Distributed Information Processing

7th Lecture

Eom, Hyeonsang (엄현상)
Department of Computer Science & Engineering
Seoul National University

©Copyrights 2016 Eom, Hyeonsang All Rights Reserved
Outline

- Distributed Shared Memory
  - Introduction
  - Memory Consistent Models
  - Memory Coherence
  - Other Consistency Models
    - Weak Consistency
    - Release Consistency
    - Entry Consistency

- Q&A
Distributed Shared Memory (DSM)

- **Definition**
  - Sharing Data between Processors That Do Not Share Physical Memory

- **Comparison with Message Passing**
  - No Marshalling
  - Normal Synchronization
  - Possibly Comparable Efficiency

Marshalling: Structured Data Items & Values → External Data Representation
DSM (Cont’d)

Implementation Approaches

- **Hardware**
  - E.g., Dash Multiprocessor (64 Nodes in a NUMA)

- **Paged Virtual Memory**
  - E.g., IVY

- **(Platform-Independent) Middleware**
  - E.g., Linda (Collection of Immutable Data Items)

The Page-Based Approach Is Flexible (Enabling Shared-Memory Programs to Run on Distributed-Memory Machines) Because No Particular Structure on DSM Is Required.
Memory Consistency Model

Model Specifying the Consistency Guarantees about the Values of Read Objects

With Copies of Objects Read and Objects Updated by Processes

Process 1

\[
\begin{align*}
\text{br} & := b; \\
\text{ar} & := a; \\
\text{if}(\text{ar} \geq \text{br}) & \text{ then} \\
\text{print ("OK");}
\end{align*}
\]

Process 2

\[
\begin{align*}
a & := a + 1; \\
b & := b + 1;
\end{align*}
\]

Could the Combination ar=0 and br=1 Occur?
Consistency Models

- Linearizability

  - L1: \( \text{R}(x)a \Rightarrow \) Either \( \text{W}(x)a \) before it or no write before it if \( a \) is the initial value of \( x \)

  - The order is consistent with the real times

- Sequential Consistency

  - L1 & the order is consistent with the program (execution) order

  - The combination \( \text{ar}=0 \) and \( \text{br}=1 \) could not occur under sequential consistency

\[\begin{align*}
\text{Process 1} & : \text{br} := b; \\
& \text{ar} := a; \\
& \text{if}(\text{ar} \geq \text{br}) \text{ then print } ("\text{OK}"");
\end{align*}\]

\[\begin{align*}
\text{Process 2} & : \text{a} := a + 1; \\
& \text{b} := b + 1;
\end{align*}\]
Consistency Models (Cont’d)

- **Causal Consistency**
  - Read Value Is Consistent with the Happened-before Relationship

**Illustration: Distinction between Sequential & Causal Consistency**

\[ P1: W(x) a \]

\[ P2: W(x) b \]

\[ P3: R(x) a \quad R(x) b \]

\[ P4: R(x) b \quad R(x) a \]

→Time

The Sequence Is Causally-Consistent, But Not Sequentially-Consistent
Consistency Models (Cont’d)

- **FIFO or Pipelined RAM Consistency**
  - Order of Writes Issued by any Given Processes Is Consistent

**Illustration: Distinction between Causal & FIFO Consistency**

P1: \( W(x) a \)

P2: \( R(x) a \quad W(x) b \quad W(x) c \)

P3: \( R(x) b \quad R(x) a \quad R(x) c \)

P4: \( R(x) a \quad R(x) b \quad R(x) c \)

→Time

The Sequence Is FIFO-Consistent, But Not Causal-Consistent
## Summary of Consistency Models

Models Not Using Synchronization Operations

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td>Linearizability</td>
<td>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp</td>
</tr>
<tr>
<td>Sequential</td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time.</td>
</tr>
<tr>
<td>Causal</td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td>FIFO</td>
<td>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order</td>
</tr>
</tbody>
</table>
Memory Coherence

- Sequential Consistency on a Location-by-Location Basis
  - Agreement on the Order of Writes to the Same Location without Necessarily Agreeing on the Order of Writes
Update Options

- **Write-Update/Broadcast**: Multicast of Local Updates
  - **Multiple-Reader/Multiple-Writer Sharing**
    - Achieving Sequential Consistency with totally ordered (blocking) multicast

- **Write-Invalidate**: Acknowledged Invalidation of Copies before the Write
  - **Multiple-Reader/Single-Writer Sharing**
    - Achieving Sequential Consistency

Inexpensive Reads

More Suited to Page-Based DSM
Other Consistency Models

**Weak Consistency (WC)**

- Accesses to Synchronous Vars for a Data Store Are Sequentially Consistent
- Operations on a Synchronous Var Are Performed after All Previous Writes Have Completed
- Operations Are Performed after All Previous Operations on Synchronous Vars Have Been Performed

A Single Operation Synchronizes All Its Copies of the Data Store

Weak Consistency Enforces Consistency on a Group of Operations
Other Consistency Models

Illustration: Valid vs Invalid WC Sequences

P1: $W(x) \alpha \ W(x) \beta \ S$

P2: $R(x) \alpha \ R(x) \beta \ S$

P3: $R(x) \beta \ R(x) \alpha \ S$

→ Time

P1: $W(x) \alpha \ W(x) \beta \ S$

P2: $S \ R(x) \alpha$

P2: $S \ R(x) \beta \ R(x) \alpha \ S$

Valid WC Sequence

Invalid WC Sequence
Synchronization Accesses

Characteristics
- Concurrency
- At Least One Write

Types
- `Acquire(int &lock): // Call by Ref`
  ```
  while (testAndSet(lock)=1)
  skip;
  ```
- `Release(int &lock): // Call by Ref`
  ```
  lock := 0;
  ```

The Function Sets the Lock to 1 and Returns 0 If It Is 0; Otherwise, It Returns 1
Other Consistency Models

Release Consistency (RC)

- Acquire and Release Operations Are Sequentially Consistent
- Release Operations Are Performed after All Previous Operations Have Completed
- Operations Are Performed after All Previous Acquire Operations Have Been Performed

Once a Release Has Occurred, Another Process Acquiring a Lock Can Read Only Data Modified by the Process Performing the Release

Acquire as Entering a Critical Section and Release as Leaving a Critical Section
Other Consistency Models

Illustration: Processes under RC

Process 1:

Acquire (); // enter the critical section
a := a + 1;
b := b + 1;
Release (); // leave the critical section

Process 2:

Acquire (); // enter the critical section
print ("a = ", a, "; b = ", b);
Release (); // leave the critical section

The Critical Sections Enforce Consistency: \(a=b=0\) or \(a=b=1\)

No Blocking Up to This Point: It Is When Communication Is Required

The Programmer or a Compiler Is Responsible for Labeling Reads and Writes as Releases or Acquires
Other Consistency Models

- Implementation: Lazy RC (in Contrast to the Eager RC)
  - Communication is Delayed until the Next-Acquire Time
    - Saving the network bandwidth

- Issue: False Sharing
  - Having Data Belonging to Two Independent Processes in the Same Page with at Least One Writing Process
    - Single-Writer vs Multiple-Writer Protocols

Leading to Unnecessary Communication
Other Consistency Models

Entry Consistency (Associating Shared Data with Synch Vars)

- First Acquire Makes the Latest Values Visible
- Write Requires Entering the Critical Section
- Multiple Reads May Be Performed after the Writer Has Left It

This Avoids the Tendency to False Sharing at the Expense of Increased Programming Complexity: e.g., Midway

Illustration: Valid Entry Consistency Sequence

P1: \( \text{Acq}(x) \ W(x) \ a \ \text{Acq}(y) \ W(y) \ b \ \text{Rel}(x) \ \text{Rel}(y) \)

P2: \( \text{R}(y) \ \text{NIL} \ \text{Acq}(x) \ \text{R}(x) \ a \)

P3: \( \text{Acq}(y) \)
## Summary of Consistency Models

Models Using Synchronization Operations

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Shared data can be counted on to be consistent only after a synchronization is done</td>
</tr>
<tr>
<td>Release</td>
<td>Shared data are made consistent when a critical region is exited</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered.</td>
</tr>
</tbody>
</table>