Aurora MySQL Write IO Optimization Research

03-08-2019

Summary:

The purpose of this research paper and local MySQL 5.7 experiments is to try to reverse-engineer the inner workings of how AWS charges Aurora MySQL write IOs in order develop some practical tips for reducing this cost category.

Research:

“Write IOs are only consumed when pushing transaction log records to the storage layer for the purpose of making writes durable. **Write IOs are counted in 4KB units**. For example, a transaction log record that is 1024 bytes will count as one IO operation. However, concurrent write operations whose transaction log is less than 4KB can be batched together by the Aurora database engine in order to optimize I/O consumption. Unlike traditional database engines Amazon Aurora never pushes modified database pages to the storage layer, resulting in further IO consumption savings.” (https://aws.amazon.com/rds/aurora/faqs/)

“A relational database, at core, is a redo log that always advances, even to apply the rollback of a transaction. The data pages that comprise the database are really just point-in-time cached instantiations of the application of the redo log.” (https://aws.amazon.com/blogs/database/amazon-aurora-under-the-hood-quorum-reads-and-mutating-state/)

Quote from Anurag Gupta (Vice president of AWS) in response to Percona article (https://www.percona.com/blog/2015/11/16/amazon-aurora-looking-deeper/): "Buffer page writes are zero because **Aurora only writes log pages to the storage tier**. That tier generates data block records on its own (similar to how log-structured storage systems work). No checkpointing or writing of dirty data blocks out of cache is required.”

Aurora non-modifiable database configs of interest:
innodb_change_buffering = none
innodb_checksum_algorithm = none
innodb_checksums = OFF
innodb_doublewrite = OFF
innodb_flush_log_at_trx_commit = 1 (Logs are written and flushed to disk at each transaction commit. Full ACID compliance.)

“Q. Amazon Aurora replicates each chunk of my database volume six ways across three Availability Zones. Does that mean that my effective storage price will be three or six times what is shown on the pricing page?

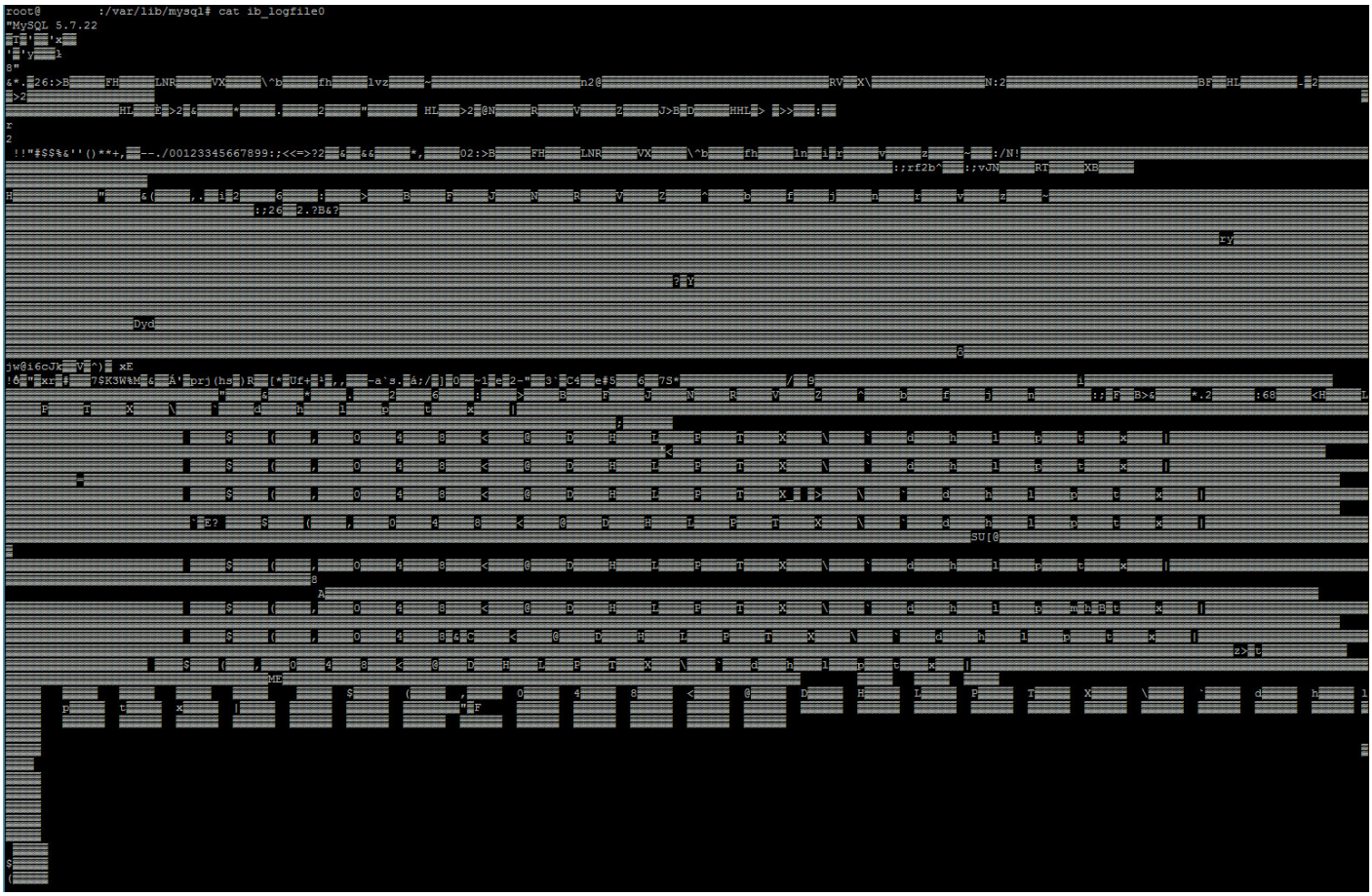
No. Amazon Aurora’s replication is bundled into the price. **You are charged based on the storage your database consumes at the database layer**, not the storage consumed in Amazon Aurora’s virtualized storage layer.” (https://aws.amazon.com/rds/aurora/faqs/)

What is the MySQL Redo Log?

ib_logfile0 and ib_logfile1:

```
root@        :/var/lib/mysql# ls -lh
total 2.1G
-rw-r----- 1 mysql mysql   56 Mar  8 16:13 auto.cnf
-rw------- 1 mysql mysql 1.7K Mar  8 16:13 ca-key.pem
-rw-r--r-- 1 mysql mysql 1.1K Mar  8 16:13 ca.pem
-rw-r--r-- 1 mysql mysql 1.1K Mar  8 16:13 client-cert.pem
-rw------- 1 mysql mysql 1.7K Mar  8 16:13 client-key.pem
-rw-r----- 1 mysql mysql  410 Mar  8 16:13 ib_buffer_pool
-rw-r----- 1 mysql mysql  12M Mar  8 16:17 ibdata1
-rw-r----- 1 mysql mysql 1.0G Mar  8 16:17 ib_logfile0
-rw-r----- 1 mysql mysql 1.0G Mar  8 16:13 ib_logfile1
-rw-r----- 1 mysql mysql  12M Mar  8 16:17 ibtmp1
drwxr-x--- 2 mysql mysql 4.0K Mar  8 16:13 mysql
drwxr-x--- 2 mysql mysql 4.0K Mar  8 16:13 performance_schema
-rw------- 1 mysql mysql 1.7K Mar  8 16:13 private_key.pem
-rw-r--r-- 1 mysql mysql  451 Mar  8 16:13 public_key.pem
-rw-r--r-- 1 mysql mysql 1.1K Mar  8 16:13 server-cert.pem
-rw------- 1 mysql mysql 1.7K Mar  8 16:13 server-key.pem
drwxr-x--- 2 mysql mysql  12K Mar  8 16:13 sys
drwxr-x--- 2 mysql mysql 4.0K Mar  8 16:17 test_schema
root@        :/var/lib/mysql#
```

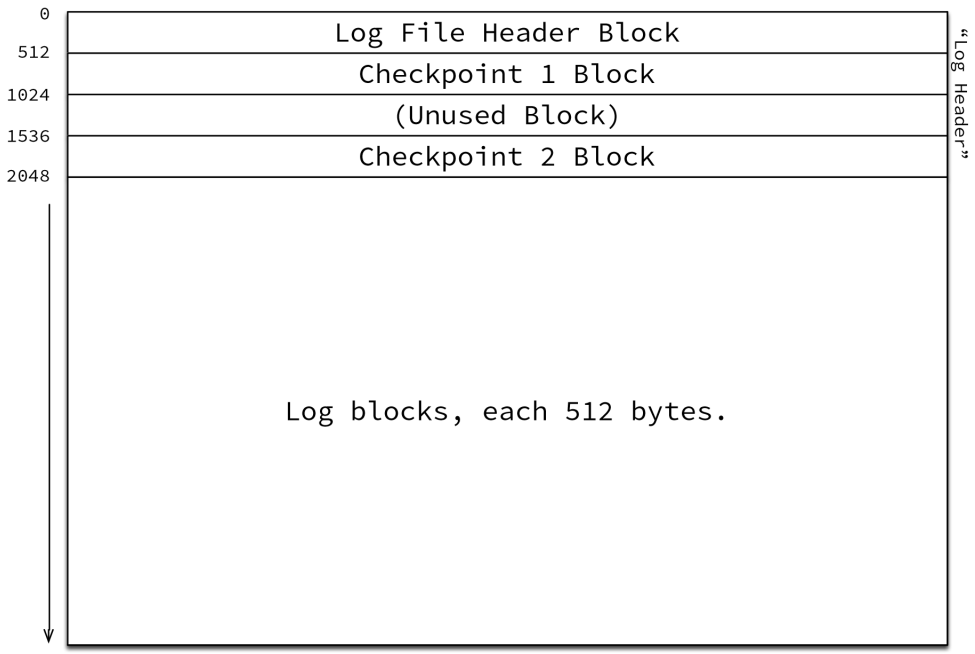
cat ib_logfile0:



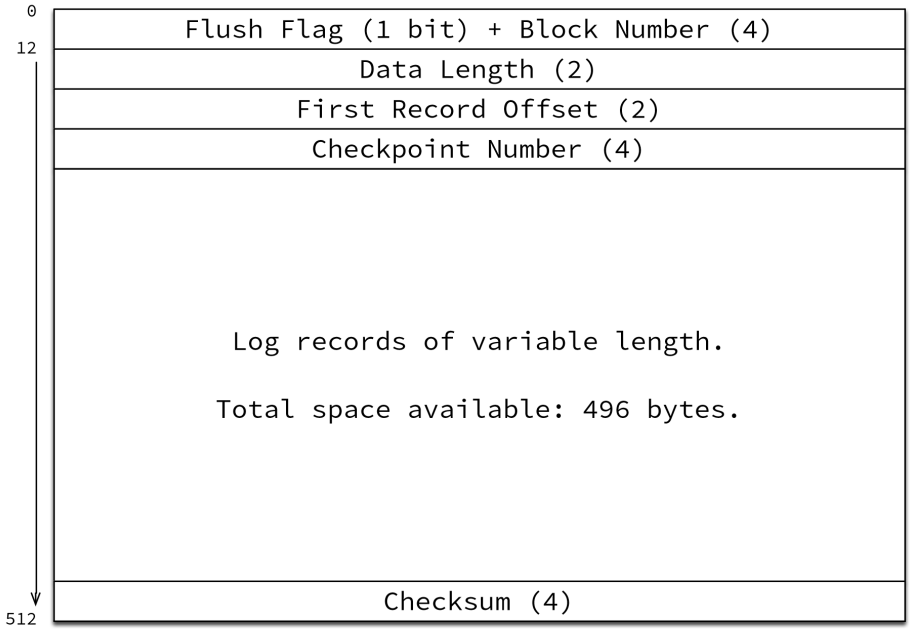
hexdump -C ib_logfile0:

```
root@        :/var/lib/mysql# hexdump -C ib_logfile0
00000000  00 00 00 01 00 00 00 00  00 00 00 00 00 00 22 00  |..............".|
00000010  4d 79 53 51 4c 20 35 2e  37 2e 32 32 00 00 00 00  |MySQL 5.7.22....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 0c 83 54 bb  |..............T.|
00000200  00 00 00 00 00 00 00 06  00 00 00 00 00 27 92 90  |.............'..|
00000210  00 00 00 00 00 27 78 90  00 00 00 00 01 00 00 00  |.....'x.........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000003f0  00 00 00 00 00 00 00 00  00 00 00 00 d0 0c cd 90  |................|
00000400  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000600  00 00 00 00 00 00 00 07  00 00 00 00 00 27 93 04  |.............'..|
00000610  00 00 00 00 00 27 79 04  00 00 00 00 01 00 00 00  |.....'y.........|
00000620  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000007f0  00 00 00 00 00 00 00 00  00 00 00 00 8a e0 cb c5  |................|
00000800  80 00 00 12 02 00 00 0c  00 00 00 01 38 00 00 00  |............8...|
00000810  00 00 00 22 0c 3b 00 01  1b 00 01 1f 3b 00 00 02  |...".;......;...|
00000820  00 00 00 18 08 04 00 00  00 26 00 04 00 00 00 2a  |.........&.....*|
00000830  00 04 00 00 00 2e 83 00  04 00 00 00 32 00 04 00  |............2...|
00000840  00 00 36 00 04 00 00 00  3a 00 04 00 00 00 3e 00  |..6.....:.....>.|
00000850  04 00 00 00 42 f0 ff ff  ff ff 02 00 00 00 46 00  |....B.........F.|
00000860  04 00 00 00 48 f0 ff ff  ff ff 02 00 00 00 4c 00  |....H.........L.|
00000870  04 00 00 00 4e 00 04 00  00 00 52 f0 ff ff ff ff  |....N.....R.....|
00000880  02 00 00 00 56 00 04 00  00 00 58 f0 ff ff ff ff  |....V.....X.....|
00000890  02 00 00 00 5c 00 04 00  00 00 5e 00 04 00 00 00  |....\.....^.....|
000008a0  62 f0 ff ff ff ff 02 00  00 00 66 00 04 00 00 00  |b.........f.....|
000008b0  68 f0 ff ff ff ff 02 00  00 00 6c 00 04 00 00 00  |h.........l.....|
000008c0  76 00 04 00 00 00 7a f0  ff ff ff ff 02 00 00 00  |v.....z.........|
000008d0  7e 00 04 00 00 00 80 f0  ff ff ff ff 02 00 00 00  |~...............|
000008e0  84 00 04 00 00 00 86 00  04 00 00 00 8a f0 ff ff  |................|
000008f0  ff ff 02 00 00 00 8e 00  04 00 00 00 90 f0 ff ff  |................|
00000900  ff ff 02 00 00 00 94 00  08 00 00 00 6e 00 00 00  |............n...|
00000910  00 01 04 00 00 00 32 40  04 00 00 00 ae f0 ff ff  |......2@........|
00000920  ff ff 04 00 00 00 b2 f0  ff ff ff ff 04 00 00 00  |................|
00000930  b6 f0 ff ff ff ff 04 00  00 00 ba f0 ff ff ff ff  |................|
00000940  04 00 00 00 aa 01 01 00  00 00 ae 80 fe 01 00 00  |................|
00000950  00 ae 80 fa 04 00 00 00  aa 02 04 00 00 00 52 00  |..............R.|
00000960  02 00 00 00 56 80 9e 04  00 00 00 58 00 02 00 00  |....V......X....|
00000970  00 5c 80 9e 04 00 00 00  9e f0 ff ff ff ff 02 00  |.\..............|
00000980  00 00 a2 00 04 00 00 00  a4 f0 ff ff ff ff 02 00  |................|
00000990  00 00 a8 00 04 00 00 00  4e 01 04 00 00 00 3a 02  |........N.....:.|
000009a0  04 00 00 00 32 80 80 04  00 00 00 d6 f0 ff ff ff  |....2...........|
000009b0  ff 04 00 00 00 da f0 ff  ff ff ff 04 00 00 00 de  |................|
000009c0  f0 ff ff ff ff 04 00 00  00 e2 f0 ff ff ff ff 04  |................|
000009d0  00 00 00 d2 01 04 00 00  00 42 00 02 00 00 00 46  |.........B.....F|
000009e0  80 c6 04 00 00 00 48 00  02 00 00 00 4c 80 c6 04  |......H.....L...|
000009f0  00 00 00 c6 f0 ff ff ff  ff 02 00 00 cb 97 f8 32  |...............2|
00000a00  00 00 00 13 02 00 00 00  00 00 00 01 00 ca 00 04  |................|
00000a10  00 00 00 cc f0 ff ff ff  ff 02 00 00 00 d0 00 04  |................|
00000a20  00 00 00 3e 01 04 00 00  00 32 80 c0 04 00 00 00  |...>.....2......|
00000a30  fe f0 ff ff ff ff 04 00  00 01 02 f0 ff ff ff ff  |................|
00000a40  04 00 00 01 06 f0 ff ff  ff ff 04 00 00 01 0a f0  |................|
00000a50  ff ff ff ff 04 00 00 00  fa 01 04 00 00 00 ee 00  |................|
00000a60  02 00 00 00 f2 80 c6 04  00 00 00 f4 f0 ff ff ff  |................|
00000a70  ff 02 00 00 00 f8 00 04  00 00 00 48 00 02 00 00  |...........H....|
00000a80  00 4c 80 ee 04 00 00 00  cc 00 02 00 00 00 d0 80  |.L..............|
00000a90  ee 04 00 00 00 3e 02 04  00 00 00 32 81 00 04 00  |.....>.....2....|
00000aa0  00 01 26 f0 ff ff ff ff  04 00 00 01 2a f0 ff ff  |..&.........*...|
00000ab0  ff ff 04 00 00 01 2e f0  ff ff ff ff 04 00 00 01  |................|
```

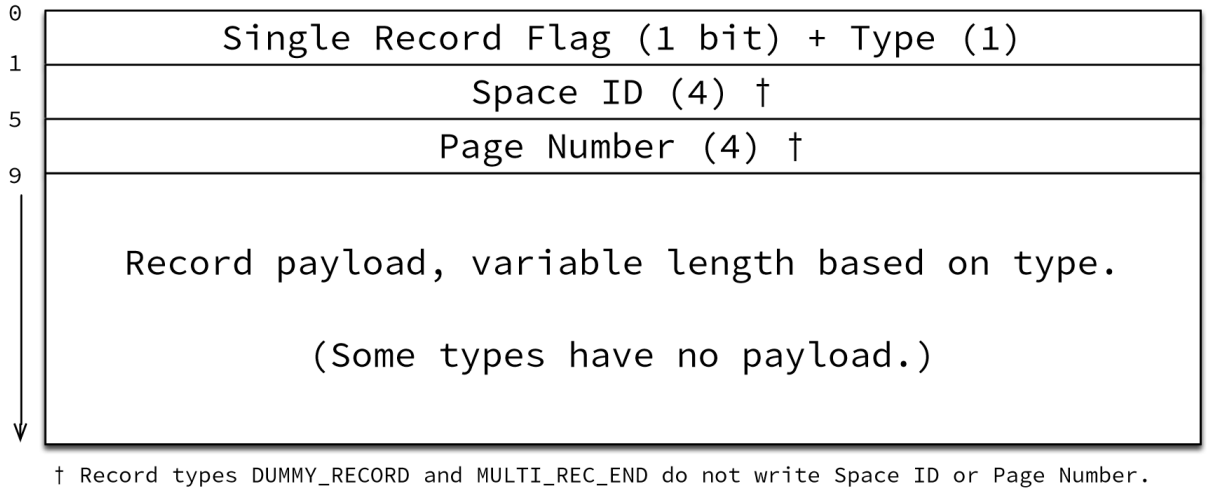
Log File 0 Overview



Log Block



Log Record Overview



IO experiments/tests:

Note: Aurora performs a certain number of background writes, in my test environment this totaled an average of 16,080 writes per hour or 1,340 writes per 5 minute interval. This average value was subtracted out of all reported test values.

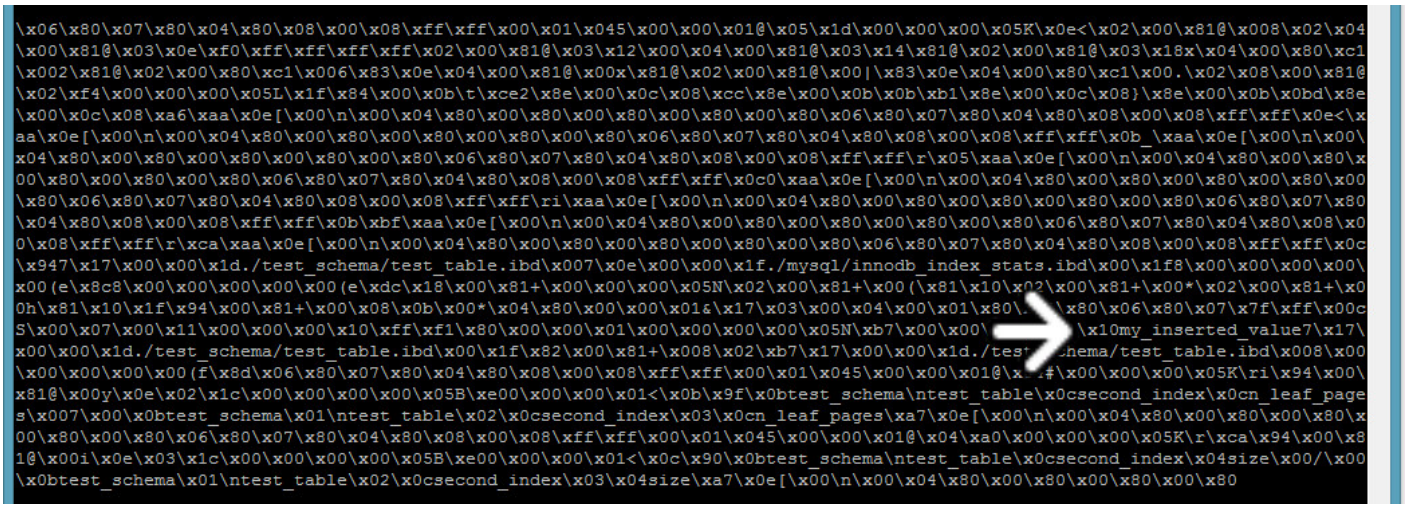
Note: Properly evaluating write IOs requires waiting 2 hours between tests since Amazon changes when they actually generate a report value every hour. Also, the best interval for reporting is the 5 minute interval, but keep in mind this value is merely the hourly value divided by 12.

Note: Here is an example of how to retrieve write IO metrics from the command line, which also allows you to go further back than the AWS front end.

```
aws cloudwatch get-metric-statistics
 --namespace AWS/RDS
 --metric-name VolumeWriteIOPs
 --statistics Sum
 --dimensions Name=DbClusterIdentifier,Value=your-cluster-name Name=EngineName,Value=aurora
 --period 300
 --start-time 2019-03-01T08:00:00Z
 --end-time 2019-03-02T08:00:00Z;
```

Test 1: Does inserting into an indexed column cost more IOs than a non-indexed column?
Control: 64,588
Test: 141,205 (219% of control)
Conclusion: Yes, inserting into an indexed column costs **more** IOs than a non-indexed column.

Before screenshot:



After screenshot:

Test 2: Does inserting a 255 character value cost more IOs than storing a 32 character value in the same column type of VARCHAR(255)?
Control: 141,205
Test: 202,412 (143% of control)
Conclusion: Yes, inserting longer values into a VARCHAR field costs **more** IOs.

Test 3: Does inserting rows in batches of 100,000 rows per INSERT statement cost less IOs than running single INSERT statements?
Control: 141,205 (2nd run dropped index: 56,264)
Test: 100,266 (2nd run dropped index: 36,150) (71% and 64% of control)
Conclusion: Yes, inserting multiple rows per INSERT statement costs **less** IOs than single INSERT statements.

Test 4: Does sleeping between INSERT statements increase IOs?
Control: 141,205
Test: 182,340 (129% of control)
Conclusion: Yes, sleeping between INSERT statements costs **more** IOs. Conversely, running multiple INSERT statements in quick succession decreases IOs, probably due to batching of records.

Test 5: Does inserting a 32 character string into a VARCHAR(255) cost more IOs than a CHAR(32)?
Control: 61,870
Test: 55,251 (89% of control)
Conclusion: No, inserting a 32 character string into a VARCHAR(255) does not cost more IOs than a CHAR(32), it's actually slightly **less**.

Test 6: Does **updating** an indexed column value cost more IOs than a non-indexed column?
Control: 289,098
Test: 412,710 (143% of control)
Conclusion: Yes, updating an indexed column value costs **more** IOs than a non-indexed column.

Test 7: Does **deleting** rows cost more IOs than truncating a table?
Control: 65,040
Test: 210,433 (324% of control)
Conclusion: Yes, deleting rows costs **more** IOs than truncating a table.
Confirmed by redo log: A DELETE statement must flag every row as deleted creating many log entries.

Test 8: Does **altering** an indexed column cost more IOs than a non-indexed column if values are not truncated?
Control: 237,553
Test: 233,993 (99% of control)
Conclusion: No, altering an indexed column costs **the same** as a non-indexed column.

Test 9: Does updating rows via a single "UPDATE…WHERE id IN(id,id,id)" cost less IOs than individual update statements?
Control: 75,584
Test: 73,296 (97% of control)
Conclusion: Yes, updating rows via an "UPDATE…WHERE id IN(id,id,id)" costs **slightly less** IOs than individual update statements.

Test 10: Does updating rows via an “UPDATE…WHERE value IN(value,value,value)” cost more IOs than an “UPDATE…WHERE id IN(id,id,id)”?
Control: 73,296
Test: 73,589 (100% of control)
Conclusion: No, updating rows via an “UPDATE…WHERE value IN(value,value,value)” costs the **same** IOs as an “UPDATE…WHERE id IN(id,id,id)”.
Confirmed by redo log: The **values** of an updated or deleted row are not logged in the redo log to identify the row of the transaction, the **primary key** column is used.

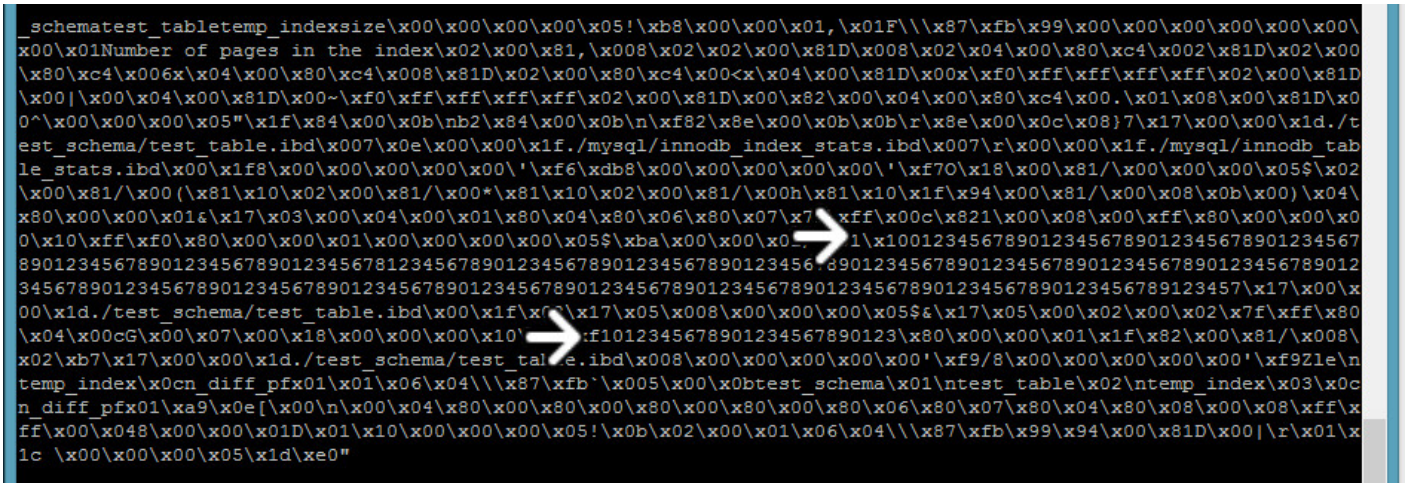
Test 11: Does an “INSERT IGNORE INTO table VALUES(unique,unique,unique)” cost more IOs if all unique values already exist (no new inserts)?
Control: 141,305
Test: 143,974 (102% of control)
Conclusion: No, an “INSERT IGNORE” costs about the **same** IOs as the control when no new inserts are performed.

Test 12: Does enabling binlogging cost more IOs when performing INSERT statements?
Control: 81,218 (execution time 71 seconds)
Test: 259,644 (320% of control) (execution time 143 seconds, 201% of control)
Conclusion: Yes, enabling binlogging costs a **lot more** IOs when performing INSERT statements. Also, enabling binlogging **doubles** query execution time.

Test 13: Does enabling binlogging cost more IOs when performing UPDATE statements?
Control: 155,415
Test: 185,805 (120% of control)
Conclusion: Yes, enabling binlogging costs **more** IOs when performing UPDATE statements.

Redo Log Test 1: Does inserting a value into a field with an index of specified length less than the max of the column reduce the size of the transactions in the redo log?
Result: Yes, the smaller the index size the smaller the transactions in the redo log for INSERTS/UPDATES on the indexed field. Please note that if the variable “innodb_large_prefix” is set to “OFF” (default prior to MySQL 5.7.7), the max length of a single column index is 767 bytes, with utf8mb4 encoding (max 4 bytes per character) this limits an index to 191 characters.

Inserted string in screenshot below is “0123456789” repeated over and over:



Redo Log Test 2: Does creating a view or inserting/updating an underlying table's data increase the number of transactions in the redo log?
Result: No, creating a VIEW or inserting/updating an underlying table's data does NOT create additional transactions in the redo log, a VIEW is a virtual table.

Conclusion:

Practical recommendations to reduce Aurora MySQL write IOs:

- Since Aurora "write IOs are counted in 4KB units" and "concurrent write operations whose transaction log is less than 4KB can be batched together", the general idea behind reducing write IOs is to reduce the size of the redo log records. (https://aws.amazon.com/rds/aurora/faqs/)

- Don't perform any unnecessary INSERT statements. Use caching like Redis where applicable for things like metrics and push log records to ELK or S3 buckets where applicable. Think twice before writing any INSERT statement to see if there are better alternatives to store that data.

- Don't insert any unnecessary or redundant columns. Evaluate the use of every column and eliminate those that are not used by the application.

- Decrease the max length of any VARCHAR field to the minimum required length. Do we really need 1,000 characters of a product title? Do we even need 255 characters? Can we truncate longer values and still perform the necessary tasks?

- Remove any unnecessary indexes, especially on columns with high writes/low reads.

- Use the smallest specified length possible on VARCHAR single and multi-column indexes, especially on columns with high writes/low reads:
Example:
Index the first 24 characters of the "username" column:
ALTER TABLE users ADD INDEX username_index (username(24));
Laravel:
$table->index([DB::raw('username(24)')]);

- If possible, try to use an integer as the primary key column of all tables (default behavior). Otherwise, keep the length of the primary column as short as possible since this is used to identify every update/delete operation, and it creates an index for every inserted row.

- Since UPDATE statements log **both** the old and new values in the redo log, avoid frequently updating long VARCHAR fields.

- Disable binlogging if possible. Definitely do NOT enable binlogging on any Staging or Development Aurora instances. If you clone a Production instance to use for Staging or Development, make sure you also change the cluster parameter group to one that has a "binlog_format" parameter value of "OFF" (may require a reboot to apply). Explore potential alternatives to binlogging for Production instances.

Helpful resources and links:

https://blog.jcole.us/innodb/
https://github.com/jeremycole/innodb_diagrams/blob/master/images/InnoDB_Log_Structures.pdf
https://dev.mysql.com/doc/dev/mysql-server/8.0.11/PAGE_INNODB_REDO_LOG.html
https://dev.mysql.com/doc/dev/mysql-server/8.0.11/PAGE_INNODB_REDO_LOG_FORMAT.html
https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-logging.html
https://github.com/KasperFridolin/mysql_forensics/blob/master/iblogfile_parser.py
https://www.percona.com/blog/2015/11/16/amazon-aurora-looking-deeper/
https://www.slideshare.net/AmazonWebServices/amazon-aurora
https://forums.aws.amazon.com/thread.jspa?messageID=835303#835303
https://dba.stackexchange.com/questions/142224/in-mysql-how-exactly-does-data-flows-from-query-to-disk
https://www.sba-research.org/wp-content/uploads/publications/WSDF2012_InnoDB.pdf
https://aws.amazon.com/blogs/database/amazon-aurora-under-the-hood-quorum-reads-and-mutating-state/