CS439: Fall 2011 – Midterm 2

Instructions

- Stop writing when"time" is announced at the end of the exam. I will leave the room as soon as I've given people a fair chance to bring me the exams. I will not accept exams once my foot crosses the threshold.
- You may use a scientific, non-programmable, unnetworked calculator. You may not use a programmable calculator or a calculator on your phone.
- Place your name and uteid on **all** pages of the exam **now.** You will not have time to do this after "time" is announced.
- This midterm is closed book and notes.
- If a question is unclear, write down the point you find ambiguous, make a reasonable interpretation, write down that interpretation, and proceed.
- For full credit, show your work and explain your reasoning and any important assumptions.

  Write brief, precise, and legible answers. Rambling brain-dumps are unlikely to be effective. **Think before you start writing** so that you can crisply describe a simple approach rather than muddle your way through a complex description that "works around" each issue as you come to it. **Perhaps jot down an outline** to organize your thoughts. And remember, a **picture can be worth 1000 words.**
- **Write your name and uteid on every page of this exam.**

1 File systems (15)

Suppose I have a multi-level index file system (like the fast file system FFS for Unix), and this file system has 2KB blocks, inodes with 10 direct, 1 indirect, 1 double-indirect, 1 triple-indirect, and 1 quadruple-indirect pointer, and 64-bit block identifiers. Estimate to within 2% the maximum size file this system supports.

Solution: A 2KB block stores $2^{11}/2^3 = 2^8 = 256$ block identifiers. So, a single indirect block can cover 2^8 blocks, a double indirect can index 2^{16} blocks, a triple indirect can index 2^{24} blocks, and a quadruple indirect can index 2^{32} blocks. So, we have $2^{32} + 2^{24} + 2^{16} + 2^8 + 10$ blocks and each block is 2^{11} bytes, so we have approximately $2^{32} * 2^{11} = 2^{43}$ bytes in a large file; a bit over 8TB.

2 File systems (20)

Consider the (tiny) disk with 64 sectors shown on the last page of this exam (feel free to tear it off), which stores a FFS-like file system. The disk reserves the first 16 sectors for its inode array. Each sector stores four 4-byte words. An inode fills a sector and contains 2 direct pointers, 1 indirect pointer, and 1 double-indirect pointer. An inumber is a 4-byte integer. In a directory, a file name is a 4-byte array of 1-byte characters. A block ID is a 4-byte integer. The root directory's inumber is 0.

1. For the file system on this disk how large (in sectors) is the file with inumber 1?

   Solution: 1 sector

2. For the file system on this disk how large (in sectors) is the file with inumber 4?

   Solution: 3 sectors

3. For the file system on this disk, list the file names for the root directory

   Solution: HELP, ME , MARY, HAD

4. For th file system on this disk, what is the inumber of file /MARY/ABLE?

   Solution: 10

5. For the file system on this disk, what do I get if I read the entire file /ME/WAS?

   Solution: APT 7 ABLE 10 MARY 2 MOVE 8

6. Can this file system support soft links? Why or why not?

   Solution: It can, but only to paths whose names fit in the limited-sized files this systme supports

3 Disk performance (40)

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle speed	7200 RPM
Average seek time read/write	10.5 ms/ 12.0 ms
Maximum seek time	19 ms
Track-to-track seek time	1 ms
Transfer rate (surface to buffer)	54–128 MB/s
Transfer rate (buffer to host)	375 MB/s
Buffer memory	16 MB
Reliability	
Nonrecoverable read errors per sectors read	1 sector per 10^{14}
MTBF	600,000 hours
Product life	5 years or 20,000 power-on hours
Power	
Typical	16.35 W
Idle	11.68 W

Supose I have a disk such as the one described in the table above and a workload consisting of a continuous stream of updates to random blocks of the disk.

Assume that the disk scheduler uses the SCAN/Elevator algorithm.

- What is the throughput in number of requests per second if the application issues one request at a time and waits until the block is safely stored on disk before issuing the next request?

  Solution: Since the seek $+\frac{1}{2}$ rot + xfer = .012 + $\frac{1}{2}$ 60/7200 + 512/54,000,000 = 16.2ms per request, so the throughput is 1/.0162 = 61.9 requests/s

- What is the throughput in number of requests per second if the application buffers 100 MB of writes, issues those 100 MB worth of writes to disk as a batch, and waits until those writes are safely on disk before issuing the next 100 MB batch of requests?

  Solution: 100MB of writes is 200,000 512-byte writes. Each track has about 100MB * 60/7200 = 833KB on it, so there are about 320GB/4 / 833KB = 100,000 tracks on each surface and about 400K tracks overall. So, each track has an average of .5 updates buffered for it in that 100MB. We're going to ignore the cases when a track has multiple requests to it in a batch, and we're going to estimate the seek time as the track-to-track seek time.
  I'm going to estimate the rotation time as 1/2 rotation; we can probably do a bit better since the average "cylinder" has 2 sectors on it, SCAN can pick them in a good order.
  So, we have seek + rot + xfer = 1ms + 4.2ms + 512/54,000,000 = 5.2ms per request, so the throughput is about 1/.0052 = 192 requests/sec.

Suppose that we must ensure that – even in the event of a crash – the ith update can be observed by a read after crash recovery only if all updates that preceeded the ith update can be read after the crash. That is, we have a FIFO property for updates – the i+1'st update cannot "finish" until the ith update finishes. (1) Design an approach to get good performance for this workload. (2) Explain why your design ensures FIFO even if crashes occur. (3) Estimate your approach's throughput in number of requests per second. (For comparison with the previous part of the problem, your solution should not require significantly more than 100MB of main-memory buffer space.)

- (1) Design an approach to get good performance for this workload. (Be sure to explain how writes, reads, and crash recovery work.)

  Solution: We can use a replay log. Write 100 MB to buffer, then to log, then commit the log, then replay to disk. In normal operation, reads first check buffer then disk. On crash recovery, block reads until replay completes.

- (2) Explain why your design ensures FIFO even if crashes occur.

  Solution: FIFO is assured b/c reads always observe the all previously completed writes (via read from buffer or via replay after crash.)

- (3) Estimate your approach's throughput in number of requests per second.

  Solution: Performance will be as above + the time to write to the log. The log will take under 1 second to write (100MB/128MB/s + about 10 1ms seeks, adding less than 1s/200,000 = 5 microseconds per update. This increases our per updae time from 9.3ms per request (previous quesiton) to 9.300005ms per request – a negligible amount..

4 Disk reliability (25)

Consider a RAID system with 20 disks of size 1TB (1TB = 10^{12} bytes) that is arranged into 2 groups of 10 disks each, allowing 9 data blocks and one parity block to be stored across a group of 10 disks. Assume disk failures are uncorrelated and that the MTTF of a single disk is 1.5 million hours. Assume a MTTR (mean time to repair) of 10 hours and a sustained bandwidth of 100MB/s.

- Considering only complete disk failures (e.g., ignore unreadable sector errors) what is the MTTDL (mean time to data loss — the mean time until a double-disk failure in a group in this case) for this system?

  Solution: mttdl = mttf**2/N*(g-1)(MTTR)= 1.5*10**6 **2/(20*9*10) = 1.25*10**9 hours

- Suppose that one failed disk in a group is being replaced. Assuming that physical replacement was instantaneous (e.g., using a hot spare), what fraction of the other 9 disks' bandwidth will be consumed by recovery assuming that recovery for the new disk must complete within 10 hours?

  Solution: We need to read each disk in its entirety to reconstruct the bad disk, so we need to transfer 1TB at 100MB/s from each disk. 10**12/10**8 = 10**4 seconds. So, we use 10,000 seconds during that 10 hours for recovery. 10,000/(10*3600) = 0.2778

- Suppose that the unrecoverable read error rate is 1 sector per 10^{14} bits read. Considering just the I/O necessary to rebuild one failed disk in the above example, what is the probability of encountering an unrecoverable read error during a rebuild?

  Solution: We need to read 9 disks in their entirety, so we need to read 9TB for a rebuild, which is 72 * 10^{12} bits. This is about $\frac{72*10^{12}}{10^{14}} = \frac{7.2*10^{13}}{10^{14}} = 0.72$ of the expected time to hit an error; so the likelihood of hitting an error is high.
  We can be more precise. Assuming that each bit has a 10^{-14} chance of being wrong, then each bit has a 1-10^{-14} chance of being correct, and the chance of reading all bits successfully is $P_S = (1 - 10^{-14})^{9*8*10^{12}} = 0.487$

- Ignoring the result from the previous part of the problem problem, assume that there is a 10% chance that the system encounters a bit error during recovery. Estimate the MTTDL for the system accounting for both whole disk failures and unrecoverable read errors.

  Solution: The expected time to the first failure is MTTF/N = 1.5*10**6/10 = 1.5*10**5.
  What is the probability that we will fail to recover from the first failure? Two bad things can happen. First, we can encounter a uncorrectable bit error. Second, we can suffer a second failure in the group before recovery completes. We can look at each of these two processes as a failure rate. Considering only nonrecoverable bit errors, we have a failure rate of 0.1 * (1/1.5*10**5) = 1/1.5*10**6. Considering only double-disk failures, we have a failure rate of 1/(1.25*10**9) (from above.) Combining these rates, we have a total failure rate of 1/1.5*10**6 + 1/1.25*10**9 ≈ 1.5*10**6.
  Notice that simply weighting the separate recovery times for the two cases doesn't work. For example, suppose that you have a disk that fails once every 1000 years but there is a 10% chance that your data center will explode each year. We know that the MTTDL must be less than 10 years based on the second factor alone, but if we try adding (0.9 * 1000 + 0.1 * 1) we get 900.1 years. Instead, we need to add the rates of failures and then invert that rate: MTTDL = 1/(1/1000 + 1/10) = 9.901 years.

0	1	2	3	4	5	6	7
28	25	18	0	16	38	17	23
19	0	44	29	60	45	54	39
0	0	42	0	50	0	21	41
0	0	0	0	0	56	0	0
8	9	10	11	12	13	14	15
24	47	25	62	27	37	50	39
57	0	33	0	35	46	0	46
0	0	0	0	59	0	0	53
0	0	0	0	0	0	63	61
16	17	18	19	20	21	22	23
'ZED '	'ABLE'	43	'MARY'	53	22	'BAR '	47
10	13	21	13	63	35	8	43
'ZIPP'	'WAS '	19	'HAD '	24	0	'COOL'	38
3	7	-1	10	10	0	-1	31
24	25	26	27	28	29	30	31
55	'BAR '	'A '	'APT '	'HELP'	'LAMB'	'BAR '	31
0	9	11	2	4	14	14	27
0	'ZED '	'LITL'	'ZED '	'ME '	'WAS '	'ABLE'	19
0	8	6	8	9	12	6	42
32	33	34	35	36	37	38	39
'SNOW'	'THE '	'CAR'	'ABLE'	'FOO '	'APT '	3	'MARY'
8	14	8	9	12	7	6	20
'ABLE'	'END '	'POOL'	'BEEF'	'MARY'	'ABLE '	9	'MARK'
2	15	13	10	1	10	12	40
40	41	42	43	44	45	46	47
'MARK'	42	0	'CAR '	7	11	'MARY'	'FLCE'
7	44	0	3	8	4	2	12
'MARY'	0	43	'ABLE '	10	2	'MOVE'	'WAS '
9	0	0	11	11	8	8	13
48	49	50	51	52	53	54	55
'WHTE'	'ABLE'	51	'ABLE'	'BAR '	15	'ABLE'	24
8	2	0	8	11	14	5	0
'AS '	'WAS '	0	'WAS '	' '	13	'FOO '	0
12	14	0	6	-1	12	14	0
56	57	58	59	60	61	62	63
0	58	8	52	'APT '	43	'FAST'	0
0	0	20	0	7	18	15	0
0	0	15	0	'FOO '	31	'FOOD'	0
57	61	98	0	12	56	14	46