Today's Topics - Distributed Shared Memory

Slide 1

- The Shared Memory Abstraction, why?
- Approaches to implementation;
- Consistency Models:
  - Strict Consistency;
  - Sequential Consistency
  - Release Consistency;
  - Causal Consistency;
- Various implementations of shared memory.

Reading all of Chapter 16.

The Shared Memory Abstraction

Slide 2

- To program a distributed system you need to send messages.
- Programmers are not used to writing programs via messages it is more natural for most people to think in terms of process sharing a common address space.
- It is possible to implement shared memory relatively efficiently compared with the equivalent message passing program.

Insert Figure 16.1

Shared Memory - Problems

Slide 3

Remember the operating systems course where you had shared memory between processes and you had problems when two processes running in parallel might not be updated across context switches:

`x:=1;`
`....`
`x := 0;`
`....`
	`....`
	`if(x == 0) { dosomething; }`
	`....`

This was solved with semaphores and atomic actions.

In a distributed environment things will get even more complicated.

Implementation Approaches to DSM

Slide 4

- *Hardware* Shared-memory multiprocessor architectures. Problem to solve Cache consistency, manages memory access over a shared high speed bus.
- *Paged Virtual Memory* Collection of homogeneous computers, each has a page in virtual memory that is shared, by modifying the VM system shared memory can be simulated.
- *Middleware* Implement the shared memory abstraction. Process make calls to the middleware when it wants to access shared memory. Not as transparent as the above two approaches but has the advantage of allowing you view shared data at a higher level of abstraction.

Some Middleware Approaches

We will loook at the middleware approach.

Slide 5

- *Byte-Oriented* Shared memory seen as a traditional address space. Read and write bytes.
- *Object-oriented* The shared memory is structured as a collection of language-level objects. Languages such as Orca impose extra consistency semantics on the objects.
- *Immutable Data* DSM is a shared collection of data that can be written to and read by everybody. When an item of data is written it can not be changed. Linda has the idea of a tuple space where a tuple is a sequence of one or more data items e.g. `<"fred",1958>,<"sid",1964>`. Although tuples can not be changed they can be removed.

Byte-Oriented - Abstraction

Slide 6

- We concentrate on byte-oriented shared memories. It is conceptually the easiest to deal with.
- Each process make reads and write to the central store. We will use the following notation:
  - $R(x)a$ - a *read* operation reading a value a from location x.
  - $W(x)a$ - a *write* operation writing a value a to location x.

Consistency Models

A consistency model is a specification of how the system will act.

There are two extremes:

Slide 7

- . No consistency what soever. When you read a value it could be any value previously written. This is possible because of arbitrary network delays.

P1 :	$W(x)a$	$W(x)b$			
P2 :			$W(x)c$		
P3 :				$R(x)b$	$R(x)b$
P4 :		$R(x)a$			$R(x)c$

- Have no idea what value might be read back after one is written will make programming very hard.

Strict Consistency

Slide 8

- Any read on a data item x returns a value corresponding to the result of the most recent write on x.

P1 :	$W(x)a$	$W(x)b$			
P2 :			$W(x)c$		
P3 :				$R(x)c$	$R(x)c$
P4 :		$R(x)b$			$R(x)c$

Problems with Strict Consistency

Slide 9

- The problem with clocks that we talked about in earlier lectures means that it is hard to get an idea what the most recent write might be.
- Further with unbounded network delays it is impossible to implement this.

So we look for weaker models of consistency.

Sequential Consistency

Slide 10

Sequential Consistency is the strongest version of consistency used. We will give two equivalent definition.

> The result of any execution is the same as if the (read and write) operations by all the processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its programs.

Sequential Consistency - More precise definition

Slide 11

A DSM is said to be sequentially consistent if *for any execution* there is some interleaving of the series of operations issued by the all the processes that satisfies the following two criteria:

SC1 The Interleaved sequence of operations is such that if $R(x)a$ occurs in the sequence, then the last write operation that occurs before it in the interleaved sequence is $W(x)a$.

SC2 The order of operations in the interleaving is consistent with the program order in which each individual client executed them.

Essentially it means that each process sees the same sequence of writes.

Sequential Consistency - Examples

Slide 12

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)b	R(x)a

This is sequentially consistent. But

P1:	W(x)a				
P2:		W(x)b			
P3:			R(x)b		R(x)a
P4:				R(x)a	R(x)b

Is not because P3 and P4 see a different order of writes.

Implementing Sequential Consistency

Slide 13

- Sequential consistency can be implemented by a central server that holds all the data and orders the reads and write.
- In Practise it is too costly to implement in a real distributed system.
- So we must go for weaker forms of consistency.

Causal Consistency

- If $W(x)a \rightarrow W(y)b$ then all process must see $W(x)a$ before it sees $W(y)b$.

Slide 14

P1 : W(x)a		W(x)c		
P2 :	R(x)a	W(x)b		
P3:	R(x)a		R(x)c	R(x)b
P4:	R(x)a		R(x)b	R(x)c

Can be implemented using Lamport Timestamps to keep track of who has seen what.

Coherence based Consistency

Slide 15

- Split the data up and only require reads and writes are sequentially consistent on individual locations.
- Can be implemented with a server for each object.

Weak Consistency

Slide 16

- One of the ideas of weak consistency is to try to reduce the overhead by giving some of the burden to the programmer.
- As with semaphores the basic idea is to aquire and release common bits of storage.
- One uses locks or synchronisation variables. A process acquires a lock when it unlocks it and another process acquires that lock that process will get the latest values written.

Weak Consistency

Slide 17

1. Access to synchronisation variables associated with a data store are sequentially consistent.
2. No operation on a synchronisation variable is allowed to be performed until all previous writes have been completed everywhere.
3. No read or write operations on data items are allowed to be performed until all previous operations to synchronisation variables have been performed.

Weak Consistency

Slide 18

P1:	W(x)a	W(x)b		S		
P2:			R(x)a			
P3:					S	R(x)b

Notice that P2 does not aquire the lock and hence has no does not necessarily get the latest value of x.

Release Consistency

Slide 19

Weak consistency has the problem that when a synchronisation variable is accessed, the data store does not know whether this is being done because the process is either finished writing the shared data or is otherwise about to start reading data.

Consequently it must take the actions required in both cases, namely making sure that all locally initiated writes have been completed as well as gathering all writes from other copies.

Release Consistency

Slide 20

1. Before a read or write operation on shared data is performed, all previous acquires done by its process must have completed successfully;
2. Before a release is allowed to be performed, all previous reads and writes done by the process must have been completed;
3. Access to synchronisation variables are sequentially consistent (sometimes FIFO) with respect to each other.

Release Consistency

Slide 21

P1:	Acq(L)	W(x)a	W(x)b	Rel(L)		
P2:					Acq(L)	Rx(b) Rel(L)

Release Consistency Conceptual Implementation

Slide 22

- Central server, grants the lock to one process at a time.
- When a process acquires the locks works on a local copy of the data.
- When it releases the data local copies are sent back to the central server.
- In a sense it is no different from sequential consistency except that groups of operations are made atomic by the release/acquire blocks.

Summary

Slide 23

- Shared memory can be implemented in middleware.
- It provides a good abstraction for programmers who don't want to use messages.
- Consistency is a problem, the more consistency you require the larger the overhead.
- Look at the book for implementations of shared memory.