基于B-Tree和LSM存储引擎之基础概念篇

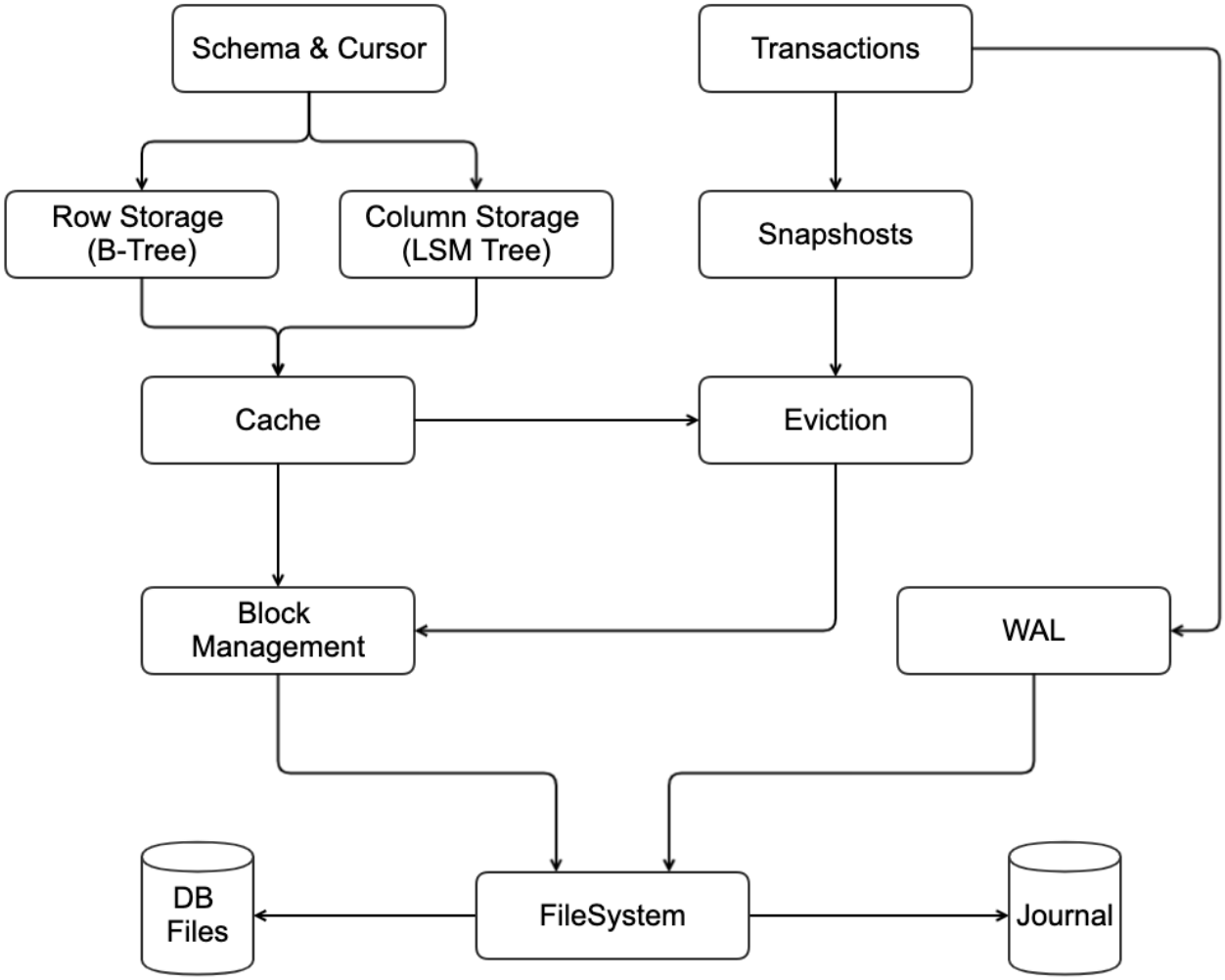
作者	时间	QQ技术交流群
perrynzhou@gmail.com	2022/04/18	672152841

存储内核技术交流

微信扫描二维码，关注我的公众号

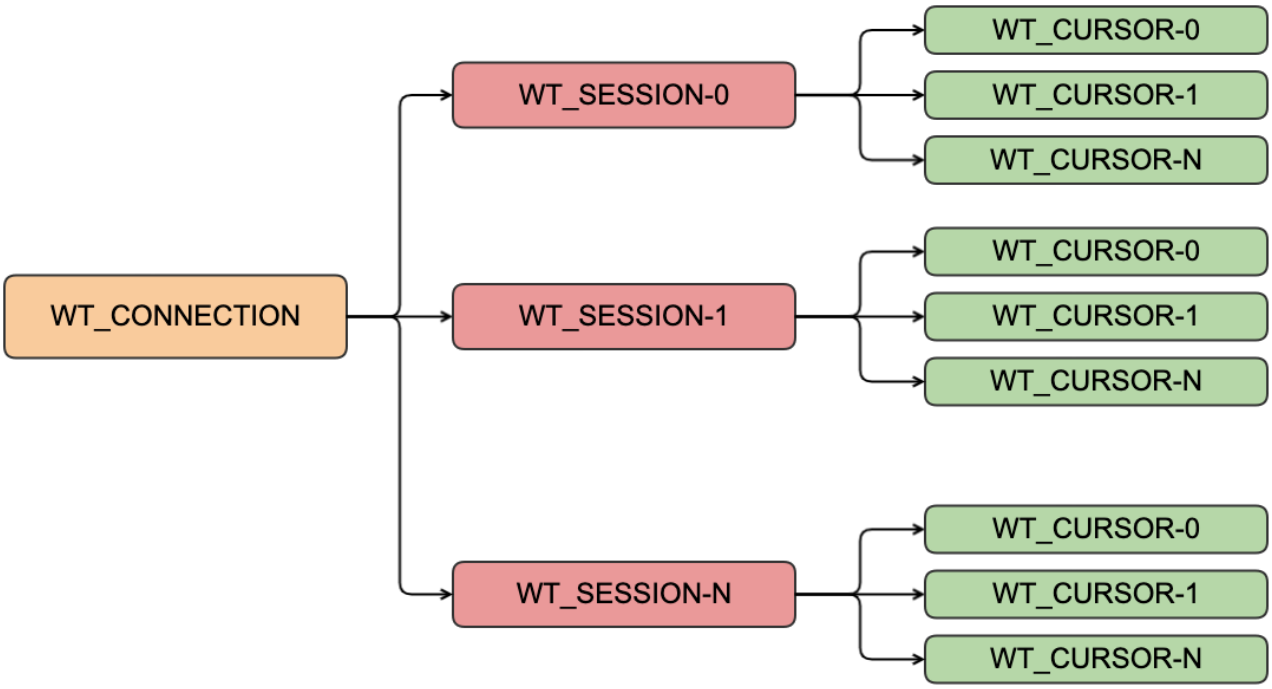
开源存储问题解答社区:https://github.com/perrynzhou/deep-dive-storage-in-china

介绍



- 目前市面上大部分存储引擎是基于`rocksdb`支持`LSM`树，现在介绍的是`wiredtiger`存储引擎支持两种数据结构`b-tree`和`lsm`,支持行存储和列存储，全面支持ACID事务模式，通过配置参数灵活定制自己需要的存储类型.采用的设计思路和数据库设计思路非常相似。在常规应用中写比例重的业务采用`lsm`方式存储性能相对比较高；针对读比例比较重的业务采用`b-tree`比较合适。
- `wiretiger`数据流的写入是采用`wal`模式，先刷日志，写完日志就立即返回。后台有`checkpoint`的线程定期刷脏的`page`到磁盘，这样的方式既保证了写入数据安全，也适当的保证了性能。这种模式和`postgresql`中的`checkpoint`进程类似。

基本概念



WT_CONNECTION

- WT_CONNECTION在wiredtiger存储引擎实例的处理实例，WT_CONNECTION具有排他性的属性，引擎实例任何一刻只会有一个connection处理。
- WT_CONNECTION是通过wiredtiger_open函数进行初始化，当在wiretiger_open的configure参数中设定了create,在调用函数时候会创建这个数据库；如果数据库存在则会忽略创建这个动作。数据库实例启动，wiredtiger引擎内部会启动多个工作线程服务，这些线程服务包括staticstics、logging、eviction、checkpoint、cache management等服务。connection(WT_CONNECTION)涉及到很多操作方法，具体参照如下:

```
// WT_CONNECTION 的结构数据
(gdb) whatis conn
type = WT_CONNECTION *
(gdb) p *conn
$8 = {
  close = 0x7ffff7dd6cb0 <__conn_close>,
  debug_info = 0x7ffff7dd773e <__conn_debug_info>,
  reconfigure = 0x7ffff7dd7d55 <__conn_reconfigure>,
  get_home = 0x7ffff7dd68b3 <__conn_get_home>,
  configure_method = 0x7ffff7dd68cc <__conn_configure_method>,
  is_new = 0x7ffff7dd6c98 <__conn_is_new>,
  open_session = 0x7ffff7dd817b <__conn_open_session>,
  query_timestamp = 0x7ffff7dd85f0 <__conn_query_timestamp>,
  set_timestamp = 0x7ffff7dd8a59 <__conn_set_timestamp>,
  rollback_to_stable = 0x7ffff7dd8eb8 <__conn_rollback_to_stable>,
  load_extension = 0x7ffff7dd618c <__conn_load_extension>,
  add_data_source = 0x7ffff7dd268a <__conn_add_data_source>,
  add_collator = 0x7ffff7dd166b <__conn_add_collator>,
  add_compressor = 0x7ffff7dd1f4a <__conn_add_compressor>,
```

```
  add_encryptor = 0x7ffff7dd33bf <__conn_add_encryptor>,
  add_extractor = 0x7ffff7dd3dde <__conn_add_extractor>,
  set_file_system = 0x7ffff7ddbc21 <__conn_set_file_system>,
  add_storage_source = 0x7ffff7dd4e2f <__conn_add_storage_source>,
  get_storage_source = 0x7ffff7dd5479 <__conn_get_storage_source>,
  get_extension_api = 0x7ffff7dd583c <__conn_get_extension_api>
}

// 针对connection的操作方法
(gdb) whatis stdc
type = const WT_CONNECTION
(gdb) set print pretty on
(gdb) p stdc
$5 = {
  close = 0x7ffff7dd6cb0 <__conn_close>,
  debug_info = 0x7ffff7dd773e <__conn_debug_info>,
  reconfigure = 0x7ffff7dd7d55 <__conn_reconfigure>,
  get_home = 0x7ffff7dd68b3 <__conn_get_home>,
  configure_method = 0x7ffff7dd68cc <__conn_configure_method>,
  is_new = 0x7ffff7dd6c98 <__conn_is_new>,
  open_session = 0x7ffff7dd817b <__conn_open_session>,
  query_timestamp = 0x7ffff7dd85f0 <__conn_query_timestamp>,
  set_timestamp = 0x7ffff7dd8a59 <__conn_set_timestamp>,
  rollback_to_stable = 0x7ffff7dd8eb8 <__conn_rollback_to_stable>,
  load_extension = 0x7ffff7dd618c <__conn_load_extension>,
  add_data_source = 0x7ffff7dd268a <__conn_add_data_source>,
  add_collator = 0x7ffff7dd166b <__conn_add_collator>,
  add_compressor = 0x7ffff7dd1f4a <__conn_add_compressor>,
  add_encryptor = 0x7ffff7dd33bf <__conn_add_encryptor>,
  add_extractor = 0x7ffff7dd3dde <__conn_add_extractor>,
  set_file_system = 0x7ffff7ddbc21 <__conn_set_file_system>,
  add_storage_source = 0x7ffff7dd4e2f <__conn_add_storage_source>,
  get_storage_source = 0x7ffff7dd5479 <__conn_get_storage_source>,
  get_extension_api = 0x7ffff7dd583c <__conn_get_extension_api>
}
```

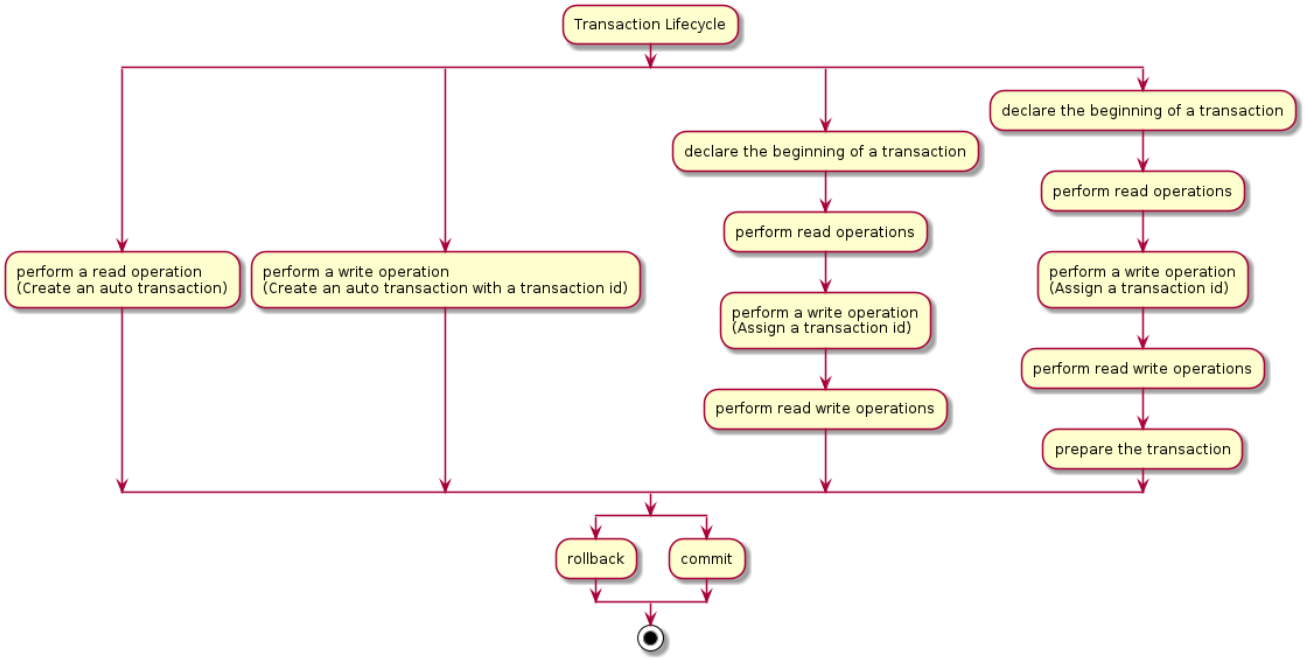
WT_SESSION

- wiredtiger引擎实例打开连接初始化后，如果应用发请求wiredtiger引擎使用session(WT_SESSION类型)来接受请求,所有的操作都是在WT_SESSION这个上下文进行。
- session初始化是通过初始化的connection(WT_CONNECTION类型)参数和open_session函数进行初始化，一个connection可以创建很多session,但是每个session只能属于一个connection.session创建的上限是通过session_max参数在执行wiredtiger_open函数时候传入。引擎内部的eviction也是通过创建session来执行。

```
// session的实例
(gdb) whatis session
type = WT_SESSION *
```

```
// 定义了一些方法和操作
(gdb) p *session
$9 = {
  connection = 0x5555555592a0,
  app_private = 0x0,
  close = 0x7ffff7eea9bb <__session_close>,
  reconfigure = 0x7ffff7eeb1a3 <__session_reconfigure>,
  flush_tier = 0x7ffff7efac34 <__session_flush_tier>,
  strerror = 0x7ffff7efabfe <__wt_session_strerror>,
  open_cursor = 0x7ffff7eec0e9 <__session_open_cursor>,
  alter = 0x7ffff7eeca57 <__session_alter>,
  create = 0x7ffff7eed887 <__session_create>,
  compact = 0x7ffff7efcb06 <__wt_session_compact>,
  drop = 0x7ffff7ef0d08 <__session_drop>,
  join = 0x7ffff7ef2670 <__session_join>,
  log_flush = 0x7ffff7eee472 <__session_log_flush>,
  log_printf = 0x7ffff7eeef11 <__session_log_printf>,
  rename = 0x7ffff7eef8a8 <__session_rename>,
  reset = 0x7ffff7ef0764 <__session_reset>,
  salvage = 0x7ffff7ef37be <__session_salvage>,
  truncate = 0x7ffff7ef4838 <__session_truncate>,
  upgrade = 0x7ffff7ef5835 <__session_upgrade>,
  verify = 0x7ffff7ef6327 <__session_verify>,
  begin_transaction = 0x7ffff7ef6b64 <__session_begin_transaction>,
  commit_transaction = 0x7ffff7ef70d7 <__session_commit_transaction>,
  prepare_transaction = 0x7ffff7ef7888 <__session_prepare_transaction>,
  reset_snapshot = 0x7ffff7ef9066 <__session_reset_snapshot>,
  rollback_transaction = 0x7ffff7ef821e <__session_rollback_transaction>,
  timestamp_transaction = 0x7ffff7ef885d <__session_timestamp_transaction>,
  query_timestamp = 0x7ffff7ef8c41 <__session_query_timestamp>,
  checkpoint = 0x7ffff7efa28a <__session_checkpoint>,
  transaction_pinned_range = 0x7ffff7ef912e
<__session_transaction_pinned_range>,
  transaction_sync = 0x7ffff7ef964b <__session_transaction_sync>,
  breakpoint = 0x7ffff7efafbe <__wt_session_breakpoint>
}
```

WT_TXN



- 事务是把一组操作打包在一起作为一个原子操作，执行过程中要么失败要么成功，不会存在中间的状态。在session内任何时刻只会有一个运行的事务，并且事务属于这个session.从wiredtiger设计角度来讲，事务的实现是通过wal的日志，搭配上checkpoint机制来做.在引擎内部事务是以WT_TXN作为表达的数据结构，保证了ACID属性

WT_CURSOR

- WT_CURSOR是用来获取和更改数据，在引擎API层可以通过open_cursor来创建cursor(类型是WT_CURSOR).cursor包含了查找、遍历、获取、设置这几大类方法。其中最经典的用法就是在btree中通过key来查找value，同时cursor可以用来索引数据的访问、日志文件、元数据等。

```
// cursor的操作方法
(gdb) whatis cursor
type = WT_CURSOR *
(gdb) p *cursor
$10 = {
  session = 0x55555557cc50,
  uri = 0x5555555c3f00 "table:stud",
  key_format = 0x555555578470 "S",
  value_format = 0x5555555d4070 "S",
  get_key = 0x7ffff7e38faf <__wt_cursor_get_key>,
  get_value = 0x7ffff7e3a906 <__wt_cursor_get_value>,
  set_key = 0x7ffff7e3909f <__wt_cursor_set_key>,
  set_value = 0x7ffff7e3b0f0 <__wt_cursor_set_value>,
  compare = 0x7ffff7e088b2 <__curfile_compare>,
  equals = 0x7ffff7e08e86 <__curfile_equals>,
  next = 0x7ffff7e0945a <__curfile_next>,
  prev = 0x7ffff7e09e55 <__curfile_prev>,
  reset = 0x7ffff7e0a355 <__curfile_reset>,
  search = 0x7ffff7e0a776 <__curfile_search>,
```

```
  search_near = 0x7ffff7e0aced <__curfile_search_near>,
  insert = 0x7ffff7e0b26f <__curfile_insert>,
  modify = 0x7ffff7e0c1d0 <__curfile_modify>,
  update = 0x7ffff7e0c969 <__curfile_update>,
  remove = 0x7ffff7e0d131 <__curfile_remove>,
  reserve = 0x7ffff7e0d906 <__curfile_reserve>,
  close = 0x7ffff7e0e085 <__curfile_close>,
  reconfigure = 0x7ffff7e3da29 <__wt_cursor_reconfigure>,
  cache = 0x7ffff7e0e62a <__curfile_cache>,
  reopen = 0x7ffff7e0e96d <__curfile_reopen>,
  uri_hash = 0,
  q = {
    tqe_next = 0x5555555ac910,
    tqe_prev = 0x55555557cdc8
  },
  recno = 0,
  raw_recno_buf = "\000\000\000\000\000\000\000\000",
  json_private = 0x0,
  lang_private = 0x0,
  key = {
    data = 0x7fffffffe170,
    size = 4,
    mem = 0x0,
    memsize = 0,
    flags = 0
  },
  value = {
    data = 0x7fffffffe170,
    size = 4,
    mem = 0x0,
    memsize = 0,
    flags = 0
  },
  saved_err = 0,
  internal_uri = 0x5555555e0830 "file:stud.wt",
  flags = 18415620
}
```

Snapshot

- 快照是一个非常重要的机制，引擎会维护多个版本的数据，快照可以识别特别版本的数据。这个机制非常重要确保读写数据时候的可见性规则。`wt(wiredtiger)`提供两种隔离级别`snapshot`和`read-commited`;对于`snapshot`会存在事务的整个生命周期，对于`read-commited`保证事务提交后其他的事务可以读取到最新数据。
- 快照是通过捕获全局的事务状态来实现，快照存在于`session`的事务中封装在`WT_TXN`数据结构中，其中包括`最大事务ID`、`当前事务ID列表`、`最小事务ID`.`最大事务ID`是通过读取全局的事务计数器得到的，当事务ID等于或者大于快照记录的`最大事务ID`，那么这个事务是无法看到快照数据；`当前事务列表`在快照创建的时候记录当前并发活跃的事务ID，当事务ID存在

于快照记录的`当前事务列表`,当前的事务ID无法看到快照数据；当事务ID小于快照记录的`最小事务ID`，则当前事务可以看到快照数据。