CS 499 – Distributed Systems  
Friday, December 04, 2009

Final Exam review:

* Logical Clocks (11)
  + Vector (12)
  + normal Lamport clocks
  + example of applications
* Global State (13)
  + Terminology
    - cuts
    - consistent cuts binary
    - Diagram!!!
  + Snapshot algorithm
    - limitations
* Concurrency Control (16)
  + serial equivalence & how achieve
  + 2-phase locking
  + strict 2 phase locking
  + strict execution
    - how applies to Optimistic Concurrency Control
  + Deadlocks
    - philosophy
    - how to resolve
    - Etc…
* Coordination and agreement & questions (8 questions) (21)
  + Mutual Exclusion
  + Central Server Algorithm
    - in context of Mutual Exclusion conditions
  + overall picture of Election Algorithm
    - What process are we really choosing?
      * Bully Algorithm – form quizzes
  + Consensus Question - form quizzes
    - in context of the internet
* Multicast Communication (three questions) ( 28)
  + ordering principles
    - comparing
    - understanding
      * using the example from the bulletin board
* Grad Project Questions
  + //slide presentations should be online
  + one question for each topic area
  + two questions for grad students

CS 499 – Distributed Systems  
Monday, August 24, 2009

Course Website: http://flagstaff.cse.nau.edu:25176/teaching/  
>> CS 565  
Instructor: Dr. Wolf Dieter Otte

Notes:

* Distributed systems – when computers rely upon another one in sync to function.
* This course is meant for Graduate Course Level. They tend to have more rigors. Graduates have to perform some services in teaching. This is why the course will be taught in a more strict way.
* Class Lectures on Monday and Wednesday, the topics seem to be focused on the theories of distributed systems. Friday will have more practical works and discussion about practical work.
  + Dr. Otte will conduct about three projects during the course of the class. The first project will be to design and implement a P2P system, a system that sends a message to all the others in a system.
  + The graduates will become leaders of undergraduates to create a team to complete the Projects.
* Topics: Research projects that will require teams to learn and present.
* Grading
  + Professional Paper 🡪 20%
  + Class presentation 🡪 15%
  + Team Projects 🡪 25%
  + Quizzes 🡪 10%
    - At least a quiz once a week
  + Midterm 🡪 10%
  + Final Exam 🡪 20%
* The Graduates are suppose to take more of an active lead in the class and are resource for the Undergraduates; they are teaching us.
* Q: Was there a pre-defined programming language set for the course?
  + No, there is no restriction for the projects on programming languages.
* Q: Can we be given virtual machines on your server for our project needs (sandboxing)?
  + Maybe. Dr. Otte will get back to us on if we will be giving us VM’s.
* Rough outline of the course can be found at the course website.
* Course Outline:
  + Introduction to distributed systems (1 week)
  + system models (2 weeks)
  + time and global states (3 weeks)
  + coordination and agreement (1 week)
  + transactions and concurrency control (3 weeks)
  + replication (3 weeks)
  + load balancing/distribution (1 week)
  + p2p networks (1 week)

TODO:

* Download/Print Syllabus
* Choose teams – composed of one Graduate student and three to four Undergraduates.
  + Team:
    - Graduate: Ryan Middleton, rmm34@nau.edu, 928-221-7210
    - Undergraduates:
      * Brian, bjc76 [at] nau.edu, 614-425-6054
      * Talbert, talbert.tso [at] gmail.com, 928-699-8755
      * Andy Arminio, aga7 [at] nau.edu, 928-707-0095
      * Bernie Citron, bc229 [at] nau.edu, 480-329-6054
  + First Assignment: P2P chat program
    - As of right now we will code in Java.
    - Team agreed to use subversion for our communication services.
* Review and update yourself with SOCKET programming.

CS 499 – Distributed Systems  
Wednesday, August 26, 2009

Notes:

* go over the boring stuff: introductory stuff today
* Intro to Distributed Systems (DS):
  + What is a DS?
    - “A distributed system is a collection of independent which give the user the impression, that he is working with one computer”
  + Definition by Mullender
    - “A distributed system is various computers doing something together. Thus it is characterized by three things:
      * a number of independent computers, each consisting of CPU…i/o
      * Some of the i/o are used for communications between c computers 🡪 communication network. if the computers cannot communicate with each other, this distributed system is not very interesting
      * The computers share a common state. if state is thought of a range of global invariants, the maintenance of these invariants require the common/coordinated cooperation of the computers.”
  + Sun Def:
    - “the network is the computer”
  + Coulouris Def:
    - “We define a distributed system as one in which hardware or software components located at networked computers communicate and coordinate their acions only by passing messages.”
  + Class Def:
    - No common physical clock
    - No shared memory
      * // concurrency
    - geographic separation
    - Autonomy, heterogeneity
* Motivation for DS:
  + Problems are too big for one to handle
    - >> Load distribution, **Performance boost**
    - On one computer you can get stuck waiting for one process to finish. once that process finishes it allows other processes to continue normally.
    - Performance boost 🡪 coarsely grained programs
  + Bridge distances
    - resource access
    - // Hackers 🡪 “use resources that are not their”
  + Reliability
  + Scalability
    - “upgradability”
* // Inherently DS are not good!
* Parallel systems are close to DS, therefore we need to define the two to see the difference they share:
  + Multiprocessor Systems (UMA – Uniform Memory Access)
    - Direct access to shared memory
    - connected by bus (commonly used)
    - switches (two by two switches) 🡪 Omega network and Butterfly network
      * Multi-switch
  + Multi-computer parallel system (NUMA -
    - no direct access to main memory
      * but that is no fully limiting its access
    - each sections have their own memory
    - commonly don’t have clock
  + Array processors
    - tightly copulated
  + Relation between SW components
    - Image (look for)
* Classifications of Primitives of DS:
  + Synchronous (send/receive)
    - Handshake between sender and receiver!
  + Asynchronous (send)
    - control network after the data to the buffer
  + Blocking (send/receive)
    - control returns after processing of primitives
  + Non-blocking (return right away)
    - control returns to process immediately
    - check way handle
    - wait (handle)
    - // Blocking can be synchronous and asynchronous (only for sending) 🡪 research at home
* Challenges to DS:
  + Transparency “there but not seen”
    - Access Trans., location Trans. >> Network Trans.
      * don’t care where the resources are, you just care of getting it (i.e., mirrors)
    - Migration Trans.
      * move things around without the user knowing anything (no consequence to the user)
    - Replication Trans.
      * duplicating a file without seeing the process
    - Concurrency Trans.
      * keeping data synchronized without user input
    - Failure Trans.
      * when something fails and something takes over and the user is unaware of the issue
    - Performance Trans.
      * the load changes and the performance is still good
    - Mobility Trans. (close to Migration)
      * Close to access trans.
      * but deal with mobile devices; laptop, wireless devices
    - Scaling Trans.
      * if you add more to the system the system does not fail
  + // Visible (there and seen), Virtual (not there but seen)

TODO:

* THIS FRIDAY: Present our conceptual design to the project: Peer to peer
  + network with the assumption that one peer is always there (any peer)
  + peers joining, leaving, connecting to another
  + free to use any architecture
* FRIDAY AFTER: Documentation
* FRIDAY AFTER THAT: Checkpoint

CS 499 – Distributed Systems  
Wednesday, September 02, 2009

Presentation:

Joe Flieger gave a quick presentation of his design.  
 Provides for chat rooms and connection to a larger network

Lecture Notes:

* in depth discussion on communication primitives
* message primitives could be seen from two different perspectives
  + blocking
  + non-blocking
  + synchronous
  + asynchronous
* First picture
  + sender / receiver
  + four scenarios
  + blocking – def – do you wait till sending complete to terminate connection
    - else you initially send then move on to a new process
  + Synchronous – def- waiting for the receiver to send back a signal that they got the message
    - else you continue to different process without waiting for confirmation
  + // Since most of the class is behind in course involvement we will postpone this lecture
* Challenges: System Perspective
  + API for communications
    - “ease of use”
  + Synchronization
    - how do you sync a world that does not have a singular world clock?
  + Data storage and Access
    - where do you host it
    - how do you make it available
    - how do you maintain it
  + Consistency and Replication
    - how do you org replication
    - how to keep replica consistent
      * do you need to keep it consistent
    - difference between a strict and loose system
  + Fault Tolerance
    - how are you going to organize fault tolerance
    - how do you organization fault tolerance in reality of nodes failing and links failing
  + Scalability and Modularity
    - how do you organize your system so it is scalable easily
    - is the user aware of shortages or not
    - higher load 🡪 add more resources to maintain the same
  + Naming
    - how do you name resources so they are readable, discoverable, searchable
  + Processes
    - code migration
    - mobile agents
    - software that moves from machine to machine
    - A process is a container that …
  + There will be more challenges that will either be discussed later or are there for your own independent research
* A Model for Distributed Computations
  + “state”
  + What is a distributed program?
    - Def: composed of a set of an asynchronous processes (p1, p2, …, pN)
  + Process execution and message transfers are asynchronous
    - notation:
      * Cij 🡪 channel from Pi to Pj
      * Nij 🡪 message from Pi to Pj
    - message transmissions delay are finite but unpredictable
* A Model of Distributed Execution
  + “distributed program”
  + a processes can consist of actions (in the end)
    - “a process can just execute some actions”
    - processes in the classical UNIX sense of the work
  + Actions are atomic and indivisible
    - “atomic and indivisible” 🡪 bank transactions
    - “atomic and divisible” 🡪 database management system
    - modeled as three types of events
      * interval events
      * sent message events
      * receive message events

Side Notes:

* Majority of students have never had CS 460 before
* the Friday after next, students that know about sockets should not come to this class
* then after that we will discuss more about socket programming

CS 499 – Distributed Systems  
Wednesday, September 09, 2009

Class notes:

* This Friday will be a workshop in a socket programming
  + look into the programming codes and review before this Friday
  + this is more of a crash course in getting
* For Monday, we will begin Quizzes
  + we will start a series of quizzes that will test us on book and lecture material

Lecture Notes:

* A model of Distributed Systems
  + Events
    - Events can be internal event, message sent event, message received event
    - exi 🡪 event x on process Pi
    - for message sent(m), receive(m)
    - events change the state of a DS
      * internal events changes state of process
      * send event 🡪 process sending message, channel
      * receive event 🡪 process receiving message, channel
    - Execution of process Pi 🡪 e1i, e2i, e3i, …, eni 
      * something is happening when you send a message
    - History Hi = (hi, 🡪 i)
      * hi is the set of events produced by Pi
      * “🡪” is the binary relationship that orders the events
    - Send/Receive Events
      * flow of information between processes
        + create a causal dependency between sender and receiver

“🡪msg” is the binary relationship

* + - Space-Time Diagram
      * Look to figure in book; three lines representing time. Each line represent a process and events are dots on the graph.
      * relationships between events are not necessarily related
    - Causal Precedence
      * distributed application 🡪 results in a set of distributed events
      * H = Uibi 🡪 “Binary relationship on history”
        + (H)istory, (U)nion
        + for all exi, forall eyj are elements of H, exi 🡪 eyj ⬄ Set{

exi 🡪 ieyj,

* + - * for any two events ei and ej we say ei is unrelated to ej; no causal dependency between the two
      * For any two events ei and ej,
        + ei -/-> ej =/=> ej -/-> ei
        + e.g. e22-/-> e21 =/=> e21 -/-> e22
        + ei 🡪 ej => ej -/-> ei
    - Concurrent Events
      * If ei -/-> ej & ej -/-> ei
      * then => ei || ej

CS 499 – Distributed Systems  
Friday, September 11, 2009

Notes:

* Socket Programming
  + Server Side
    - ServerSocket( portNumber );
    - accept ()
      * returns
      * socket() //connection is established and we now have an object
        + getInputStream()
        + getOutputStream()
    - // Peer
      * listening for request
      * does have a server socket as well
  + Client Socket
    - uses the same socket object
    - socket ( IP, port )
  + Port number on the server side does not change
    - packets have port 80
    - two tuples are sent by the client to ensure delivery
      * IP of Server
      * Port of the Server
      * IP of the Client
      * Port of the Client
  + Synchronization
    - its best to synchronize methods to protect your data
    - synchronize does not allow multiple threads to access the same method at the same time
  + Cubby Hole Example
    - Producers produce data into the cubby hole
    - Consumers uses the data
    - if a consumers occupies the cubby hole with no data to use, then the producer cannot place the data the consumer needs.
    - so the consumers is asked to wait while the producer supplies the data
    - After the Producer is done producing data, the Producer “notifies” the worker thread to wake up.

CS 499 – Distributed Systems  
Monday, September 14, 2009

Class Notes:

* Last time we talked about Causal Relationships
  + model for the execution of a distributed systems
  + events can be whatever you want them to be
    - single machine instructions
    - multiple machine instructions
  + Last time we pinned down what concurrency is
* Quiz 1
  + Question 1:
    - Answer: example: e11 || e12
  + Question 2
    - Answer: no, only if ei <-> ej
  + Question 3
    - Answer: example: e11 , e31

Lecture Notes:

* What communication channels do we have?
* Model of DS
  + Models of Communication Networks
    - FIFO, Causal Ordering (CO), non-FIFO
      * FIFO – first in first out
      * non-FIFO
        + acts like a set and receiver, no succession is implied
        + just picks out messages from the set
      * Causal Ordering
        + two messages: m(i, j)& m(k, j) iff

send( m(i, j) 🡪 send( m(k, j)) =>

rec( m(i, j)) 🡪 rec( m(k, j))

* Time
  + In our minds we think of time in a physical sense, but this is pretty much useless
  + So we need to break this mentality and go more for Logical Time
  + but we will cover physical time because it is still used in the real world
  + Physical Time:
    - What is it?
      * depended on pulses
      * need to scale these pulses
      * example: Hardware clock
        + alpha \* Hi(t) + beta = Ci(t)

H – Hardware Clock

alpha – speed(?)

beta – unknown

* + - Example:
      * satellites in space have to have a slower tick
  + There exists Drift and Drift rate
  + We need to have some correctness
    - => make assumption concerning some bound D.
    - Montinicity Requirement
  + Logical Time
    - break stuff into events

CS 499 – Distributed Systems  
Monday, September 21, 2009

Class Notes:

* Project 2 is due this Friday
* Project 1 Documentation is due this Friday
* Will need another “static” site. Basically another server.
  + Brian will use the ACM Server for the course jobs.
* QUIZ Wednesday

Lecture Notes:

* With an example of two Servers running. (Server A and Server B)
  + o: Tb = Ta + o

Server A

Server B

Ti-3

Ti-2

Ti-3/t

Ti-1

t’

Ti

Ti

* + Ti-2 = Ti-3 + o + t
  + Ti = Ti-1 + t’ – o
  + 🡺
  + di = a + b = t + t’
  + a = Ti-2 – Ti-3
  + b = Ti – Ti-1
  + o = a – t
  + o = t’ – b
  + 2o = a – b – t + t’
  + o = (a - b)/2 + (t’ – t)/2
  + o = oi + (t’- t)/2
    - oi 🡪 estimate for o
  + 🡺 Zero in a totally symmetric exchange
  + oi – di/2 <= 0 <= oi + di/2
    - -di/2 🡪 (-t-t’)/2
    - +di/2 🡪 (t+t’)/2
  + // What this algorithm allows for is pairs
    - <oi, di>
    - // in reality you would create a series of pairs and choose the lowest one.

Ti-2 = Ti-3 + t + o and Ti = Ti-1 + t’ – o

di = t + t’ = Ti-2 – Ti-3 + Ti – Ti-1

o = oi + (t’ – t)/2 where oi = (Ti-2 – Ti-3 + Ti-1 – Ti)/2

* Logical Clocks
  + // Physical time for us doesn’t really matter
    - The reason for this is that we are concerned with ordering and …
  + An alternative to physical clocks
  + Happened before relationship
    - order of events on a process as observed
    - Sending happens before Receiving
    - HB is transitional
      * e1 🡪 e2 & e2 🡪 e3
        + e1 🡺 e3
  + Lamport’s Logical Clock
    - // General Def: all that a logical clock is, is an integer counter that only increase
      * Software counter, only incrementing
      * Processes have their own clock
      * used to timestamp events
        + How do you do that?

you start somewhere

you use timestamps to move a time on a separate process when you are switching between processes

* + - * + What does this by us?

example:

e 🡪 e’ 🡺L(e) < L(e’)

The reverse is not true!

* + Vector Timestamps
    - What do you expect?
      * multiple elements to the timestamps
      * as many components as processes
    - What do you win?
      * overcomes the drawback (the shortcoming) of Lamport’s timestamps;
      * L(e) < L(e’) =/=> e 🡪 e’
    - How do you construct the vector timestamps
    - Vector clock Vi at process Pi 🡪
      * array of integers, dim N
      * VC1: initially Vi[j] = 0, for j = 1…N
      * VC2: before Pi timestamps event it sets Vi[i] = Vi[i]++
      * VC3: Pi piggybacks t = Vi

TODO:

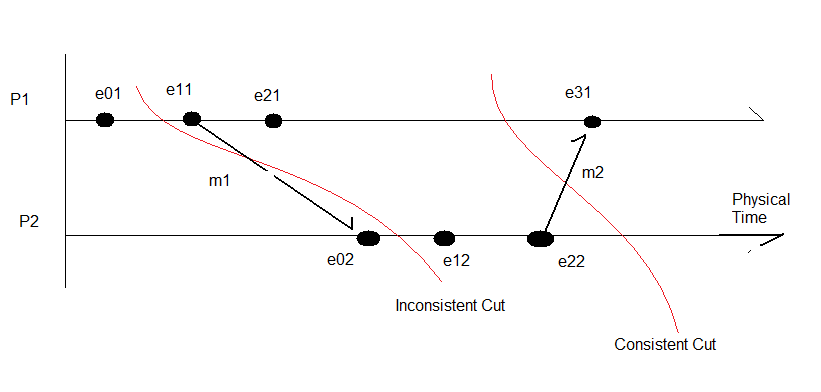
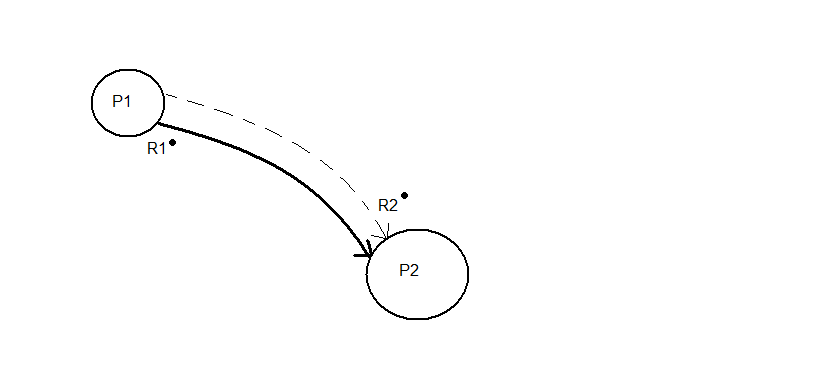
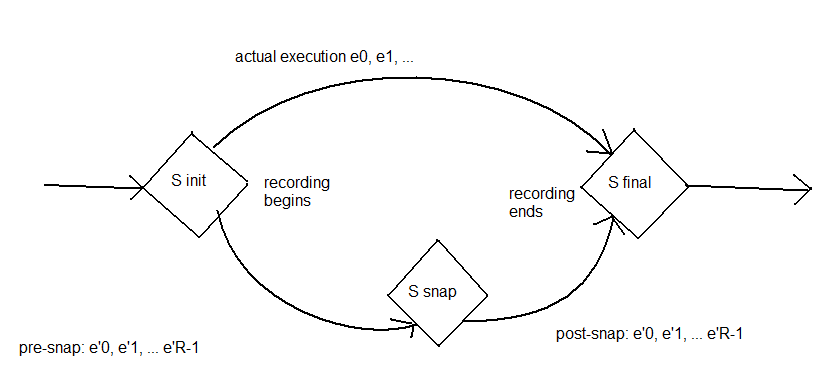
* Project 2 Due FRIDAY!!
* Project 1 Documentation Due FRIDAY!

CS 499 – Distributed Systems  
Monday, September 28, 2009

Class Notes:

* This Friday is self study on the rest of the chapter
  + the chapter pages can be found on the course website
* Will finish up global states discussion today

Lecture Notes:

* Global State
  + if / thens
    - garbage collection?
    - variable reference?
  + stability?
* State
  + is what is recorded right before an event.
    - so that it can be indexed and cataloged
  + Could reenact an event if given enough records.
    - However because we don’t know the full conditions that define what happened in the last event, therefore it is possible to have a different outcome than the original.
* Cut
  + 
  + //TODO: get def of cut
  + C = h1^(c1) U h2^(c2) U h3^(c3)…hN^(cN)
  + Consistent “… “ 🡪“linearization”
* Chandy and Lamport
  + “Snapshot”
  + message – that doesn’t interfere with normal activity
  + Marker Receiving Rule (on Channel C):
    - If (process recorded state?)
      * Not recorded
        + record state of Channel C as <empty> set
        + start observing mark on all other channels
        + execute Marker Sending Rule
      * Recorded
        + do the above as well as record the state of the channels
    - // both must happen atomically
  + Marker Sending Rule
    - record your state
    - send out marker message on all outgoing channels
  + How do you start?
    - any process can start the algorithm by pretending that a marker message was received (on no channel)
  + When the marker message comes in you begin recording
  + 
  + 
* We will begin a new chapter: Transactions and (something)
  + “this involves databases”
  + examples of transaction in database
    - queries
    - (send command and receive confirmation)
  + Transaction (DEF):
    - a container
    - ??
    - list of actions that needs to be handled as one
  + ACID
    - Atomicity – transaction can be done as a whole or not at all
    - Consistency – a series of actions takes from one consistent state to another consistent state
    - Isolation – one transactions does not have side effects other transactions
      * you should feel as if it’s you and the database only
    - Durability – once a transaction occurs the system is durable. transactions to permanent storage

TODO:

* Look into Christian’s Method
* Look into Berkley method

CS 499 – Distributed Systems  
Wednesday, September 30, 2009

Class Notes:

* Quiz 3 – Global State Predicates
  + example of stable
  + example of unstable
  + Snapshot algorithm 🡪 it allows you to draw conclusion on …

Lecture Notes:

* ACID
  + Atomicity
  + Consistency
  + Isolation
  + Durability
* ACID
  + Banking example:
    - Operation of the Account interface
      * deposit (amount)
      * withdraw(amount)
      * getBalance( ) 🡪 amount
      * setBalance(amount)
    - Operations of the Branch interfaced
      * create(name) 🡪 account
      * lookup(name) 🡪 account
      * branchTotal( ) 🡪 amount
    - CORIDORNATOR INTERFACE
      * openTransaction( ) 🡪 trans
      * closeTransactions(trans) 🡪(commit, abort)
      * abortTransaction(trans)
    - Notes
      * the goal is to produce as many transactions as possible
      * easy to guarantee consistency 🡺 serialize the transactions
        + but to strictly serialize is not the solution
  + Quick look at what a transaction example:
    - **SUCCESSFUL**  
      openTransaction  
      *operation  
      operation*  
      .  
      .  
      *operation*  
      closeTransaction
    - **Aborted by client**openTransaction  
      *operation  
      operation  
      .  
      .  
      operation*  
      abortTransaction
    - **Aborted by Server**openTransaction  
      *operation  
      operation  
      .  
      . //server aborts transaction  
      .  
      operation ERROR*  
      reported to client
    - Options for the Client:
      * Start over
    - Why was aborted?
* Concurrency Control:
  + Example: “Lost update problem”:
    - two transactions T and U
    - Order of operations leads to conflict
    - Solution would be for all of one transaction segment to complete first before the other.
  + Example: “Inconsistent Retrieval”:
    - Transactions V and W
    - Solution was to serialize all of the transaction
* Serial Equivalence
  + // this is to provide the solutions to the above examples above

TODO:

CS 499 – Distributed Systems  
Wednesday, October 07, 2009

Lecture Notes:

* Midterm this Friday!
* Chapters 1 – 5
* Chapters 1 and 2:
  + Otte should provide us better notes to cover these subjects.
  + The book did not cover this well enough
* Chapters 3 – 5
  + The book covers these well enough

C lass Notes:

* Chapter 1
  + What is a Distributed System?
  + Autonimity
  + Heterogeneous
  + Motivation for Distributed Systems
    - Resouce sharring
    - modularity
    - scalibilty
    - incremental
  + Parallel Systems
    - multi process system
    - multi parallel system
    - // Def – tightly coupled…
  + // DS (def) – loose coupling of loosely …
  + Classification of Primative
    - sync
      * no real question
      * execution:
        + ratio between working on something that make sense
        + overhead
        + large granularity for Distributed Systems

Why is this so?

* + - async
      * no real question
      * execution:
        + (same as above)
    - blocking
    - non-blocking
  + Challenges
    - From System Perspective
    - Algorithmic Perspective
      * How it challenging for algorithms for DS
    - Network Transparency
      * access transparency
        + internet

powered by URL’s

access is the same as long as you stay in your browser

* + - * location transparency
    - Provide for Transparency
    - Concurrency
    - Synchronization/Asynchronization
    - Consistency
    - Fault Tolerance
    - API for Communications
* Chapter 2
  + Causal Relationships
    - e2 -/-> e1
    - e1 -/-> e2
    - e1 || e2
    - e2 🡪 e3 🡺 e1 🡪 e3
  + Concurrency
    - not interested in physical concurrency
* Chapter 3
  + Taken directly from the book, Chapter 11.1 – 11.4
  + Physical Time
    - know and describe the algorithms for physical time
  + Precision
    - Christian’s Method
  + know when an algorithm is internal/external synchronization
  + When you know your clock is off
    - When the clock is to slow…
      * set clock for future
    - When the clock is to fast…
      * harder to fix
  + Network Time Protocol (NTP)
    - //might see numbers on exam
    - in principle you should know how this is done
  + Subject
    - Lamport’s clock
    - Vector clock
    - Overhead
      * Dimension
      * if you have a lot of processes then you have potentially have overhead
      * but because of today’s technology this is really not that bad
* Chapter 4
  + Global States
  + Terms
  + Snapshot Algorithm
    - allows for conclusion about properties, we only looked at stable properties
    - requirements:
      * garbage collection
    - // instable
      * debugging
  + Process
    - prefix
    - front tier event
    - from the prefix we should know the global state of the process
  + Cut
    - sum of the prefixes
    - //consistent vs insconsistent
    - you can cut anywhere
      * but does it make sense to perform a bad cut
    - Consistent Cuts
      * // linearization
      * talked about
    - Need to explain
      * that it does terminate
      * it is consistent
      * ….
    - You really only need a consistent cut
  + Transactions
    - ACID Properties
    - Serial [Equivalence]
      * how you would order transactions
* Chapter 5
* From book
  + Time
    - 11.1 - 11.4
  + Global State
    - 11.5 – 11.6
  + Transactions & Concurrency Control
    - 13
  + Coordination & Agreement
    - 12.2
    - 12.3
    - 12.5
  + Message ordering & Group Communication
    - 12.4
  + Deadlock Detection
    - 13.4.1

TODO:

CS 499 – Distributed Systems  
Monday, October 19th, 2009

TODO:

* <late> Get updated notes from other student

C lass Notes:

Lecture Notes:

* Deadlocks
  + What are they?
    - resources, that cannot be shared
    - process are waiting for other processes to release a resource
    - cycles processors waiting for each other
  + What can you do about it?
    - avoid
    - breaking deadlock cycles
      * abort transactions by looking at the cycles
      * time-outs
        + Are they a good idea?

for heavily accessed items they are accessed a lot so they would look like they are in a dead lock

// Not an ideal method for dealing with deadlocks

V

T

W

C

U

V

V

B

* Coordination and Agreement
  + Mutual Exclusion (ME)
    - Critical Sections (CS) – pieces of code that act on data on resources, and they need to be executed in an atomic way. If they cannot be executed in this way you will have corrupted data.
      * Actions need to be executed in a atomic way; those actions do not need to be atomic.
      * There is no concurrency when dealing with critical sections
    - Conditions:
      * ME 1 (safety): only one process is allowed to execute in the CS at a given time
      * ME 2 (liveness): all processes should have a fair chance to execute CS
        + Examples: “nice” in Linux, priority in task manager in Windows.
  + Central Server Algorithm:

4

2

p1

1. Request token

Queue of requests

2. Release token

3. Grant token

* + Ring Algorithm

token

CS 499 – Distributed System  
Wednesday, October 21, 2009

TODO:

* Look over the answers for the quiz below, they me show up again on the next exam.

Class Notes:

* Quiz 5
  + What is two-phase locking”?
  + See 1: what additional requirement does “strict two-phase locking” add?
  + What are the protocol rules in optimistic concurrency control that (automatically) guarantees strict execution? Remember that you need to look at reads-reads and writes-writes of past transactions versus the current transaction.
  + // this may show up again it the exam.

Lecture Notes:

* Mutual Exclusion (ME) :
  + // “liveness” will be used, not proper still but still could be used
  + Conditions: (continued from last lecture)
    - ME 3 (fairness): processes are completed in the order they arrive
  + Algorithm:
    - Multicast & Logical Clocks (creators: Ricard & Agravola)
      * + *On initialization*

*state* := RELEASED;

* + - * + *To enter the section*

*state* := WANTED;

Multicast *request* to all processes;

T := request’s timestamp;

*Wait until* (number of replies received = (N -1));

*state* := HELD:

* + - * + *On receipt of a request <Ti, pi> at pj( i ≠ j )*

*if* (*state* = HELD or (*state* = WANTED *and* (*T, pj*) < (*T, pi*)))

*then*

queue *request* from *pi* without replying;

*else*

reply immediately to *pi*;

end if

* + - * + *To exit the critical section*

*state* := RELEASD

reply to any queued requests;

* + - * What does this algorithm provide us?
      * Because of the “if” statement we are ensuring fairness because we are comparing timestamps.
        + No two processes should have the same timestamp.
    - Maekawa’s Algorithm:
      * Just mentioned in passing
      * will not be on the next exam
  + Performance of Algorithms:
    - Central Server Algorithm
      * Enter: 2 (round trip)
      * Exit: 1
      * Sync Delay: 2 (round trip)
        + time for process to releasing the Critical Section and for another to pick it back up
    - Ring Based
      * Enter: 0…N
      * Exit: 1
      * Sync Delay: 1…N
        + time for process to releasing the Critical Section and for another to pick it back up

CS 499 – Distributed Systems  
Wednesday, October 28, 2009

TODO:

Class Notes:

Lecture Notes:

* Mutual Exclusion is a big thing.
* From OS, what made this difficult
  + How do you synchronize operations?
    - not having a Machine Instruction for locking and setting
* Elections
  + To designate a process for a task
  + // When a process fails
    - the others has an agreement about how to deal with it
  + Choose Process with highest ID
    - // Assume we want to use Algorithms to find the greatest Identifier
      * so we can simplify the action of choosing
    - E1 (safety):
      * A participating process has “elected” set to undetermined or a process P, where P is the non-crashed process with biggest ID
      * // finding a valid process
      * // safety is the most important thing
    - E1 (liveness):
      * All processes participate and set elected to valid ID or crash
      * // make sure all processes participate
* Algorithms:
  + No Central Server algorithm
    - // (?) we assume this is the default
  + Ring-Based Algorithm
    - * system can be asynchronous
      * Processes don’t crash
    - Participants/non-participants
    - Distinguish between: Election/Elected messages
    - If an a process wants to \_\_\_\_\_\_\_\_\_\_\_ it sends a election message to its successor with ID value.
      * What happens when a process gets an election message
      * Flow Chart:

ID greater than my own

Forward

Am I Participating?

Put own ID in

Do nothing

Set status to participant

Send out elected message

* + Bully Algorithm
    - //make Assumptions
    - A-priori knowledge of all the process IDs
      * // (A-priori is the best assumption)
      * The system is synchronous
        + this is a requirement
    - Properties (?)
      * Election message, Answer Message, Coordinator Messages
      * Processes can crash
    - Algorithm:
      * Process with highest ID can send out coordinator message.
        + // Highest ID becomes the Coordinator
      * Process can send out election message
        + // any process
        + but can only send to higher ID processes
        + Waits for Answer Message

If none ( in T ) 🡪 send coordinator message

If there is answer 🡪 waits for coordinator

if you don’t get coordinator message, start new election

* + - * + If you get an election message 🡪 start an election

CS 499 – Distributed System  
Friday, October 30, 2009

TODO:

Class Notes:

* Otte will provide over for the rest of the semester
  + Upcoming Monday (in a week? November 9th) Graduate
  + Next week:
    - Monday: message ordering & group comm.
    - Wednesday: message ordering & group comm.
    - Friday: transactions & C onc. Control.
* Then Presentation

Lecture Notes:

CS 499 – Distributed Systems  
Monday, November 02, 2009

TODO:

* Read the book for the topic, “Consensus” - page 499

Class Notes:

* Quiz Wednesday
  + Covers the Consensus topic from book
  + Page 499
* (Group) Presentation for the presentation will be Monday.
* Veteran’s Day Wednesday (no classes)
* (Grad) Presentation will be Friday

Lecture Notes:

* Bully Algorithm
  + can only send messages to higher ID processes
  + Highest ID basically is the bully
* // Done with Election and Concurrency
* Consensus
  + What kind of system do we have?
    - // two fundamental systems
    - (1) Synchronous System;
      * can make assumptions on max/min time of message transfer
      * (i.e.; wait until message has left local buffer and has transferred on the wire)
    - (2) Asynchronous System
      * (“Hey I’m here, I will do what you need but in the mean time I will doing this…”)
    - // (Note: This is not directly related to synchronous and asynchronous communication.)
    - Do we have upper bounds on…?:
      * Message Transfer Time
      * Process Execution Time
      * Clock Drift
  + Fun thinking exercise
    - * Oranges and Apples are two divisions in Pepperland
      * Reside on two different hilltops, separated by a canyon
      * In the blue canyon live the Blue Meanies
      * Apples and Oranges can send messages to each other, by sending messengers
    - Question: can they reach an agreement on:
      * Who leads the charge?
      * When to charge?
    - Case 1:
      * Messengers don’t get lost, but can take forever
      * It’s possible to reach an agreement on who leads the charge
      * They cannot agree on when to charge
    - Case 2:
      * Messengers can deliver within a given aount of time
      * synchronous system, reliable message transfer
      * We can also agree on when to charge
        + Minimum and Maximum time
      * The leader would send message “Charge” and then wait for minimum time
      * On reception of message the counterpart waits for X, then charge
      * The counterpart would join the attack not later than:
        + max-min+X
    - Case 3:
      * reliable message transfer, but blue Meanies can defeat either Oranges or Apples (processes can fail)
      * Asynchronous System: you cannot distinguish whether
        + counterpart was defeated
        + message is just slow
      * (assume) Synchronous System: (processes can fail but the processes …)
        + “we can say for sure, whether the counterpart system failed or not”
    - Case 4:
      * Unreliable message transfer (messengers are caught or can be caught.), no process failure
      * task: to agree on charge or surrenders
      * Consistency matters
      * example
        + Apples 🡪 “Charge” 🡪 Oranges
        + However the message could almost get in a circular loop between Oranges and Apples.
  + What make our modern day Internet usable
    - timeouts
    - assume its synchronous
      * for the most part we are justified for our assumptions
    - our system is based on send, fail, send again, full fail.

CS 499 – Distributed Systems  
Wednesday, November 04, 2009

TODO:

Class Notes:

* Quiz 6
  + (1) The mutual exclusion algorithms we know from operating systems don’t work in distributed systems. Why?
  + (2) Why is it essential for the bully algorithm to be run in synchronous systems (only)?
  + (3) Consensus: is it possible to agree on simple facts in an unreliable (concerning message), asynchronous system? How about reliable (again concerning message transfer) asynchronous systems?
    - (a) no -
    - (b) yes – this is possible by assuming the system is synchronous.
* Schedule
  + Friday – Lecture: Multicast Ordering
  + Monday – Project Due/Presentation
  + Wednesday – no class
  + Friday – Grad presentation
  + Monday - (possibly Replication lecture)

Lecture Notes:

* Why can the Internet have consensus?
  + Given that we know, the Internet is an asynchronous system.
  + Answer: Timeouts
    - The rate of packets that are thrown out are high and reaches a drop limit (timeouts)
  + Answer: We assume the system is synchronous.
* Multicast-Communication
  + Introduction
    - Def: Multicast – variation on Broadcast.
      * We are not interested in the technical information on how to do Multicasting
      * We are interested in the implementation (not the technical)
    - Multicast vs. Broadcast
    - Closed Group vs. Open Group Multicast
      * Closed Group:
        + Message can be sent in within the group but no processes outside the group can send messages.
      * Open Group:
        + Outside processes can send messages to the group
        + But also inner group processes send messages to each other
      * BOTH can send message from inside the group to outside processes.
    - Distinction between Receiving & Delivering messages:
      * Question: how are you going to order the message at the receiver site(s)?
        + // pretty much all we will discuss about
        + understand: in order to understand a message ordering

Basically learn that you do not always deliver a message as soon as that is ready.

Rather you wait till all messages are ready then delivered to receiver in a set order.

* + - General Technique

Incoming messages

Hold-back queue

When delivery guarantees are met

Delivery queue

Message processing

Delivery

* + - * “hold-back queue”
        + sets the order of messages to be delivered
        + // waits for messages to be completed before sending.
      * Delivery Queue
        + set of messages ready to deliver
        + then delivered
    - What delivery guarantees are we waiting for?
      * FIFO ordering:
        + applies to a single sending process
        + If a process sends a message m before m’, then every process that delivers m’ will deliver m before m’
      * Causal Ordering
        + can be applied to multiple process

FIFO is implied

* + - * + If multicast(g, m) 🡪 multicast(g, m’)
        + Every process that delivers m’ will deliver m before m’.
      * Total ordering
        + If a process delivers message m before m’ then any other process that delivers m’ will deliver m before m’.
        + All messages will arrive in order, everywhere.

CS 499- Distribute System  
Friday, November 06, 2009

TODO:

Class Notes:

* Quiz 6 graded and returned.
  + (1) OS Algorithms cannot be applied to DS because you cannot have the same access to the same place in memory
    - TSL – cannot be used
  + (2) Bully depends on Sync because we need timeouts
  + (3)
    - (a) yes
    - (b) no
* Projects Due – Monday
* Wednesday – No class
* Friday – Grad Presentation, Project Documentation Due

Lecture Notes:

* Multicasting
  + // last lecture, unless graduates finish up early
  + Three ordering processes
    - FIFO
    - Causal
      * When a process responds to another process then those processes must maintain order across the all transactions.
    - Total
      * all messages, across the whole group, across all processes - have the same order
      * unambiguously
  + Implementation of Ordering Algorithms
    - FIFO Ordering
      * Sequence #’s
      * Process P has two variables:
        + S 🡪 counts how many messages P has sent

// each other process knows this number when multicasted

* + - * + Rgq - sequence # of the latest message P has delivered from process q (sent to g)

g – Group

q – place holder of processes in g

* + - * On Multicasting, message from P to group g:
        + P piggybacks S on message m.
        + P multicast m
        + P increments S
      * On Delivering message m from q:
        + P checks S contained in the message == Rgq + 1

If this is true 🡪 then deliver!

Let R 🡪 S

else keep it back

If S > R + 1

// too new, missing messages in between

Message place in hold-back queue, until all intermediate messages have been delivered & S = R + 1.

* + - Total Ordering
      * // sequence number used to keep messages in a group
      * sequencer – used to keep count and broadcast to group
      * (1) Algorithm for group member p
        + On initialization: rg := 0;
        + To TO-Multicast (total ordering) message m to group g

B-mulitcast ( g U { sequencer(g) }, <m, i> );

* + - * + On B-deliver( <m, i> ) with g = group(m)

Place <m, i> in hold-back queue;

* + - * + On B-deliver( morder = < “order”, i, S>) with g = group( morder )

wait until <m, i> in hold-back queue and S = rg;

TO-deliver m;

rg = S + 1;

* + - * (2) Algorithm for sequence of g:
        + On initialization: sg := 0;
        + On B-delver( <m, i> with g = group(m)

B-multicast( g, < “order”, i, sg> );

sg := sg + 1;

* + - Causal Ordering
      * Algorithm for group member pi( i = 1, 2, …, N )
      * ….
      * // save for beginning of Monday