Speeding up MariaDB on Fusion-IO

Jan Lindström
Principal Engineer
SkySQL - The MariaDB Company
Disclaimer

- This work is result of co-operation with Fusion-IO especially
  - Dhananjoy Das
  - Torben Mathiasen
Contents

- Background
- Proposed method
- Benchmark results
- Current status
Background

- InnoDB doublewrite
- InnoDB compression
InnoDB DoubleWrite

- Writes data twice
  - First page is written to doublewrite buffer
  - This write is synchronously flushed
  - Then page is written to tablespace
- Required for ACID compliance
InnoDB Compression

- InnoDB stores uncompressed data in 16K pages
- 16K pages are compressed into a fixed compressed page size of 1K, 2K, 4K, 8K
- Compressed page size is chosen at table creation
- Compression is performed using regular software compression libraries (zlib)
- Table updates appended to Page Modification Log (mlog) at the end of the compressed (8K) page
- When mlog gets full, page is recompressed
InnoDB compression drawbacks

- **Memory**
  - **Space:** Both uncompressed and compressed pages stored in buffer pool
  - **Access:** Updates are applied to both copies in memory
- **CPU consumption**
  - Software compression libraries (decompress on read from disk, recompress on split)
  - Split & Recompress & Rebalance when mlog overflows
- **Capacity benefit**
  - Fixed compression page size - sets a bound on compression benefit
- Poor adoption
Proposed solution

- Atomic writes
- Compressed tables
- TRIM
Atomic Writes

- ioMemory NVMFS
  - Open, read, write, seek, close.
  - Plus, new primitives to exploit characteristics of non-volatile memory
    - Basic write
    - atomic write
    - Transactional write.
    - Persistent Trim
- DFS_IOCTL_ATOMIC_WRITE_SET_IOW(0x95, 2, uint)
- ioctl(file, DFS_IOCTL_ATOMIC_WRITE_SET, &atomic_option)
Page compression (NVM compression)

- Only store uncompressed 16KB pages in buffer pool (memory)
- When page is modified, it is compressed just before it is written (fil layer) and only a compressed size (aligned to sector boundary) is written.
- When page is read, it is decompressed before it is on buffer pool.
- Implemented by using a new page type.
- This solution allows different compression algorithms used. Currently zlib and lz4 supported.
trim unused 512B sectors in compressed page.

- `fallocate(file, FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, off, trim_len);`
- NVMFS file system reports that less space is used on media
- 16K uncompressed sector is now compressed & ‘thin-provisioned’ down to e.g 3.5K on Flash
Examples

- CREATE TABLE A(B INTEGER) ENGINE=InnoDB
  PAGE_COMPRESSED=1;
- CREATE TABLE B(C INTEGER) ENGINE=InnoDB
  ATOMIC_WRITES=ON;

CREATE TABLE t3 (a int KEY, b int) DATA
  DIRECTORY='/dev/fioa' PAGE_COMPRESSED=1
  PAGE_COMPRESSION_LEVEL=4
  ATOMIC_WRITES='ON';
Benchmark results - TPC-C

TPC-C like workload
MariaDB 10
1000 warehouses - 75GB DRAM

New Order TX

MySQL uncompressed
MySQL compression
Fusion-io Compression
Benchmark results - Linkbench
Benchmark conclusions

- Standard InnoDB Compression
  - 80% reduction in performance
  - 60% improvement in capacity
- Fusion Accelerated Compression
  - 10% reduction in performance
  - 60% improvement in capacity
  - 4x fewer writes to storage
Status

- Beta release of MariaDB-FusionIO 10.0.9 released 02-04-2014.
  - Both InnoDB and XtraDB storage engines supported
  - Change size ~13KLOC
Thank you

Q&A