Analysis of Concurrent and Distributed Programs

Burcu Kulahcioglu Ozkan and Soham Chakraborty

09.02.2022

Outline

(1) Introduction

- Course logistics
- Lecture content
- Course projects

Overview of concurrency

- Paradigms
- Concepts

Interactions

Concurrent Programming: Paradigms

Multiple computations *that may/may not influences one another*

- Parallelism (SIMD, MIMD, ...)
  - Independent computation on multiple threads/processes on independent data
- Distributed computing
  - Independent computation on multiple machines with message passing
- Asynchronous programmming
  - Multiple tasks on single/multiple threads that may share memory
  - Event driven systems
- Shared memory concurrency
  - Multiple threads that communicate by shared memory

Concurrent Programming: Motivation

Pros

+ Concurrency improves performance

Cons

- Concurrent programming is hard
- May result in tricky bugs in programs

Requires careful analysis

Concurrent Programming: Motivation

Pros

\+ Concurrency improves performance

Cons

\- Concurrent programming is hard

\- May result in tricky bugs in programs

Requires careful analysis

This course !

Example

$$X = 0;$$

$$X = X + 1; \;\Big\|\; X = X + 1;$$

What is the final value(s) of X?

Example

$$X = 0;$$

$$X = X + 1; \;\|\; X = X + 1;$$

What is the final value(s) of X?

- Expected: $X = 2$.

Example

$$X = 0;$$

$$\begin{array}{l|l} a = X; & b = X; \\ a = a + 1; & b = b + 1; \\ X = a; & X = b; \end{array}$$

Example: Execution (1)

$$X = 0;$$

$$\begin{array}{ll|l} a = X; & // \ 0 & b = X; \\ a = a + 1; & & b = b + 1; \\ X = a; & & X = b; \end{array}$$

Example: Execution (1)

$$X = 0;$$

$$\begin{array}{ll|l} a = X; & // \ 0 & b = X; \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \end{array}$$

$$X = 0;$$

$$\begin{array}{l} a = X; \quad // \ 0 \\ a = a + 1; \\ X = a; \quad // \ 1 \end{array} \;\Bigg\|\; \begin{array}{l} b = X; \quad // \ 1 \\ b = b + 1; \\ X = b; \end{array}$$

Example: Execution (1)

$$X = 0;$$

$$\begin{array}{ll|l} a = X; & // \ 0 & b = X; \quad // \ 1 \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \quad // \ 2 \end{array}$$

Example: Execution (2)

$$X = 0;$$

$$\begin{array}{l|l} a = X; \quad // \ 0 & b = X; \quad // \ 0 \\ a = a + 1; & b = b + 1; \\ X = a; & X = b; \end{array}$$

Example: Execution (2)

$$X = 0;$$

$$\begin{array}{l|l} a = X; \quad // \ 0 & b = X; \quad // \ 0 \\ a = a + 1; & b = b + 1; \\ X = a; \quad // \ 1 & X = b; \quad // \ 1 \end{array}$$

Example

$$X = 0;$$

$$X = X + 1; \;\|\; X = X + 1;$$

What is the final value(s) of X?

Example

$$X = 0;$$

$$X = X + 1; \;\|\; X = X + 1;$$

What is the final value(s) of X?

- Expected: $X = 2$.
- Reality: $X \in \{1, 2\}$

Analysis Questions

Analysis Questions

What are the semantics of these primitives?

Analysis Questions

What are the semantics of these primitives?

Given a concurrent program and an outcome,

Analysis Questions

What are the semantics of these primitives?

Given a concurrent program and an outcome,

- Is it correct?
- Is it a bug?
- What are common concurrency bugs?
- Will it happen in *all* execution?
- Will it happen in *at least one* execution?

Analysis Questions

What are the semantics of these primitives?

Given a concurrent program and an outcome,

- Is it correct?
- Is it a bug?
- What are common concurrency bugs?
- Will it happen in *all* execution?
- Will it happen in *at least one* execution?

After finding a bug:

- How to fix it?
- Is it the best way to fix it?
- Is it affecting performance?

Going Back to the Example

$$X = 0;$$

$$\begin{array}{ll|l} & & \\ a = X; & // \ 0 & b = X; \quad // \ 1 \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \quad // \ 2 \end{array}$$

✓

$$X = 0;$$

$$\begin{array}{ll|l} & & \\ a = X; & // \ 0 & b = X; \quad // \ 0 \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \quad // \ 1 \end{array}$$

✗

Going Back to the Example

$$X = 0;$$

$$\begin{array}{ll|l} a = X; & // \ 0 & b = X; \quad // \ 1 \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \quad // \ 2 \end{array}$$

✓

$$X = 0;$$

$$\begin{array}{ll|l} a = X; & // \ 0 & b = X; \quad // \ 0 \\ a = a + 1; & & b = b + 1; \\ X = a; & // \ 1 & X = b; \quad // \ 1 \end{array}$$

✗

Solution: use lock/unlock primitives.

$$X = 0;$$

$$\begin{array}{l|l} lock(m) & lock(m) \\ a = X; & b = X; \\ a = a + 1; & b = b + 1; \\ X = a; & X = b; \\ unlock(m) & unlock(m) \end{array}$$

$$\mathit{Init}$$

$$\begin{array}{l|l} \mathit{lock}(m) & \mathit{lock}(m) \\ S_1; & S_2; \\ \mathit{unlock}(m) & \mathit{unlock}(m) \end{array}$$

$$\Downarrow$$

$\mathit{Init}; S_1; S_2;$ or $\mathit{Init}; S_2; S_1;$

lock/unlock

- Reduces concurrency

Solution: Tradeoff

$$Init$$

$$\begin{array}{l|l} lock(m) & lock(m) \\ S_1; & S_2; \\ unlock(m) & unlock(m) \end{array}$$

$$\Downarrow$$

$Init; S_1; S_2;$ or $Init; S_2; S_1;$

lock / unlock

- Reduces concurrency

Goal: minimal lock/unlock for correct programming

Lock Based Concurrency

Locking mechanism.

- Primitive: mutex
- Properties: mutual exclusion

Errors

- Datarace
- Atomicity violation

Pitfalls

- Deadlock
- Livelock
- Termination

Lock free programming

- Non-blocking concurrency

Analysis

Concurrency analysis for multithreaded programs

- Race detection
- Atomicity violation detection

Analysis techniques

- Static and dynamic analysis
- Model checking
- testing

Weak Memory Concurrency

Sequential consistency (SC)

Weak memory (Non-SC) models

- TSO
- PSO
- RMO
- RA
- ...

Analysis of weak memory programs

Distributed Programming

- Many flavors of concurrency
  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

Distributed Programming

- Many flavors of concurrency
  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

Distributed Programming

- Many flavors of concurrency

  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

- More on distributed concurrency

Distributed Programming

- Many flavors of concurrency
  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

- More on distributed concurrency

- Concurrency analysis for distributed programs

Distributed Programming

- Many flavors of concurrency

- Single-threaded/multi-threaded asynchronous
- Event driven
- Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

- More on distributed concurrency

- Concurrency analysis for distributed programs

- Replicated systems: Linearizability & CAP Theorem

Distributed Programming

- Many flavors of concurrency
  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

- More on distributed concurrency

- Concurrency analysis for distributed programs

- Replicated systems: Linearizability & CAP Theorem

- Replicated systems: Strong vs weak consistency in distributed systems

- Many flavors of concurrency
  - Single-threaded/multi-threaded asynchronous
  - Event driven
  - Distributed

- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

- More on distributed concurrency

- Concurrency analysis for distributed programs

- Replicated systems: Linearizability & CAP Theorem

- Replicated systems: Strong vs weak consistency in distributed systems

- Replicated systems: Strong vs weak isolation in distributed systems