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

MS Project Defense

Jason Winnebeck

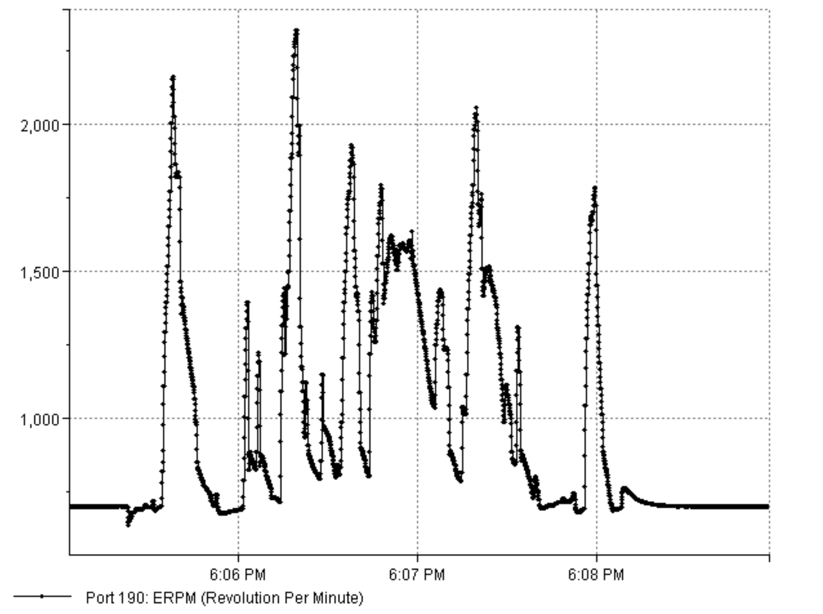
October 5, 2010

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

Coming Up

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

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
}
```

Coming Up

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

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

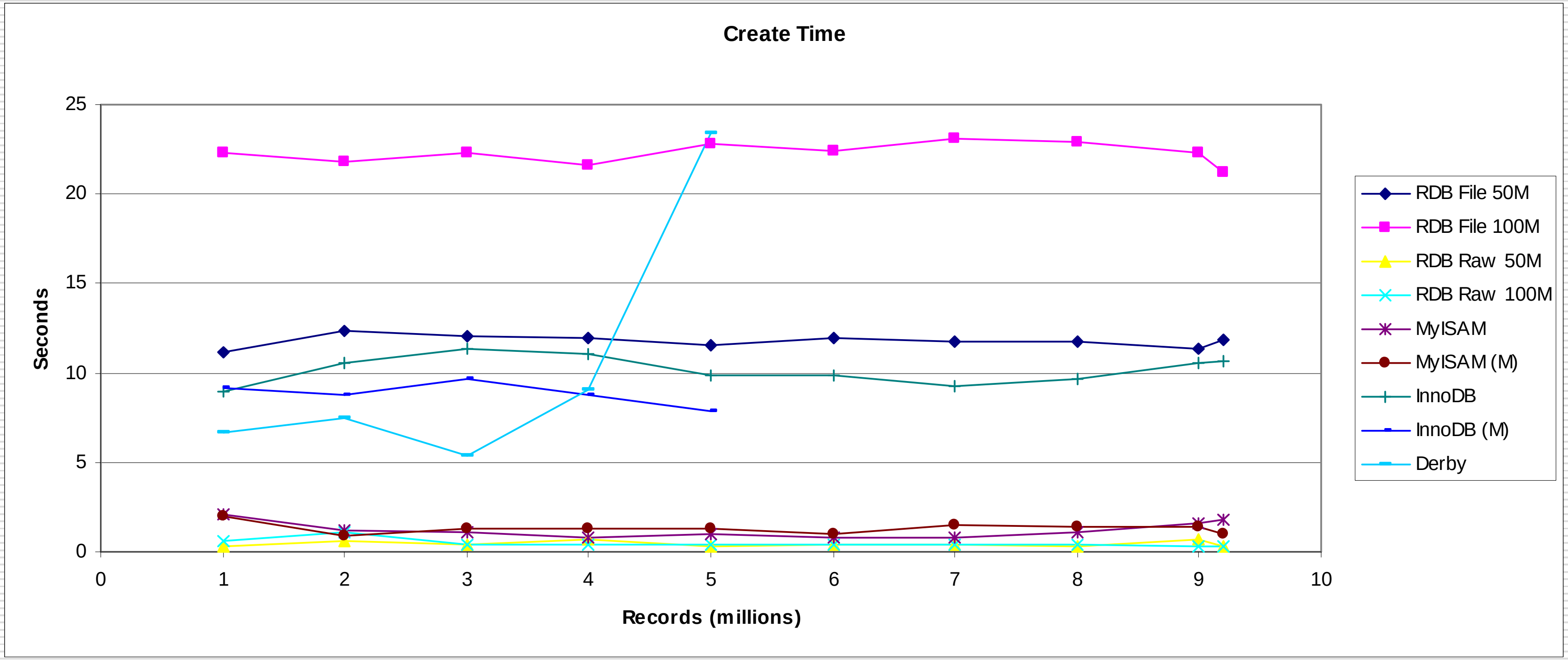
Benchmark Environment

- 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

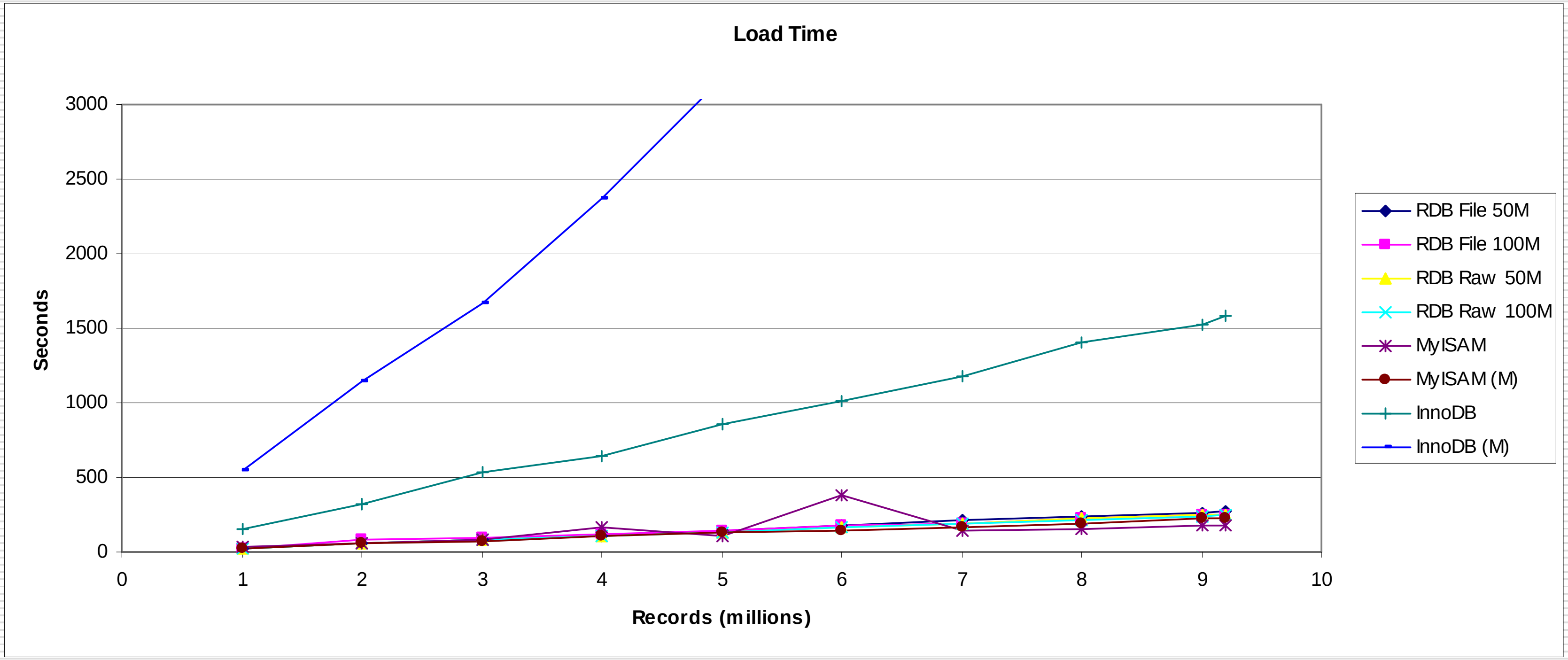
Benchmark Process

- 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

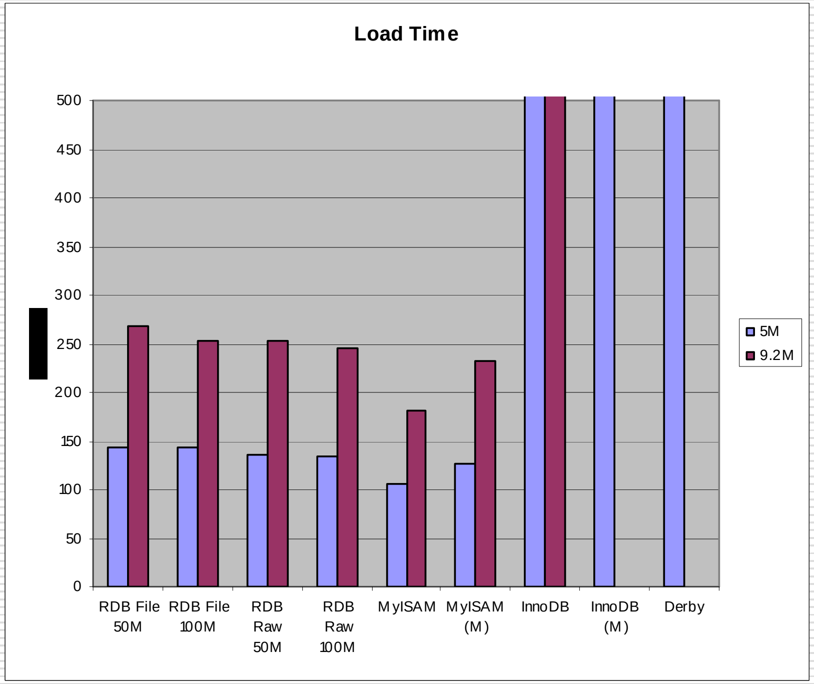
Results – Creation Time



Results – Load Time



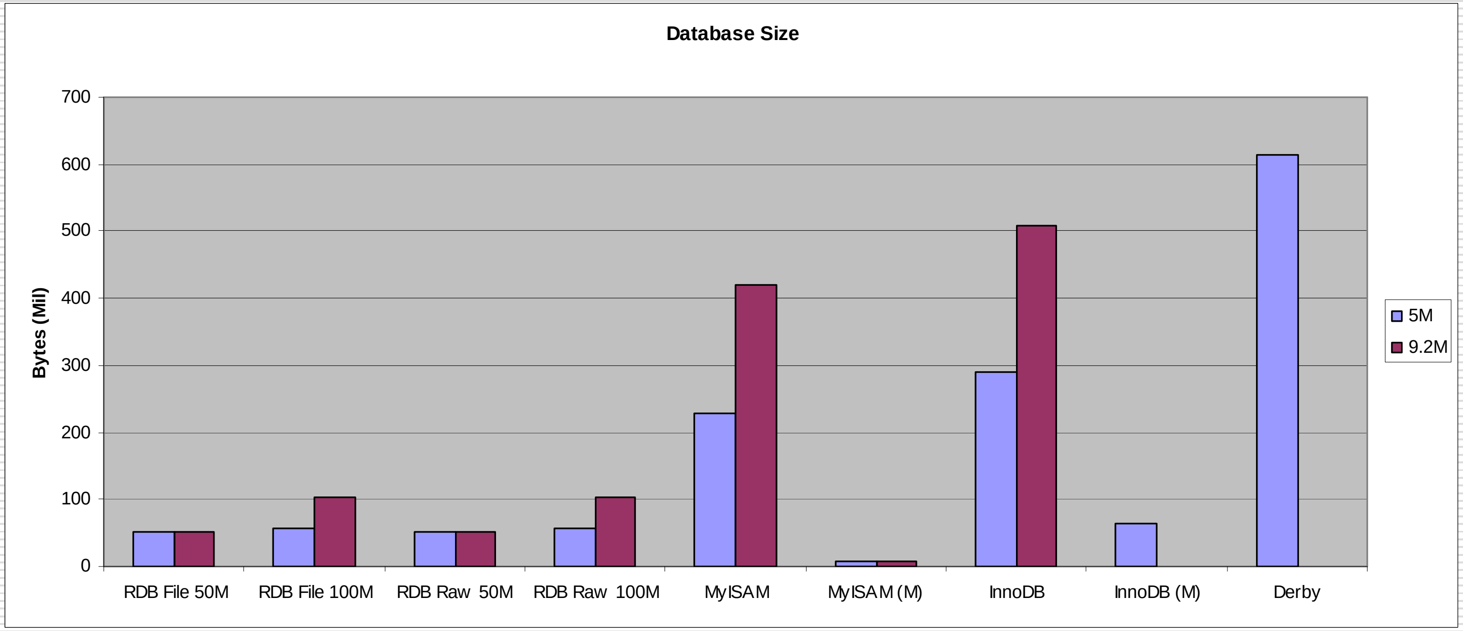
Results – Load Time



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%	

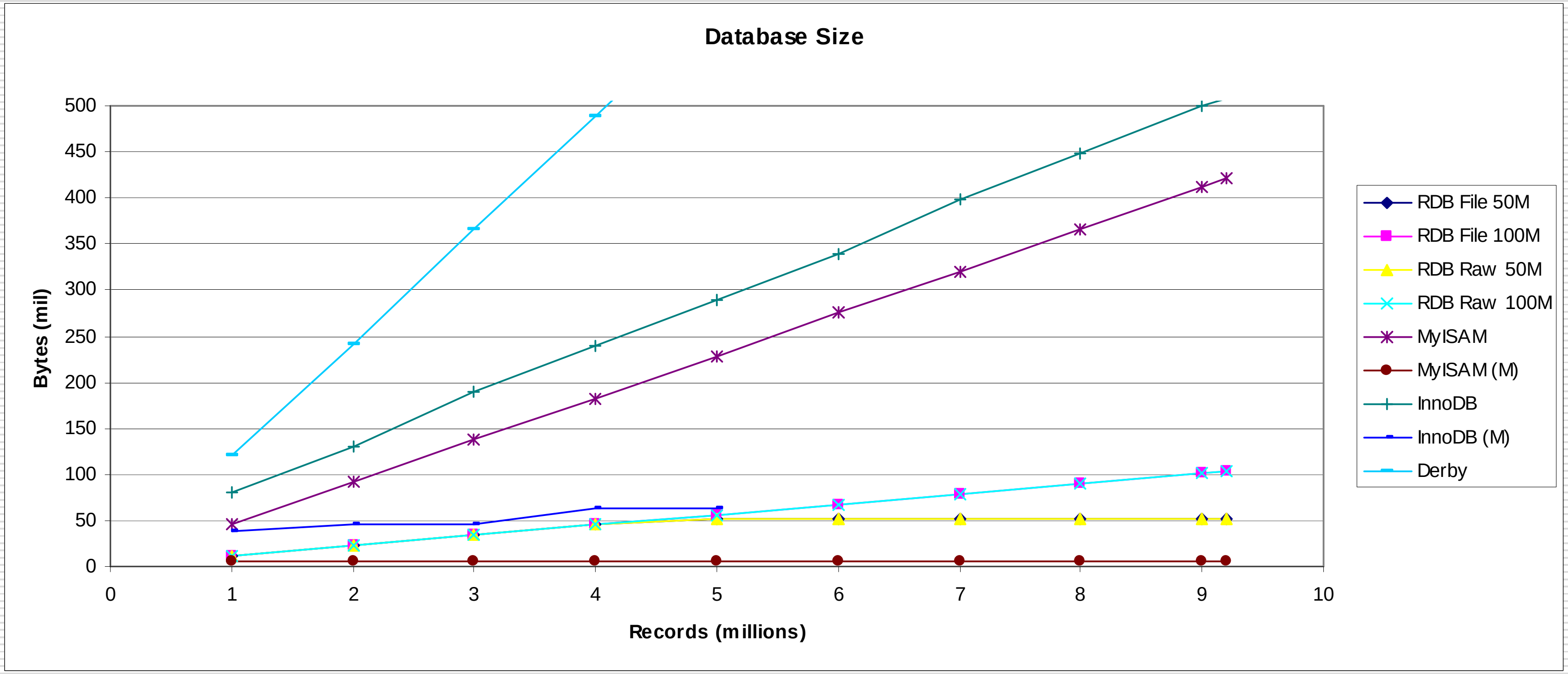
Implementation	5M Records	9.2M Records
InnoDB	859 s	1589 s
InnoDB (M)	3174 s	
Derby	21971 s	

Results – Database Size

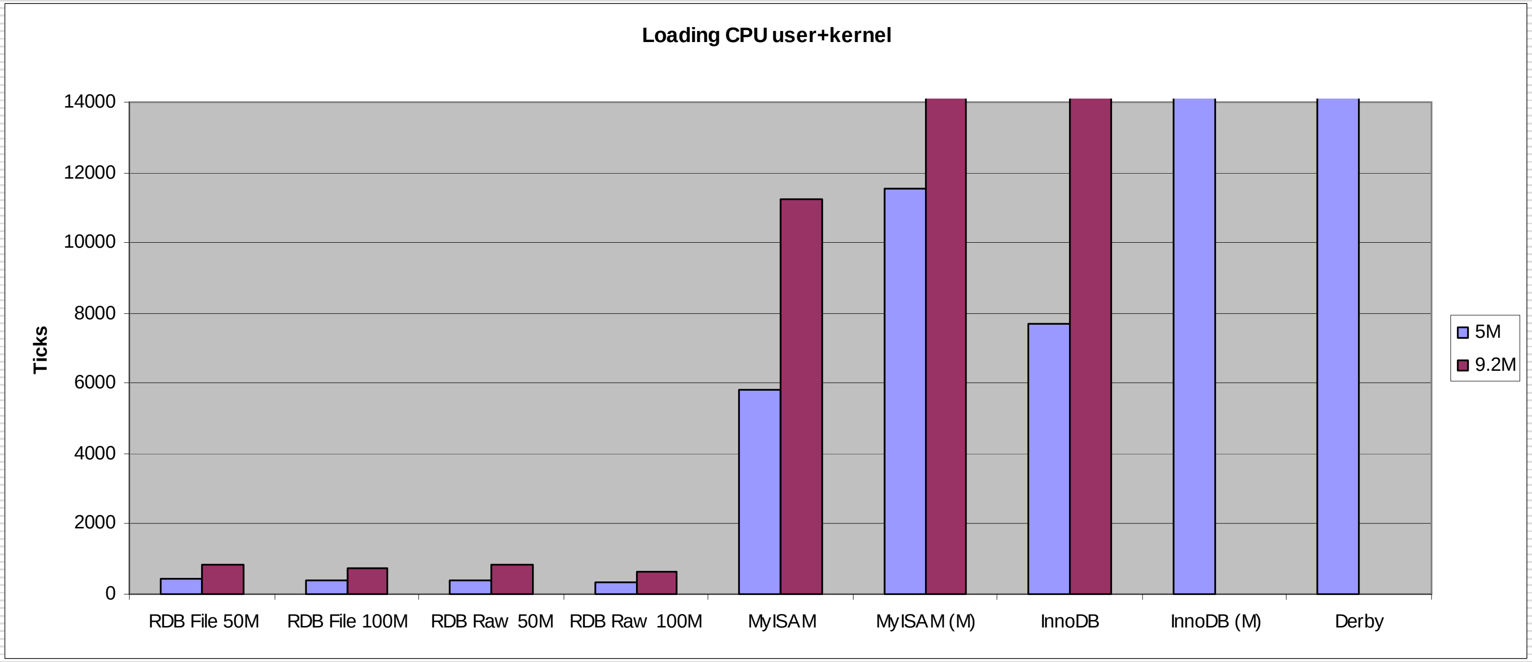


Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	75.3%	75.4%	75.3%	75.4%
MyISAM (M)	-727.6%	-721.3%	-727.6%	-721.3%
InnoDB	80.5%	79.6%	80.5%	79.6%
InnoDB (M)	16.7%		16.7%	
Derby	90.8%		90.8%	

Results – Database Size

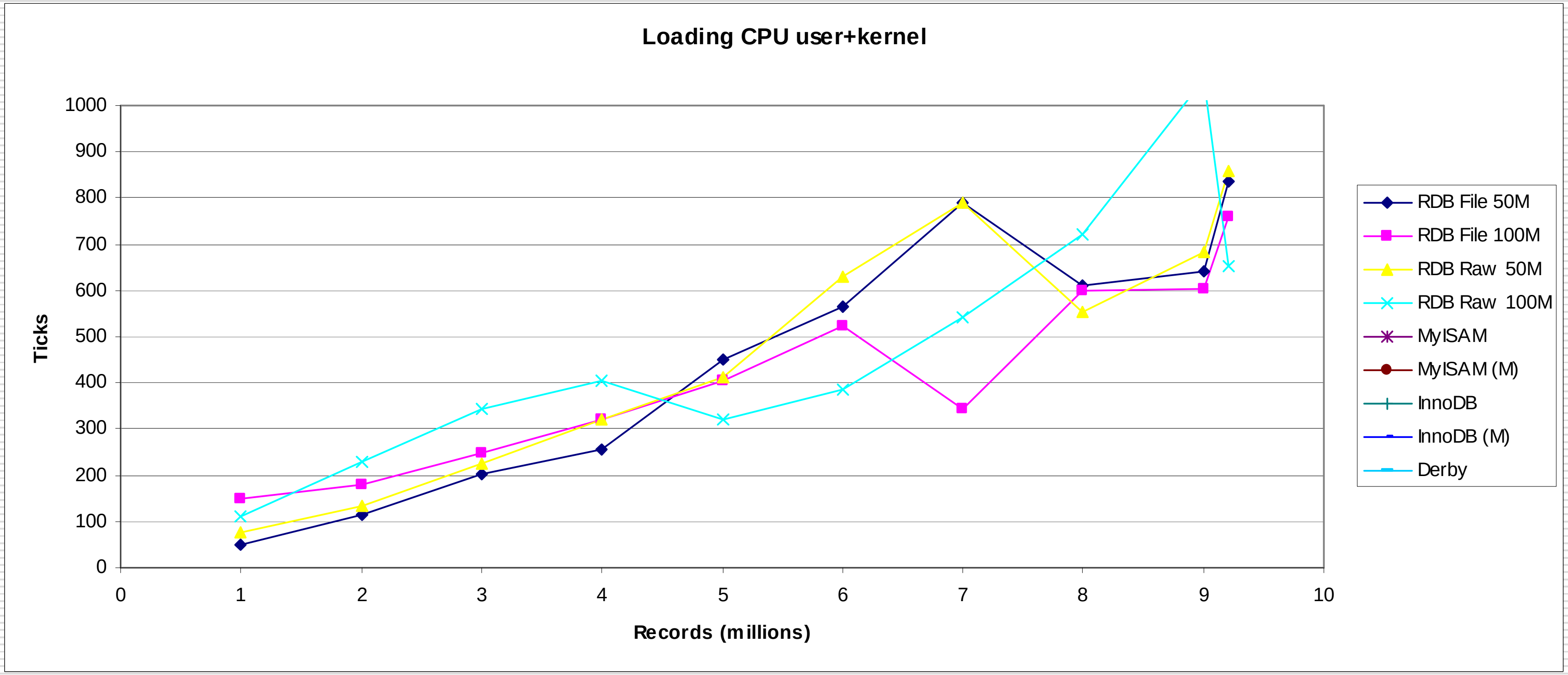


Results – CPU Utilization

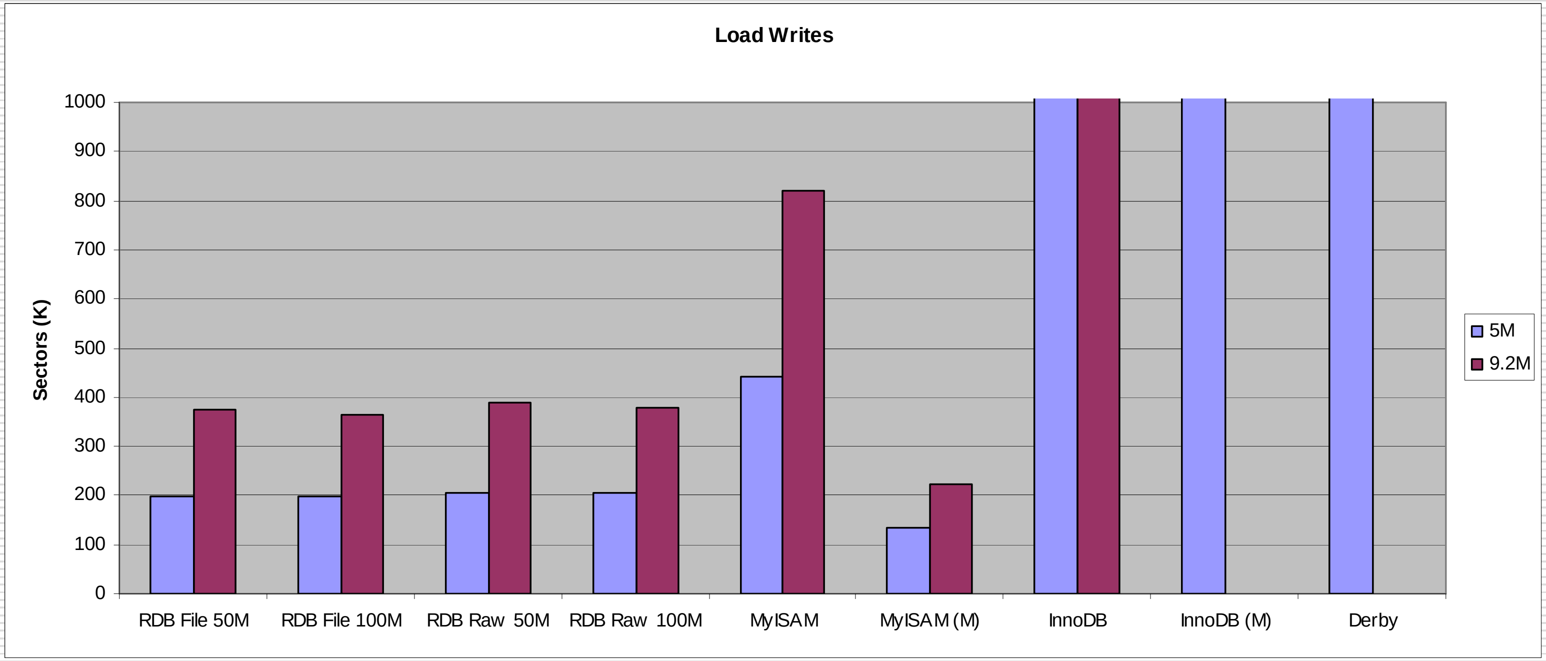


Implementation	5M File	9.2M File	5M Raw	9.2M Raw
MyISAM	93.0%	93.3%	94.5%	94.2%
MyISAM (M)	96.1%	96.1%	96.4%	96.0%
InnoDB	94.7%	95.6%	95.8%	96.2%
InnoDB (M)	98.9%		99.0%	
Derby	99.4%		99.5%	

Results – CPU Utilization

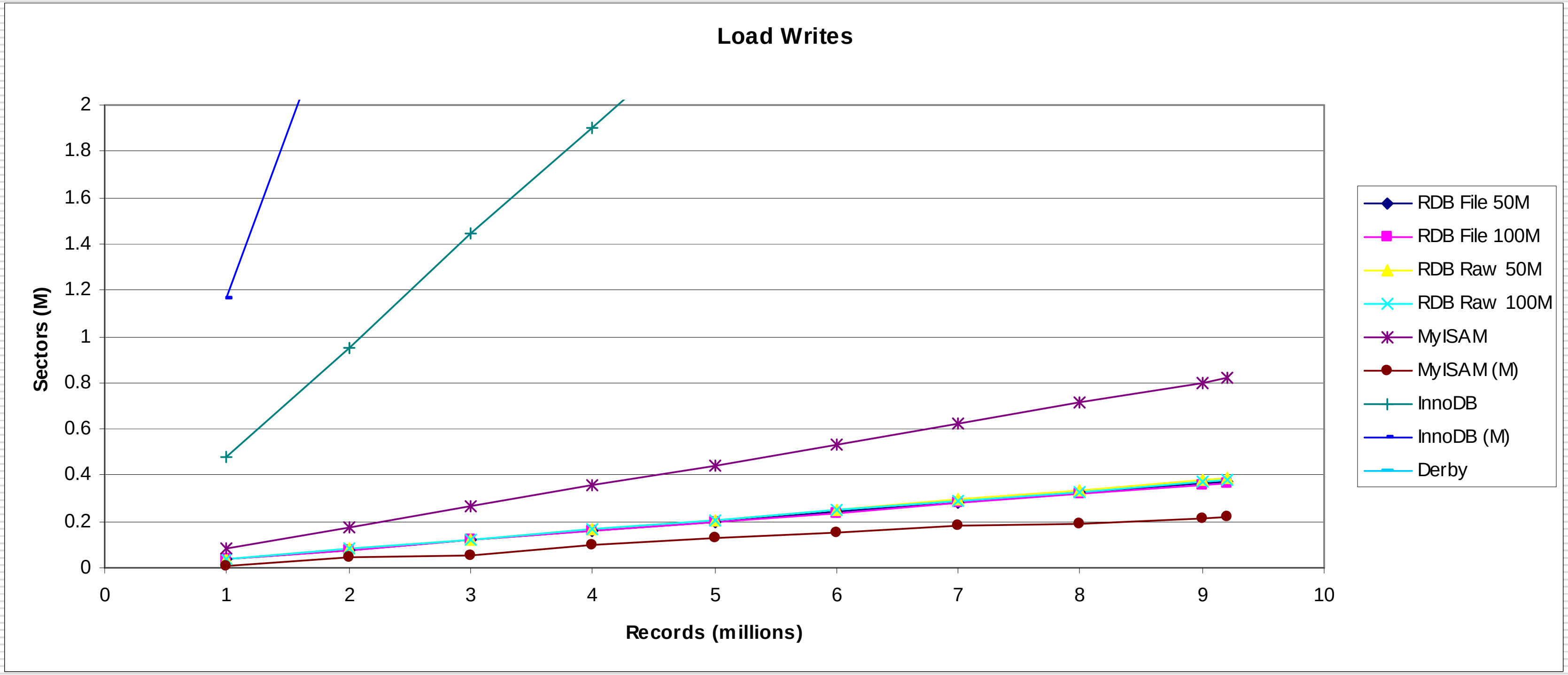


Results – Disk Writes



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

Further Details

- The slides following are further details on topics not covered entirely in the previous slides

Data Streams

- Sample is a fixed-size, user-specified structure with named fields
  - Integers (8-64 bit)
  - Real numbers (32-64 bit IEEE-754)
  - Booleans
- Samples have 64-bit timestamps in ascending order
- Can be in either a known or unknown (discontinuity) state

Stream Codecs

- Allows compression and reconstruction of stream data on a per-element basis. User-specified or built-in:
  - SampledAlgorithm: every point
  - StepAlgorithm: only on any change
  - DeadbandAlgorithm: only if it changes enough

Data Gathering

- Gathering of all items, or based on start and end times
- Iterate over the records (after codec)
- Iterate over stream time intervals (codec reconstruction), allowing:
  - Start/end time and timespan
  - Average
  - Minimum
  - Maximum
  - Resampling (value at time X)
  - Integral

Metadata Section

- Data block size, in file blocks
- Maximum streams (data index section size is calculated from this)
- Stream information:
  - User ID (integer)
  - Name
  - Ordered list of record elements
    - Name
    - Codec algorithm used (such as SampledAlgorithm)
    - Data type
    - Whether or not the element is required in the record (nullable/optional)

Index Section

- Data index for a stream is a fixed-size circular buffer of data block indices
- Index block fields (one file block):
  - Time of first record in all referenced data blocks
  - Time of last record
  - Array of data block indices
  - Checksum and modification sequence number

Transactions

- Only covers data block allocation and transfer from stream A to B (delete+add)
- Transaction entry types:
  - Change in unallocated block pointer
  - Allocation of a free block (leftover from recovered DB)
  - Remove a block from stream
  - Add block to stream
- Transaction log is fixed size circular buffer, preallocated

Transaction Recovery

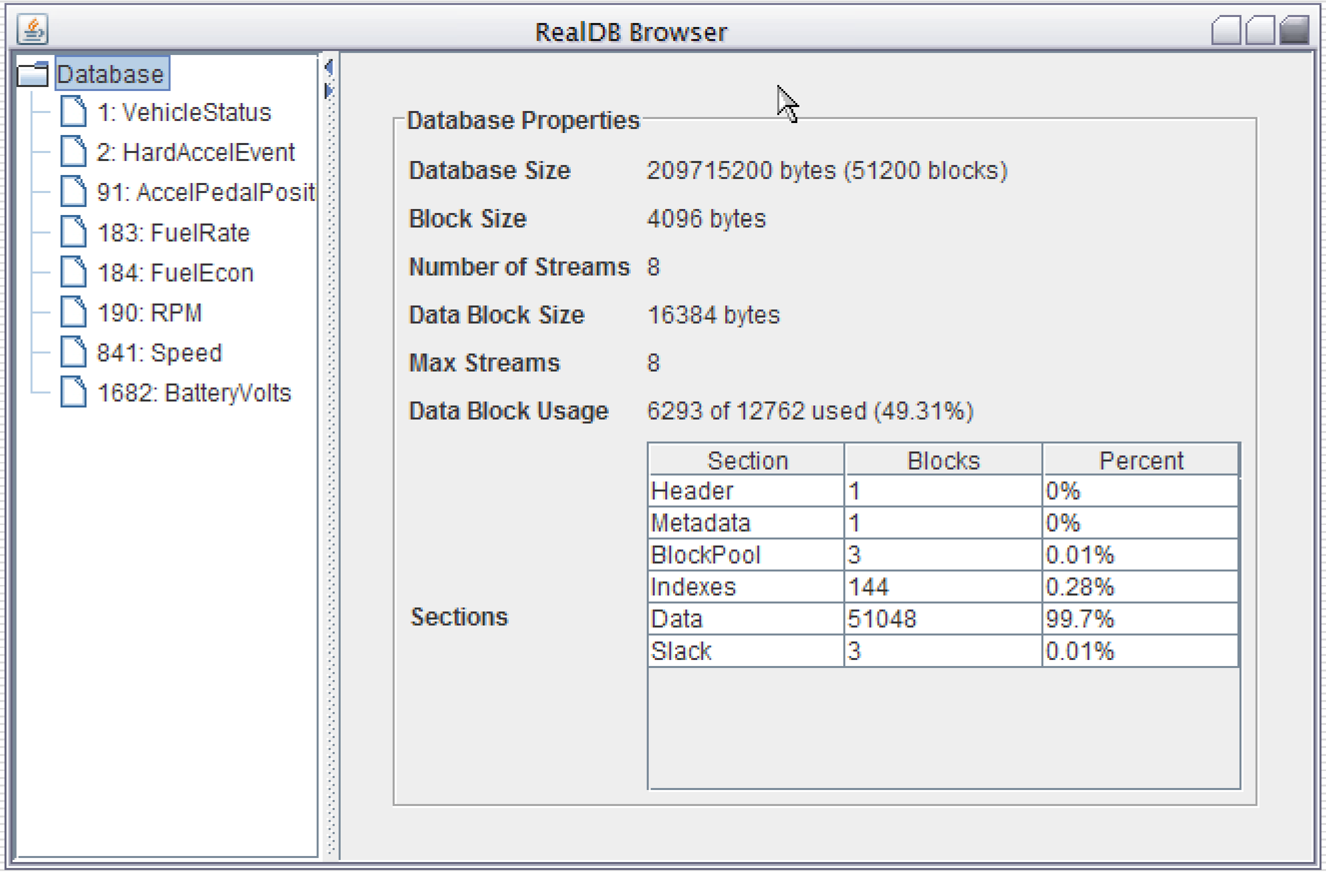
- Each transaction records the “current state” and the change to make
- Recovery process:
  - Iterate over each transaction
  - If current state equals that described, make the change
  - For removed blocks, add to an in-memory list of free blocks to reallocate

Transaction Example

- Steps for transferring a block when database is full:

Action	What happens if system crashes immediately after
Find the index whose first block is the oldest data block (source index)	Nothing; no disk modifications yet
Start a transaction with a Remove Block entry	Remove Block is replayed, block placed back on free list
Remove the first block from the source stream's index	Block added to (memory) free list
Write the data block	Block added to (memory) free list
End the transaction with a Add Block entry	Add block is replayed
Add the block to the destination stream's index	Nothing; transaction completed

RealDB Browser



- Database
  - 1: VehicleStatus
  - 2: HardAccelEvent
  - 91: AccelPedalPosit
  - 183: FuelRate
  - 184: FuelEcon
  - 190: RPM
  - 841: Speed
  - 1682: BatteryVolts

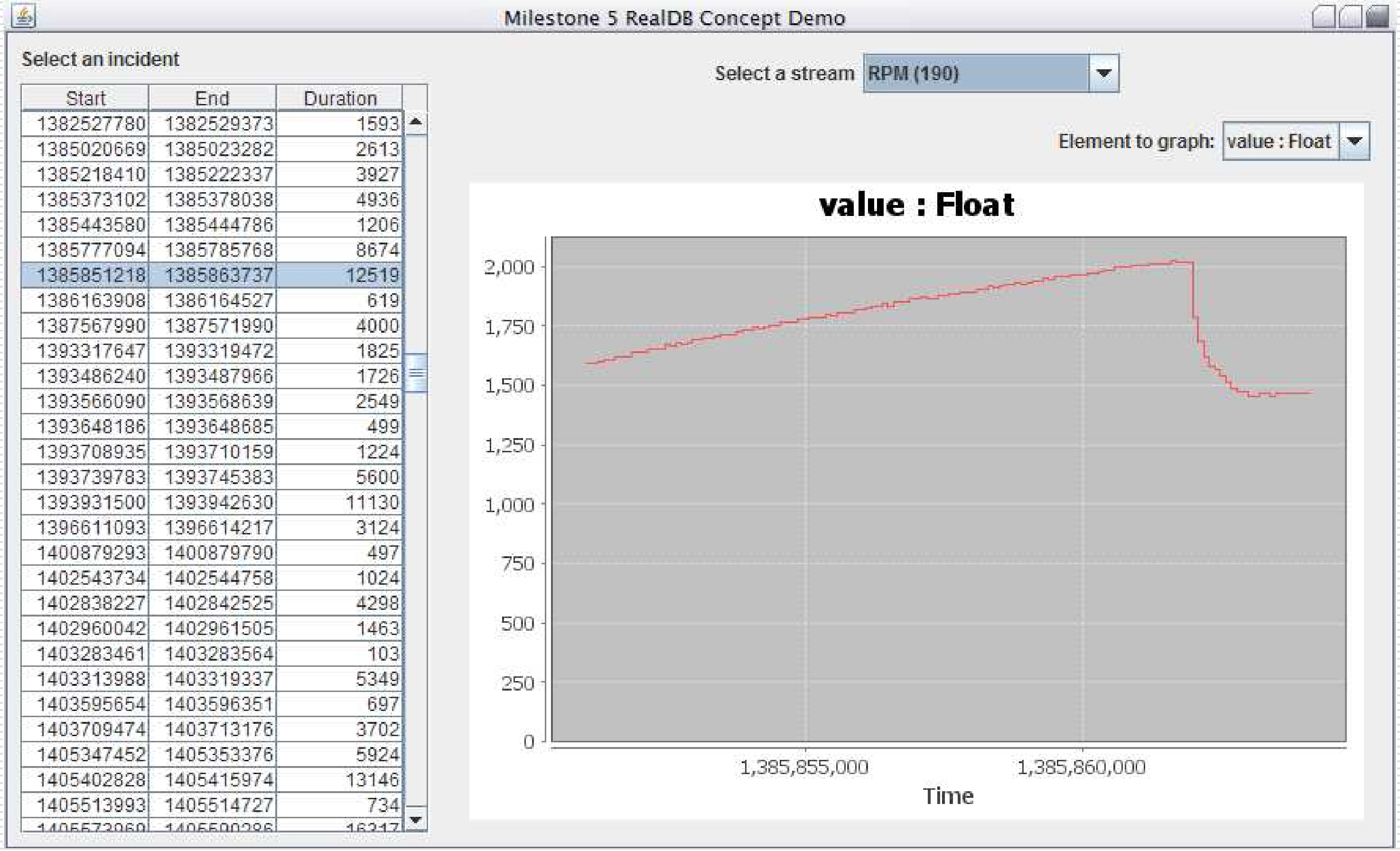
Database Properties

Database Size	209715200 bytes (51200 blocks)
Block Size	4096 bytes
Number of Streams	8
Data Block Size	16384 bytes
Max Streams	8
Data Block Usage	6293 of 12762 used (49.31%)

Sections

Section	Blocks	Percent
Header	1	0%
Metadata	1	0%
BlockPool	3	0.01%
Indexes	144	0.28%
Data	51048	99.7%
Slack	3	0.01%

RealDB Proof-of-Concept



Select an incident

Start	End	Duration
1382527780	1382529373	1593
1385020669	1385023282	2613
1385218410	1385222337	3927
1385373102	1385378038	4936
1385443580	1385444786	1206
1385777094	1385785768	8674
1385851218	1385863737	12519
1386163908	1386164527	619
1387567990	1387571990	4000
1393317647	1393319472	1825
1393486240	1393487966	1726
1393566090	1393568639	2549
1393648186	1393648685	499
1393708935	1393710159	1224
1393739783	1393745383	5600
1393931500	1393942630	11130
1396611093	1396614217	3124
1400879293	1400879790	497
1402543734	1402544758	1024
1402838227	1402842525	4298
1402960042	1402961505	1463
1403283461	1403283564	103
1403313988	1403319337	5349
1403595654	1403596351	697
1403709474	1403713176	3702
1405347452	1405353376	5924
1405402828	1405415974	13146
1405513993	1405514727	734
1405573969	1405590286	16317

Select a stream: RPM (190)

Element to graph: value : Float

Benchmark RDL

```
SET blockSize     = 4096
SET fileSize      = 52428800 #or 104857600
SET maxStreams    = 8
SET dataBlockSize = 4

CREATE STREAM RPM WITH ID 190 {
  value float
}
CREATE STREAM FuelRate WITH ID 183 {
  value float
}
CREATE STREAM AccelPedalPosition WITH ID 91 {
  value float
}
CREATE STREAM BatteryVolts WITH ID 1682 {
  value float
}
CREATE STREAM FuelEcon WITH ID 184 {
  value float
}
CREATE STREAM Speed WITH ID 841 {
  value float
}
CREATE STREAM VehicleStatus WITH ID 1 {
  collecting boolean WITH CODEC StepAlgorithm
}
CREATE STREAM HardAccelEvent WITH ID 2 {
  active boolean WITH CODEC StepAlgorithm
}
```

Benchmark SQL – MySQL MyISAM

```
DROP DATABASE IF EXISTS `realdb_benchmark`;

CREATE DATABASE `realdb_benchmark`;

CREATE TABLE `realdb_benchmark`.`FloatData` (
  `streamId` INTEGER UNSIGNED NOT NULL,
  `time` BIGINT UNSIGNED NOT NULL,
  `value` FLOAT NOT NULL,
  `discontinuity` BIT NOT NULL,
  PRIMARY KEY (`streamId`, `time`),
  INDEX `Index_Time`(`time`)
)
ENGINE = MyISAM;

CREATE TABLE `realdb_benchmark`.`BooleanData` (
  `streamId` INTEGER UNSIGNED NOT NULL,
  `time` BIGINT UNSIGNED NOT NULL,
  `value` BIT NOT NULL,
  `discontinuity` BIT NOT NULL,
  PRIMARY KEY (`streamId`, `time`),
  INDEX `Index_Time`(`time`)
)
ENGINE = MyISAM;
```