# マルチプロセッサシステムにおける
# スケーラビリティーボトルネックの分析手法

NEC 共通基盤ソフトウェア研究所

堀川　隆

## はじめに

近年のITシステムは、マルチコアCPUの普及に伴ってCPU資源が強化されていますので、CPU以外の要素が性能上のボトルネック（以下、ボトルネック）になることがあります。特に、排他制御のためのロック（論理資源）が新たなボトルネックになると、物理資源（CPU, Disk, Network, Memory）を強化するという単純な対策では性能は向上しない、すなわちシステムにスケーラビリティーがないことから、その対応は困難になりがちです。

このチュートリアルでは、このような状況への体系的な対処として、イベント・トレースを利用した方法を紹介します。事例として、この方法をデータベース処理のベンチマーク・プログラムに適用し、1)マルチコア・マシンの処理時間に大きく影響していたロックを特定できたこと、2)プログラム構造を見直すことでスループット性能を向上できたこと、を紹介します。

## マルチプロセッサ特有のボトルネック

プログラムにおいて複数のプロセッサが同時に実行すると処理結果が不正となるプログラム部分（クリティカル・セクション）の保護は、ロックなどによる排他制御によって実現されています。クリティカルセクションを実行しようとして複数のプロセッサが競合すると、ロックを獲得できなかったスレッドはその解放を待ち合わせるため、必要な処理が進まなくなります。これが、ロックがボトルネックになるメカニズムです。

排他制御のバリエーションには、1)保護対象データの読み込み操作については同時実行を可能とするためのRead-Write Lock（rw_lock）、2)並行実行する処理（スレッド）の数を性能上の観点から制限するためのもの（conc_lock）、があります。

複数のプロセスやスレッドが多数のデータを共有して行う処理が基本となっているデータベース管理システム（DBMS）の場合、それらのデータへのアクセスを制御する排他制御の影響が大きくなり、効率的な並列実行が阻害されがちです。IT システムの重要な土台であるDBMSは、排他制御によるボトルネックが発現する典型例といえます。

# イベント・トレースによるボトルネックの分析

## DBMS プログラムへのプローブ設置

イベント・トレースは、測定対象システムの動作を時系列として記録したデータで、測定点（プローブ）を設置したプログラムを実行させて採取します。ここでは、物理資源の使用状況を把握する目的でカーネル内に設置したプローブに加え、各種ロックについての待ち状況を把握する目的で、測定対象DBMSのmysql[1]内部にもプローブを設置しました。プローブを設置した関関数を図１に示します。

```
mutex: mutex_enter_func()
rw_lock: rw_lock_s_lock_func(),
         rw_lock_x_lock_func()
conc_lock: innodb_srv_conc_enter_innodb()
```

図１　プローブを設置した関数

## ベンチマークの実行方法と結果

ここでは、16 プロセッサ（4core CPU ×4）のマシンによる DBT-1 ベンチマーク・プログラムの実行を測定しました。mysql や DBT-1 の構築やベンチマーク実行方法、および、mysql の実行パラメータの設定は、OSS iPedia にある資料に沿って実施しました。mysql の実行パラメータの内、並行実行処理の上限値（innodb_thread_concurrency）は、ロック待ち状況に大きく影響するので、6, 8, 12, 20, 30, 50, 無制限と変化させて測定を実施しました。また、性能上の観点から、予め立ち上げておくスレッド数（thread_cache_size）を 200 としました。

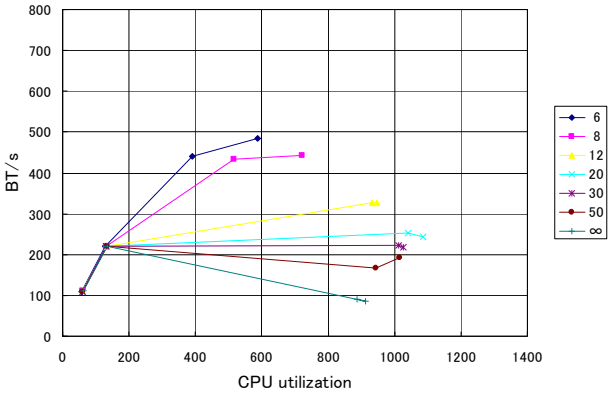


図２　CPU 使用率とスループット

ベンチマーク実行結果として、innodb_thread_concurrency 毎にプロットした CPU 使用率（x 軸）とスループット（y 軸）の関係を図２に示します。CPU 使用率は、1CPU 分を 100%としていますので、16CPU では最大 1600%になります。スループット値は、DBT-1 の結果集計プログラムが出力する BT/s（BogoTransation/秒）です。図からは、並行実行処理の上限値を 6 や 8 に制限した場合を除き、CPU 使用（消費）がスループット向上に役立っていないことが分かります。

## ロック待ち時間の分析

スペースの関係上、並行実行処理数が無制限のケース、および最も高いスループットを示したケース（並列実行処理の上限値は 6）の２種類の結果を図３に示します。

---

[1] mysql のバージョンは 5.0.75、使用したストレージエンジンは innodb

並行実行処理数の上限値が有限の場合（図３左）、conc_lock の待ち時間が大半となりますが、本質的なボトルネックは、conc_lock によって保護された処理に隠されていると考えられます。並行実行処理数が無制限のケース（図３右）は、その本質的なボトルネックを明確にするためのものです。これより、rw_lock である btr_search_latch の状態を管理する変数への排他アクセスを実現するための mutex に関する競合が最大のボトルネックとなっていることが分かりました。

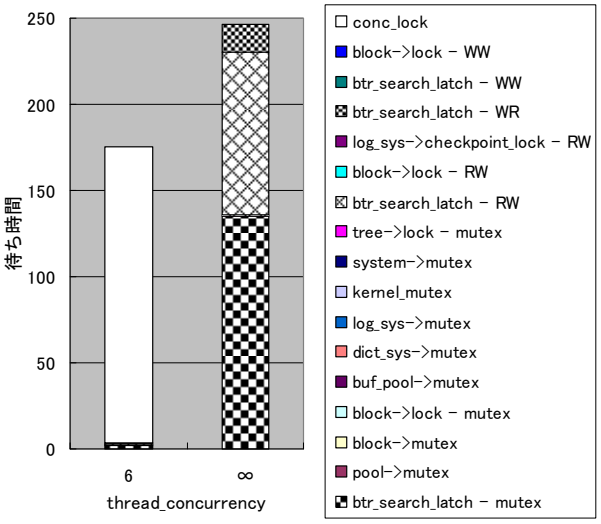


**図３　ロック待ち時間の内訳**

凡例には、待ち時間がゼロでなかった lock を示しています。大半の lock は待ち時間がゼロに近いため、棒グラフでは存在していないように見えます。

# チューニング実施および効果の検証

## チューニング内容 DBMS

問題となっている mutex で保護されている変数は、幸いにも、1)アクセス方法が単純、2)アクセスしている箇所は少ない、3) データのサイズが小さい、という特徴がありました。このため、rw_lock に関する mutex を不要とする改造は容易と考え、Compare And Swap（CAS）命令を利用した lock-free な方式を実装しました。

具体的には、1)mutex による排他制御を行ってアクセスする rw_lock 構造体中のメンバをメンバを CAS 操作可能な 64 ビットに集約、2)rw_lock 変数（lock）をアクセスする部分を図４に示す構造に変更、という書き換えを行ないました。なお、1)に際して、メンバの 1 つでる writer_thread は 64bit データ（ポインタ）の pthread_self()から 32bit データの gettid()を使用するように変更しています。

```
mutex_enter(&(lock->mutex));
  lockのrw_status部分を更新;
mutex_exit(&(lock->mutex));
```

(a) 元のコード

```
while (1) {
  new = old = lockのrw_status部分;
  newを更新
  if( CAS( &lock, old, new ) が成功 ) break;
}
```

(b) CASによりlock-freeとしたコード

**図４　lock-free な rw_status の更新**

(b)は、理論的には ll/sc（Load-Link/Store-Conditional）として定式化されています。局所変数の old に lock の rw_status 部分を代入しているのが ll 操作, CAS 命令が sc 操作に対応しています。

## 効果

改造した mysql で DBT-1 を実行させたときの CPU 使用率とスループットの関係を図５に示します。図２との比較から、負荷が高い領域でのスループットが改善されていることが分かり

ます。これにより、イベント・トレースを利用した測定・分析は、ボトルネックを正しく特定できていたといえます。

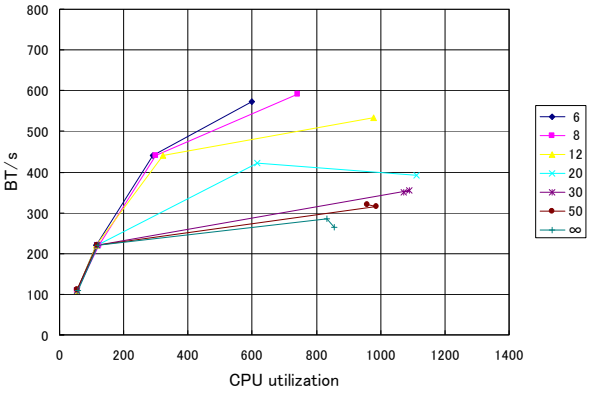


**図5　CPU 使用率とスループット（チューニング後）**

## おわりに

排他制御（ロック）がボトルネックとなっている状況への体系立てた対処として、イベント・トレースを利用する方法を紹介しました。個々のボトルネックはシステム基盤やアプリケーションによって異なる様相を呈しますが、その対応に際して基本となる考え方は共通しています。本稿がマルチコア・システムの性能をチューニングする際の参考になれば幸いです。

## 参考文献

[1] Database Test Suite, http://sourceforge.net/projects/osdldbt/

[2] MySQL に対応した評価ツール DBT-1 を利用したハードリソース変更によるパフォーマンスへの影響の考察, http://ossipedia.ipa.go.jp/capacity/EV0612260303/

[3] OSS 技術開発・評価コンソーシアム,「OSS 性能・信頼性評価 / 障害解析ツール開発」, DB 層～OSDL DBT-1/3 による DBMS 評価編～

http://www.ipa.go.jp/software/open/forum/development/download/051115/db-dbt.pdf

[4] Compare-and-swap, http://en.wikipedia.org/wiki/Compare-and-swap

[5] Load-Link/Store-Conditional,

http://en.wikipedia.org/wiki/Load-Link/Store-Conditional

[6] 堀川　隆、DBMS におけるスケーラビリティーボトルネックの分析、情報処理学会研究報告、2009 年 8 月、Vol. 2009-EVA-29, No. 2.