

21-03

SLIGHTLY BETTER CHECKPOINTS

Pause modifying txns while the DBMS takes the checkpoint.

- Prevent queries from acquiring write latch on table/index pages.
- Don't have to wait until all txns finish before taking the checkpoint.

40:01 – 40:05

and then there's concurrency control protocol up above to figure out whether they're allowed to read certain things

然后这里有一个并发控制协议，依据它来确定是否允许它们（知秋注：指读事务）读取某些内容

40:05 – 40:10

we can ignore all that ,but it's all the write transactions, we're just gonna we're just gonna pause them

我们可以忽略所有这些，但对于所有这些写事务，我们要做的就是暂停它们

SLIGHTLY BETTER CHECKPOINTS

Pause modifying txns while the DBMS takes the checkpoint.

- Prevent queries from acquiring write latch on table/index pages.
- Don't have to wait until all txns finish before taking the checkpoint.



40:11 – 40:12

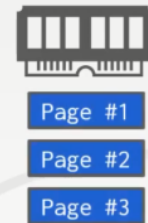
so it would look something like this

so 它看起来像这样

SLIGHTLY BETTER CHECKPOINTS

Pause modifying txns while the DBMS takes the checkpoint.

- Prevent queries from acquiring write latch on table/index pages.
- Don't have to wait until all txns finish before taking the checkpoint.



40:12 – 40:14

so say I have in memory I have three pages

So, 假设在内存中, 我有3个page

40:14–40:17

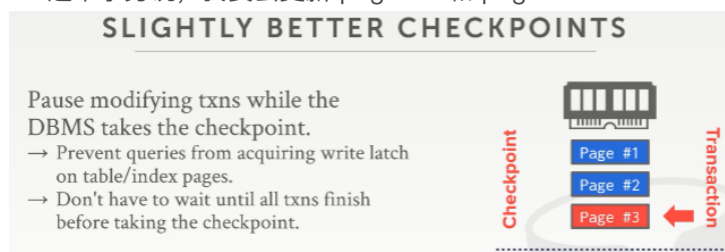
I have a transaction that checkpoint would occur at the same time

我有一个事务在运行, 与此同时, 发生了checkpoint 操作

40:17 – 40:20

so say this transaction is going to update page #3 and page #1

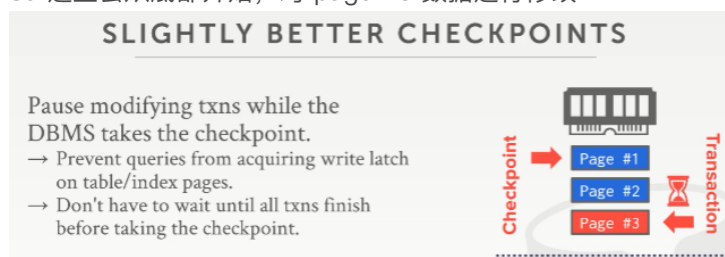
so 这个事务说, 我要去更新 page #3 和 page #1



40:21 – 40:25

so there's going to start at the bottom here, apply its change to page #3

so 这里会从底部开始, 对 page #3 数据进行修改



40:25 – 40:28

and then before it can update page #1, the checkpoint starts

然后, 在它更新page #1前, checkpoint 开始了

40:28 – 40:30

so we have to stall our transaction

so 我们必须将事务停下来

40:30 – 40:35

right because gonna try to acquire the write latch on that page #1, you can't do that,

because the checkpoints occurring

因为该事务会去尝试获取有关page #1的写锁(write latch), 你不能这么做, 因为此时, checkpoint发生了

40:35 – 40:36

so it just stalls

so 该事务就停下来了

40:37 – 40:43

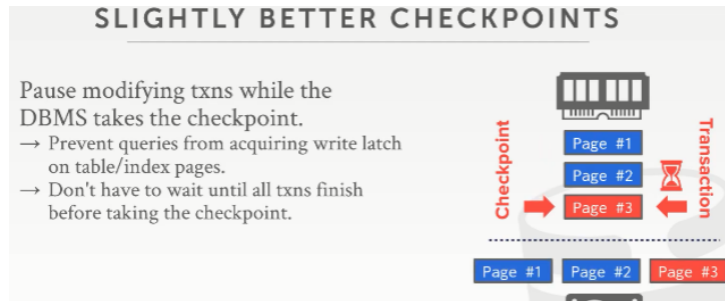
so now the checkpoint all it's really doing is just a sequential scan or a scan of every single page and our buffer pool

so 现在, checkpoint真正要做的就是对每个页面和缓冲池的顺序扫描

40:43 – 40:45

and then it flushing them out the disk

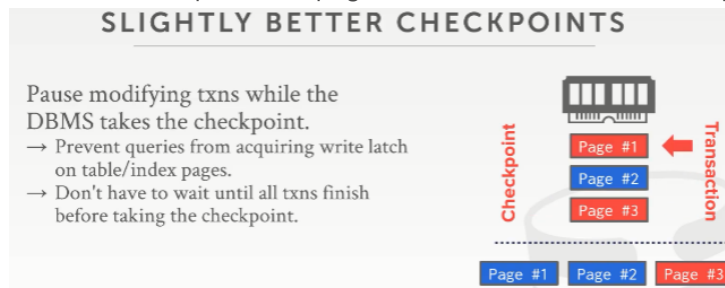
然后, 它将这些都刷出到磁盘



40:45 – 40:51

so our checkpoints going to write out page #1 #2 #3 with the modification that the transaction made to page #3

so 我们的checkpoint会将 page #1 #2和已经被该事务修改的 page #3刷出到磁盘



40:52 – 40:55

then the checkpoint finishes our transaction get stalled

然后checkpoint 结束, 并结束我们事务的等待状态, 让它继续执行

40:55 – 40:57

and then we now update page #1

然后我们开始更新page #1

40:57 – 41:04

all right the problem is now our snapshot in our database, for one query this transaction executed, we saw half their changes

all right, 现在问题来了, 我们的快照在我们的数据库中了, 我们有看到与这个查询相关的事务已经执行的落地了一半

41:05 – 41:09

so our checkpoint or the state of the database on disk is not consistent

我们的checkpoint 或磁盘上数据库的状态不一致 (知秋注: 记录的并不是一个完整的事务落地)

41:12 – 41:20

so in order to handle this, we want to record some additional metadata to figure out, what transactions were running at the time we took the checkpoint

so 未来解决这个东西, 我们想要记录一些其他元数据以找出我们在制作checkpoint时有哪些事务正在运行

41:20 – 41:26

and what pages were dirtied in our buffer pool ,while we took the checkpoint

在我们制作checkpoint 时, 我们的buffer pool中有哪些dirty page

41:26 – 41:33

so that we can use that information to figure out later on, oh well this guy updated page #1 and I missed it on my checkpoint

so 我们可以使用这些元数据信息来找到, 然后这个事务更新了page #1, 我的这个checkpoint 其实就会少了这个更新

41:33 – 41:39

so I knew that I didn't make sure that I want to replay any log record of this guy to put me back in the correct state here

so 我知道我不确定要重演这个事务的任何日志记录来使我回到此处的正确状态 (知秋注: 因为checkpoint包含了一段无法确定的是否完成的事务, 当下一个checkpoint出现, 上一个就会消失, 那就会有一段缺失的信息, 这个要考虑的)

41:40 – 41:41

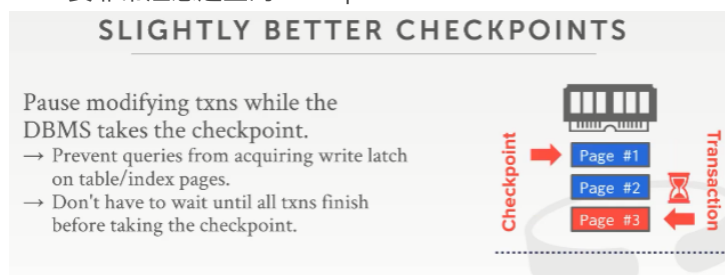
you, that, yes

请说

41:46 – 41:48

now be very careful in which here checkpoint

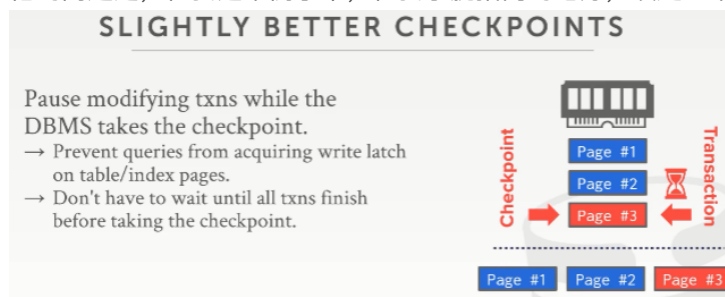
now 要非常注意这里的checkpoint



42:02 – 42:11

yes yes his question is in my example here, I show in wrists like a brute force or a coarse grain write latch on the entire system

他的问题是, 在我这个例子中, 在我手腕指向的地方, 设定一个全局的粗粒度的写锁



42:11 – 42:15

so this guy has to finish his checkpoint before this guy's allowed to go

这个事务只有在checkpoint 制作结束后才允许执行

42:16 – 42:21

or could I say well I'll just release the write latch on page #1 and then I on update it

或者, 我可以这么说, checkpoint 只要释放了关于page #1的写锁, 然后, 该事务就可以更新它

42:21 – 42:24

yes but you still have the same torn up day problem

但是你依然会有这个一模一样的事务撕裂的问题存在 (知秋注: 参照前面我的注释)

42:25 – 42:31

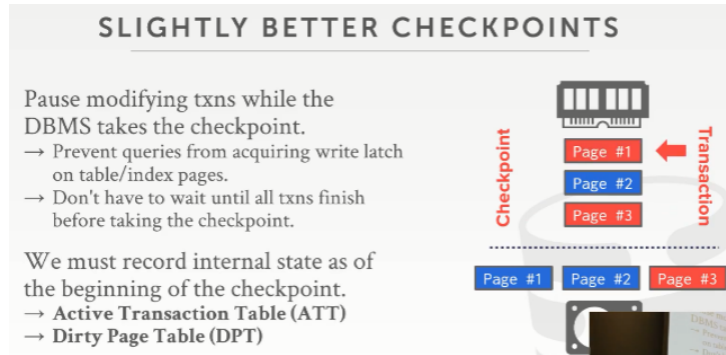
I'm not serious I'm using this as a straw man to say, that what you say, this is a bad idea and we'll see how to do it in a better way

我使用这个例子来说, 对于你讲的, 确实不是一个好的idea, 我们来看看该如何更好地做到这一点

but that's not an obvious optimization yes
但 那确实不是一个好的优化策略，请讲

but that's not an obvious optimization yes

但 那确实不是一个好的优化策略，请讲



42:44 – 42:50

this question is when I say we care about do we care about dirty pages which is gonna be the dirty page table

这个问题是，当我说我们要留意dirty page 时，将这些dirty page 管理到一张dirty page table 中

42:51 – 42:57

do we care about transactions that are paused or any dirty page

我们是否还需要关心事务暂停或因它产生的任何dirty page

42:57 – 43:03

in the real system is any dirty page, in there in we will do fuzzy checkpoints, we do not actually pause transactions, yes

在真实的系统中，对于任何dirty page，在我们做fuzzy checkpoint时，我们实际上不需要暂停事务，请讲

43:12 – 43:14

what's sorry what's it is you don't

sorry, 你再说下

43:31 – 43:40

So the statement is, you don't need the dirty page table, and the active transaction table,

so 他要表达的是, 你不需要这个dirty page table和 active transaction table

43:40 – 43:42

because if you just replayed everything

因为如果你就是要重演 (replay) 一切

43:42 – 43:47

and then reverse them when having put you in the correct state, yes

在你要回滚到正确状态的时候，撤销这些脏修改就是了

43:48 – 43:50

I think I agree with you

我同意你的看法

43:52 – 43:54

but that's gonna be super slow

但这会超级慢

43:56 – 44:01

because you because you're gonna have to update your gonna bring back every single page modify it right

因为你必须对一个个单独的页面进行修改更新，以让它回归正常

44:01 – 44:03

well where's with this metadata we can avoid that

通过这个元数据，我们可以避免这样的工作

44:08 – 44:13

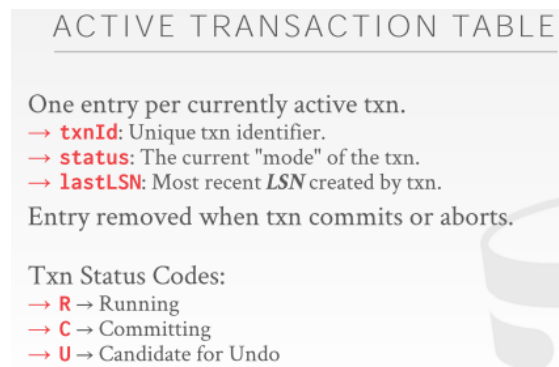
No not true ,you have to go farther back potentially ,give you more slides we'll get there

不对，你可能会做更多的事情，后面我们会谈到

44:14 – 44:16

because you don't know what's been written disk

因为你不知道哪些已经写入磁盘了



44:19 – 44:23

all right so right so there's the active transaction table and the dirty page tables

all right, so, 这里有Active Transaction Table 和 Dirty Page Table

44:23 – 44:27

we're gonna record this information when we take the checkpoint starts

当我们开始制作checkpoint 时，我们要将它们记录在案

44:27 – 44:32

we're gonna write it out with fuzzy checkpoints when the checkpoints ends

当checkpoints 结束时，我们将使用fuzzy-checkpoints将其写出

44:32 – 44:35

and then we'll see these two concepts come up again when we do recovery

然后当我们进行恢复时，我们将再次看到这两个概念

44:35 – 44:40! ! ! !

we're gonna basically replay the log and populate this information to figure out what we need to commit or undo

我们将重演 (replay) 这个日志并根据此信息以找出需要提交或撤消的内容

44:41 – 44:50

so in the active transaction table, this is going to be for every single actively running

transaction at the time the CheckPoint starts, or to record its txnId, its status

so 在这个 active transaction table中，在CheckPoint 开始的时候，该table里面放着一条条活动的正在运行的事务记录，每条记录都有该事务的txnId,它的状态status

44:50 – 44:53

and then the lastLSN that was created by this transaction .

以及创建该事务是最近的那条LSN: lastLSN

44:53 – 44:57

so the status is either what it's running , committing

so 事务状态码status包含了Running、committing

44:57 – 45:02

right so it's Candidate, but before we get to txn-end, or it's something that we think we have to undo

status也包含了Candidate，Candidate表示在我们到达TXN-END之前，遇到一些情况使得我们不得不做undo操作

45:03 – 45:10

it may have to undo we don't know yet right because it's we don't know what its final outcome is gonna be

因为我们不知道它的最终结果会是什么

45:11 – 45:16

and again when we see a transaction end message ,we can we could we can remove this from the ATT

当我们看到一个事务结束的消息后，我们要从ATT（ACTIVE TRANSACTION TABLE）上移除有关该事务的记录

45:16 – 45:18

because we know we're never going to ever see it again

因为我们知道，我们永远不会再次看到这个事务了

45:18 – 45:21

so that's why there's no like completed or finished here

so 这就是为什么status没有包含completed 或finished这样的字眼了

45:22 – 45:27

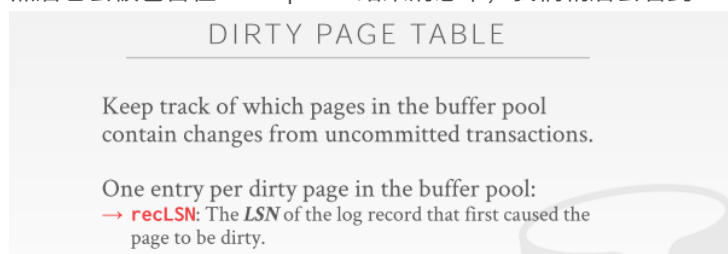
so this would this be hanging out internal memory we can populate this, while we take the checkpoint

在我们制作checkpoint 时，ATT会被放置在内存中

45:28 – 45:31

but then it's included in the checkpoint end message which we'll see in a second

然后它会被包含在checkpoint 结束消息中，我们稍后会看到



45:33 – 45:39

then the dirty page table was just keeping track of all the pages that are in the buffer pool ,pages that have been dirtied in the buffer pool

接着，对于dirty page table，它就是用来跟踪缓冲池中的所有dirty pages

45:39 – 45:41

that were modified by uncommitted transactions

这些dirty pages都是由未提交的事务修改过的

45:41 – 45:50

and for this one we're just going to record the recLSN , which is the log record of the first transaction that modified this page that made it dirty

对于每一条数据（非log）上的recLSN，它代表了自上次该page刷出后，日志记录中第一个修改这个page数据使其变为dirty page的事务条目（知秋注：即recLSN指向日志中第一条使该page变为dirty page的LSN）

45:50 – 45:51

since it was brought into memory

自该page被加载入内存中开始（知秋注：事务修改数据经过刷出落地磁盘后，buffer中清空了，此时，再次将一个page加载进来，若有一个事务将它修改为dirty page，那这个修改代表的LSN就是这里的recLSN）

SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, T_2 is still running and there are two dirty pages (P_{11} , P_{22}).

WAL	
< T_1 BEGIN>	
< T_2 BEGIN>	
< T_1 , A→P ₁₁ , 100, 120>	
< T_1 COMMIT>	
< T_2 , C→P ₂₂ , 100, 120>	
<CHECKPOINT	
ATT={ T_2 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 START>	
< T_2 , A→P ₁₁ , 120, 130>	
< T_2 COMMIT>	
< T_3 , B→P ₃₃ , 200, 400>	
<CHECKPOINT	
ATT={ T_3 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 , B→	

MU-DB

45:53 – 45:57

so let's see a slightly better version of checkpoints that's using this information
so 让我们来看下使用此信息的更好版本的checkpoint

SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, T_2 is still running and there are two dirty pages (P_{11} , P_{22}).

WAL	
< T_1 BEGIN>	
< T_2 BEGIN>	
< T_1 , A→P ₁₁ , 100, 120>	
< T_1 COMMIT>	
< T_2 , C→P ₂₂ , 100, 120>	
<CHECKPOINT	
ATT={ T_2 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 START>	
< T_2 , A→P ₁₁ , 120, 130>	
< T_2 COMMIT>	
< T_3 , B→P ₃₃ , 200, 400>	
<CHECKPOINT	
ATT={ T_3 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 , B→	

MU-DB

45:58 – 46:04

so we see now in our checkpoint entry, we and our log record we're gonna have the ATT
so 我们现在可以看到ppt中我们的checkpoint entry, 我们的log 记录中包含了ATT

46:04 – 46:07

and at this point here we only have one transaction running t_2
在此处, 我们只有一个名为 T_2 的事务中运行

46:07 – 46:09

so that's the only thing we have inside there
so 我们这里的ATT中只有这一个事务

SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, T_2 is still running and there are two dirty pages (P_{11} , P_{22}).

WAL	
< T_1 BEGIN>	
< T_2 BEGIN>	
< T_1 , A→P ₁₁ , 100, 120>	
< T_1 COMMIT>	
< T_2 , C→P ₂₂ , 100, 120>	
<CHECKPOINT	
ATT={ T_2 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 START>	
< T_2 , A→P ₁₁ , 120, 130>	
< T_2 COMMIT>	
< T_3 , B→P ₃₃ , 200, 400>	
<CHECKPOINT	
ATT={ T_3 },	
DPT={P ₁₁ , P ₂₂ >	
< T_3 , B→	

MU-DB

46:10 – 46:17

and then we have the dirty page table, and we have P11 P23 right, because there's P11 was modified here

接着, 我们看到了dirty page table (DPT), 我们有两个dirty page: P11 P23, 这个P11在这里被修改了

46:18 – 46:23

so I pick P11 P22, p11 was modified here and P22 was modified here
so 我选了P11 P22, p11在这里被修改了, p22在这里被修改了

46:23 – 46:30

so the syntax I'm showing now is like, here's the object that was modified ,and it's pointing to what what the page number was, right

so 我现在展示的语法就像，这个对象被修改了，它指向它所在的那个page number

46:30 – 46:35

so in this case here, we don't record anything about transaction t1

so 在这个例子中，我不需要去记录任何关于T1的东西

46:35 – 46:38

because transaction t1 committed before my checkpoint started

因为在我的checkpoint开始之前，T1已经提交了

46:38 – 46:41

so I don't care about it at this point anymore

so 我无须关心它任何东西

SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, T_2 is still running and there are two dirty pages (P_{11}, P_{22}).

At the second checkpoint, T_3 is active and there are two dirty pages (P_{11}, P_{33}).

This still is not ideal because the DBMS must stall txns during checkpoint..

WAL

```
<T1 BEGIN>
<T2 BEGIN>
<T1, A→P11, 100, 120>
<T1 COMMIT>
<T2, C→P22, 100, 120>
<CHECKPOINT
  ATT={T2},
  DPT={P11, P22}>
<T3 START>
<T2, A→P11, 120, 130>
<T2 COMMIT>
<T3, B→P33, 200, 400>
<CHECKPOINT
  ATT={T3},
  DPT={P11, P33}>
<T3, B→P33, 400, 600>
```

46:43 – 46:47

so then now in the second checkpoint t3 is still active

so 然后，在第二个checkpoint 条目这里，可以看到，T3是活跃的

46:47 – 46:52

and then we have two dirty pages here ,because t2 committed before our transaction started

接着，我们这里有两个dirty page，因为在我们T3开始前，T2已经提交了

46:53 – 46:53

right

46:55 – 47:01

so this is still not ideal because we're still stalling all our transactions in order to take this

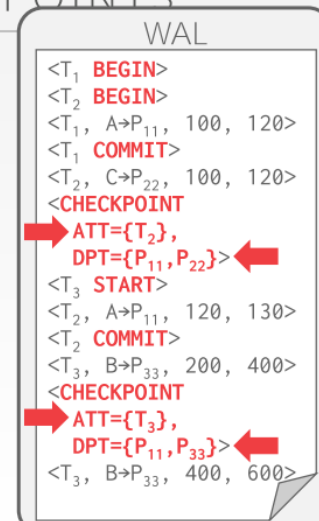
so 为了制作checkpoint，我们现在依然没有办法，依然要让我们所有的事务暂停下来

SLIGHTLY BETTER CHECKPOINTS

At the first checkpoint, T_2 is still running and there are two dirty pages (P_{11}, P_{22}).

At the second checkpoint, T_3 is active and there are two dirty pages (P_{11}, P_{33}).

This still is not ideal because the DBMS must stall txns during checkpoint...



47:02 – 47:08

right so we're pausing everything at this point here, these guys are not allowed to modify it

so 我们现在在这里暂停所有操作，不允许这些事务修改它

47:08 – 47:11

and so this is just saying we wrote out a checkpoint

so 这就是在说我们写出了 checkpoint

47:11 – 47:19

but oh by the way here's some stuff that that that could have been modified during this time to make sure that you find it

顺便说一下，这里有些东西可能在这段时间内已经修改过，以确保你找到它

47:20 – 47:24

so so like the first one, the first checkpoint a scheme I showed you

so 就像我这里给你们展示的 checkpoint scheme

47:25 – 47:26

nobody actually does this one either

实际上也没有人这么做

FUZZY CHECKPOINTS

A **fuzzy checkpoint** is where the DBMS allows active txns to continue the run while the system flushes dirty pages to disk.

New log records to track checkpoint boundaries:

- **CHECKPOINT-BEGIN**: Indicates start of checkpoint
- **CHECKPOINT-END**: Contains ATT + DPT.

47:28 – 47:35

everyone instead does fuzzy checkpoints, since that support high performance checkpoints are doing fuzzy checkpoints

相反，大家都使用 fuzzy checkpoint，就因为 fuzzy checkpoint 的高性能

47:35 – 47:44

so fuzzy checkpoint is just where we're gonna allow transactions to keep on running keep on modifying the database, while we're taking the checkpoint

so fuzzy checkpoint 会在我们制作 checkpoint 的时候，允许事务继续运行，继续修改

47:45 – 47:52

and so in order to record the boundaries of, when the checkpoint started, when the checkpoint finishes to know whether something could have been written out that we missed

因此，在checkpoint开始时记录边界，当checkpoint完成时就能知道我们是否错过了某些内容

47:53 – 47:57

we add a fuzzy checkpoint-begin and checkpoint-end log message

我们这log中添加了checkpoint-begin和checkpoint-end信息

47:57 – 47:59

so the begin is just telling us when the checkpoint started

checkpoint-begin就是告诉我们checkpoint工作开始了

47:59 – 48:01

and the end tells us when it finishes

checkpoint-end是指checkpoint工作完成了

48:01 – 48:08

and this will include the ATT and DPT that that that occurred during during the execution of the checkpoint

checkpoint-end会包含checkpoint执行期间的ATT 和 DPT信息

FUZZY CHECKPOINT

The **LSN** of the **CHECKPOINT-BEGIN** record is written to the database's **MasterRecord** entry on disk when the checkpoint successfully completes.

Any txn that starts after the checkpoint is excluded from the ATT in the **CHECKPOINT-END** record.

UDB

WAL

```
<T1 BEGIN>
<T2 BEGIN>
<T1, A→P11, 100, 120>
<T1 COMMIT>
<T2, C→P22, 100, 120>
<CHECKPOINT-BEGIN>
<T3 START>
<T2, A→P11, 120, 130>
<CHECKPOINT-END
  ATT={T2},
  DPT={P11}>
<T2 COMMIT>
<T3, B→P33, 200, 400>
<CHECKPOINT-BEGIN>
<T3, B→P33, 10, 12>
<CHECKPOINT-END
  ATT={T3},
  DPT={P33}>
```

48:10 – 48:13

so to go back here now,so now we have our checkpoint-begin

so 我们回到这里，此时我们有一个checkpoint-begin

48:13 – 48:16

And then checkpoint does these things

然后checkpoint 做了它要做的事情

FUZZY CHECKPOINT

The **LSN** of the **CHECKPOINT-BEGIN** record is written to the database's **MasterRecord** entry on disk when the checkpoint successfully completes.

Any txn that starts after the checkpoint is excluded from the ATT in the **CHECKPOINT-END** record.

UDB

WAL

```
<T1 BEGIN>
<T2 BEGIN>
<T1, A→P11, 100, 120>
<T1 COMMIT>
<T2, C→P22, 100, 120>
<CHECKPOINT-BEGIN>
<T3 START>
<T2, A→P11, 120, 130>
<CHECKPOINT-END
  ATT={T2},
  DPT={P11}>
<T2 COMMIT>
<T3, B→P33, 200, 400>
<CHECKPOINT-BEGIN>
<T3, B→P33, 10, 12>
<CHECKPOINT-END
  ATT={T3},
  DPT={P33}>
```

48:17 – 48:23

and then in the checkpoint-end we include that we have transaction t2, because t2 started before the checkpoint started

接着就是这个 checkpoint-end，我们可以看到，它（ATT）里面包含了T2，因为t2在checkpoint 开始之前开始

48:23 – 48:29

and then the dirty page table tells us that P11 was modified during during the checkpoint as well

然后 从dirty page table（DPT）中可以看到P11是checkpoint 执行期间有发生修改的dirty page

48:29 – 48:38

right we don't need to include t3 here, because the t3 started before our checkpoint started , sorry , started after my checkpoint started

这里，我们不需要包含T3，因为T3是在我checkpoint开始后才开始的（checkpoint只制作落地它开始前的那些状态，关于它制作期间发生的修改，只需要关注它开始前活动的事务即可）

48:39 – 48:46

so the once we have the checkpoint and written out to disk successfully

so 一旦我们有了checkpoint 并成功写出到磁盘

48:46 – 48:51

and we can which means we flushed all the pages out, that we wanted to take you in the checkpoint

这意味着我们刷出了所有pages，这些page内容都在checkpoint中了

48:51 – 48:57

then we go ahead and update our MasterRecord to now include the the point to the checkpoint-begin

然后，我们继续，更新我们的MasterRecord，将它指向这个checkpoint-begin: checkpoint-begin

48:57 – 49:01

because that's going to be our anchor point where we start our analysis during recovering

因为我们会将它作为我们的一个锚点，在恢复的过程中用于开始我们的日志分析

49:02 – 49:08

because we're going to know at this point here ,right here's all we flushed all the dirty pages

因为我们知道，在这个位置，我们将所有dirty page刷了出去

49:08 – 49:10

but we kept track of maybe ones that we may have missed

但我们需要持续跟踪那些我们可能错过的修改（checkpoint执行期间，ATT事务对page又进行了修改，但并没有落地）

49:10 – 49:11

because because they got modified

因为这些page发生了修改

49:13 – 49:13

yes

49:17 – 49:22

what do you keep the checkpoint what ? what do mean where did he keep it

你的意思是它是如何跟踪维护的

49:31 – 49:37

when you say when you mean like the wrong record, it goes in the log, but when it ends you shove it in

就像你说的，这有一个错误的记录，它在log中，但当它结束，你该怎么办

49:40 – 49:44

when and when you have scanned through the buffer pool and written out all the dirty pages to disk

当你需要通过对缓冲池进行扫描并将所有dirty pages写出到磁盘时

49:45 – 49:49

and after you flush them out you fsync right, because you make sure it's durable

在你将它们刷出到磁盘后（你可以通过fsync做到），因为你要确保它已经被持久化了

49:49 – 49:54

then you add the log entry here, and it's committing I'm using committing that quotes

然后，你在这里添加一条log日志（CHECKPOINT-END），它代表着“提交”了，这里的提交我使用了引号

49:54 – 49:57

because like a regular transaction, I flushed the log record for this to disk

因为这就像是一个正常的事务一样，我将关于这个CHECKPOINT的log日志刷出到磁盘了

50:01 – 50:01

yes

50:13 – 50:18

this question is I holding any locks on the entire database why I write this thing out

他的问题是，我已经拿着整个数据库的锁了，为什么它（checkpoint）还能写出（知秋注：多版本数据！！）

50:19 – 50:19

no

50:26 – 50:27

What mean, in here

什么意思？这里？

50:31 – 50:31

yes

50:59 – 51:05

yes I think I think what is saying there is a sort of stop the world moment here, where

you briefly flush this thing out

我想他说的是，在这里有一个STW（stop the world，全局暂停），我们可以利用这段时间快速将checkpoint相关内容刷出到磁盘

yes

51:05 – 51:08

but that's not a blink it's a minor thing

但它可不是一个短暂的小事情

51:14 – 51:18

correct yes they can change whatever else they want to change yeah, in the regular

buffer pages yes

没错，它们可以在常规的buffer pages中去修改它们想修改的

51:21 – 51:21

okay

ARIES – RECOVERY PHASES

Phase #1 – Analysis

→ Read WAL from last checkpoint to identify dirty pages in the buffer pool and active txns at the time of the crash.

Phase #2 – Redo

→ Repeat all actions starting from an appropriate point in the log (even txns that will abort).

Phase #3 – Undo

→ Reverse the actions of txns that did not commit before the crash.

51:24 – 51:25

So now let's do recovering

so 现在, 我们来看 recover阶段

51:26 – 51:32

after all that after 40 minutes of minutiae of log scene with summary and fuzzy checkpointing

在聊了40分钟日志场景的细节和fuzzy checkpointing 后

51:32 – 51:33

let's talk about actually recover this

我们来讨论下recover

51:33 – 51:40

and then and given that everything we've set up now that we have all this extra metadata, that we're recording

现在, 我们设定并已经有了所有这些额外的元数据

51:40 – 51:41

recovery actually is not going to be that bad

recovery (恢复) 实际上不会那么糟糕了

51:42 – 51:45

but the tricky part is just figuring out where you start each of these phases in the log
但棘手的部分就是你要弄清楚在日志中每个阶段开始的位置

51:46 – 51:51

so the analysis phase you're gonna look at your MasterRecord for the database on disk
so 在分析阶段, 你将查看数据库在磁盘上的MasterRecord

51:51 – 51:55

and that's gonna give you the location of where the last checkpoint–begin in the log
通过它可以获取到日志中最后一个checkpoint–begin所在的位置

51:55 – 52:03

so you jump to that location ,and you scan forward to time till you reach the end of the log

so 你就可以跳到这个位置, 然后向前扫描, 直到到达日志末尾

52:03 – 52:11

and then you're just going to populate the DPT and ATT to keep track of ,what think what was going on in the system at the moment of the crash

然后，你就能拿到DPT 和 ATT去追踪系统崩溃时发生了什么

52:13 – 52:19

and then that's gonna figure out what transactions you need to abort ,which one's actions you need to make sure that you commit

然后就可以弄清楚你需要中止哪些事务，需要执行哪些操作以确保提交

52:20 – 52:26

then in the redo phase ,you're gonna jump to some appropriate location in the log

接着在redo阶段，你会跳到日志中的某个适当位置

52:26 – 52:32

where you know there's could be potential changes from transactions that did not make it safely to disk

即你知道的，有些未提交的事务可能会存在潜在的变化，可能无法将其修改安全地落地到磁盘

52:33 – 52:37

and you can start reapplying those changes until you get to the end of the log

你可以开始重演这些更改，直到到达日志末尾

52:38 – 52:43

and you're gonna do this for any every transaction you see, even ones that are you know end aborting

你会为每个看到的事务执行此操作，哪怕其中一些事务最终中止了

52:43 – 52:47

because on the analysis phase, you see everything you know to the first pass

因为在分析阶段，在过一遍后，你会看到所有

52:47 – 52:49

so you know what's gonna commit what's gonna abort

so 你会知道哪些事务提交了，哪些中止了

52:49 – 52:54

so then in the redo phase just for safety reasons ,we're just gonna do you know reapply everything

so 在redo阶段，为了安全，我们就是要重演日志中的所有内容

52:55 – 53:03

then the undo phase, now you're gonna go back in reverse order from the end the log up until some point

接着，在undo 阶段，现在，你将从日志末尾开始以相反的顺序返回到某个点

53:03 – 53:08

to reverse any changes from transactions that you know did not commit

撤消你未提交的事务中的任何更改

53:09 – 53:16

and when the undo phase is done, then the database is now in a state that success state
当undo阶段结束，那么数据库现在处于已成功的状态

53:16 – 53:22

that success at the moment of the crash with no partial updates from abortive transactions

这个成功，在在此意味着，此时，因系统崩溃而中止的事务所发生的更新被撤销

53:22 – 53:28

and all changes from committed transactions have been applied to disk, in the back ,yes
提交的事务中的所有更改都已应用到磁盘中

53:31 – 53:32

Next slide yeah okay

下一张幻灯片

31

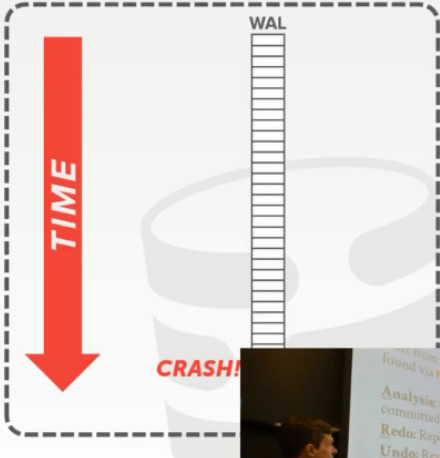
ARIES – OVERVIEW

Start from last **BEGIN-CHECKPOINT** found via **MasterRecord**.

Analysis: Figure out which txns committed or failed since checkpoint.

Redo: Repeat all actions.

Undo: Reverse effects of failed txns.



CMU-DB

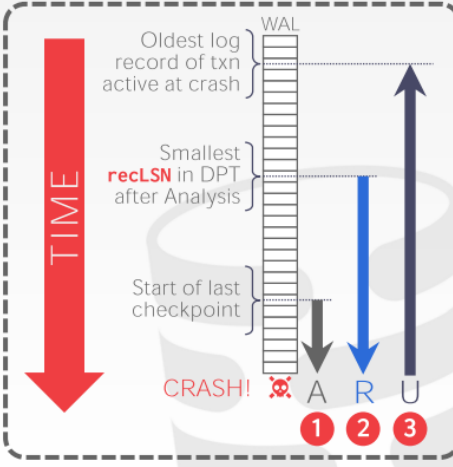
ARIES – OVERVIEW

Start from last **BEGIN-CHECKPOINT** found via **MasterRecord**.

Analysis: Figure out which txns committed or failed since checkpoint.

Redo: Repeat all actions.

Undo: Reverse effects of failed txns.



53:33 – 53:37

so again three Phase : analysis redo and undo

so 再次来看这三个阶段: analysis 、redo 和undo

53:37 – 53:43

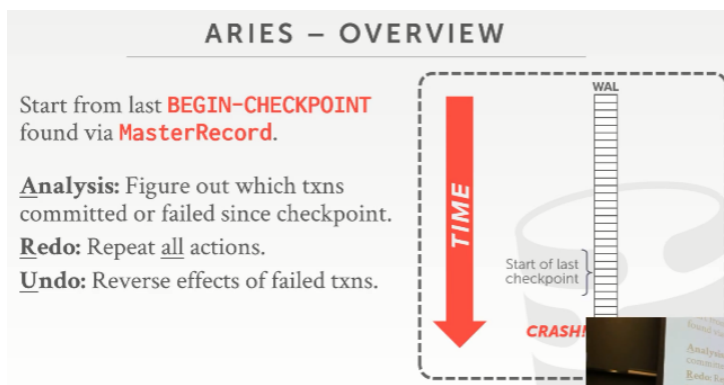
so the very beginning we look at the begin the figure out where the begin checkpoint is in the MasterRecord

so 在最开始, 我们要知道checkpoint-begin的位置可以有MasterRecord来确定

53:43 – 53:45

and that's where we're gonna begin our analysis, right

这就是我们开始做分析的地方

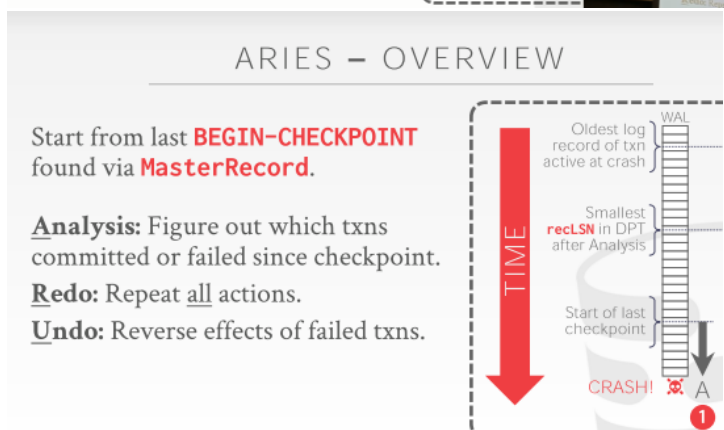


53:45 – 53:49

so let's say that this log record here is the start in the last checkpoint
so 我们会说对于这里的log日志来说，它的开始位置在最后的那个checkpoint

53:49 – 53:52

because again that's in our MasterRecord we know where that where that is
因为那就是我们的MasterRecord所指向的位置所在



53:52 – 53:59

and then now we're to scan forward through time and look at these log records and build out our ATT and DPT

然后我们将向前顺着时间线浏览并查看这些日志记录，并构建我们的ATT和DPT



53:59 – 54:09

and then now we got to figure out well what for the redo phase, what is the smallest recLSN in the dirty page table that we found after do our analysis
我们必须弄清楚redo阶段中，分析后发现的DPT（dirty page table）中所包含page的最小recLSN是多少

54:09 – 54:14

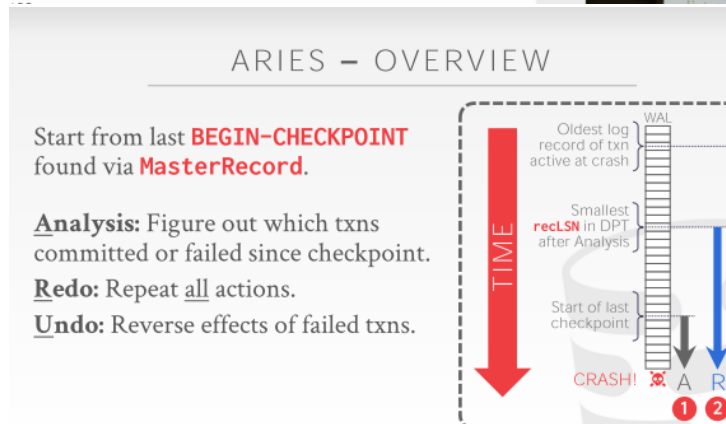
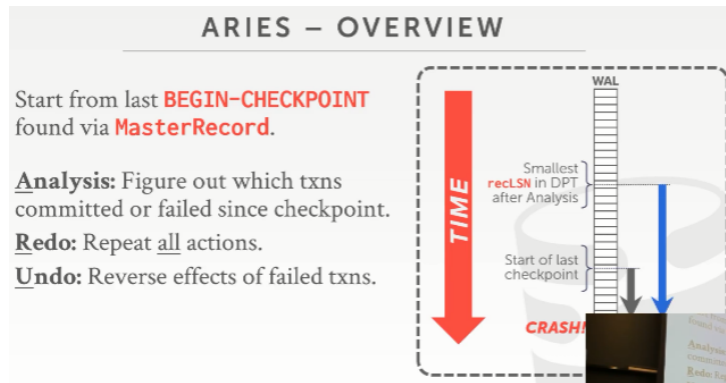
right so this is telling us this is the location of the first log record

so 这个最小的recLSN可以告诉我们这是第一个修改该page的日志记录位置

54:14 – 54:20

the oldest log record that modified a page that may have not been written a disk

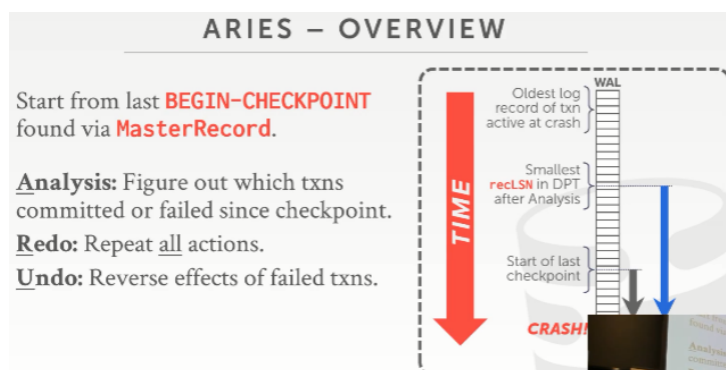
这个最早的修改page的日志条目所对应的表数据可能并未写出到磁盘



54:20 – 54:24

so when we redo we jumped at this point and reapply all our changes

so 当我们进入redo阶段时，我们会跳到这个位置来重演所有我们的修改



54:26 – 54:32

and then now in the undo phase, we start at the the end point and go back in time

接着，我们在undo阶段，我们从结束点开始，进行时光逆转

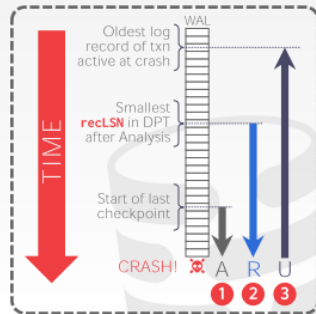
ARIES – OVERVIEW

Start from last **BEGIN-CHECKPOINT** found via **MasterRecord**.

Analysis: Figure out which txns committed or failed since checkpoint.

Redo: Repeat all actions.

Undo: Reverse effects of failed txns.



54:32 – 54:41

up until some point where we know that this is the oldest transaction, that got a border that was actively running, while we took our checkpoint

向上，直到某个位置，在此，我们知道此处是我们制作这个checkpoint 时，所涉及的最老的那个活动事务，

54:41 – 54:43

and we reverse all those changes

我们撤消所有这些更改

54:43 – 54:48

so the errors are sort of showing you the boundaries of how far you get to go back in time in the log

由此，我们可以得到到底需要在log中从哪里开始恢复，该怎么做

54:49 – 54:55

so you know I'll go through these more precisely in context LSNs one by one

我将在这个上下文中更准确地逐一遍历这些LSN（知秋注：确定了范围，逐一遍历即可）

54:55 – 55:01

but this clear at a high level what we're doing , analysis goes forward time, redo goes forward a time

我们简单的从一个高级层面来看是怎么做的，analysis 和redo 阶段是按时间线往前（图中是向下箭头）走的

55:01 – 55:03

and then undo goes backwards in time

undo 阶段是按时间线往后走的（图中是向上箭头）

55:04 – 55:09

and for undo I may not be undoing every single log record I see here

对于undo来说，我可能不会撤消在这里看到的每个日志记录

55:10 – 55:15

right it's just for the transactions that identify in my ATT after the analysis ,that should not have committed

撤销操作仅针对于位于我所分析后得到的ATT中所定义的活动事务，这些事务并没有提交

ANALYSIS PHASE

Scan log forward from last successful checkpoint.

If you find a **TXN-END** record, remove its corresponding txn from **ATT**.

All other records:

→ Add txn to **ATT** with status **UNDO**.

→ On commit, change txn status to **COMMIT**.

For **UPDATE** records:

→ If page **P** not in **DPT**, add **P** to **DPT**, set its **recLSN=LSN**.

55:19 – 55:24

okay so this sort of summarizes are more concretely what I just said

这个ppt相对于我刚才说的，更加详细具体

55:25 – 55:28

so analysis phase we're gonna scan the log forward from the last successful checkpoint

analysis 阶段，将从最后一个成功的checkpoint 按时间线从上到下往前去扫描日志

55:28 – 55:33

anytime you find a transaction end record during an analysis we can remove it from my ATT

在analysis期间，不管任何时候，只要你发现了一个 TXN-END 记录，我们就可以将该事务id从我们的ATT中移除

55:34 – 55:41

right otherwise for any other record, if it's first time I've ever seen this transaction, we add it to the ATT with the status of undo

否则，对于其他任何记录，如果这是我第一次看到的事务，我们将其添加到ATT中，以便后续进行undo

55:42 – 55:46

because we don't know, because we're going forward in time, we don't know whether it's gonna abort later on

因为我们是按照时间线往前走的，我们并不知道该事务是否被中止了

55:48 – 55:53

if we see a commit record then we just change its status to commit, like I said one is sinless and we can remove it

如果我们看到一个commit 记录，然后我们就将它的修改提交，和我之前说的一样，将它从ATT中移除

55:53 – 56:01

and then for any update record, we're gonna look to see whether the page that that's in the update record that's being modified is in our DPT

然后，对于任何一个更新记录，我们都要去看看它所针对的page是不是在我们的DPT 中

56:01 – 56:05

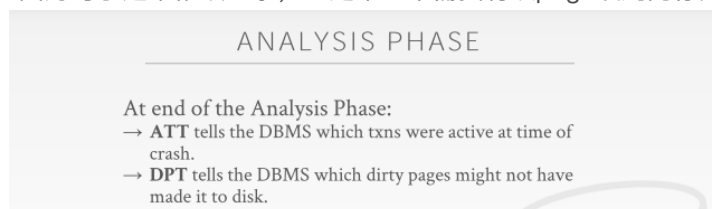
if not then we go ahead and add it ,and we set the recLSN to be LSN

如果没在，就添加进去，然后设定该page的recLSN为该修改记录的LSN

56:06 – 56:13

because this is this again this is telling us ,this is the log record that, first made this page dirty when it was brought into memory

因为对于这个修改记录，它是第一个需要将该page读到内存并修改的存在，所以它属于recLSN



56:16 – 56:18

so now at the end of the analysis phase

so 在analysis阶段的最后，

56:18 – 56:23

the ATT is gonna tell us what are all the active transactions that we had running in the system at the moment of the crash

通过这个ATT，我们就可以知道在系统崩溃的时候，当前系统中运行的所有活动的事务有哪些

56:24 – 56:31

and the DPT are gonna tell us, what are the dirty pages that could have been in our buffer pool, that may have not been written a disk

通过这个 DPT, 我们就可以知道当时有哪些dirty pages没有写入磁盘

56:32 – 56:35

and we're doing this we have to build this table

我们要做到这些, 就需要去构建这个DPT

56:35 – 56:40

because again we're not logging out every time we do a buffer pool flush to a page on the disk

因为我们并不会每次在log刷出去的时候, 都去将对应的数据库表buffer pool中的page也刷出到磁盘

56:40 – 56:42

we're not recording that in the log

我们并没有将这个DPT记录在log中

56:42 – 56:48

or the log records do not that tell us potential will get modified, and we're trying to reconstruct it

log并不会告诉我们那些潜在已经修改的dirty pages, 我们需要尝试重新构建它

56:48 – 56:49

yes

请讲

56:53 – 57:00

this question is how do I know for sure whether pages be written to disk, like in the log or in the real world, like on hardware or analog

他的问题是, 我该怎么做才能确定这些表数据page是否被写出到磁盘了,

57:04 – 57:04

you can't

你无法做到

57:06 – 57:08

because I there's no information that tells me that it's been written it's

因为并没有信息可以告诉我们它们已经被写出到磁盘了

57:10 – 57:14

not entirely true, we see redo we'll see in a second

不完全对, 我们等会会看到redo阶段的内容

57:14 – 57:25

but in general, if you know the LSN of a log record you're looking at is less than the recLSN of the log of the page as exists on disk

但通常, 如果一条log 记录的LSN 小于该page的recLSN

57:26 – 57:32

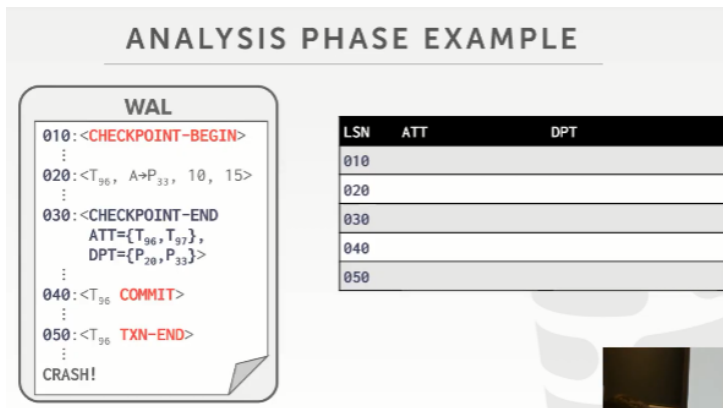
then you know that that your change got written out the disk but to the page got read not the disk,

然后你就知道, 这个修改已经写出到磁盘了, 但我们并没有从磁盘中读到该修改后的数据,

57:32 – 57:36

then it got dirty again by another one but your thing got written out before then

那它就被另一个后面的修改操作给修改了, 变为dirty 数据落地了



57:39 – 57:43

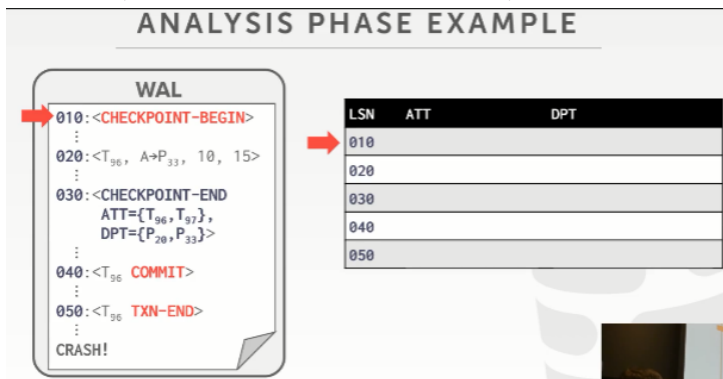
all right so quick overview of the analysis phase

all right, 我们快速概览下分析阶段

57:43 – 57:50

and so here I'm just showing you the with the ATT into DPT or gonna look like at these different LSN

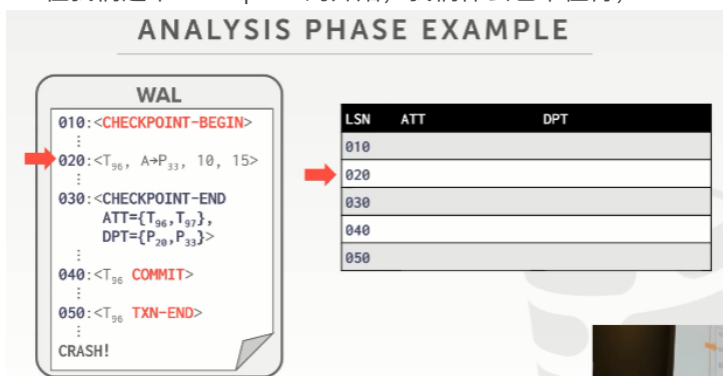
so 在这里, 我向大家展示ATT到DPT的过程, 并联系到这些不同的LSN



57:50 – 57:55

so begin our checkpoint ,we don't know anything ,so the ATT and DPT are empty

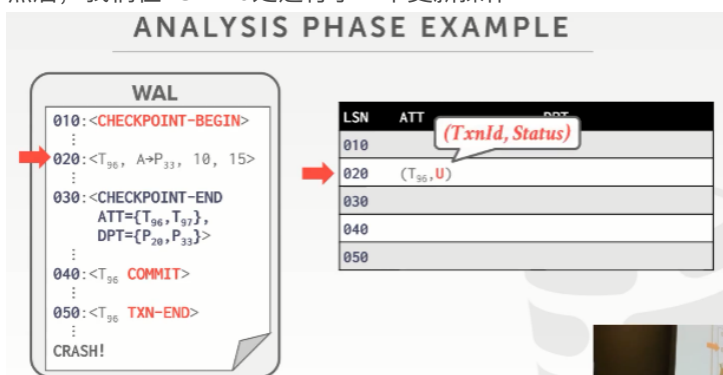
so 在我们这个checkpoint的开始, 我们什么也不值得, so ATT和DPT现在是空的



57:55 – 57:59

then we do an update in LOG sequence 20

然后, 我们在LSN 20处进行了一个更新操作



57:59 – 58:02

so for this one here we have transaction T96

so 对于这个020, 它的事务是T96

58:02 – 58:03

it's the first time we've ever seen it

它是我们首次看到的修改

58:04 – 58:08

right because again we don't have a begin record here ,because it began before a checkpoint started

因为我们这里并没有看到该事务开始的日志条目(即< T96, BEGIN >),因为这个条目在 checkpoint 之前发生的

58:09 – 58:17

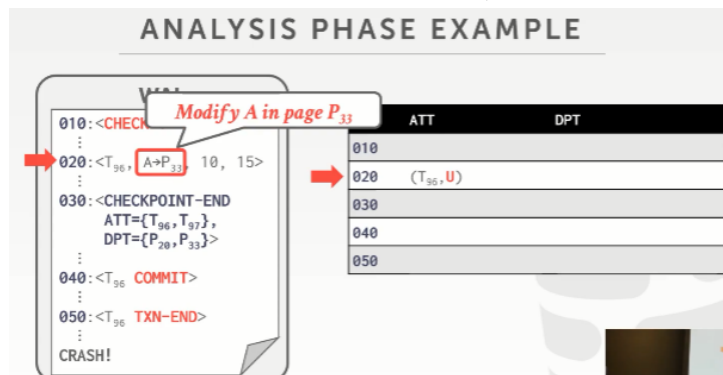
so we see that ,and we update our ATT to say hey we ever turns out in here T 96

so, 在我们看到它后, 我们会更新我们的ATT, 意思是, 嘿! 这里有个活动的事务T96

58:17 – 58:20

and the status is a candidate for undo ,because we don't know whether it's going to commit or not

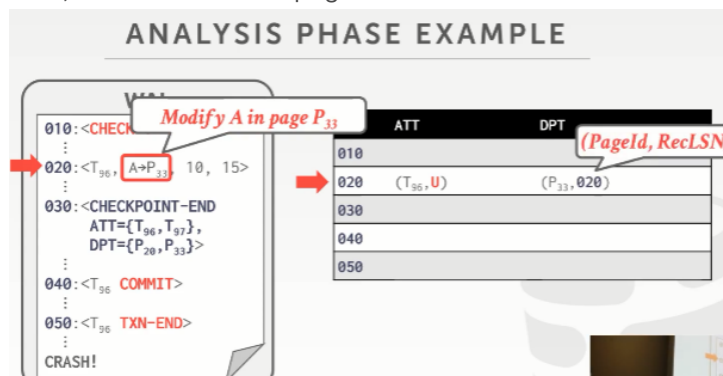
即将该事务列为undo阶段的潜在操作对象, 因为我们并不知道它是否在后面提交了



58:21 – 58:23

and then we see that it modified page 33

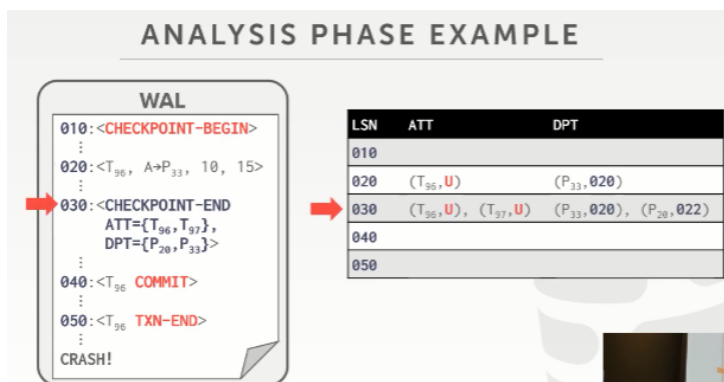
然后, 我们看到它修改了page 33



58:23 – 58:30

so we add that to our dirty page table with the recLSN of our log record here

so 我们将该page添加到DPT (dirty page table), 并将该LSN作为该page的recLSN



58:31 – 58:38

then now our txn-ends, and now we get more information about the what's in the extras actual table on DPT

然后看CHECKPOINT-END这里（Andy口误），现在我们可以从它内部的ATT、DPT中获取到更多的信息

58:38 – 58:42

so now we see that there was a t97 that we didn't see in between our checkpoint

so 我们有看到，有个叫 T97的事务，我们并没有在我们的checkpoint 制作期间看到

58:42 – 58:47

so we know that there's some one transaction up above this checkpoint start point ,

so 我们知道，那它（T97）是在这个checkpoint 开始的时间点之前就存在的，只不过并没有提交

58:47 – 58:52

that did some stuff that we may need to go look at as well

那这些东西我们也需要去关注的

58:52 – 58:57

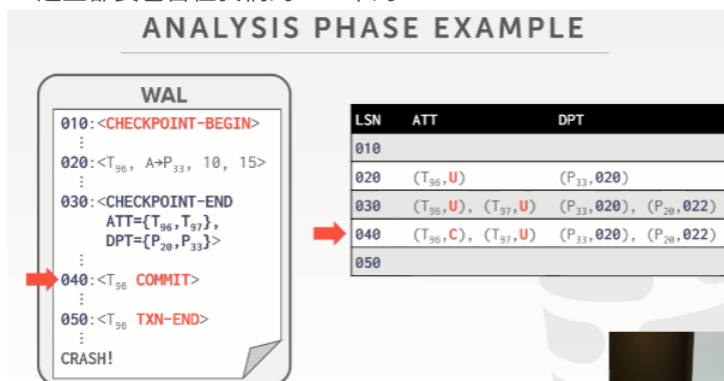
and then there's also a new page 20 that was also modified

然后，在DPT中，我们有看到，也有一个新的page（P20）发生了修改

58:58 – 59:00

so you want to include that in our DPT as well

so这些都要包含在我们的DPT中的



59:01 – 59:03

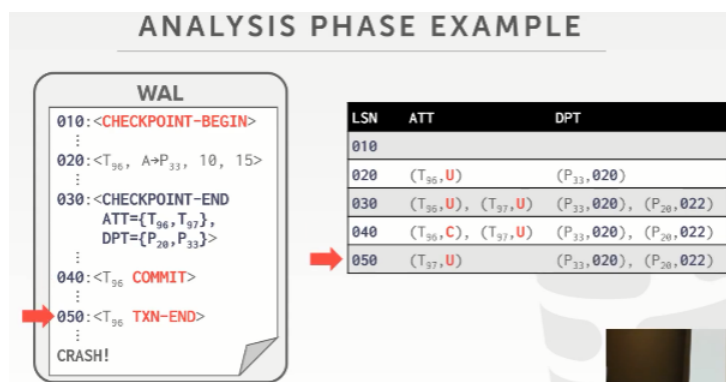
so now we see T 96 commits here

so 这里，我们看到T96 提交了

59:04 – 59:07

So we flip its status to be committing

so 我们将它的状态设定为commit



59:08 – 59:11

and then when we see the transaction and message here

然后，我们在这里看到T96的结束消息TXN-END

59:12 – 59:15

then we know that we can remove it from ATT

然后，我们知道我们可以将T96从ATT中移除了

59:15 – 59:22

but now at the point of the crash, you see that there is t97 still hanging out here with it with a new candidate status

但，现在，在这个点，服务器崩溃了，此时，你有看到T97还在活动状态，它还是属于undo潜在操作对象

59:22 – 59:27

so we know that this transaction made some changes up above our checkpoint

so 我们知道，T97做的修改在我们这个checkpoint 之上

59:27 – 59:30

that we didn't see in our log ,that we need to go back and make sure we reverse

我们在我们的log中并未看到，我们需要回滚并撤销它的修改

59:31 – 59:33

because we don't know whether those pages got written out to just yet

因为我们并不知道这些page是否写出磁盘了

59:36 – 59:36

yes

59:46–59:47

your question is

你的问题是

59:47–59:52

is it possible for the after the analysis phase the ATT and DPT are empty

是否有可能在analysis 阶段之后，ATT 和DPT都是空的

59:52 – 59:55

so therefore you know that nothing there's nothing was dirty

这样，也就意味着不存在dirty page了

59:55 – 59:58

could you just say I'm good ,yes

对的