What's New in OpenLDAP

Howard Chu
CTO, Symas Corp / Chief Architect OpenLDAP

FOSDEM'14

OpenLDAP Project

- Open source code project
- Founded 1998
- Three core team members
- A dozen or so contributors
- Feature releases every 12-18 months
- Maintenance releases roughly monthly

A Word About Symas

- Founded 1999
- Founders from Enterprise Software world
  - *platinum* Technology (Locus Computing)
  - IBM
- Howard joined OpenLDAP in 1999
  - One of the Core Team members
  - Appointed Chief Architect January 2007
- No debt, no VC investments

Intro

Howard Chu

- Founder and CTO Symas Corp.
- Developing Free/Open Source software since 1980s
  - GNU compiler toolchain, e.g. "gmake -j", etc.
  - Many other projects, check ohloh.net...
- Worked for NASA/JPL, wrote software for Space Shuttle, etc.

What's New

- Lightning Memory-Mapped Database (LMDB) and its knock-on effects
  - Within OpenLDAP code
  - Other projects
- New HyperDex clustered backend
- New Samba4/AD integration work
- Other features
- What's missing

LMDB

- Introduced at LDAPCon 2011
  - Full ACID transactions
  - MVCC, readers and writers don't block each other
  - Ultra-compact, compiles to under 32KB
  - Memory-mapped, lightning fast zero-copy reads
  - Much greater CPU and memory efficiency
  - Much simpler configuration

LMDB Impact

- Within OpenLDAP
  - Revealed other frontend bottlenecks that were hidden by BerkeleyDB-based backends
  - Addressed in OpenLDAP 2.5
    - Thread pool enhanced, support multiple work queues to reduce mutex contention
    - Connection manager enhanced, simplify write synchronization

OpenLDAP Frontend

- Testing in 2011 (16 core server):
  - back-hdb, 62000 searches/sec, 1485 % CPU
  - back-mdb, 75000 searches/sec, 1000 % CPU
  - back-mdb, 2 slapds, 127000 searches/sec, 1250 % CPU - network limited
- We should not have needed two processes to hit this rate

Efficiency Note

- back-hdb 62000 searches/sec @ 1485 %
  - 41.75 searches per CPU %
- back-mdb 127000 searches/sec @1250 %
  - 101.60 searches per CPU %
- 2.433x as many searches per unit of CPU
- "Performance" isn't the point, *Efficiency* is what matters

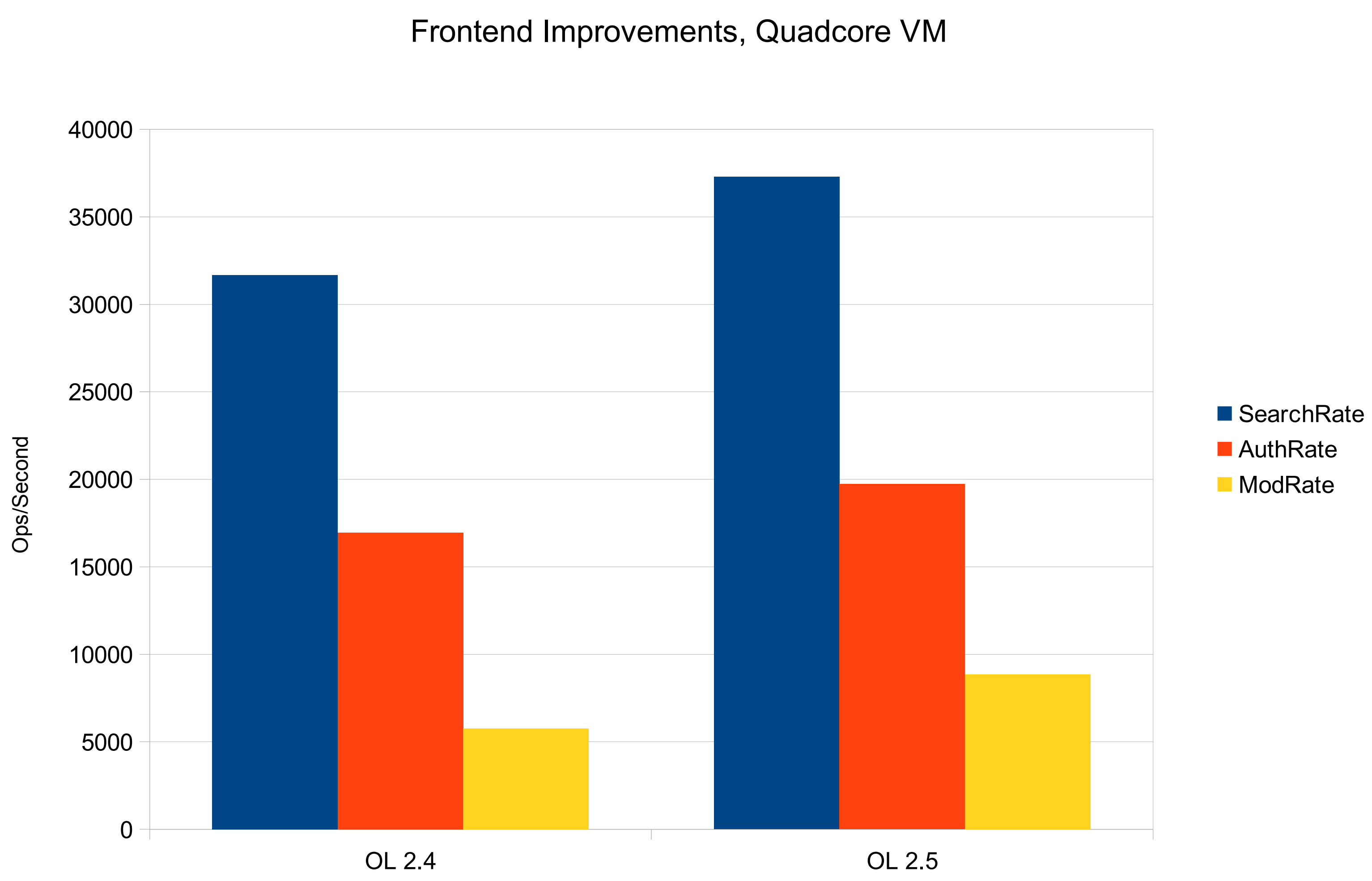
OpenLDAP Frontend

- Threadpool contention
  - Analyzed using mutrace
  - Found #1 bottleneck in threadpool mutex
  - Modified threadpool to support multiple queues
  - On quad-core laptop, using 4 queues reduced mutex contended time by factor of 6.
  - Reduced condition variable contention by factor of 3.
  - Overall 20 % improvement in throughput on quad-core VM

OpenLDAP Frontend

- Connection Manager
  - Also a single thread, accepting new connections and polling for read/write ready on existing
  - Now can be split to multiple threads
    - Impact depends on number of connections
  - Polling for write is no longer handled by the listener thread
    - Removes one level of locks and indirection
    - Simplifies WriteTimeout implementation
    - Typically no benchmark impact, only significant when blocking on writes due to slow clients

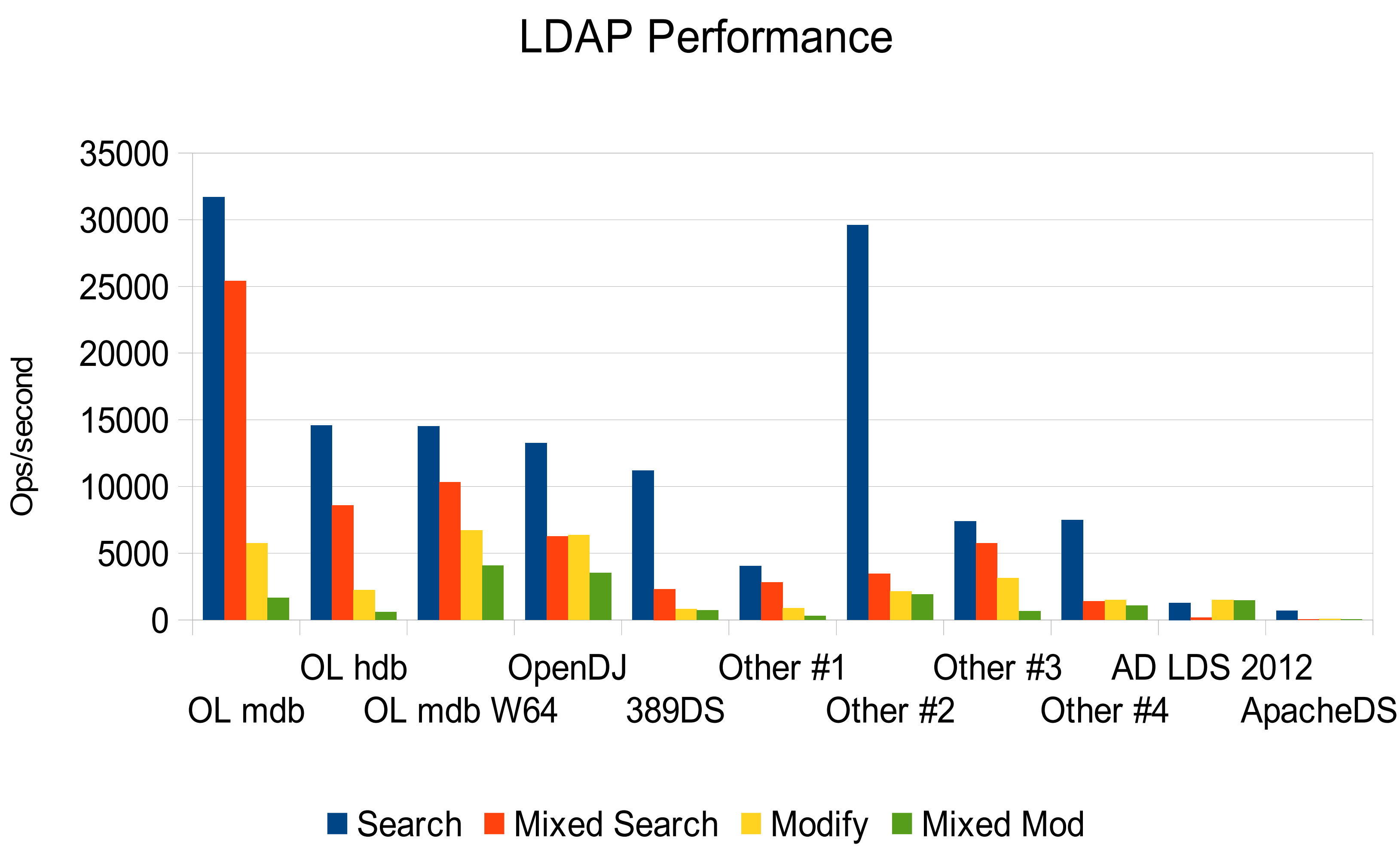
OpenLDAP Frontend



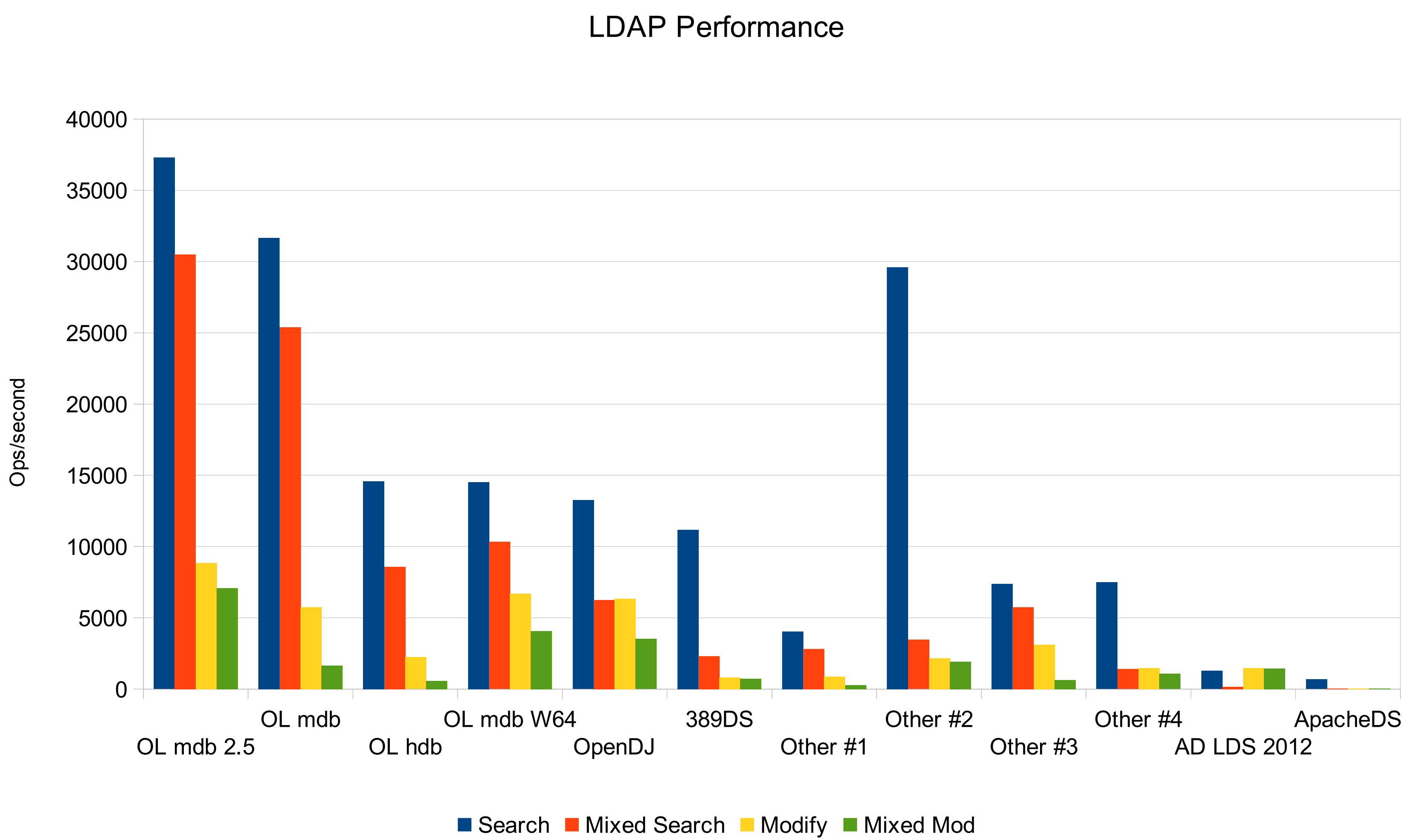
OpenLDAP Frontend

- Putting it into context, compared to :
  - OpenLDAP 2.4 back-mdb and hdb
  - OpenLDAP 2.4 back-mdb on Windows 2012 x64
  - OpenDJ 2.4.6, 389DS, ApacheDS 2.0.0-M13
  - Latest proprietary servers from CA, Microsoft, Novell, and Oracle

OpenLDAP Frontend



OpenLDAP Frontend



LMDB Impact

- Adoption by many other projects
  - Outperforms all other embedded databases in common applications
    - CFengine, Postfix, PowerDNS, etc.
  - Has none of the reliability/integrity weaknesses of other databases
  - Has none of the licensing issues...
  - Integrated into multiple NoSQL projects
    - Redis, SkyDB, Memcached, HyperDex, etc.

LMDB Microbenchmark

- Comparisons based on Google's LevelDB
- Also tested against Kyoto Cabinet's TreeDB, SQLite3, and BerkeleyDB
- Tested using RAM filesystem (tmpfs), reiserfs on SSD, and multiple filesystems on HDD
  - btrfs, ext2, ext3, ext4, jfs, ntfs, reiserfs, xfs, zfs
  - ext3, ext4, jfs, reiserfs, xfs also tested with external journals

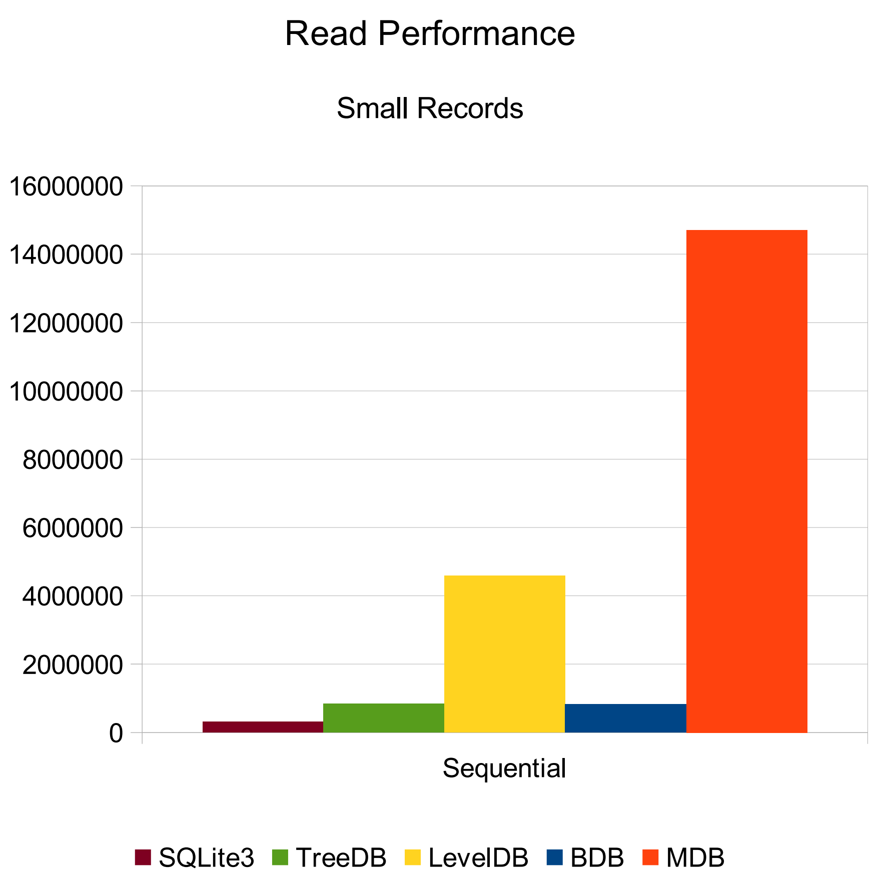
LMDB Microbenchmark

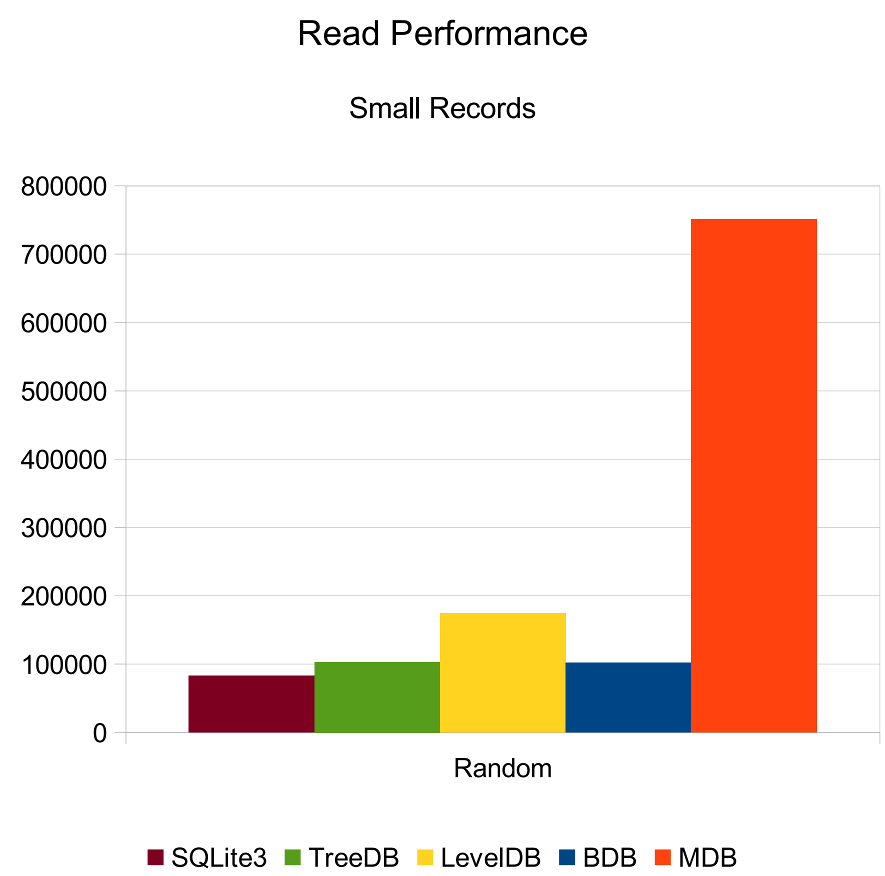
- Relative Footprint

text	data	bss	dec	hex	filename
272247	1456	328	274031	42e6f	db_bench
1675911	2288	304	1678503	199ca7	db_bench_bdb
90423	1508	304	92235	1684b	db_bench_mdb
653480	7768	1688	662936	a2764	db_bench_sqlite3
296572	4808	1096	302476	49d8c	db_bench_tree_db

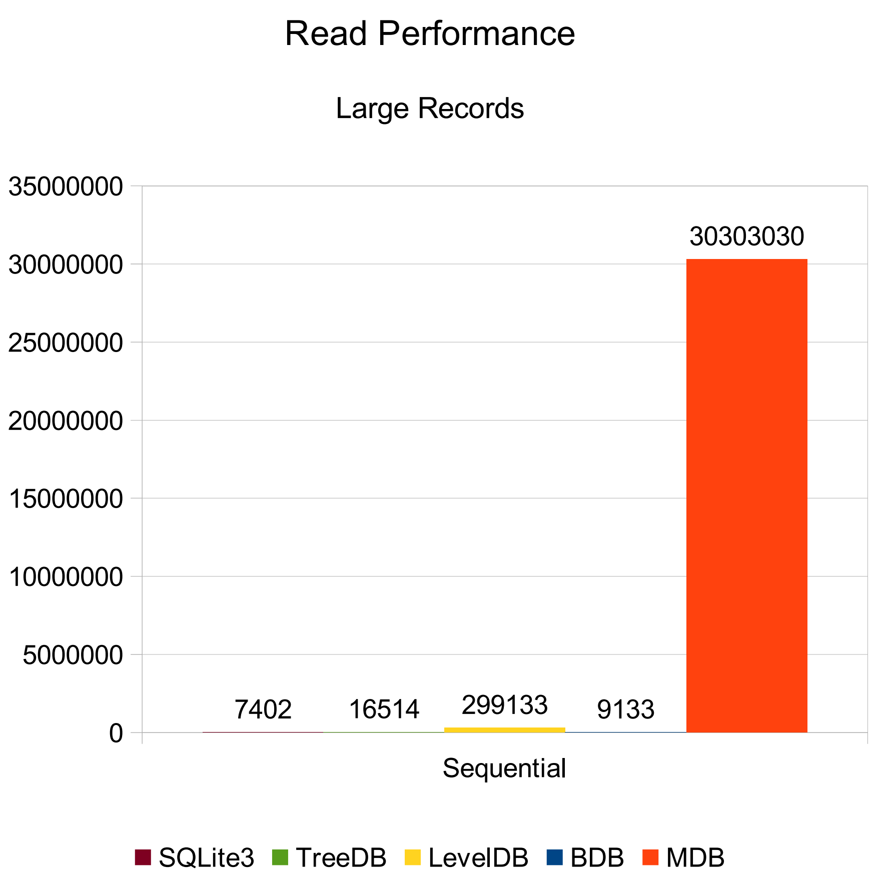
- Clearly LMDB has the smallest footprint
  - Carefully written C code beats C++ every time

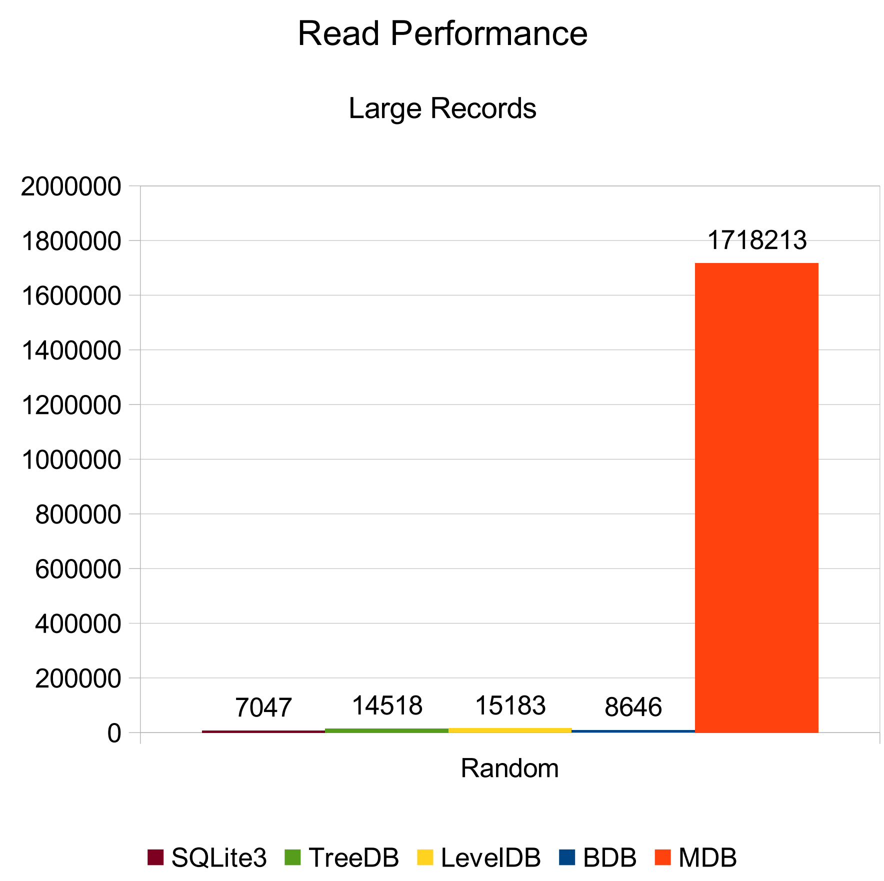
LMDB Microbenchmark



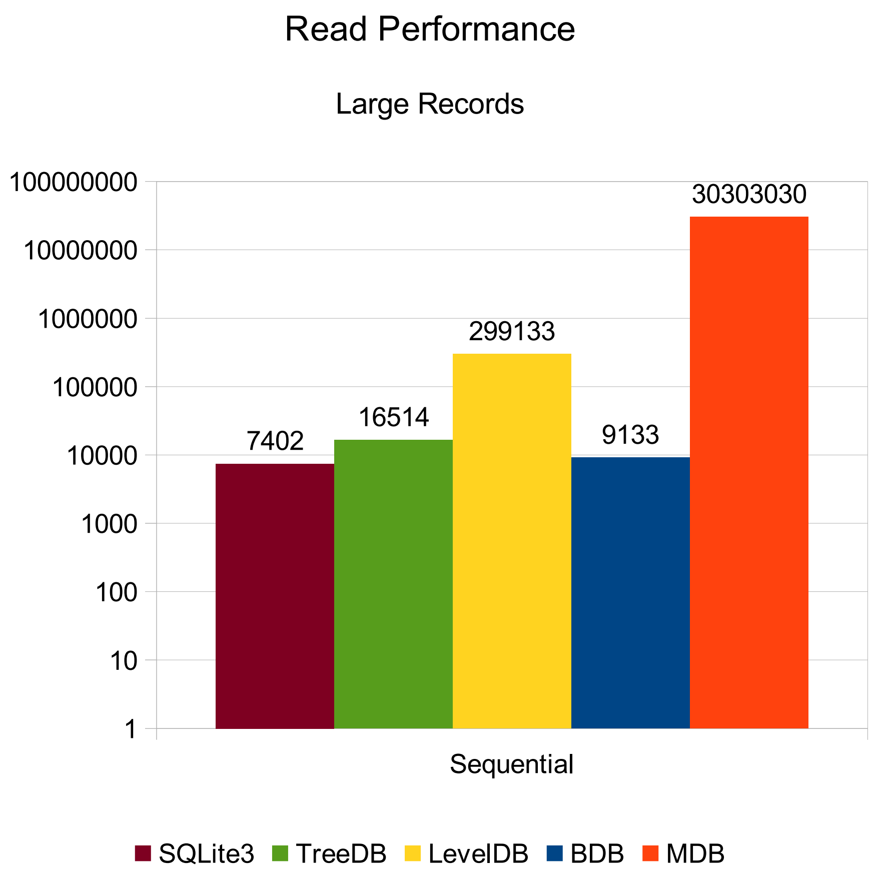


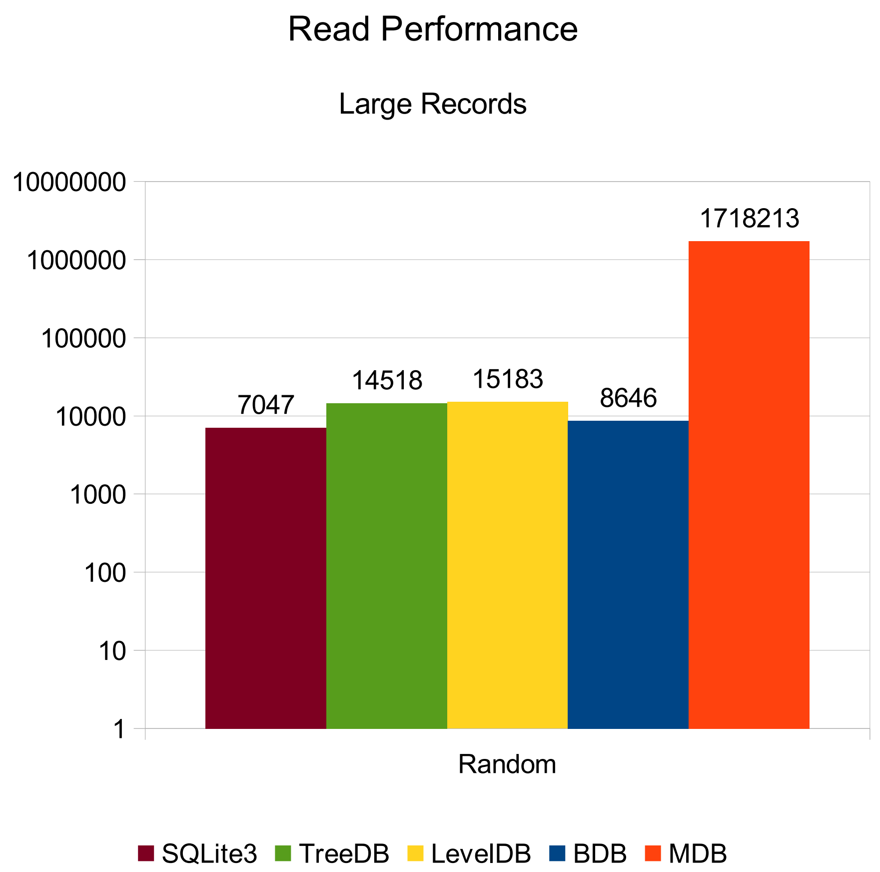
LMDB Microbenchmark



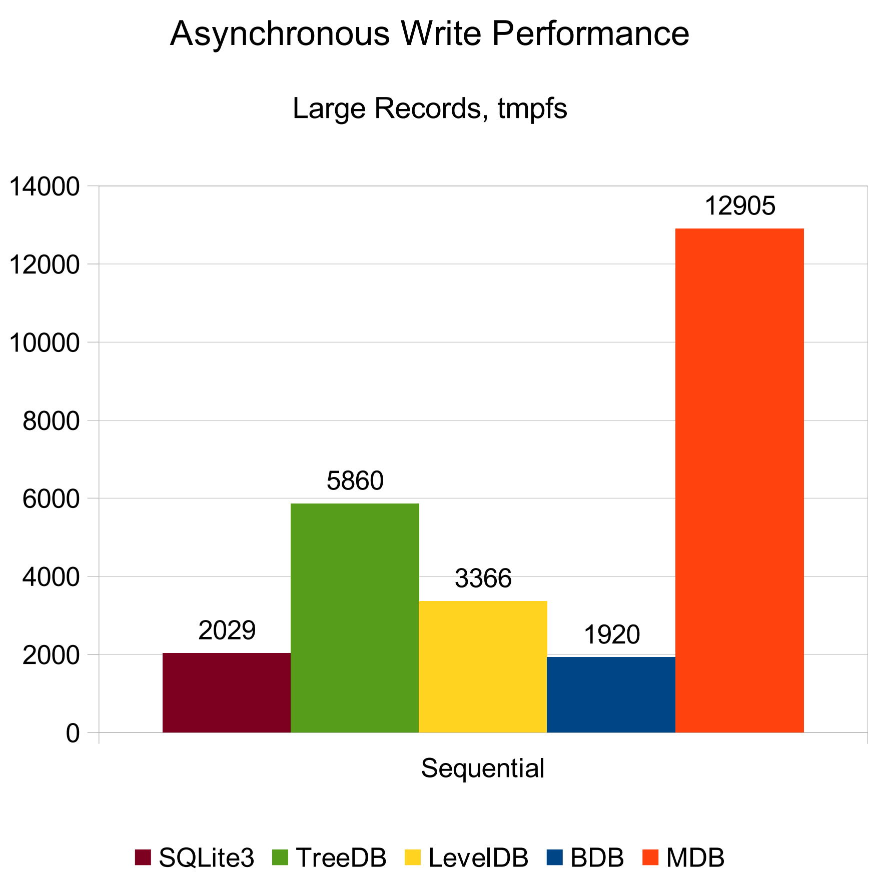


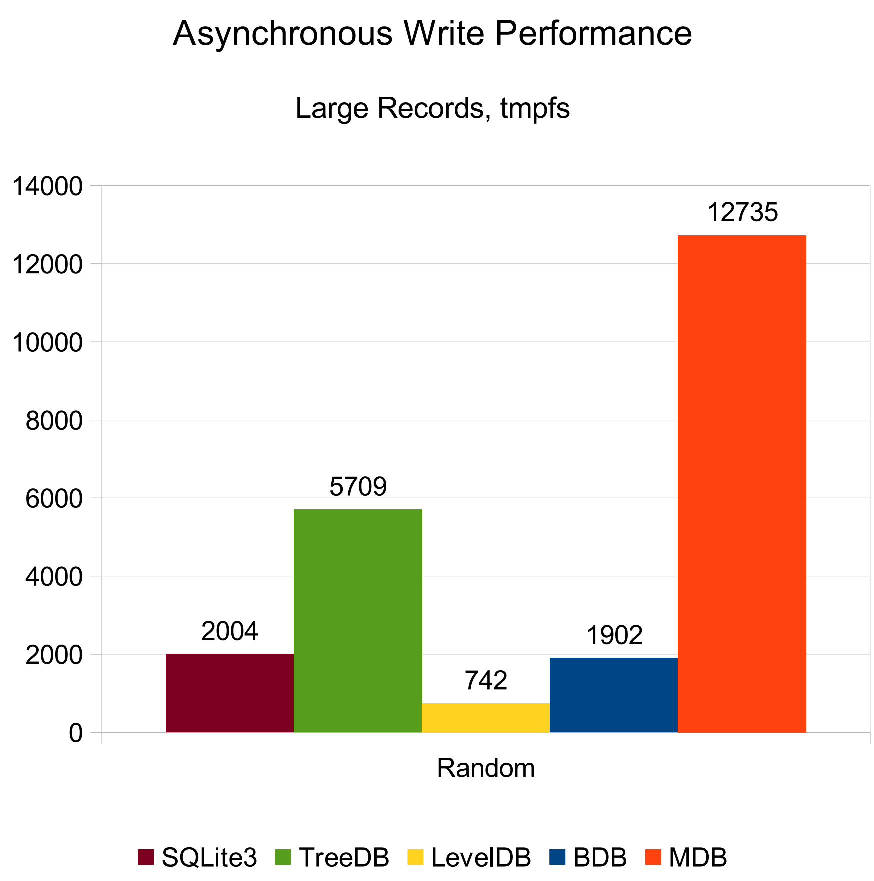
LMDB Microbenchmark



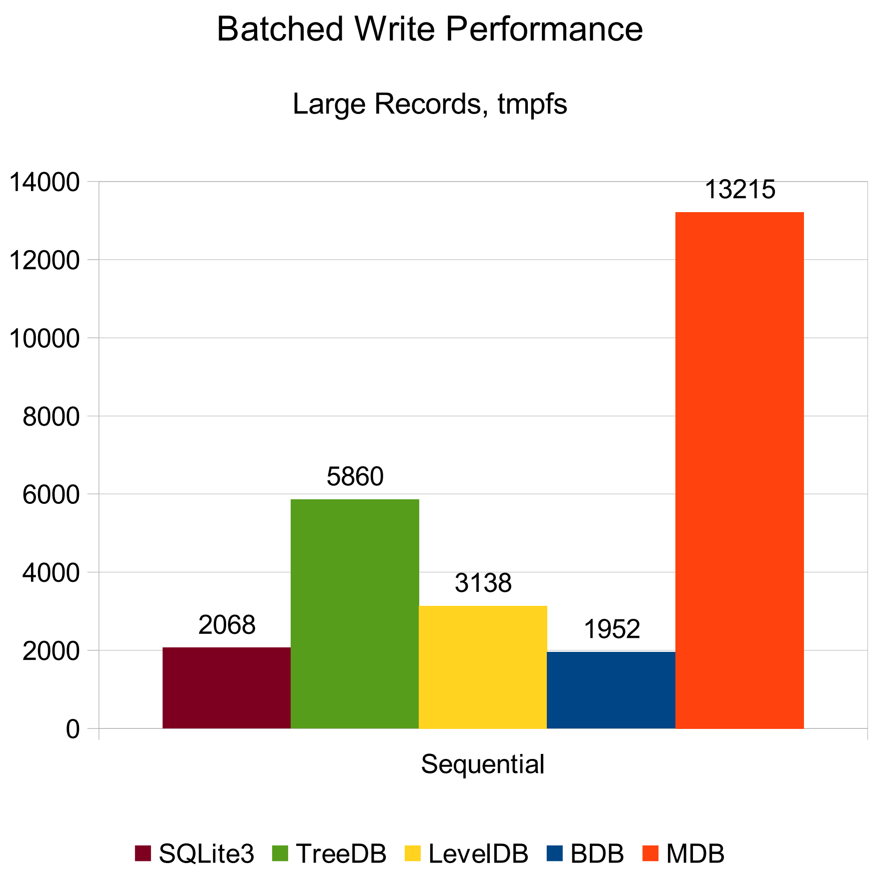


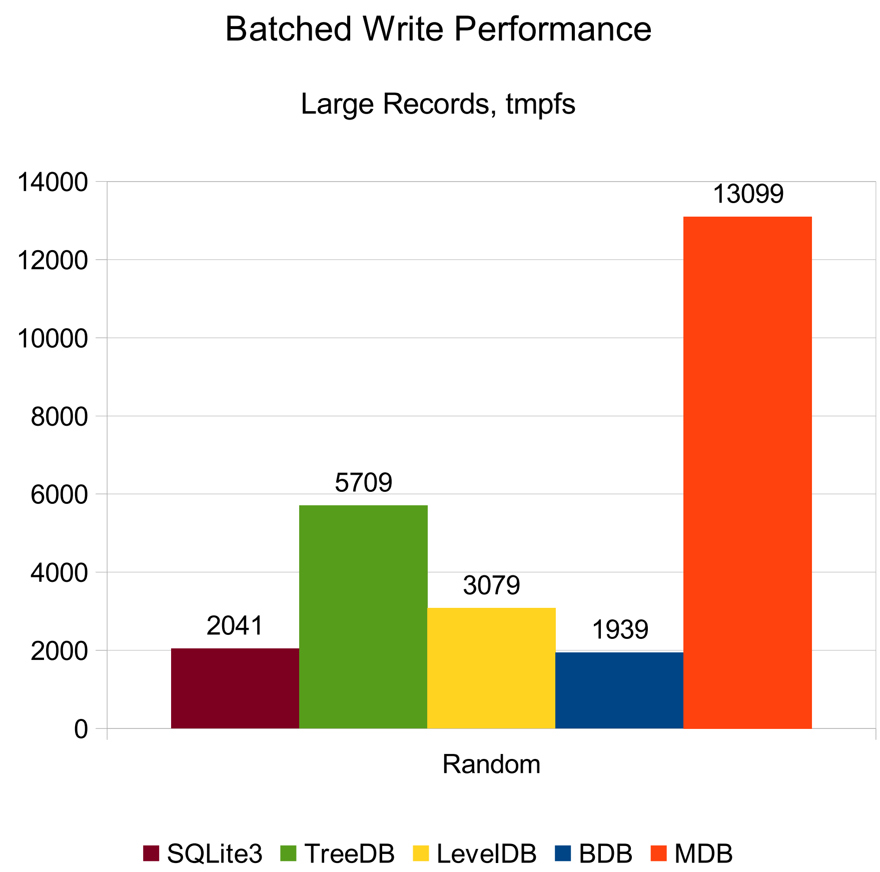
LMDB Microbenchmark



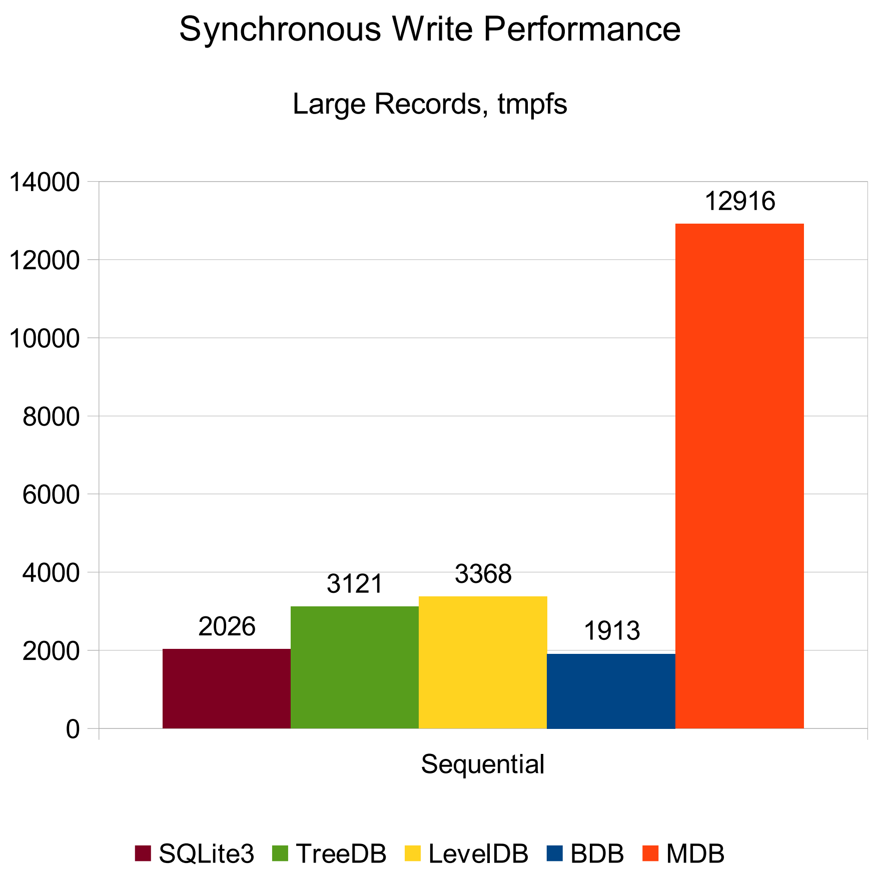


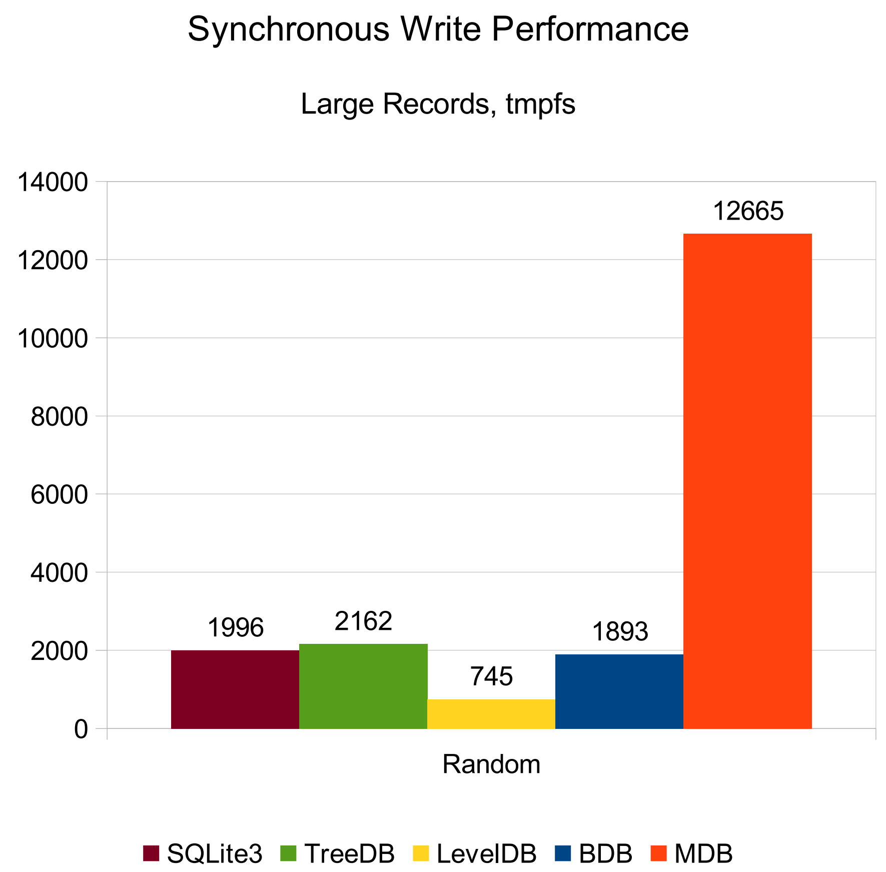
LMDB Microbenchmark



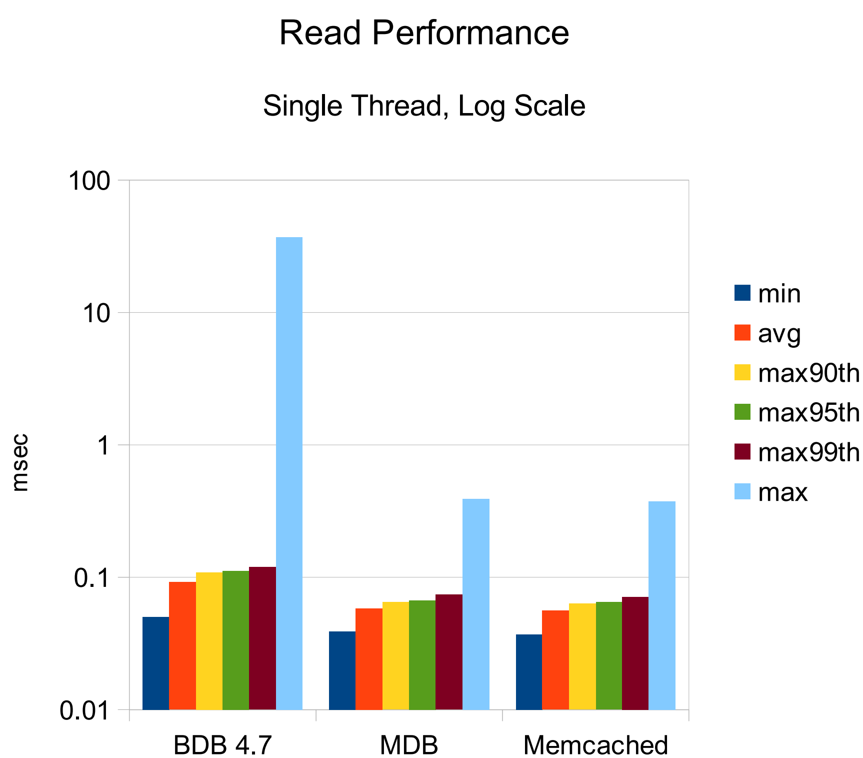


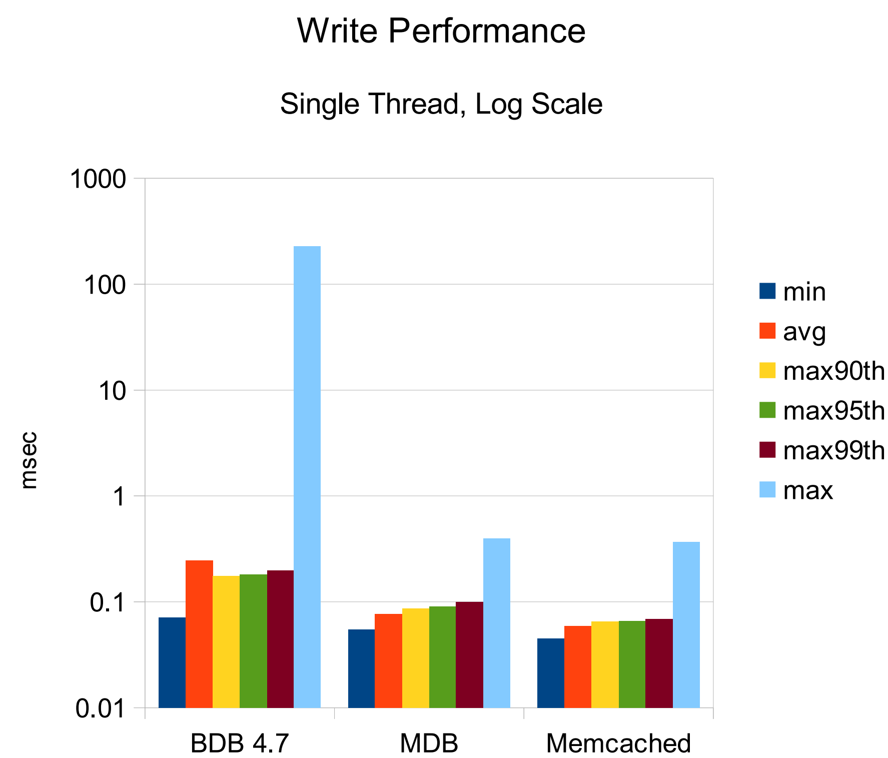
LMDB Microbenchmark



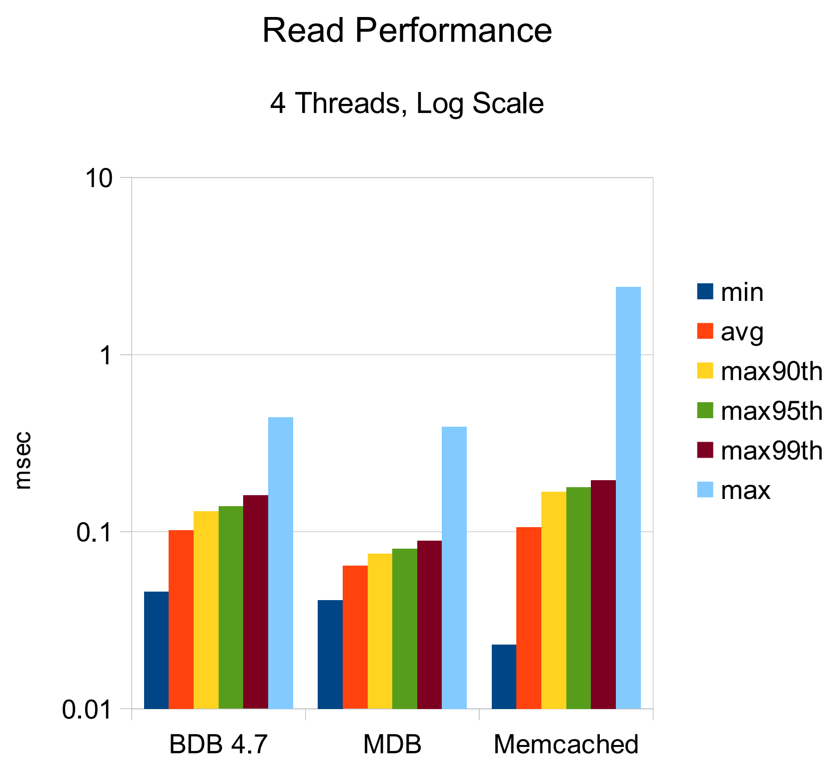


MemcacheDB

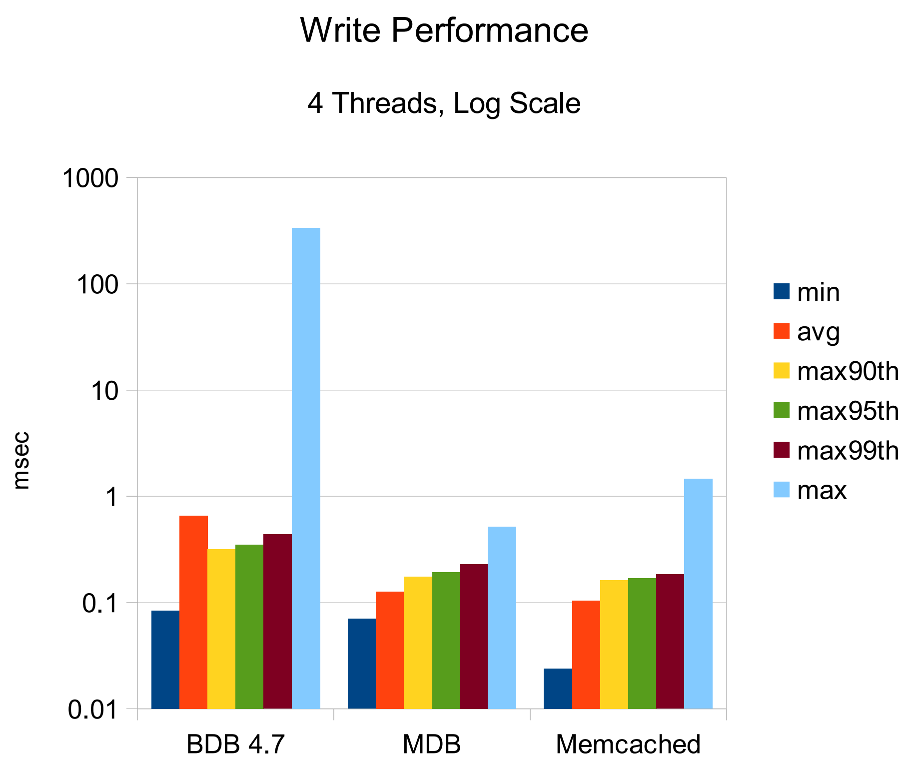

Read Performance
Single Thread, Log Scale
msec
100
10
1
0.1
0.01
BDB 4.7
MDB
Memcached
min
avg
max90th
max95th
max99th
max


Write Performance
Single Thread, Log Scale
msec
1000
100
10
1
0.1
0.01
BDB 4.7
MDB
Memcached
min
avg
max90th
max95th
max99th
max

MemcacheDB



Read Performance
4 Threads, Log Scale
msec
10
1
0.1
0.01
BDB 4.7
MDB
Memcached
min
avg
max90th
max95th
max99th
max

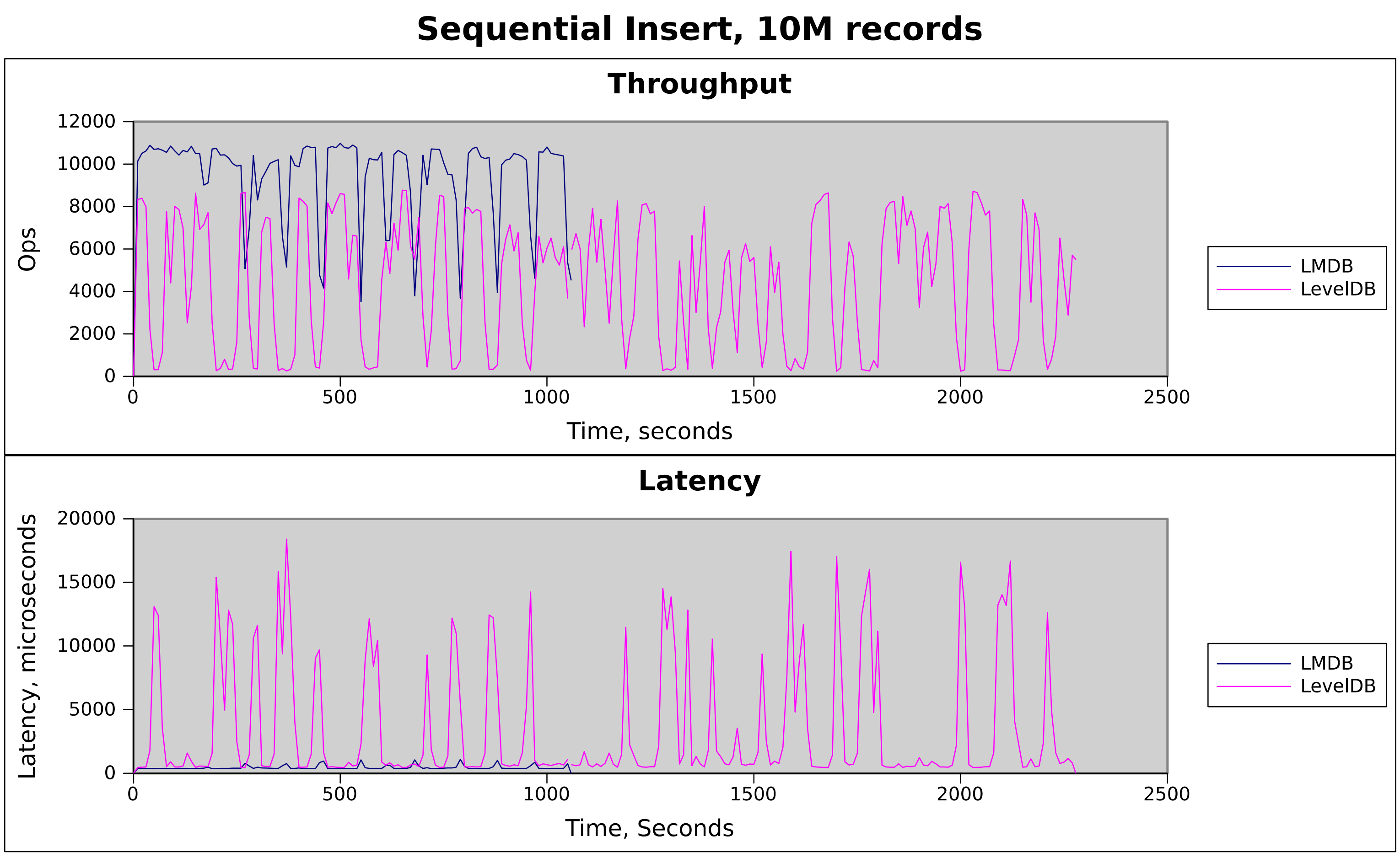


Write Performance
4 Threads, Log Scale
msec
1000
100
10
1
0.1
0.01
BDB 4.7
MDB
Memcached
min
avg
max90th
max95th
max99th
max

HyperDex

- New generation NoSQL database server
  - http://hyperdex.org
  - Simple configuration/deployment
  - Multidimensional indexing/sharding
  - Efficient distributed search engine
  - Built on Google LevelDB, evolved to their fixed version HyperLevelDB
  - Ported to LMDB

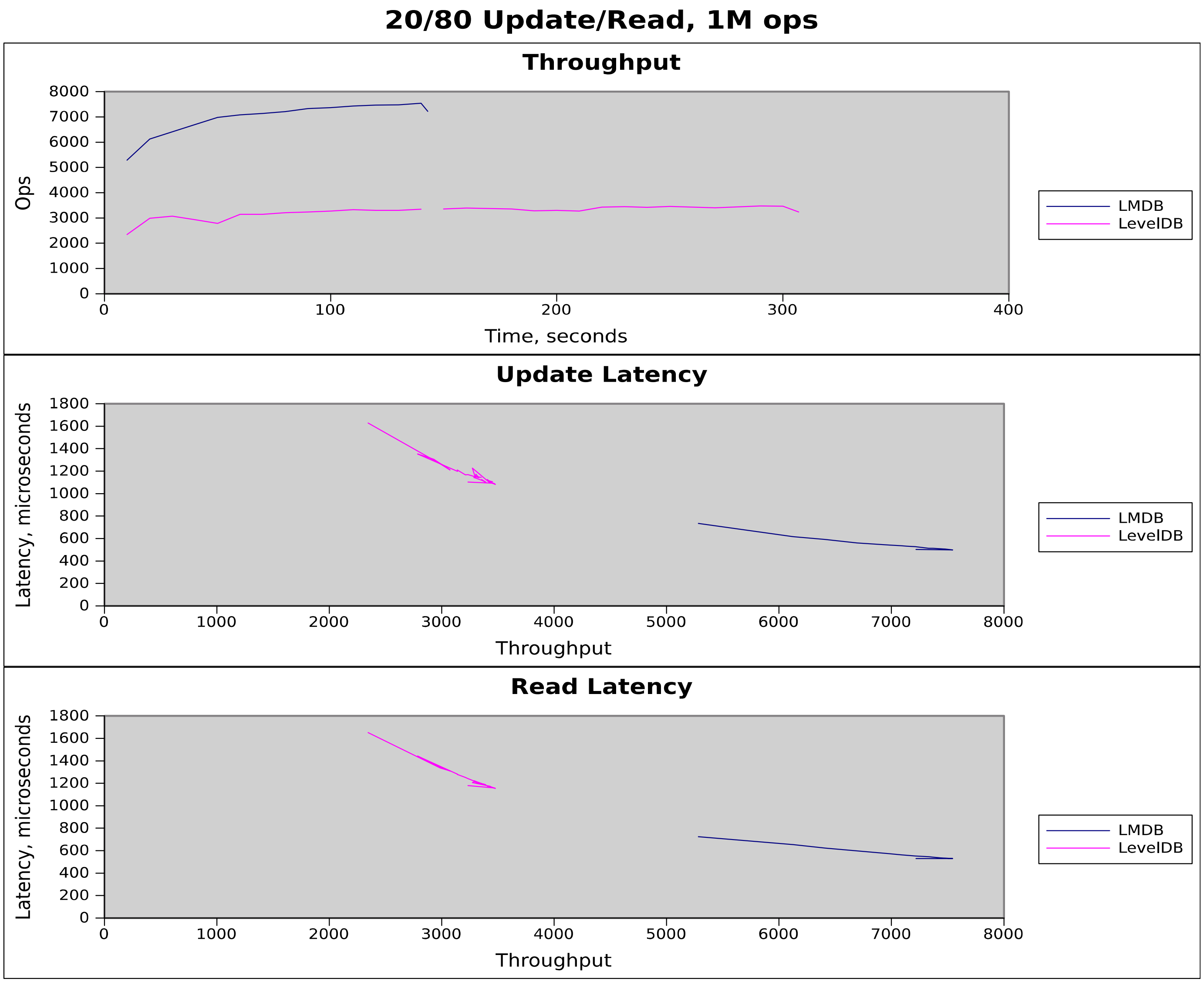
LMDB, HyperDex



LMDB, HyperDex

- CPU time used for inserts :
  - LMDB 19:44.52
  - HyperLevelDB 96:46.96
- HyperLevelDB used 4.9x more CPU for same number of operations
- Again, performance isn't the point. Throwing extra CPU at a job to "make it go faster" is stupid.

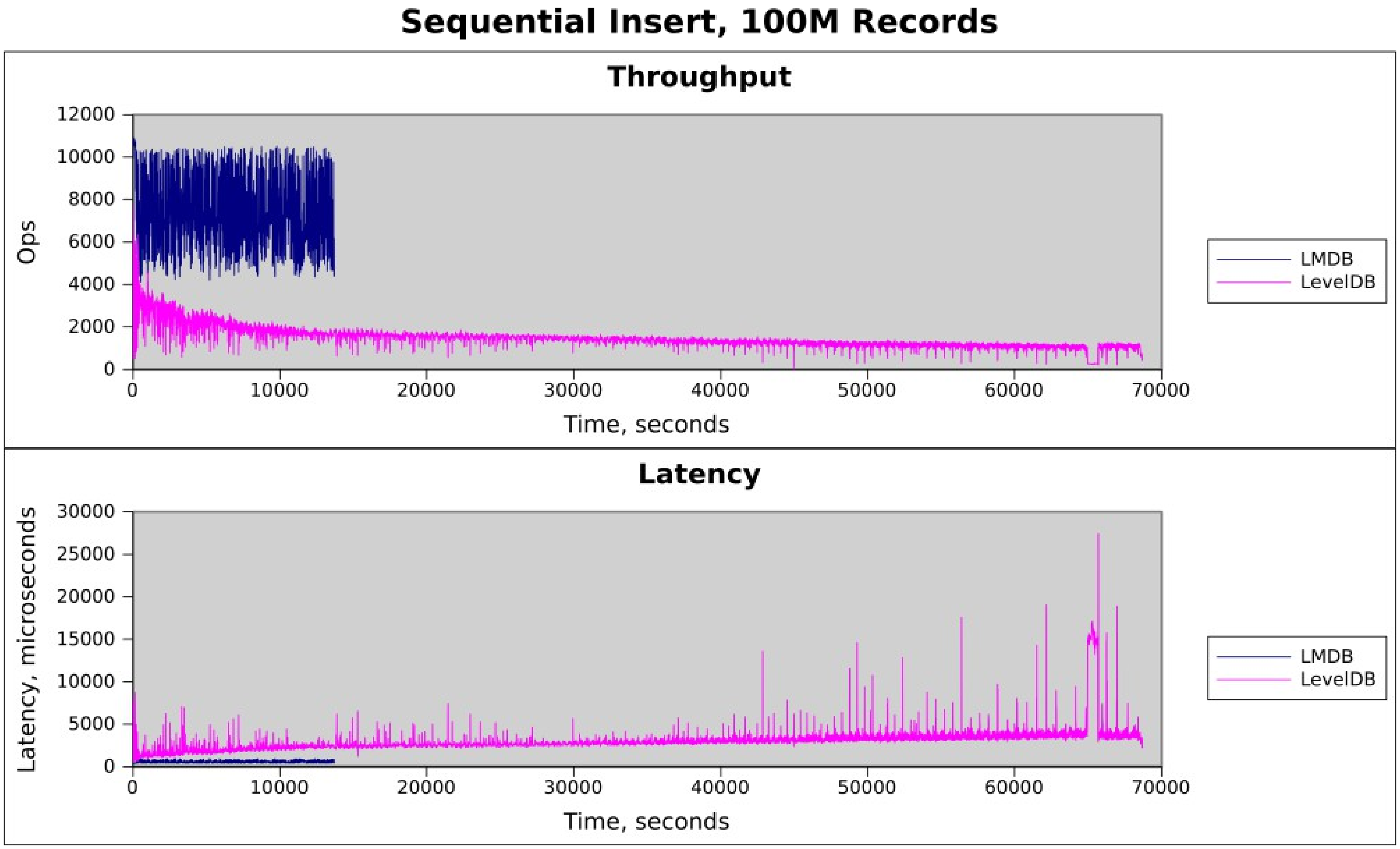
LMDB, HyperDex



LMDB, HyperDex

- CPU time used for read/update :
  - LMDB 1:33.17
  - HyperLevelDB 3:37.67
- HyperLevelDB used 2.3x more CPU for same number of operations

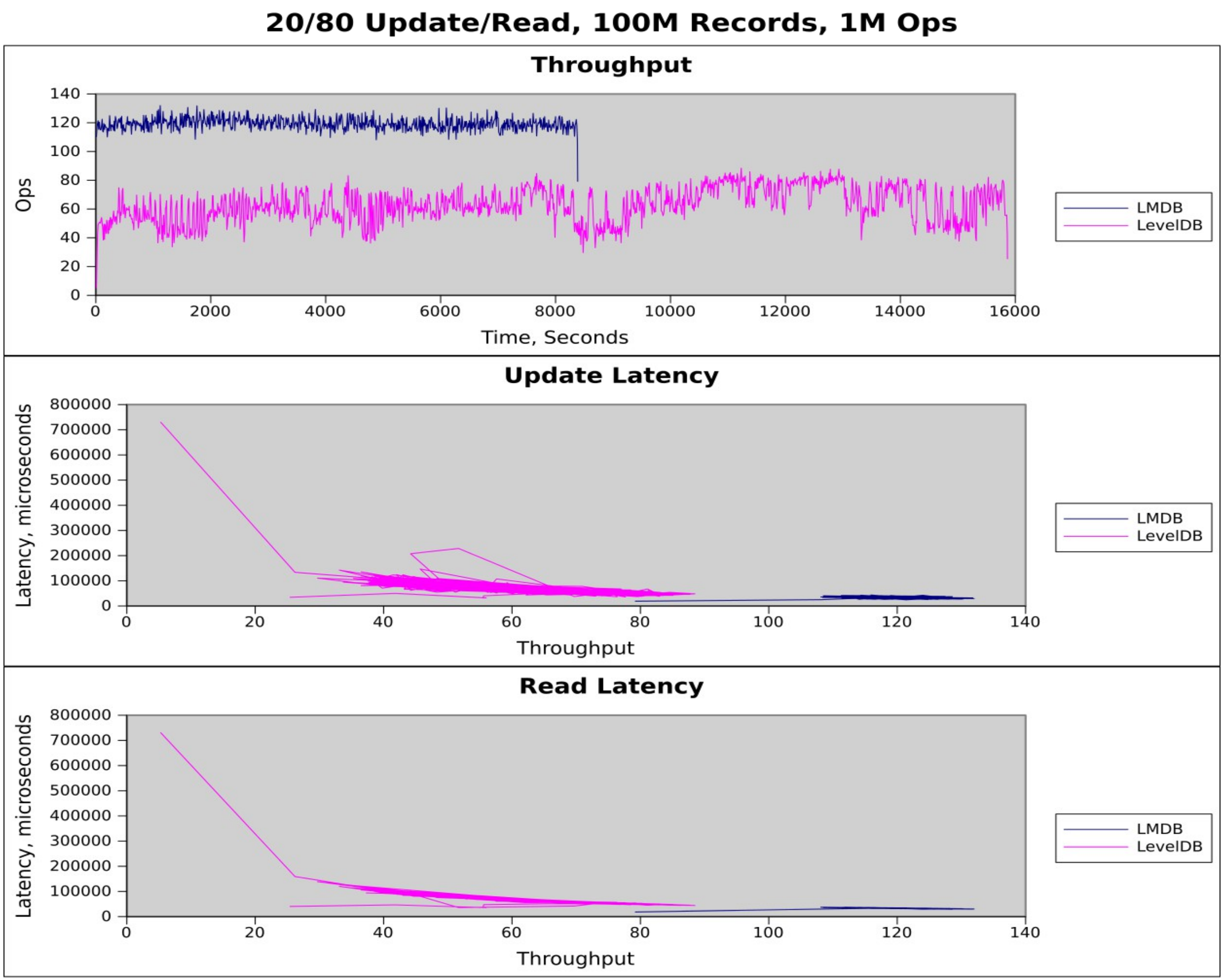
LMDB, HyperDex



LMDB, HyperDex

- CPU time used for inserts :
  - LMDB 227:26
  - HyperLevelDB 3373:13
- HyperLevelDB used 14.8x more CPU for same number of operations

LMDB, HyperDex



LMDB, HyperDex

- CPU time used for read/update :
  - LMDB 4:21.41
  - HyperLevelDB 17:27
- HyperLevelDB used 4.0x more CPU for same number of operations

back-hyperdex

- New clustered backend built on HyperDex
  - Existing back-ndb clustered backend is deprecated, Oracle has refused to cooperate on support
- Nearly complete LDAP support
  - Currently has limited search filter support
  - Uses flat (back-bdb style) namespace, not hierarchical
  - Still in prototype stage as HyperDex API is still in flux

Samba4/AD

- Samba4 provides its own ActiveDirectory-compatible LDAP service
  - built on Samba ldb/tdb libraries
  - supports AD replication
- Has some problems
  - Incompatible with Samba3+OpenLDAP deployments
  - Originally attempted to interoperate with OpenLDAP, but that work was abandoned
  - Poor performance

Samba4/AD

- OpenLDAP interop work revived
  - two opposite approaches being pursued in parallel
    - resurrect original interop code
    - port functionality into slapd overlays
  - currently about 75 % of the test suite passes
  - keep an eye on contrib/slapd-modules/samba4

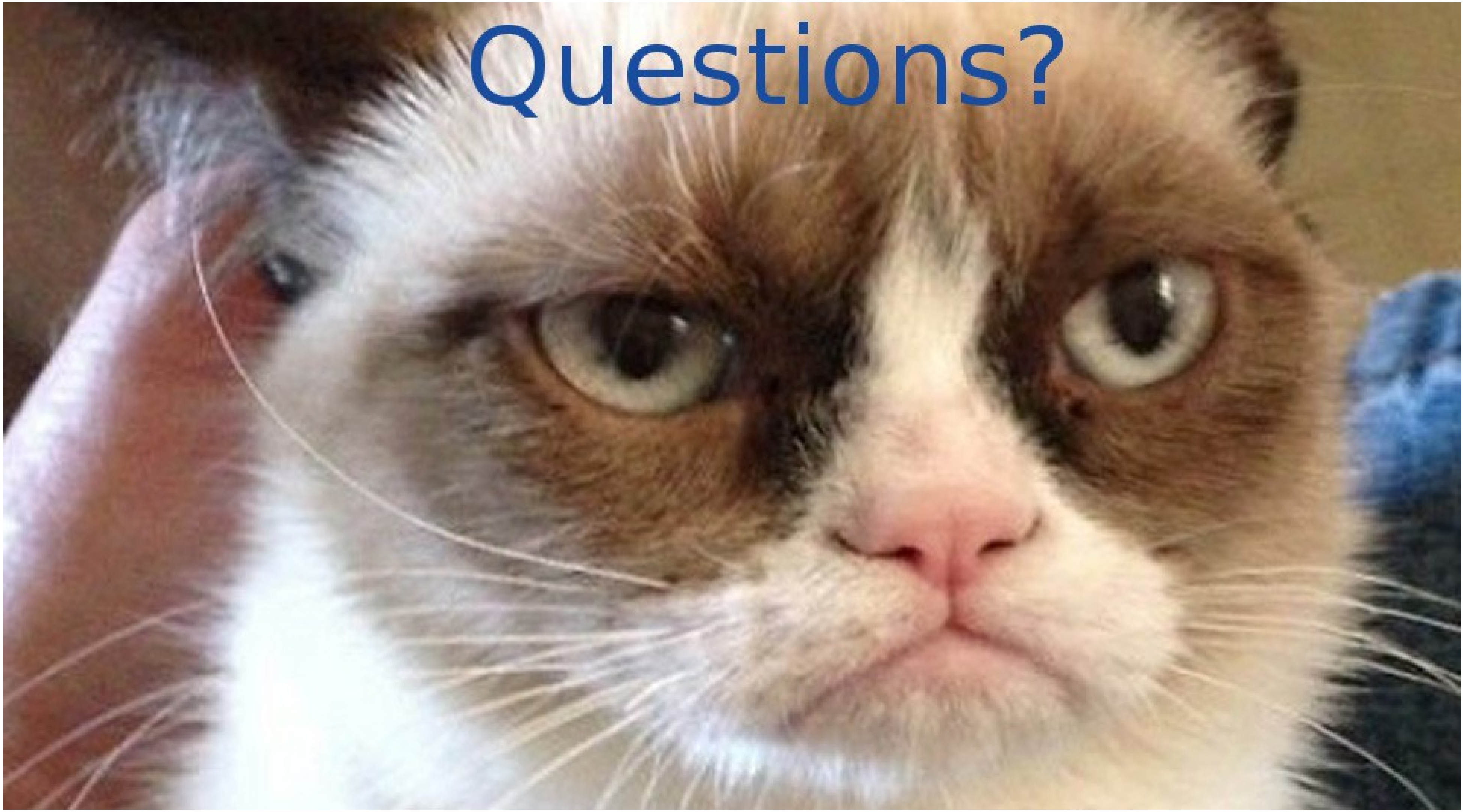
Other Features

- cn=config enhancements
  - Support LDAPDelete op
  - Support slapmodify/slapdelete offline tools
- LDAP transactions
  - Needed for Samba4 support
- Frontend/overlay restructuring
  - Rationalize Bind and ExtendedOp result handling
  - Other internal API cleanup

What's Missing

- Deprecated BerkeleyDB-based backends
  - back-bdb was deprecated in 2.4
  - back-hdb deprecated in 2.5
  - both scheduled for deletion in 2.6
  - configure switches renamed, so existing packager scripts can no longer enable them without explicit action

Questions?



Thanks!