WiredTiger实现探秘

许鹏

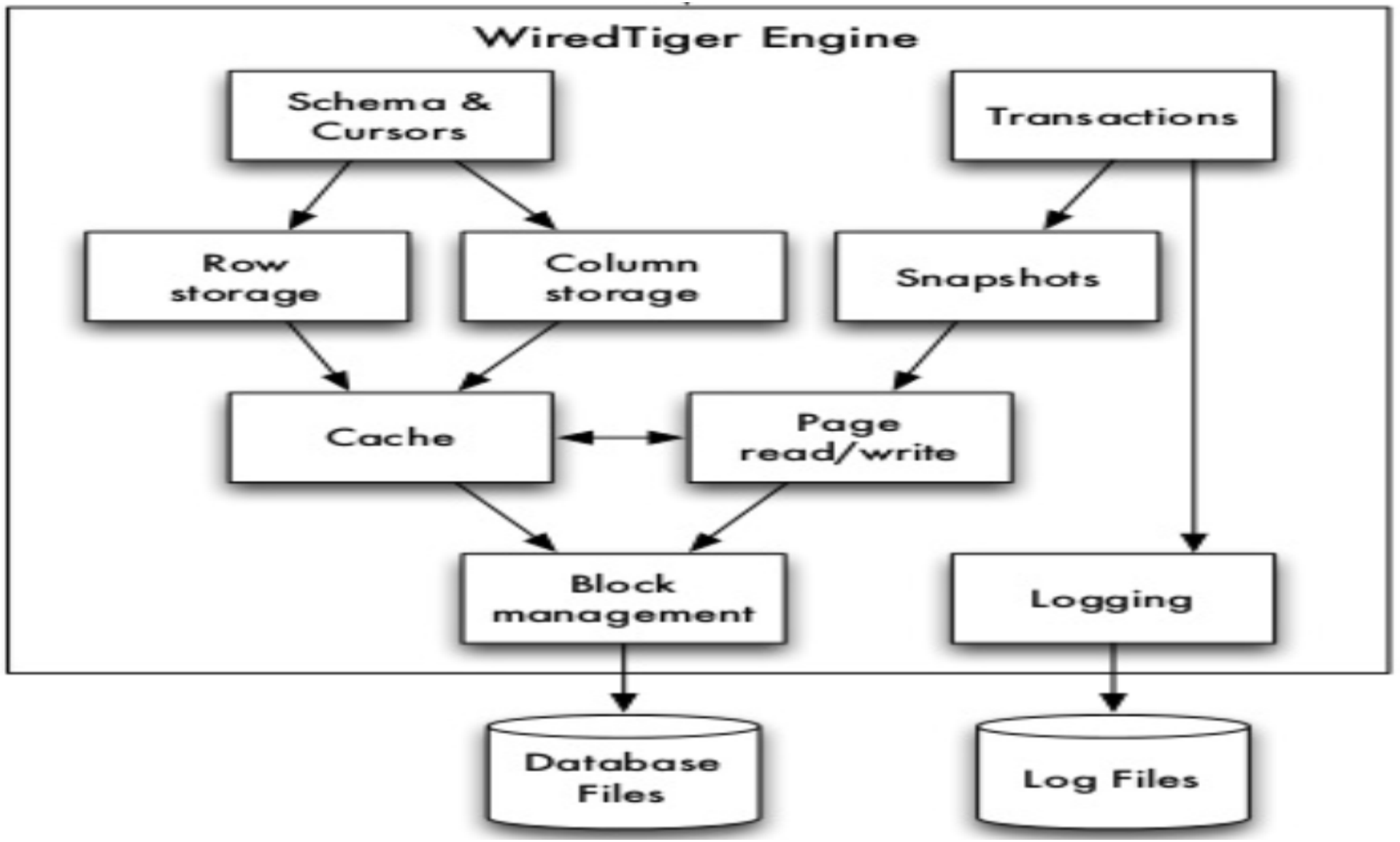
摘要

- 无锁化规避线程间竞争

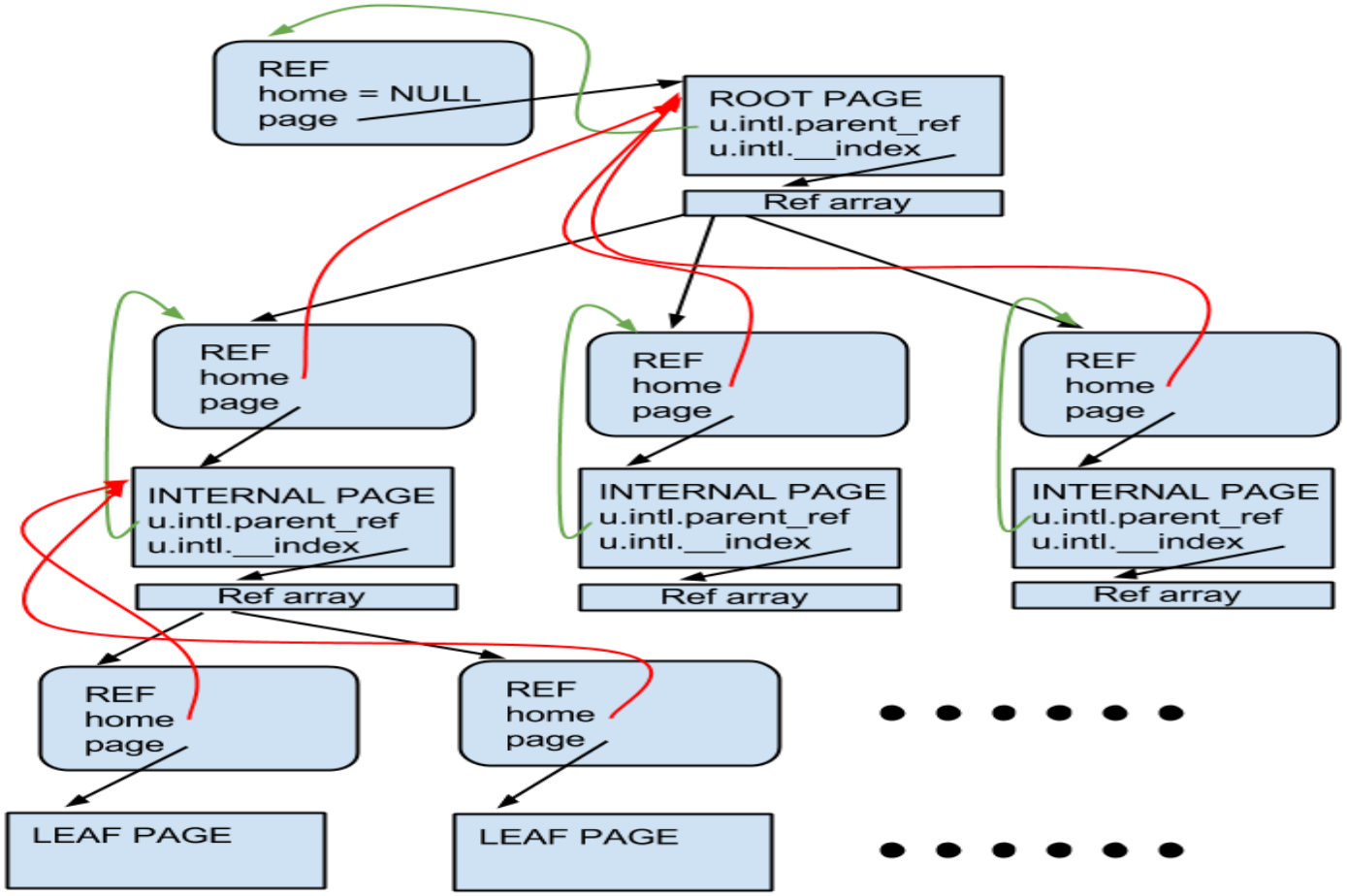
- 页面布局(memory layout)

- 缓存和文件压缩

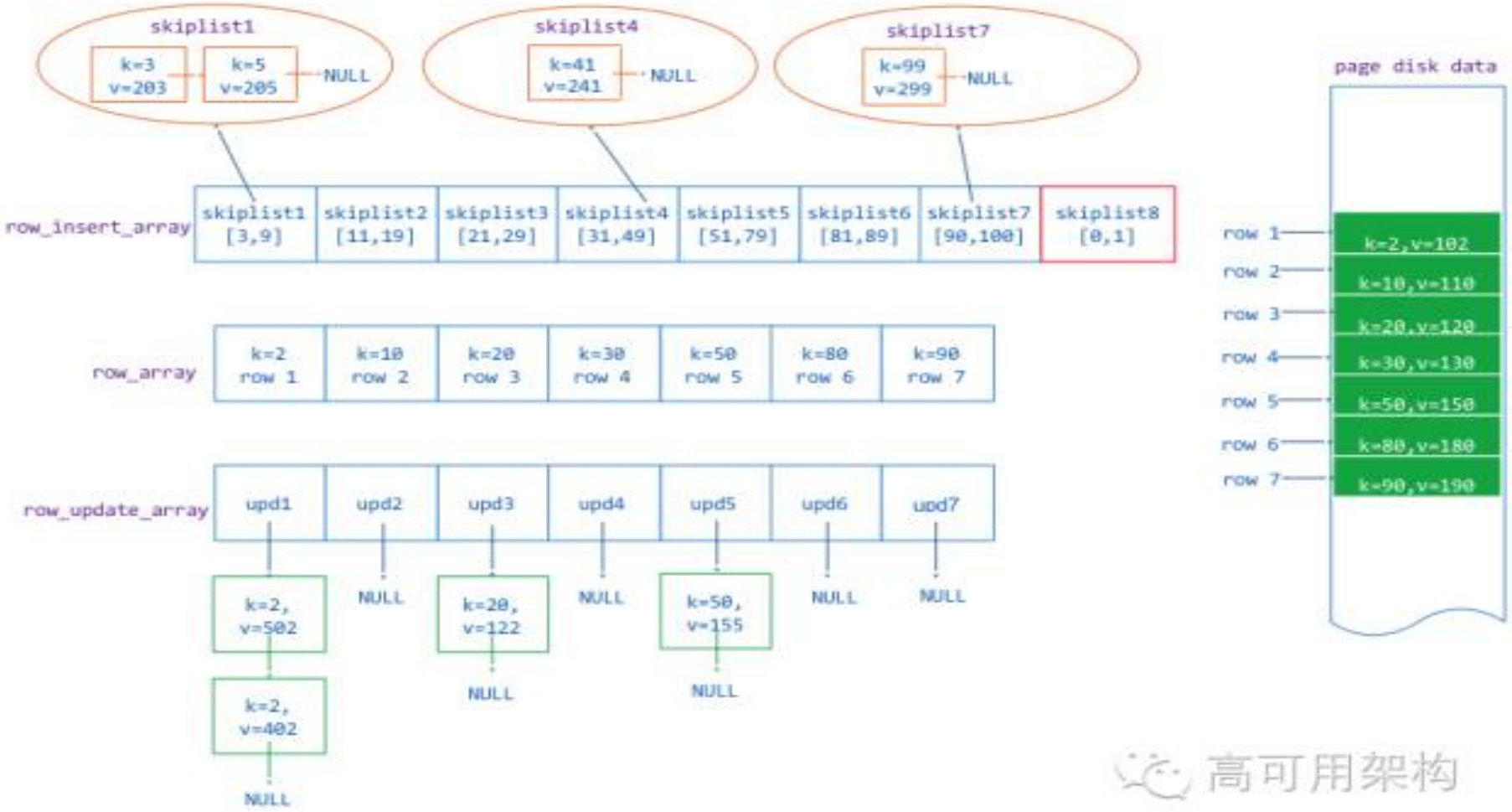
整体架构


WiredTiger Engine
Schema & Cursors
Transactions
Row storage
Column storage
Snapshots
Cache
Page read/write
Block management
Logging
Database Files
Log Files

内存管理 – 页面管理 BTree



内存管理 – 叶子结点



初始化写入 – 数据没有落入磁盘

p *((WT_CURSOR_BTREE*)cursor).btree.root.page.u.intl.__index.index[0].page

```
$97 = {
  u = {
    intl = {
      parent_ref = 0x0,
      split_gen = 0,
      __index = 0x0
    },
    row = 0x0,
    fix_bitf = 0x0,
    col_var = {
      col_var = 0x0,
      repeats = 0x0
    }
  },
  entries = 0,
  type = 7 '\a',
  flags_atomic = 0 '\000',
  unused = "\000",
  read_gen = 201,
  memory_footprint = 746,
  dsk = 0x0,
  modify = 0x6933a0,
  cache_create_gen = 1,
  evict_pass_gen = 0
}
```

p ((WT_CURSOR_BTREE*)cursor).btree.root.page.u.intl.__index.index[0].page.modify.u2

```
$99 = {
  intl = {
    root_split = 0x693080
  },
  column_leaf = {
    append = 0x693080,
    update = 0x0,
    split_recno = 0
  },
  row_leaf = {
    insert = 0x693080,
    update = 0x0
  }
}
```

Row Leaf --叶子结点的内存信息

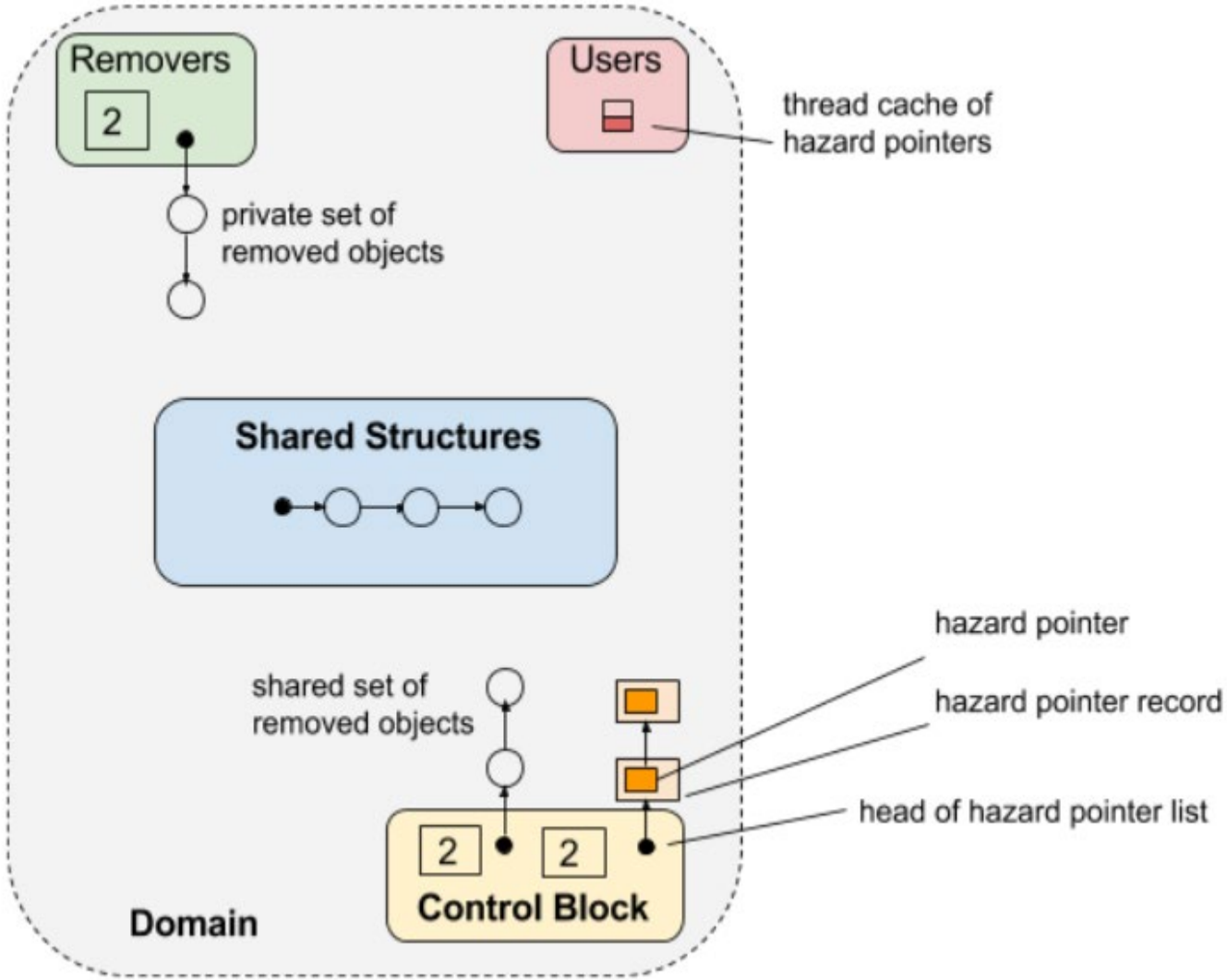
- 从磁盘加载数据
- 修改已经存在的key
- 关注如下两个成员变量
  - dsk
  - Modify
- row表示从dsk位置开始的偏移量

GDB调试指令

```
p *((WT_CURSOR_BTREE*)cursor).btree.root.page.u.intl.__index.index[0].page
```

```
$61 = {
  u = {
    intl = {
      parent_ref = 0x69fe30,
      split_gen = 0,
      __index = 0x0
    },
    row = 0x69fe30,
    fix_bitf = 0x69fe30 "\277",
    col_var = {
      col_var = 0x69fe30,
      repeats = 0x0
    }
  },
  entries = 2,
  type = 7 '\a',
  flags_atomic = 2 '\002',
  unused = "\000",
  read_gen = 201,
  memory_footprint = 555,
  dsk = 0x69edd0,
  modify = 0x69fe50,
  cache_create_gen = 1,
  evict_pass_gen = 0
}
```

页面释放 – Hazard Pointer



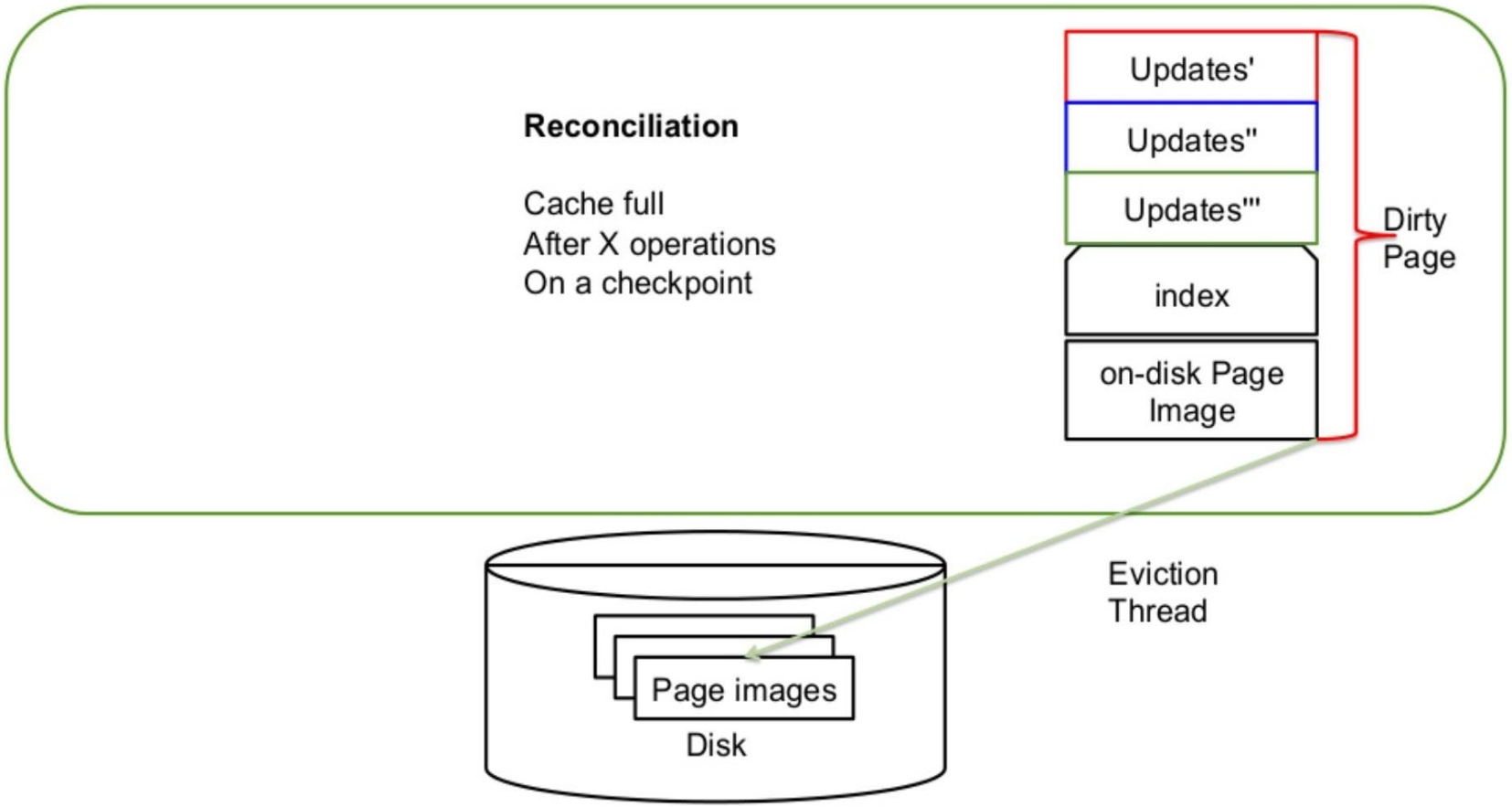
1.建立一个全局数组，数组容量为线程数目，每个线程只能修改自己的数组元素，而不允许修改其他的数组元素，但可以读别的数组元素。

2.当线程尝试访问一个关键数据节点时，先把该节点指针赋给自己的数组元素（即不要释放这个节点）。

3.每个线程自己维护一个私有链表，当线程准备释放掉某个节点时，将该节点放入到链表中。当链表内的数目达到一个设定的数目后，遍历该链表用于释放链表内所有节点。

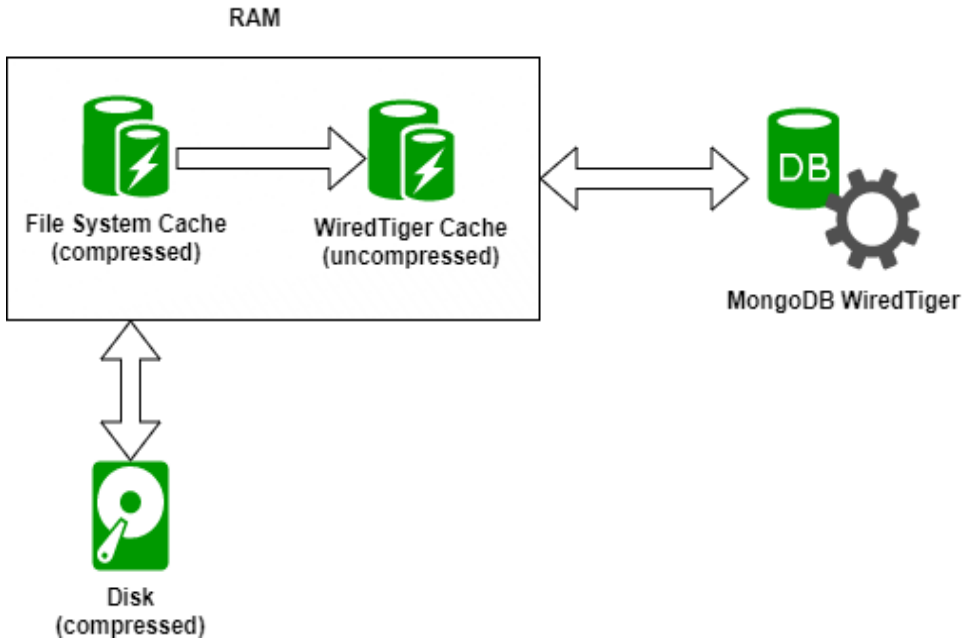
4.当释放节点时，需要检查全局数组，确定没有任何一个线程的数组元素与当前指针相同时，就释放该节点。否则仍然滞留在自己的链表中。

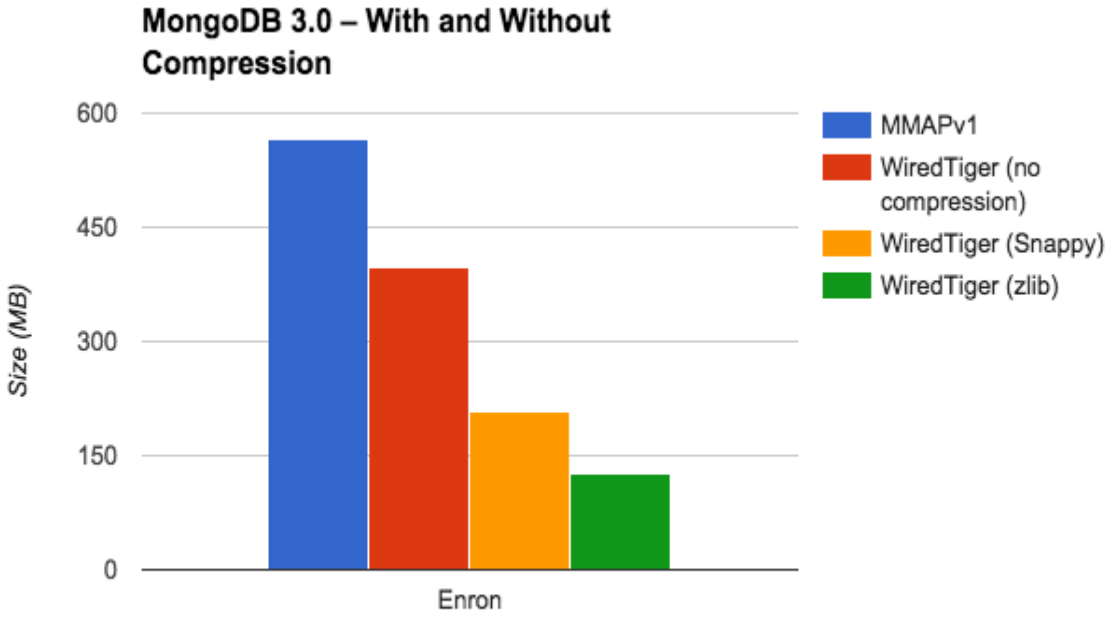
数据持久化



- Checkpoint
  - Occur at an interval of ***60 seconds***
- Journal log
  - Sync to disk every ***50 milliseconds***

文件压缩





- 默认使用Snappy压缩算法
- *缓存中是解压后的格式，占用内存空间远大于磁盘*

并发控制访问

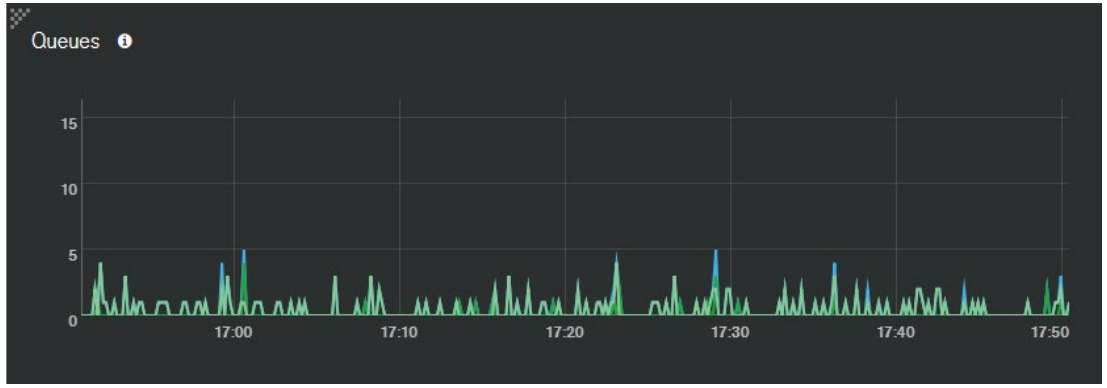

```
struct __wt_rwlock {            /* Read/write lock */
    volatile union {
        uint64_t v;          /* Full 64-bit value */
        struct {
            uint8_t current;    /* Current ticket */
            uint8_t next;       /* Next available ticket */
            uint8_t reader;     /* Read queue ticket */
            uint8_t readers_queued; /* Count of queued readers */
            uint32_t readers_active;/* Count of active readers */
        } s;
    } u;

    int16_t stat_read_count_off;    /* read acquisitions offset */
    int16_t stat_write_count_off;   /* write acquisitions offset */
    int16_t stat_app_usecs_off; /* waiting application threads offset */
    int16_t stat_int_usecs_off; /* waiting server threads offset */

    WT_CONDVAR *cond_readers;   /* Blocking readers */
    WT_CONDVAR *cond_writers;   /* Blocking writers */
};
```

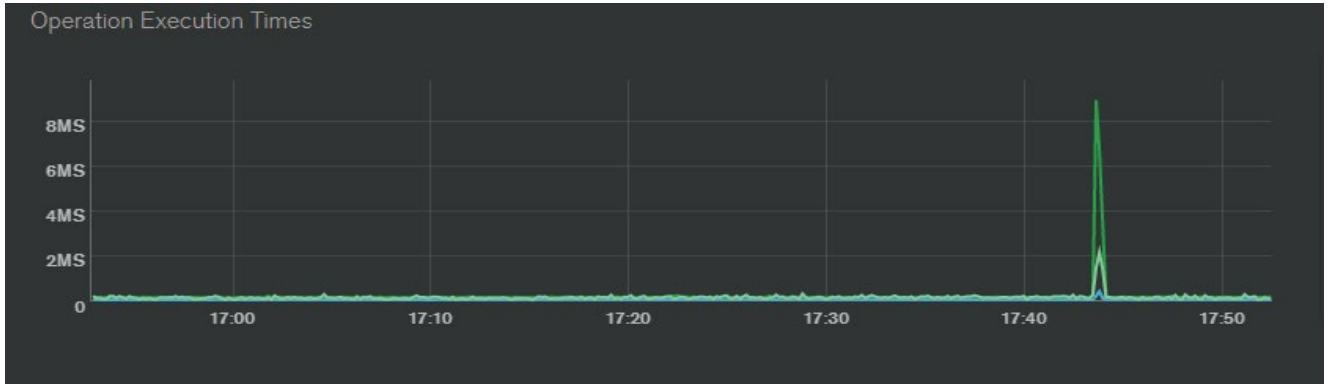
db.serverStatus().wiredTiger.concurrentTransactions

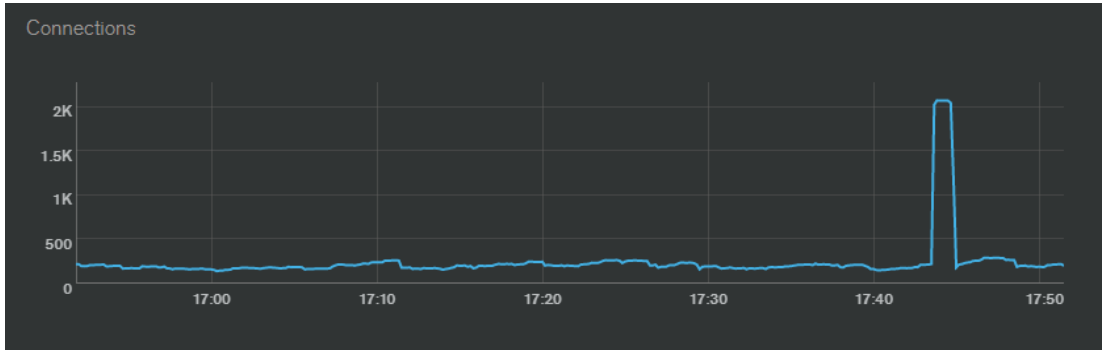
性能指标监控

- db.server.status().wiredtiger
- db.coll.stats()

Notes: 根据上述统计信息计算出mongotop和mongostat的结果







缓存命中率


```
function missrate(intervalMs) {
  var statsStart = db.serverStatus().wiredTiger.cache;
  var startTime = new Date();

  sleep(intervalMs);

  var endTime = new Date();
  var statsEnd = db.serverStatus().wiredTiger.cache;

  var logicalReads =
    statsEnd['pages requested from the cache'] -
    statsStart['pages requested from the cache'];
  var physicalReads =
    statsEnd['pages read into cache'] - statsStart['pages read into cache'];
  var elapsedTime = endTime - startTime;
  var missRate = physicalReads * 100 / logicalReads;
  var logicalReadRate =
    Math.round(logicalReads * 1000 * 100 / elapsedTime) / 100;
  var physicalReadRate =
    Math.round(physicalReads * 1000 * 100 / elapsedTime) / 100;

  print('Elapsed time (ms)', elapsedTime);
  print('logical Read Rate IO/s', logicalReadRate);
  print('physical Read Rate IO/s', physicalReadRate);
  print('wiredTiger miss rate', Math.round(missRate * 100) / 100);
}
```

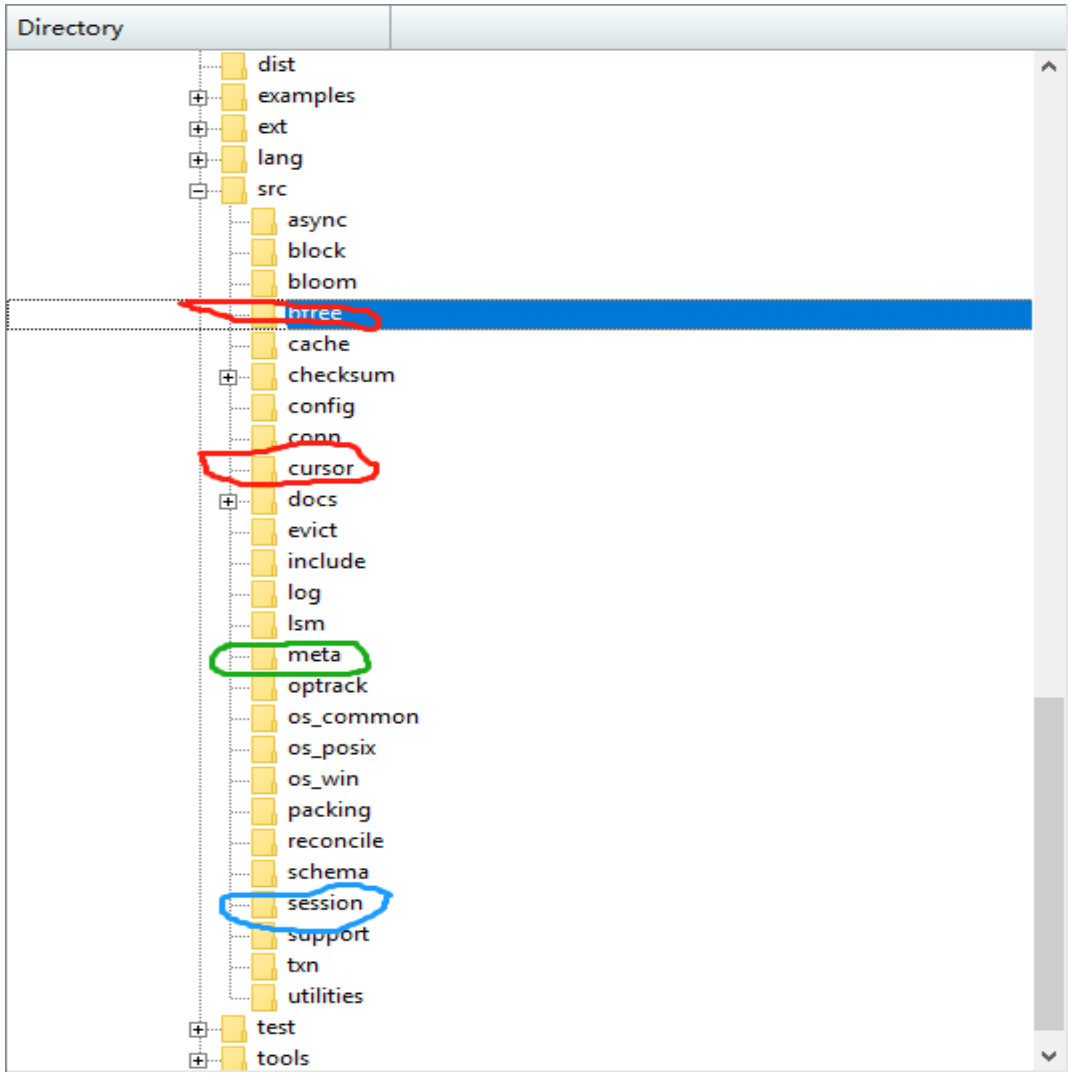
- wiredTiger.cache
- ‘pages requested from cache’
- ‘pages read into cache’
- ‘unmodified page evicted’

参数设置


```
db.createCollection(
"users",
{ storageEngine: { wiredTiger: { configString:
"leaf_page_max=64kb, leaf_value_max=64MB" } } }
)
```

Demo程序

```
#include "wiredtiger.h“
#include  <stdio.h>
int main(int argc, char** argv)
{
char* home = "/tmp/wt";
WT_CONNECTION *conn;
WT_SESSION *session;
WT_CURSOR *cursor;
const char* key, *value;
int ret;
wiredtiger_open(home, NULL, "create", &conn);
conn->open_session(conn, NULL, NULL, &session);
//create a table
session->create(session, "table:demo", "key_format=S, value_format=S");
//open the table
session->open_cursor(session, "table:demo", NULL, NULL, &cursor);
cursor->set_key(cursor, "name");
cursor->set_value(cursor, "peng");
cursor->insert(cursor);

cursor->set_key(cursor, "iris");
cursor->set_value(cursor, "teacher");
cursor->insert(cursor);

cursor->reset(cursor);
while ( (ret = cursor->next(cursor) ) == 0 ) {
cursor->get_key(cursor, &key);
cursor->get_value(cursor, &value);
printf("Got Record: %s, %s\n", key, value);
}
conn->close(conn, NULL);
}
```



Demo程序运行和调试

编译wiredtiger

```
git clone https://github.com/wiredtiger/wiredtiger.git
cd wiredtiger
sh autogen.sh
./configure && make
```

编译

```
gcc -g -I. -o /tmp/wt_demo.o -c ~/working/wt_demo.c
gcc -L .libs/ $PWD/.libs/libwiredtiger-3.1.0.so /tmp/wt_demo.o -o /tmp/wt_demo
LD_LIBRARY_PATH=$PWD/.libs /tmp/wt_demo
```

调试

```
set print pretty
br wt_demo.c:37
r
set scheduler-locking on
```

Q & A