

RealDB: Low-Overhead Database for Time-Sequenced Data Streams in Embedded Systems

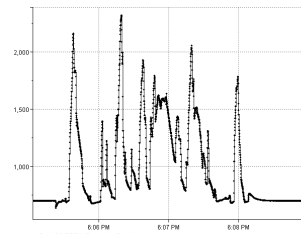
MS Project Defense
Jason Winnebeck
October 5, 2010

Coming Up

- Problem
 - Implementation
 - Results
 - Lessons Learned and Future Work
-

Problem

- Storage of high-frequency, ordered time series data from multiple sources
- Embedded environment
 - Low-powered hardware (sub Ghz ARM or x86)
 - Limited Space (0.5 to 2GB, solid state)
 - No knowledgeable operator at deployed location
 - Unreliable power source



Hypothesis

- Building a data storage solution specific to data streams can substantially improve performance over a “traditional” (relational) database engine for the embedded environment, while maintaining scalable performance in writing and recovery
 - $O(1)$ recovery time for a specific configuration
 - $O(n)$ size and time to write records
 - Ability to maintain a fixed size
-

Application Assumptions and Trade-offs

- Data is collected and written in order
 - Database engine is single thread and client
 - Truncation of records just written to the stream before a fault is acceptable for fast, unattended recovery
-

System Assumptions

- The operating system performs writes to the disk in the same order as RealDB performs them when operated in synchronous mode
 - When the system has a power fault, blocks (bounded by some finite, known size) previously written are unmodified
 - The data contained in the block being written to during a power fault is undefined on next start
-

Goals

- Unattended operation and availability; recovery runs within a fixed time for a given configuration
 - Durability: A failure does not cause loss of data written before a successful flush command
 - RealDB is scalable to arbitrarily large data sets.
 - Insert a single point (including delete): $O(1)$
 - Lookup single point: $O(\log n)$
 - Retrieve range: $O(n)$
 - Minimize write cycles to keep SSD wear to a minimum
 - Compact database size
 - Minimize CPU utilization
-

Relational Database Solutions

- Stream data can be recorded in a SQL table as timestamped rows
 - RealDB will be compared against:
 - MySQL InnoDB (5.1.46): transactional server
 - MySQL MyISAM: non-transactional server
 - Apache Derby (10.5.3.0_1): transactional embedded
-

File Format Design

- Fixed size file with blocks of a configurable size, which must be a multiple of physical block size
 - Laid out in sections:
 - File Header
 - Metadata Section – describes streams
 - Block Pool – manages block allocation and transactions
 - Data Index – preallocated, tracks blocks allocated to each stream
 - Data Section – contains raw stream data
-

Key Design Properties

- Data recorded in order eliminates a lot of indexing
 - Direct, embedded API eliminates SQL interface overhead
 - Fixed file size with fixed sections minimizes allocations
 - Backup blocks for transaction-free atomic changes in individual indices; never overwrite the only copy
 - Circular buffers keep modifications only at the ends: limits backup blocks, O(1) performance
 - Transaction logging only on data block allocation
 - Size management: overwrite oldest data block – bounded delete overhead
 - Works on any contiguous memory range: in-memory, pre-allocated file, or raw disk partition
-

RealDB Definition Language (RDL)

```
SET blockSize = 2048
SET fileSize = 204800
SET maxStreams = 3
SET dataBlockSize = 2

CREATE STREAM Test WITH ID 1 {
  value float NULL //will use SampledAlgorithm by default
}

CREATE STREAM CarSnapshots WITH ID 2 {
  rpm float WITH CODEC DeadbandAlgorithm PARAMS (deadband=50.0),
  speed float WITH CODEC DeadbandAlgorithm PARAMS (deadband=5),
  passengers uint8 WITH CODEC StepAlgorithm,
  driving boolean WITH CODEC StepAlgorithm
}
```

Benchmark Metrics

- Database size (assuming no filesystem overhead). For RealDB this measures the utilized space, since the datafiles files are a fixed size (50MiB and 100MiB).
 - DB startup and creation:
 - time
 - disk sectors reads/writes
 - DB load and shutdown:
 - time
 - disk sectors read/write
 - reads/writes disk milliseconds
 - user and kernel mode jiffies (including those of child processes)
-

Benchmark Dimensions

- Implementation
 - RealDB file-based
 - RealDB partition-based
 - Derby
 - MySQL MyISAM
 - MySQL InnoDB
- Size Management (Y/N)
- Number of Records (every 1M to 9M, then 9.2M)

Note: Not all Derby / InnoDB tests run due to extremely poor performance

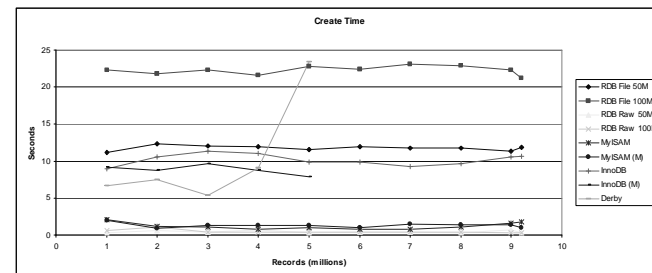
Benchmark Environment

- Ubuntu GNU/Linux 9.10 (Karmic)
 - kernel 2.6.31-21-generic
- Intel Core 2 2.4 GHZ E6600
 - (CPU frequency scaling left on)
- 2GiB RAM
- Java 1.6: OpenJDK 6b16-1.6.1-3ubuntu3
- USB 2.0 memory card reader with a 4GB CompactFlash card
 - Measured 5.7MiB/s write
 - Measured 6.8MiB/s read
- MySQL 5.1.46
 - Connector/J JDBC Driver 5.1.12
 - JDBC parameter rewriteBatchedStatements = true
- Derby 10.5.3.0_1

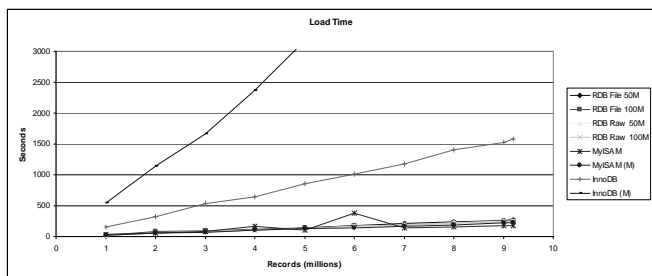
Benchmark Process

- Insert rows in time order – “playback” of the sensor data
- For SQL:
 - JDBC PreparedStatements (at start)
 - Batch inserts every 1000 rows
 - Size management: After each batch, delete rows older than 10,000 seconds

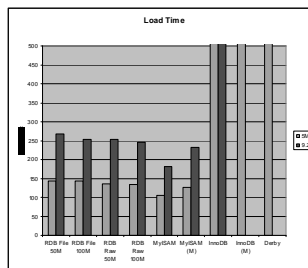
Results – Creation Time



Results – Load Time



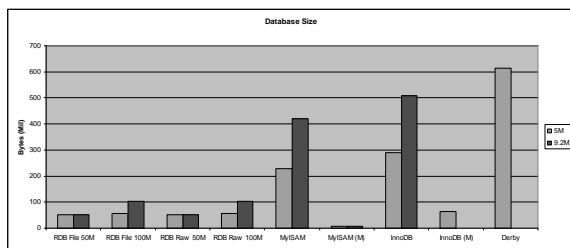
Results – Load Time



Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	-36.6%	-39.8%	-26.8%	-36.2%
MyISAM (M)	-12.9%	-15.6%	-6.8%	-9.2%
InnoDB	83.2%	84.1%	84.4%	84.5%
InnoDB (M)	95.5%		95.7%	
Derby	99.3%		99.4%	

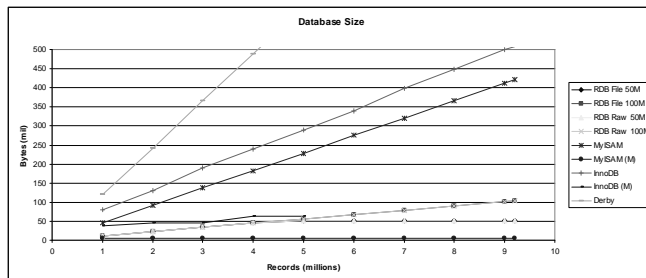
Implementation	5M Records	9.2M Records
InnoDB	859 s	1589 s
InnoDB (M)	3174 s	
Derby	21971 s	

Results – Database Size

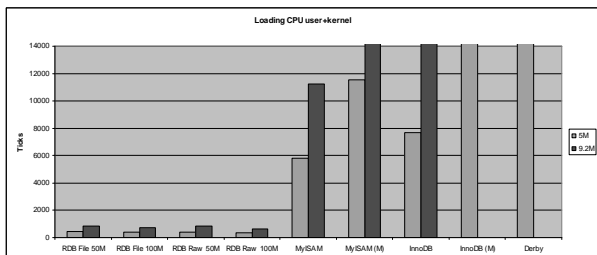


Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	75.3%	75.4%	75.3%	75.4%
MyISAM (M)	-727.6%	-721.3%	-727.6%	-721.3%
InnoDB	80.5%	79.6%	80.5%	79.6%
InnoDB (M)	16.7%		16.7%	
Derby	90.8%		90.8%	

Results – Database Size

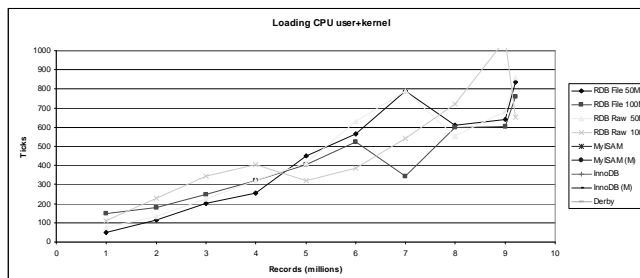


Results – CPU Utilization

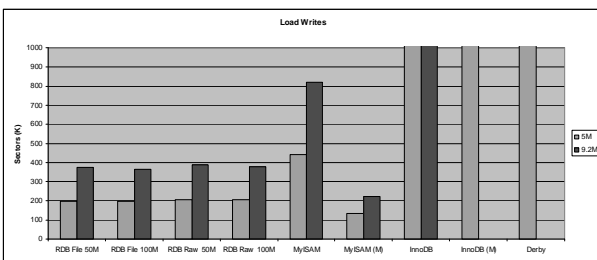


Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	93.0%	93.3%	94.5%	94.2%
MyISAM (M)	96.1%	96.1%	96.4%	96.0%
InnoDB	94.7%	95.6%	95.8%	96.2%
InnoDB (M)	98.9%		99.0%	
Derby	99.4%		99.5%	

Results – CPU Utilization

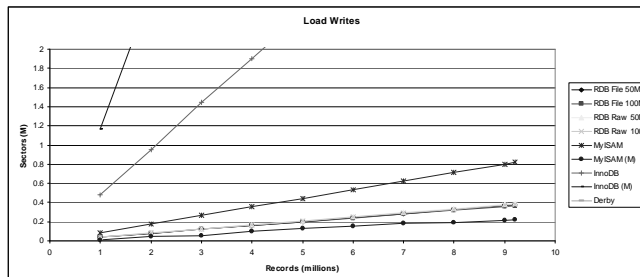


Results – Disk Writes



Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	55.3%	55.4%	53.7%	53.9%
MyISAM (M)	-49.9%	-67.5%	-55.0%	-74.1%
InnoDB	91.6%	91.7%	91.4%	91.4%
InnoDB (M)	97.0%		96.9%	
Derby	98.6%		98.5%	

Results – Disk Writes



Results – Summary

- Total load time is reduced by 95%-96%
 - Compared to the fastest reliable RDBMS implementation, InnoDB
 - Database size is reduced by 75%-81%
 - compared to the smallest RDBMS, MyISAM, and 81% compared to InnoDB, the smallest reliable RDBMS
 - CPU utilization is reduced by 93%-99%
 - compared to MyISAM and InnoDB
 - Loading writes reduced by 54%-91%
 - Without size management, versus MyISAM (54%) and InnoDB (91%)
 - Sectors written when size management is required is inconclusive because it was not possible to replicate the same delete methods in SQL as was used in RealDB, but MyISAM may require about half as many writes. InnoDB requires more writes when size management is required.
-

Conclusions

- RealDB achieves its goal of significantly improved performance over the compared RDBMS implementations for the problem of data streams.
 - RealDB achieves the complexity requirements
 - Bounded recovery time
 - Bounded insert
 - Linear load (disk and CPU)
 - Other advantages versus traditional RDBMS
 - SQL cannot guarantee DB size limit
 - Does not require file system (which can fail)
 - Disadvantages
 - Fixed DB structure/size impacts upgrades
 - Forced flush can leave unused "slack space" in data blocks
 - Would probably need to complement an embedded SQL, but RealDB allows it to be light-weight
-

Lessons Learned

- Transactions were needed, which added delay and complexity
 - Data storage and data compression too much for one project; choice was to focus on the former
 - Version control comments, notes, code comments critical when working sporadically
 - There must be some limit to redesign or shift directions on new knowledge; at some point, document and continue
-

Future Work

- Modify DB after creation
 - Multiple outstanding transactions
 - Fix "slack space" on flush
 - Investigate why RealDB takes more time but uses less CPU and writes (effective resource use)
 - Memory usage and read metrics
 - Improvements to selecting block to delete
 - Compare to other RDBMS like SQLite
 - Compare to non-RDBMS alternatives
-