

# Systems Comprehensive Exam, Fall 2008

## 1 Instructions

This is a closed-book, closed-notes exam with a total of 100 points. You may not use any external source for answering these questions, including but not limited to the Internet, books, notes, or other people. Please direct any questions about this exam to Professor Barney Maccabe. Professor Maccabe may be reached either in person in his office in 301F Farris, by phone at 280-3542 (cell), or by email at [maccabe@cs.unm.edu](mailto:maccabe@cs.unm.edu). Turn your exam in to Lynne Jacobson by 5:00 PM MDT on Wednesday, August 20, 2008. Exams *will not* be accepted after this time except by prior arrangement.

Type or write your answers to the stated number of questions in each of the following three sections. Make any *reasonable* assumptions necessary to answer the question, but be sure to state any assumptions that you make.

## 2 Short Answer - Answer 3 of 4 (30 points)

*Briefly* answer 3 of the following 4 questions. Your answer should be no longer than *one* paragraph.

1. What are the three types of misses that can occur in a hardware cache system, and how might each of the following changes effect the rates of the different types of misses in a caching system?
  - a. Doubling the size of the cache
  - b. Changing from direct mapped to fully associative cache
  - c. Increasing cache line size in a direct-mapped cache, total cache size stays the same
2. Recall that disk scheduling algorithms control the order in which a sequence of disk blocks are written to disk. Briefly outline the FIFO and SCAN (a.k.a. elevator) algorithms, and the advantages and disadvantages of each.
3. Briefly compare and contrast the TCP and UDP protocols both in terms of the semantics they provide to applications and the mechanisms they contain for implementing these semantics.
4. Describe, briefly, the goal of the *two-phase commit* algorithm in distributed systems, and roughly outline the fundamental aspects of the algorithm.

## 3 Medium Answer - Answer 2 of 3 (40 points)

Provide detailed answers to two of the following three questions. Be sure to state any assumptions you make and to fully justify your answers. Limit your answers to approximately one to two pages in length.

1. Hardware transactional memory is a relatively recent innovation that provides basic support for atomic non-blocking sequences of instructions. It does so by introducing new instructions like `begin-transaction`, which designates that start of a sequence of atomic instructions, and `commit-transaction r1`, which attempts to commit the transaction and stores whether the transaction committed or aborted into

register `r1`. These hardware memory transactions commit if no other processor modified a memory location read or written by this processor during the transaction, and abort otherwise.

In the simplest cases, limited transactional memory (i.e. the only kind we are considering here, with limited-length non-nesting transactions) is generally implemented largely through additions to the processor caching system. For example, uncommitted writes are stored only in the store buffers and (write-back) caches of the processor conducting the transaction to avoid exposing inconsistent results to other processors, and memory bus snooping is used to detect when an ongoing transaction should fail.

- a. How would such instructions ease the design and implementation of multi-processor safe data structures for multi-core systems, compared to existing instructions, for example `compare-and-swap`, `load-linked/store-conditional`, or `test-and-set`?
  - b. Many multi-processor operating systems have moved to per-processor local schedulers (with no locks and explicit load balancing/load stealing between processor scheduling queues) instead of a single global scheduler. As you already know, this avoids the locking costs of a global scheduler, but reduces the efficiency of the resulting scheduling decisions. Will efficient hardware transactional memory support, allowing the removal of locks in the global scheduler queues in favor of transactions, make global scheduling preferable to local scheduling? Thoroughly justify your answer.
  - c. Some operating system data structures are, unfortunately, not easy to localize per-processor and currently require the use of locks, for example process page tables. With efficient hardware transactional memory support, would it be appropriate to remove the locks currently used in these data structures in favor of hardware transactions? Thoroughly justify your answer.
2. File system indexing features have been added to many operating systems, both as low-level operating system functionality with kernel support (e.g. Apple Spotlight) and as add-on user-level programs (e.g. Google Desktop). What are the advantages and disadvantages of each of these approaches to indexing and searching file systems?
  3. Distributed hash tables, e.g. Chord, are now essential data structures in a number of distributed peer-to-peer systems. Thoroughly describe what a distributed hash table is, how items are looked up in distributed hash table scheme, for example Chord, and how these systems tolerate the failure (specifically, crashing) of nodes in the system.

## 4 Design - Answer 1 of 2 (30 points)

Provide a *full* and *detailed* answer to one of the following two questions. Be sure to state any assumptions you make and to fully justify your answer.

### 4.1 Provenance

The *provenance* of computer data, that is the *complete* history of how and when it was created, is an increasingly important piece of metadata. For example, the full provenance information about a data file could indicate the process that generated it and which input files that process used to generate the output file, and all of the files used, directly or indirectly, to create those input files and the associated process, and so on. Similarly, the provenance of an object file might contain information on which compiler and assembler were used to create it, and which source files it was generated from, and so on. Such information can be used for debugging, experiment documentation and verification, and variety of data mining purposes, to name a few.

Consider systems for handling provenance information in the file system as metadata for files in that file system.

- Propose a system for representing, storing, and maintaining provenance information for files in a file system. Be sure to consider the implications of the provenance of existing files depending on files that have been deleted.
- Consider two techniques for gather provenance information for files, one which provides tools to allow users and programs to annotate files they create and modify with provenance information, and one which automatically infers provenance information based on user and system activity.
  - How might you add support to the operating system to automatically infer provenance information for a file based on system activity?
  - What are the advantages and disadvantages of such automatically inferred provenance information compared to manual annotation of files with provenance information?

## 4.2 Cloud Computing

*Cloud Computing* is a generic term used to describe a system where a user's computation, service, and storage resides "out there" distributed across servers and peers in the Internet (the cloud). The various Google services (gmail, google docs, map-reduce, etc.) are but one example of this. Such systems allows users to access their computational state from anywhere network connectivity is available through devices ranging from mobile phones to laptops to desktops.

Performance, fault tolerance, and heterogeneity are all significant technical issues in such a system. Using your every-day computing demands (compilation, simulation, document editing, email, web surfing, Starcraft, etc.) as a model, thoroughly discuss what issues in these areas arise from moving such activities in the cloud, and outline systems for moving your day-to-day computational needs into the cloud by dealing with these issues.