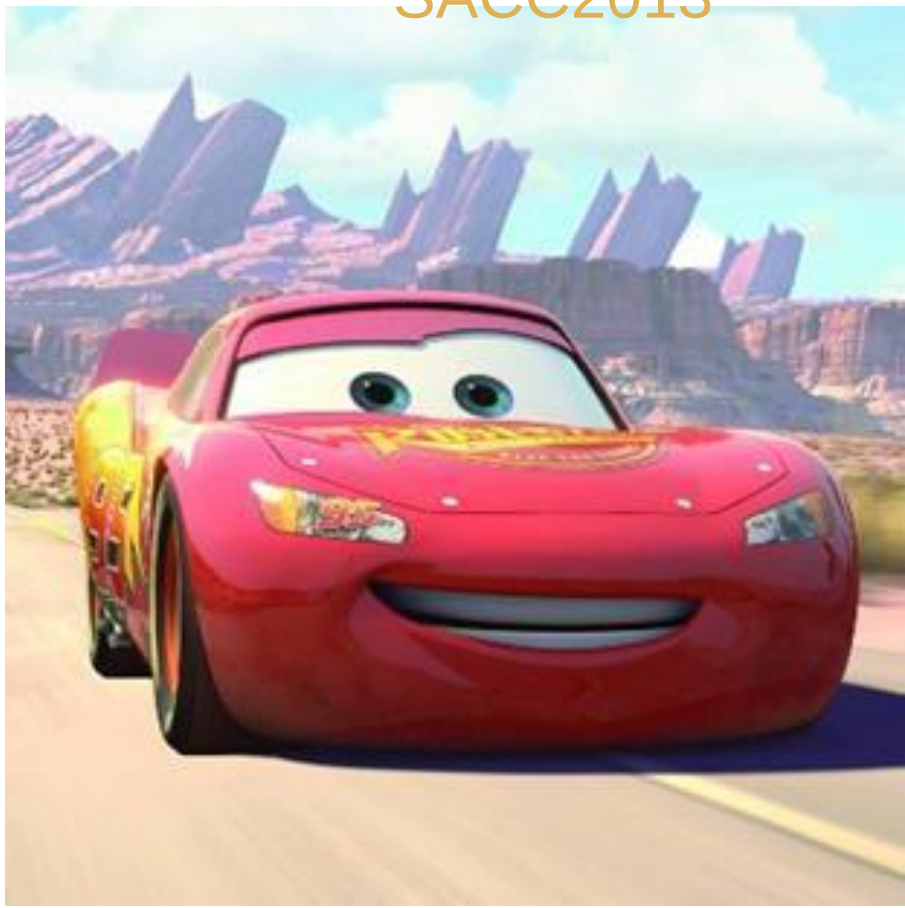
SACC2013

数据库Capacity实践

性能来自业务特性和技术特性的结合

韦连友

速度！速度！



- 数据快速增长

  大数据分析。Capacity需求越来越高。

  （可扩展性）

- 业务快速变化

  一个好的IDEA或产品战略，需要快速实现，才能获得市场先机。

  （可维护性 ）

如何应对

- 明确使命

  应用=技术+业务。以技术创新（集成创新）实现业务创新。

- 理解本质算法

  理解业务特性的内在算法和技术特性的内置算法，并将它们有机结合（平衡和妥协）。

- 做好规划

  要适度超前。不要过度超前，业务和技术在不断发展变化。

- 提高执行力

  知易行难。Ownership.

业务和技术结合案例1

- 订单分析

  90%的订单, 2周内完成。95%的订单一个月内完成。99%的订单2个月内完成，但24个月内也无法保证100%完成。于是报表分析总是要处理所有的数据 。

- 解决方案

  1. 随着数据量增长，不断升级硬件。
  2. 使用云数据库，不断增加节点。
  3. 梳理订单流程，保证订单在2-3月内100%完成。
  4. 使用物化视图，做增量刷新。

业务和技术结合案例2

- 我的订单

  各个电商的“我的订单”的设计都不完全一样，除了用户体验的考虑外，所对应的技术实现方式也不一样。

- 设想

  用户最关心的是什么？ 假设：未完成的订单和刚刚完成的订单！

  我们可以区分未完成的订单（包括刚刚完成的）和已完成的订单，那么技术实现方式又会大为不同。

- 性能优化

  降低复杂度（1,n,n*m）：加快速度、降低次数、减少扫描范围。
  可实现数量级的性能提升。

- 数据库选型

  不同的数据库特性适合不同的业务场景。
  可实现性价比和效率。

- 架构演变

  硬件升级，读写分离，垂直拆分，水平拆分。
  可实现线性扩展和业务优先级。

性能优化

- 流程规范和工具（主动式）
- topsql监控和优化（被动式）
- 优化案例

主动式—基本

- 80%的性能问题，都来自基本的东西。

- 绑定变量，执行计划
- 索引，分区表
- 事务，批量事务，锁
- 表关联，in list等sql写法
- 性能优化案例
- 事故案例

。。。。。。

知道，理解，用得好。 说出来，写出来，让别人懂。 是不同的层次！

主动式—工具

- 开发了海葵开发支持系统。开发先做sql review，dba再review。

>> 海葵生产->测试 联系DBA | 查 询 | 导 出 | 表结构 | DML执行 | 脚本执行 | DBError | Mongo查询 | 重新登录

mysql-gsslog ▾ 表 字段 索引 序列 清空 For example: user_data2.seq_end_user

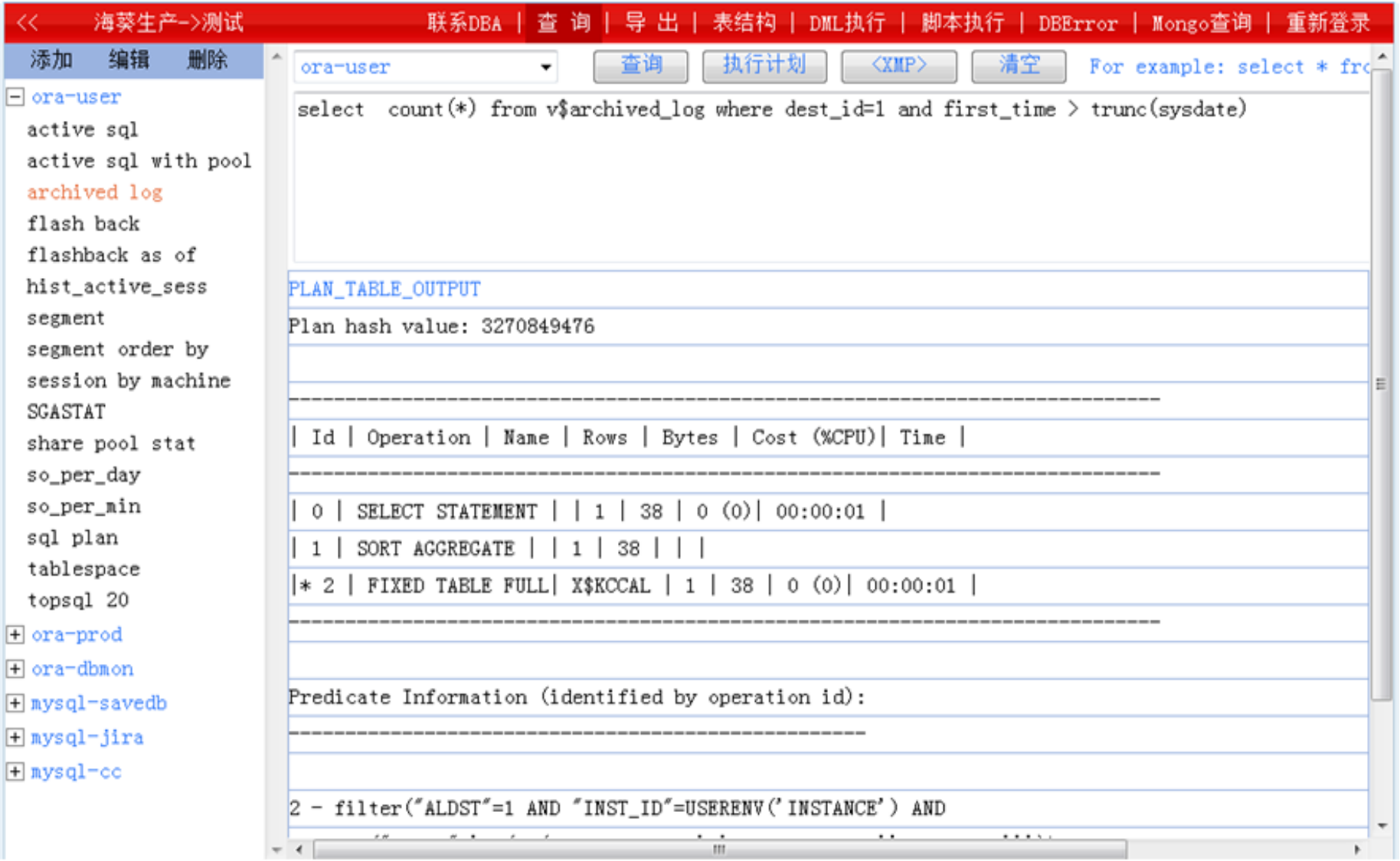
gss_invoke_log_1

TABLE_SCHEMA	TABLE_NAME	TABLE_ROWS	AVG_ROW_LENGTH	AUTO_INCREMENT	COLUMN_NAME	ORDINAL_POSITION	COLUMN_TYPE	NUMERIC_SCALE	COLUMN_DEFAULT	IS_NULLABLE	COLUMN_COMMENT
gss_log	gss_invoke_log_1	43961892	188	43988756	ID	1	bigint(18)	0	null	NO	主键，采用自动增长方式生成
gss_log	gss_invoke_log_1	43961892	188	43988756	CREATE_TIME	2	datetime	null	null	YES	记录创建的时间
gss_log	gss_invoke_log_1	43961892	188	43988756	INVOKE_SOURCE	3	varchar(20)	null	null	YES	修改数据的调用源
gss_log	gss_invoke_log_1	43961892	188	43988756	INVOKER_IP	4	varchar (200)	null	null	YES	修改数据的调用源IP
gss_log	gss_invoke_log_1	43961892	188	43988756	TICKET	5	varchar(36)	null	null	YES	调用操作标识
gss_log	gss_invoke_log_1	43961892	188	43988756	SERVICE_NAME	6	varchar(36)	null	null	YES	调用源的服务名称
gss_log	gss_invoke_log_1	43961892	188	43988756	MAIN_REMARK	7	varchar(50)	null	null	YES	调用源的主要备注
gss_log	gss_invoke_log_1	43961892	188	43988756	SECONDARY_REMARK	8	varchar (250)	null	null	YES	调用源的附加备注
gss_log	gss_invoke_log_1	43961892	188	43988756	SERVER_IP	9	varchar (200)	null	null	YES	gss服务器的IP
gss_log	gss_invoke_log_1	43961892	188	43988756	OP_CODE	10	varchar (200)	null	null	YES	调用操作码
gss_log	gss_invoke_log_1	43961892	188	43988756	OP_CODE_TYPE	11	int(6)	0	null	YES	调用操作码类型

共 11 条记录

主动式— 工具2

SACC2013



主动式—测试

- 技术特性测试

  Exadata测试。

  Mysql的测试（不同cpu数的服务器、不同linux内核版本、
  不同mysql版本和 不同类型文件系统的组合）。

  Mongodb和Mysql的对比测试。

- 新硬件的测试

  PCIe，SSD等等。

- 案例

  1. Mysql单表可以支持多少条记录？ B树效率 or 命中率 or Mutex？

被动式—Oracle

- Oracle awr的topsql是按累计耗资源排序的。按平均单条最耗资源排序的topsql ，以及全表扫描的topsql，可以更早定位和解决潜在的性能隐患。并提供界面供开发查询各种维度的topsql。

elapsed_time,	elapsed_time /executions
cpu_time ,	cpu_time /executions
buffer_gets,	buffer_gets /executions
disk_reads,	disk_reads /executions
iowait,	iowait /executions
……	

被动式—MYAWR

- 结合mysql慢查询和performance_schema等，我的一个小伙伴模仿awr开发了myawr。 已经开源： https://github.com/noodba/myawr.

TOP 20 SQL

checksum	db name	ts_min	ts_max	ts_cnt	Query_time_sum	Query_time_pct_95	Lock_time_sum	Lock_time_pct_95	Rows_sent_sum	Rows_sent_pct_95	sample
17518085470290754732	test	2013-05-21 09:32:56	2013-05-21 19:14:40	31705	11346.3	0.526151	6573.05	0.35612	0	0	update xxxx_test ...
8783231904896245142	test	2013-05-21 08:55:12	2013-05-21 21:27:49	894	5128.98	6.98463	0.044442	0.0000632544	894	1	SELECT count(id) as
3360175983660887134	test	2013-05-21 08:54:54	2013-05-21 21:27:23	151	1861.87	15.2466	0.007739	0.0000697379	151	1	SELECT count(id) as
133096531796313320993	test	2013-05-21 21:48:26	2013-05-21 23:15:39	293	265.682	0.992137	0.031267	0.000119276	145903	487.094	SELECT b.id, b...
14565464025717298162	test	2013-05-21 01:48:45	2013-05-21 21:48:35	20	239.178	16.8094	0.00456	0.000316473	0	0	INSERT INTO AAAA_
13312995569039381386	test	2013-05-21 00:03:31	2013-05-21 23:59:56	321166	65.3112	0.000204002	19.4566	0.0000807304	321166	1	select last_insert_i
103545740566486809809	test	2013-05-21 18:19:21	2013-05-21 18:19:21	1	41.0879	41.0879	0.00011	0.00011	20	20	SELECT ID, MI...
4977342209851205OO	test	2013-05-21 00:05:20	2013-05-21 23:51:39	1913	17.1919	0.0416167	0.202465	0.000131501	18598	9.82808	select * from (s...
9637830689706458981	test	2013-05-21 23:39:24	2013-05-22 00:01:16	76	8.10788	0.140929	0.008022	0.000119276	38000	500	SELECT b.id, b...
14126161800600546731	test	2013-05-21 21:47:48	2013-05-21 21:47:48	1	4.76375	4.76375	0.000105	0.000105	0	0	Update test. bat...
17471270398124473757	test	2013-05-21 23:39:04	2013-05-21 23:39:04	1	2.35301	2.35301	0.000077	0.000077	0	0	Update test.exce...
16832470833228384350	test	2013-05-21 10:09:00	2013-05-21 18:27:46	25	2.11104	0.134218	0.005074	0.000214202	154	7.70056	select t.operator_i
13191748705407557481	test	2013-05-21 09:54:07	2013-05-21 21:21:21	51	1.18354	0.0243324	0.01047	0.0000697379	7	0.992137	select t.ID,...
12670572290300457907	test	2013-05-21 01:48:53	2013-05-21 23:49:09	195	0.907769	0.00486338	0.015995	0.0000934555	161	0.992137	select * from liv...
7307725764902681358	test	2013-05-21 09:55:48	2013-05-21 21:21:21	41	0.888554	0.0231737	0.002063	0.0000664171	7	0.992137	select t.ID,...
4414610479190720S2	test	2013-05-21 09:55:15	2013-05-21 21:21:21	32	0.707033	0.0231737	0.001794	0.0000632544	7	0.992137	select t.ID,...
59594964694652907S8	test	2013-05-21 00:16:17	2013-05-21 23:58:42	2453	0.627288	0.000273382	0.151339	0.0000768861	2298	0.992137	SELECT id,t.hits FR
78400363582974852I3	information_schema	2013-05-21 14:12:02	2013-05-22 00:01:02	590	0.385329	0.000799713	0.033036	0.0000632544	131570	223	select * from perfor
16675922391026170S3	test	2013-05-21 09:39:35	2013-05-21 17:58:50	13	0.306861	0.0481765	0.000869	0.0000732248	13	1	SELECT MIN(PUBLIS
4588707098242522987	information_schema	2013-05-21 14:12:02	2013-05-22 00:01:02	590	0.29407	0.000568341	0.054093	0.000113596	590	1	select VARIABLE_VA

TOP SQL DETAIL

checksum	sql detail
17518085470290754732	update test set plag = 2, uptime = now(), opid = 1 where ordid = 116459410
8783231904896245142	SELECT count(id) as count FROM zzz_log where code='200001' and ccdoe='100001'
3360175983660887134	SELECT count(id) as count FROM zzzz where code='200001'
4588707098242522987	select VARIABLE_VALUE from information_schema.GLOBAL_VARIABLES where VARIABLE_NAME='PERFORMANCE_SCHEMA'
.....more.....	.....more.....

Mysql WorkLoad Report

Host Name	Ip addr	Port	Db role	Version	Uptime
db2.11	192.168.2.11	3306	master	5.5.27	0y 2m 2d 7h 55mi 33s

	Snap Id	Snap Time	Threads_connected	Threads_running
Begin Snap:	1	2013-05-21 14:12:02	481.00	3.00
End Snap:	589	2013-05-22 00:00:02	450.00	2.00
Elapsed:	35280 (seconds)			

Some Key Load Info

	Per Second
TPS:	347
Com_select(s):	120
Com_insert(s):	336
Com_update(s):	11
Com_delete(s):	0
Innodb t_row PS:	351
Innodb_rows_inserted(s):	336
Innodb_rows_updated(s):	14
Innodb_rows_deleted(s):	0
Innodb_rows_read(/s):	109494
Innodb_data_reads(s):	2
Innodb_data_writes(s):	708
Innodb_data_read(kb/s):	35
Innodb_data_written(kb/s):	1117
Innodb_os_log_fsyncs(s):	1
Innodb_os_log_written(kb/s):	602

Some Key Hits

	Percentage
key_buffer_read_hits %:	99.99
key_buffer_write_hits %:	45.22
Innodb_buffer_read_hits %:	99.99
Query_cache_hits %:	0
Thread_cache_hits %:	68.55

Top 10 Timed Events

event_name	wait time(picsecond)	wait count
wait/synch/cond/sql/MYSQL_BIN_LOG::update_cond	34636103255361957	11871685
wait/synch/cond/mysys/my_thread_var::suspend	7008666931980153	25357

Active SQL监控

- 第一时间发现，尽快优化，或保存现场”以供事故分析。

主题: prod-10.0.1.131:1521:item_1: 10 wait events. - 2013-08-16 00:22:54

5731	7415	543	0qqfqpd6gvwar	3757186007	PROD_DATA2	xen2221-vm01	inshop/openapi-network	enq: TX - row lock contention	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id = :2 and
8879	7415	110	0qqfqpd6gvwar	3757186007	PROD_DATA2	xen2221-vm01	inshop/openapi-network	enq: TX - row lock contention	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id = :2 and
7777	7415	617	0qqfqpd6gvwar	3757186007	PROD_DATA2	xen2221-vm01	inshop/openapi-network	enq: TX - row lock contention	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id = :2 and
7495	7415	195	0qqfqpd6gvwar	3757186007	PROD_DATA2	xen4201-vm03	inshop/openapi-network	enq: TX - row lock contention	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id = :2 and
12607	7415	282	0qqfqpd6gvwar	3757186007	PROD_DATA2	xen4201-vm03	inshop/openapi-network	enq: TX - row lock contention	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id
112	7415	349	0qqfqpd6g	3757186007	PROD_DA	xen4202-	inshop/openapi-	enq: TX - row lock	UPDATE adplan_con_product set STATUS = :1 , UPDATE_TIME = sysDate WHERE id = :2 and

去关联的例子

- 去关联可以大幅提高性能，也是拆分和迁移NoSQL的必要条件。

```
select count(1)  from a, b
    where a.id = b.a_id
      and a.create_time >= :1 and a.create_time < :2
       and b. product_id = :3
```

时间范围是1天的话， create_time选择性大约是几百分之1。
如果是热门商品，product_id 的选择性可能也是 千分之1。
如果把a.create_time冗余到b表中，sql可以改写为：

```
select count(1) from b
    where b.create_time >= :1 and b.create_time < :2
      and b. product_id = :3
```

product_id+create_time 的选择性是几十万分之1.

其他经典案例

- 缓存

  多级缓存：存储缓存、db缓存、KV、应用缓存、CDN、client缓存。

- 时间稳定性

  性能随时间累积而变慢。不必要的扫描范围扩大。增量和全量的报表处理。

- 100%覆盖

  99%的场景，单次查询需要1ms。1%的场景，单次查询需要几秒。为了覆盖100%的场景，单次查询都是几秒。

- 批量处理

  所谓批，不是1个，不是所有，而是N。批量提交。bitmap索引的批量特性。

- 预处理

  可以共用的或反复使用的中间表。物化视图。

- 过度需求

  合理的输入控制。技术反向优化业务。dba反向优化技术实现。

数据库选型-SQL

- Oracle RAC & Exadata
- RAC要做好应用分离。可以接近线性。
- Exadata在特定场景下的性能优势非常明显。恶劣场景依然保持线性。

- Mysql & Mysql 5.6
- 复杂sql性能较差。 多cpu支持较差。高并发下吞吐量衰减很快。
- Mysql 5.6的多cpu支持，已基本达到线性。可以不必拆分得太细。

数据库选型-NoSQL

- NoSQL将SQL的部分功能做了极致的实现
- 充分利用廉价内存。（降低一致性要求）
- 无关联。（单表查询）
- 无限扩展。（无限并行度的并行查询）
- Map-Reduce。（无限分区的分区表）

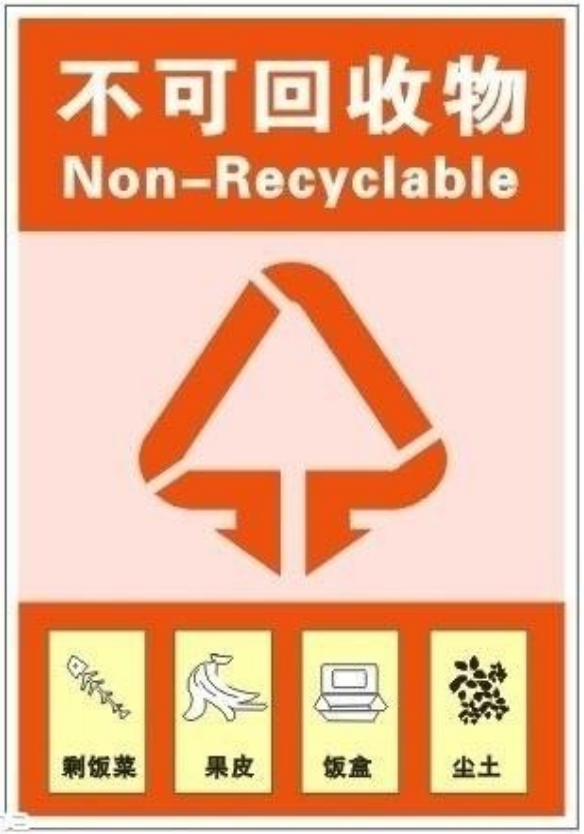
- Mongodb替代Mysql做单表查询

  内存充足的情况下，将mysql的数据一致性要求降到最低，其性能还是不及mongodb。
- Hadoop替代Oracle做并行报表处理

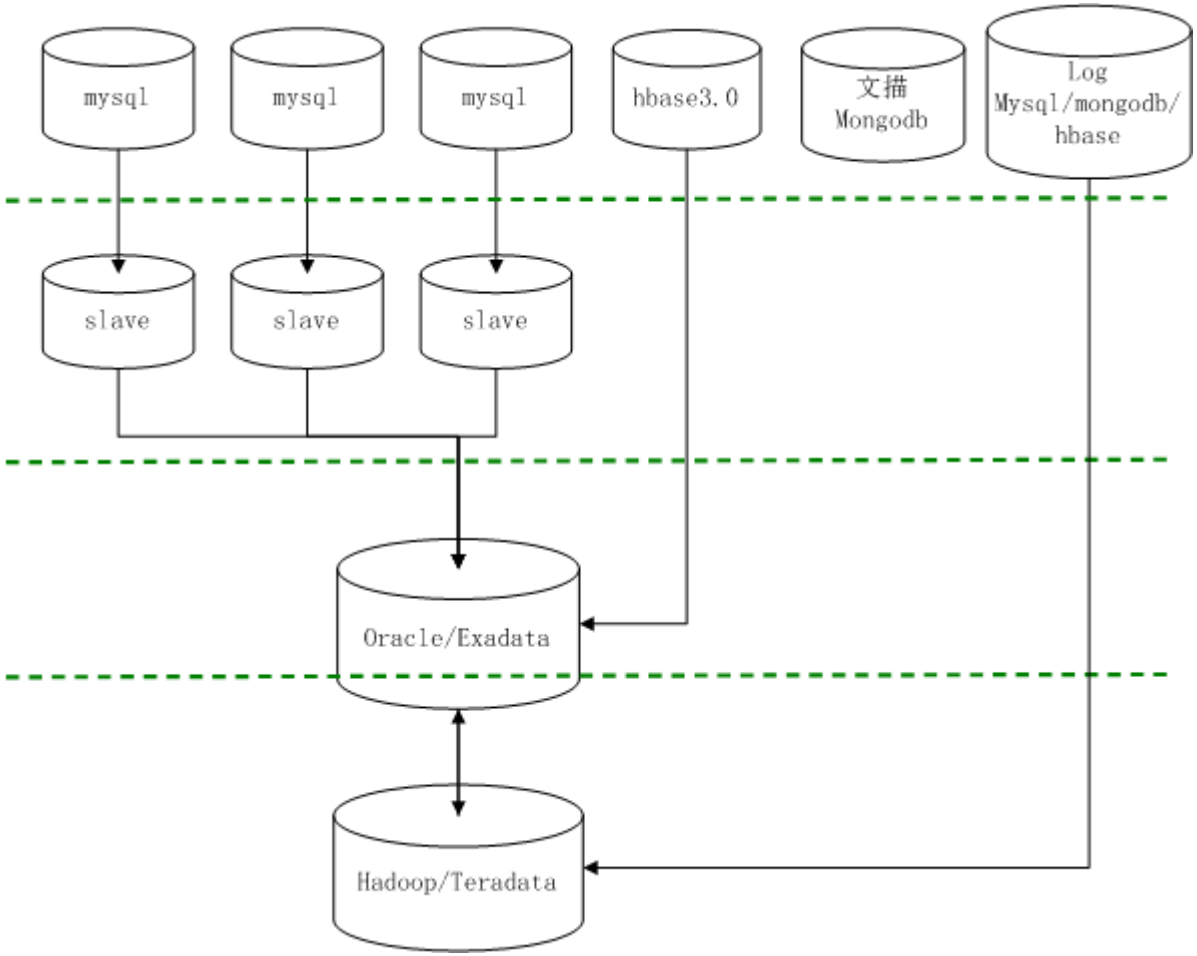
  100个节点（12core）的hadoop集群。

  1200Core的Oracle或100个节点（12core）的RAC（2400个分区，2400个并发度），做并行查询。

- 分类产生价值，垃圾变成资源



分类别分层次的架构



交易和实时查询

只读查询（接近实时）

跨库查询和复杂查询
（延迟几分钟）

数据分析和挖掘

- 硬件升级

  系统架构越简单越稳定。优先升级硬件，尤其不要吝啬内存。
  代码透明。

- 读写分离

  通常90%以上的性能压力都来自查询。
  可实现线性扩展和部分业务优先级划分。
  需要保证延迟最小，延迟被业务接受，程序支持读写分离。

- 垂直拆分

  核心业务，非核心业务，日志等分类处理。
  可实现线性扩展和业务优先级划分。同时减小备库延迟，有利读写分离。
  需要做好去关联，业务解耦，异步事务补偿等。

- 水平拆分

  适用于订单等重点业务的线性扩展。
  需要强大的中间层支持。云架构！

没有秘方

也没有神龙秘籍

唯有真正理解业务和技术

提高执行力

才能事半功倍地实现满足应用需求的Capacity

Thanks.

BLOG: www.yhddba.com

招聘邮箱: dba@yihaodian.com