

なんちゃって Early Lock Release 実装と評価

2012年 1月21日

NEC サービスプラットフォーム研究所

堀川 隆

人と地球にやさしい情報社会を
イノベーションで実現する
グローバルリーディングカンパニー

NECグループビジョン2017

主旨

■ **効果があると云われている Early Lock Release を、勉強がてら PostgreSQL に実装してみました**

- **結果としては「期待外れ」に終わってしまいましたが、
処理フローを調査して改善を試みる、という一連の作業は何らかの参考にして頂ける可能性があると考え、一種の失敗談として披露させていただきます。**

■ **性能評価では非同期コミットやSSDを使うケースも、比較のために測定しました**

- **定量的な視点から、WALのボトルネックを実感できるかと思えます。**

■ **内容**

- **WAL書き込みの処理フローと Early Lock Release の実装方式**
- **性能評価方法と結果**

WALによる性能低下を緩和する方策

■ **非同期commit** (石井達夫, <http://journal.mycom.co.jp/special/2007/postgresql/007.html>, 2007/11/20)

- **期待度大のバージョンアップ - PostgreSQL 8.3の改良点を徹底分析**

8 **非同期コミット(2) - トランザクションの消失を最小限に**

非同期コミットのメリット

これ(cf. fsync=off 設定)に対して、8.3で追加された非同期コミットでは、こうしたリスクを避けることができる。データベースの整合性が失われることはなく、**クラッシュ時にも直近のトランザクションが失われるだけである**。不運にも同期書き込み前にシステムがクラッシュするなどしてトランザクションが失われた場合は、それを再実行すれば良い。

→ **直近とはいえ、commitした(はずの)トランザクションが失われて良いのか？**

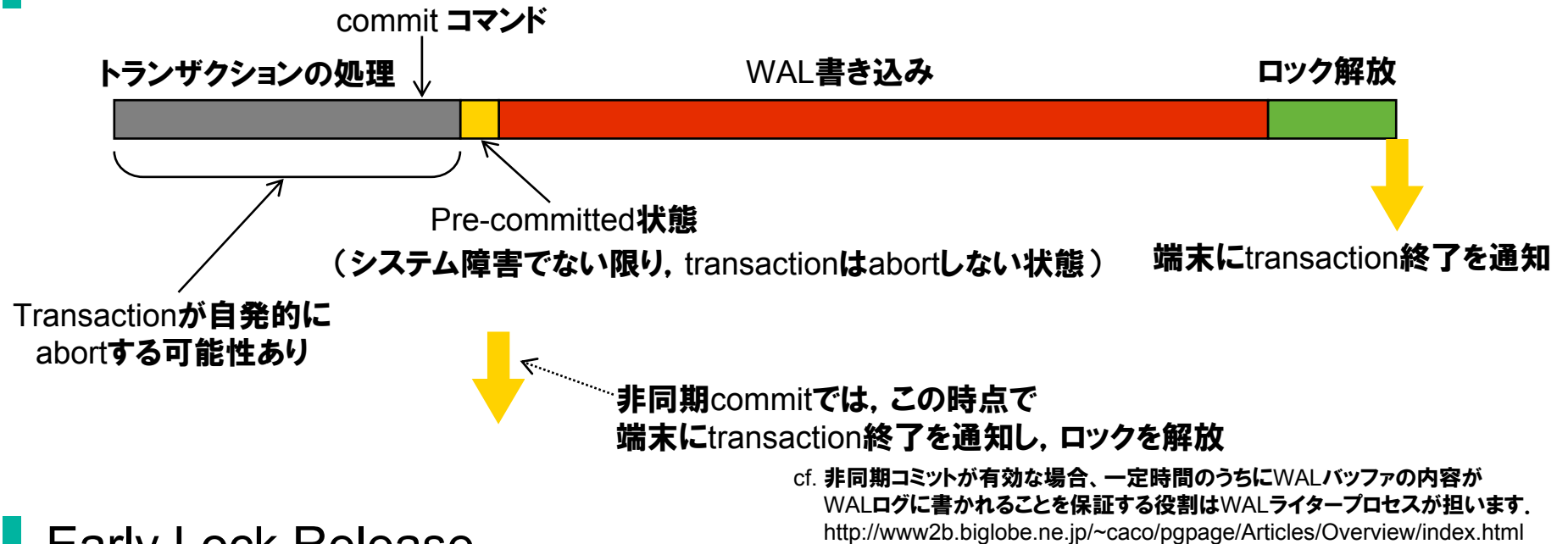
■ **Early Lock Release**

- Johnson, R. and Pandis, I. and Stoica, R. and Athanassoulis, M. and Ailamaki, A., [Aether: A scalable approach to logging](#), Proceedings of the VLDB Endowment VLDB Endowment, Volume 3 Issue 1-2, September 2010.

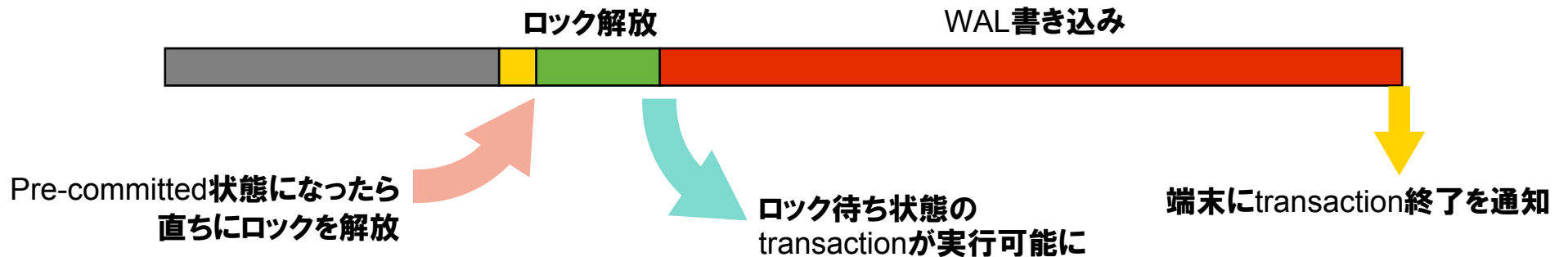
Early Lock Release (ELR) removes log flush latency from the critical path by ensuring that only the committing transaction must wait for its commit operation to complete: ... We hypothesize that this is largely due to the effectiveness of **asynchronous commit** ...

Early Lock Release (イメージ)

従来



Early Lock Release

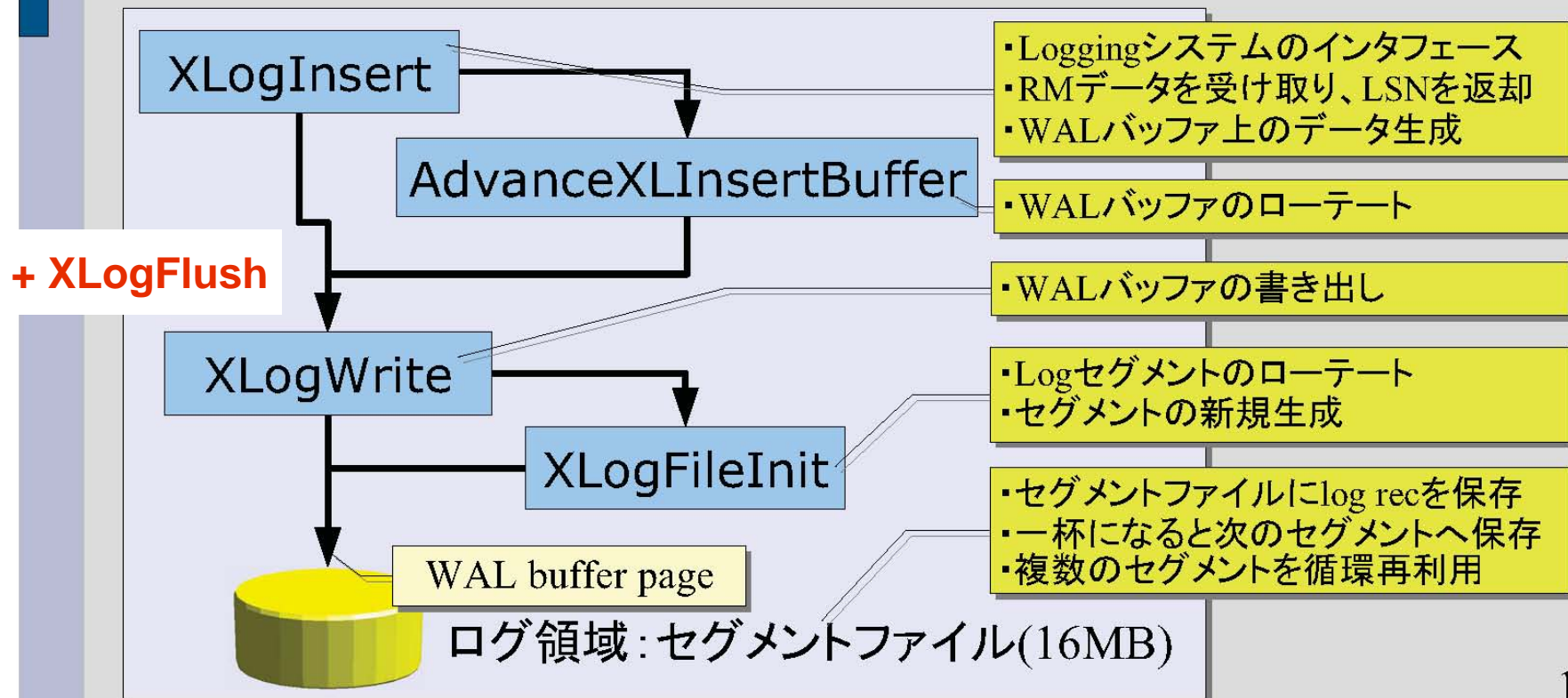


Empowered by Innovation **NEC**

実装

Loggingシステム内部構造

- 関数レベルでの内部構造(下図)
 - 在り処 `src/backend/access/transam/xlog.c`



17

Commitに伴うWAL書き込み動作

Stack trace

(gdb) bt

#0 pg_fdatasync (fd=36) at fd.c:299

#1 0x080c8bcf in issue_xlog_fsync () at xlog.c:6392

#2 0x080beeb4 in XLogWrite (WriteRqst={Write = {xlogid = 0, xrecoff = 6281584}, Flush = {xlogid = 0, xrecoff = 6281584}}, flexible=0 '¥0', xlog_switch=0 '¥0') at xlog.c:1614

#3 0x080bf2e9 in XLogFlush (record={xlogid = 0, xrecoff = 6281584}) at xlog.c:1741

#4 0x080b8a1c in RecordTransactionCommit () at xact.c:949

#5 0x080b93dd in CommitTransaction () at xact.c:1675

#6 0x080b9d3a in CommitTransactionCommand () at xact.c:2274

#7 0x0824b647 in finish_xact_command () at postgres.c:2322

#8 0x082493df in exec_simple_query (query_string=0x84ac870 "create table abc(a int);") at postgres.c:1017

#9 0x0824d15f in PostgresMain (argc=4, argv=0x8454520, user_name="postgres") at postgres.c:3572

#10 0x08217982 in BackendRun (port=0x8467d18) at postmaster.c:3207

#11 0x08216f0a in BackendStartup (port=0x8467d18) at postmaster.c:2830

#12 0x08214928 in ServerLoop () at postmaster.c:1274

#13 0x08214335 in PostmasterMain (argc=1, argv=0x8451578) at postmaster.c:1029

#14 0x081b6707 in main (argc=1, argv=0x8451578) at main.c:188

関係するのはこの辺り

XLogFlushの直前で
XLogInsertを実行

トランザクション処理本体

<http://d.hatena.ne.jp/taedium/20090215/p1>

全体処理の俯瞰 主にCommitTransaction()

```
static void  
CommitTransaction(void)  
{
```

```
...
```

```
latestXid = RecordTransactionCommit();
```

```
ProcArrayEndTransaction(MyProc, latestXid);  
CallXactCallbacks(XACT_EVENT_COMMIT);  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_BEFORE_LOCKS,  
                      true, true);  
  
AtEOXact_Buffers(true);  
AtEOXact_RelationCache(true);  
AtEarlyCommit_Snapshot();  
AtEOXact_Inval(true);  
smgrDoPendingDeletes(true);  
AtEOXact_MultiXact();  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_LOCKS,  
                      true, true);  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_AFTER_LOCKS,  
                      true, true);
```

```
...
```

```
}
```

WAL書き込み

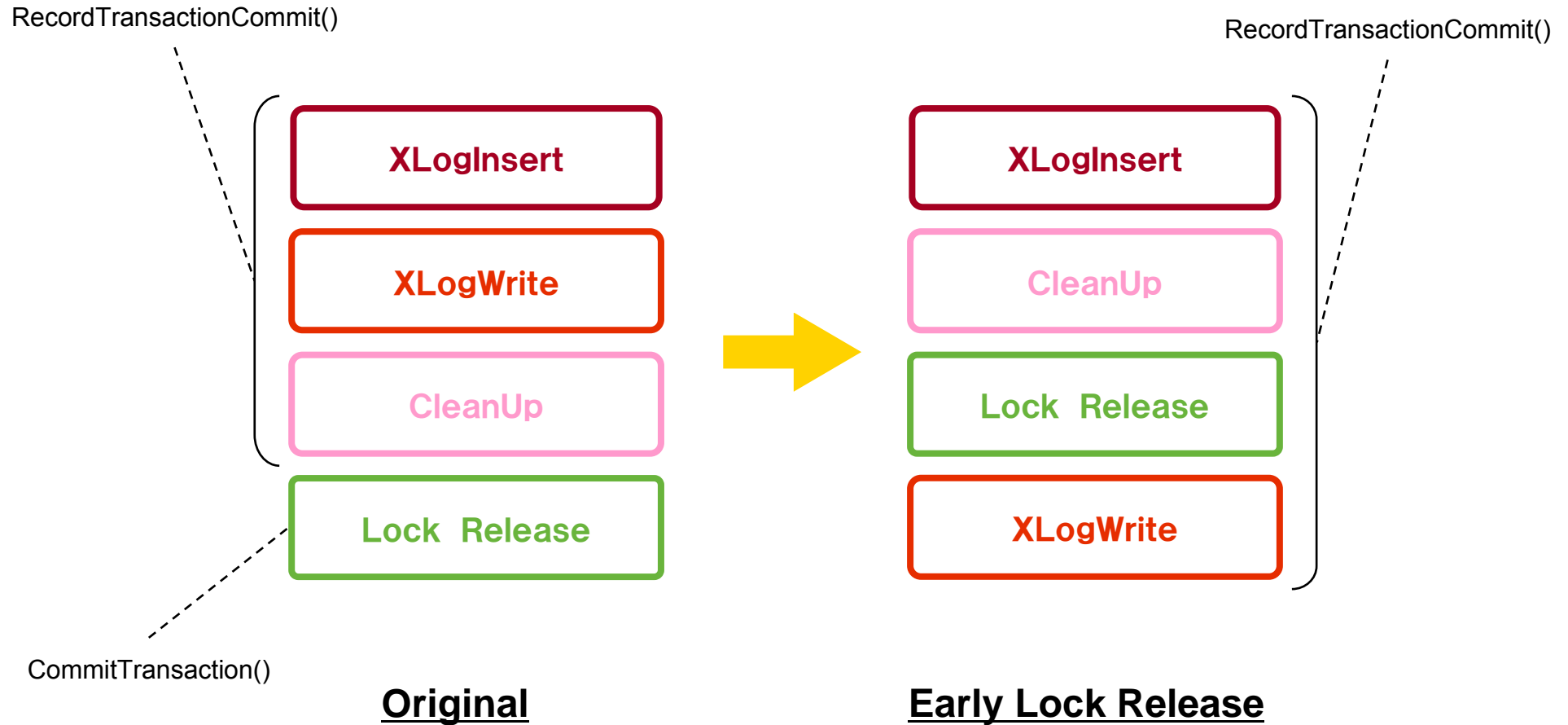


入れ替えたい

latestXidの依存関係により、
単純には入れ替えられない

ロック解放

ELRの実装案(その1)



XLogWrite処理の精査

```
if (XactSyncCommit || forceSyncCommit || haveNonTemp)
{
```

```
    if (CommitDelay > 0 && enableFsync &&
        CountActiveBackends() >= CommitSiblings)
        pg_usleep(CommitDelay);
```

```
    XLogFlush(XactLastRecEnd);
```

XLogWrite

```
        if (markXidCommitted)
            TransactionIdCommitTree(xid, nchildren, children);
```

CleanUp

```
    }
else
```

```
    {
        XLogSetAsyncXactLSN(XactLastRecEnd);
```

```
        if (markXidCommitted)
            TransactionIdAsyncCommitTree(xid, nchildren, children, XactLastRecEnd);
    }
```

非同期コミット時の処理

```
    if (markXidCommitted)
    {
        MyProc->inCommit = false;
        END_CRIT_SECTION();
    }
```

XLogWrite

実装内容

```
if (markXidCommitted)
    TransactionIdCommitTree(xid, nchildren, children);
```

CleanUp

```
latestXid = TransactionIdLatest(xid, nchildren, children);
XactLastRecEnd.xrecoff = 0;
cleanup:
    if (rels)
        pfree(rels);
```

CleanUp

Lock Release

```
if (XactSyncCommit || forceSyncCommit || haveNonTemp)
{
    if (CommitDelay > 0 && enableFsync &&
        CountActiveBackends() >= CommitSiblings)
        pg_usleep(CommitDelay);

    XLogFlush(XactLastRecEnd);
```

XLogWrite

```
if (markXidCommitted)
{
    MyProc->inCommit = false;
    END_CRIT_SECTION();
}
```

XLogWrite

```
XactLastRecEnd.xrecoff = 0;
```

実装案(その2) WAL書き込みはWalWriterに任せる

```
static void  
CommitTransaction(void)  
{
```

```
...  
latestXid = RecordTransactionCommit();
```

```
ProcArrayEndTransaction(MyProc, latestXid);  
CallXactCallbacks(XACT_EVENT_COMMIT);  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_BEFORE_LOCKS,  
                      true, true);  
  
AtEOXact_Buffers(true);  
AtEOXact_RelationCache(true);  
AtEarlyCommit_Snapshot();  
AtEOXact_Inval(true);  
smgrDoPendingDeletes(true);  
AtEOXact_MultiXact();  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_LOCKS,  
                      true, true);  
ResourceOwnerRelease(TopTransactionResourceOwner,  
                      RESOURCE_RELEASE_AFTER_LOCKS,  
                      true, true);
```

```
...
```

```
}
```

非同期Commitと
同じ動作をさせる

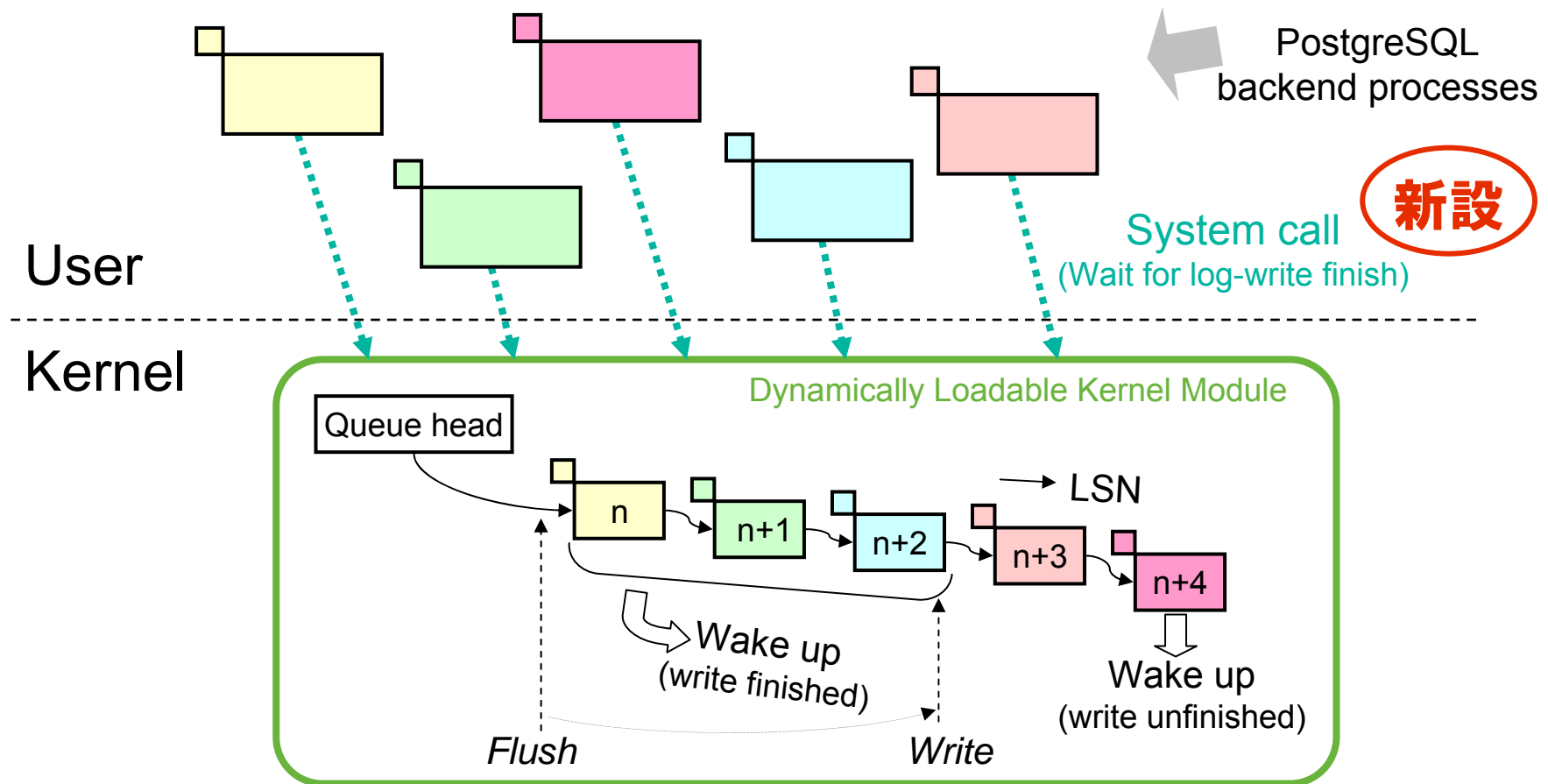
+ WalWriterDelay
を最小値にする

ロック解放

XlogInsertしたWALの
書き込み完了を待つ

Linux カーネルモジュールを作成

- Specialized process wake-up mechanism implemented by DLKM
 - Backend processes are woken up according to their LSN



実装内容

```
bool      XactSyncCommit = false; xacct. c

static TransactionId
RecordTransactionCommit(void)
{
    ...
    savedXactLastRecEnd = XactLastRecEnd;
    ...
}

static void
CommitTransaction(void)
{
    ...
    latestXid = RecordTransactionAbort(false);
    ...
}

```

← savedXactLastRecEndまでのWAL
書き込み完了を待つsystem call発行

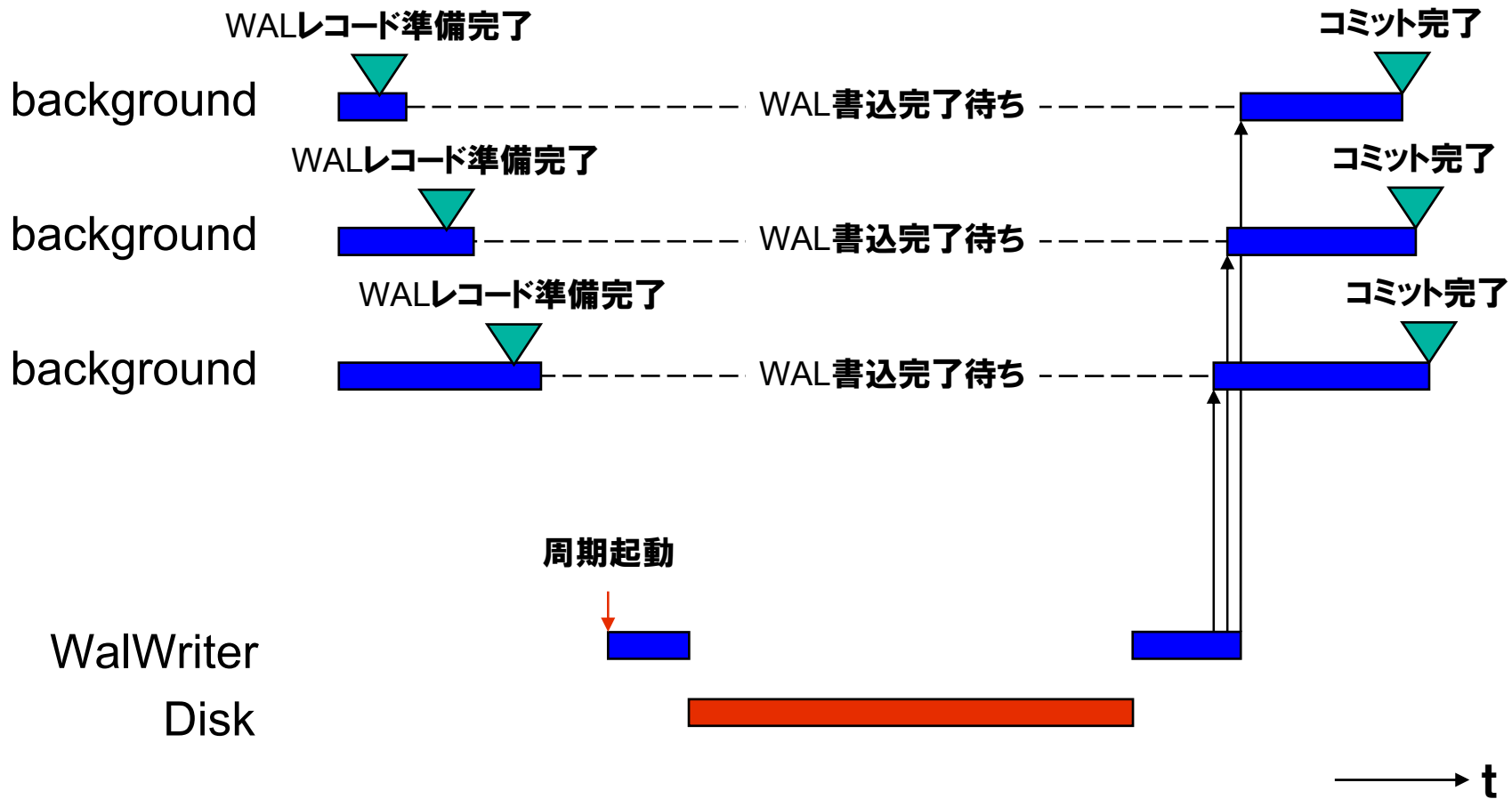
```
static void xlog. c
XLogWrite(XLogwrtRqst WriteRqst, bool flexible, bool xlog_switch)
{
    ...

    Write->LogwrtResult = LogwrtResult;
}

```

← LogwrtResult.WriteまでのWAL
書き込み完了を通知する
system call発行

動きをよく考えてみると...



LogWriter っぽい

Performance

評価方法

対象

- PostgreSQL (baseは9.0.4, NUM_BUFFER_PARTITIONSは256に変更)
 - normal
 - normal - **非同期**commit (postgresql.conf にて synchronous_commit = off とする)
 - Early lock release
- **WAL記録用デバイス**
 - Hard Disk
 - Solid State Drive

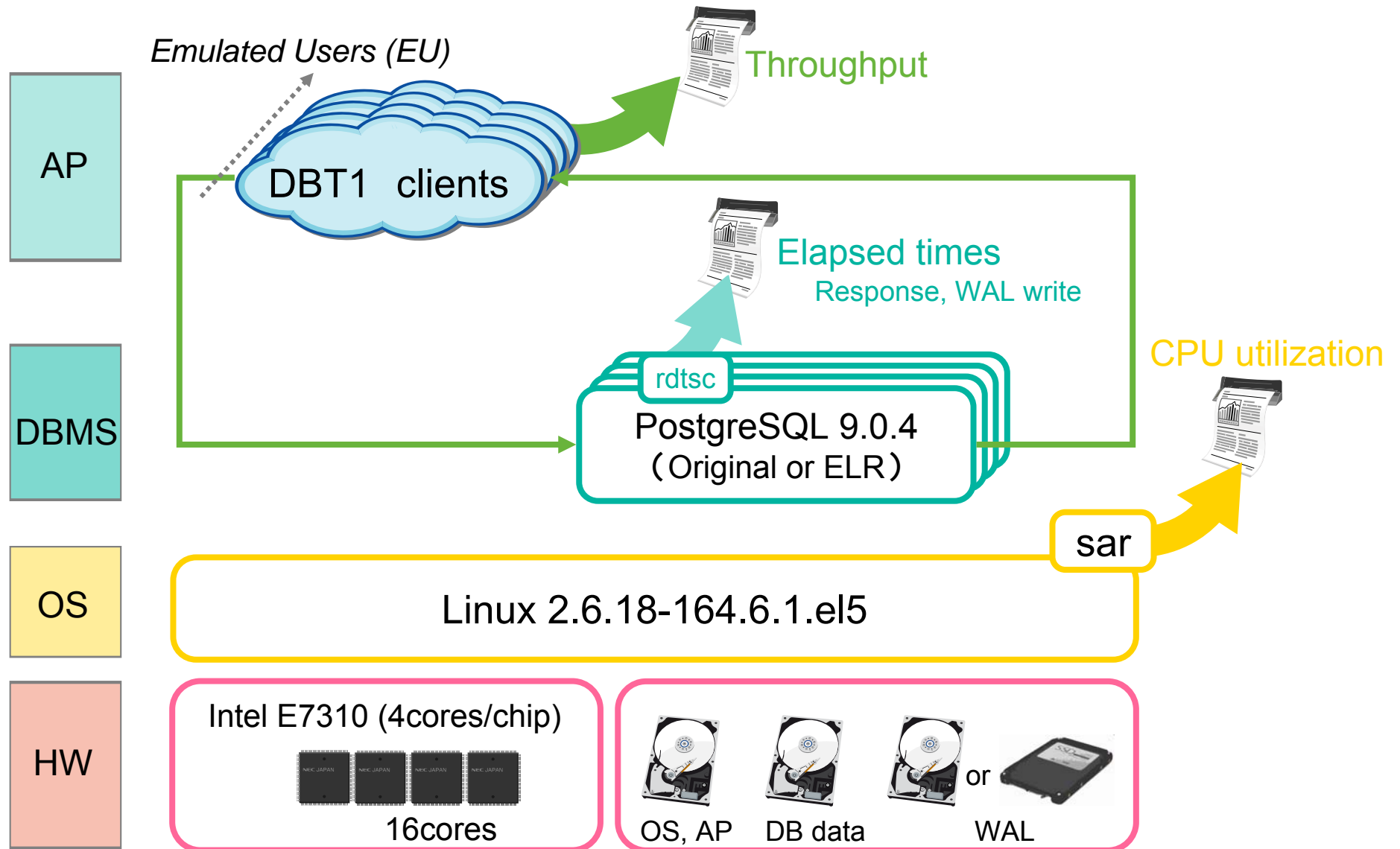
ベンチマーク

- DBT-1 (pxgc用を流用)
- **多重度: 16 → 32**

内容: 負荷を変化させて下記を測定

- **スループット, CPU使用率**
- **exec_simple_query(), XLogFlush()の実行回数と実行時間**

システム構成 & 測定項目



(補足) rdtsc による時間計測について

概要

- 機能: CPU内蔵のtime stamp counter(TSC)値を取得
- 分解能: CPUクロック(サブナノ秒オーダ)
- コア毎にTSCがある

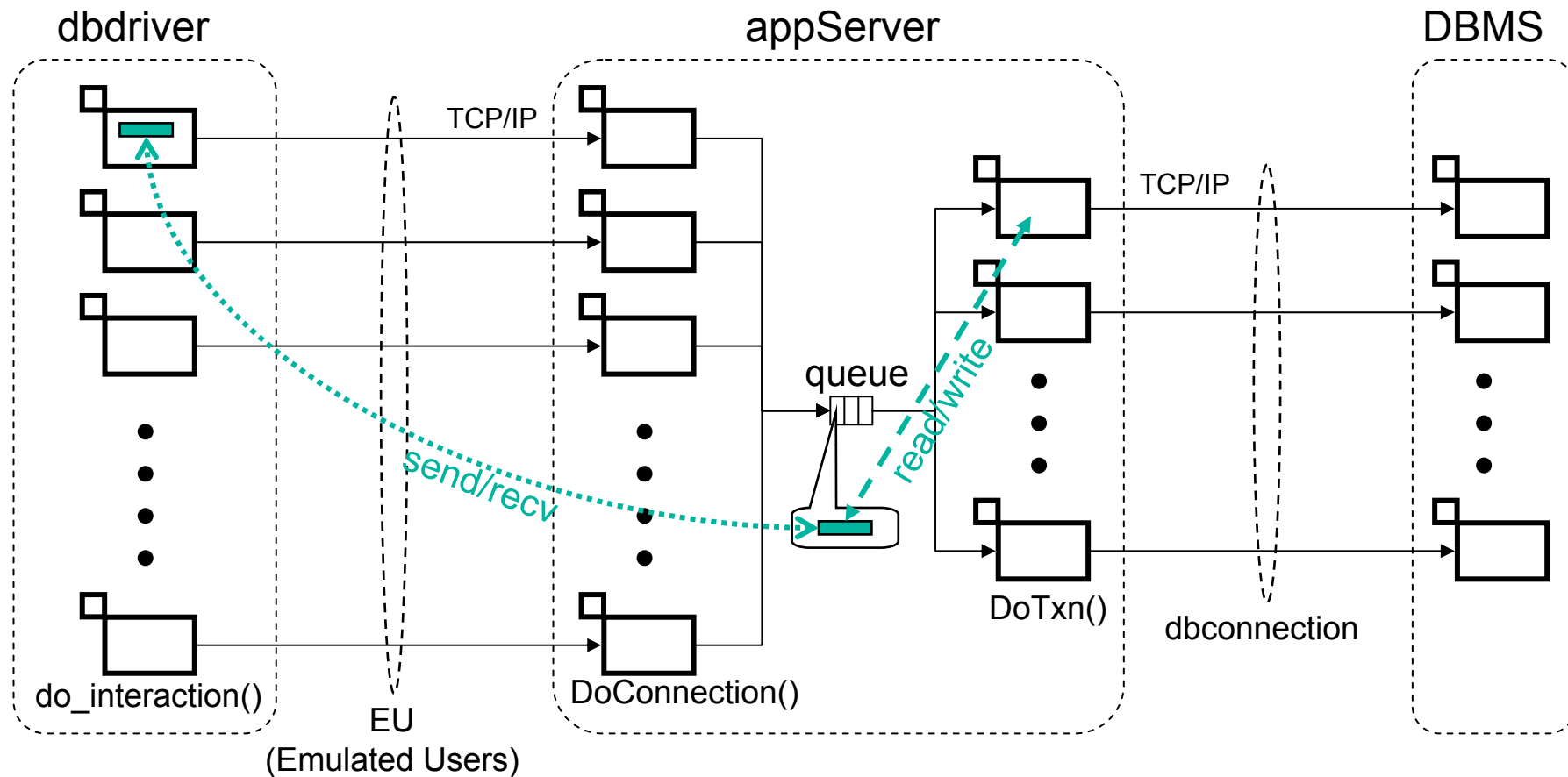
問題点

- マルチコア(またはマルチプロセッサ)環境では、それぞれのTSCの値は(厳密に言う
と)同じではないため注意が必要

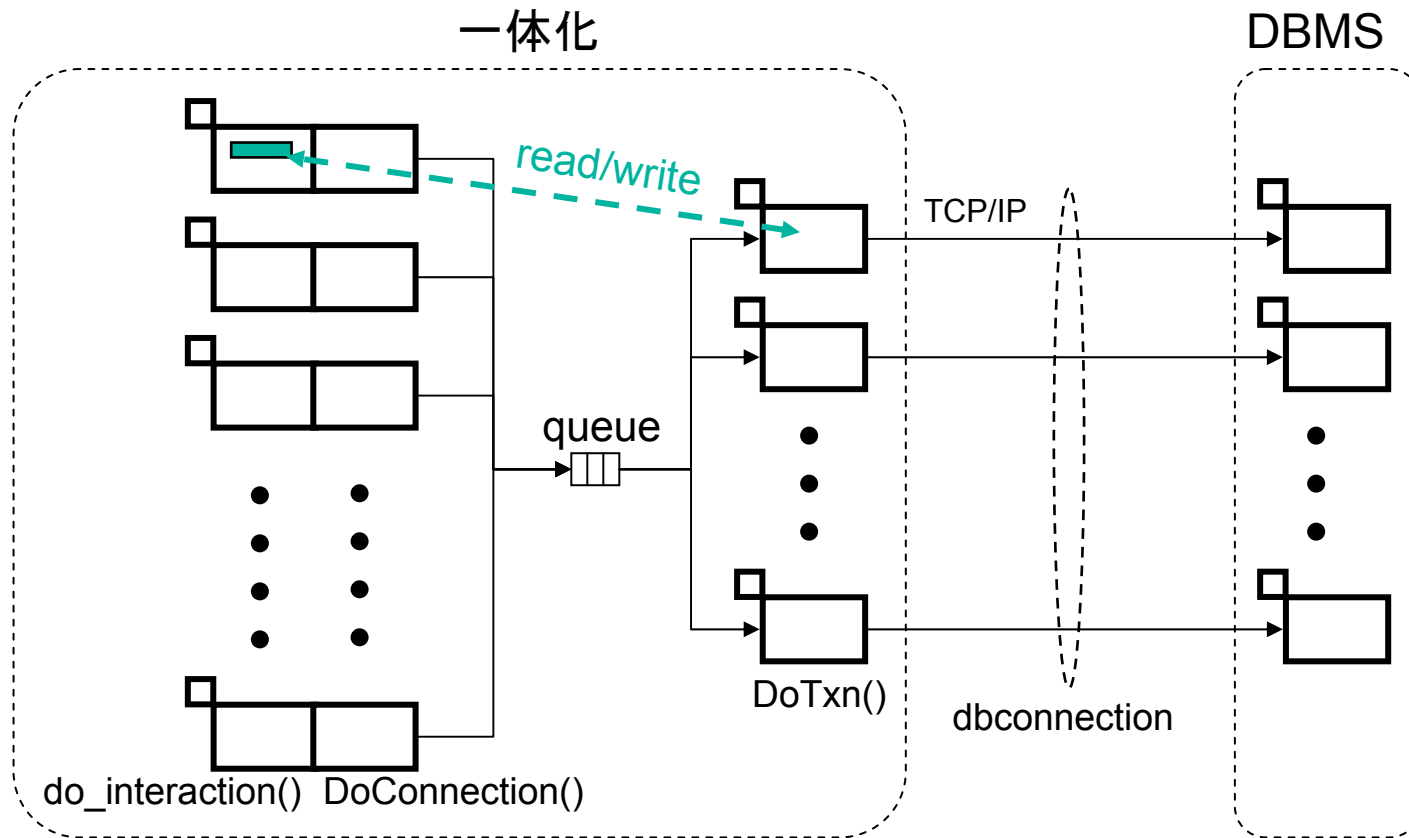
対策: (今回は)気にしない

- Linux カーネルは 2 目以降の CPU をブートする際に、1 番目の CPU の TSC に
2 番目以降の CPU の TSC を同期させようとする。
 - 誤差は1 μ 秒未満 (↓ Linuxのboot log)
Intel(R) Xeon(R) CPU E7310 @ 1.60GHz stepping 0b
CPU 1: Syncing TSC to CPU 0.
CPU 1: synchronized TSC with CPU 0 (last diff 0 cycles, maxerr 978 cycles)
- 多数のデータを集計するので、±の誤差は平均化される
 - core A → core B と core B → core A のデータはほぼ同数になるはず。

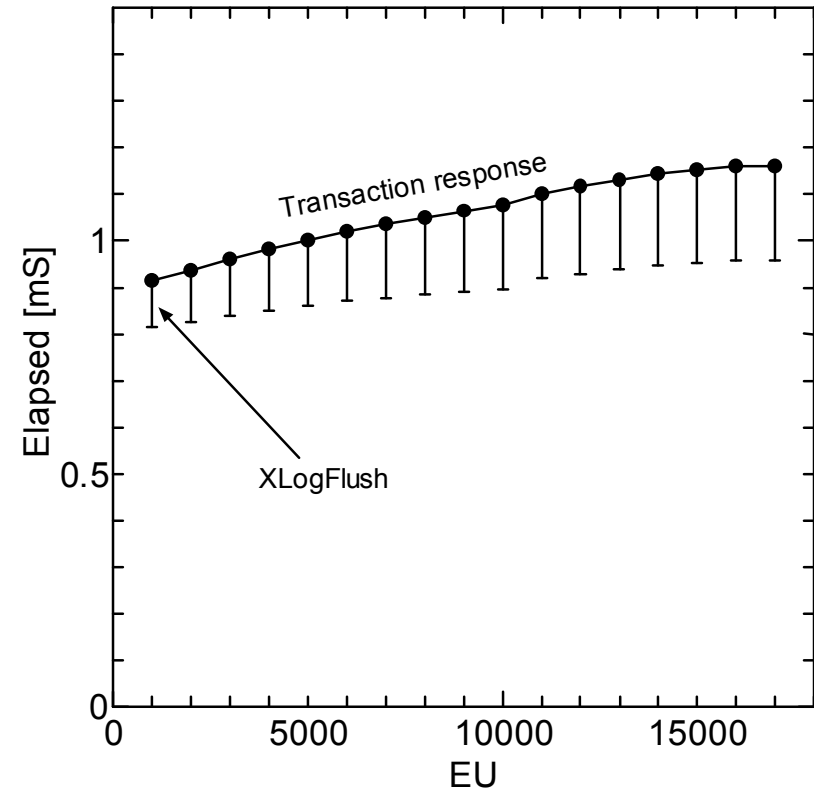
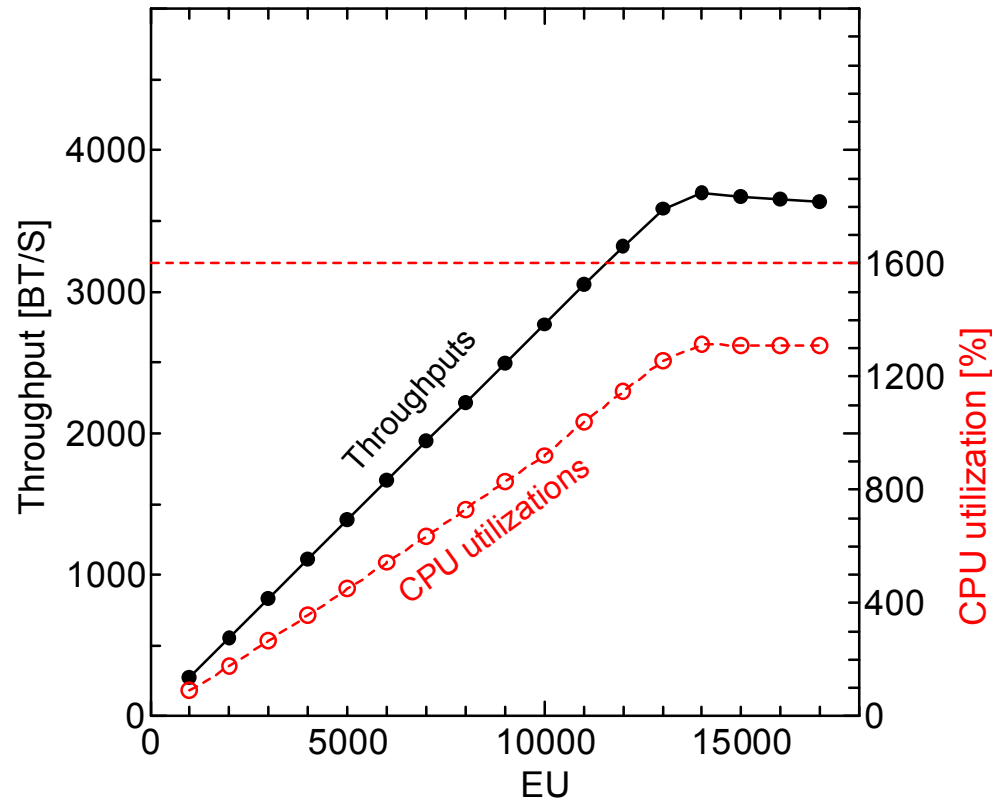
(余談) DBT-1負荷発生ツールのチューニング before



(余談) DBT-1負荷発生ツールのチューニング after

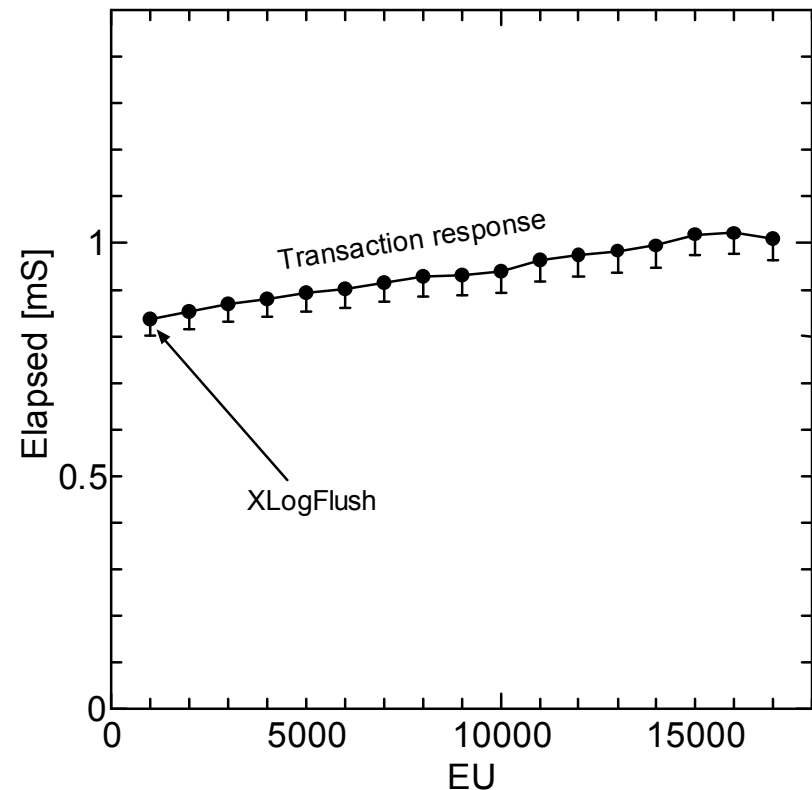
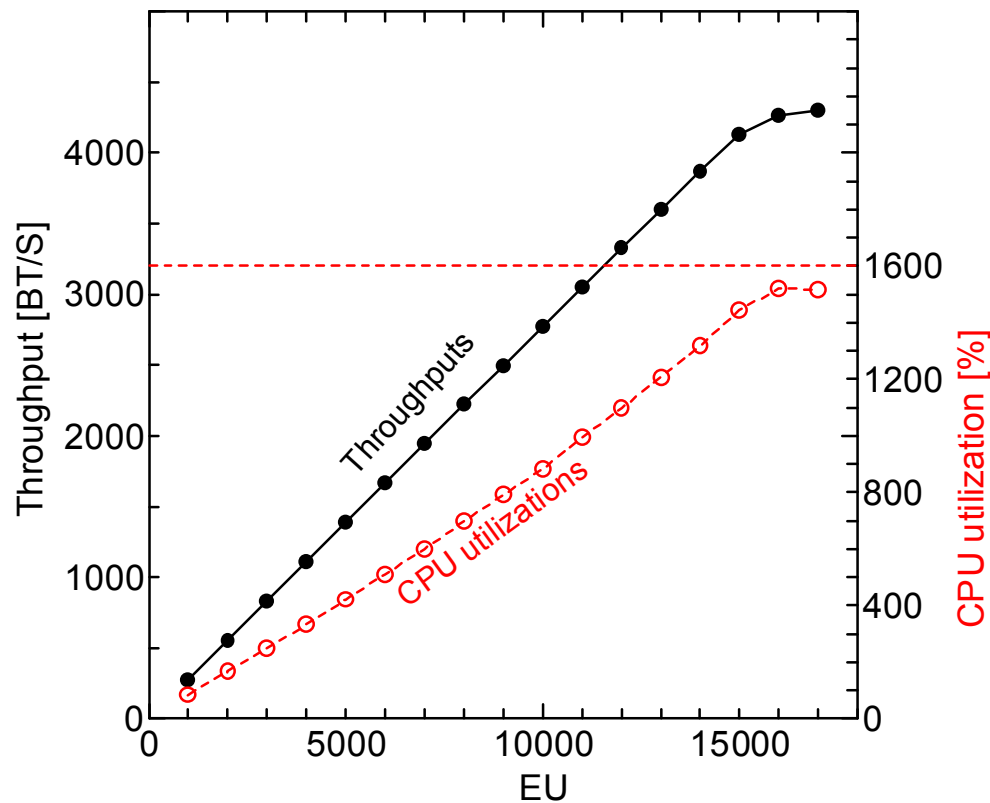


測定結果 (original)



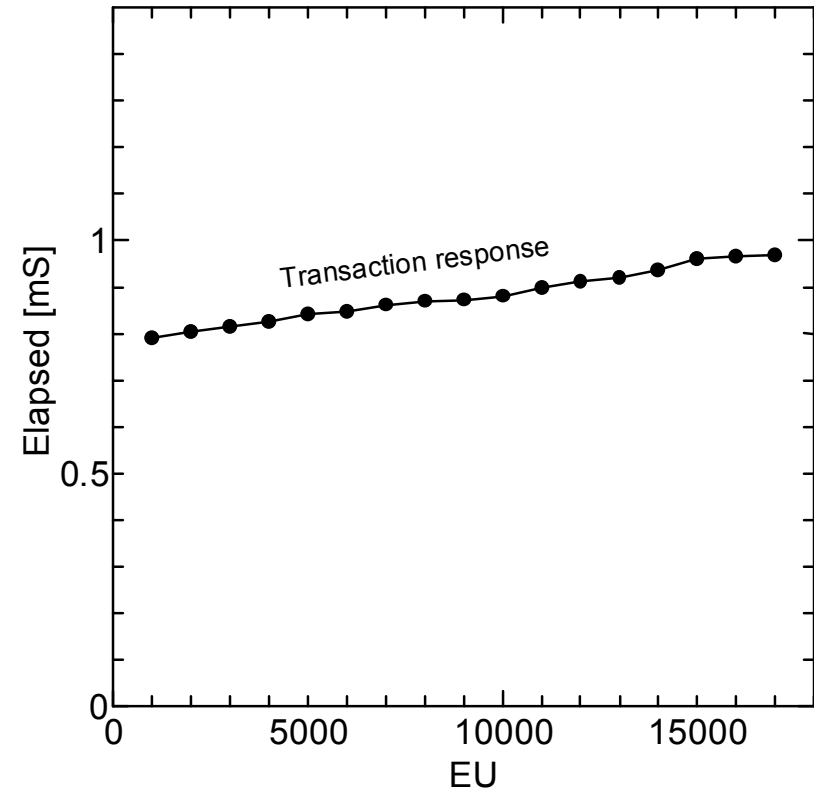
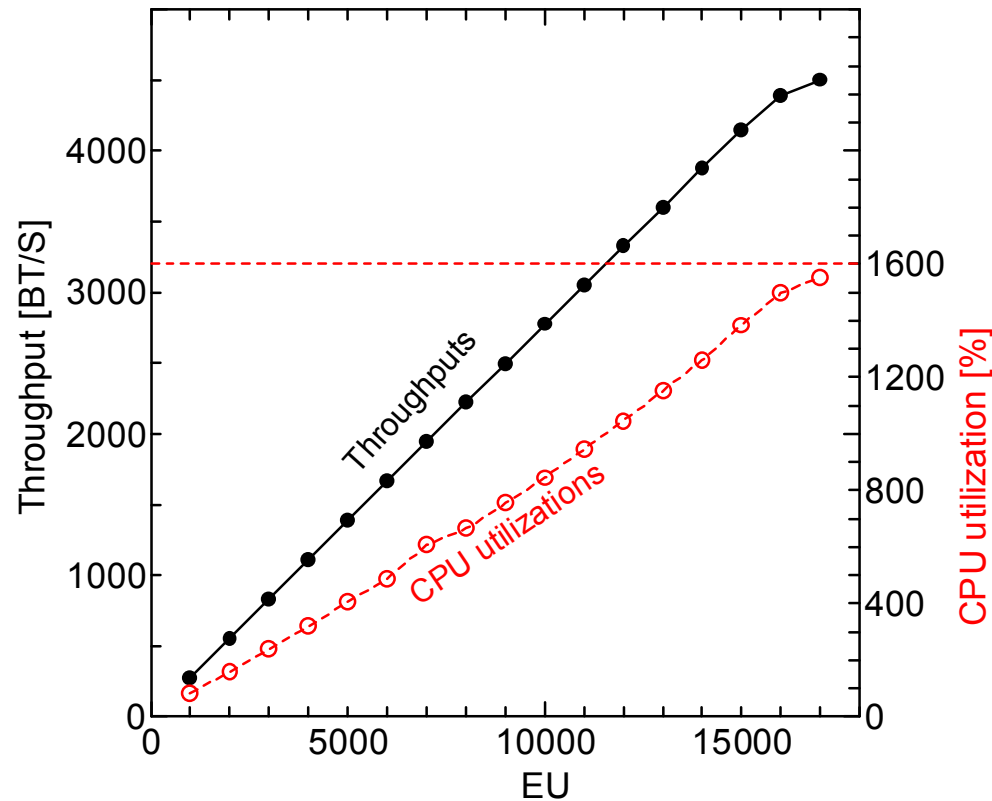
スループットは3700弱で飽和, CPU使用率も1300%程度で頭打ち
XLogFlush()がレスポンスに占める割合は10~17%

測定結果 (original, WALをSSDにwrite)



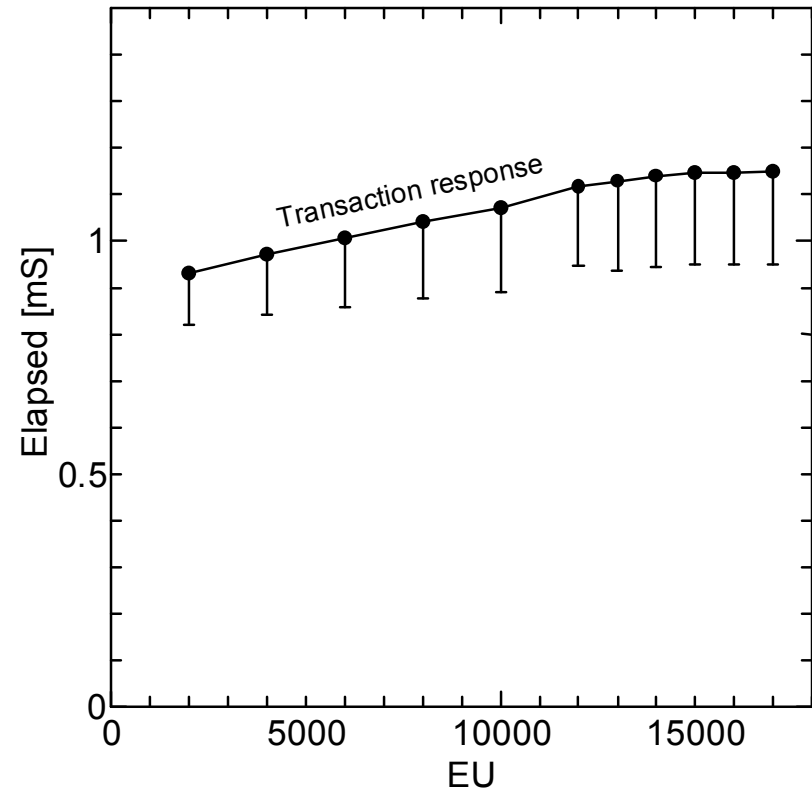
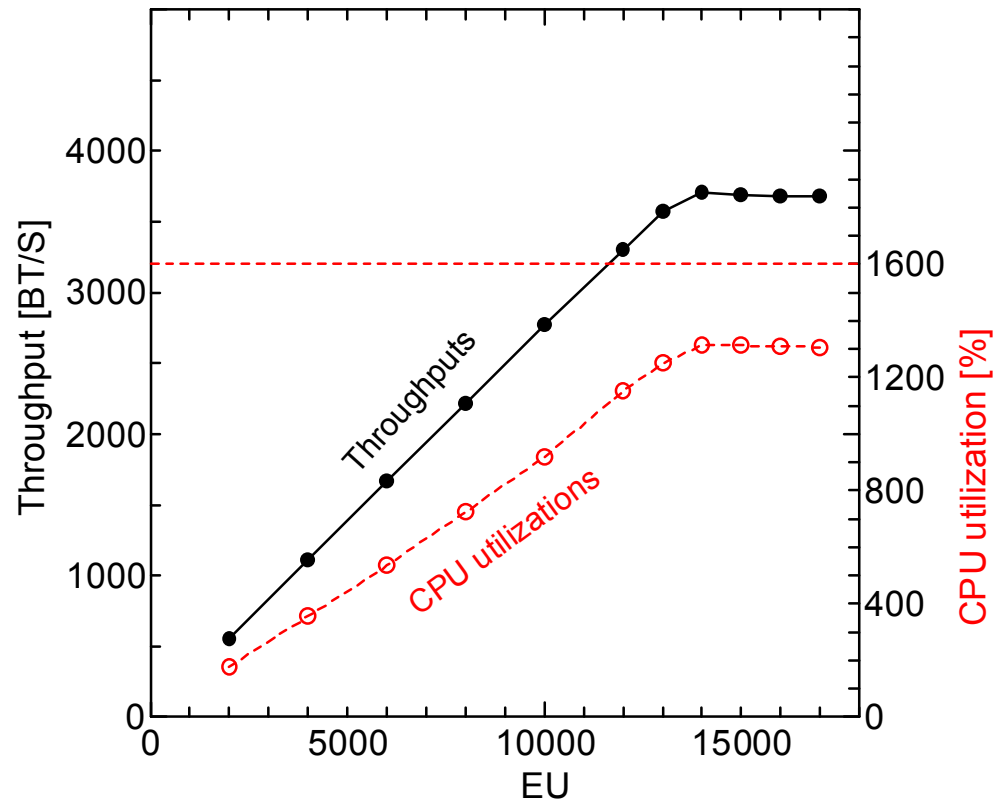
スループットの飽和点は4300まで向上, CPU使用率も1500%まで上昇
XLogFlush()がレスポンスに占める割合は4~4.5%に低下

測定結果 (original, 非同期commit, WALはHDへwrite)



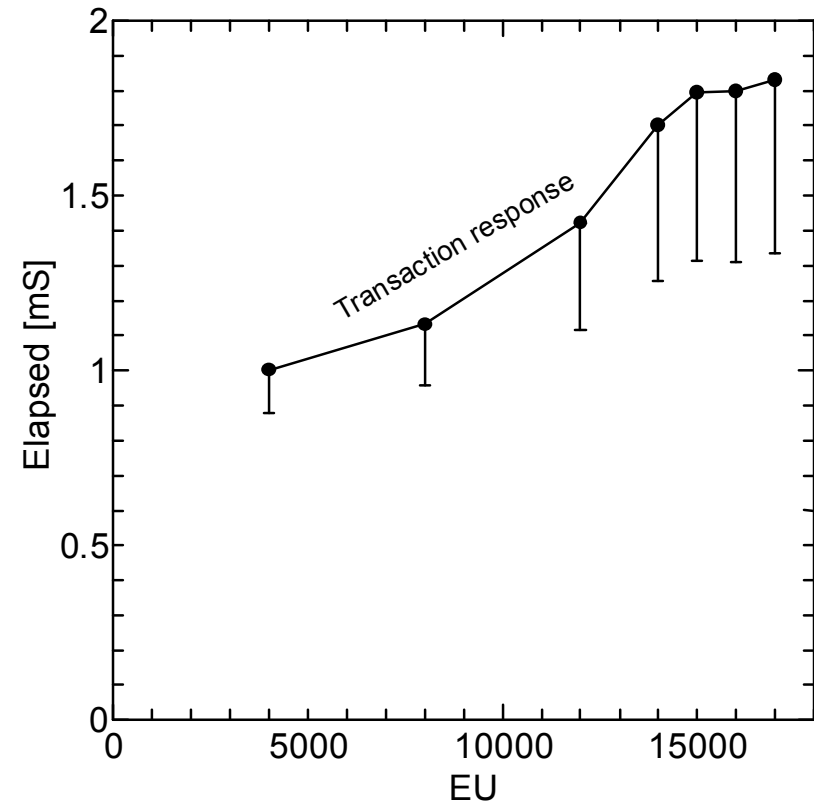
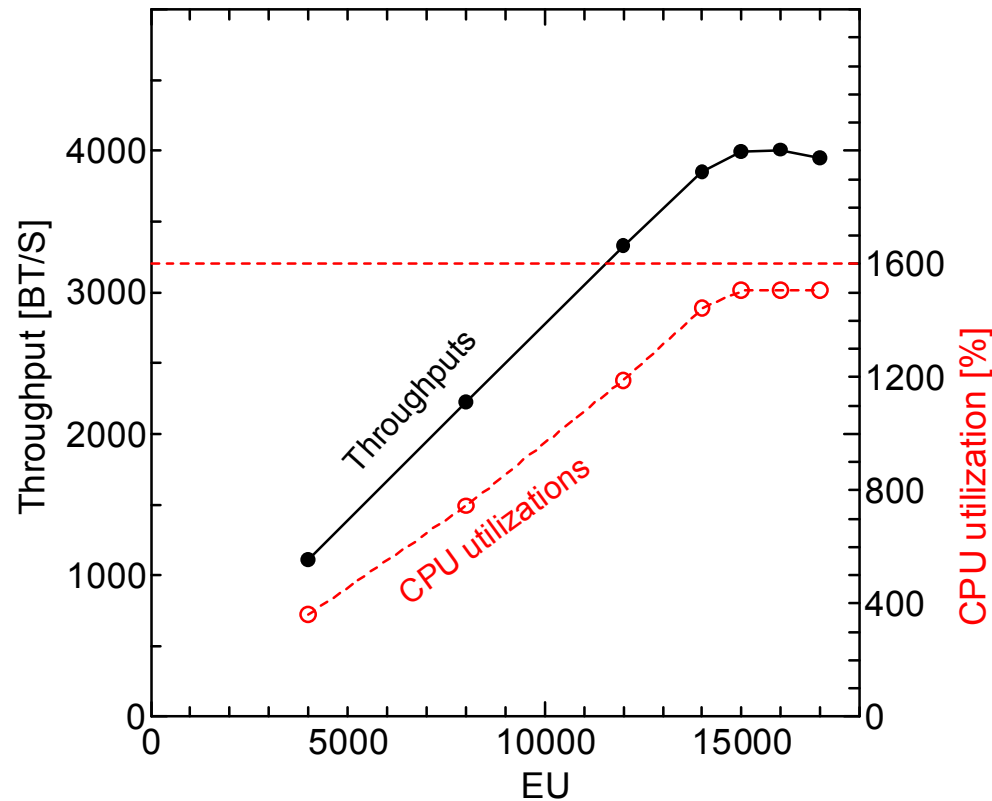
スループットの飽和点は4500弱, CPUはほぼ使い切る
XLogFlush()の影響はnegligible

測定結果 (Early lock release 1)



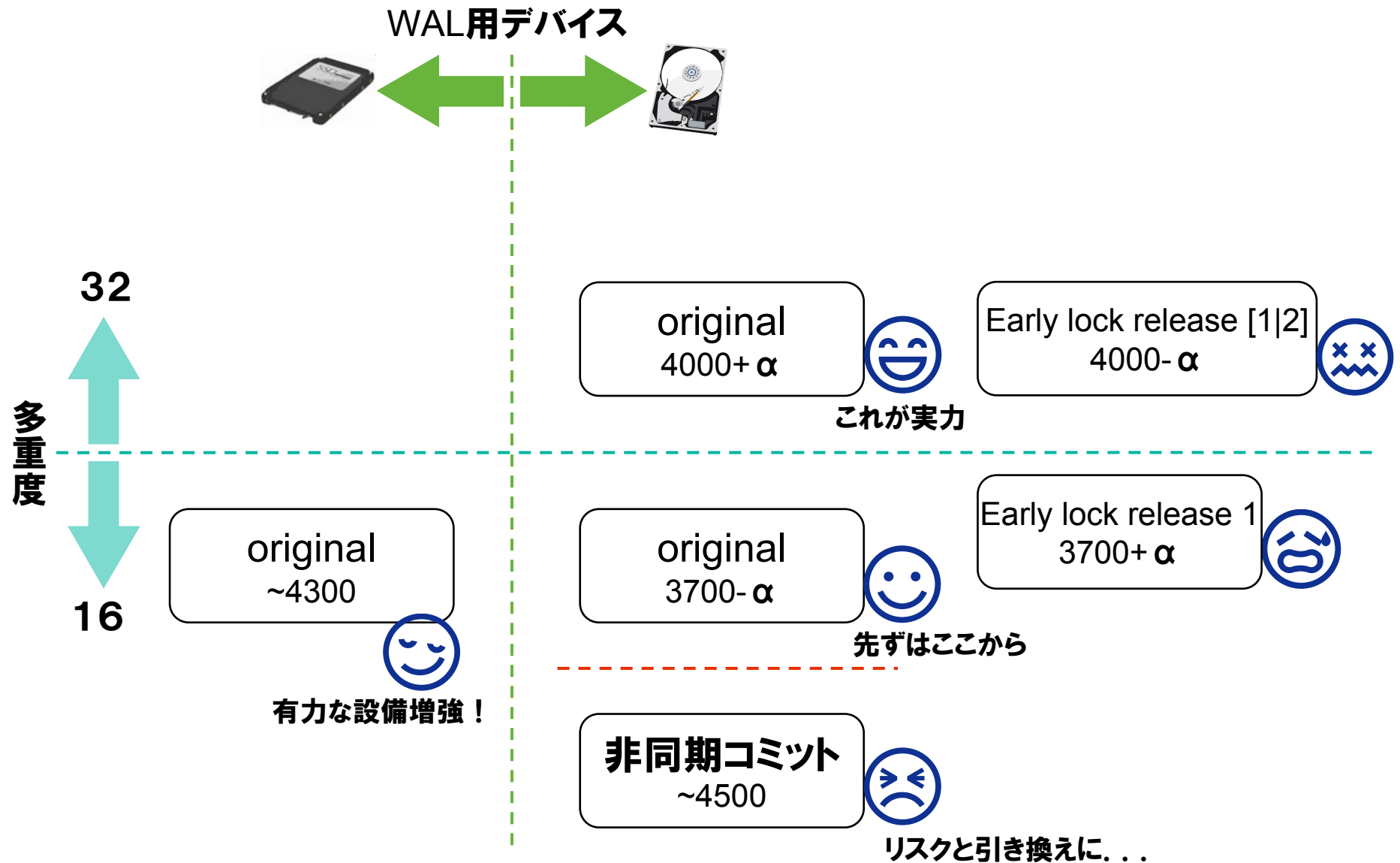
Originalとほぼ同じ結果 orz

測定結果 (original, 多重度を32に増やす)



スループットは4000程度は出るようになり, CPU使用率も概ね飽和
XLogFlush()がレスポンスに占める割合は, 高負荷領域で25%程度に増加

測定結果 俯瞰



まとめ

PostgreSQL への Early lock release(ELR) の実装を試みた

- 論文で言及されているような(非同期コミットと同等)効果はなかった orz
(多重度の最適な値への調整は, 現状, 未実施)
- 試行錯誤の処理フローを調査して改善を試みる, という作業の流れを(何らかの)参考にして頂ければ幸いです.

その他

- SSDをwal用diskとして使用する効果は大
(各所で言及されていると思いますが...)
- 非同期コミットは最大の効果
 - 障害発生直前のトランザクションはデータ復旧を諦める, という犠牲と引き換え

Empowered by Innovation

NEC