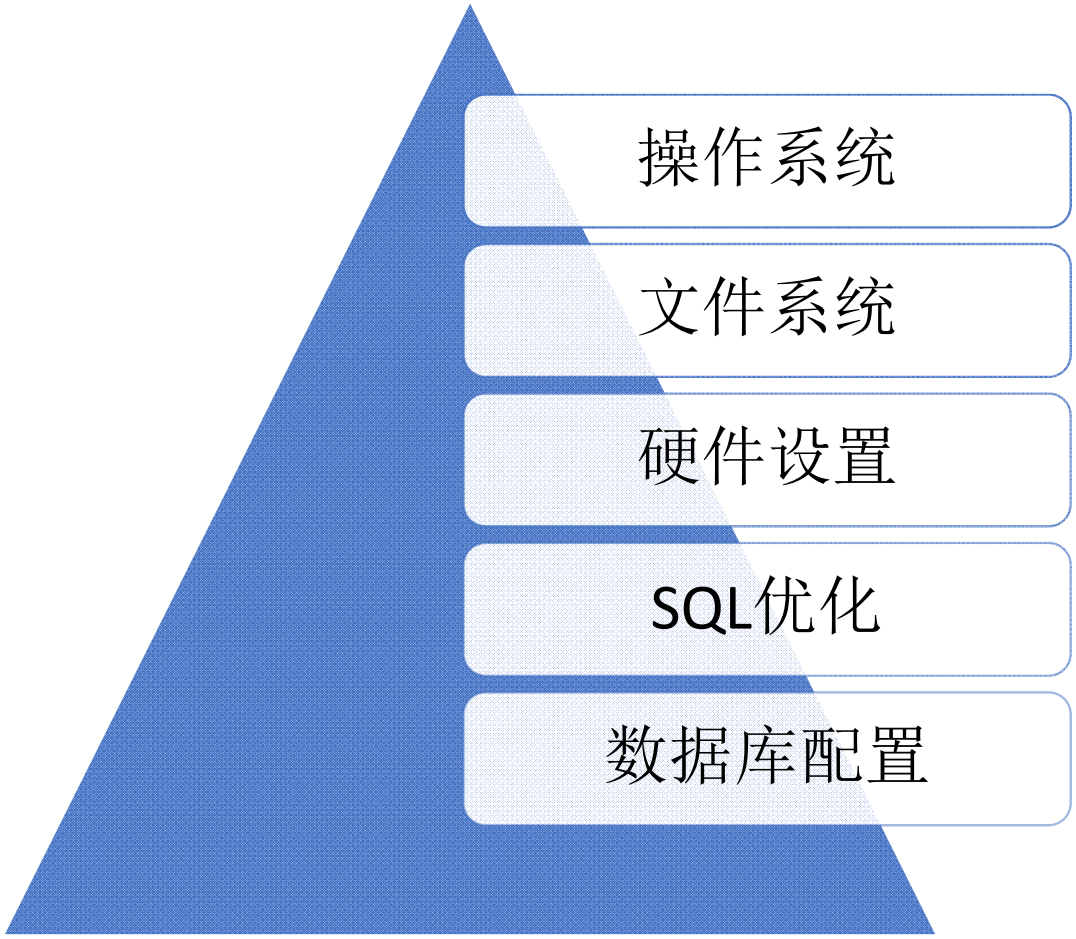
MySQL数据库
性能调优



数据库配置

- InnoDB存储引擎与PostgreSQL非常不同
- InnoDB的缓冲池用来管理所有数据库对象
- 写文件操作通过O_DIRECT选项来避免两次缓存
- InnoDB缓冲池越大性能越好
  - 通常60%~80%
- PostgreSQL缓冲池仅用来管理
- 强烈的依赖操作系统的缓存来
- PostgreSQL缓存越大性能越差
  - 通常25%~30%

```
innodb_buffer_pool_size = 100G
innodb_buffer_pool_instances = 16
innodb_page_size = 4096
innodb_flush_method = O_DIRECT
```

```
# MySQL 5.7 online resize buffer pool

mysql> set global innodb_disable_resize_buffer_pool_debug=off;
Query OK, 0 rows affected (0.00 sec)

mysql> set global innodb_buffer_pool_size=256*1024*1024;
Query OK, 0 rows affected (0.00 sec)
```

数据库配置

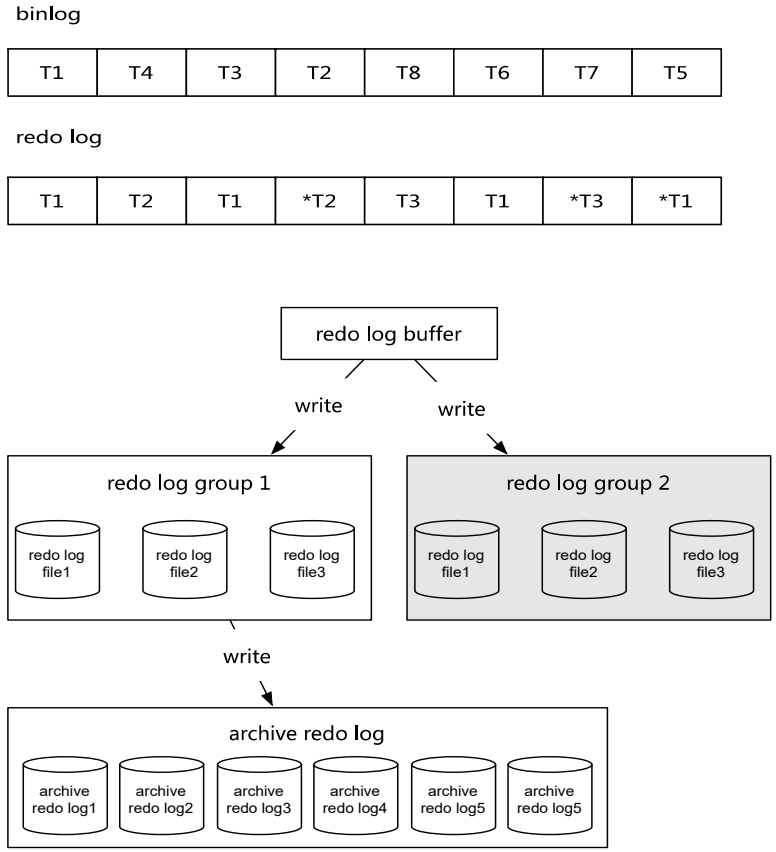
- FUZZY CHECKPOINT
  - 刷新部分脏页，对系统影响较小
  - 5.6：独立的刷新线程
  - 5.7：并行刷新线程
  - innodb_io_capacity
- SHARP CHECKPOINT
  - 刷新全部的脏页，系统hang住
  - innodb_fast_shutdown
- Neighbor Page Flush
  - innodb_flush_neighbors

```
innodb_io_capacity = 1000/4000/8000
innodb_page_cleaners = 1 / 4
innodb_fast_shutdown = 0/1
innodb_flush_neighbors = 0/1/2
```

数据库配置

- 重做日志
  - 记录页操作的日志
  - 与二进制日志完全不同
  - 循环覆盖写
  - 默认没有类似PG或者Oracle的归档
- 重做日志大小限制
  - before 5.6: max 4G
  - start from 5.6: max 512G

```
innodb_log_file_size = 1900M / 4G
innodb_log_buffer_size = 8M / 32M
innodb_log_files_in_group = 2/3
innodb_log_group_home_dir = /redolog/
```



数据库配置

- undo段
  - 实现回滚
  - 实现MVCC功能
  - PostgreSQL没有undo段！！！
- undo段数量
  - before MySQL 5.5: 1024
  - start from MySQL 5.5: 128*1024
- undo回收
  - purge

```
mysql> SHOW VARIABLES LIKE 'innodb_undo%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| innodb_undo_directory    | .     |
| innodb_undo_logs         | 128   |
| innodb_undo_tablespaces  | 3     |
+--------------------------+-------+
3 rows in set (0.00 sec)

mysql> SHOW VARIABLES LIKE 'datadir';
+---------------+--------------------------------+
| Variable_name | Value                          |
+---------------+--------------------------------+
| datadir       | /Users/david/mysql_data/data/ |
+---------------+--------------------------------+
1 row in set (0.00 sec)

mysql> system ls -lh /Users/david/mysql_data/data/undo*
-rw-rw----  1 david  staff    10M 11 22 16:55 /Users/david/mysql_data/data/undo001
-rw-rw----  1 david  staff    10M 11 22 16:51 /Users/david/mysql_data/data/undo002
-rw-rw----  1 david  staff    10M 11 22 16:51 /Users/david/mysql_data/data/undo003
```

innodb_undo_directory = /undolog/
innodb_undo_logs = 128
innodb_undo_tablespaces = 3

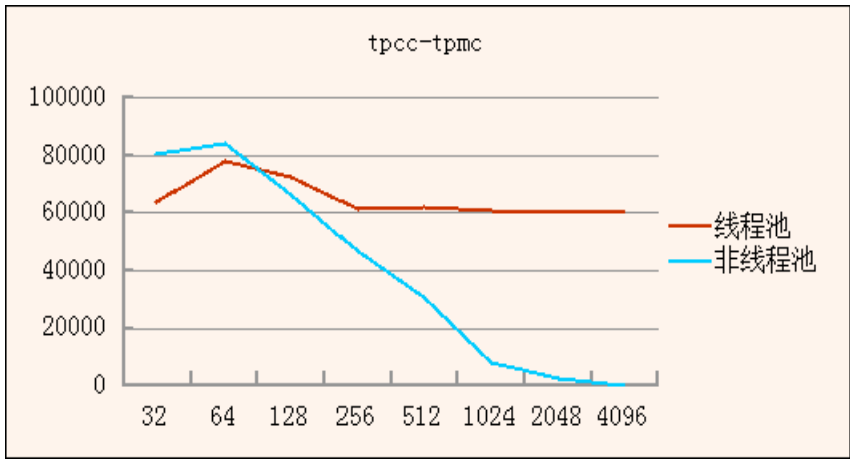
innodb_undo_log_truncate = 1
innodb_max_undo_log_size = 1G
innodb_purge_rseg_truncate_frequency = 128

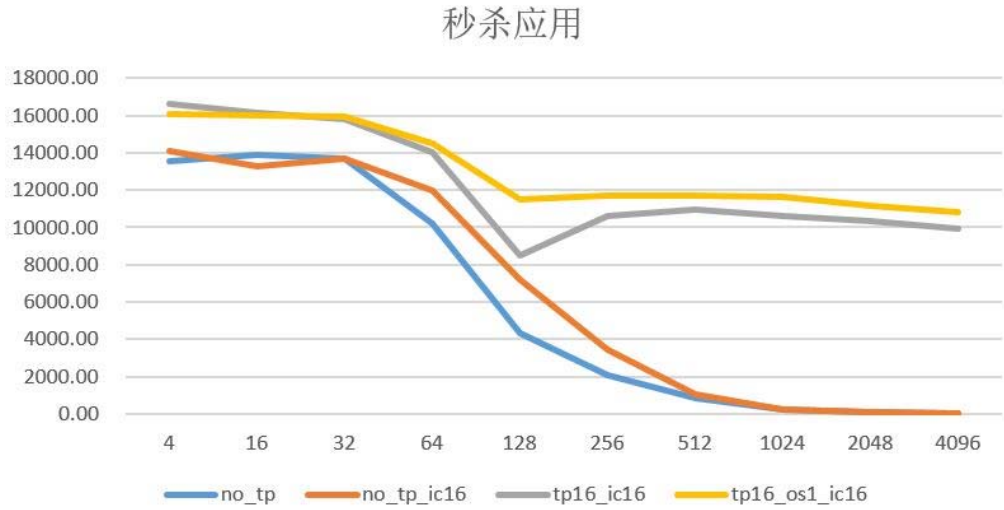
innodb_purge_batch_size = 300
innodb_purge_threads = 4/8

数据库配置

- 线程池
  - 保障高并发下的性能平稳
  - MariaDB线程池没有优先级队列
  - 推荐MySQL/InnoSQL/Percona线程池
  - 推荐默认开启用线程池

```
thread_handling = pool-of-threads
thread_pool_size = 32 # CPU
thread_pool_oversubscribe = 3
extra_port = 3333 #额外的端口
```





数据库配置

- MySQL日志配置
  - binary log
  - error log
  - slow log
  - general log（通常不推荐）
    - events_statements_current
    - events_statements_history(_long)

```
[mysqld]
performance_schema

mysql> select * from setup_consumers\G
*************************** 1. row ***************************
  NAME: events_stages_current
ENABLED: NO
.....
```

```
log-bin = /binlog/mysqld-bin
log-expire-day = 7
```

```
syslog
syslog_tag = stock #mysqld_stock
```

```
log-slow-queries
long_query_time = 2
log-queries-not-using-indexes
log-slow-admin-statements
min-examined-row-limit
log_throttle_queries_not_using_indexes
log_slow_slave_statements
```

SQL优化

- 子查询
  - before 5.6：
    - lazy: rewrite to exists
    - poor performance
  - from 5.6：（MariaDB 5.3）
    - semi-join

```
mysql> show variables like optimizer_switch'\G
*************************** 1. row ***************************
Variable_name: optimizer_switch
        Value:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstmatch=on,subquery_materialization_cost_based=on,use_index_extensions=on,condition_fanout_filter=on,derived_merge=on
```

SQL优化

```
SELECT o_custkey FROM orders
WHERE o_custkey IN
   ( SELECT c_custkey FROM customer
      WHERE c_acctbal < -500 );
```

```
select `dbt3`.`orders`.`o_custkey` AS `o_custkey` from
`dbt3`.`customer` join `dbt3`.`orders` where
((`dbt3`.`orders`.`o_custkey` = `dbt3`.`customer`.`c_custkey`)
and (`dbt3`.`customer`.`c_acctbal` < -(500)))
```

MySQL 5.6

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	customer	range	PRIMARY,idx_c_acctbal	idx_c_acctbal	9	NULL	6802	100.00	Using where; Using index
1	PRIMARY	orders	ref	i_o_custkey	i_o_custkey	5	dbt3.customer.c_custkey	7	100.00	Using index

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	orders	index	NULL	i_o_custkey	5	NULL	1502510	100.00	Using where; Using index
2	DEPENDENT SUBQUERY	customer	unique_subquery	PRIMARY,idx_c_acctbal	PRIMARY	4	func	1	100.00	Using where

```
select `dbt3`.`orders`.`o_custkey` AS `o_custkey` from `dbt3`.`orders` where
<in_optimizer>(`dbt3`.`orders`.`o_custkey`,<exists>(<primary_index_lookup>(<cache>(`dbt3`.`orders`
.`o_custkey`) in customer on PRIMARY where ((`dbt3`.`customer`.`c_acctbal` < <cache>(-(500))) and
(<cache>(`dbt3`.`orders`.`o_custkey`) = `dbt3`.`customer`.`c_custkey`)))))
```

SQL优化

```
SELECT * FROM part
WHERE p_partkey IN
    ( SELECT l_partkey FROM lineitem
        WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-02-01' )
ORDER BY p_retailprice DESC LIMIT 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	part	ALL	PRIMARY	NULL	NULL	NULL	199755	Using temporary; Using filesort
1	PRIMARY	<subquery2>	eq_ref	distinct_key	distinct_key	5	func	1	
2	MATERIALIZED	lineitem	range	i_l_shipdate,i_l_suppkey_partkey,i_l_partkey	i_l_shipdate	4	NULL	157032	Using index condition

SQL优化——JOIN

- JOIN语法
- JOIN算法
- JOIN经典问题

SQL优化——语法

```
SELECT … FROM
a,b
WHERE a.x = b.x
```

```
SELECT … FROM a
INNER JOIN b
on a.x = b.x
```

```
SELECT … FROM a
JOIN b
on a.x = b.x
```

Q：上述这些语法是否有区别？

A：没有任何区别

Q：哪个性能更好？

A：没有任何区别

A：好吧，如果要认真算的话，那么3最好，因为字节数最少

Q：那为什么需要不同的语法？

A：ANSI SQL 89、ANSI SQL 92语法标准

A：ANSI 92标准开始支持OUTER JOIN

A：INNER JOIN可以省略INNER关键字

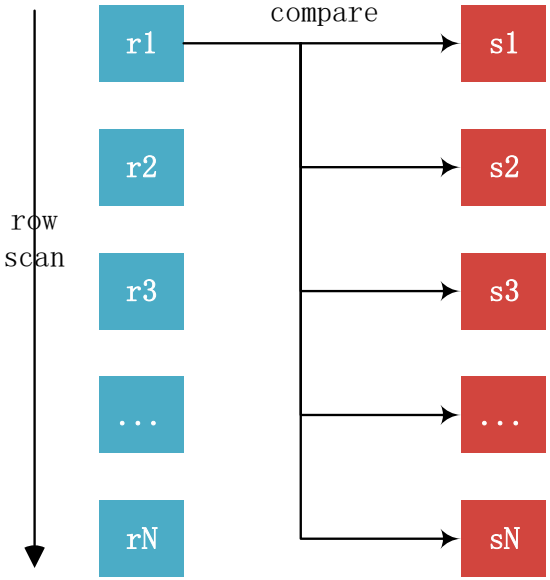
SQL优化——JOIN算法

- nested_loop join
  - simple nested-loop join
  - index nested-loop join
  - block nested-loop join
- classic hash join
  - Only support in MariaDB
- bached key access join
  - from MySQL 5.6
  - from MariaDB 5.5

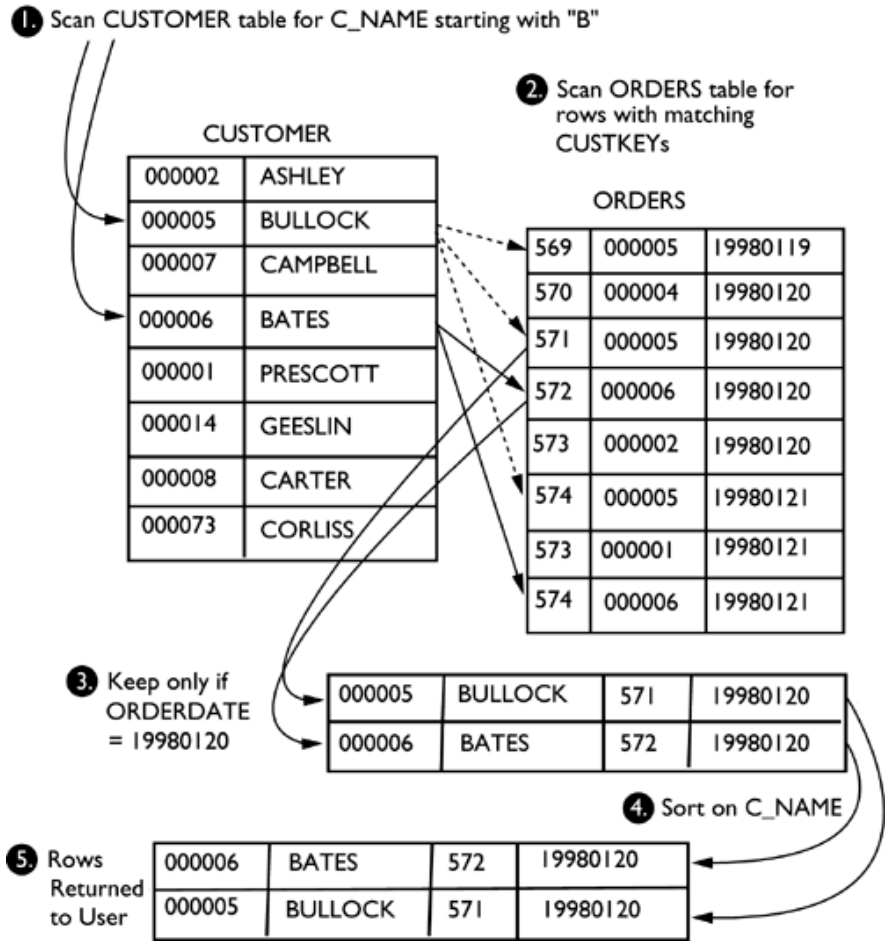
SQL优化——JOIN算法

- simple nested_loop join

```
For each row r in R do
        For each row s in S do
                If r and s satisfy the join condition
                        Then output the tuple <r,s>
```



扫描成本： O（Rn×Sn）

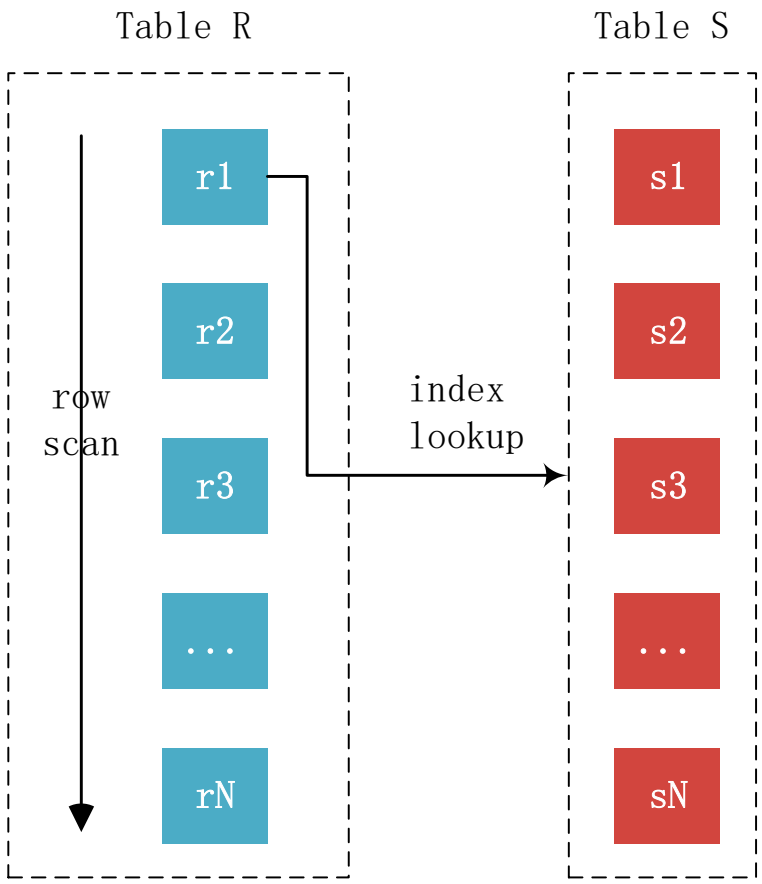


SQL优化——JOIN算法

- index nested_loop join

```
For each row r in R do
        lookup r in S index
                if found s == r
                        Then output the tuple <r,s>
```

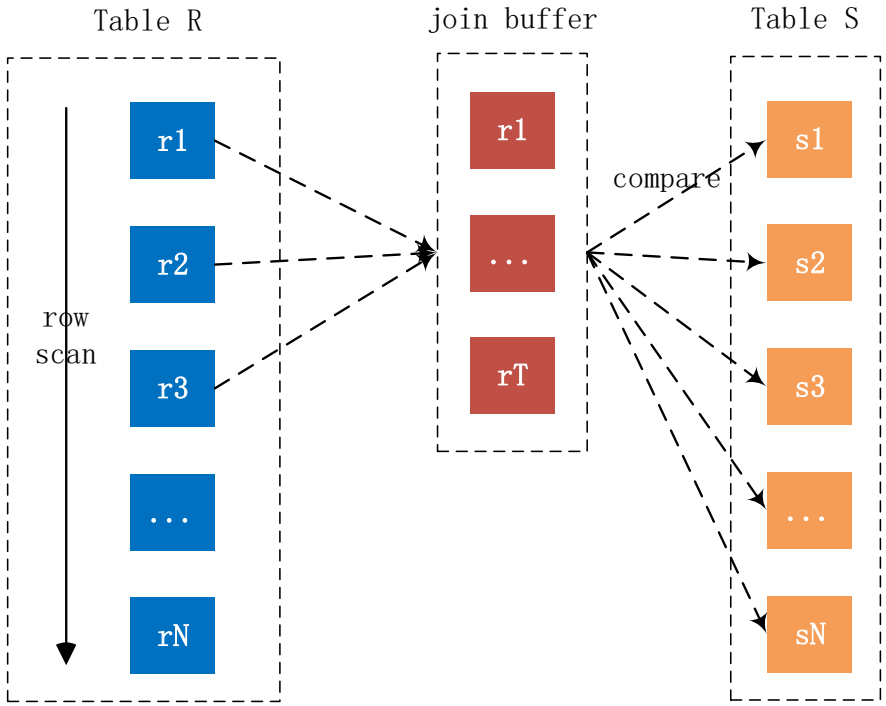
扫描成本： O（Rn）
优化器倾向于使用小表做驱动表



SQL优化——JOIN算法

- block nested-loop join
  - 优化simple nested-loop join
  - 减少内部表的扫描次数

```
For each tuple r in R do
        store used columns as p from R in join buffer
        For each tuple s in S do
                If p and s satisfy the join condition
                        Then output the tuple <p,s>
```



系统变量join_buffer_size决定了Join Buffer的大小
Join Buffer可被用于联接是ALL，index，range的类型
Join Buffer只存储需要进行查询操作的相关列数据，而不是整行的记录
扫描成本呢？

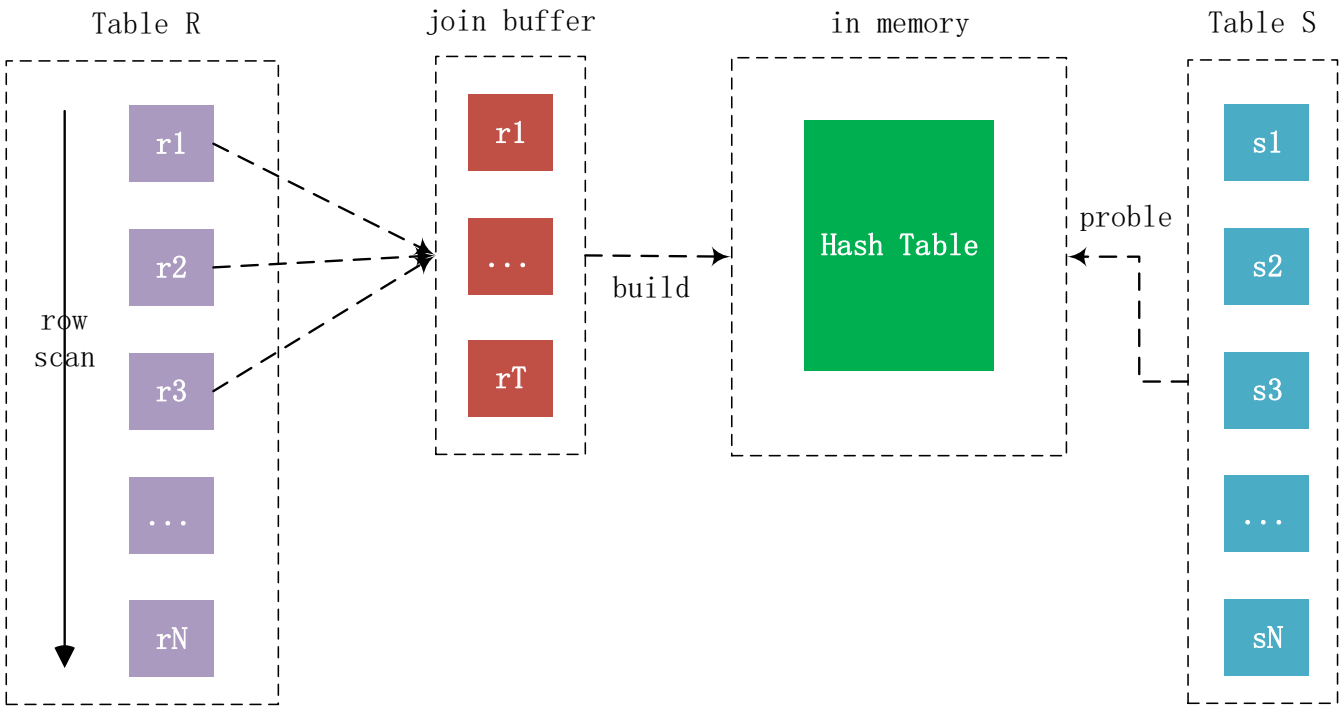
SQL优化——JOIN算法

- classic hash join
  - based on block nested loop join
    - inner table may scan many times
  - not grace hash join
    - inner table scan only once

```
For each tuple r in R do
        store used columns as p from R in join buffer
        build hash table according join buffer
         for each tuple s in S do
                probe hash table
                if  find
        Then output the tuple <p,s>
```

```
SET join_cache_level=4+;
SET optimizer_switch='join_cache_hashed=on';
```

SQL优化——JOIN算法



减少外表扫描次数
减少内表比较次数
扫描成本？

SQL优化——JOIN算法

```
SELECT MAX(l_extendedprice) FROM orders, lineitem
WHERE
o_orderdate BETWEEN '1995-01-01' AND '1995-01-31'
AND l_orderkey=o_orderkey;
```

MySQL 5.5 *125.3sec*

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	range	PRIMARY,i_o_orderdate	i_o_orderdate	4	NULL	42008	Using where; Using index
1	SIMPLE	lineitem	ref	PRIMARY,i_l_orderkey,i_l_orderkey_quantity	PRIMARY	4	dbt3.orders.o_orderkey	1	

MariaDB 5.3 *23.104sec* ~5x

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	range	PRIMARY,i_o_orderdate	i_o_orderdate	4	NULL	42008	Using where; Using index
1	SIMPLE	lineitem	hash_ALL	PRIMARY,i_l_orderkey,i_l_orderkey_quantity	#hash#PRIMARY	4	dbt3.orders.o_orderkey	5994679	Using join buffer (flat, BNLH join)

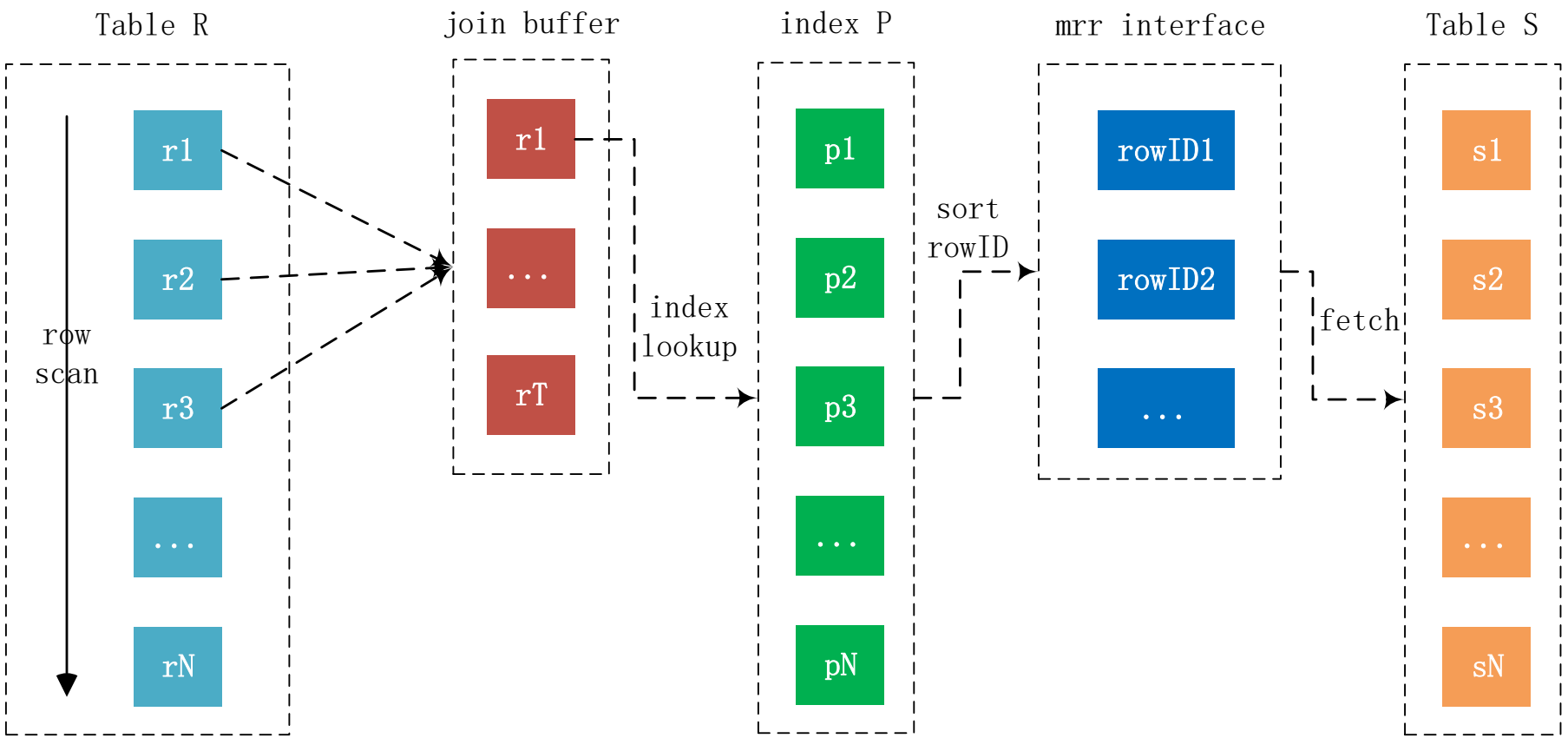
SQL优化——JOIN算法

- batched key access join
  - not enabled by default
  - optimize random I/O

```
For each tuple r in R do
        store used columns as p from R in join buffer
        For each tuple s in S do
                If p and s satisfy the join condition
                        use mrr inter face to sort row Id
                        Then output the tuple <p,s>
```

```
mysql> SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

SQL优化——JOIN算法



SQL优化——JOIN经典问题

- 行号问题

```
mysql> select * from employees order by emp_no limit 10;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 |
|  10006 | 1953-04-20 | Anneke     | Preusig   | F      | 1989-06-02 |
|  10007 | 1957-05-23 | Tzvetan    | Zielinski | F      | 1989-02-10 |
|  10008 | 1958-02-19 | Saniya     | Kalloufi  | M      | 1994-09-15 |
|  10009 | 1952-04-19 | Sumant     | Peac      | F      | 1985-02-18 |
|  10010 | 1963-06-01 | Duangkaew  | Piveteau  | F      | 1989-08-24 |
+--------+------------+------------+-----------+--------+------------+
10 rows in set (0.00 sec)
```

SQL优化——JOIN经典问题

- 子查询

```
SELECT emp_no,dept_no,
(SELECT COUNT(1) FROM dept_emp t2
WHERE t1.emp_no<=t2.emp_no) AS row_num
FROM dept_emp t1;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	t1	index	NULL	emp_no	4	NULL	331883	Using index
2	DEPENDENT SUBQUERY	t2	index	PRIMARY,emp_no	emp_no	4	NULL	331883	Using where; Using index

SQL优化——JOIN经典问题

- CROSS JOIN

```
SELECT emp_no,dept_no,@a:=@a+1 AS row_num
FROM dept_emp,(SELECT @a:=0 ) t;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	system	NULL	NULL	NULL	NULL	1	
1	PRIMARY	dept_emp	index	NULL	emp_no	4	NULL	331883	Using index
2	DERIVED	NULL	NULL	NULL	NULL	NULL	NULL	NULL	No tables used

SQL优化——JOIN经典问题

- 实现类似Oracle的rank()函数

id	rank_column	rank
1	10	1
2	20	2
3	30	3
4	30	3
5	30	3
6	40	4
7	50	5
8	50	5
9	50	5

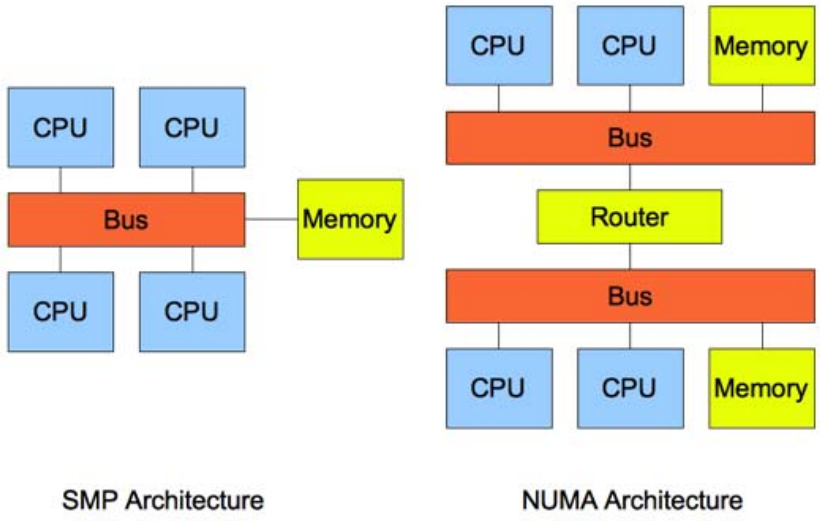
```
SET @prev_value = NULL;
SET @rank_count = 0;
SELECT id, rank_column, CASE
   WHEN @prev_value = rank_column THEN @rank_count
   WHEN @prev_value := rank_column THEN @rank_count := @rank_count + 1
END AS rank
FROM rank_table
ORDER BY rank_column
```

软硬件设置

- 内存
- 网卡
- RAID卡
- SSD

软硬件设置——内存

- NUMA
  - Non-Uniform Memory Access
  - 非一致存储访问结构



NUMA的内存分配策略有四种：

1. default：总是在本地节点分配
2. bind：强制分配到指定节点上
3. interleave：在所有节点或者指定的节点上交织分配
4. preferred：在指定节点上分配，失败则在其他节点上分配

软硬件设置——内存

- 单实例MySQL
- 考虑关闭NUMA特性
  - BIOS中关闭
  - 内存启动关闭numa=off
  - MySQL启动时关闭

- 多实例MySQL
- 通过NUMA绑定到指定CPU

```
kernel /vmlinuz-2.6.32-220.el6.x86_64 ro
root=/dev/mapper/VolGroup-root rd_NO_LUKS
LANG=en_US.UTF-8 rd_LVM_LV=VolGroup/root rd_NO_MD
quiet SYSFONT=latarcyrheb-sun16 rhgb crashkernel=auto
rd_LVM_LV=VolGroup/swap rhgb crashkernel=auto quiet
KEYBOARDTYPE=pc KEYTABLE=us rd_NO_DM  numa=off
```

```
numactl --interleave=all  mysqld &
```

```
numactl –hardware
numactl –cpubind=0 –localalloc
```

```
echo "vm.swappiness = 0" >>/etc/sysctl.conf
```

软硬件设置——网卡

- 网卡软中断

08:57:08 PM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
08:57:09 PM	all	32.47	0.00	9.72	5.59	0.13	6.29	0.00	0.00	45.81
08:57:09 PM	0	39.80	0.00	13.27	5.10	0.00	5.10	0.00	0.00	36.73
08:57:09 PM	1	39.39	0.00	5.05	1.01	0.00	48.48	0.00	0.00	6.06
08:57:09 PM	2	38.54	0.00	9.38	6.25	0.00	4.17	0.00	0.00	41.67
08:57:09 PM	3	38.38	0.00	12.12	6.06	2.02	4.04	0.00	0.00	37.37
08:57:09 PM	4	33.33	0.00	11.46	7.29	0.00	3.12	0.00	0.00	44.79
08:57:09 PM	5	37.00	0.00	10.00	7.00	0.00	4.00	0.00	0.00	42.00
08:57:09 PM	6	28.12	0.00	11.46	6.25	0.00	3.12	0.00	0.00	51.04
08:57:09 PM	7	35.35	0.00	9.09	6.06	0.00	3.03	0.00	0.00	46.46
08:57:09 PM	8	28.00	0.00	12.00	6.00	0.00	3.00	0.00	0.00	51.00
08:57:09 PM	9	53.47	0.00	9.90	6.93	0.00	4.95	0.00	0.00	24.75
08:57:09 PM	10	25.51	0.00	7.14	4.08	0.00	2.04	0.00	0.00	61.22
08:57:09 PM	11	29.59	0.00	10.20	6.12	0.00	4.08	0.00	0.00	50.00
08:57:09 PM	12	20.62	0.00	10.31	4.12	0.00	2.06	0.00	0.00	62.89
08:57:09 PM	13	32.67	0.00	7.92	4.95	0.00	2.97	0.00	0.00	51.49
08:57:09 PM	14	18.37	0.00	7.14	4.08	0.00	1.02	0.00	0.00	69.39
08:57:09 PM	15	24.24	0.00	8.08	6.06	0.00	2.02	0.00	0.00	59.60

软硬件设置——网卡

- 解决方案
  - 启用网卡多队列
    - set_irq_affinity.sh
    - service irqbalance stop

```
#./set_irq_affinity.sh 0 eth0
#./set_irq_affinity.sh 8 eth1
```

05:59:05 PM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
05:59:06 PM	all	63.27	0.00	14.83	1.14	0.00	12.18	0.00	0.00	8.58
05:59:06 PM	0	57.73	0.00	14.43	1.03	0.00	20.62	0.00	0.00	6.19
05:59:06 PM	1	55.10	0.00	16.33	2.04	0.00	21.43	0.00	0.00	5.10
05:59:06 PM	2	62.24	0.00	17.35	1.02	0.00	11.22	0.00	0.00	8.16
05:59:06 PM	3	62.63	0.00	15.15	1.01	0.00	14.14	0.00	0.00	7.07
05:59:06 PM	4	63.92	0.00	16.49	1.03	0.00	13.40	0.00	0.00	5.15
05:59:06 PM	5	63.00	0.00	14.00	1.00	0.00	14.00	0.00	0.00	8.00
05:59:06 PM	6	67.00	0.00	12.00	1.00	0.00	14.00	0.00	0.00	6.00
05:59:06 PM	7	65.66	0.00	13.13	1.01	0.00	12.12	0.00	0.00	8.08
05:59:06 PM	8	51.02	0.00	12.24	1.02	0.00	26.53	0.00	0.00	9.18
05:59:06 PM	9	57.00	0.00	13.00	1.00	0.00	24.00	0.00	0.00	5.00
05:59:06 PM	10	55.10	0.00	13.27	1.02	0.00	24.49	0.00	0.00	6.12
05:59:06 PM	11	58.00	0.00	13.00	0.00	0.00	22.00	0.00	0.00	7.00
05:59:06 PM	12	55.56	0.00	12.12	1.01	0.00	23.23	0.00	0.00	8.08
05:59:06 PM	13	60.61	0.00	14.14	0.00	0.00	18.18	0.00	0.00	7.07
05:59:06 PM	14	54.55	0.00	12.12	2.02	0.00	22.22	0.00	0.00	9.09
05:59:06 PM	15	58.33	0.00	14.58	1.04	0.00	18.75	0.00	0.00	7.29

软硬件设置——RAID卡

- BBU
  - Battery Backup Unit
  - 非低端RAID卡都带BBU
  - 需要电池保证写入的可靠性
  - 电池有充放电时间
- RAID卡缓存
  - Write Backup
  - Write Through
  - 写缓存并非默认开启

软硬件设置——RAID卡

查看电量百分比

```
[root@test_raid ~]# megacli -AdpBbuCmd -GetBbuStatus -aALL |grep "Relative State of Charge"
Relative State of Charge: 100 %
```

查看充电状态

```
[root@test_raid ~]# megacli -AdpBbuCmd -GetBbuStatus -aALL |grep "Charger Status"
Charger Status: Complete
```

查看缓存策略

```
[root@test_raid ~]# megacli -LDGetProp -Cache -LALL -a0
Adapter 0-VD 0(target id: 0): Cache Policy:WriteBack, ReadAdaptive, Direct, No Write Cache if bad BBU
```

软硬件设置——RAID卡

缓存策略

WT (Write through)
WB (Write back)
NORA (No read ahead)
RA (Read ahead)
ADRA (Adaptive read ahead)
Cached
Direct

-RW|RO|Blocked|RemoveBlocked | **WT|WB|ForcedWB** [-Immediate]
|RA|NORA | DsblP | Cached|Direct | -EnDskCache|DisDskCache |
CachedBadBBU|NoCachedBadBBU
-Lx|-L0,1,2|-Lall -aN|-a0,1,2|-aALL

```
[root@test_raid ~]# megacli -LDSetProp WT -L0 -a0

Set Write Policy to WriteThrough on Adapter 0, VD 0 (target id: 0) success
```

SSD

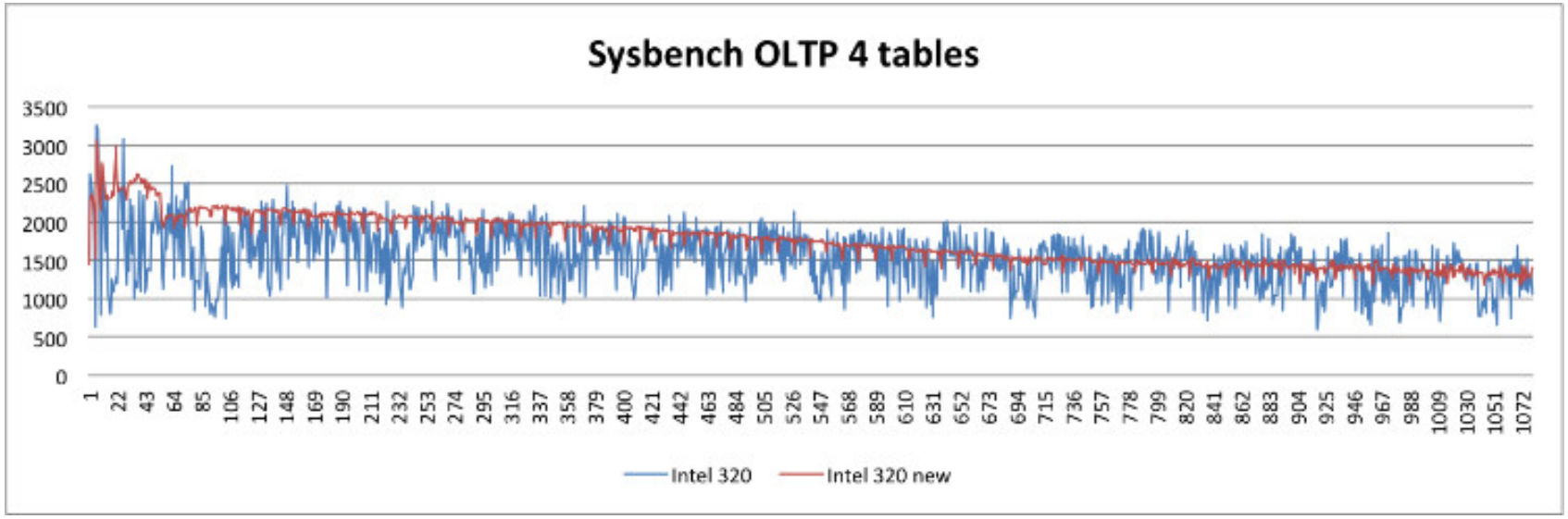
SSD

- 纯电设备
- 由Flash Memory组成
- 没有读写磁头
- IOPS高
  - 50000+ IOPS
  - 读写速度非对称

- 性能指标
  - RPM （rotations per minute）
    - 5400
    - 7200
    - 10000
    - 15000
- SATA
  - 120 ～ 150 IOPS
- SAS
  - 150 ～ 200 IOPS

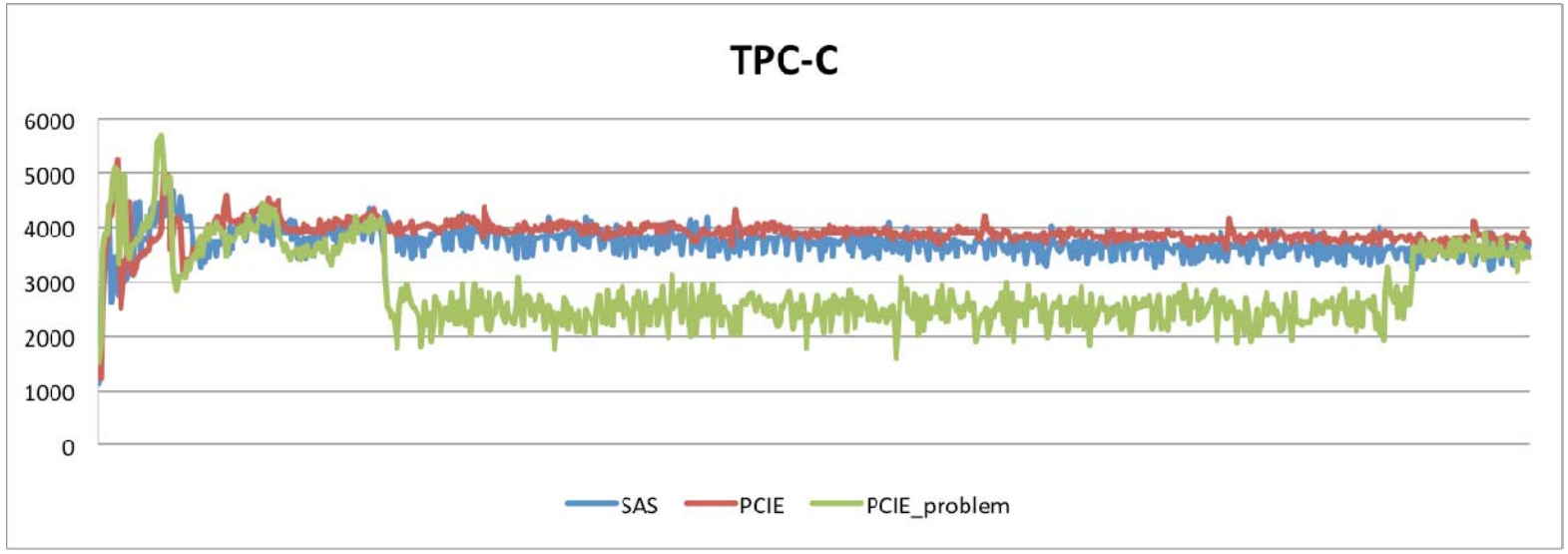
SSD

- 性能下降



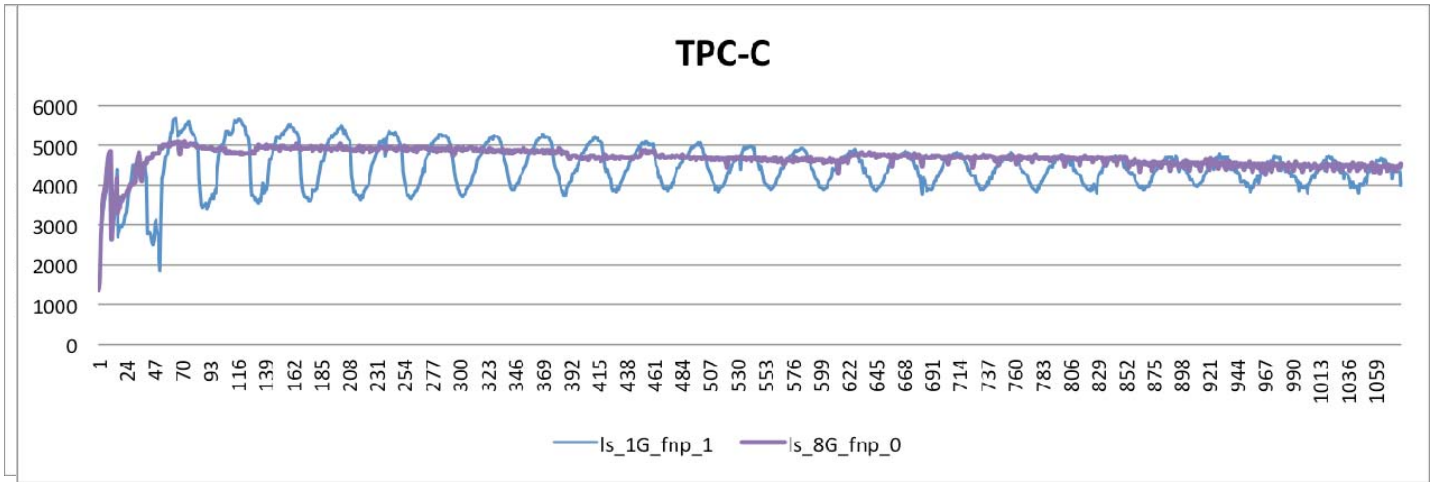
SSD

- 莫名的性能波动



SSD

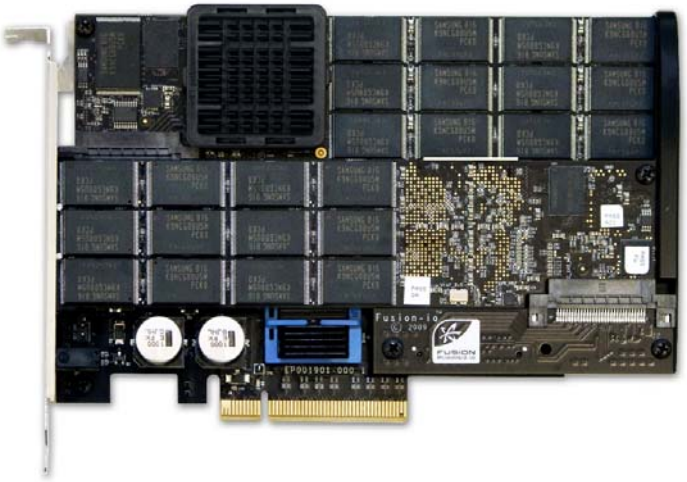
- SSD与数据库优化
  - 磁盘调度算法设置为：deadline或者noop
  - InnoDB存储引擎参数设置
    - innodb_flush_neighbors=0
    - innodb_log_file_size=4G



结论：
- 性能更平稳
- 可以有大约15%的性能提升

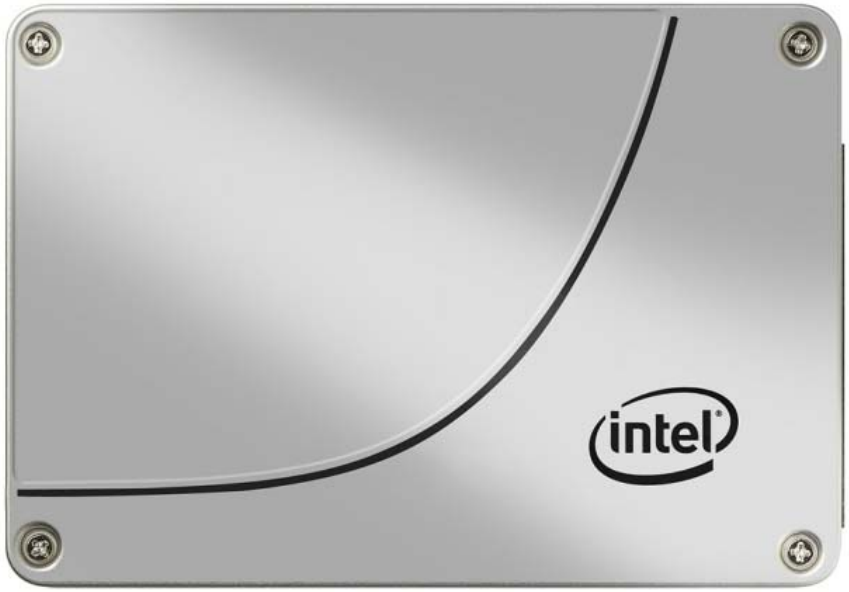
SSD

- SSD选择
  - PCIE or SATA/SAS
  - SATA/SAS益于安装与升级
  - SATA/SAS与PCIE的性能差距逐渐缩小
  - PCIE的性能很少有应用可以完全使用
  - **优先选择SATA/SAS接口的SSD**
- SSD品牌推荐
  - Intel
  - FusionIO
  - 宝存



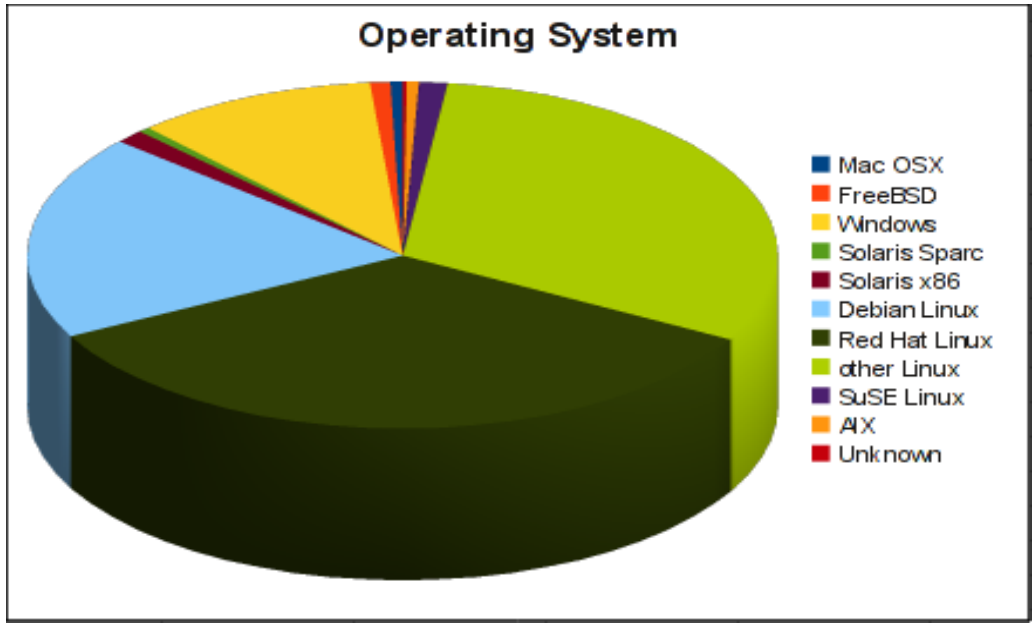
SSD

- 选购时注意“寿命”指标
  - Intel 3500：275T
  - Intel 3700：每天全量写10次，保证5年



文件系统与操作系统

- 文件系统
  - 推荐xfs/ext4
  - noatime
  - nobarrier
- 操作系统
  - 推荐Linux操作系统
  - 关闭swap
  - 磁盘调度算法




```
mount -o noatime,nobarrier /dev/sdb1 /data
```