ACS Assignment4

Cristian Marius-Florin wdx186
Fernando José Faustino Alves xgd375

1 Recovery Concepts

1.1

A system implementig force: it means that DBSM forces all updates to disk before commiting.
A system implementing no-steal: there are no dirty writes for uncomitted transactions (the buffer manager can't arbitrary write a page to disk to free buffer space).
A redo reverts the effects of an undo; An undo reverts the changes. It is not needed to implement a scheme for redo nor undo as there are no dirty writes and the updates are forced to disk before the commit. So let's assume the following cases:
1. The crash happens in the middle of the transaction. This means that the buffer is lost, no dirty pages are on disk and nothing is commited on disk. We just start again.
2. We have an abort statement and the crash happens before the commit. It is the same situation as case 1.
3. We assume that the reads and writes are atomic, this meaning that if we reach the commit step and there is a crash, the files are already on disk.

1.2

Nonvolatile storage (hard-disk): retains data while the power is off.
Stable storage (theorateically data is never lost): retains data and duplicates on multiple nonvolatile storages.
Nonvolatile storages survive power failures but don't survive calamities, fires, bombs, floods, or other of this category that phisically destroys them. The stable storages consisting of a cluster of nonvolatile storages spread on a certain geographic area survive the calamities or at least some of them.

1.3

Before writing a page to disk every update log record that describes a change must be force to stable storage.
When a transaction commits, it force-writes a commit type log record and after the record is appended to the log, and the tail is written to stable storage. This is needed to ensure that in case of a crash (with a succeded commit) , we don't commit twice during the recovery.

2 ARIES

LSN	Last_LSN	TRAN_ID	TYPE	PAGE_ID
1	-	-	begin CKPT	-
2	-	-	end CKPT	-
3	NULL	T1	update	P2
4	3	T1	update	P1
5	NULL	T2	update	P5
6	NULL	T3	update	P3
7	6	T3	commit	-
8	5	T2	update	-
9	8	T2	update	P3
10	6	T3	end	-

CRASH!!

Analysis phase:
Scan forward through the log starting from LSN 1;
LSN 2: Initialize XACT TABLE; Initialize DPT; Both empty;
LSN 3: Add [T1, LSN 3] to XACT; Add [P2, LSN 3] to DPT;
LSN 4: Set [T1, lastLSN 4] in XACT; Add [P1, LSN 4] to DPT;
LSN 5: Add [T2, LSN 5] to XACT; Add [P5, LSN 5] to DPT;
LSN 6: Add [T3, LSN 6] to XACT; Add [P3, LSN 6] to DPT;
LSN 7: Change T3 status to "Commit" in XACT table; remove P3 from DPT
LSN 8: Set [T2, lastLSN 5] to XACT;
LSN 9: Set [T2, lastLSN 8] to XACT; Add [P3, LSN 9] to DPT;
LSN 10: Change T3 status to "End" in XACT table; remove T3

Redo phase:
Scan forward through the log starting from LSN 3;
LSN 3: Read page P2, check pageLSN, if pageLSN<3 REDO LSN 3 and set pageLSN to 3;
LSN 4: Read page P1, check pageLSN, if pageLSN<4 REDO LSN 4 and set pageLSN to 4;
LSN 5: Read page P5, check pageLSN, if pageLSN<5 REDO LSN 5 and set pageLSN to 5;
LSN 6: Read page P3, check pageLSN, if pageLSN<6 REDO LSN 6 and set pageLSN to 6; LSN 9: Read page P3 if it has been flushed, check pageLSN stored in the page, it will be 6, REDO LSN 9 and set pageLSN to 6.

Undo phase:
T2 must be undone. Put LSN 9 in ToUndo;
Write Abort record to log for T2;
LSN 9: Undo LSN 9 - write CLR for P3 and undoNextLSN=5; Undo P3; put LSN 5 in ToUndo;
LSN 5: Undo LSN 5 - write CLR for P5 and undoNextLSN=NULL; Undo P5;
T1 must be undone. Put LSN 4 in ToUndo;
Write Abort record to log for T1;
LSN 4: Undo LSN 5 - write CLR for P1 and undoNextLSN=3; Undo P1; put LSN 3 in ToUndo;
LSN 3: Undo LSNN 3 - write CLR for P2 and undoNextLSN=NULL; Undo P2.

T3 is winner transaction; T1,T2 are loser transactions.
After the Analysis phase: All pages are in the DPT and T1 and T2 are in the XACT.

3 Questions for Discussion on the Performance Measurements

3.1

Our experiments were run on a single computer with two cores Intel I5 6200U clocked to 2.30 GHz, each core has 3 private levels of cache, the L1 with 32 KBytes the L2 with 256KBytes and L3 with 3Mbytes. The machine runs on a 64 bit Windows 10.

The data generated was done by generating 12 stockbooks with different isbns and number of copies equal to the number of iterations per book (the first book has 0 copies the second 1 and so on). This data corresponds to what we want because after testing we got that the percentage of goodput / throughput was higher than 99the successful client interaction was around 60% (we repeated the procedure 20 times and we got an average of 60,85%).

For our measures, we did 20 repetitions of the results and applied the average on all of them so we can make plots with the best data possible. Here is some values collected:

With Serialization Type text using RPC (averages after 20 runs)

Number of work load threads = 10
Number of total runs = 1000
Average Latency = 657,849966 milliseconds
Aggregate goodput = 1,030 * 10^{-6} transactions per nanosecond

This means that the throughput is really low and we can handle more clients (workload threads), but I first tested with the same values for different serialization and for local test.

With serialization type binary using RPC (averages after 20 runs)

Number of workloads threads = 10
Number of total runs = 1000
Average Latency = 328,531809 milliseconds
Aggregate goodput = 2,36 * 10^{-6} transactions per nanosecond

With binary, the latency reduces a lot and throughput is higher this is

because it is easier and faster to transport binary rather than text, that is also way it allowed to transport more data in less time (bigger goodput).

Local Test (averages after 20 runs)

Number of workloads threads = 10
Number of total runs = 1000
Average Latency = 1,583318 milliseconds
Aggregate goodput = 7.512 $*10^{-4}$ transactions per nanosecond

In this case since it is a local test there is a lot more data being sent and the latency is way lower. We then proceeded to try these settings with more runs

Even though we knew that if we added more runs the difference would be just on the Latency and the throughput would be the same since the number of threads and the amount of data is the same, so we just did it locally.

Number of workloads threads = 10
Number of total runs = 10000000
Average Latency = 6679,919 milliseconds
Aggregate goodput = 7.712 $*10^{-4}$ transactions per nanosecond

Like expected the values were basically the same.
So now we are going to change the number of workloads threads but maintain an RPC and binary environment with 1000 runs which was the best parameters for us. We will also change the aggregate of goodput to a more appropriate scale : transactions per millisecond

Clients	10	12	14	16	18	20
	22	24	26	28		
Latency(ms)	9431,58	11276.82	12591,97	13313,59	14932,95	15317,26
	16743,44	17920,97	18003,75	20208,75		

Clients	10	12	14	16	18	20	22	24	26	28
Goodput (transaction/ms)	6,34	6,39	6,70	7,22	7,24	7,87	7,89	8,07	8,15	8,33

For different addresses we have:

Clients	10	12	14	16	18	20	22
	24	26	28				
Latency	9558,91	11543,89	12120,43	13002,23	14767,29	1521 0,05	16559,10
(ms)	17909,58	19228,01	20677,01				

Clients	10	12	14	16	18	20	22	24	26	28
Goodput	6,27	6,24	6,91	7,32	7,57	7,85	7,96	8,04	8,11	8,12
(transaction/ms)										

3.2

As expected as the number of clients increases the latency increases immensely while the throughput does it a lot less significantly. Since we are using RPC and we are creating more and more threads it is natural that the processor will take more time to assist every process. While on the throughput since the number of clients is bigger more transactions are needed per second so the slight increase. The latency numbers are not even bigger because as said in question one we used binary serialization which does not let it increase too much.

The difference between the curves on both cases it is not significant even though it was done across the address space, but it should have been different on the latency since some addresses have more latency and of course if you change the latency the throughput should change as well.

Note: The graphs of this question are done in the other pdf, because we had some problems putting them here with LaTeX, we hope that will not diminish our grade.

3.3

They are reliable specially if used with RPC, since it reproduces with high fidelity what it is going to happen in the bookstore every day. Even though they are good metrics to calculate some points on this system (they have all the characteristics of good performance metric such as linearity,reliability, repeatability, easiness of measurement , consistency and independence) they are not enough since they do not consider lot of factors, to do it we needed more specific tests such as testing the clock rate or used one of the processor test to calculate the number of instructions done per second.