DTCC2013

InnoDB架构分析以及TNT引擎的优势

网易杭研院：　何登成

微博：　何_登成

博客：　深入MySQL内核

邮箱：　he.dengcheng@gmail.com

Outline

- Why MySQL?
- Why InnoDB?
- Why TNT?
- TNT vs InnoDB: Tests

Why MySQL

- **MySQL能做什么？**

> @AlibabaDBA：这个双十一狂欢节，淘宝核心系统全部去IOE，使用PC和MySQL，众集群表现稳定，其中某核心集群总共执行293亿次SQL/天，集群总QPS达86万/秒，集群TPS11万/秒，单机QPS高达6.5万/秒，分布式MySQL架构再一次经受住了考验。

> @hello丁原：Alipay MySQL集群 双11当天支撑了156亿次SQL，和oracle集群还是有很大差距，希望明年东方能刮过西方。。。

> @twitter: ...in particular, on the number of Tweets per second (TPS). Last night, Twitter averaged about 9,965 TPS from 8:11pm to 9:11pm PT, with a one-second peak of 15,107 TPS at 8:20pm PT and a one-minute peak of 874,560 TPM.

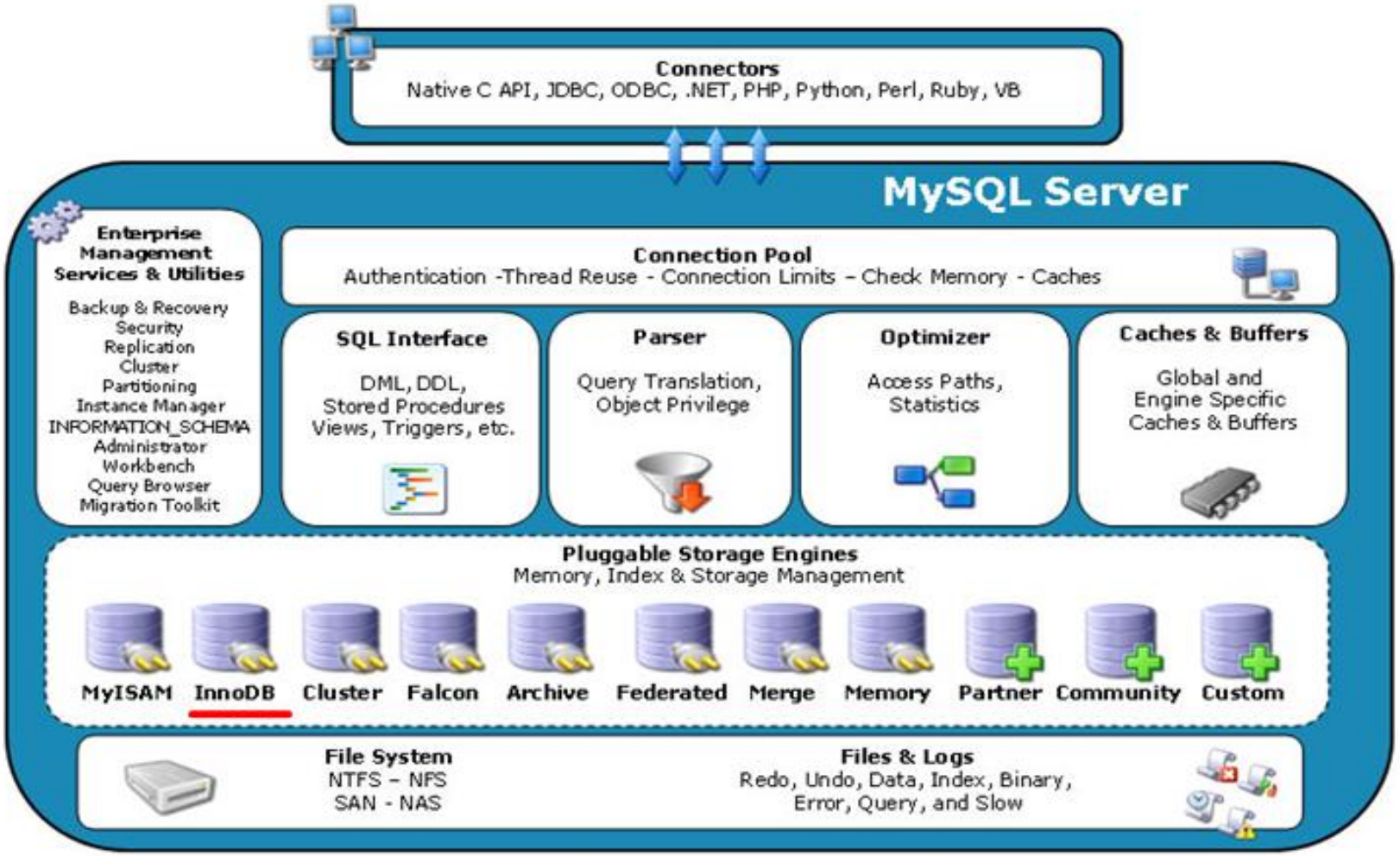
Why MySQL

- 使用最广的开源数据库
  - 评分标准
- 成熟的社区与研发
  - Oracle’s MySQL
  - Percona
  - MariaDB
  - Planet MySQL

April 2013

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1560.59	+27.20
2.	↑ 3.	MySQL	Relational DBMS	1342.45	+47.24
3.	↓ 2.	Microsoft SQL Server	Relational DBMS	1278.15	-40.21
4.	4.	PostgreSQL	Relational DBMS	174.09	-3.07
5.	5.	Microsoft Access	Relational DBMS	161.40	-8.77
6.	6.	DB2	Relational DBMS	155.02	-4.31
7.	7.	MongoDB	Document store	129.75	+5.52
8.	↑ 9.	SQLite	Relational DBMS	88.94	+5.68
9.	↓ 8.	Sybase	Relational DBMS	80.16	-5.25
10.	10.	Solr	Search engine	46.15	+2.99
11.		Teradata	Relational DBMS	44.93	
12.	11.	Cassandra	Wide column store	38.57	+2.21
13.	12.	Redis	Key-value store	35.58	+3.15
14.	13.	Memcached	Key-value store	24.80	-0.17
15.	14.	Informix	Relational DBMS	24.00	+0.10
16.	15.	HBase	Wide column store	21.84	+1.40
17.	16.	CouchDB	Document store	18.72	+0.42

MySQL Architecture



Why InnoDB

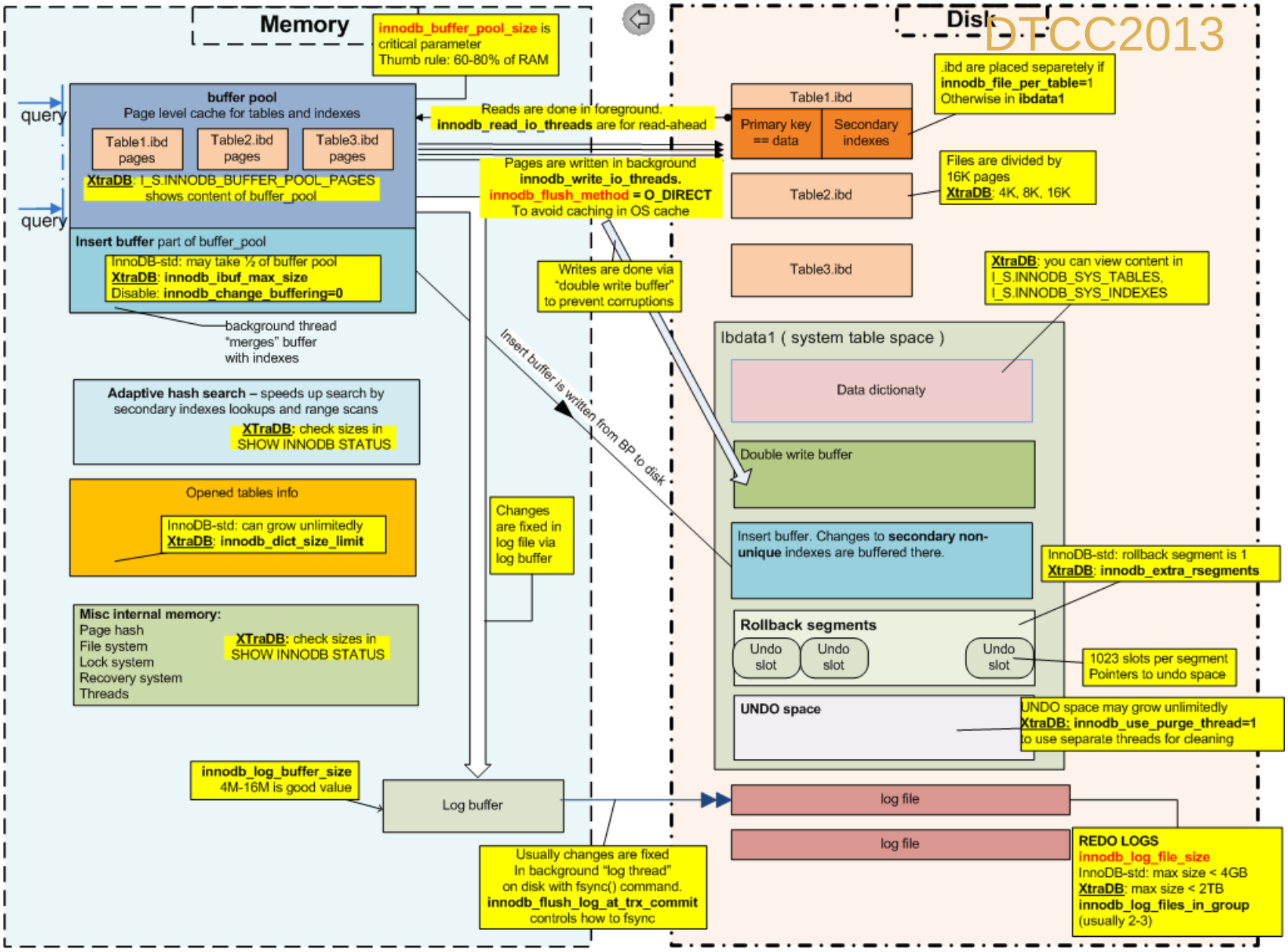
- **事务引擎**
  - 在目前MySQL的所有官方开源引擎中，InnoDB是唯一的一个支持事务的通用性引擎；
- **使用广泛**
  - 淘宝、支付宝、网易、Facebook、Twitter...，均使用InnoDB存储引擎；
- **研发**
  - Oracle’s InnoDB团队每年都会推出新版本；
  - Percona’s XtraDB发展较快；
  - 各大公司提供改进Patch(Google/Facebook/Twitter/Alibaba/NetEase...)

InnoDB Architecture

- General Architecture
  - OLTP引擎
    - 索引组织表(聚簇索引包含完整数据)；
    - 行存储；行级锁；
  - MVCC(多版本)
    - 核心架构与Oracle类似；
    - 历史版本存储于Rollback Segment(回滚段)；
  - 文件组织
    - 数据存储于Tablespace；
    - 日志存储于Log File；
  - 内存管理
    - Buffer Pool；

InnoDB Architecture(续)

- **架构优化**
  - Insert Buffer
    - Cache二级索引的更新，提高更新效率；
  - Adpative Hashing
    - 提高二级索引查找效率；
  - Async I/O
    - 提高I/O效率；
  - Fuzzy Checkpoint
    - 降低检查点操作，对于系统性能影响；
    - 可提高崩溃恢复的效率；
  - Double Write
    - 解决磁盘Half-Writtern问题；


Memory
Disk
DTCC2013
innodb_buffer_pool_size is critical parameter
Thumb rule: 60-80% of RAM
query
query
buffer pool
Page level cache for tables and indexes
Table1.ibd pages
Table2.ibd pages
Table3.ibd pages
XtraDB: I_S.INNODB_BUFFER_POOL_PAGES shows content of buffer_pool
Insert buffer part of buffer_pool
InnoDB-std: may take ½ of buffer pool
XtraDB: innodb_ibuf_max_size
Disable: innodb_change_buffering=0
background thread "merges" buffer with indexes
Adaptive hash search – speeds up search by secondary indexes lookups and range scans
XTraDB: check sizes in SHOW INNODB STATUS
Opened tables info
InnoDB-std: can grow unlimitedly
XtraDB: innodb_dict_size_limit
Misc internal memory:
Page hash
File system
Lock system
Recovery system
Threads
XTraDB: check sizes in SHOW INNODB STATUS
Reads are done in foreground.
innodb_read_io_threads are for read-ahead
Pages are written in background
innodb_write_io_threads.
innodb_flush_method = O_DIRECT
To avoid caching in OS cache
Writes are done via "double write buffer" to prevent corruptions
Insert buffer is written from BP to disk
Changes are fixed in log file via log buffer
innodb_log_buffer_size
4M-16M is good value
Log buffer
Usually changes are fixed In background "log thread" on disk with fsync() command.
innodb_flush_log_at_trx_commit controls how to fsync
Table1.ibd
Primary key == data
Secondary indexes
.ibd are placed separetely if innodb_file_per_table=1
Otherwise in ibdata1
Table2.ibd
Files are divided by 16K pages
XtraDB: 4K, 8K, 16K
Table3.ibd
XtraDB: you can view content in I_S.INNODB_SYS_TABLES, I_S.INNODB_SYS_INDEXES
Ibdata1 (system table space)
Data dictionaty
Double write buffer
Insert buffer. Changes to secondary non-unique indexes are buffered there.
InnoDB-std: rollback segment is 1
XtraDB: innodb_extra_rsegments
Rollback segments
Undo slot
Undo slot
Undo slot
1023 slots per segment
Pointers to undo space
UNDO space
UNDO space may grow unlimitedly
XtraDB: innodb_use_purge_thread=1
to use separate threads for cleaning
log file
log file
REDO LOGS
innodb_log_file_size
InnoDB-std: max size < 4GB
XtraDB: max size < 2TB
innodb_log_files_in_group
(usually 2-3)

What's Good in InnoDB

- **按照个人理解，排名分先后**
  - MVCC
    - InnoDB采用了于Oracle类似的多版本实现策略；
    - 此策略是InnoDB能够成功的基础；
  - Simple First
    - InnoDB架构实现简单，包括：并发控制；B+-Tree；大对象...
    - 简单先行，是InnoDB成功的第二个基础；
  - Cluster Index
    - 索引组织表，是InnoDB在互联网简单OLTP应用中高性能的基础；
  - ...

What’s Bad in InnoDB

- **研发进度慢**
  - 用户需求，往往要几年后才能加上；
- **新版本稳定性**
  - 新版本发布之后，Bug较多(新版本功能越多，稳定周期越长)；
  - 目前广泛使用的，仍旧是MySQL 5.1，MySQL 5.5逐步推广...
- **InnoDB目前在Oracle手中**
  - 考虑到Oracle自己的产品，以及其针对开源的策略，有使用风险；
- **缺乏有效的度量数据**
  - 运维过程中无法定位问题所在；

What’s Bad in InnoDB(续)

- Simple First后遗症较多
  - 但节点可扩展性较差：
  - 高可用支持较差：
  - 不支持行级缓存：
  - 压缩实现较差：
  - 索引、大对象实现效率较低；
  - ...

  单机多实例？
  运维困难；
  需要与Memcached配合使用；
  压缩功能使用较少；

- Cluster Index
  - 索引查询(辅助索引)，性能较差；

- MVCC
  - 外存空间消耗较大；
  - 旧版本回收对性能有较大影响；

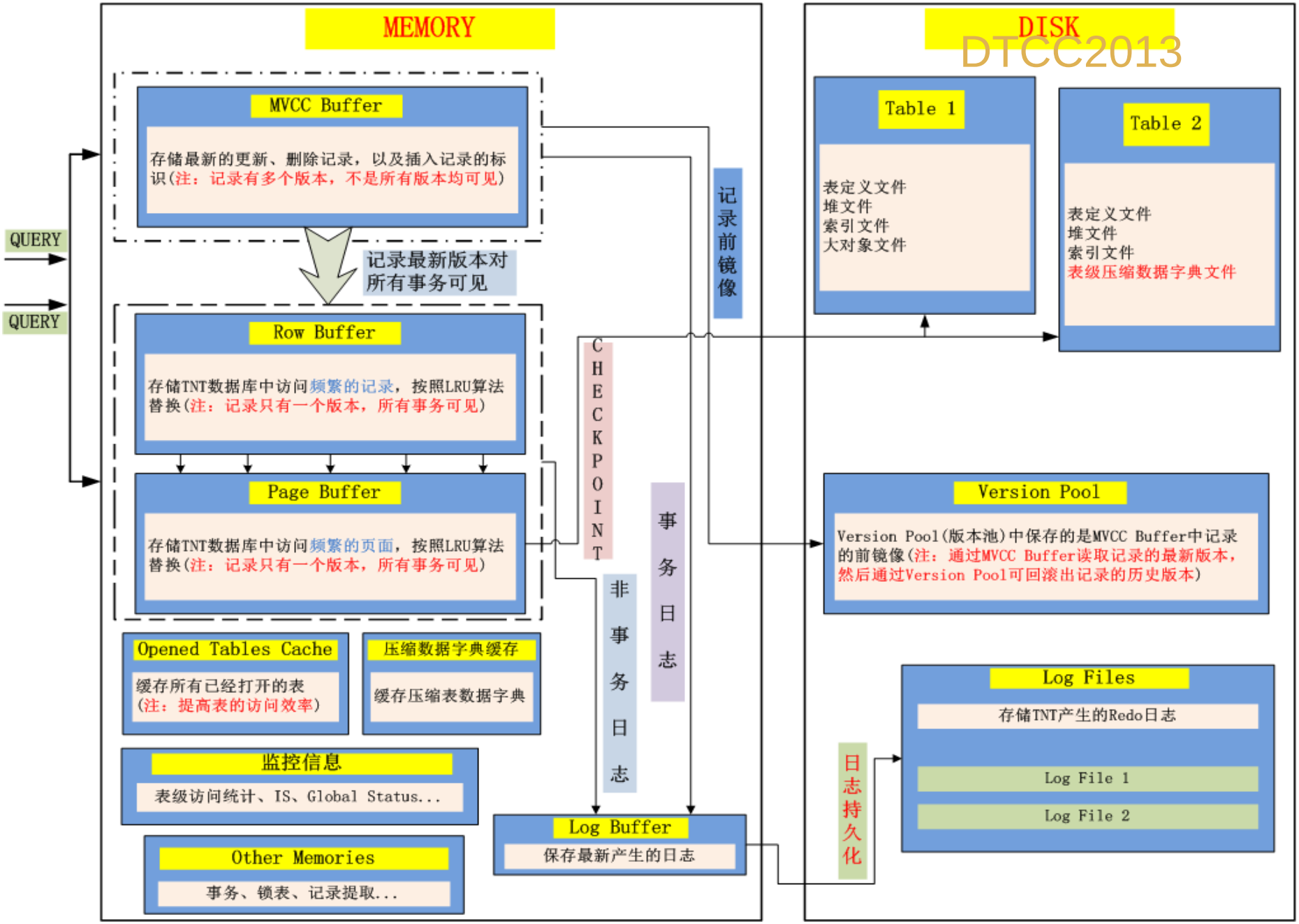
- ...

Why TNT

- What’s TNT?
  - 由网易杭研研发的具有自主知识产权的，基于MySQL的存储引擎；
  - TNT在网易研发的前一版本存储引擎NTSE的基础上发展而来；
  - NTSE
    - Non-Transaction Storage Engine
    - 非事务存储引擎；
  - TNT
    - Transaction Non-Transaction (Storage Engine)
    - 事务与非事务统一支持的存储引擎；

TNT Architecture

- General Architecture
  - OLTP引擎
    - 堆组织表；
    - 行存储；行级锁；
  - 持久化依赖于NTSE
    - 充分吸收NTSE的优势；
    - 减少重复轮子；
  - MVCC
    - 多版本支持，读写不冲突；
    - 历史版本存储于**版本池**(类似于InnoDB的回滚段)
    - 行级多版本，与InnoDB一致；
    - 内存多版本，外存单一版本；

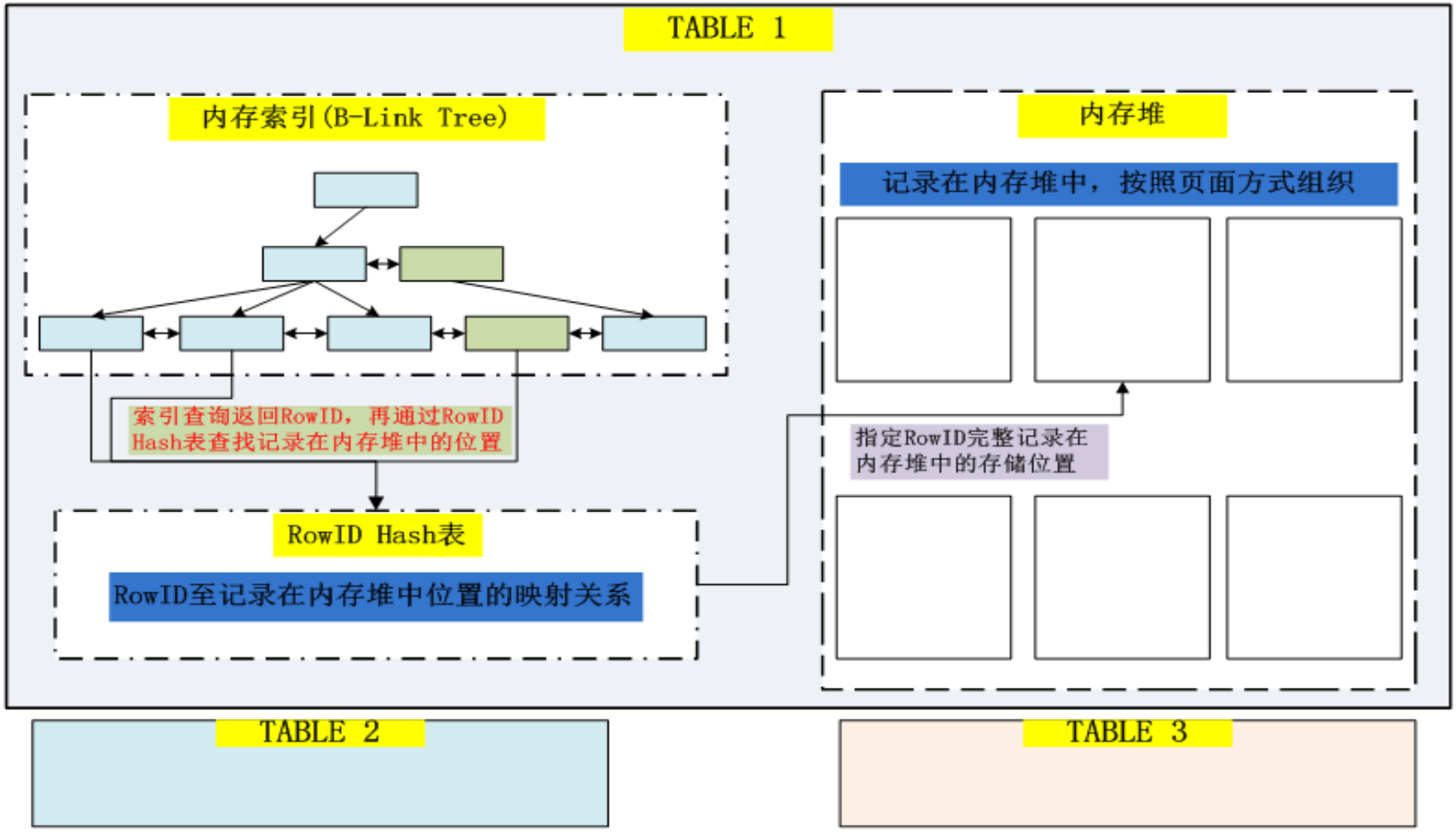

MEMORY
DISK
DTCC2013
MVCC Buffer
存储最新的更新、删除记录，以及插入记录的标识(注：记录有多个版本，不是所有版本均可见)
记录最新版本对所有事务可见
记录前镜像
QUERY
QUERY
Row Buffer
存储TNT数据库中访问频繁的记录，按照LRU算法替换(注：记录只有一个版本，所有事务可见)
Page Buffer
存储TNT数据库中访问频繁的页面，按照LRU算法替换(注：记录只有一个版本，所有事务可见)
CHECKPOINT
非事务日志
事务日志
Table 1
表定义文件
堆文件
索引文件
大对象文件
Table 2
表定义文件
堆文件
索引文件
表级压缩数据字典文件
Version Pool
Version Pool(版本池)中保存的是MVCC Buffer中记录的前镜像(注：通过MVCC Buffer读取记录的最新版本，然后通过Version Pool可回滚出记录的历史版本)
Opened Tables Cache
缓存所有已经打开的表(注：提高表的访问效率)
压缩数据字典缓存
缓存压缩表数据字典
Log Files
存储TNT产生的Redo日志
Log File 1
Log File 2
日志持久化
监控信息
表级访问统计、IS、Global Status...
Log Buffer
保存最新产生的日志
Other Memories
事务、锁表、记录提取...

TNT的架构创新

- MVCC Cache
  - 内存多版本，外存单一版本
  - 减少了多版本存储开销；减少了多版本回收开销；
- Row Cache
  - 记录级缓存，相对于页面级缓存，提高了内存利用率；
  - 内嵌记录级缓存，省去了搭建Memcached的开销；
  - 内嵌记录级缓存，缓存能够与数据库保持一致性；
- 基于数据字典的压缩
  - 压缩/解压粒度更细，记录级/列级压缩解压粒度；压缩/解压效率更高；
  - 表级数据字典采集，每个表拥有自己的数据字典，压缩率更高；
  - 数据字典可在线重采集，表可以在线重压缩；

MVCC Buffer

1. 更新、删除记录，对所有事务可见之前，首先保存在MVCC Buffer中；
2. 插入记录不进入MVCC Buffer，但在所有事务可见之前，需要将插入标识(RowID)保存在MVCC Buffer中；



MVCC Buffer的维护

随着更新、删除记录的增加，MVCC Buffer会不断扩大，消耗大量内存；针对此问题，TNT引擎后台有定时任务，扫描MVCC Buffer中的每一张表，将对所有事务均可见的记录刷出，写回到Row Buffer与Page Buffer，释放MVCC Buffer占用空间(Row Buffer/Page Buffer，将由Checkpoint线程负责脏数据写出磁盘，Scavenger线程负责LRU替换)

Row Cache

Row Cache，属于TNT的行级缓存，相对于Page Cache，Row Cache只缓存访问频率较高的记录，而非记录所属的整个页面，提高了内存利用的效率。Row Cache，在TNT中被称之为MMS(Main-Memory Storage)，顾名思义，MMS是支持持久化的，MMS中的记录更新、删除操作，需要记录Redo日志。

Table 1

每一张开启Row Cache的用户表，在Row Cache中都对应这一个MMS Table结构。MMS Table按照页面组织记录，并且按照记录的实际存储大小，划分为不同的Page Class，类似于Linux中的Slab分配器。

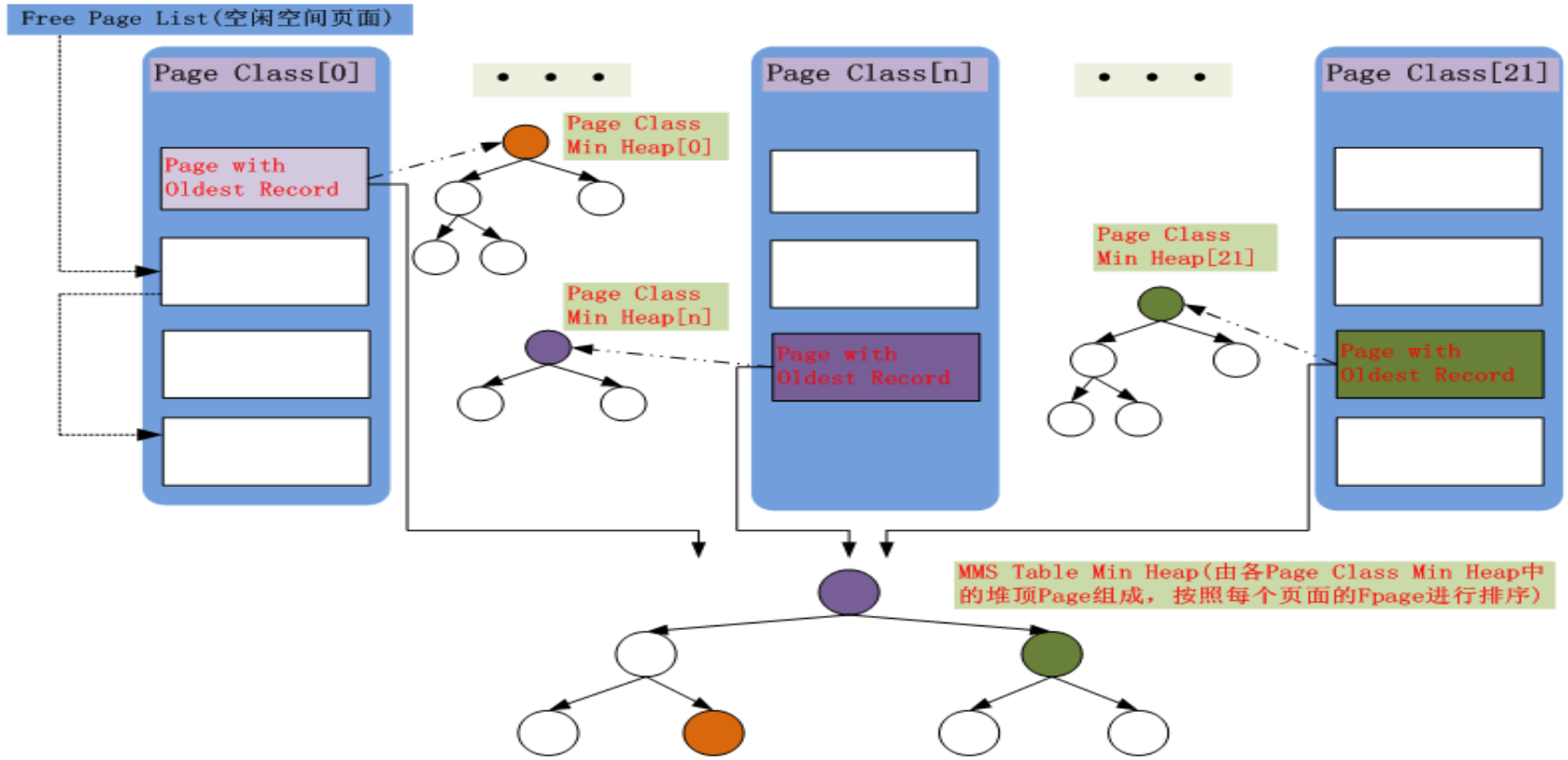


Table 2

Table 3

基于全局数据字典的压缩

- **记录压缩**
  - TNT的采用的是全表基于数据字典的记录级压缩，压缩算法参考LZ77/78，Deflate等；
- **数据字典**
  - 针对每个压缩表，首先采集数据字典。所谓数据字典，就是记录中出现频率较高的字符串；
- **测试效果**

项目	InnoDB	MSSQL	DB2	ORACLE	NTSE
压缩前(MB)	15625.0	11913.0	12179.0	11550.0	13111.0
压缩后(MB)	7841.0	7275.0	4818.0	8681.0	3700
压缩比	50%	61%	40%	75%	28.2%

  - 注1：TPC-H 10GB数据量压缩比；
  - 注2：数据压缩，TNT完全继承自NTSE；
  - 注3：其他数据库测试结果，参考自 http://database.chinaunix.net/a2011/0629/1211/000001211050_1.shtml

TNT引擎的其他优势

- **高可用设计**
  - 创建/删除索引、Optimize等操作，在线完成，过程无需拷贝表，最终短暂锁表；
  - 新增列操作，瞬时完成，只修改表定义，不修改数据；
- **监控与诊断**
  - 提供各方面的监控信息，方便运维管理；
- **高并发支持**
  - 内部并发瓶颈较少
- **研发进度保障**
  - TNT自主研发，能更快的响应Bug与用户需求；
- **完备的性能测试框架**
  - TNT有一整套完备的性能测试框架，用于定位系统的性能瓶颈等；
- **事务与非事务切换功能**
  - TNT引擎，可以同时支持事务表与非事务表，由用户选择；

TNT的监控与诊断

- 丰富的监控信息
  - 对象级监控信息
    - 表、堆、
    - 索引、MMS
    - ...
  - 并发冲突监控
    - Mutex、RW Lock
    - Intention Lock
  - 其他监控
    - 连接、内存等

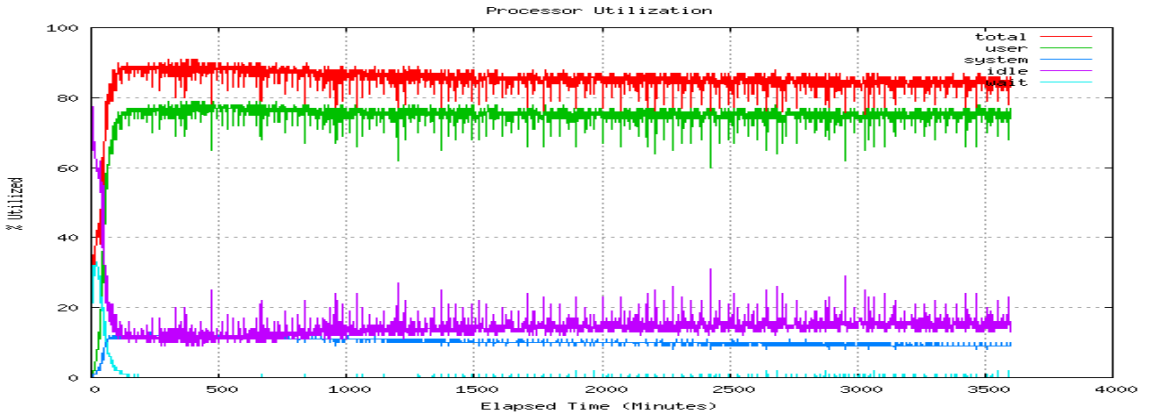
表名	说明及常用功能
TNT_NTSE_BUF_DISTRIBUTION	TNT各数据库对象占用页面缓存情况，可用于确定占用TNT内存较多的表或数据库对象
TNT_NTSE_DBOBJ_STATS	TNT各表各类数据库对象操作统计
TNT_NTSE_HEAP_STATS	各TNT表堆存储组件的操作统计
TNT_NTSE_INDEX_STATS	各TNT表索引存储组件的操作统计
TNT_NTSE_LOB_STATS	各TNT表大对象存储组件的操作统计
TNT_NTSE_MMS_STATS	各TNT表Row Cache存储组件的操作统计
TNT_NTSE_MUTEX_STATS	TNT内部各类互斥锁的统计信息
TNT_NTSE_RWLOCK_STATS	TNT内部各类读写锁的统计信息
TNT_NTSE_TABLE_STATS	各TNT表操作统计信息
TNT_TNT_TRANSACTION_SYS_STATS	TNT内部事务情况的统计信息
TNT_TNT_INDEX_STATS	TNT的索引统计信息
TNT_TNT_MHEAP_STATS	MVCC Buffer中堆的统计信息
TNT_TNT_MEMORY_INDEX_STATS	MVCC Buffer中索引的统计信息

TNT的性能测试框架

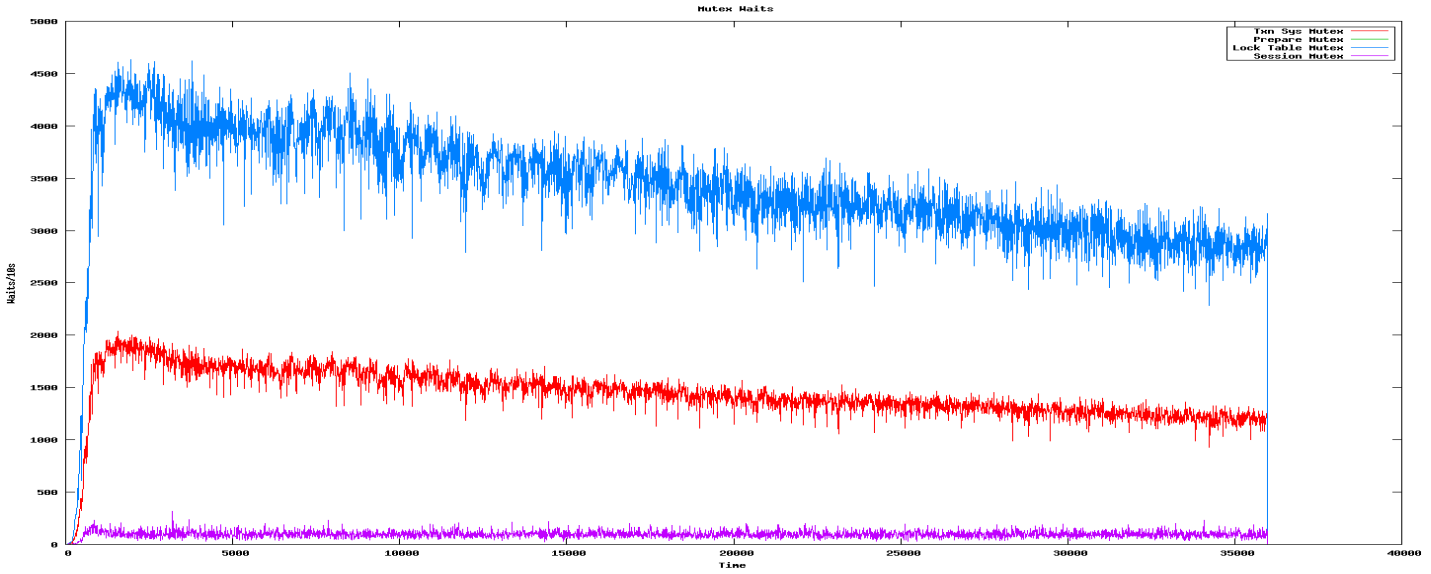
- **性能测试框架**
  - 实时收集各类统计信息并绘制为可视图，方便问题的跟踪与定位
  - **操作系统层面信息收集**
    - CPU利用率走势；
    - I/O利用率走势；
    - I/O合并情况、I/O队列大小等；
    - ...
  - **TNT引擎内部信息收集**
    - 采集TNT系统内部提供的监控信息；
    - 并发冲突信息；
    - 脏页情况
    - ...

TNT性能测试框架(举例)

- 操作系统层面信息
  - CPU利用率



- TNT引擎内部信息
  - Mutex等待次数

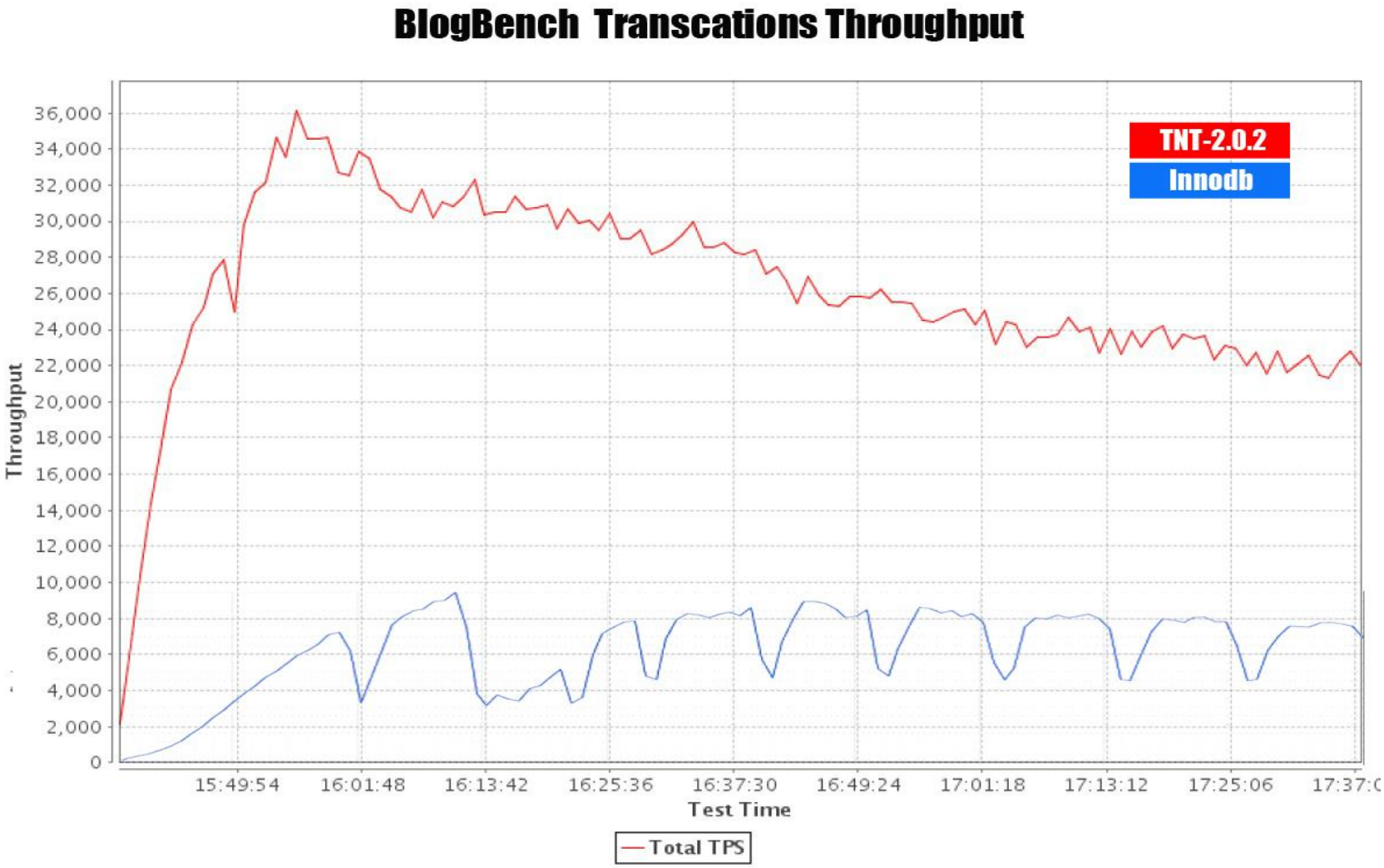


TNT目前的不足之处

- TNT引擎目前的不足之处
  - **新生产品，未经过线上应用挑战**
    - 引入TCPCopy工具，模拟线上应用测试；
  - **TNT对于长事务的支持较差**
    - 长事务，会导致内存多版本无法回收，内存使用量增加；
    - 适用于短事务的OLTP应用；
  - **TNT在很多细节上的优化，还不够**
    - TNT的很多模块，缺乏在于细节上的深入优化(这也跟我们产品初创阶段有关)；
    - TNT 2.0.2-RC版本代码量
      - 11万行左右；
    - MySQL 5.6.10-GA版InnoDB代码量
      - 16万行左右；

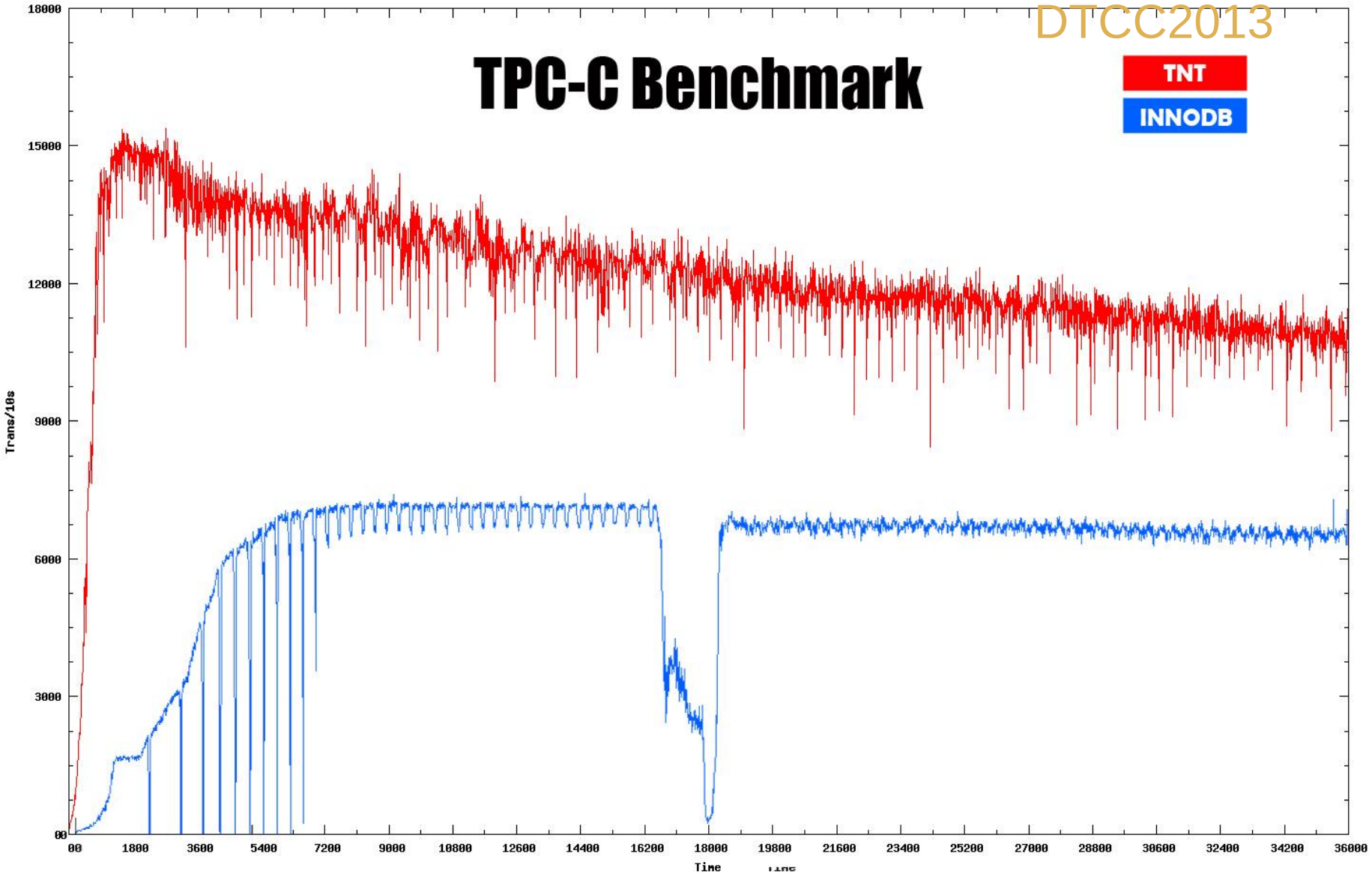
TNT vs InnoDB: Tests

- TNT与InnoDB的对比测试
  - **测试环境**
    - Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz 32-Core
    - 8块SAS盘，RAID-0 128G Memory
    - MySQL 5.1.49(TNT基于此版本开发)
  - **Blogbench测试**
    - Blogbench测试工具，为自主研发的测试工具，用于模拟网易博客类应用；
    - 测试准备
      - 1000万记录；10小时运行；
      - 内存使用为InnoDB数据量的30%；
  - **TPCC测试**
    - TPCC，测试数据库OLTP性能的基准工具；
    - 测试准备
      - TPCC客户端：Percona开源tpcc-mysql工具；
      - 100-warehouses；32-threads；10小时；25G Memory；


BlogBench Transcations Throughput
TNT-2.0.2
Innodb
Throughput
36,000
34,000
32,000
30,000
28,000
26,000
24,000
22,000
20,000
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
15:49:54
16:01:48
16:13:42
16:25:36
16:37:30
16:49:24
17:01:18
17:13:12
17:25:06
Test Time
Total TPS

TNT vs InnoDB：Blogbench

- 结果分析
  - TPS
    - TNT：　　Total(189601210)　　Avg(26331)
    - InnoDB：　　Total(48353499)　　Avg(6715)
    - TNT的avg TPS为InnoDB的4倍左右；
  - Response Time
    - TNT：　　Avg(1 ms)　　90%(1 ms)
    - InnoDB：　　Avg(4 ms)　　90%(14 ms)
    - TNT的响应时间要优于InnoDB；
  - InnoDB
    - 8MB Log Buffer；1 GB Log File；关闭Binlog；
    - innodb_flush_log_at_trx_commit = 0；


DTCC2013
TPC-C Benchmark
TNT
INNODB
Trans/10s
Time
18000
15000
12000
9000
6000
3000
00
00
1800
3600
5400
7200
9000
10800
12600
14400
16200
18000
19800
21600
23400
25200
27000
28800
30600
32400
34200
36000

TNT vs InnoDB: TPCC

- 结果分析
  - TNT
    - TPMC在InnoDB的2倍左右（72588.891 vs 36160.988）；
    - 每秒的TPS已经较为稳定，但是与InnoDB还是有所差距；
  - InnoDB
    - 未详细定位奇异点产生的原因；
    - InnoDB的调优集中于参数层面，未修改源码；
    - 参数设置：
      - innodb_flush_log_at_trx_commit = 0;
      - binlog_ignore_db

总结与展望

- MySQL
  - MySQL数据库使用广泛；
- InnoDB
  - InnoDB存储引擎有其架构优势，也有不足之处；
- TNT引擎
  - 自主研发，相对于InnoDB有功能优势；
    - In-Memory MVCC; Row Cache; Compression; ...
  - 在测试中，表现出比InnoDB更好的性能；
  - 2013-06-30，将发布第一个GA版本；
    - 功能增强 vs 性能优化

参考资料

- *Peter Zaitsev*, InnoDB Architecture and Internals.
- *Peter Zaitsev*, InnoDB Architecture and Performance Optimization.
- *Heikki Tuuri*, InnoDB: Architecture, Status, and New Featuers.
- *DB-Engines*, DB-Engines Ranking
- *DB-Engines*, Method of calculating the scores of DB-Engines Ranking
- *ChinaUnix*, 数据库横评 5款主流行式数据库评测总结

Questions?

谢谢大家！