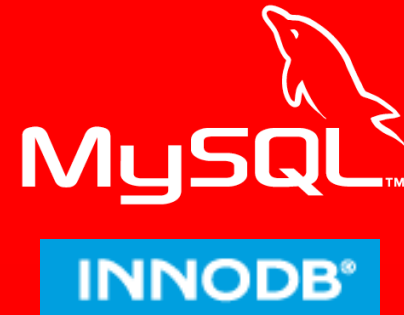
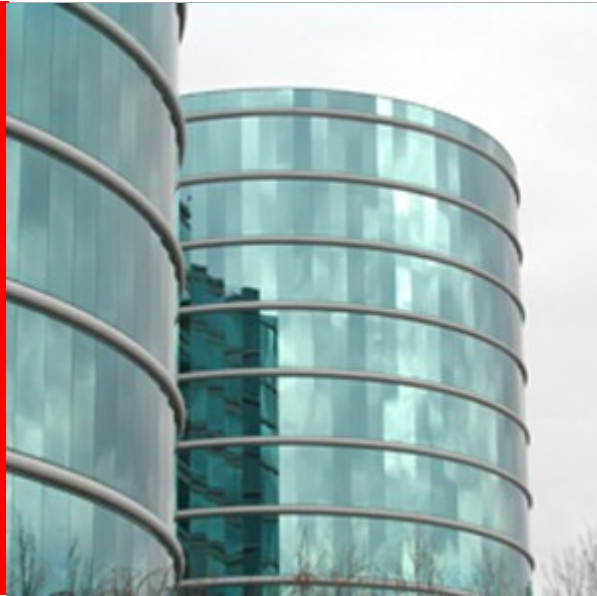


ORACLE®



InnoDB Plugin: Performance Features and Benchmarks

MySQL Conference and Expo, 2010

10:50 am, April 15, 2010

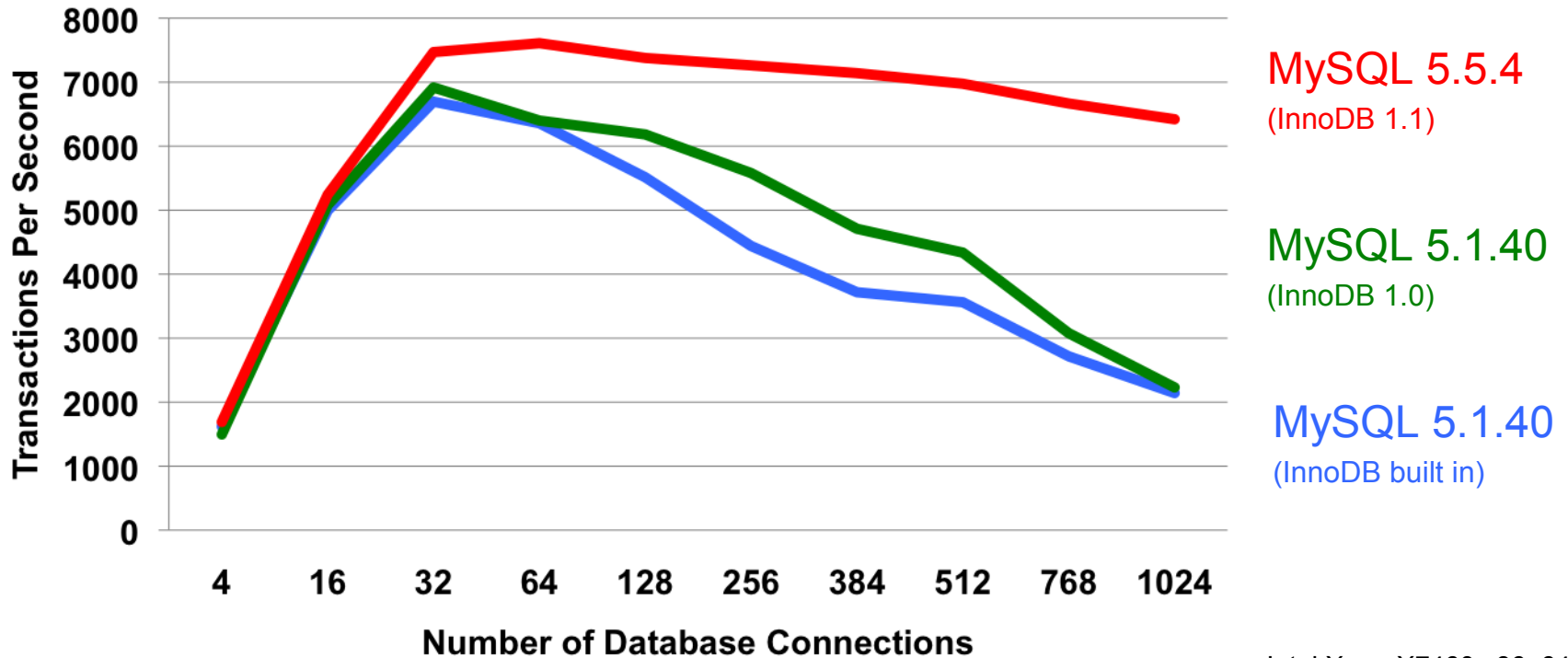
Jimmy Yang
InnoDB@Oracle

John Russell
InnoDB@Oracle

Calvin Sun
InnoDB@Oracle

MySQL 5.5/InnoDB 1.1 SysBench Benchmarks

MySQL 5.5 vs. 5.1 - Read Only



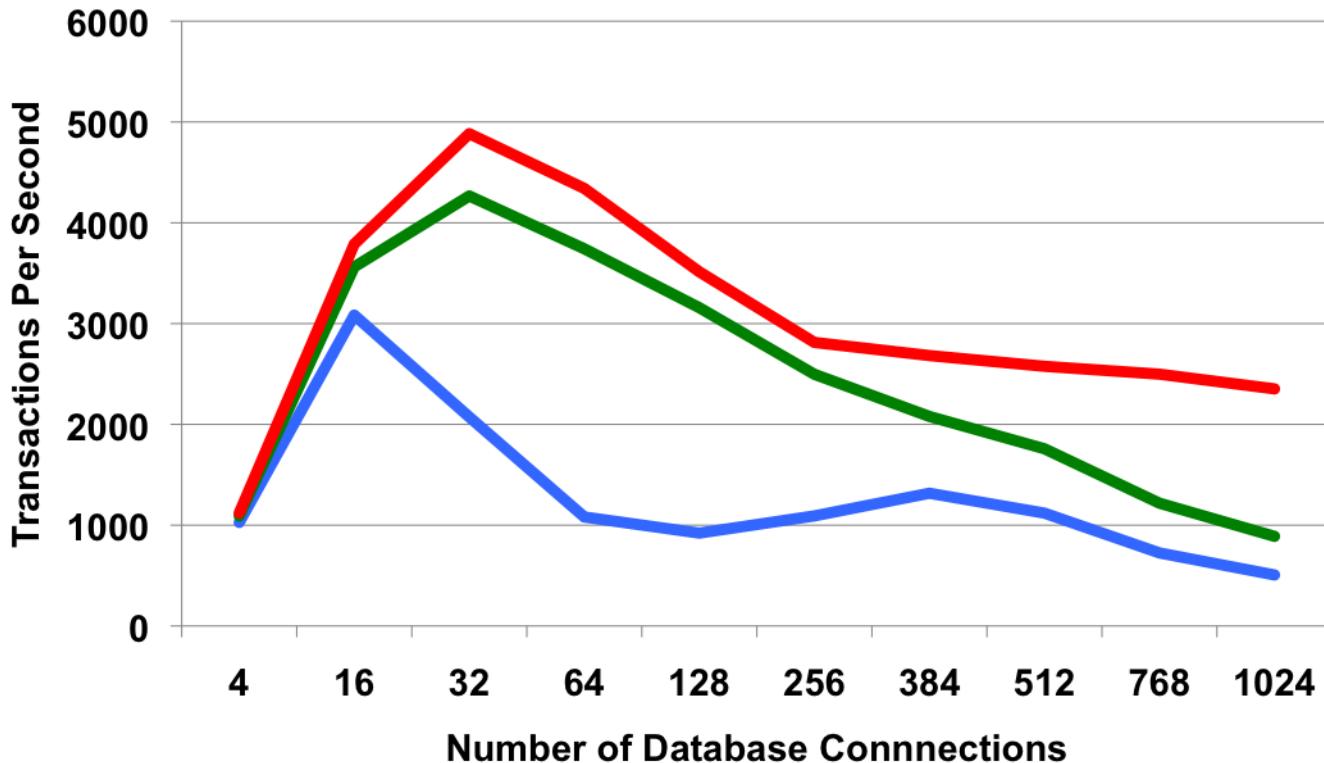
At

- 188% performance gain for MySQL 5.5 over 5.1.40 (InnoDB 1.0)
- 200% performance gain for MySQL 5.5 over 5.1.40 (InnoDB built in)

Intel Xeon X7460 x86_64
4 CPU x 6 Cores/CPU
2.66 GHz, 32GB RAM
Fedora 10

MySQL 5.5/InnoDB 1.1 SysBench Benchmarks

MySQL 5.5 vs. 5.1 Read/Write



MySQL 5.5.4
(InnoDB 1.1)

MySQL 5.1.40
(InnoDB 1.0)

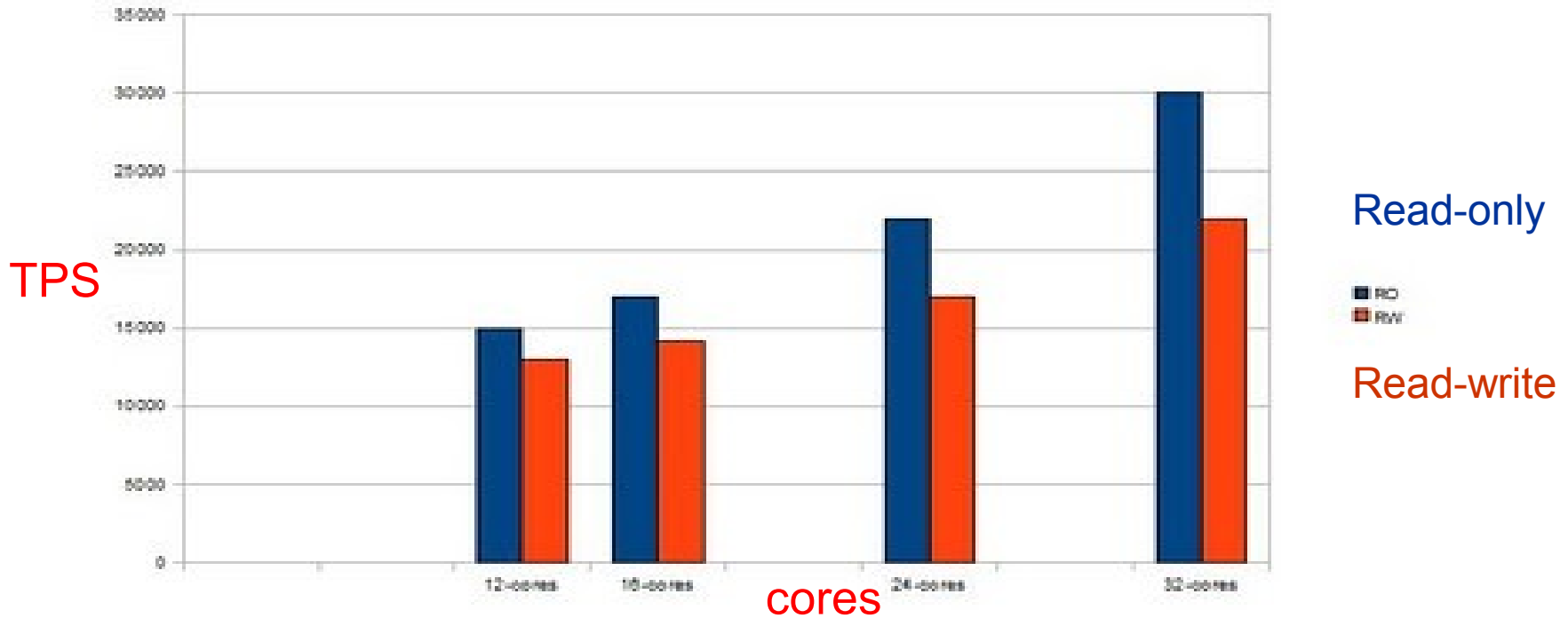
MySQL 5.1.40
(InnoDB built in)

At 1024 Connections:

- 164% performance gain for MySQL 5.5 over 5.1.40 (InnoDB 1.0)
- 364% performance gain for MySQL 5.5 over 5.1.40 (InnoDB built in)

Intel Xeon X7460 x86_64
4 CPU x 6 Cores/CPU
2.66 GHz, 32GB RAM
Fedora 10

MySQL 5.5 / InnoDB 1.1 Scalability dbStress



MySQL 5.5.4 is reaching higher TPS levels than others

MySQL 5.5.4 is better prepared now to scale up to 32 cores

<http://dimitrik.free.fr/blog/archives/2010/04/mysql-performance-554-dbstress.html>

New Features in InnoDB 1.1

- Multiple Buffer Pool Instances
- Improved Recovery Performance
- Extended InnoDB Change Buffering
- Support for Native AIO on Linux
- Multiple Rollback Segments
- Separate Flush List Mutex
- Improved Purge Scheduling
- Improved log_sys mutex
- Performance Schema Support

Multiple Buffer Pool Instances

- Buffer Pool mutex protects many data structures in the Buffer Pool: LRU, Flush List, Free List, Page Hash Table
- It is a hot mutex. For a sysbench tests, it is acquired around 700k/sec, held about 50% of time.
- InnoDB Performance schema also confirms this:

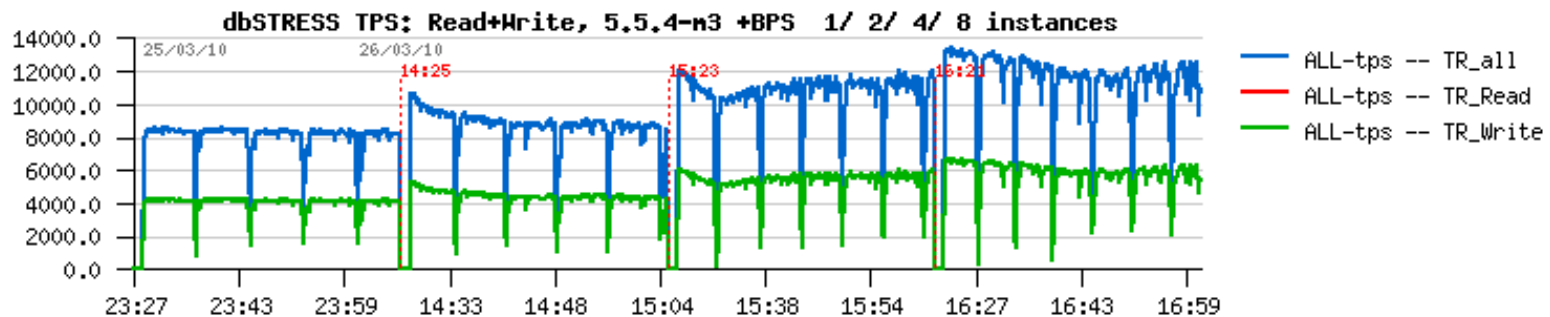
<i>EVENT_NAME</i>	<i>COUNT_STAR</i>	<i>SUM_TIMER_WAIT</i>	<i>AVG_TIMER_WAIT</i>
<i>buf_pool_mutex</i>	<i>1925253</i>	<i>264662026992</i>	<i>137468</i>
<i>buffer_block_mutex</i>	<i>720640</i>	<i>80696897622</i>	<i>111979</i>
<i>kernel_mutex</i>	<i>243870</i>	<i>44872951662</i>	<i>184003</i>
<i>purge_sys_mutex</i>	<i>162085</i>	<i>12238011720</i>	<i>75503</i>
<i>trx_undo_mutex</i>	<i>120000</i>	<i>11437183494</i>	<i>95309</i>
<i>rseg_mutex</i>	<i>102167</i>	<i>14382126000</i>	<i>140770</i>
<i>fil_system_mutex</i>	<i>97826</i>	<i>15281074710</i>	<i>156206</i>

Multiple Buffer Pool Instances

- Solution is to split the buffer pool into multiple buffer pool instances
- Analyzing this split it turned out that we could avoid holding more than one buffer pool mutex instance in all query execution code, only in some code executed rarely was it necessary to hold all mutexes at the same time
- Sysbench RW on 16-cores improves 10%
- Improves Read Only performance as well
- Very large improvement on 32-core/threaded

Multiple Buffer Pool Instances

- `-innodb-buffer-pool-instances=x`



Improved Recovery Performance –The Problem

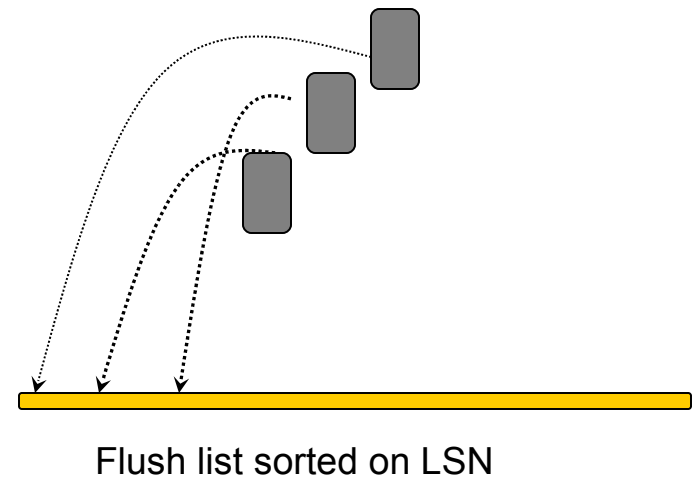
- 3 phases in Recovery



- Scan is slow
 - redo logs need to be read from the disk into a hash table that grows in the buffer pool
 - Need to track the hash table size to avoid exhausting the buffer pool
 - Problem: calculate the hash table size by traversing the list of blocks allocated
 - Approximately $O(n^2)$ number of log block to be scanned

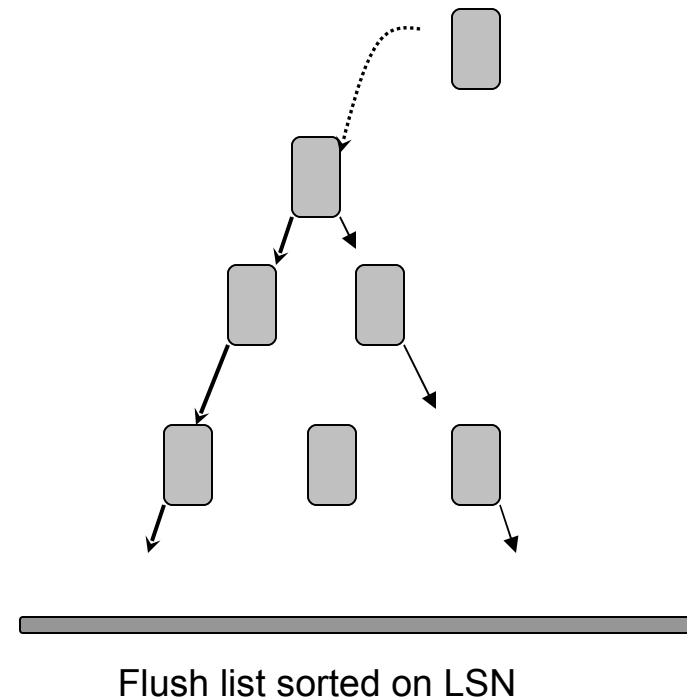
Improved Recovery Performance – The Problem

- Redo Application is slow
 - 'dirty' pages need to be inserted into the “flush_list”
 - List is sorted according to the LSN of the oldest modification to a page
 - List could be large, with large number of dirty pages at the time of crash
 - Insertion into the list was a linear search



Improved Recovery Performance – The Solution

- Resolve the redo scan phase problem
 - Simple and effective solution, caching the hash table size in the header.
Simple and Effective.
- Resolve the redo application phase problem
 - Insert redo log entries into a red-black tree which is sorted on LSN
 - This mechanism is only effective during recovery, and red-black tree would be discarded
 - Run time flush list remain to be list, as LSN are monotonically growing



Improved Recovery Performance – The Result



Recovery in InnoDB 1.0.6

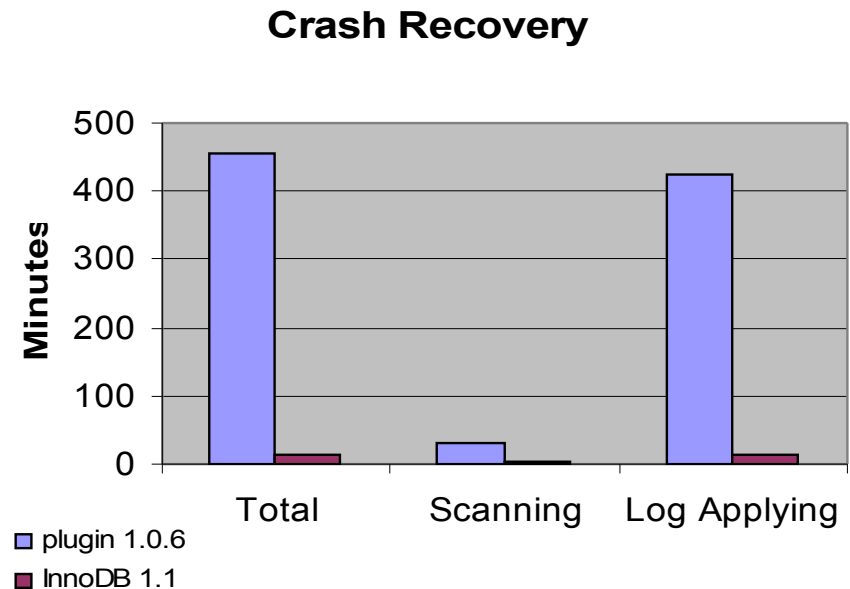


Recovery in InnoDB 1.1

Improved Recovery Performance - BenchMark

- 60m sysbench OLTP Read/Write test
 - innodb-buffer-pool-size=18g
 - innodb-log-file-size=2047m
 - Kill the server after 20 minutes
 - Modified DB pages 1007907
 - Redo bytes: 3050455773

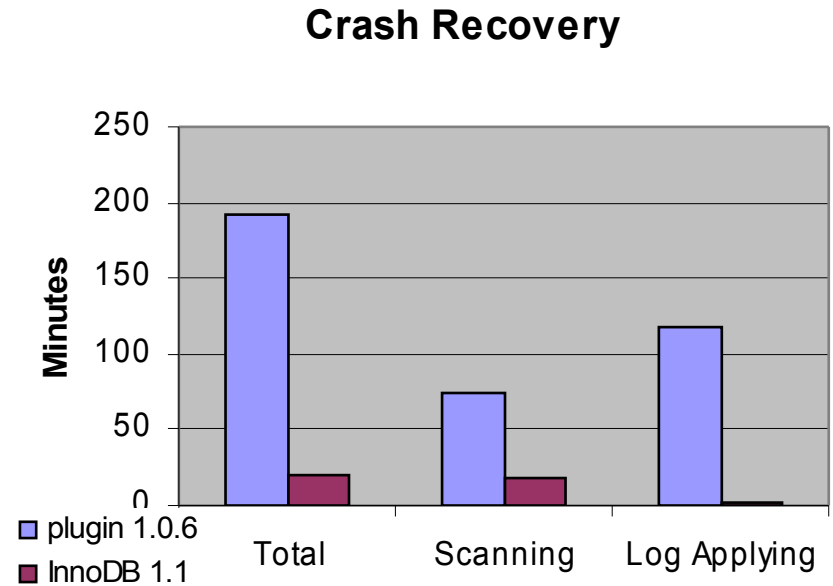
	Total (min)	Scan	Apply
Plugin 1.0.6	456	32	426
InnoDB 1,1	14	2	12
Improvement	32	16	35.5



Improved Recovery Performance – BenchMark

- Crash recovery with dbt2 test
 - 50 warehouses
 - database (9800MB)
 - innodb_log_file_size = 2x1950MB
 - buffer_pool=12GB
 - started test and kill after 5 mins.

	Total (min)	Scanning	Log Applying
Plugin 1.0.6	192	75	117
InnoDB 1,1	20	17.7	2.3
Improvements	9.6	4	51



Extended InnoDB Change Buffering

- Insert Buffering:
 - InnoDB has been buffering insertions to the secondary indexes if the page is not found in the buffer pool. The idea is to avoid extra I/O that is caused by this.
- Delete and Purge Buffering:
 - In InnoDB 1.1 this functionality is extended from insertions to deletions and purging. In any of these cases if the page is not in the buffer pool the operation will be buffered.

Extended InnoDB Change Buffering

- Workloads that should see advantage from this feature:
 - I/O bound and
 - Have secondary indexes and
 - Have considerable amount of DML happening.
- `innodb_change_buffering` (introduced in plugin 1.0) is now extended to accept more values. Default is now 'all'.

Extended InnoDB Change Buffering- The result

- Deletes are a lot faster

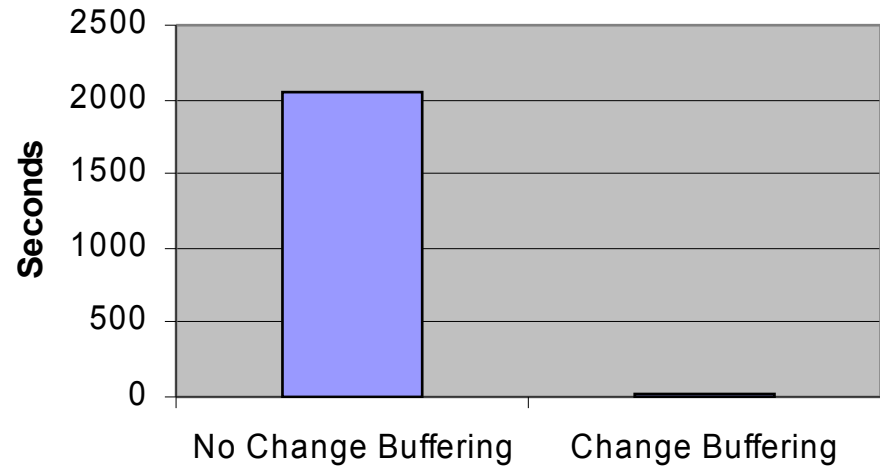


Extended InnoDB Change Buffering – The Benchmark

- Table with 5 Million rows
- Six secondary indexes
- Table size 3G
- Buffer Pool 1G
- Bulk delete of 100,000 rows

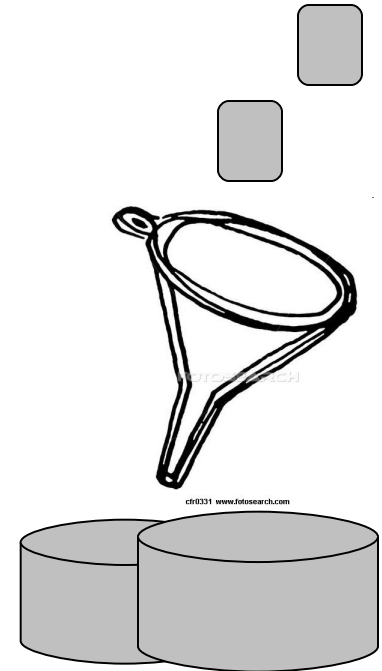
	100K Row Deletion (seconds)	Deletion Rate (rows/second)
Without Change Buffering	2041	50
With Change Buffering	14.54	8000

Deletion with Change Buffering



Support for Native AIO on Linux

- With the exception of Windows InnoDB has used 'simulated AIO' on all other platforms to perform certain IO operations. (InnoDB has always used native AIO on Windows)
- 'Simulated AIO' is still synchronized IO from OS perspective, even though it appears to be asynchronous from the context of a query thread which queues the requests in a queue and return
- Changed to true asynchronous IO on Linux in this release.



Support for Native AIO on Linux

- This enhancement uses a Linux library libaio, which interfaces kernelized native AIO on Linux
- The potential here is to improve scalability of heavily IO bound systems. A system that shows many pending reads/writes in 'show engine innodb status\G' is probably going to gain from this. More IOs can be dispatched to kernel at one time, if the underlying OS can service more requests in parallel then we'll take advantage of that.
- A new switch “innodb_use_native_aio” which enables the user to fall back to simulated aio by setting parameter = 0.

Multiple Rollback Segments

- Traditionally InnoDB has used one rollback segment which has the limitation of 1023 concurrent transactions
- This feature increases the number of rollback segments to 128 each capable of servicing 1K trxs.
- You do NOT need to create a new database to take advantage of this feature. You just need a slow and clean shutdown.
- More importantly this feature also significantly reduces contention on the rollback segment mutex which has started showing up as a very hot mutex with scalability changes in other parts of the code. So this feature is both a scalability and performance feature.

Separate Flush List Mutex

- `flush_list` is a list of 'dirty' blocks in the buffer pool.
- At each `mtr_commit` (note it is not `trx_commit`. A `trx` can have many mini transactions known as `mtr`) if there are any blocks dirtied (i.e.: written to) by the `mtr` then they are inserted in the `flush_list`.
- Previously this happened while holding the buffer pool mutex.
- Now the flush list has its own mutex called `flush_list_mutex`.
- Transparent to the user. No knob to set. Out of the box better parallelism.

Improved Purge Scheduling

- Purge operation
 - In the InnoDB multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement.
 - Only when InnoDB can discard the update undo log record written for the deletion can it also physically remove the corresponding row and its index records from the database. This removal operation is called a purge.
 - This was being performed by the background master thread.

Improved Purge Scheduling – The Problem

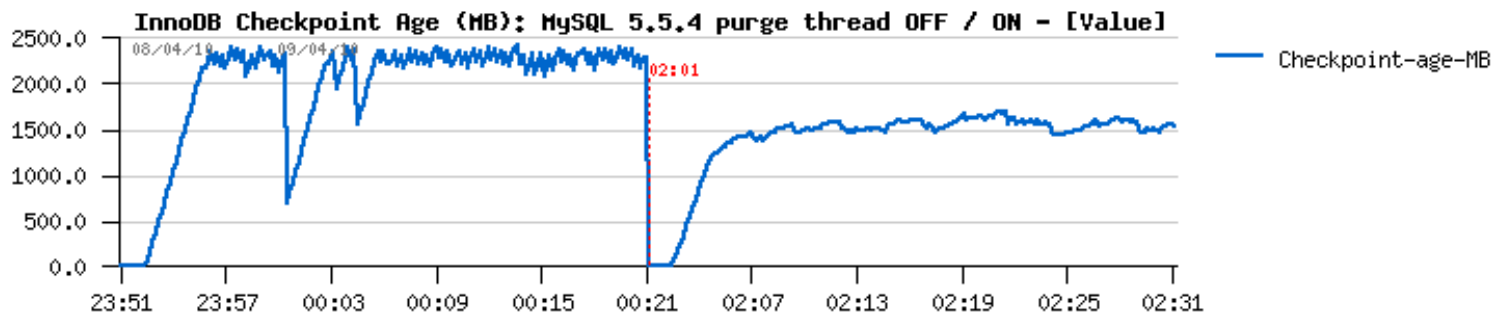
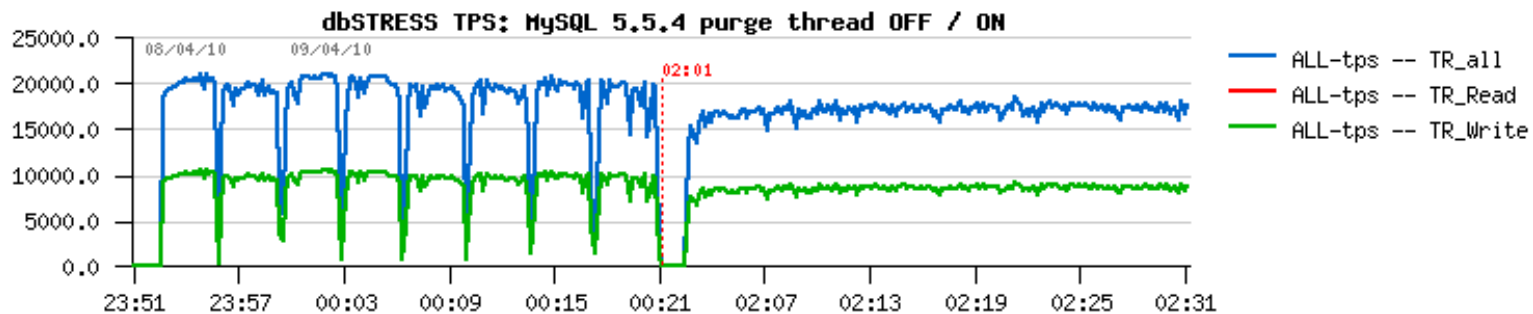
- In high transaction scenario, master thread can be blocked to execute only purge activities for a long time
- This leads to master thread not properly flushing dirty pages, not doing checkpoints regularly as it should
- It is possible that the purge thread could lag behind the deletion operation, and table becomes inflated.
- The idea is to allocate more resources to the purge thread if needed.

Improved Purge Scheduling – The Solution

- The solution is to use a dedicated thread for the sole purpose of purging.
- In theory this should not only make purging more effective but it also frees up master thread to do other important stuff like flushing of dirty pages etc.
- `innodb_purge_threads`, set to 0 to switch to traditional mode.
- `innodb_purge_batch_size` allows the user to specify the size of a purge batch, its value can range from 1 to 5000, with default of 20,


Improved Purge Scheduling – The Benchmark

dbSTRESS: Read+Write & Purge Thread

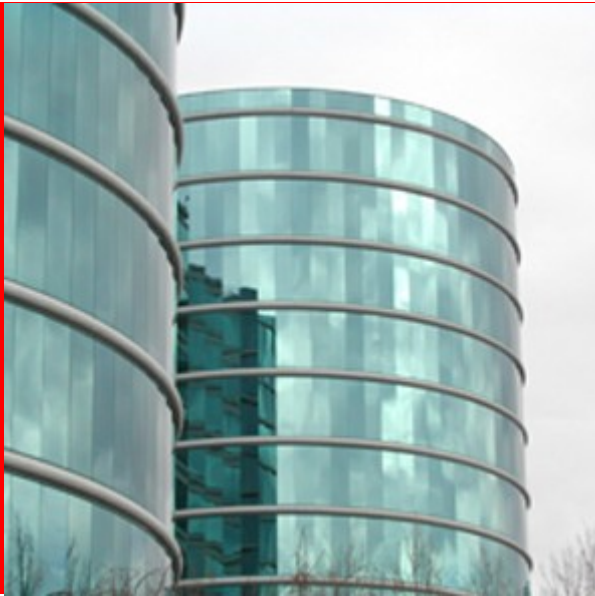


References and More Information

- Session “InnoDB: Status, Architecture, and New Features”, Young (MySQL Database Group), Sun (Oracle Corp.)
- Session “What's New in MySQL 5.5? Performance/Scale Unleashed!” - Rob Young (MySQL Database Group), Mikael Ronstrom (MySQL) 10:50am Tuesday, 04/13/2010
- Blog: <http://blogs.innodb.com/>



Q&A
QUESTIONS
ANSWERS



Thanks for attending!

*InnoDB Plugin: Performance Features and
Benchmarks*

MySQL Conference and Expo, 2010

10:50 am, April 15, 2010