Distributed Programming

Text Book: Distributed Programming Theory and Practice by A.Uday Shankar

Rahul R.Pitale
Department of Computer Engineering
PCCOE

Content

- Introduction
- **Simple Lock**
- **Bounded Buffer**
- **Message Passing Services**
- Distributed Lock Service
  - Distributed Lock Using Timestamp
- Object Transfer Service
  - Object Transfer using path reversal
- Distributed Shared Memory Service
  - Single copy and multi copy distributed shared memory

Introduction

- **Program**: It is set of Instructions
- **Intructions**:Set of Commands
- **Process**: It is instance of computer program
- **Thread**: Is a lightweight process or active entity which execute program

- Single Threaded
  - One Thread is executing at any time
- Multi – Threaded
  - Multiple thread executing it simultaneously interacting with each other while executing

Issues

- So many request at same time
- High demand of users in both processing power and data storage
- For Example:
  - Facebook : 1.44 Billion Monthly Active Users
  - Active Servers : 180,000
  - 100+ Milions Operations per second
  - 50 Million photos are viewed by every second
  - 43 Billions photos are uploaded each month

Distributed Systems

- Moore's Law : The Processing power of CPU gains double for every 18 Months
- A Distributed System is a collection of independent computers that appears to its users as a single coherent system

Example

- Social Networking (FB)
- Distributed File System (Hadoop)
- Google Web Server
- Grid Computing

Why Distributed Systems

- Resource Sharing
- Computation Speedup
- Reliablity
- Communication

Distributed Programming Lang

- Scala
- Bloom
- Julia
- Limbo
- Java
- D Programming
- R Programming
- Jcuda,Ccuda,PyCuda etc.
- Cython,Jython,etc..

Distributed Programming

- Made up of Multiple Program
- It is one form of parallel programming
- Program split into parts of program which run on different computing units (Process)
- Ex : One part is running on Intel and other is running on AMD Processor

Distributed Program

- Is is a set of Instruction
- It has main code and function
- A Program starts when it is instantiated
- Thread starts executing systems main code
- Thread leaves the system when it reaches the end of main code
- But System remains in existence until it is explicitly terminated

Example

- First Approach

```
Program A()
{
---------
function mod (b,c)
{
return(d)
}
}
```

  - **Here A is a program**
  - **Mod is a function hving two parameters a & b**
  - **Return the reminder of b divided by c**
  - **I/p – instantantion and mod (b,c)**
  - **O/p- return of this call**

- **Second Approach**

```
Program Y()
{
instantiate Z()
function gcd(u,v)
{
r<- mod(a,b)
return(w)
}
}
```

```
Program Z()
{
 -----
function mod (p,q)
{
    ----
}
}
```

- Y Program has function gcd
- Having two parameters and return gcd
- Y makes use of program z to help in execution
- Y instantitates Z and gcd repatdly calls the z system mod function

Y

- Here one thread is active at any time
- When a system calls the function of another system
- Calling System becomes Idle after making the call
- And Resume the execution only after call returns

- **Third Approach**

```
Program Lock()
{
---
function acq ()
{
return
}
function rel ()
{
return
}
}
```

- It has two functions
- acq and rel
- Suppose there are n no of users
- So different users can call acq concurrently
- Means user request the lock when it calls acq and acquires the lock when the call return
- Lock return the call only when lock is not currently held by user – blocking
- A release call is non-blocking

- **Fourth Approach**

Program	Program	Program	Program
Prodcons()	Producer()	Lock()	Consumer()
{	{	{	{
}	}	}	}

- ProdCons starts three other Systems
- Producer puts item into customer
- Bothe producer and consumer make use of lock to synchronize their activities
- If consumer system is faster it may obtain the lock several times
- If producer system is faster it will acuire lock before the lock system has been created

Simple Lock Program (pg no 43)

- Three Inputs : acq,rel,end
- In acq and rel – requires calling thread tid,mytid
- Now thread i calls acq
- It sets xreq[i] true
- And busy waits until xreq[i] false
- And return

- Thread i calls rel
- It sets xacq to false and returns
- Function end can be called by any thread
- Now It executes **endSystem** initiating termination of the lock system

- mysid.acq – allows calling thread to already hv lock
- mysid.rel – allows calling thread to not have lock

Bounded Buffer

- It is also called as Producer Consumer Problem
- It shows that how two process can store a common storage space

Algorithms using Two Variables

- Producer
  - Producer A
  - {
  - Produce an item
  - Place item in buffer
  - Increment pointer
  - }
- Consumer
  - Consumer A
  - {
  - Take item from buffer
  - Increment pointer
  - Consume item
  - }

Bounded Buffer Service

- Bounded Buffer Service Program (pg no 65)
  - N is the size of buffer
  - It has three input functions
    - put(x) – to append item x to buffer
    - get() - to remove an item from buffer and return it.
    - end() - to stop calls of put and get
  - buff – is the buffer
  - ending variable is true if end has been called
  - Variable putBusy(getBusy) is true iff a put(get) call is ongoing

- Bounded Buffer Service Inverse Program (pg no 67)
  - Additional parameter is used **bb**
  - The input function mysid.put(x) in service program becomes an output function doPut(x),it calls bb.put(x)
  - Input function mysid.get() in the service program becomes doGet(x),it calles bb.get()
  - Input function mysid.end() in the service program becomes and output function doEnd(),it calles bb.end()

- Function put(x) -
  - it is called whenever end has not been called and no put call is ongoing
  - It append x to buffer and return only if buff has space for x
- Function get()
  - It is called whenever end has not been called and no get call is ongoing
  - It removes the item from the head of buff and returns it only if buff is not empty,otherwise it waits
- Function end()
  - It is called only if it has not been called already

- Bounded Buffer Implementation Using Awaits (pg no 69)
  - Algorithm
    - Initally empty
    - Now wait for non-full buffer
      - await(buff.size<N)
    - Add item to buffer
      - buff.append(item)
    - Wait for non-empty buffer
      - await(buff.size>0)
    - Get item from buffer and assign to x

- Implementing awaits with locks and condition variables (pg no 73)
  - Two Condition variables cvspace and cvItem
  - cvspace block a thread in a put if buffer is full
  - cvItem block a thread in get if the buffer is empty

- Implementation Using Semaphores
  - Three Sempaphores
  - mutex
    - It used lyk a lock
    - Means at most one thread can access the thread at any time
  - nitem
    - It is counting semaphore that tracks the number of items in the buffer and block the consumer if buffer is empty
  - nspace
    - It tracks the number of spaces in the buffer and block the producer if the buffer is full

Message Passing Services

- It is also called as channels
- A channels allows a user at one location to transmit a message to be received by a user at another location
- Channel is a service that is spread over different locations
- Users access the service via multiple systems
- A channel has set of addresses (Mac,IP,URL,etc)
- A channel can be connection-less and connection Oriented

- Connection-less
  - User can send and recieve messages without any prior intimation to service
- Connection Oriented
  - User can send and recieve messages to a remote user only after a connection to the remote user is established

Types of Channels

- Connection-less Channels
  - FIFO (First in First Out)(program pg no 93)
    - It transfer the msg without loss between any two address
  - Lossy(It lose messages in transit) (pg no 98)
    - It is a fifo channel excepts msgs can be lost
  - LRD (lose,reorder,duplicate msg in transit) (pg no 99)
    - Any msg sent in the past can be (again) receievd
    - Sequence of msg (sent and recv from anywhere)
- Connection – Oriented Channels
  - Only Basic form of connection establishment and termination (pg no 101 and 102)

Connection Oriented Channel

- Connection management and data transfer within connections
- Addr j
  - **closed** : inactive
  - **accepting** – waiting for any remote connect request
  - **opning** – waiting for response to local connect request
  - **open** – connected to remote address
- Input Functions
  - **j.accept()** - closed -> accepting->open //server
  - **j.connect()** - closed ->opning->open/closed // client
  - **j.close()** - open->closing->closed
  - **j.tx(msg)** – only while open
  - **j.rx()** - only while open

Distributed Lock Service

- It is a lock whose users may be spread over different locations
- Ex: users at different computers of network
- At each location there is an access system through which users access the lock

Service Program (pg no 226)

- A service has one parameter
- The set of addresses of the locations at which the lock can be accesses
- At each address j,the program has an access system with two input functions i.e acq() and rel()
-

Explanation of Service Program

- Parameter ADDR is the set of addresses of locations at which lock can be accessed
- Variable eating is null if no one has the lock
- Otherwise it stores the tid of the user
- Ex – Tid eating <- null
- Variable users is a map indexed by addresses such that users[j], for address j, is the set of tids of users currently accessing the service at j.
- i.e users at j that are waiting for the lock and eating user that acquired the lock j
- Variable v is a map from addresses to sids such that v[j] stores the sid of the access system at address j.

- Each access system v[j] has two input functions
- i.e v[j].acq and v[j].rel
- v[j].acq can be called only by non-eaing user
- v[j].rel can be called only by eating user that acquired the lock

Distributed Lock Using Timestamps

- Timestamps : is used for synchronization
- Ex: You receive a letter only after i send it
- Lamport Timestamp
  - Used in all Distributed System
- Each system has integer variable and ID
- When system executes a staatement that is ordered,it increses its clock and send to all other system
- Msg containing the clock value and its tid
- Clock value is called Timestamp.
- When a system recieves the msg it increases its clock to a value higher than msg timestamp.

Request Scheduling Problem and Solution using Timestamp

- Request Scheduling Problem
  - Consider a collection of system attached to a fifo channel
  - Users issue requests to the system
  - Each request is to be served by local system
  - Request can conflict
  - Conflicting requests should not be served simultaneously.
  - Now we want mechanism that informs each system when to serve a request issued by its user.

Solution using Timestamp (pg no 233)

- Each system j maintains the some variables lyk
  - clk- timestamp clock, initially 0
  - rts-map over address other than j, rts[k[stores highest timestamp recieved from k,initailly 0
  - req-set containing [x,t,k]for every request x with extended timestamp[t,k] that system j has recieved or genrated.
  - Now the system exchange request,release and ack msges
  - Request msg has form [REQ,x,t,k]
  - REQ-Constant,x – request,t- timestamp,k-senders address
  - Release msg has form [REL,x,t,k]
  - Ack msg has[ACK,t,k]

- Distributed Lock Program (Pg no 234)
  - For request scheduling solution for the case where every two request conflict
  - It has one paramaeter
  - Same as dining philospher problem
  - System j is thinking,hungry or eating
  - It becomes hungry when it introduces a request into timestamp
  - Eating when it starts to serve the request
  - Thinking when it finishes serving the reqeust

- When system j becomes hungry it increment its clock
- Send request msg to every other system
- System j starts eating when the extended timestamp [req[j],j]
- When system j stops eating, it send a release msg to other system
- When system j receieves request msg it updates clk and send ack msg.
- When system j recieves a release msg it removes entry for k from req.

Object Transfer Service

- Consist of set of object.
- Characterzied by id and mutable value.
- A system can aquire an object ,make changes to its value and release the object.
- When a system aquires an object its value should be what it was at its release.
- So Object is just like a token.

- Access System has three input functions.
- acq(oid) – to acuire oid and get its value.
- rel(oid.oval) – to release object oid with oval.
- rxReq() - to recieve a request for an object.

Service Program (pg no 271)

- It has four parameters
- ADDR,set of addresses of the service,OID,set of ids of the object, OVAL,the set of ids of objects and initObjs
- Now Variable Obj is map over ADDR such that obj[j] is the set of objects held by j's user
- Variable req is map over ADDR such that reqs[j] is the set of objects for which j's user has received request.
- Variable val is a map over OID such that oid is in val.keys

- The Access System v[j] has three input functions
- v[j].acq(oid),v[j].rel(oid.oval) and v[j].rxReq()
- V[j].acq(oid) is called to acquire object oid
- v[j].rel(oid.oval) is called to release object oid with value oval to service
- v[j].rxReq() is called to receive a request for an object that is either owned or being requested by j's user.

Object Transfer Using Path Reversal

- It is used distributed program that implements the object-transfer service over fifo channel
- Each system maintains for each object a LAST POINTER that is either nil or points to another system.
- The path of last pointers leading out of any system ends at the system holding the object.
- To acquire the object a system j sends a request that gets forwarded along the last pointer path leading out of j it reaches the system with the object.

- In distributed program if at most one request is in transit at any time.
- Then the sequence of last pointer redirections results in a path reversal in the in-tree of last pointer
- The amortized cost of path reversal is logN msg send.

Distributed Path-Reversal Program (pg no 278)

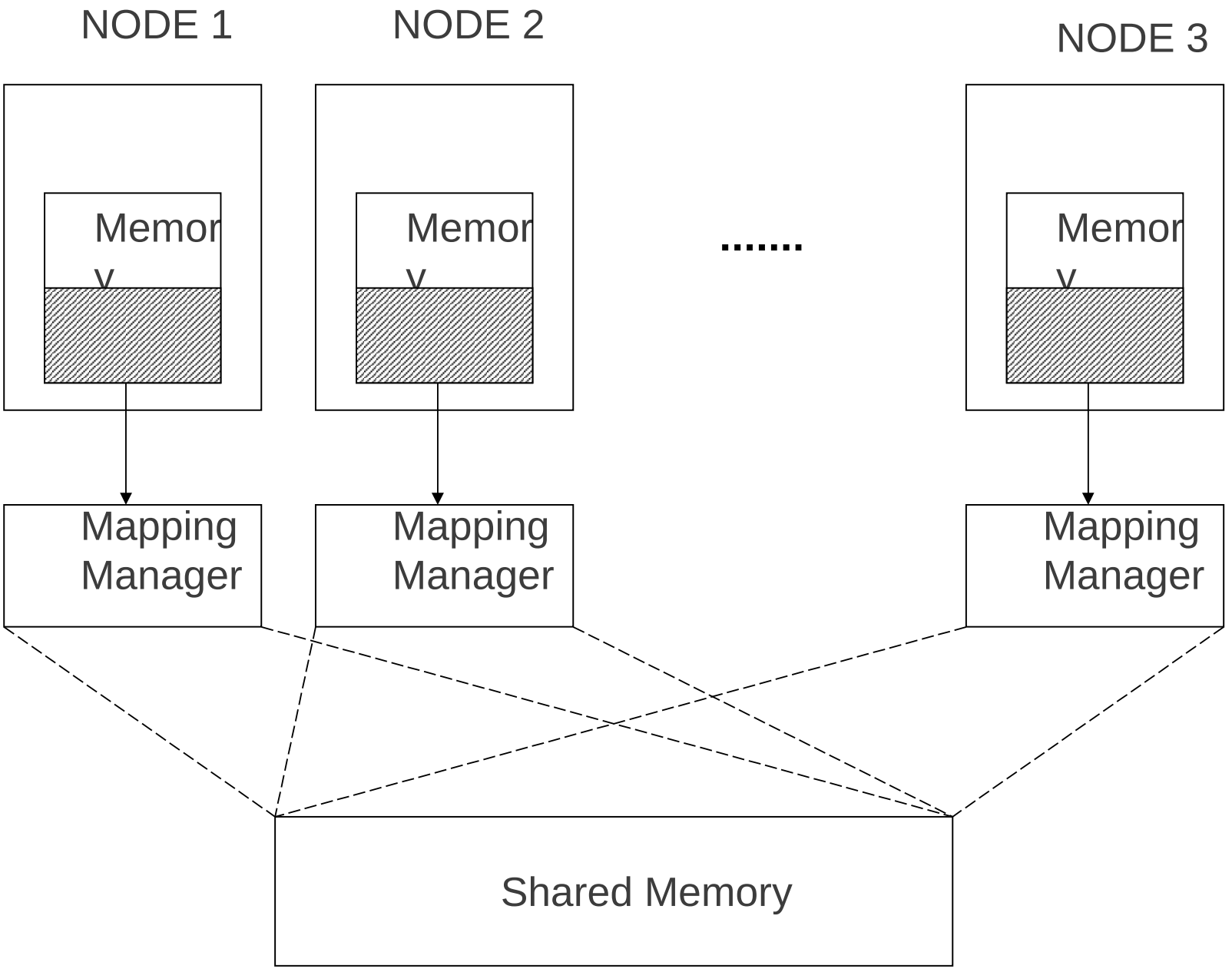
- It has two parameters: ADDR, the addresses of service and a0,the initial address of the object.
- Now It strats fifo channel with addresses ADDR.
- Then starts a componenent system at each address
- The system exchange request msg and object carrying msg
- A request msg is a tuple – [REQ,j]
- Object carring msg is a tuple – [OBJ]
- A system is eating if it has the object.
- Hungry if it has ongoing request for object.
- Otherwise thinking

- Each system j has variable 1st which is its last pointer and variable nxt reffred to as next pointer
- Each ranges over address an a nil value different from any address.
- Variable 1st equals the address in the last request msg recieved by system j after it last becom hungry
- It is nil if no such request msg has been recieved.
- Variabl nxt is nil if system j thinking or it it is non-thinkingand has not recieved a request msg since it last became non-thinking.
- Nxt address in the first request msg received by system j after becoming non-thinking.

Distributed Shared Memory Service

- The component system of a distributed system can intract by message passing or by shared memory.
- It has read and write shared meory locations
- Distributed System uses msg passing to implement a shared memory accessible to component system on different computers.
- The memory address space is divided into pages.
- And pages are allocated among the component system.
- When a component system attempt access a page that is not locally present the ditributed shared meory implementation brings the page to component system.

Distributed shared memory (Cont.)



Types of Distributed Shared Memory

- Single copy distributed Shared Memory
- Multicopy Distributed Shared Memory