to left

- 4: upon receiving $\langle S \rangle$ from right
- 5: if $|S| = n$ then // your message is back
- 6: if id_i is the unique maximum of S then become *elected* // the leader
- 7: else become *non-elected* // a nonleader
- 8: else // concatenate your id to the message and forward
- 9: send $\langle S \cdot id_i \rangle$ to left

Definition

- Let **P be some predicate** on executions, for example, at least one leader is elected.
- Then $\Pr[P]$: probability of the event

$$\{R \in \mathfrak{R} : \text{exec}(R) \text{ satisfies } P\}$$

Analysis

- **What is the probability that the algorithm terminates with a leader?**
- This happens when a single processor has the maximum identifier, 2.
- The probability that a single processor draws 2 is the probability that $n-1$ processors draw 1, and one processor draws 2, times the number of possible choices for the processor drawing 2, that is,

$$\binom{n}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \left(1 - \frac{1}{n}\right)^{n-1} = c > \left(1 - \frac{1}{n}\right)^n \rightarrow \frac{1}{e}$$

Contd...

- It is simple to show that every processor terminated after sending exactly n messages; moreover, at most one processor terminates in an elected state.
- In some executions, for example, when two processors choose the pseudo-identifier 2, no processor terminates as a leader, However, the above analysis shows that this happens with probability less than $1 - 1/e$

Message Complexity

- **Theorem 1:** *There is a randomized algorithm that, with probability $c > 1/e$, elects a leader in a synchronous ring; the algorithm sends $O(n^2)$ messages.*

2. Synchronous Iterated Algorithm and Expectation

- It is pleasing that the probability of termination in previous algorithm does not decrease with the **ring size n** .
- However, we may wish to increase the probability of termination, at the expense of more time and messages.
- The algorithm can be modified so that each processor receiving a message with **n pseudo-identifiers** checks whether a unique leader exists (in Lines 5-7 of Algorithm1).
- If not, the processor chooses a new pseudo-identifier and iterates the algorithm. We will show that this approach amplifies the probability of success.

Contd...

- Let us consider the second option in more detail. In order to repeat the **Algorithm1**, each processor will need to access the random number source multiple times, in fact, potentially infinitely often.
- To completely specify an execution of the algorithm, we will need to specify, for each processor, the **sequence** of random numbers that it obtains.
- Thus $\mathcal{R}$ now becomes the set of all ***n-tuples***, each element of which is a possibly infinite sequence over $\{1,2\}$.

Analysis

- For the iterated algorithm, **the probability that the algorithm terminates at the end of the k th iteration is equal to the probability that the algorithm fails to terminate in the first $k-1$ iterations and succeeds in terminating in the k th iteration.**
- The analysis of Algorithm1 shows that the probability of success in a single iterations is **$c > 1/e$.**
- Because the probability of success or failure in each iteration is independent, the desired probability is

$$(1 - c)^{k-1} c$$

- This probability tends to 0 as k tends to ∞ ;thus the probability that the algorithm terminates with an elected leader is 1.

Time Complexity of Iterated Algorithm

- **Worst-case number of iterations:** ∞
- **Expected number of iterations:** $1/c < e$
- Let **T be a random variable** that, for a given execution, is the value of the complexity measure of interest for that run (for instance, the number of iterations until termination).
- Let **$E[T]$ be the expected value of T** , taken over all $R \in \mathcal{R}$
- That is,

$$E[T] = \sum_{x \text{ is a value of } T} x \cdot \Pr[T = x]$$

Contd...

- Note that by the definition of $\Pr[P]$ above, this is ultimately taking probabilities over $\mathcal{R}$
- With this definition, we have the following theorem:
- **Theorem 2:** *There is a randomized algorithm that elects a leader in a synchronous ring with probability 1 in $(1/c) \cdot n < e \cdot n$ expected rounds; the algorithm sends $O(n^2)$ expected messages.*

Conclusion

- Randomization has **proved to be a very powerful tool** for designing distributed algorithms.
- It allows us to solve problems in situations **where they cannot be solved by deterministic algorithms.**
- This lecture extends the formal model to include randomization and describes the **randomization algorithm for Leader Election Problem.**
- We have presented two randomized algorithms:
(i) **Synchronous One-Shot Algorithm with $O(n^2)$ messages** and (ii) **Synchronous Iterated Algorithm with $O(n^2)$ expected messages**