

程序猿都应该知道的 MySQL秘籍

叶金荣 <http://imysql.com>

公众号: imysql_wx

2016.5.14

关于我



- 叶金荣
- Oracle MySQL ACE
- 国内最早的MySQL推广者
- From 2006, <http://imysql.com>
- 从事MySQL相关工作10余年
- 擅长MySQL性能优化
- 现专注MySQL DBA人才培养
- 公众号: [imysql_wx](#) (MySQL中文网)



Agenda

- MySQL优化秘籍
 - InnoDB or MyISAM?
 - InnoDB表应该怎么玩
 - 一些优化参考
 - 非典型DBA怎么玩好MySQL
- 值得期待的5.7新特性

先从一个 **成见** 开始

还死守MyISAM? out了

- 读多写少的场景下，真的MyISAM就合适吗
- 源起：[InnoDB还是MyISAM 再谈MySQL存储引擎的选择](#)
- TA观点
 - MyISAM的读性能是比InnoDB强不少
 - MyISAM索引和数据分开，且索引有压缩，内存使用率相对更高
 - MyISAM可以直接覆盖MYD/MYI文件恢复数据，相对更快
 - count(*)和order by效率低，且count(*)会锁表
 - InnoDB的insert和update太快了，导致从库更不上
 - MyISAM有merge类型，可以快速count(*)

InnoDB的好处

· 我的观点

- 大多数业务中，95%以上的场景，都可以采用InnoDB引擎
- InnoDB可以把数据、索引、有修改的数据放在内存buffer中，而且有自适应哈希索引、change buffer merge等等，事实上更高效
- InnoDB也可直接导出表空间文件后，在目标服务器上导入。当然了，不能直接在线拷贝，需要稍加处理，但并不麻烦
- 没有索引时，count(*)和order by肯定效率低，不区分引擎。InnoDB是事务表，所以全表（无WHERE条件时）count(*)确实会慢一些，但并不会锁表
- InnoDB的insert和update太快了，导致从库跟不上 —— 妹的，这个竟成了缺点？
- MyISAM有merge类型，可以快速count(*) —— 可以采用表分区特性，如果仅仅只是为了COUNT(*)，则可以采用redis等第三方来计数

赶紧转投InnoDB怀抱

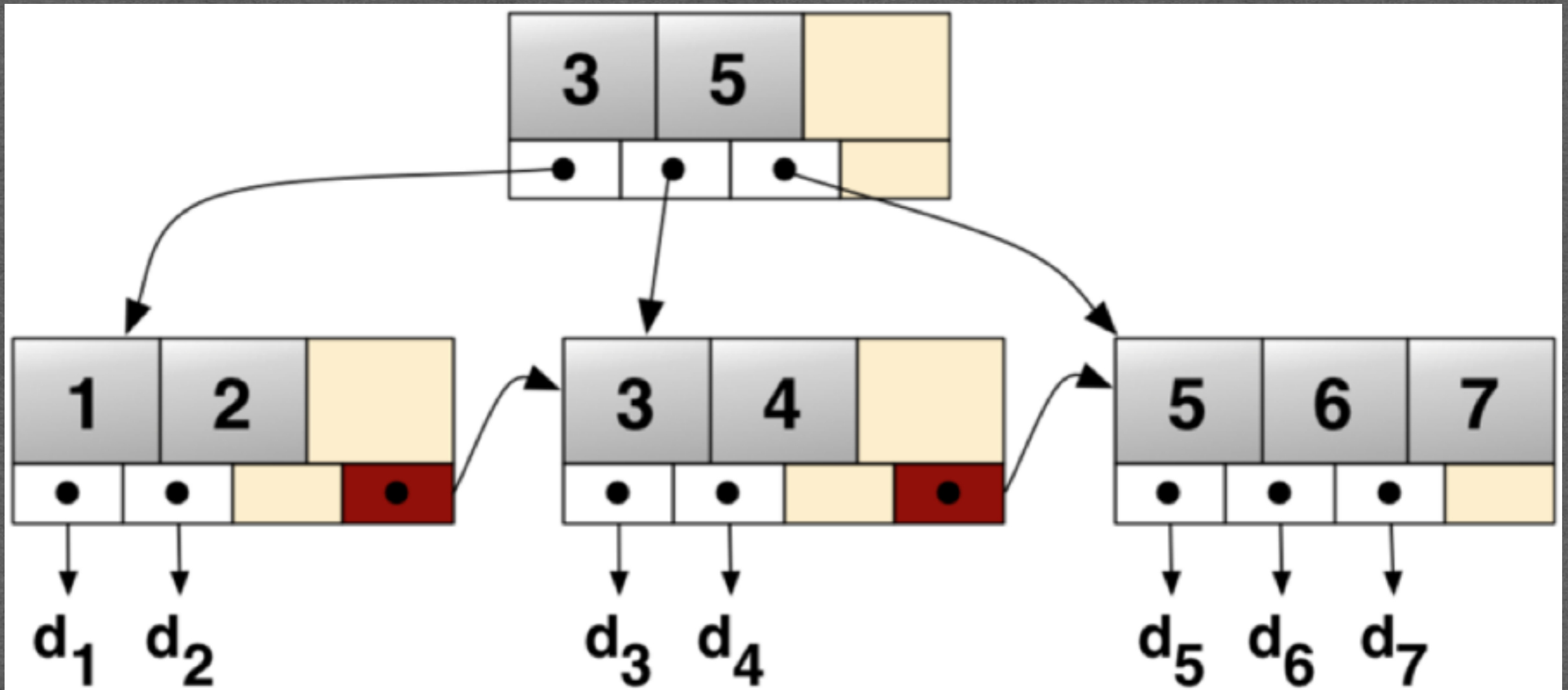
- 从MyISAM转成InnoDB的好处
 - 完整事务特性支持，获得更高并发TPS；
 - 支持crash recovery。MyISAM不能自动修复，且耗时更久；
 - 事实上，只读效率更高。因为InnoDB把数据及索引同时放在buffer中，而MyISAM则只缓存了索引；
- 参考：[\[MySQL FAQ\]系列 — 从MyISAM转到InnoDB需要注意什么](#)

so, InnoDB怎么玩好?

InnoDB的**正确玩法**

- 首先，要正确认识InnoDB
 - 基于B+树结构的聚集索引组织表（类似ORACLE的IOT概念）
 - 表数据的逻辑存储顺序取决于聚集索引的顺序
 - 默认选择主键作为聚集索引，无合适主键时，就用内置生成的ROW_ID作为聚集索引
 - InnoDB的行锁是加在索引上的
 - 支持4个事务隔离级别，默认的RR解决了幻读问题

InnoDB的**正确玩法**



InnoDB的**正确玩法**

- InnoDB表都要有一个主键，且主键最好没有业务用途，不要修改主键值
- 主键最好是保持顺序递增，随机主键值会导致聚集索引树频繁分裂，随机I/O增多，数据离散，性能下降
- 若无特殊需要，要开启事务自动提交 `autocommit=1`。把大事务拆分成多个小事务，小步快跑方式分开提交，避免有大事务未提交导致长时间行锁等待
- 没有索引的更新，可能会导致全表数据都被锁住，和表级锁等同后果
- 定义列属性时，长度预估够用就好，没必要用特别大的数据类型。VARCHAR/TEXT等数据类型中实际存储数据长度越小越好，否则发生行溢出（off-page storage）时对性能影响可能很大
- 不要直接SELECT *读取全部列，可能会导致更多的I/O读

一些优化参考

联合索引怎么用

- 联合索引k1 (c1, c2, c3)
- 下面哪个SQL不能完全用到整个联合索引的?
 - WHERE c1 = ? AND c2 IN (?, ?) AND c3 = ? ✓
 - WHERE c1 = ? AND c2 = ? ORDER BY c3 ✓
 - WHERE c3 = ? AND c1 = ? AND c2 IN (?, ?) ✓
 - WHERE c1 = ? AND c2 IN (?, ?) ORDER BY c3 ✗
- 5.6以前建议新增 (c1, c3) 索引, 5.6以后可以用ICP特性

JOIN怎么优化

- inner join

```
mysql> explain select a.* from t_user_log a inner join t_user b on
      a.uid = b.uid order by a.logintime desc limit 100;
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: b
      type: index
      possible_keys: idx_uid
      key: idx_uid
      key_len: 4
      ref: NULL
      rows: 100105
      Extra: Using index; Using temporary; Using filesort
***** 2. row *****
      id: 1
      select_type: SIMPLE
      table: a
      type: ref
      possible_keys: uid_logintime,idx_uid
      key: idx_uid
      key_len: 4
      ref: yejr.b.uid
      rows: 5
      Extra: NULL
```

JOIN怎么优化

- straight join

```
mysql> explain select a.* from t_user_log a straight_join t_user b on
      a.uid = b.uid order by a.logintime desc limit 100;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: a
        type: index
possible_keys: uid_logintime,idx_uid
         key: logintime
      key_len: 4
         ref: NULL
      rows: 100
     Extra: NULL
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: b
        type: ref
possible_keys: idx_uid
         key: idx_uid
      key_len: 4
         ref: yejr.a.uid
      rows: 1
     Extra: Using index
```

JOIN怎么优化

- INNER JOIN是自动选择JOIN顺序，可能不是最优
- LEFT JOIN & STRAIGHT_JOIN 都是强制左边的表作为驱动表
- RIGHT JOIN则相反
- 多表JOIN中，排序列如果不属于驱动表，则无法利用索引完成排序

硬件、系统、配置选项 优化

硬件优化

- BIOS配置优化

- CPU设置最大性能模式，关闭C1E, C-stats
- 内存设置最大性能模式
- 关闭NUMA

- RAID配置优化

- RAID-10
- CACHE & BBU
- WB & FORCE WB

- 使用PCIe-SSD等高速I/O设备

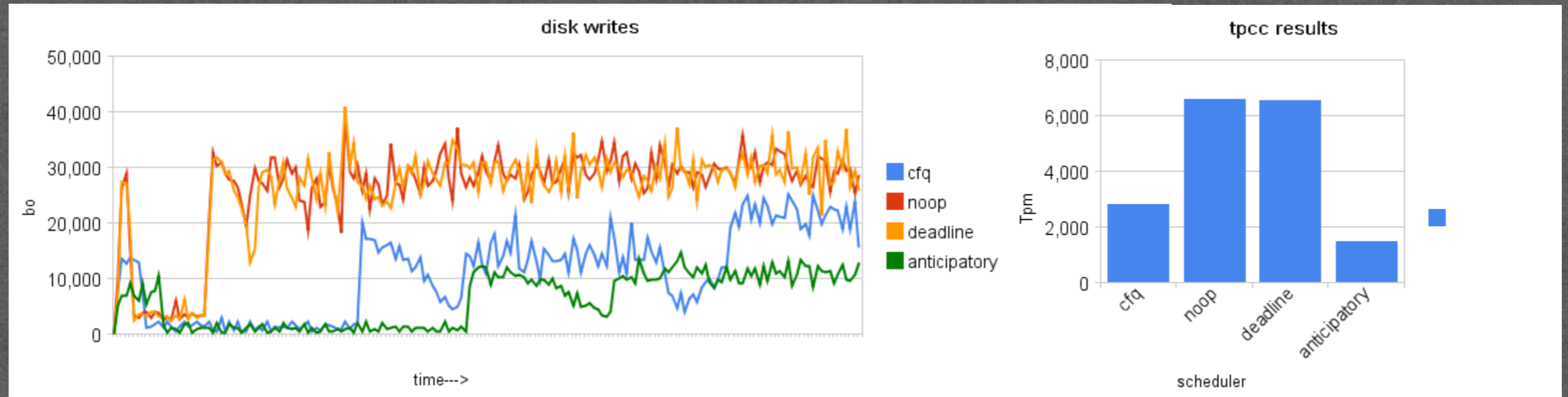
I/O子系统一般是最大瓶颈所在，提高IOPS能力的几种方法：

1. 换SSD, PCIE-SSD (提高IO效率, 普通SAS盘5000以内的iops, 而新设备可达到数万或者数十万iops)
2. 少做IO的活 (合并多次读写为一次, 或者前端加内存CACHE; 或者优化业务, 消除IO)
3. 加大内存 (更多hot data和dirty data放在内存中, 减少物理IO)
4. 调整文件系统为xfs (相比ext3、ext4提高IOPS能力, 高io负载下表现更佳)
5. 调整raid级别为raid 1+0 (相比raid1、raid5等提高IOPS能力)
6. 调整写cache策略为wb或force wb (利用阵列卡cache, 提高iops)
7. io scheduler (优先使用deadline, 如果是SSD, 则使用noop)

系统优化

- io scheduler
 - deadline
 - noop
 - 坚决不能cfq

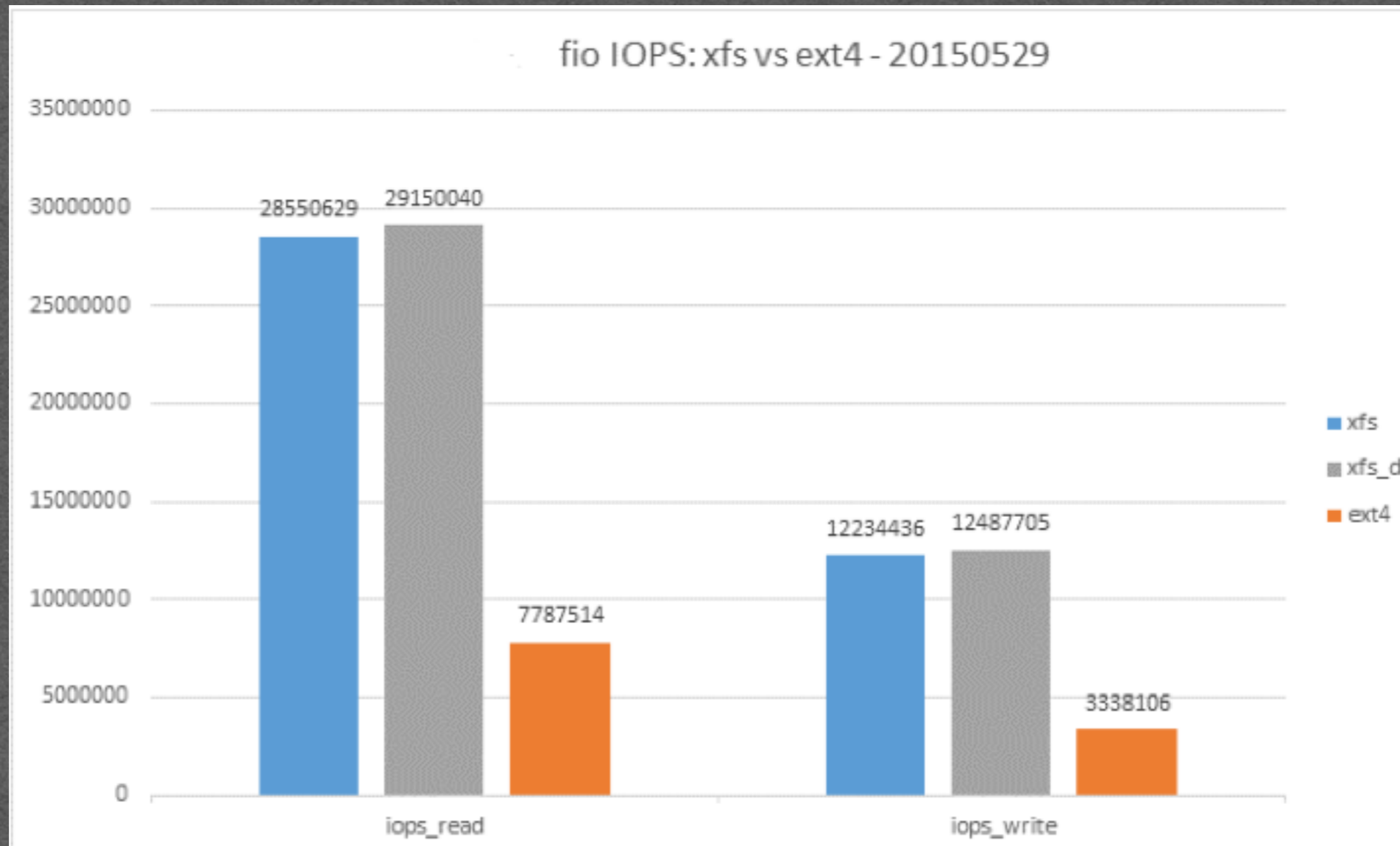
系统优化



系统优化

- fs
 - xfs
 - ext4、zfs等
 - 坚决不能ext3

系统优化



MySQL参数优化

- innodb_buffer_pool_size, 约物理内存的50% ~ 70%
- innodb_data_file_path, 初始化大小至少1G
- 5.6以上版本, 设置独立undo表空间
- innodb_log_file_size, 5.5及以上1G以上, 5.5以下建议不超512M
- innodb_flush_log_at_trx_commit, 0=>最快数据最不安全, 1=>最慢最安全, 2=>折中
- innodb_max_dirty_pages_pct, 25%~50%为宜
- innodb_io_capacity, 普通机械盘=>1000左右, SSD=>10000左右, PCIe SSD=>20000以上

MySQL参数优化

- `key_buffer_size`, 设置32M以下
- `sync_binlog`, 0=>最快数据最不安全, 系统自己决定刷新binlog的频率; 1=>最慢最安全, 每个event刷新一次; N=>每N个事务刷一次binlog
- `long_query_time`, 建议设置小于0.5秒
- `open_files_limit` & `innodb_open_files`, 建议65535
- `max_connections`, 突发最大连接数的80%为宜, 过大容易导致全部卡死
- `thread_handling` = "pool-of-thread", 启用线程池
- `query_cache_size` & `query_cache_type`, 关闭

没踩过 **坑** 的都非正常人

有哪些坑

- QUERY CACHE, 简称QC
- 绝大多数情况下鸡肋, 最好关闭
- QC锁是全局锁, 每次更新QC的内存块锁代价高, 很容易出现Waiting for query cache lock状态
- `query_cache_size = 0 & query_cache_type = 0`, 关闭
- 参考: <http://t.cn/RAF4d7z> <http://t.cn/RAF4d7Z>

ibdata1文件暴增

- ibdata1文件都存储了什么内容?
 - Data dictionary
 - Double write buffer
 - Insert buffer
 - **Rollback segments**
 - **UNDO space**
 - Foreign key constraint system tables

ibdata1文件暴增

- ibdata1文件暴增原因
 - 大量事务，产生大量的undo log
 - 有旧事务长时间未提交，产生大量旧undo log
 - file i/o性能差，purge进度慢
 - 32bit系统下有bug

ibdata1文件暴增

- ibdata1文件暴增解决
 - 升级到5.6及以上（64-bit），采用独立undo表空间
 - 增加purge线程数 `innodb_purge_threads`
 - 提高file i/o能力
 - 事务及时提交，不要积压
 - 默认打开`autocommit = 1`
 - 检查开发框架，确认`autocommit=0`的地方，事务结束后都有提交或回滚

隐式类型转换

- 有个手机号码列
 - iphone CHAR(11) NOT NULL DEFAULT ''
- 有下面这个查询
 - WHERE iphone = 139000000000

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t1	ALL	vid		NULL	NULL	5	Using where

隐式类型转换

- 怎么破
 - 改成WHERE iphone = '139000000000'
 - 或 iphone 列类型改成 BIGINT UNSIGNED

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t1	ref	vid	vid	30	const	1	Using index condition

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	t1	ref	vid	vid	4	const	1	NULL

连接数过高

- too many connections的处理
- 常规的做法：想办法杀掉多余的连接，加大连接数
- 其实应该是这样：限制连接数，设定 `max_user_connections`
- 如果是这样呢： `extra-port`
- 建议：定时检查，干掉慢查询，避免阻塞，自我保护

非典型DBA怎么玩好

MySQL

小规模集群怎么运维

- 备份
 - 从mysqldump (字符集) => xtrabackup (混合引擎)
 - 自动(异地)备份、检查、恢复验证
- 高可用、故障转移
 - 双机+keepalived高可用 (脑裂)
 - 多机MHA高可用 (节点多)
 - 备用异地机房 (成本高)
 - 利用延迟复制特性预防严重误操作

表数据从十万到千万表怎么玩

- 10万表，一般靠简单索引就能搞定
- 100万表，需要开始考虑在线DDL的风险了，以及一些超过1秒的SQL
- 1000万，怎么做备份，上线前预先做SQL解析，预防严重新更难问题
- 需要有辅助手段，监视超过N秒的SQL，能快速自动处理，并且报警出来，后续跟进优化
- 大数据量下，分库分表未必是银弹，反而可能是累赘，优先做冷热数据分离或归档
- 分库分表时优先在同一个实例下，跨实例一般需要靠谱的proxy才行
- pt-query-digest + anemometer

大量日志如何存储

- 存储入库优先用tokudb引擎
- 历史归档导入大数据分析平台
- 日志表可以按时间段分表（分区）
- 定期创建/删除/归档分表（分区）

拥抱5.7

从5.6升级方案

- 逻辑升级方案
 - mysqldump导出
 - 新建实例，导入，恢复数据
- 物理升级方案
 - fast_shutdown=0
 - 拷贝物理文件或xtrabackup备份
 - 新建实例，恢复数据（redo、undo log格式不一样，需要重新初始化）
 - 用mysql_upgrade升级P_S、I_S、mysql库
- 不推荐从5.5到5.7直接升级

5.7重大变化

- 禁用的功能
 - innodb monitor table功能
 - 用innodb_status_output、innodb_status_output_locks开关
 - old-password格式
 - 不安全，赶紧升级好了
 - skip-innodb选项（不能禁用InnoDB了）

值得关注的MySQL 5.7 新特性

- 安全性提升，强化账户管理及密码策略
 - 初始安装默认会设置随机密码（首次登入后需修改）
 - 初始化随机密码保存于error log中
 - 只有root@localhost账号，再无匿名账号
 - 新账号创建后密码默认有过期时间
 - 5.7.10前默认360天，5.7.10后默认0天（不过期）
 - 支持SSL/TLS链接方式

值得关注的MySQL 5.7 新特性

- 标准化提升，不再兼容一些“不规范”用法
 - 默认启用**STRICT_TRANS_TABLES**、**NO_ZERO_IN_DATE**、**ERROR_FOR_DIVISION_BY_ZERO** 等几种规则
 - 例如：向CHAR(20)写入30个字符会报错，而不仅是告警
 - 不再被诟病“太智能”了

值得关注的MySQL 5.7 新特性

- 功能性提升，使用起来更便利
 - Online DDL增强
 - InnoDB引擎修在线调整VARCHAR长度
 - 增加/修改非主键列瞬间完成
 - 修改索引名（非主键）
 - InnoDB buffer pool管理增强
 - 在线动态调整
 - 导出导入增强

值得关注的MySQL 5.7 新特性

- 性能大幅提升
 - InnoDB对Fusion-io Non-Volatile Memory (NVM)原生支持
 - InnoDB索引创建、重建批量加载，提高效率
 - Optimizer Hints
 - 重写了大部分解析器，优化器和成本模型
 - InnoDB只读事务性能提升
 - 改进 InnoDB 的可扩展性和临时表的性能，从而实现更快的网络和大数据加载等操作

值得关注的MySQL 5.7 新特性

- 复制功能提升
 - 并行复制提升
 - 多源复制
 - 在线修改replication filter
 - 在线启用 GTIDs, 和增强的半同步复制
 - 支持Group Replication (实验中)

值得关注的MySQL 5.7 新特性

- 新特性
 - MySQL Router
 - 支持Generated Columns
 - 支持JSON
 - 增加sys schema
 - 支持多个trigger
 - 支持GIS