Project1: Finding the ideal LRU_scan_depth

Siyeol Choi
2019315530

1. INTRODUCTION

This lab experiment observes how does innodb_lru_scan_depth affect the operation method of buffer manager and investigates the ideal lru_scan_depth regarding its tpmC and page miss scenario. The experiment is the ensemble of past experiments on "Monitoring Buffer Miss Scenario in MySQL/InnoDB" and "Monitoring performance metrics by changing the buffer sizes".

2. METHODS

Since LRU scan depth can be modified easily through changing variables in my.cnf, varying innodb_lru_scan_depth along 128 256 512 1024 2048 4096 8192 can give out different results.

While running the TPC-C benchmark, used iostat with -mx option in order to give out a summarized statistic of the iostat. Also, monitored and compared TpmC among the scan depth.

By logging with the fprintf stderr calculated the ratio of step 1,2,3.

3. Theoretical Analysis

By browsing the code base of buf0lru.cc and buf0flu.cc, were able to understand the flow of srv_LRU_scan_depth fetched from innodb_lru_scan_depth in my.cnf.

How innodb_lru_scan_depth value is fetched from the cnf file, applied into .cc code, and affects the operation method of buffer manager is as follows.

1) handler/ha_innodb.cc file fetches the innodb_lru_scan_depth from the my.cnf

2) fetched value is saved as srv_LRU_scan_depth locaed at buf0lru.cc and buf0flu.cc

3) in buf0lru.cc, we can locate the srv_LRU_scan_depth by navigating buf_LRU_get_free_block() → buf_LRU_scan_and_free_block(buf_pool, n_iterations > 0) → buf_LRU_free_from_common_LRU_list() → buf_LRU_free_from_unzip_LRU_list. In the function, the iteration among blocks is scanned until it reaches the scan depth.

4) in buf0flu.cc, we can locate the srv_LRU_scan_depth by navigating pc_flush_slot() → buf_do_LRU_batch() → buf_flush_LRU_list_batch(). In the function, it iterates until the length of the free pages in the buffer pool reaches the sum of srv_LRU_scan_depth and withdraw_depth.

4. Performance Evaluation

4.1 Experimental Setup

[System Setup]

MySQL version 5.7.33

OS Ubuntu 20.04.1 LTS (GNU/Linux 4.19.104-microsoft-standard

CPU model Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

cpu cores 4

Vendor: Msft

Product: Virtual Disk

Revision: 1.0

Compliance: SPC-3

User Capacity: 274,877,906,944 bytes [274 GB]

Logical block size: 512 bytes

Physical block size: 4096 bytes

[Benchmark setup]

Type Configuration

DB size 2GB (20 warehouse)

Buffer Pool Size 100M

Benchmark Tool tpcc-mysql

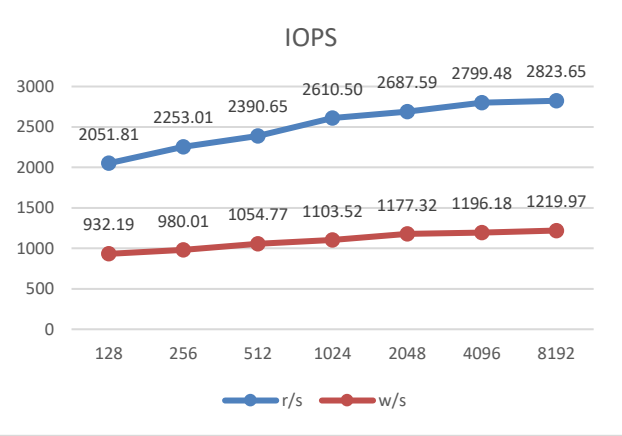
Runtime 1200s

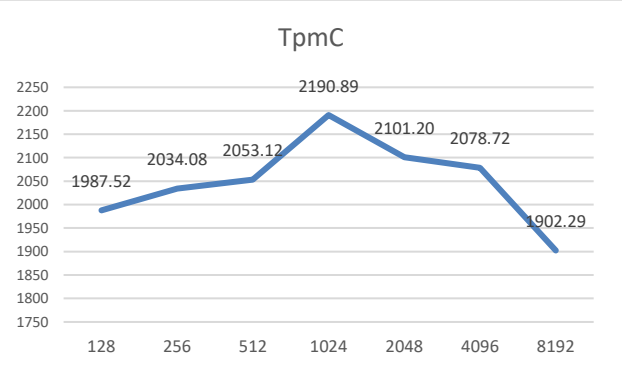
Connections 8

4.2 Experimental Results

[IOstat and TpmC among scan_depth]

Scan_depth	TpmC	r/s	w/s
128	1987.52	2051.81	932.19
256	2034.08	2253.01	980.01
512	2053.12	2390.65	1054.77
1024	2190.89	2610.50	1103.52
2048	2101.20	2687.59	1177.32
4096	2078.72	2799.48	1196.18
8192	1902.29	2823.65	1219.97



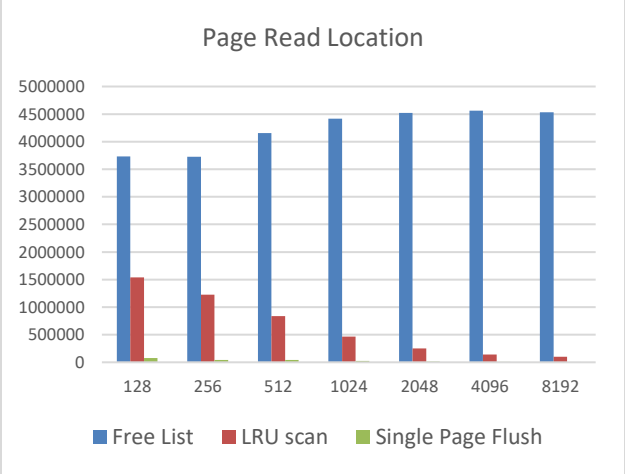


By summarizing and analyzing the change of TpmC and r/w IOPS among the variation of innodb_lru_scan_depth with the graph and the table, we found out that when "innodb_lru_scan_depth increases, IOPS increases. However, TpmC has different tendencies since while innodb_lru_scan_depth increases until 1024, which is a default value, TpmC increases, but the TpmC decreases if the innodb_lru_scan_depth exceeds 1024.

Also, according to the result, we can figure out the fact that the default value 1024 for innodb_lru_scan_depth is the ideal value. Since the TpmC is the number of completed transactions per minute, having the highest TpmC means that it has the highest transaction throughput.

[Page Read Location Raio among scan_depth]

Scan_depth	Free List	LRU scan	Single Page Flush
128	3733995 (63.91%)	1539671 (26.35%)	76694 (1.43%)
256	3728509 (68.32%)	1223978 (22.43%)	45050 (0.90%)
512	4156875 (75.63%)	836085 (15.21%)	40075 (0.80%)
1024	4415407 (82.41%)	467981 (8.73%)	22909 (0.47%)
2048	4522253 (86.54%)	250167 (4.79%)	12720 (0.27%)
4096	4561326 (88.63%)	142506 (2.77%)	8954 (0.19%)
8192	4532476 (89.45%)	102640 (2.03%)	5218 (0.11%)



In this experiment, we conducted a research on how the victim page is selected upon buffer miss along the increase of the innodb_lru_scan_depth. When the innodb_lru_scan_depth increases, the ratio of step1, getting free block from free list increased. Meanwhile the ratio of step2 and step3, getting free block by LRU scan and single page flush, decreased.

This result conveys that the deeper the scan depth is, there are higher probability of having a free list so the need of scanning for LRU tail and flushing the single dirty page for the victim page decreases.

5. Conclusion

Through the research conducted by modifying the LRU scan depth, we were able to find out that the ideal LRU scan depth in the transaction throughput perspective is 1024, the default value, since TpmC is the highest.

On top of that, when the scan depth gets deeper, the ratio of getting free block from the free list increases while the ratio of getting the block by LRU scan and single page flush decreases.

6. REFERENCES

[1] Buffer miss scenario https://github.com/LeeBohyun/mysql-tpcc/blob/master/buffer_manager/buffer_miss_scenario_monitoring.md

[2] Report format https://github.com/LeeBohyun/SWE3033-F2022/blob/main/report-guide.md

[3] Scan Depth explanation https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_lru_scan_depth