Tokyo Cabinet Key-Value 数据库及其扩展应用

2010-04-02

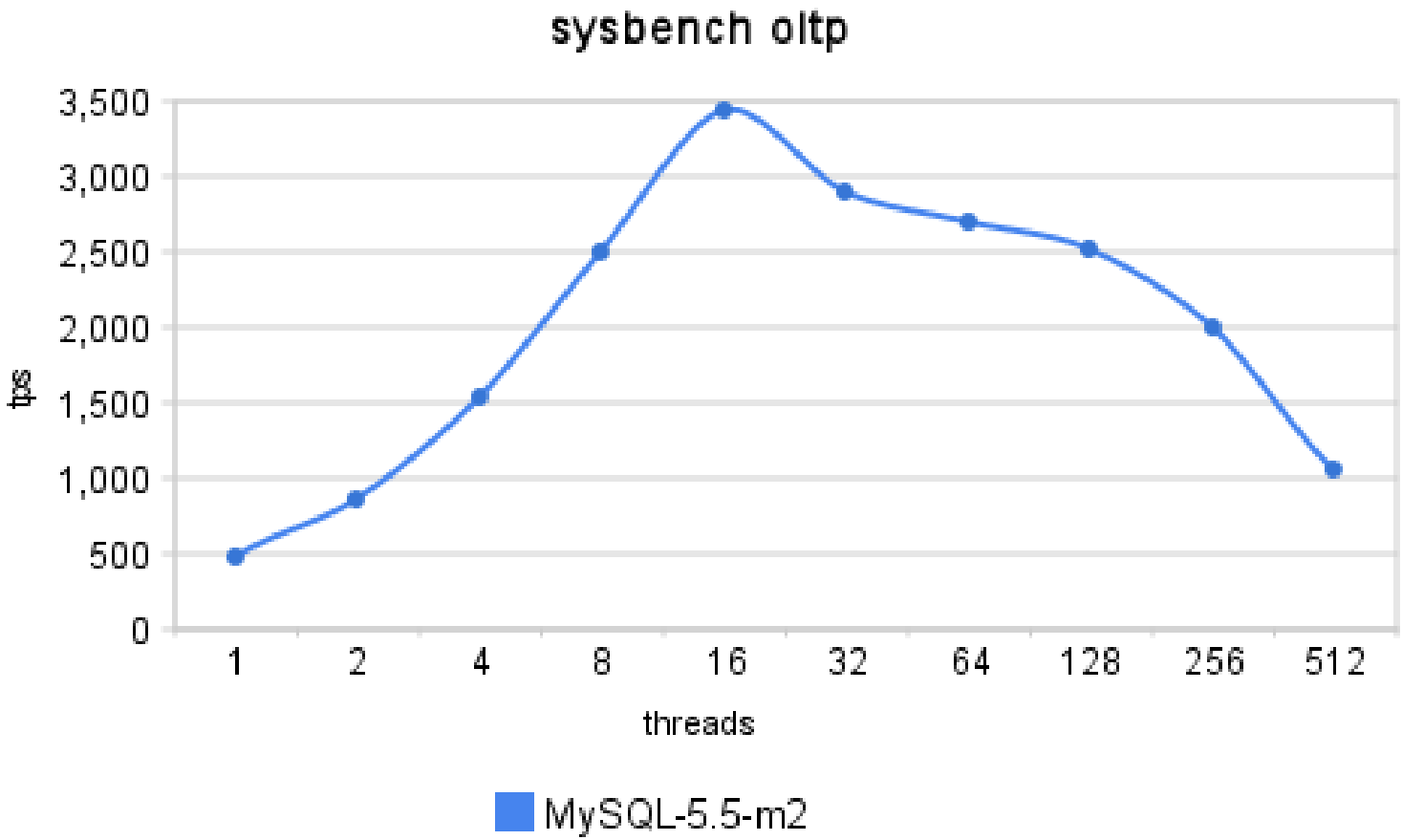
张宴

Http://blog.s135.com

传统MySQL数据库的性能问题

- 单表的记录数不断增加，查询效率降低。
- Web 2.0应用的写操作越来越多，传统的MySQL一主库多从库、读写分离模式作用有限。
- 随着并发请求数增长，MySQL性能急剧下降。

MySQL 5.5 m2数据库压力测试



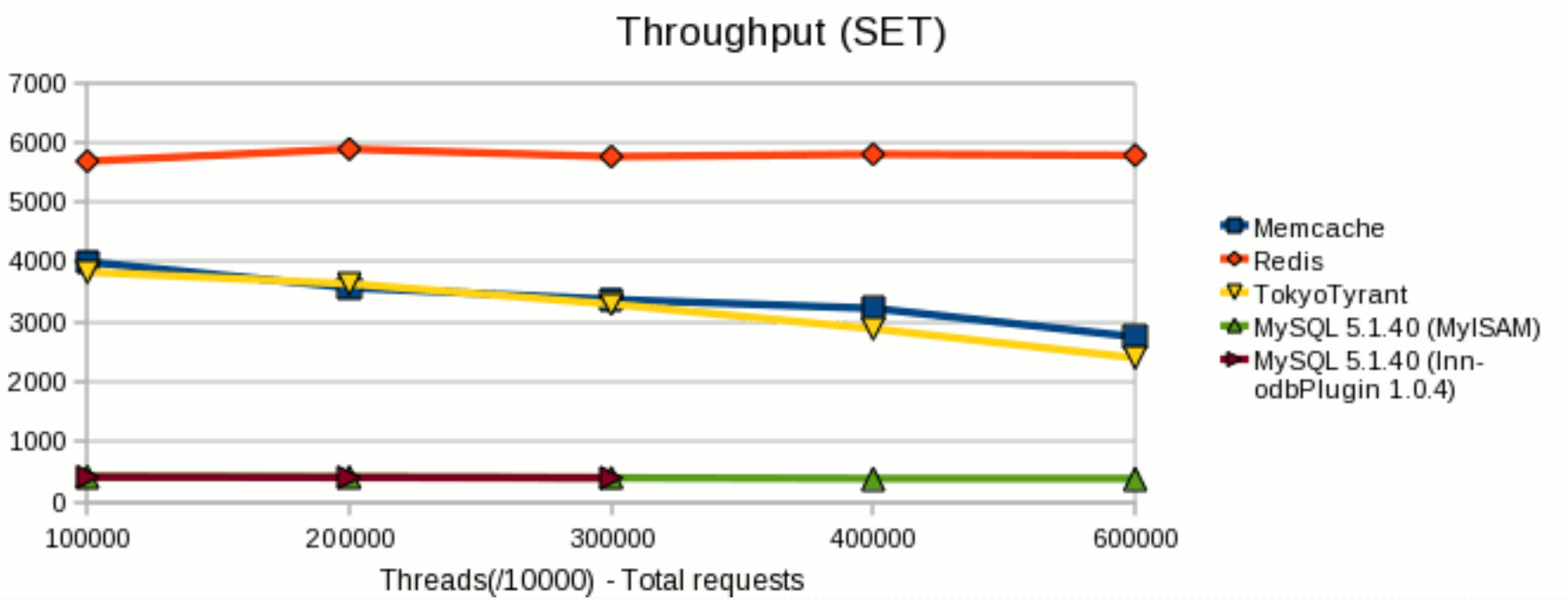
Dell R900 system (16 cores, 32GB of RAM, FusionIO + RAID10)

Tokyo Cabinet及Tokyo Tyrant 简介

- TC（Tokyo Cabinet ）是日本人 平林幹雄 开发的一款 Key-Value 键值数据库，该数据库读写非常快，哈希模式写入100万条数据只需0.402秒，读取100万条数据只需0.334秒。
- TT（Tokyo Tyrant ）是由同一作者开发的 Tokyo Cabinet 数据库网络接口。它拥有自己的协议，并支持Memcached兼容协议，也可以通过HTTP协议进行数据交换。哈希数据库读写速度大约在50000次/秒。
- TC和TT目前运行在日本最大的SNS网站MIXI，在国内也有大量的生产环境应用。

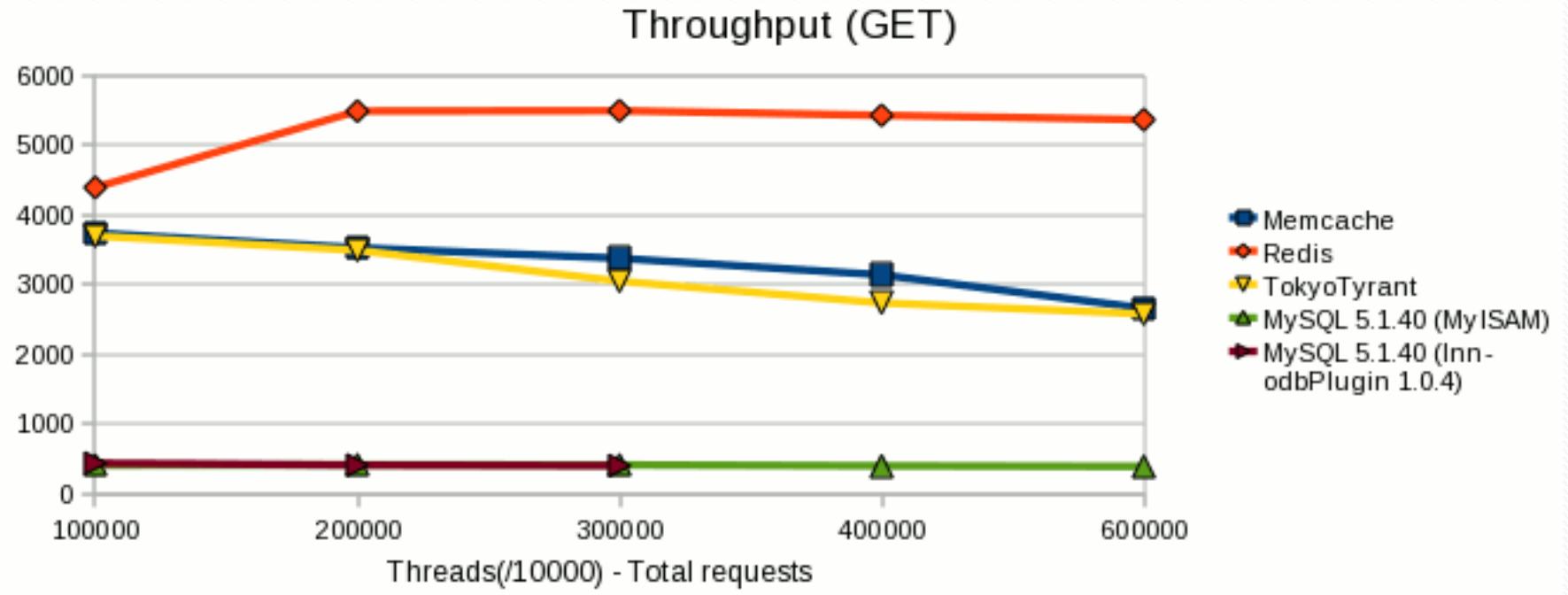
TT、Memcached、MySQL比较

- 10000线程时，写入性能



TT、Memcached、MySQL比较

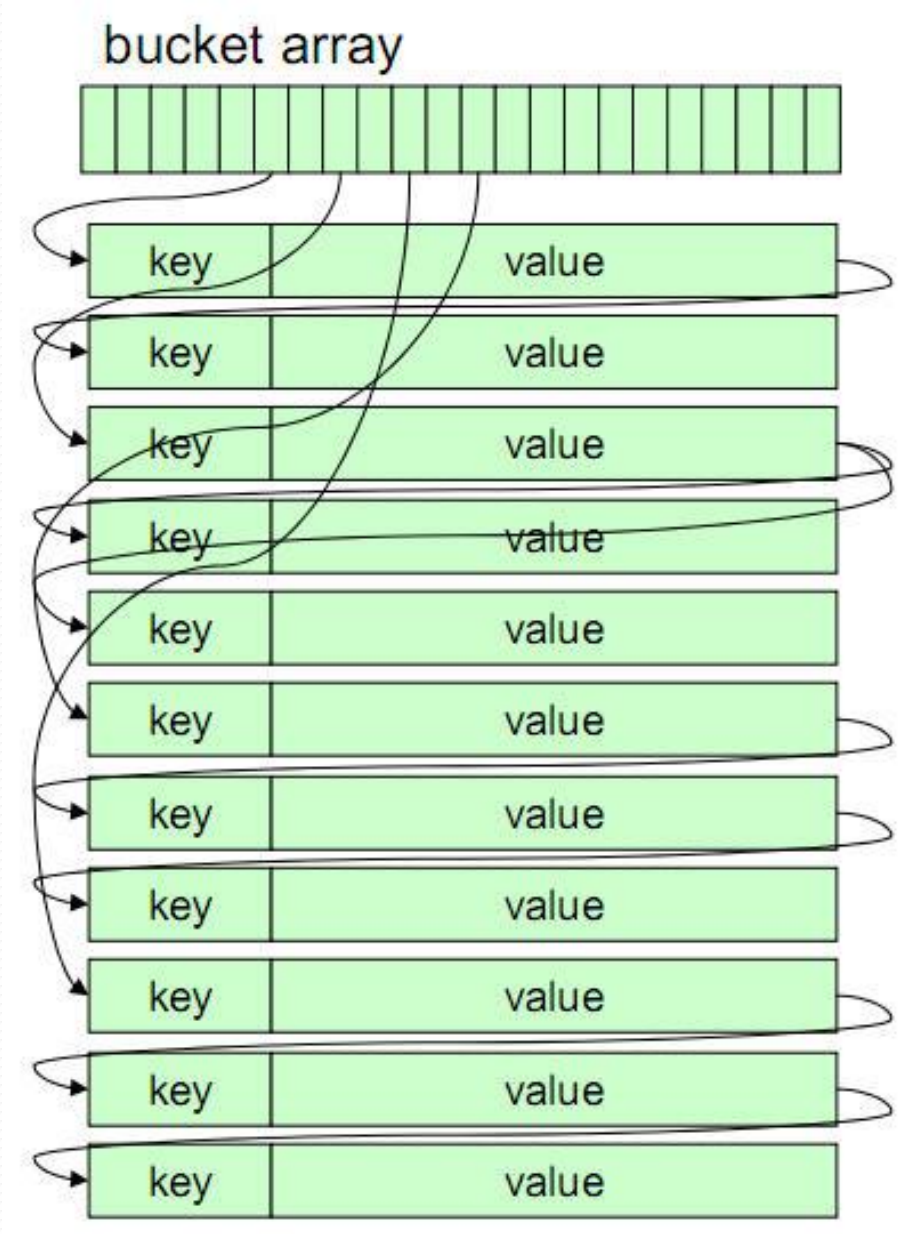
- 10000线程时，读取性能



Tokyo Cabinet的数据库类型

- TCHDB 哈希数据库
- TCBDB B+Tree数据库
- TCFDB 定长数据库
- TCTDB 表格数据库
- TCMDB 内存哈希数据库
- TCNDB 内存B+Tree数据库

TCHDB


bucket array
key
value
key
value
key
value
key
value
key
value
key
value
key
value
key
value
key
value
key
value

TCHDB哈希数据库的优化

- 很多人反应TT/TC插入数据超过一定数量后，性能会大幅度下降？
- 先对TC做个测试：

- 写入100万条：tchtest write test.tch 1000000
- 时间: 0.732秒 速度：1366120条/秒

- 写入200万条：tchtest write test.tch 2000000
- 时间: 1.718秒 速度：1164144条/秒

- 写入500万条：tchtest write test.tch 5000000
- 时间: 21.529秒 速度：232244条/秒

- 从测试来看，写入500万条数据，性能确实降低。但是，原因呢？

TCHDB哈希数据库的优化

- **修改参数后，再测试：**

- 写入100万条： tchtest write -xm 536870912 test.tch 1000000 5000000
- 时间: 0.580秒 速度： 1724137条/秒

- 写入200万条： tchtest write -xm 536870912 test.tch 2000000 5000000
- 时间: 1.105秒 速度： 1809954条/秒

- 写入500万条： tchtest write -xm 536870912 test.tch 5000000 5000000
- 时间: 2.737秒 速度： 1826817条/秒

- 可见，性能提升了不少，随着写入数据量地增加，速度依旧不减。

TCHDB哈希数据库的优化

- **关键参数（C API）：**
- bool tchdbsetxmsiz(TCHDB *hdb, int64_t xmsiz);
- Xmsiz指定了TCHDB的扩展MMAP内存大小，默认值为67108864，也就是64M，如果数据库文件超过64M，则只有前部分会映射在内存中，所以写入性能会下降。

- **其他参数（C API）：**
- bool tchdbtune(TCHDB **hdb*, int64_t *bnum*, int8_t *apow*, int8_t *fpow*, uint8_t *opts*);
- bnum指定了 bucket array的数量。推荐设置bnum为预计存储总记录数的0.5~4倍，使key的哈希分布更均匀，减少在bucket内二分查找的时间复杂度。

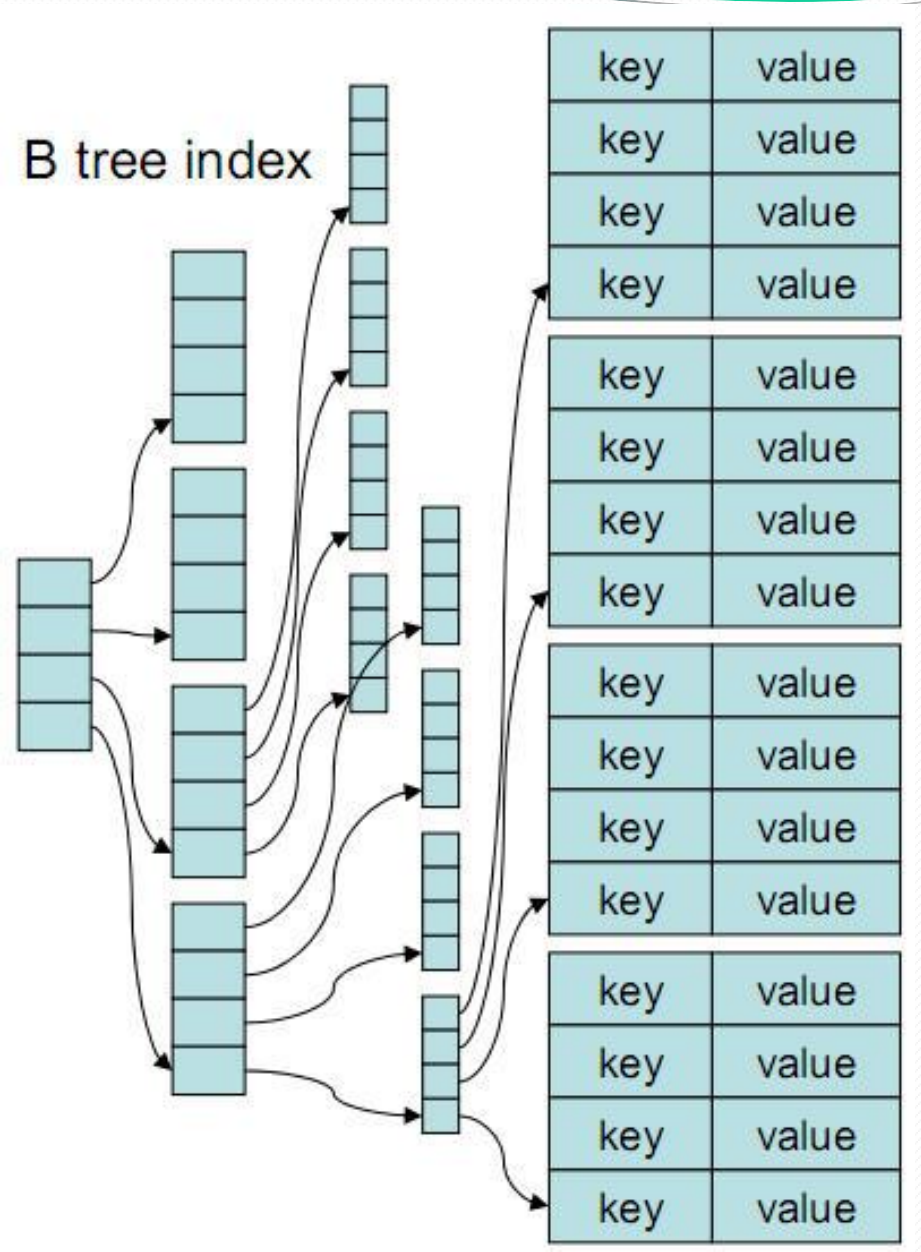
TCHDB哈希数据库的优化

- 如果存储的.tch“数据库文件大小”<=“MMAP内存大小”，TCHDB哈希数据库是个不错的选择。

- TCHDB查找速度非常快，但是对内存的要求相对较高。

- 实例：Tokyo Tyrant 的哈希数据库，优化参数如下，bnum设置为2000万条，xmsiz设置为1GB：
- ttserver -host 10.19.1.195 -port 11211 -thnum 4 -dmn -pid /data0/ttserver/ttserver.pid -log /data0/ttserver/ttserver.log -le -ulog /data0/ttserver/ -ulim 128m -sid 195 -rts /data0/ttserver/ttserver.rts /data0/ttserver/database.tch#bnum=20000000#xmsiz=1073741824

TCBDB



TCBDB B+Tree数据库的优化

- 做个测试：

- 写入100万条：tcbtest write test.tcb 1000000
- 时间: 0.994秒　　速度：1006036条/秒

- 写入200万条：tcbtest write test.tcb 2000000
- 时间: 2.028秒　　速度：986193条/秒

- 写入500万条：tcbtest write test.tcb 5000000
- 时间: 5.276秒　　速度：947687条/秒

- 从测试来看，TCBDB写入速度虽然比TCHDB低45%左右，但可以看出，写入500万条数据时，B+Tree的写入速度保持稳定。

TCBDB B+Tree数据库的优化

- **修改参数后，再测试：**

- 写入100万条： tcbtest write test.tcb 1000000 1024 2048 5000000
- 时间: 0.981秒　　速度： 1019367条/秒

- 写入200万条： tcbtest write test.tcb 2000000 1024 2048 5000000
- 时间: 1.856秒　　速度： 1077586条/秒

- 写入500万条： tcbtest write test.tcb 5000000 1024 2048 5000000
- 时间: 4.448秒　　速度： 1124101条/秒

- TCBDB没有开启xmsiz扩展MMAP内存，写入大量数据时，速度仍然能够得以保证。

TCBDB B+Tree数据库的优化

- **性能微调参数（C API）：**
- bool tcbdbtune(TCBDB *bdb, int32_t lmemb, int32_t nmemb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts);

- lmemb 用于指定被缓存的页级节点数
- nmemb 用于指定被缓存的非页级节点数，通常为页级节点数的两倍
- bnum 用于指定bucket array的数量。bnum的数量应该大于存储总记录数的1/128。

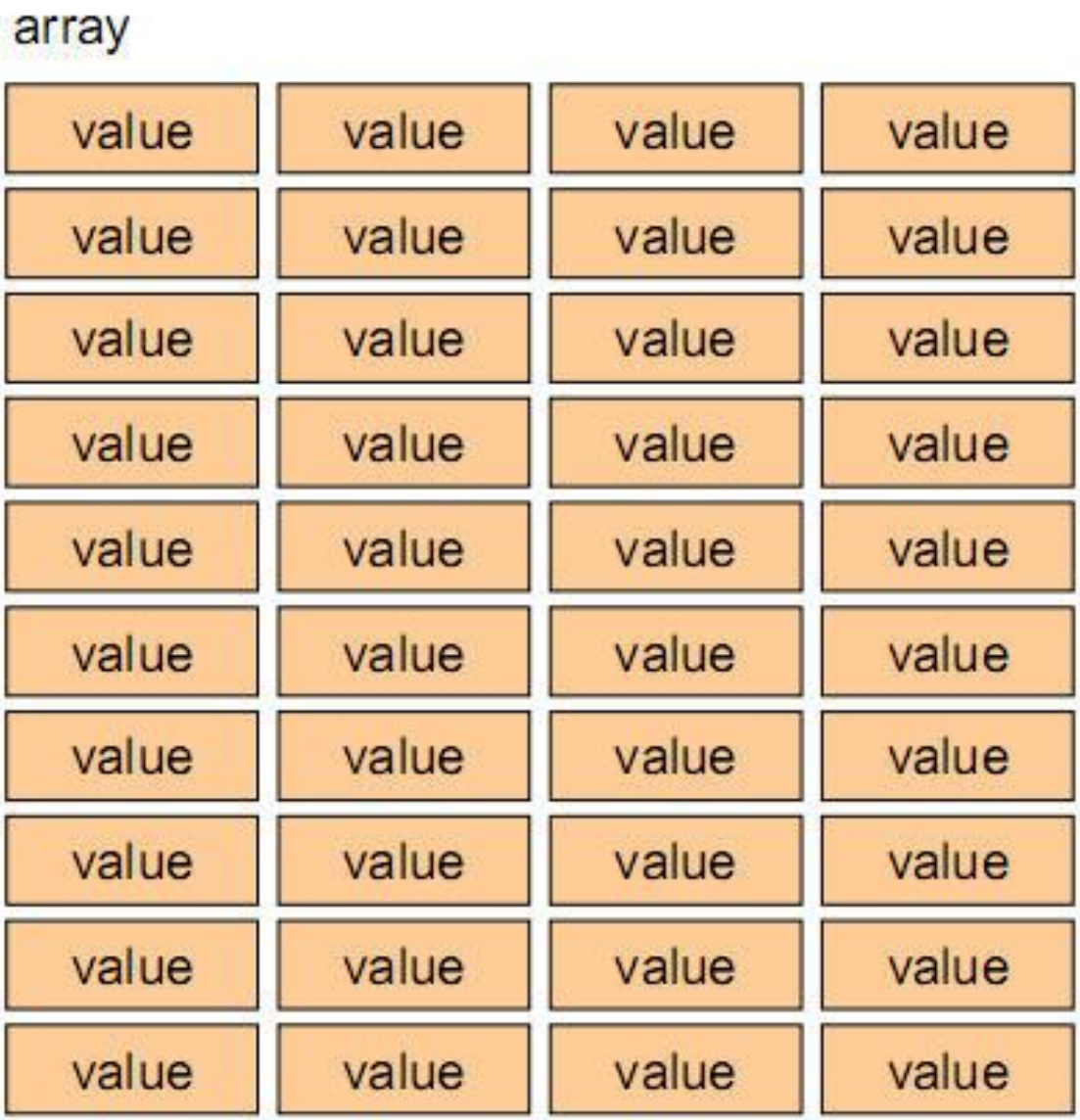
TCBDB B+Tree数据库的优化

- TCBDB(查找的时间复杂度为“O(log n)”)虽然速度比TCHDB(查找的时间复杂度为“O(1)”)慢一些，但是，它对内存的依赖度要比TCHDB小得多，随着数据量增大，写入速度也稳定，适合存储大量数据。

- 生产环境，我们存储7GB的数据， TCBDB只使用了160M的内存缓存，而TCHDB则需要设置7GB xmsiz内存才能保证速度。

- 实例：Tokyo Tyrant 的B+Tree数据库，优化参数如下：
- ttserver -host 10.19.1.195 -port 11211 -thnum 4 -dmn -pid /data0/ttserver/ttserver.pid -log /data0/ttserver/ttserver.log -le -ulog /data0/ttserver/ -ulim 128m -sid 195 -rts /data0/ttserver/ttserver.rts /data0/ttserver/database.tcb#lmemb=1024#nmemb=2048#bnum=20000000

TCFDB

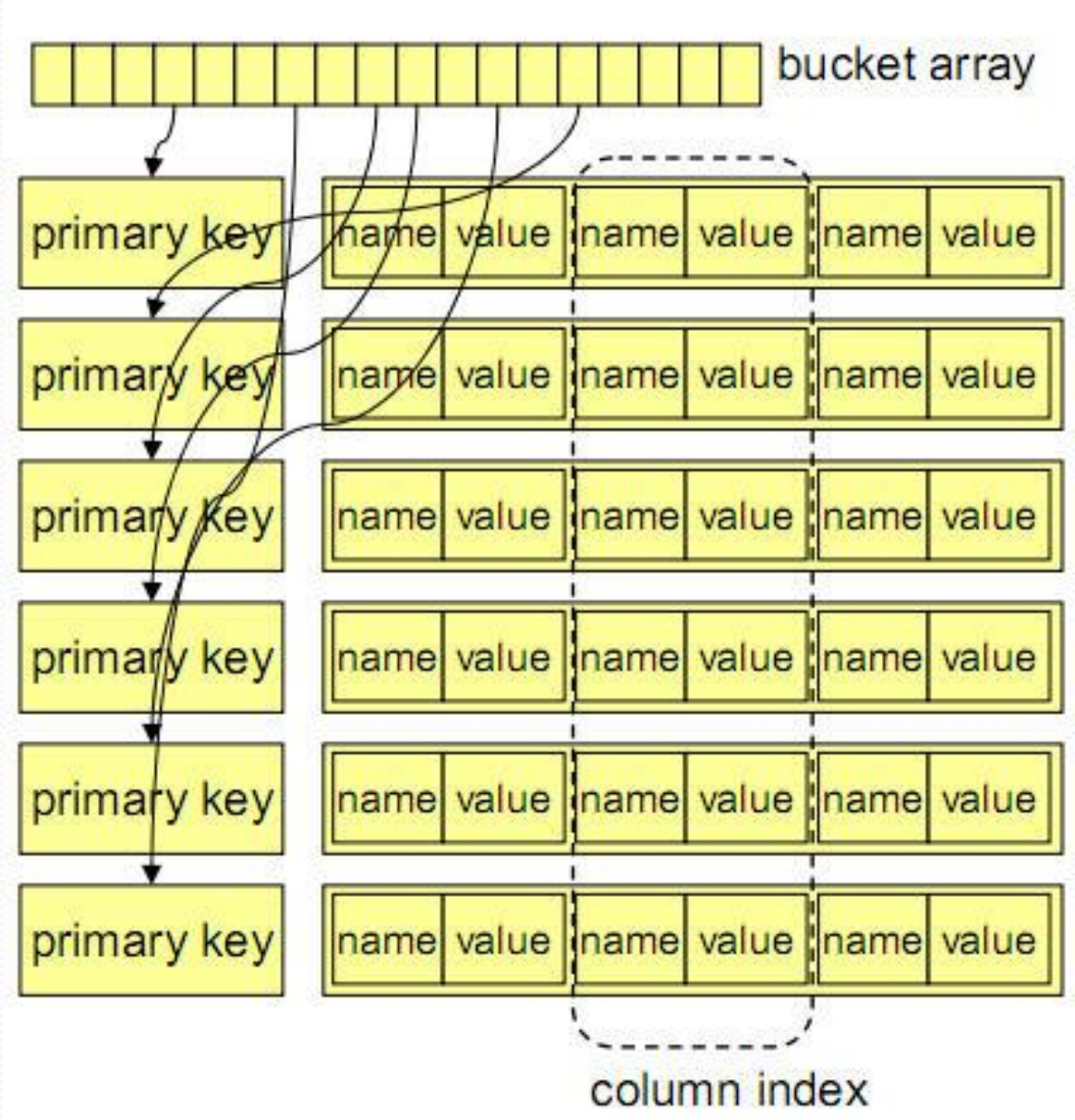

array
value value value value
value value value value
value value value value
value value value value
value value value value
value value value value
value value value value
value value value value
value value value value
value value value value

TCFDB定长数据库的优化

- **参数设置（C API）：**
- bool tcfdbtune(TCFDB *fdb, int32_t width, int64_t limsiz);
- width用于设置每条记录的长度(字节数)；
- limsiz用于设置总长度(字节数)，存储的总数据字节数不能超过limsiz设定的值。

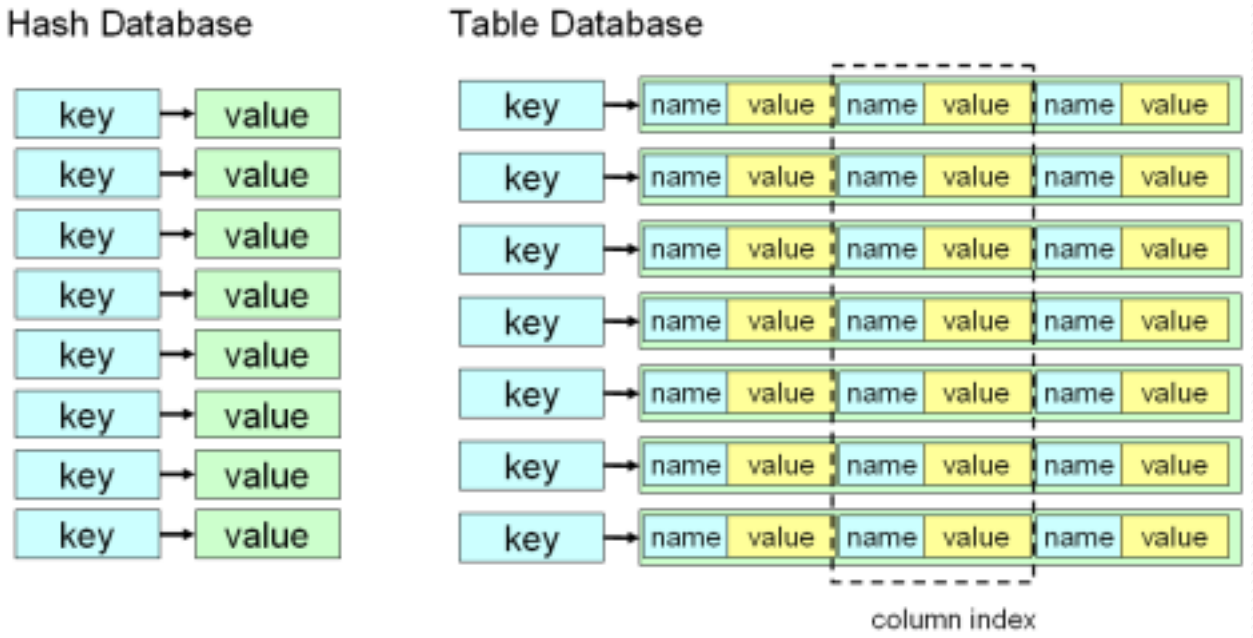
- TCFDB定长数据库读写速度快，但是存储的数据量有限。数据库文件必须在内存中完全映射。

TCTDB



TCTDB表格型数据库

- TCTDB是在 TCHDB哈希数据库的基础上，对 value 部分做的增强，因此，性能优化参数跟TCHDB类似。
- TCTDB中，key相当于关系型数据库中的主键ID，name相当于字段名，value相当于字段值。



TCTDB表格型数据库的特征

- **1、可以根据“字段”检索出符合条件的key(C API)：**
- void tctdbqryaddcond(TDBQRY *qry, const char *name, int op, const char *expr);
- Name即字段名，op代表操作类型，expr为操作对象。
- Op操作类型可以分为两类：字符型运算和数值型运算。

TCTDB表格型数据库的特征

- **数值型运算符：**
- NUMEQ：表示等于操作对象的数值（=）。
- NUMGT：表示比操作对象的数值要大（>）。
- NUMGE：表大于或等于操作对象的数值（>=）。
- NUMLT：表示比操作对象的数值要小（<）。
- NUMLE：表示小于或等于操作对象的数值（<=）。
- NUMBT：表示其大小处于操作对象文字段中被逗号分开的两个数值的中间（between 100 and 200）。
- NUMOREQ：表示同操作对象文字段中被逗号分开的多个数值中的其中一个是相同的（ IN (100,200,278)）。

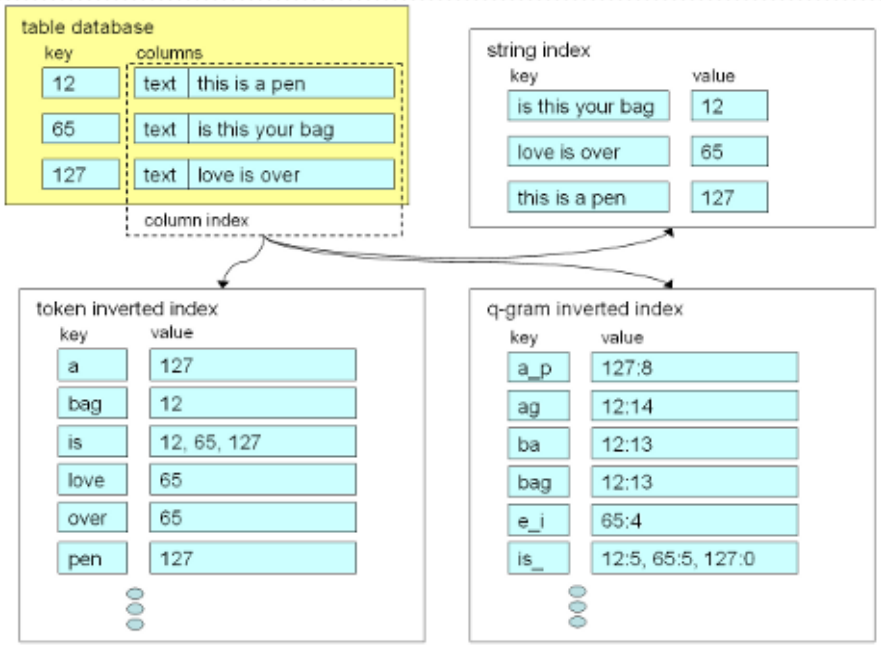
TCTDB表格型数据库的特征

- **文本型运算符：**
- STREQ：表示与操作对象的文字内容完全相同（=）。
- STRINC：表示含有操作对象文字的内容（LIKE ‘%文字%’）。
- STRBW：表示以操作对象的文字行列开始（LIKE ‘文字%’）。
- STREW：表示到操作对象的文字行列结束（LIKE ‘%文字’）。
- STRAND：表示包含操作对象的文字行列中右逗号分开部分的字段的全部（name LIKE ‘%文字㈠%’ AND name LIKE ‘%文字㈡%’）。
- STROR：表示包含操作对象文字段中逗号分开部分的其中一部分（name LIKE ‘%文字㈠%’ OR name LIKE ‘%文字㈡%’）。
- STROREQ：表示与操作对象文字段中逗号分开部分的其中某部分完全相同（ name = ‘文字㈠’ OR name =‘文字㈡’ ）。

TCTDB表格型数据库的特征

- **2、可以对“字段”建立索引，支持UTF-8全文检索(C API)：**
- bool tctdbsetindex(TCTDB *tdb, const char *name, int type);

- **type为索引类型，值如下：**
- TDBITLEXICAL：创建文本型索引
- TDBITDECIMAL：创建数值型索引
- TDBITTOKEN：创建标记倒排索引
- TDBITQGRAM：创建q-gram倒排索引
- TDBITOPT：优化索引
- TDBITVOID：删除索引



TCTDB表格型数据库的特征

- **3、支持按“字段”排序(C API)：**
- void tctdbqrysetorder(TDBQRY *qry, const char *name, int type);

- **type为排序类型，值如下：**
- STRASC：表示按照文本型字段内的文本内容在字典中排列顺序的升序。
- STRDESC：表示按照文本型字段内的文本内容在字典中排列顺序的降序。
- NUMASC：表示按照数值大小的升序。
- NUMDESC：表示按照数值大小的降序。

TCTDB表格型数据库的特征

- **4、支持检索结果数量限制(C API)：**
- void tctdbqrysetlimit(TDBQRY *qry, int max, int skip);

- 相当于SQL语句中的“limit skip, max”

TCTDB表格型数据库的特征

- 回顾TCTDB数据库的特征：

- TCTDB即具备了Key-Value数据库的高效读写性能，又具备了MySQL单表能实现的一些功能，即：
- SELECT …. FROM table WHERE …. ORDER BY …. LIMIT xxx,xxx

TCTDB表格型数据库的不足之处

- 1、功能的增强，也就意味着要牺牲一些性能。TCTDB表格型数据库的平均读取速度大约在40万条/秒，相比TCHDB哈希数据库的180万条/秒和TCBDB B+Tree数据库的100万条/秒要慢一些。
- 2、TCTDB虽然可以建立数值型索引，但是它是将所有value数据都当成字符型来处理的，无法区分value类型。
- 3、TCTDB单数据库文件存储的记录数上亿条后，性能会有比较明显的下降。
- 4、没有可扩展的能力，如果单机无法满足要求，只能通过主从复制的方式扩展。
- 5、UPDATE更新方式效率低。

金山逍遥TCSQL列表缓存数据库

- TCSQL是金山逍遥网在Tokyo Cabinet TCTDB的基础上，结合Key-Value对象缓存，借鉴SQL语句的SELECT、INSERT、UPDATE、DELETE思想与功能开发的分布式实时列表缓存数据库，可实现对列表页数据、记录条数的实时缓存。

- TCSQL采用HTTP GET/POST协议＋JSON数据交换格式在客户端、服务器端之间进行数据交互。

- TCSQL实时列表缓存数据库单机能够支撑1万以上的并发连接。1万并发连接下，QPS（每秒查询率）能够达到8000～15000次。

- TCSQL拥有了MySQL数据库单表具备的大部分功能。

TCSQL的select查询示例

- curl “http://127.0.0.1:3888/?command=select&type=*&where=pkey:NUMGE:0|title:STRINC:keyword&order_by=pkey&order_sort=NUMDESC&limit_skip=0&limit_max=10”

- 等同于MySQL的SQL查询语句：
- SELECT * FROM table WHERE pkey >= 0 AND title LIKE ‘%keyword%’ ORDER BY pkey DESC LIMIT 0,10;

- 跟MySQL的LIKE全表扫描不同的是，TCSQL支持创建倒排索引，STRINC使用的是全文检索。

通过PHP Client操作TCSQL

- <?php
- $tcsql = new Tcsql();
- //INSERT插入
- $result = $tcsql->insert($server_host, $server_port, $array_data);
- //SELECT查询
- $result = $tcsql->select($server_host, $server_port, "type=*&where=uid:NUMOREQ:1,7,29,43&order_by=pkey&order_by=NUMDESC&limit_skip=0&limit_max=10“);
- //UPDATE更新
- $result = $tcsql->update($server_host, $server_port, $array_data, “where=name:STREQ:张三|sex:NUMEQ:1“);
- //DELETE删除
- $result = $tcsql->delete($server_host, $server_port, $array_data, "where=uid:NUMEQ:5");
- ?>

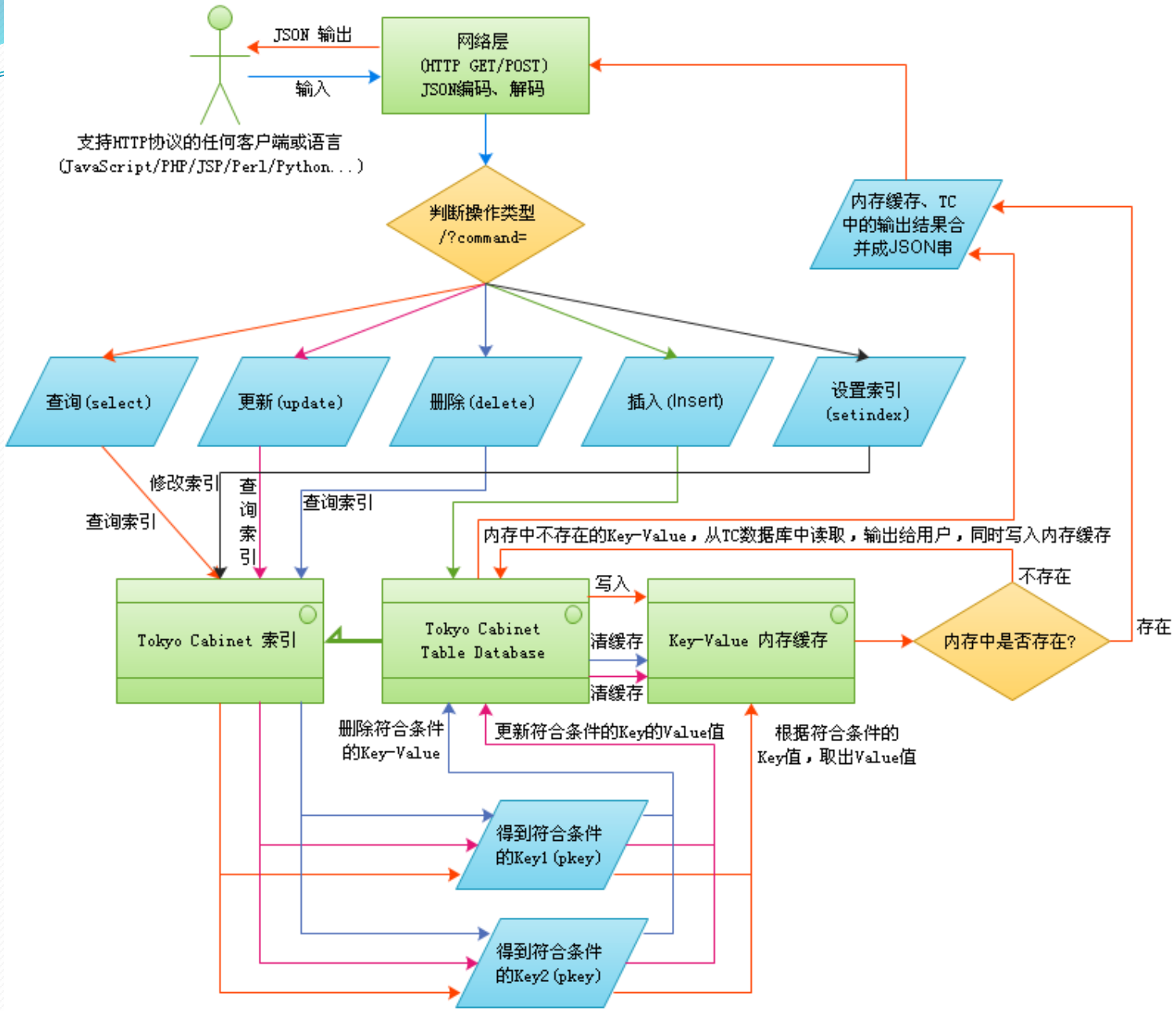
TCSQL的新特征

- 相对于原生的TCTDB，TCSQL增加了不少新特征，来解决性能、存储量、功能上的问题。

TCSQL的内存缓存加速功能

- 1、刚才已经提到，TCTDB表格型数据库的读取性能相对于TCHDB哈希数据库、TCBDB B+Tree数据库而言，要低许多。

- 2、TCSQL采用内存热点记录缓存、TCBDB B+Tree数据库永久缓存相结合的方式，将TCTDB的读取性能提升了两倍。TCTDB中的一条Key-Value数据只要一次被读取，就会永久缓存，直到更新、删除该记录时，才会删除缓存内容。

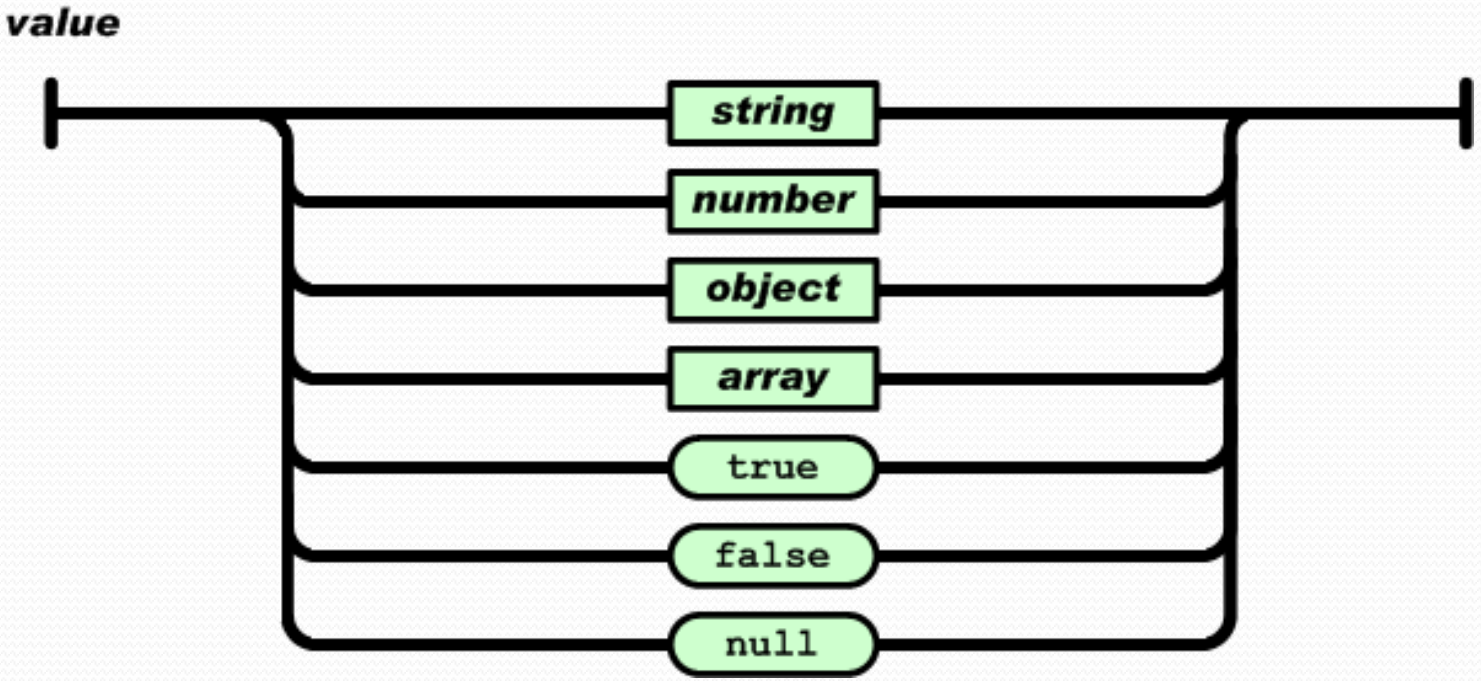

JSON 输出
输入
网络层
(HTTP GET/POST)
JSON编码、解码
支持HTTP协议的任何客户端或语言
(JavaScript/PHP/JSP/Perl/Python...)
判断操作类型
/?command=
内存缓存、TC
中的输出结果合
并成JSON串
查询(select)
更新(update)
删除(delete)
插入(Insert)
设置索引
(setindex)
修改索引
查询索引
查询索引
查询索引
内存中不存在的Key-Value，从TC数据库中读取，输出给用户，同时写入内存缓存
不存在
存在
写入
清缓存
清缓存
Tokyo Cabinet 索引
Tokyo Cabinet
Table Database
Key-Value 内存缓存
内存中是否存在?
删除符合条件
的Key-Value
更新符合条件的Key的Value值
根据符合条件的
Key值，取出Value值
得到符合条件
的Key1(pkey)
得到符合条件
的Key2(pkey)

TCSQL的内存缓存加速功能

- TCSQL的内存缓存、TCBDB B+Tree数据库联合缓存机制命中率非常高，比如按照时间倒叙显示50条记录，第一次查询就会将记录数据缓存在内存中，如果第二次查询时已经新增了3条记录（或者修改了3条记录），第二次查询只会从TCTDB表格型数据库中取出3条记录，剩余47条从内存缓存或TCBDB中取出，大大加快了读取速度。

TCSQL数据库支持的数据类型

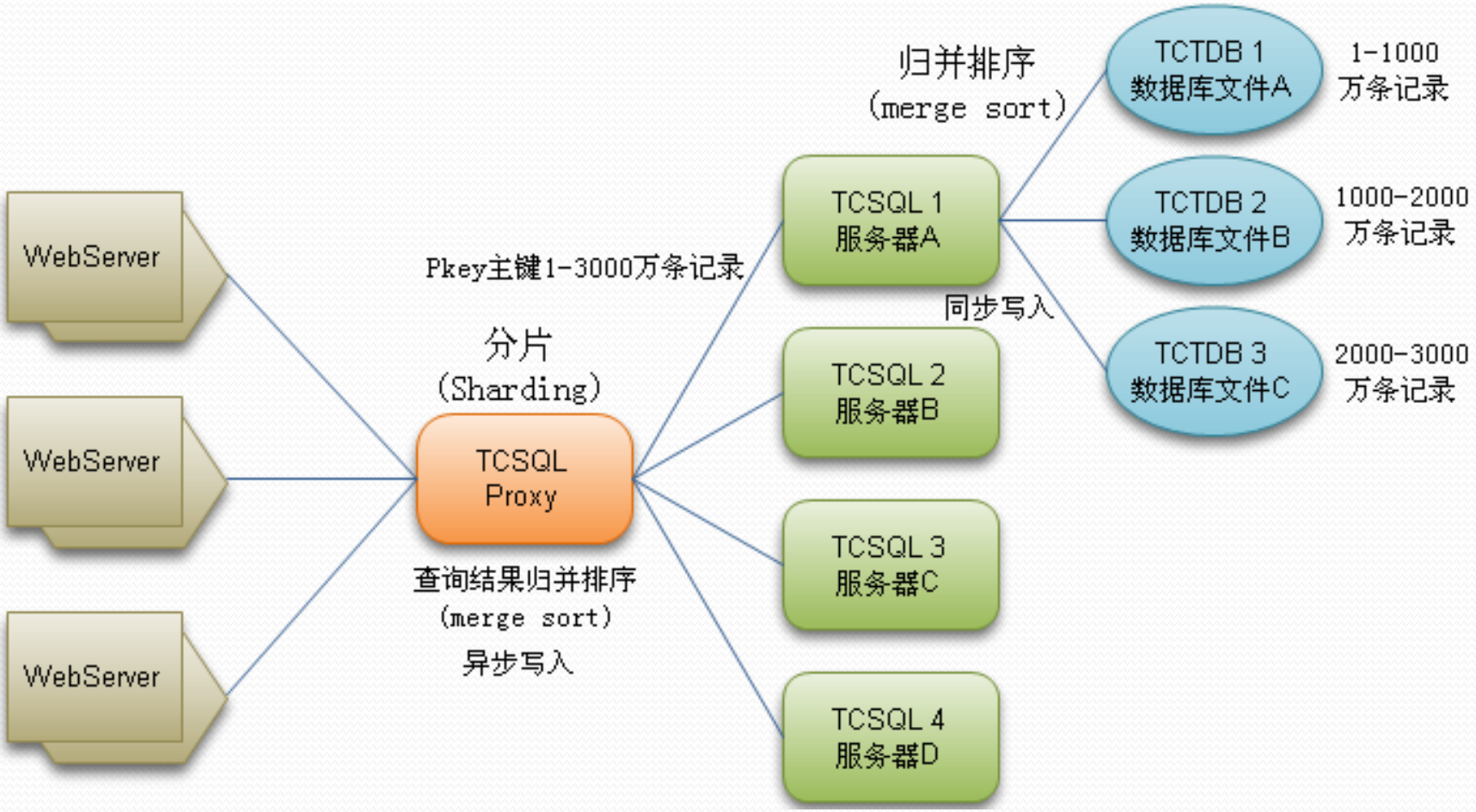
- 1、针对TCTDB无法区分value类型的问题，TCSQL增加了对字段设置数据类型的功能，它支持JSON所拥有的6种数据类型：



TCSQL数据库支持的数据类型

- 一个PHP的数组，经过json_encode()后，写入TCSQL后再读取出来，数组类型仍然能够保持原样不变：
- <?php
- $array_data['uid'] = 1234;
- $array_data['ip'] = '192.168.138.1';
- $array_data['passport'] = 'kingsoft';
- $array_data[‘nickname’] = ‘测试账号';
- $array_data[‘num1'] = 5996273.9334;
- $array_data[‘num2'] = -123456789012345;
- $array_data['u1'] = True;
- $array_data['u2'] = array("aaa", "bbb", "ccc");
- $array_data['u3'] = null;
- $array_data['u4'][‘test'] = array("aaa", "bbb", "ccc");
- $json_data = json_encode($array_data); //准备写入到TCSQL的数据
- ……
- ?>

TCSQL的分片、分布式并行查询



TCSQL的分片、分布式并行查询

- 1、TCSQL采用分片、并行计算的结构，解决了TCTDB单数据库文件存储的记录数上亿条后，性能会明显下降的不足之处。
- 2、TCSQL增强了TCTDB的可扩展能力。
- 3、适合解决Web 2.0应用中单表数据量无限增大的问题。

TCSQL的数据update更新优化

- TCTDB＋Tokyo Tyrant时，如果只更新一条记录中的某个字段的值，PHP客户端需要先读回Tokyo Tyrant中该记录的全部字段内容到一个PHP数组，再修改此数组中该字段的值，最后再将该数组写回Tokyo Tyrant，来回通过网络传输记录全部字段内容，速度慢、效率低。

- TCSQL可以对部分字段内容进行更新，在TCSQL内部处理完成修改操作，没有不必要的TCP传输。

TCSQL可作为MySQL的“从库”

- 借助MySQL触发器，以及为MySQL 5.1 编写的扩展插件函数，来让TCSQL当MySQL的从库：
- json_object()、 urlencode()、http_post()

- 接下来是一个示例：通过MySQL命令行连接到MySQL服务器，执行以下SQL，对sns_feed表创建三个MySQL触发器：sns_feed_insert、sns_feed_update、sns_feed_delete，当MySQL的sns_feed表发生增、删、改操作时，自动将修改的记录内容通过HTTP POST到TCSQL数据库（192.168.8.34:3888)。

TCSQL“从库”Insert触发器

- DELIMITER |
- DROP TRIGGER IF EXISTS sns_feed_insert;
- CREATE TRIGGER sns_feed_insert
- AFTER INSERT ON sns_feed
- FOR EACH ROW BEGIN
- SET @tcsql_result_json = (SELECT json_object(feedid as pkey,appid,icon,uid,username,dateline,friend,hash_template,hash_data,title_template,title_data,body_template,body_data,body_general,image_1,image_1_link,image_2,image_2_link,image_3,image_3_link,image_4,image_4_link,target_ids,id,idtype,hot) FROM sns_feed WHERE feedid = NEW.feedid limit 1);
- SET @tcsql_result_eval = (SELECT http_post('192.168.8.34', '3888', 'command=insert', urlencode(@tcsql_result_json)));
- END |
- DELIMITER ;

TCSQL“从库”update触发器

- DELIMITER |
- DROP TRIGGER IF EXISTS sns_feed_update;
- CREATE TRIGGER sns_feed_update
- AFTER UPDATE ON sns_feed
- FOR EACH ROW BEGIN
- SET @tcsql_result_json = (SELECT json_object(feedid as pkey,appid,icon,uid,username,dateline,friend,hash_template,hash_data,title_template,title_data,body_template,body_data,body_general,image_1,image_1_link,image_2,image_2_link,image_3,image_3_link,image_4,image_4_link,target_ids,id,idtype,hot) FROM sns_feed WHERE feedid = OLD.feedid limit 1);
- SET @tcsql_result_eval = (SELECT http_post('192.168.8.34', '3888', 'command=insert', urlencode(@tcsql_result_json)));
- END |
- DELIMITER ;

TCSQL“从库”delete触发器

- DELIMITER |
- DROP TRIGGER IF EXISTS sns_feed_delete;
- CREATE TRIGGER sns_feed_delete
- AFTER DELETE ON sns_feed
- FOR EACH ROW BEGIN
- SET @tcsql_result_eval = (SELECT http_post('192.168.8.34', '3888', concat('command=delete&where=pkey:NUMEQ:', OLD.feedid), ''));
- END |
- DELIMITER ;

TCSQL的应用场景

- **一般数据库缓存分为四种：**
- 1、Key/Value单个对象缓存，技术不难，Memcached、Squid均能实现。
- 2、列表缓存，就像论坛里帖子的列表、SNS中的Feed信息，要求实时更新。
- 3、记录条数的缓存，比如一个论坛板块里有多少个帖子，这样才方便实现分页。
- 4、复杂一点的group，sum，count查询，比如一个论坛里按点击数排名的最HOT的帖子列表。

TCSQL的应用场景

- TCSQL主要用于解决上页提到的第2、3类应用，即带条件的列表页、列表页记录数的实时缓存。
- **TCSQL在金山逍遥网的典型应用：**
- **游戏论坛：** 原版Discuz!论坛使用的MySQL内存表存放Session，在线5万人的时候，大量的并发update操作导致内存表锁死。后改用TCSQL，轻松解决了这个问题。从MySQL改为TCSQL，迁移成本也非常低。
- **SNS社区：** Feed信息采用从MySQL同步到TCSQL的方式，利用TCSQL来解决好友动态的查询压力。
- **用户行为分析：** 需要按UID、通行证、昵称、时间等条件查询基础用户信息表，采用TCSQL来存储超过2亿条数据，每次页面访问将有一次select查询。

结束