1. Soft Updates and Journaling [9 points]

(a) There are certain invariants that must be true to ensure that a file system will always be in a consistent state.

- Name at least 2 invariants that FFS respects
  1.) directory entries never point to non-existant files
  2.) an in-use disk block (whether inode or data) is never marked as free
  3.) inodes never point to data blocks of other files

- Provide at least one thing that a reasonable person might like but is not guaranteed by FFS
  1.) data blocks pointed to by an inode are properly written
  2.) unused disk block are properly marked as free

(b) Louis Reasoner has a workload that consists of creating a bunch of files in the same directory. Remembering that Journaling systems may perform additional I/O to maintain ordering information, Louis believes that FFS will perform fewer meta-data writes than Journaling. Is Louis Reasoner correct? Explain.

  **Journaling will perform MORE writes than FFS since it has to do all the same writes that FFS does plus additional writes to the journal. The difference is that the journal writes are (potentially) synchronous (depends on journal configuration), whereas all meta-data writes in FFS are synchronous.**

  So Louis is RIGHT if you consider journal entries as metadata writes.

  Or Louis is WRONG if you do not consider journal entries as metadata writes, since they do the same writes besides the journal.

(c) When evaluating Soft Updates, the authors discovered that its performance sometimes decreased when the size of the benchmark increased. That is, when the benchmarks operated on either larger data sets or for longer periods of time, the relative throughput of Soft Updates (as compared to FFS) often decreased considerably when compared to smaller data sets over shorter periods of time.

- List two causes of this effect.
  1.) Larger data sets will lead to more meta data writes which can increase the chances of rollbacks.
  2.) If there are deletes, running for a longer time will eventually have to write the delay deletes to disk which will hurt performance.

  - Soft Updates sometimes out-performs FFS-Async. Why is this surprising? Why is it happening?

    **This is surprising because FFS-Async does not do any extra operations to ensure reliability. This happens when we have workloads that do heavy deleting since Soft Updates does them in the background eventually.**

2. Rio [9 points]

(a) For reliability purposes, FFS writes metadata in a certain order. Does RIO also require metadata writes to be ordered for reliability? Explain.

  **Yes it does, since we are treating memory as persistent storage we must ensure that we cannot be in a bad state in memory should the system crash. So we must write in the right order to memory as well as the disk.**
(b) The authors of RIO injected their own bugs instead of using real ones. Why? What do they gain from this? Lose?

By injecting their own faults they tried to generate a wide variety of system crashes. By doing it themselves they had more control of the coverage of their fault injection. The only problem is fault injection cannot mimic the exact behavior of all real-world operating system crashes.

(c) Despite the fact that Rio marks the buffer cache as read only, the authors observe that it is still possible for the kernel to make wild writes to the buffer cache. Explain how Physical Addresses are sometimes used. The Rio solution is to write-protect file cache accesses by forcing all accesses to go through the TLB.

3. Caching and Prefetching [9 points]

(a) Describe a scenario(time access fetch cache contents like figure 1) where the kernel can tell if an application is acting foolish(one that uses a policy worse than the kernel's default policy, in this case LRU). Explain how it can tell.

Assuming it takes 3 cycles to fetch a data block and the application is the only thing running. It is obvious that this application is being foolish because it threw away B instead of A when making space for D even though B was coming up first which is a violation of Rule 2. A would also be replaced instead of B using the global LRU so it is clear this application is acting foolish since LRU would of been better.

(b) Does aggressive prefetching policy from Fig 1 of the paper adhere to the controlled-aggressive prefetching policy described in the paper?

It does because it always says yes to rule 3 and 4 whenever possible. It did not try to get rid of B when fetching C in the figure and it did not delay it.

(c) The LRU-list of a kernel is depicted below, where the frames Ai belong to process A, and the frames Bi belong to process B. The left end of the list corresponds to the item that has been the least recently used.

A1 -> B1 -> A2 -> B2

Suppose process A chooses to use its own page replacement policy of most recently used(MRU) while B uses the system's default policy of LRU.

Illustrate the state of the kernel's LRU-list with placeholders using the LRU-SP policy mentioned in the paper at the end of each of the following page references for the page reference string A2, B3, A3, A1, A4, B3.

  |  A2
4. GMS [9 points]

(a) Louis Reasoner has a machine that runs an important application that requires most of the CPU at all times. His application rarely touches memory tough so he believes that he can help out the community by inserting his machine into a cluster running GMS without hurting the application running on his machine. Is Louis Reasoner right? Explain

Louis Reasoner is incorrect because in addition to sharing his computer's memory, he must also share his computer's cycles which will ruin his important application. See figure 13 in the paper.

(b) Ben Bitdiddle is concerned for the well-being of his application's pages. He fears that since his pages can be anywhere in the cluster, he might lose the page's data if a node carrying it crashes or leaves the cluster. Should Ben Bitdiddle be worried?

No, all pages sent to global memory are cleaned first so he will always have the right data in his computer's disk.

(c) The authors of GMS provide many microbenchmarks to show the ease and benefits of using GMS in a cluster.

- In table 3, explain why there is a large difference in GMS and no GMS for random access.
  
  **Prefetching/cache locality in sequential access keeps no gms access latency relatively low**

- In table 4 why is GMS duplicate lower than GMS single?
  
  **Multiple clients give more chance to put in memory.**