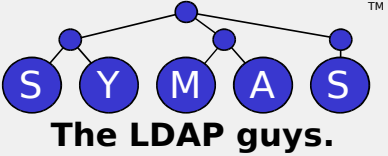
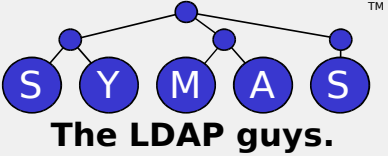


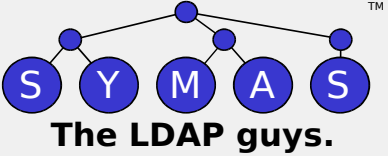
MDB: A Memory-Mapped Database and Backend for OpenLDAP

Howard Chu
CTO, Symas Corp. hyc@symas.com
Chief Architect, OpenLDAP hyc@openldap.org



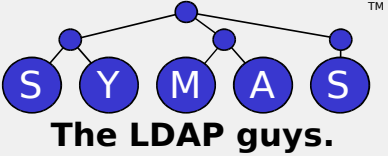
OpenLDAP Project

- Open source code project
- Founded 1998
- Three core team members
- A dozen or so contributors
- Feature releases every 18-24 months
- Maintenance releases as needed



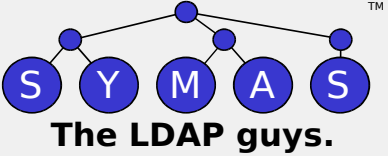
A Word About Symas

- 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



Topics

- Overview
- Background / History
- Obvious Solutions
- Future Directions



Overview

- OpenLDAP has been delivering reliable, high performance for many years
- The performance comes at the cost of fairly complex tuning requirements
- The implementation is not as clean as it could be; it is not what was originally intended
- Cleaning it up requires not just a new server backend, but also a new low-level database
- The new approach has a huge payoff

Background

- OpenLDAP already provides a number of reliable, high performance transactional backends
  - Based on Oracle BerkeleyDB (BDB)
  - back-bdb released with OpenLDAP 2.1 in 2002
  - back-hdb released with OpenLDAP 2.2 in 2003
  - Intensively analyzed for performance
  - World's fastest since 2005
  - Many heavy users with zero downtime

Background

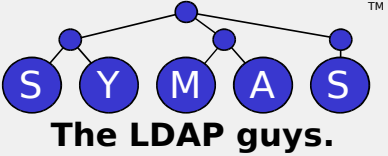
- These backends have always required careful, complex tuning
  - Data comes through three separate layers of caches
  - Each cache layer has different size and speed characteristics
  - Balancing the three layers against each other can be a difficult juggling act
  - Performance without the backend caches is unacceptably slow - over an order of magnitude...

Background

- The backend caching significantly increased the overall complexity of the backend code
  - Two levels of locking required, since the BDB database locks are too slow
  - Deadlocks occurring routinely in normal operation, requiring additional backoff/retry logic

Background

- The caches were not always beneficial, and were sometimes detrimental
  - data could exist in 3 places at once - filesystem, database, and backend cache - thus wasting memory
  - searches with result sets that exceeded the configured cache size would reduce the cache effectiveness to zero
  - malloc/free churn from adding and removing entries in the cache could trigger pathological heap behavior in libc malloc



Background

- Overall the backends require too much attention
  - Too much developer time spent finding workarounds for inefficiencies
  - Too much administrator time spent tweaking configurations and cleaning up database transaction logs

Obvious Solutions

- Cache management is a hassle, so don't do any caching
  - The filesystem already caches data, there's no reason to duplicate the effort
- Lock management is a hassle, so don't do any locking
  - Use Multi-Version Concurrency Control (MVCC)
  - MVCC makes it possible to perform reads with no locking

Obvious Solutions

- BDB supports MVCC, but it still requires complex caching and locking
- To get the desired results, we need to abandon BDB
- Surveying the landscape revealed no other database libraries with the desired characteristics
- Time to write our own...

MDB Approach

- Based on the "Single-Level Store" concept
  - Not new, first implemented in Multics in 1964
  - Access a database by mapping the entire database into memory
  - Data fetches are satisfied by direct reference to the memory map, there is no intermediate page or buffer cache

Single-Level Store

- The approach is only viable if process address spaces are larger than the expected data volumes
  - For 32 bit processors, the practical limit on data size is under 2GB
  - For common 64 bit processors which only implement 48 bit address spaces, the limit is 47 bits or 128 terabytes
  - The upper bound at 63 bits is 8 exabytes

MDB Approach

- Uses a read-only memory map
  - Protects the database structure from corruption due to stray writes in memory
  - Any attempts to write to the map will cause a SEGV, allowing immediate identification of software bugs
  - There's no point in making the pages writable anyway, since only existing pages may be written. Growing the database requires file ops (write, ftruncate) so for uniformity, file ops are also used for updates.

MDB Approach

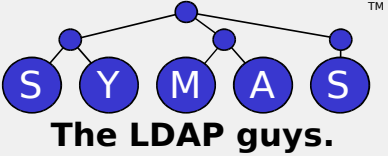
- Implement MVCC using copy-on-write
  - In-use data is never overwritten, modifications are performed by copying the data and modifying the copy
  - Since updates never alter existing data, the database structure can never be corrupted by incomplete modifications
    - Write-ahead transaction logs are unnecessary
  - Readers always see a consistent snapshot of the database, they are fully isolated from writers
    - Read accesses require no locks

MVCC Details

- "Full" MVCC can be extremely resource intensive
  - Databases typically store complete histories reaching far back into time
  - The volume of data grows extremely fast, and grows without bound unless explicit pruning is done
  - Pruning the data using garbage collection or compaction requires more CPU and I/O resources than the normal update workload
    - Either the server must be heavily over-provisioned, or updates must be stopped while pruning is done
  - Pruning requires tracking of in-use status, which typically involves reference counters, which require locking

MDB Approach

- MDB nominally maintains only two versions of the database
  - Rolling back to a historical version is not interesting for OpenLDAP
  - Older versions can be held open longer by reader transactions
- MDB maintains a free list tracking the IDs of unused pages
  - Old pages are reused as soon as possible, so data volumes don't grow without bound
- MDB tracks in-use status without locks

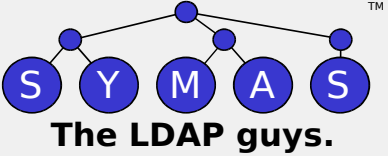


Implementation Highlights

- MDB library started from the append-only btree code written by Martin Hedenfalk for his ldapd, which is bundled in OpenBSD
  - Stripped out all the parts we didn't need (page cache management)
  - Borrowed a couple pieces from back-bdb for expedience
  - Changed from append-only to page-reclaiming
  - Restructured to allow adding ideas from BDB that we still wanted

Implementation Highlights

- Resulting library was under 32KB of object code
  - Compared to the original btree.c at 39KB
  - Compared to BDB at 1.5MB
- API is loosely modeled after the BDB API to ease migration of back-bdb code to use MDB



Btree Operation

Basic Elements

Database Page

Pgno
Misc...

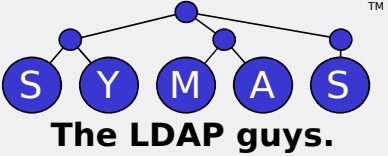
Meta Page

Pgno
Misc...
Root

Data Page

Pgno
Misc...
offset

key, data



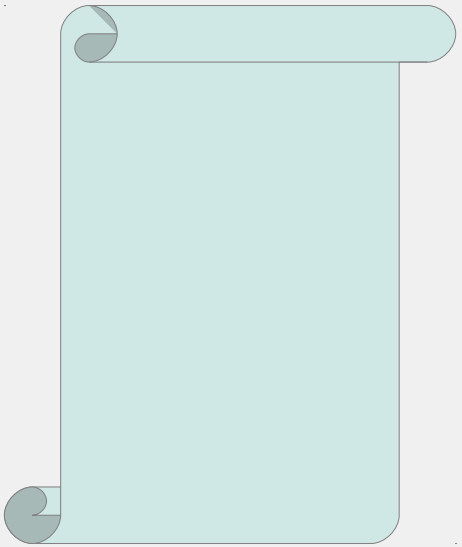
Btree Operation

Write-Ahead Logger

Meta Page

Pgno: 0
Misc...
Root : EMPTY

Write-Ahead Log



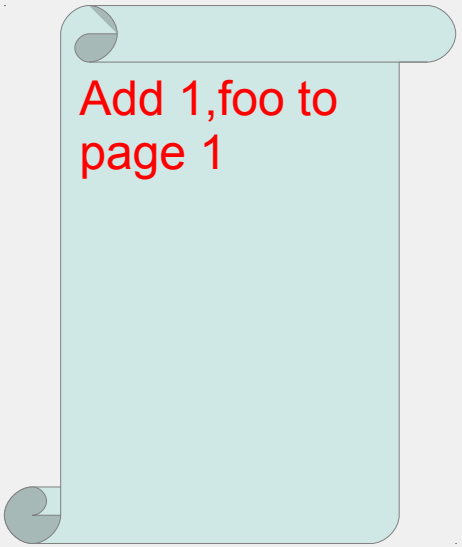
Btree Operation

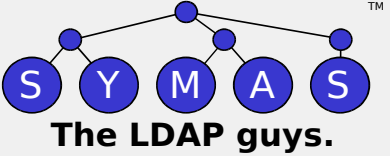
Write-Ahead Logger

Meta Page

Pgno: 0
Misc...
Root : EMPTY

Write-Ahead Log





Btree Operation

Write-Ahead Logger

Meta Page

Pgno: 0
Misc...
Root : 1

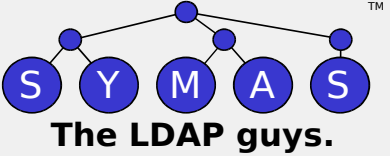
Data Page

Pgno: 1
Misc...
offset: 4000

1,foo

Write-Ahead Log

Add 1,foo to page 1



Btree Operation

Write-Ahead Logger

Meta Page

Pgno: 0
Misc...
Root : 1

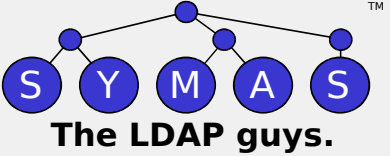
Data Page

Pgno: 1
Misc...
offset: 4000

1,foo

Write-Ahead Log

Add 1,foo to page 1
Commit



Btree Operation

Write-Ahead Logger

Meta Page

Pgno: 0
Misc...
Root : 1

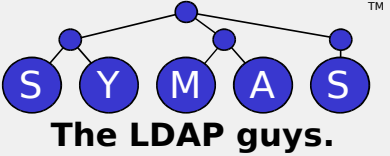
Data Page

Pgno: 1
Misc...
offset: 4000

1,foo

Write-Ahead Log

Add 1,foo to page 1
Commit
Add 2,bar to page 1



Btree Operation

Write-Ahead Logger

Meta Page

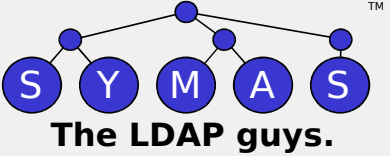
Pgno: 0
Misc...
Root : 1

Data Page

Pgno: 1
Misc...
offset: 4000
offset: 3000
2,bar
1,foo

Write-Ahead Log

Add 1,foo to page 1
Commit
Add 2,bar to page 1



Btree Operation

Write-Ahead Logger

Meta Page

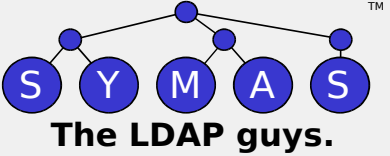
Pgno: 0
Misc...
Root : 1

Data Page

Pgno: 1
Misc...
offset: 4000
offset: 3000
2,bar
1,foo

Write-Ahead Log

Add 1,foo to page 1
Commit
Add 2,bar to page 1
Commit



Btree Operation

Write-Ahead Logger

RAM

Meta Page

Pgno: 0
Misc...
Root : 1

Data Page

Pgno: 1
Misc...
offset: 4000
offset: 3000
2,bar
1,foo

Write-Ahead Log

Add 1,foo to page 1
Commit
Add 2,bar to page 1
Commit
Checkpoint

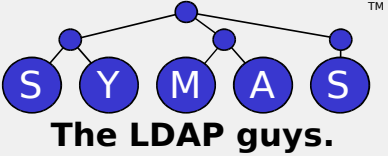
Disk

Meta Page

Pgno: 0
Misc...
Root : 1

Data Page

Pgno: 1
Misc...
offset: 4000
offset: 3000
2,bar
1,foo



Btree Operation

Append-Only

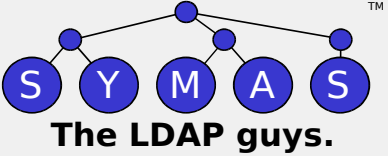
Meta Page

Pgno: 0
Misc...
Root : EMPTY

Btree Operation

Append-Only

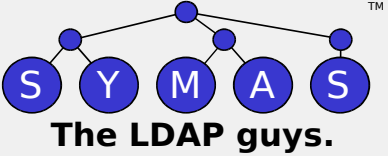
Meta Page	Data Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo



Btree Operation

Append-Only

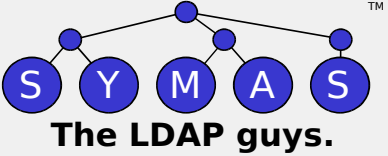
Meta Page	Data Page	Meta Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1



Btree Operation

Append-Only

Meta Page	Data Page	Meta Page	Data Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo



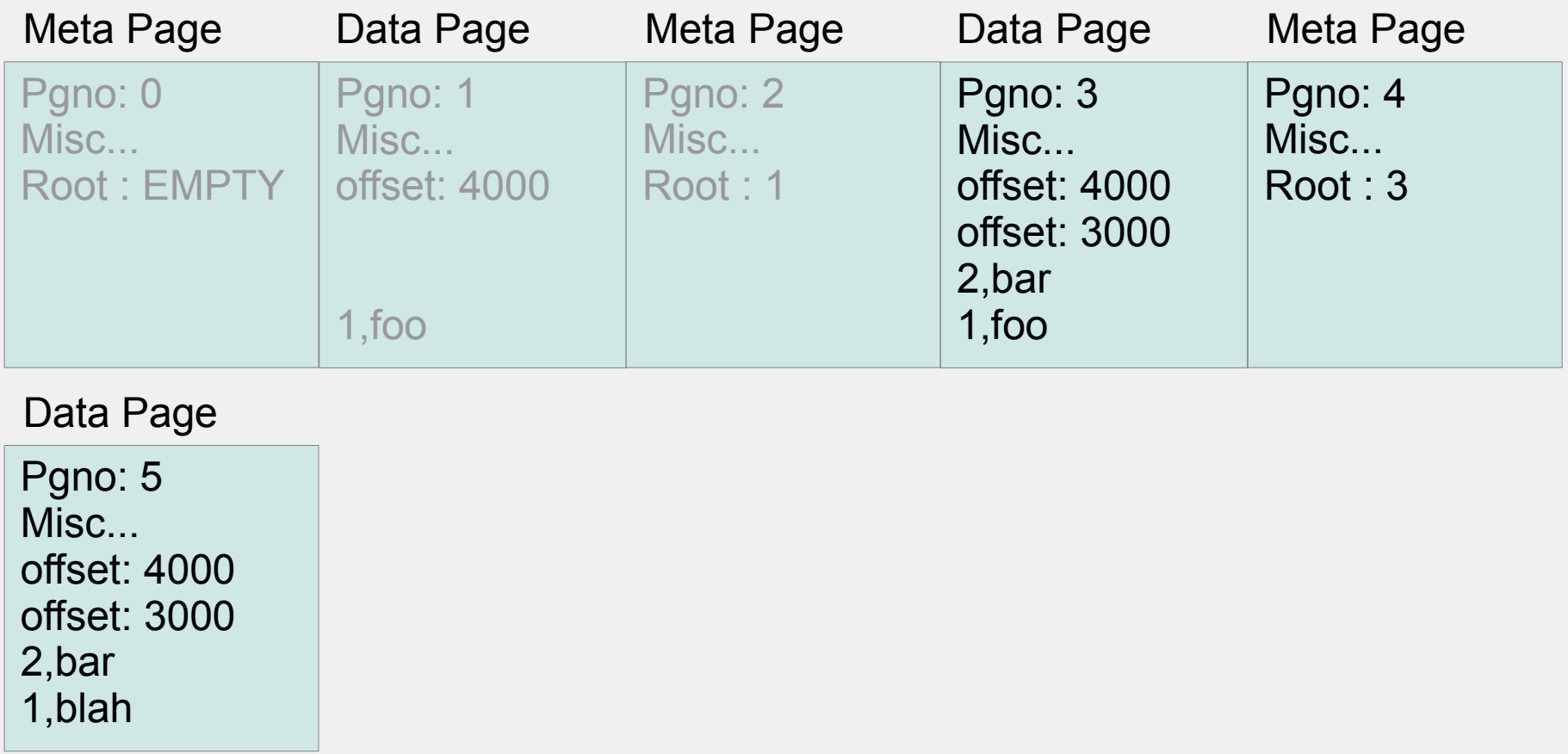
Btree Operation

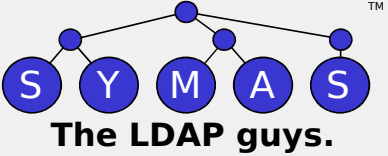
Append-Only

Meta Page	Data Page	Meta Page	Data Page	Meta Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... Root : 3

Btree Operation

Append-Only





Btree Operation

Append-Only

Meta Page	Data Page	Meta Page	Data Page	Meta Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... Root : 3

Data Page	Meta Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... Root : 5

Btree Operation

Append-Only

Meta Page	Data Page	Meta Page	Data Page	Meta Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... Root : 3

Data Page	Meta Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... Root : 5	Pgno: 7 Misc... offset: 4000 offset: 3000 2,xyz 1,blah

Btree Operation

Append-Only

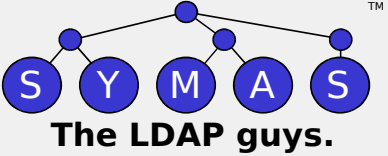
Meta Page	Data Page	Meta Page	Data Page	Meta Page
Pgno: 0 Misc... Root : EMPTY	Pgno: 1 Misc... offset: 4000 1,foo	Pgno: 2 Misc... Root : 1	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... Root : 3

Data Page	Meta Page	Data Page	Meta Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... Root : 5	Pgno: 7 Misc... offset: 4000 offset: 3000 2,xyz 1,blah	Pgno: 8 Misc... Root : 7

Btree Operation

MDB

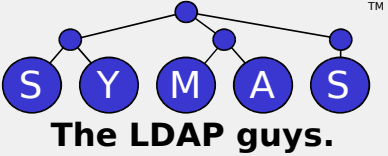
Meta Page	Meta Page
Pgno: 0 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 1 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY



Btree Operation

MDB

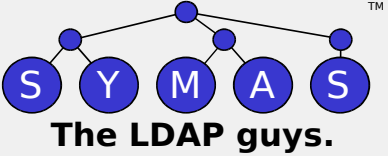
Meta Page	Meta Page	Data Page
Pgno: 0 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 1 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 2 Misc... offset: 4000 1,foo



Btree Operation

MDB

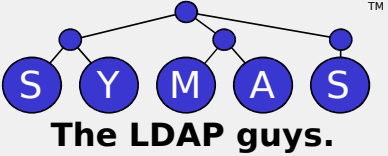
Meta Page	Meta Page	Data Page
Pgno: 0 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo



Btree Operation

MDB

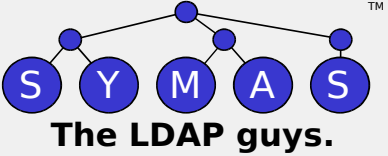
Meta Page	Meta Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo



Btree Operation

MDB

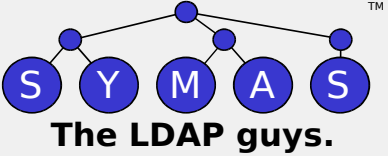
Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 0 FRoot: EMPTY DRoot: EMPTY	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2



Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2



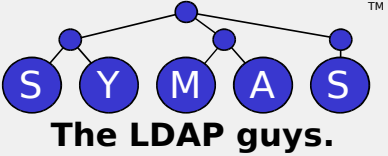
Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page

Pgno: 5
Misc...
offset: 4000
offset: 3000
2,bar
1,blah

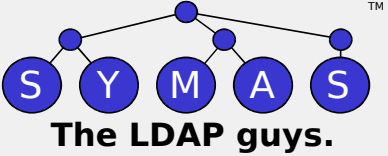


Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 1 FRoot: EMPTY DRoot: 2	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... offset: 4000 offset: 3000 txn 3,page 3,4 txn 2,page 2

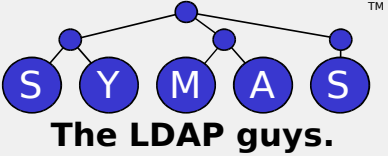


Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 3 FRoot: 6 DRoot: 5	Pgno: 2 Misc... offset: 4000 1,foo	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... offset: 4000 offset: 3000 txn 3,page 3,4 txn 2,page 2

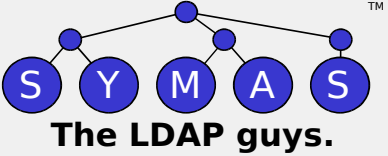


Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 3 FRoot: 6 DRoot: 5	Pgno: 2 Misc... offset: 4000 offset: 3000 2,xyz 1,blah	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... offset: 4000 offset: 3000 txn 3,page 3,4 txn 2,page 2

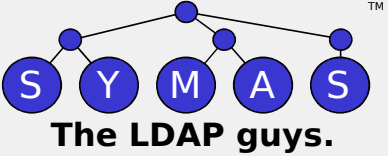


Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 2 FRoot: 4 DRoot: 3	Pgno: 1 Misc... TXN: 3 FRoot: 6 DRoot: 5	Pgno: 2 Misc... offset: 4000 offset: 3000 2,xyz 1,blah	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page	Data Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... offset: 4000 offset: 3000 txn 3,page 3,4 txn 2,page 2	Pgno: 7 Misc... offset: 4000 offset: 3000 txn 4,page 5,6 txn 3,page 3,4



Btree Operation

MDB

Meta Page	Meta Page	Data Page	Data Page	Data Page
Pgno: 0 Misc... TXN: 4 FRoot: 7 DRoot: 2	Pgno: 1 Misc... TXN: 3 FRoot: 6 DRoot: 5	Pgno: 2 Misc... offset: 4000 offset: 3000 2,xyz 1,blah	Pgno: 3 Misc... offset: 4000 offset: 3000 2,bar 1,foo	Pgno: 4 Misc... offset: 4000 txn 2,page 2

Data Page	Data Page	Data Page
Pgno: 5 Misc... offset: 4000 offset: 3000 2,bar 1,blah	Pgno: 6 Misc... offset: 4000 offset: 3000 txn 3,page 3,4 txn 2,page 2	Pgno: 7 Misc... offset: 4000 offset: 3000 txn 4,page 5,6 txn 3,page 3,4

Implementation Highlights

- back-mdb code is based on back-bdb/hdb
  - Copied the back-bdb source tree
  - Deleted all cache-management code
  - Adapted to MDB API
  - Source comprises 340KB, compared to 476KB for back-bdb/hdb - 30% smaller
  - Nothing sacrificed - supports all the same features as back-hdb

Tradeoffs?

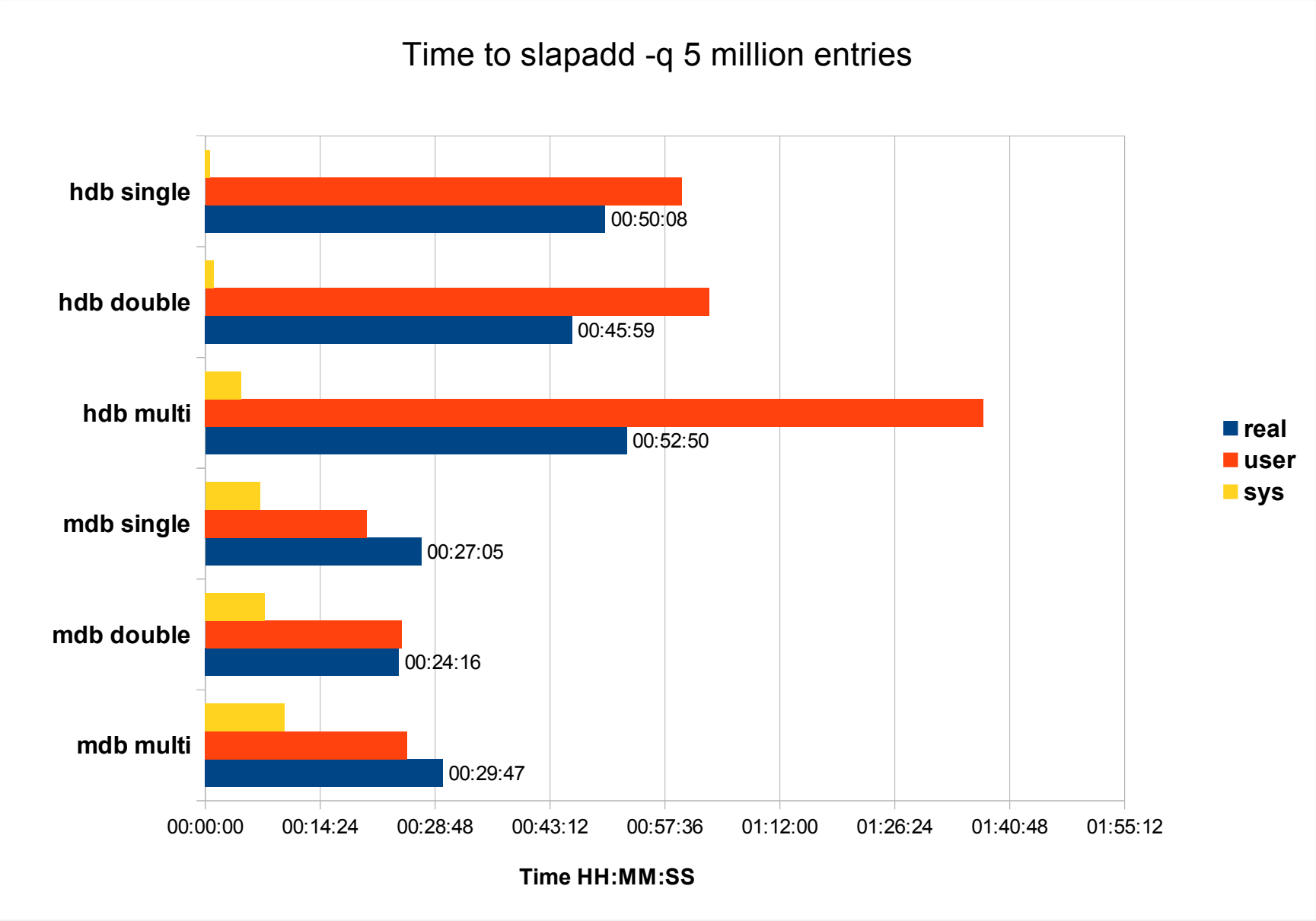
- Dropping the entry cache means we incur a cost to decode entries on every query, instead of simply operating on a fully-decoded entry in a cache
  - No problem, just make entry decoding in back-mdb cost less than an entry cache access in back-hdb
- The copy-on-write approach allows only a single writer at a time
  - BDB allows multiple concurrent writers
  - Currently MDB has much lower write throughput

Results

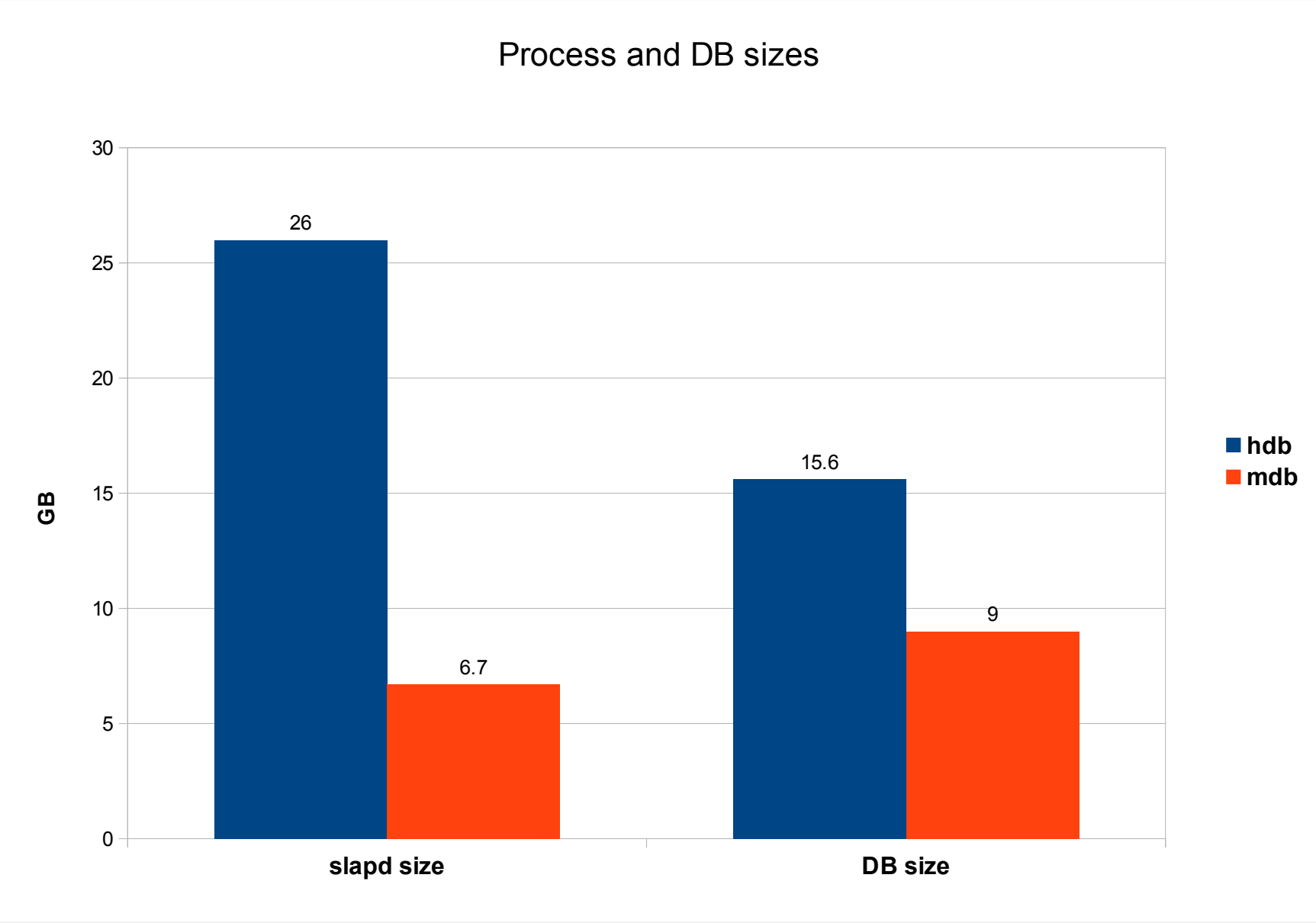
- Disclaimers
  - Due to the addition of multi-threaded tests, tcmalloc was used for the slapadd results
  - Multi-threaded slapadd results using regular malloc are much slower

The LDAP guys.

Results



Results

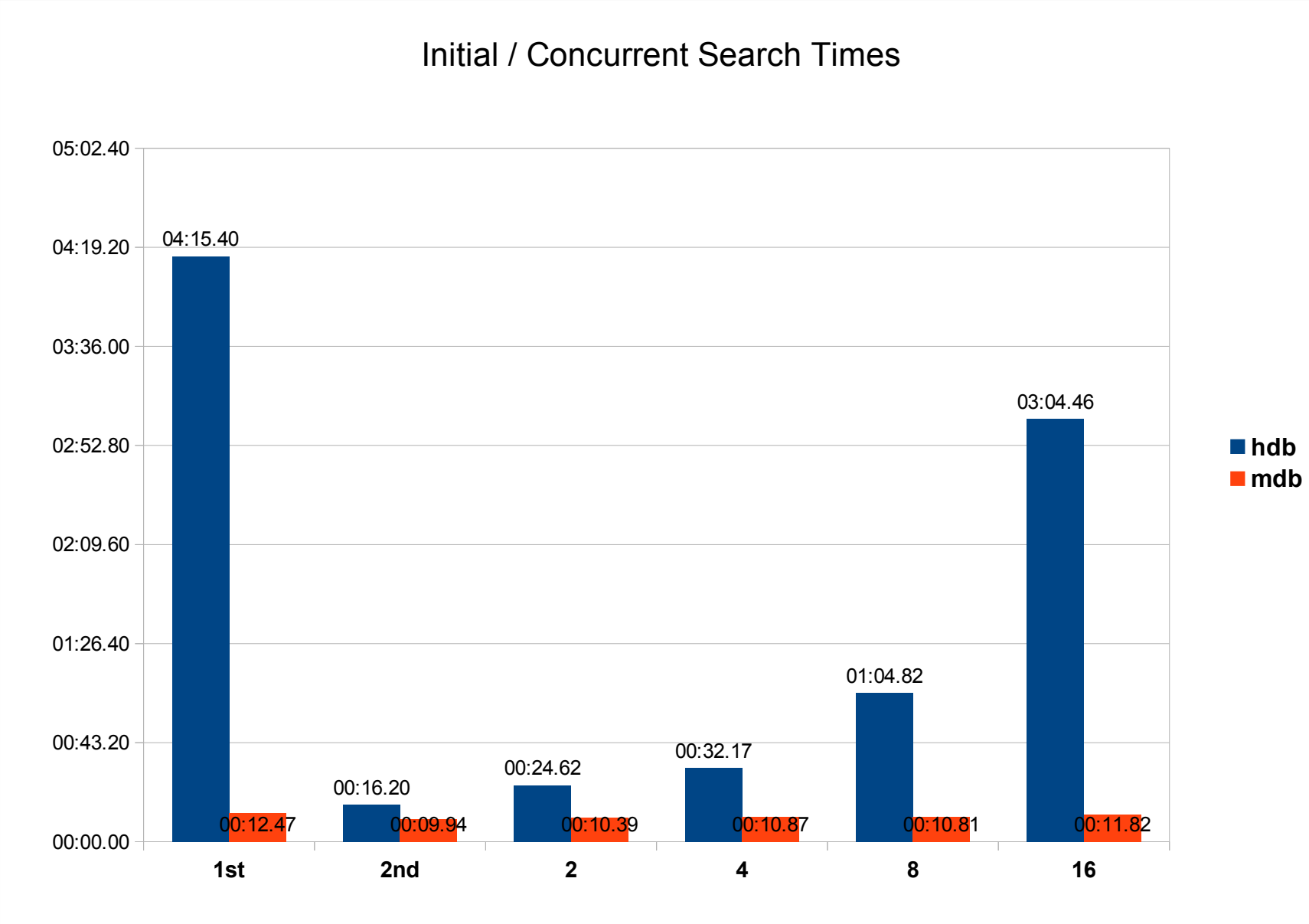

Process and DB sizes
GB
30
25
20
15
10
5
0
26
6.7
15.6
9
slapd size
DB size
hdb
mdb

Results

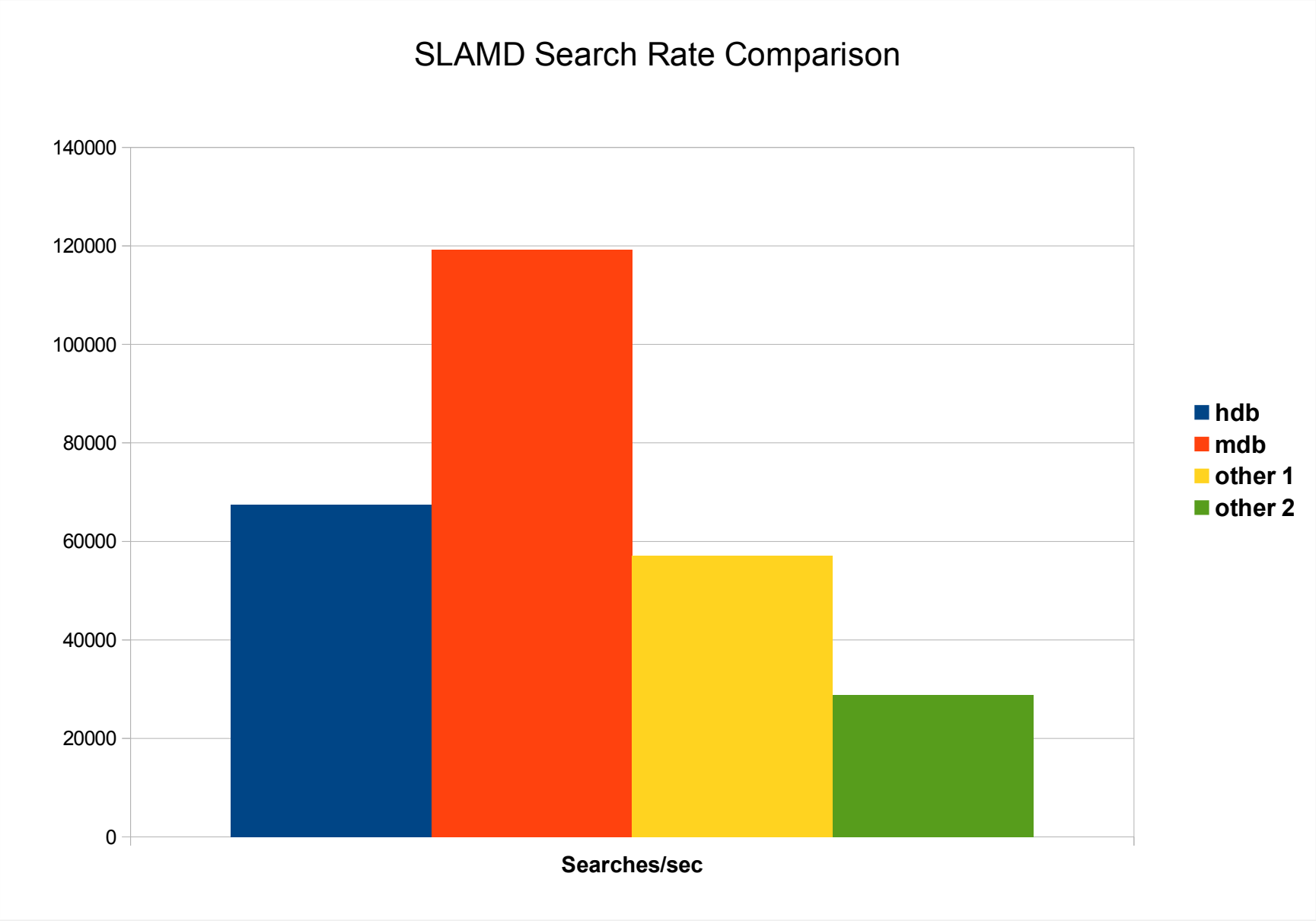
- Development has been very rapid
  - First benchmarks published 2011-09-30
- slapadd as of 2011-10-07 notably improved
  - ~9% faster single-threaded
  - ~18% faster double-threaded (pipelined)
  - DB size is ~30% smaller
    - Higher page fill factor, less wasted space
    - slapd process size down to only ~25% of back-hdb size

The LDAP guys.

Results

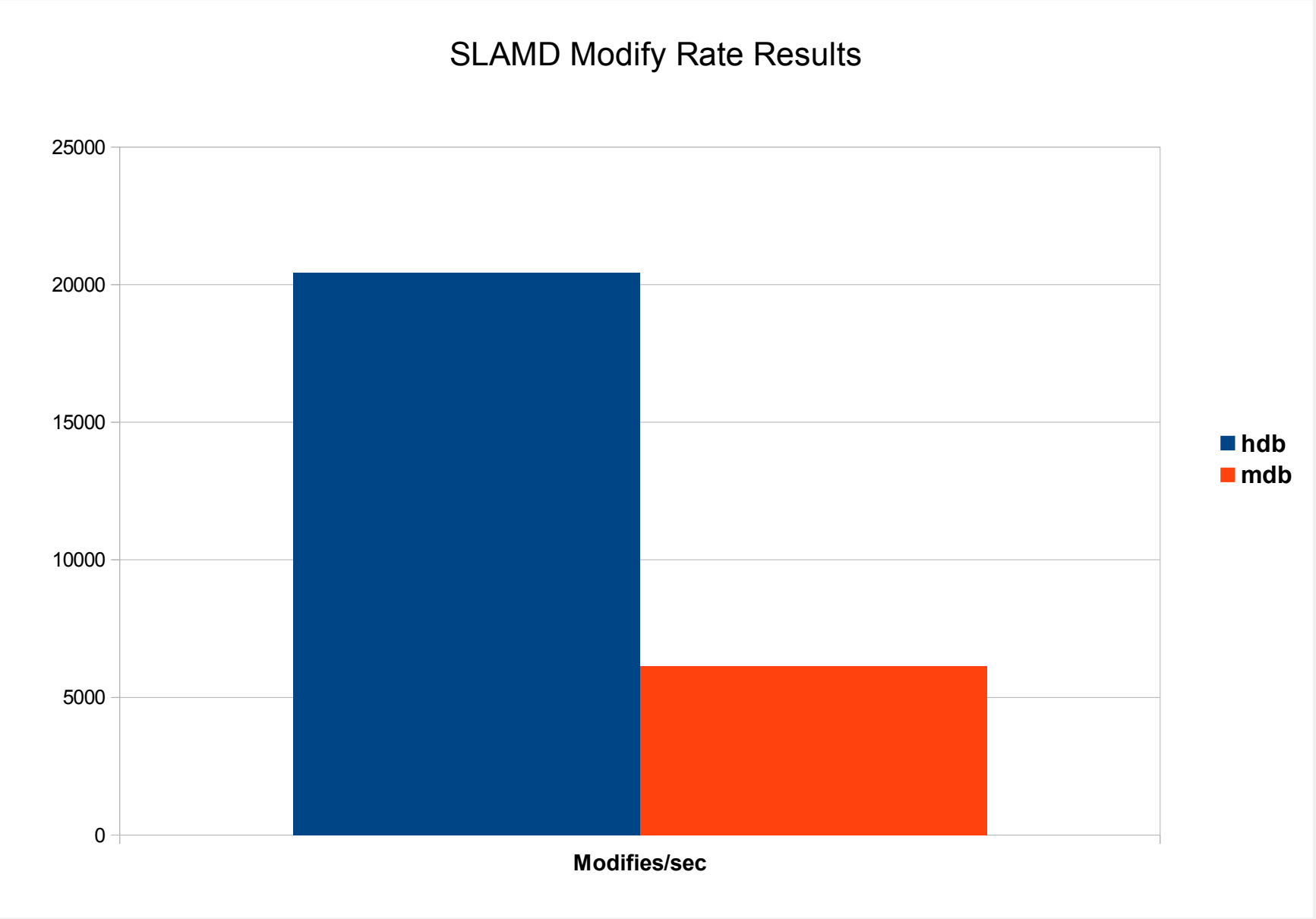


Results

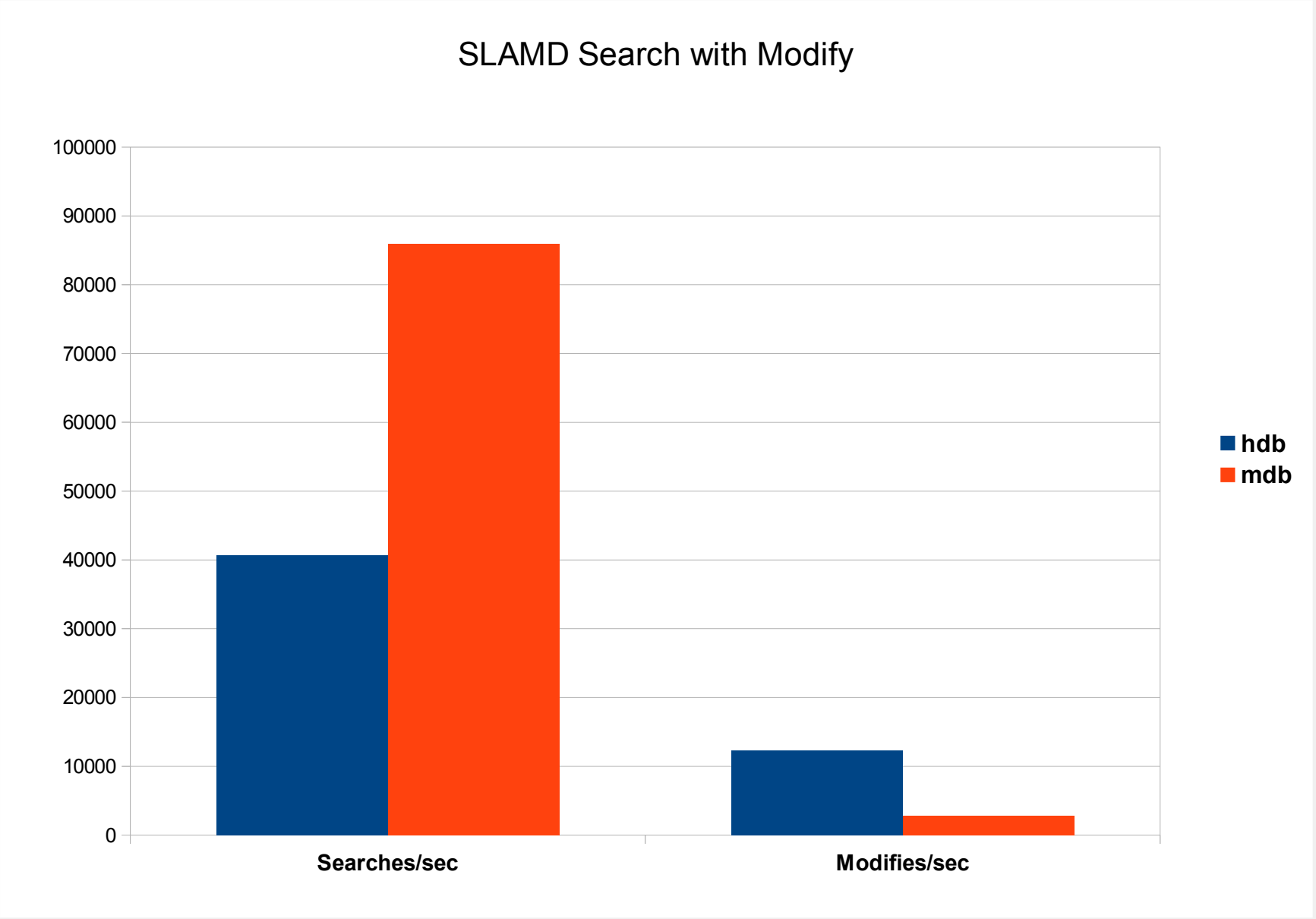

SLAMD Search Rate Comparison
140000
120000
100000
80000
60000
40000
20000
0
Searches/sec
hdb
mdb
other 1
other 2

The LDAP guys.

Results



Results


SLAMD Search with Modify
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
Searches/sec
Modifies/sec
hdb
mdb

Conclusions

- The combination of memory-mapped operation with MVCC is extremely potent for LDAP
  - Reduced administrative overhead
    - no periodic cleanup / maintenance required
    - no particular tuning required
  - Reduced developer overhead
    - code size and complexity drastically reduced
  - Enhanced efficiency
    - read performance is significantly increased

Conclusions

- The MDB approach is not just a one-off solution
  - While initially developed on desktop Linux, it has also been ported to Windows, MacOSX, and Android with no particular difficulty
  - While developed specifically for OpenLDAP, porting to other code bases is also under way
    - A port to SQLite 3.7.1 is available on gitorious
    - Replacements for BerkeleyDB in Cyrus-SASL, Heimdal are available
    - Replacement for BerkeleyDB in perl DBD planned
    - There are probably many other suitable applications for a small-footprint database library with low write rates and near-zero read overhead

Future Work

- A number of items remain on the TODO list
  - Investigate optimizations for write performance
  - Allow database max size and other settings to be grown dynamically instead of statically configured
  - Functions to facilitate incremental and/or full backups
  - Storing back-mdb entries in native format, so no decoding is needed at all
- None of these are show-stoppers, MDB and back-mdb already meet or exceed all expectations

Where Next?

- mdb code was released in 2.4.27 on an experimental basis (2011-11-24)
  - Some minor bug fixes were added in 2.4.28 and 2.4.29
- Symas Corp. is offering (for a limited time) 6 months free support to anyone using back-mdb
- Internal testing and benchmarking continues...

SQLite Footnotes

- Time to Insert 1000 random records
  - Original SQLite 3.7.7.1: 22.43 seconds
  - SQLite with MDB: 1.06 seconds
- Time to read (Select) 1000 records
  - Difference unmeasurable
  - Instruction-level profile shows >95% of execution time is spent in SQL parsing, only 2-3% in actual DB fetch

SQLite Footnotes

- How "Lite" is SQLite?
  - MDB is over an order of magnitude more efficient for writes
  - Writes are most power intensive
  - The big story here isn't the absolute speed, it's the efficiency - SQLite is used extensively in smartphones, tablets, and other devices
  - a 23:1 improvement in write efficiency translates to a significant gain in battery life