

CS244B Final Review

Spring 2011

Administration

- Final Exam: Tuesday, June 7th
- Project 2 due tonight (Friday the 27th) at 11:59pm
 - You may use any late days you have left
 - (Yes, weekend days count as late days)

Final Exam Information

- Tuesday, 6/7: 3:30-6:30pm
- Gates B03 (regular lecture location)
- Closed book, no notes
- Not written up yet, but...
expect about half short answer, half multiple choice

Final Topics

- **Cumulative: All of chapters from 1 - 8**
 - Chapter 1: “Overview”
 - Chapter 2: “Distributed Shared Memory”
 - Chapter 3: “Structuring of Distributed Systems”
 - Chapter 4: “Clocks: Real, Virtual, Logical”
 - **Chapter 5: “Directories and Naming”**
 - **Chapter 6: “Accounts and Authentication”**
 - **Chapter 7: “Transactions, Agreement, Reconciliation”**
 - Chapter 8: “Future Directions and Issues”

Final Topics Cont'd

- Assigned Readings
 - Listed at end of each chapter as “Readings”
 - “References” are optional
- 12 in total (6 prior to midterm, 6 since)
 - Hint: Give priority to ones with most focus in reader and lectures

Final Hints

- Look at prior midterms/finals:
 - <https://www.stanford.edu/class/cs244b/exams/>
 - Format is consistent year after year
(though multiple choice is relatively new)
- Look for the key points
 - Introductions and conclusions in readings have good summaries
 - Understand the bad ideas and the good ideas
 - E.g. Secrecy vs. Open security

Final Hints, Cont'd

- Read over questions *carefully*
 - Don't lose points if we ask for n examples and you give us fewer
- Don't be afraid to use point form
 - But include details, demonstrate understanding (so we know it's not just rote recitation)

Chapter 5: Directories & Naming

- Brief Overview
- More detailed notes from Tahir's review sessions at <http://www.stanford.edu/class/cs244b/files/Naming-Directories-lecture.pdf>

Chapter 5: Directories & Naming

- **Directories:** Objects mapping names to other objects
- Why use names?
 - **Correctness:** identify something unambiguously (compare with a description, which is ambiguous)
 - **Efficiency:** names typically shorter than object they refer to
 - **Flexibility:** provides indirection, can rebind

Chapter 5: Directories & Naming

- **Pure vs. Impure Names**
 - Impure includes some description, pure does not
 - E.g. “Joe” vs. “Bicycle”
- **Impure Pros and Cons**
 - + Encoded info can facilitate mapping (e.g. area codes in phone #s)
 - + Provides info even without lookup (e.g. know /dev/tty is a device)
 - - May be too large (need to encode stuff)
 - - May have restricted rebinding
(If the name encodes some info, then “bicycle” probably can’t be pointed at an elephant)

Chapter 5: Directories & Naming

- **External vs. Internal Names**
 - **External** names are relevant to the user
 - Useful for error reporting, refer to objects from user-visible interfaces, global and independent of machine/address space/etc.
 - E.g. URLs, host names, path names
 - **Internal** names are an optimisation
 - More efficient, simpler machine representations
 - Need only be relevant in a context (e.g. per-TCP connection)
 - Therefore can be smaller, and transient
 - E.g.: IP addresses, file descriptors

Chapter 5: Directories & Naming

- Cheriton's View: Each object module should implement external names for its objects
 - Analogous to file systems - they implement both inodes and file/directory/etc names
 - **Fate-sharing**: name server is also the server you want. No instances where object server is up, but can't talk to it because name server is down.

Chapter 5: Directories & Naming

- **UUIDs** considered harmful
 - UUIDs are **internal** names, not **external**
 - Problems:
 - How big is big enough for uniqueness?
(Recall: one point of internal names was small representation)
 - Big identifiers are harder to resolve, more expensive to store
 - “Solution”: add impurities to make mapping easier
But this just makes the UUID even bigger!
 - Requires a UUID lookup service
(Can't add name service to each module; must rely on 3rd party)
 - What about legacy objects? What UUIDs do we give them?

Chapter 5: Directories & Naming

- Decentralised vs. Centralised Naming
 - How to support external naming?
- Centralised Naming (e.g. internet name servers)
 - + One authority to consult
 - + More efficient communication
 - + No inconsistent responses
 - - Difficult to extend to distributed systems (scalability, etc)
- Decentralised Naming
 - + Robust, easily distributed
 - - Multiple authorities to consult (need to query many and decide)
 - - Less efficient (more servers to contact, more messages, etc)

Chapter 5: Directories & Naming

- URLs vs. Academic Naming
 - Impure! protocol://server/object -- all in one name!
 - Can't really move object without changing URL
 - So why have URLs been so successful?
 - **Brands**: google.com, ibm.com, etc. connote credibility, importance
 - **Virtualisation**: URLs can point to many servers, to load balancers, etc.
 - **HTTP redirects**: some solution to moving objects
- Power of the hack
 - Hard to get things right first time (academic/committee paradise). Large-scale systems evolve instead.

Chapter 6: Accounts & Authentication

- **Crypto/Security Terms**
 - **Confidentiality**: Others cannot read the message
 - **Integrity**: Others cannot alter the message undetected
 - **Authentication**: Can determine who sent the message
 - **Non-repudiation**: Can't deny having sent the message
 - **Public-key crypto** (asymmetric): Separate public and private keys
 - Much, much slower than symmetric
 - **Shared-key crypto** (symmetric): Single shared secret key
 - **Certificate**: Certifies a public key for entity until expiry

Chapter 6: Accounts & Authentication

- 3 Issues with secret key management:
 - **Lifetime:** The longer a key exists, the weaker it is
 - **Selection:** If key generating key used, what if it's compromised?
 - **Distribution:** How do we distribute keys?
 - External channels
 - High overhead
 - Chained keys - last key used to transmit new one
 - Once a key is known, can get future keys
 - 2-level: Distribution key used to disseminate many data keys

Chapter 6: Accounts & Authentication

- Cheriton's View: PKI Certificates are totally bogus
 - How do we know a given certificate is legit?
 - If compromise was discovered, still need to wait for **revocation** list to propagate
 - If not, too bad
 - Root key is a **single point of failure**
 - Too terrible to contemplate it being compromised
 - Liability is unclear
 - CAs generally indemnify themselves - not at fault if their bag of bits go bad

Chapter 6: Accounts & Authentication

- Problems with secrecy:
 - Single point of failure
 - If your secret key is compromised, you're hosed
 - No indication of failure
 - Can't tell when it is no longer a secret
 - Not testable
 - No way to prove it's still secret
 - "Faith-based computing" - can only believe it's secret, not know it
 - Doesn't scale
 - The more distributed the secret is, the more vulnerable it is

Chapter 6: Accounts & Authentication

- Cheriton's Proposal: Open Security
 - Focus on real security - i.e. safety - not confidentiality
 - We often don't want or need confidentiality, so why use encryption?
 - ATC Example: Security through open, broadcast communication
 - Planes hear all instructions. Can't give conflicting orders.
 - Makes failure detection possible - look for conflicts
 - Distributed Systems Examples
 - NTP and naming
 - Rely on multicast to detect conflicts

Chapter 7: Transactions, Agreement & Reconciliation

- Transactions

- Multiple updates as single operation
- All-or-nothing
- Deal with tough DS issues: concurrency, fault-tolerance

- Advantages

- Automatic undo on failure (aborts to consistent state)
- Simplified concurrency control
- Locking done automatically (client needn't acquire/release)
- Avoids priority inversion (high priority TX can abort lower)
- Deadlock resolution (can abort TX and start again)
- Batch operations (for efficiency)

Chapter 7: Transactions, Agreement & Reconciliation

- Atomic transactions look like a great mechanism
- The trouble is:
 - - They're not efficient enough
 - - Availability may not suffice
 - E.g.: Amazon wants to continue running in face of network partitions

Chapter 7: Transactions, Agreement & Reconciliation

- Implementing Transactions
 - 1-phase commit: Coordinator tells all server to commit
 - Problem: Servers can't deny the request
 - 2-phase commit: Voting phase, Commit/Abort phase
 - Problem: Servers block if coordinator dies after first phase, but before issuing commit/abort (nobody knows vote result)
 - 3-phase commit: Add a middle phase
 - Coordinator tells everyone what the vote decided, before issuing commit or abort
 - Problem: Byzantine servers
 - You servers may be misbehaving/evil - what to do?
- General idea: Can fix issues by adding RTTs, but this greatly affects performance & increases complexity

Chapter 7: Transactions, Agreement & Reconciliation

- Alternatives
 - SQL has 4 “isolation levels” to trade off performance and consistency
 - Read uncommitted - dirty reads (read values never committed)
 - Read committed - values may change
 - Repeatable read - phantom reads (extra values in next query)
 - Serializable
 - Snapshot Isolation
 - Perform transactions on COW snapshot

Chapter 7: Transactions, Agreement & Reconciliation

- **Consistency vs. Availability**
 - Classic trade-off: you can't have both without paying for it
- **Availability often trumps consistency**
 - Can you afford to block if datacenter cut in half by network partition? Can't be available and consistent
- **Dynamo**: Writes never lost. Applications resolve inconsistencies in storage.

The End

Good Luck!