メモリプロファイリングツールを利用したOLTP性能向上

Performance Tuning of OLTP Workload by a Memory Profiling Tool

吉岡 弘隆♠
Hirotaka YOSHIOKA

CPUの処理速度は年率数十%で向上しているが、メモリの処理速度の向上率はCPUに比較して低い。その結果、メモリをアクセスした時のペナルティが年々増加している。特にデータベース管理システムにおけるメモリアクセスはリスト構造の検索あるいはハッシュなどが多数をしめ、科学技術アプリケーションでみられる配列の単純なアクセスに対する最適化は効果がほとんどないことが知られている。本論文はトランザクション処理（OLTP)性能向上を目的として、パフォーマンスチューニングに必要なメモリプロファイリングツールの開発し、DBMS 等においてその効果を確認した結果を報告する。

The performance of CPU has been improved more than 50％ per year but the performance improvement of memory in latency is much lower than those of CPU. Therefore the penalty of memory access has been increased every year. Recent works show optimization on scientific workloads is not effective on commercial workloads like DBMSs. We have developed a memory profiling tool to collect performance information for OLTP and we have used the tool to tune the workload to evaluate the effectiveness of the tool.

1. はじめに

CPUの処理速度は年率数十%で向上しているが、メモリの処理速度の向上は年率数%といわれており、CPUの向上率に比較して低い。(図1)

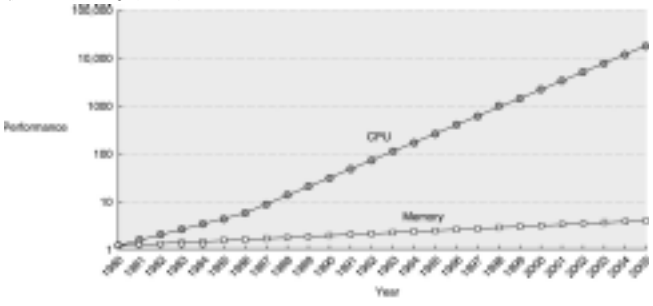

図１：[1]第5章図2からの引用

その結果、CPUとメモリの性能のギャップは年々広がっている。例えば、最近のプロセッサでは、メモリアクセス(キャッシュミス)のペナルティは200倍近くあり、メモリアクセスのコストは一定であるという仮定のもとのプログラミングモデルは破綻している。(表1)従って、より高性能なソフトウェアを実現にするためにメモリ階層を意識したプログラミングモデルが必要となってきている。

表1：Intel Xeon 2GHz/Main Memory 1GBでの実測値

メモリ階層	アクセスコスト
L1キャッシュヒット	1nsec
L2キャッシュヒット	9nsec
メインメモリ	196nsec

90年代の研究によれば、SPECベンチマークに代表する科学技術アプリケーションにおける CPI (Clock Per Instruction) に比べ、商用ワークロード(OLTP --On Line Transaction Processing)の CPI が大きいことが知られている。その要因の一つがメモリアクセス時のストールである。メモリへのアクセス待ちが CPI を上げるのである。([2])

特にRDBMS等におけるメモリアクセスは、ポインタを利用したランダムなアクセスが多く、科学技術アプリケーションでみられる単純な配列の順アクセスに対する最適化は単純には適用できない。

そこでメモリアクセスに注目した性能向上ツールが必要とされているのだが、従来のタイマ割り込みによるPC(Program Counter)のサンプリング手法でのプロファイリングでは、キャッシュミス等のメモリの動的特性についての詳細情報を得ることができない。そのためプログラマは、おおまかなホットスポット(実行時間を多く消費している場所)の位置を推定することができても、何故そこで実行時間がかかっているかを特定することが困難であった。

さらに最近のプロセッサでは、投機的実行やout of order実行、深いパイプラインなどにより、イベントを発生させた命令とPCの値が必ずしも一致しないため、上記のプロファイリングだけでは問題を特定することが益々難しくなっている。

Pentium 4/Intel Xeonでは、上記の問題に対し、ハードウェアによるパフォーマンスモニタリングファシリティをそなえ、18個のパフォーマンスモニタカウンタを持ち、各種メモリアクセスイベント(L1/L2 キャッシュミス/TLB ミス等)を測定できるようになっている。そしてPEBS (Precise Event Based Sampling) と呼ばれる機能によって、以前のプロセッサでは不可能だった、より精密なプロセッサの状態のサンプリングが可能になった。

そこで我々はPentium 4/Intel Xeonにおける上記の機能を利用してメモリプロファイリングツールを実装し、PostgreSQL(フリーのRDBMS)においてそのツールの有効性を検証した結果を示す。我々のツールを利用すれば従来のツールでは困難だったメモリイベント(L1/L2キャッシュミスなど)のボトルネックを容易に発見することができた。

また、プログラムの小さな変更では、キャッシュミスが多発するプログラムとそうでないプログラムの差を厳密に見ることは、OSのタイマの精度がミリ秒以下の差を測定することが通常困難であるため難しかったが、我々のツールでは、キャッシュミスの回数を測定するので、容易にどちらのアルゴリズムがメモリアクセス特性について優れているか判定できた。そのため、アルゴリズムの抜本的な改良のみならず、OSのタイマの粒度では測定が難しい細かい逐次的改良の積み重ねなどの効果も一つ一つ確認しながら行える。

2. メモリプロファイリングツールの実装

Intel社の32ビットマイクロプロセッサ（IA-32と記す）はモデル固有のハードウェア・パフォーマンス・モニタリング機能を持つ。我々はIA-32の上記機能を利用して、メモリプロファイリングツールをLinux上に実装した。[3, 4, 5]ツールは以下のコンポーネントからなる。

1. Linux Kennelへのパッチ

♠ 正会員　ミラクル・リナックス株式会社
hyoshiok@miraclelinux.com

2. ユーティリティ（ebs）
3. ユーザプログラム用API(Application Programming Interface)

Pentium 4/Intel Xeonから精密なイベントベースサンプリング(PEBS Precise Event-based Sampling)という機能が実装された。これはあるハードウェアイベント（例えば、L1キャッシュミス、分岐予測ミスなど）が指定した回数発生すると、その時点でのプロセッサの状態(8個の汎用レジスタ(eax, ebx, ecx, edx, esi,edi, ebp, esp)、EIP(論理アドレス)レジスタ、およびEFLAGSレジスタの値)が保存される。これはマイクロコードによってハードウェアが自動的に実行するので、プロセッサの精密な状態を保存できる。

3. ベンチマークによる検証

本ツールを利用した精密なイベントサンプリング(PEBS)がどのくらい有用な情報を提供するか検証するために、実験を行った。

表２：実験に使用したマシンの仕様

CPU	Intel Xeon
Clock	2.0GHz
Memory	1GB
L1 cache(Data)	8KB (4 way)
L1 line size	64 byte
Trace cache (Instruction)	12K uOP
L2 cache (unified)	512KB (8 way)
L2 line size	64 byte

3.1 pgbench

PostgreSQL のディストリビューションに標準的に添付されている pgbench を利用して、その時の L1/L2 キャッシュミスを測定した。トランザクションのモデルは TPC-B 型の単純なモデルである。CPU をほぼ 100%使い切るために scaling factor を 1 に設定してベンチマークを実行した。

実行例：

```
$ time /usr/local/pgsql/bin/pgbench -c 10 -t 1000 pgbench
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
number of clients: 10
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
tps = 185.972150 (including connections establishing)
tps = 186.100001 (excluding connections establishing)

real    0m53.790s
user    0m1.440s
sys     0m1.540s
```

Linux Kernel 2.4.18 で上記のベンチマーク(pgbench -c 10 -t 1000）を 4 回実行し、L1 キャッシュミスを測定し、その上位 10 位を示した。(表３）この実験では、サンプリング間隔を 10000 とした。すなわち L1 キャッシュミスが 10000 回発生するたびにその時点でのプロセッサの状態(8 個の汎用レジスタ (eax, ebx, ecx, edx, esi,edi, ebp, esp)、EIP(論理アドレス)レジスタ、および EFLAGS レジスタの値)が保存された。その結果、L1キャッシュミスについて280759個のアドレス（そのうち 4375 個がユニーク）を収集した。L1 キャッシュミスを起こしている個所の回数を頻度順に並べて累積度数をグラフ化した。(図２)頻度上位 1%で約 54%、頻度上位 10%で約 87%のキャッシュミスを発生している。

表３：L1 キャッシュミスの頻度と場所、PostgreSQL V2.3, Linux Kernel 2.4.18

頻度	アドレス	ルーチン名
15551	08122b67	LWLockRelease
15429	08122967	LWLockAcquire
14558	0812294b	LWLockAcquire
12635	08122b71	LWLockRelease
5683	081803ba	HeapTupleSatisfiesSnapshot
5556	0816ac8e	SearchCatCache
4917	080768e4	heapgettup
4653	0807ee60	_bt_isequal
3776	08175051	hash_search
3736	0811a752	write_buffer

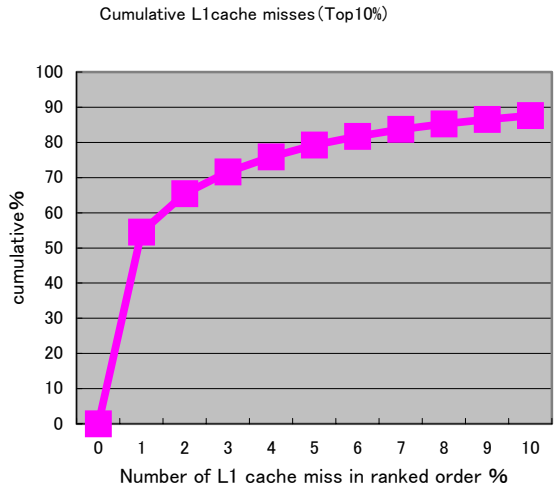

図２：L1 キャッシュミス累積度数

3.2 分析

メモリプロファイリングツールによって精密なイベントサンプリング (PEBS – Precise Event Based Sampling) レコードを取得した。PEBS レコードは、イベントが発生した時点での各レジスタ値を持つ。これをイベントが発生した時のリニアアドレス(EIP)でソートし、その頻度を数えたのが表３である。

ここで L1 キャッシュミスを多発しているルーチンを順にソースコードを見ながら性能向上できないか検討する。最初にキャッシュミス上位のルーチンである LWLockRelease および LWLockAcquire をターゲットに最適化を考える。この 2 つルーチンだけで、全キャッシュミスの 2 割程度を発生している。LWLock は Light Weight Locks で共有メモリ上のデータ構造をロックするために利用される。spinlock を利用して実装されている(下記参照)。TAS(Test and Set)はアトミックにロック変数へ値をセットするマクロであり、インテルプラットフォームでは lock xchg 命令で実装している。

コード：

```
s_lock(volatile slock_t *lock, const char *file, int line)
{
...
    while (TAS(lock))
    {
        if (++spins > SPINS_PER_DELAY)
        {
...
            (void) select(0, NULL, NULL, NULL, &delay);
            spins = 0;
        }
    }
}
```

ここで TAS()は、アトミック性を保証するためにバスをlock(占有)し、かつメインメモリにアクセスしているために非常にコストが高いことに注意されたい。ロックを保有しているプロセスがロックを開放するまで spin loop するが、そのループ中は毎回バスを占有しチェックをするので、(つまり「メモリバス」を占有する)、マルチプロセッサ環境においてスケーラビリティ上の問題がある。

そこで、while(TAS(lock))ループの内側にもうひとつループ(while(*lock))をもうけて最適化をはかってみた。

この実装では、ロックの監視をコストのかかるTAS()で毎回おこなうのではなく、キャッシュへのアクセスへと変更している。キャッシュの監視はアトミックなtest and setではないので、ロックがリリースされた後(内側のループを抜けた後)、再度、外側のループでtest and setをアトミックにおこないロックの確保を試みる。内側のスピンループ中はメインメモリにアクセスしないので、メモリコンテンションが下るのでスケーラビリティの向上がみこまれる。

改良したコード：

```
s_lock(volatile slock_t *lock, const char *file, int line)
{
...
    while (TAS(lock)) /* TASはバスを占有しコストが高い */
    {                 /* 下記のwhile()を追加した */
        while (*lock) /* キャッシュへのアクセス */
        {
            if (++spins > SPINS_PER_DELAY)
            {
...
            }
        }
    }
}
```

この非常にシンプルな改良でL1キャッシュミスにどの程度の影響があったか確認してみた。先の実験と同様、10000回ごとにサンプリングし、280759(オリジナル版)、265927個(改良版) のデータを収集した。

これによれば、該当ルーチンにおいて約15%ほどL1キャッシュミスが減少(改善)していることが確認できた。全体でみるとL1キャッシュミスの減少は約5.3%、TPS(Transaction Per Second)で約2.7%の向上が見られた。L1キャッシュミスを現象させることが実行性能（TPS）を向上させることにつながった。

表４：L1 キャッシュミスの頻度と場所(改良版)、PostgreSQL V2.3, Linux Kernel 2.4.18

頻度	アドレス	ルーチン名
13771	0812294b	LWLockAcquire
13198	08122b67	LWLockRelease
13098	08122967	LWLockAcquire
10473	08122b71	LWLockRelease
5695	081803ca	HeapTupleSatisfiesSnapshot
5417	0816ac9e	SearchCatCache
5060	080768e4	heapgettup
4210	0807ee60	_bt_isequal
3914	08175061	hash_search
3821	08095672	OpernameGetCandidates

さらなる性能向上には、より詳細な検討が必要になるがキャッシュミスを発生させているごく一部のルーチンを変更することによって性能向上を容易に行え、その効果を定量的に確認できた意義は大きいと考える。

また、今回は人手による性能向上を図ったが、キャッシュミスのパターンによっては、性能向上のヒントを自動的に提示することが可能かどうか今後検討したい。例えばコンフリクトミスが発見された場合はキャッシュカラーリングという手法でキャッシュミスを削減できる場合がある。[4]

4. 関連する研究

4.1 パフォーマンスモニタリング機能

Intel の IA-32 のパフォーマンスモニタリングファシリティを利用できるようにした Linux カーネルパッチとして下記がある。

表５：パフォーマンスファシリティ用カーネルパッチ

名前	開発者名	URL
perfctr	Mikael Petterson	[6]
brink/abyss	Brinkley Sprunt	[7]
rabbit	Don Heller	[8]
PAPI	Philip J. Mucchi, et. All	[9]

perfctr は P6 および Pentium 4/Intel Xeon に対応しているが PEBS には対応していない。abyss は PEBS には対応しているが、P6 には対応していない。また HyperThreading や SMP には対応していない。rabbit は P6 だけに対応していており、PEBS や Pentium4 には対応していない。また最近メンテナンスされていないようである。PAPI はマルチプラットフォームなパフォーマンスライブラリで、IA-32 のパフォーマンスモニタリングファシリティの機能については、perfctr を利用しているが PEBS には対応していない。

我々は perfctr をベースに PEBS 対応、Linux Kernel 2.4/2.5 対応のドライバを開発し、PEBS の機能を利用しメモリプロファイリングツールを実装しその有効性を示した。またユーザ向け API(Application Programming Interface)を開発した。

Intel 社より VTune という製品が出荷されているが、オープンソースでないため自由に変更したり、アプリケーションに組み込んで利用したりできない。またユーザ向け API を提供していないので、アプリケーションのある部分だけを測定することなどきめ細かい利用ができない。

4.2 DBMS および OLTP workloads とメモリ階層

従来キャッシュの動的特性については、SPECベンチマークに代表される科学技術計算分野では、多くの研究成果がある。([10],[11])

一方DBMS関連の商用ワークロード(OLTPなど)については、近年研究成果が報告されつつある。([2]など)しかしながら、その成果はワークロード全体でのメモリトラフィックの傾向(キャッシュミス率やCPI (Cycle perInstruction))等の報告がほとんどである。

個々の実装の、どこの部分での、どのくらいのメモリトラフィックがあるか、キャッシュミスのホットスポットがどこにあるか等のミクロな動的特性まで議論したものは少ない。今回提案した方法は、IA-32のパフォーマンスモニタリングファシリティを利用した実装なので、特別なハードウェアは一切必要なく、しかもプロセッサが提供する機能を利用するため、シミュレーションによる方式と違って、実時間での計測が可能である。またアプリケーションの変更を必要としない。

キャッシュミスなどの動的特性の分析は、SPECベンチマークに代表される科学技術計算に対するものが多かったが、IA-32の機能を利用することによって、データベース管理システムやカーネルに代表される動的特性がまだ十分知られ

ていない問題に対しても分析が可能となった。

また、Pentium 4/Intel Xeonで新たに実装されたPEBSの機能を利用すれば、容易にメモリ特性のボトルネックを同定できる。そしてそのメモリボトルネックがなぜ発生しているかの特定も可能である(レジスタ値を分析することによって、詳細な検討が可能になった)。そしてそのような要因が特定できれば、対応する解決策についても、従来知られている方法で対処することができるようになった。

5. 今後の方向と課題

5.1 実装の評価と改良

メモリプロファイリングツールの評価および改良が必要である。

PEBS によって精密なサンプリングが可能になったが、実行においてどのような影響を発生させたかの評価が必要となる。PEBS そのものはソフトウェアによる実装ではないので従来の割り込みを利用したサンプリングと違い、命令キャッシュ(トレースキャッシュ)への影響はない。しかし PEBS そのもののオーバヘッドの評価が必要である。具体的にはツールを起動した時と起動しない時のクロック数の差を評価する。サンプリング間隔は 10000 での測定の場合顕著なオーバヘッドは確認できなかった。

メモリプロファイリングツールはコマンドインターフェースしかもたないため、一般ユーザの利用には若干敷居が高いものとなった。そこで利便性を考え、GUI 化、レポートのビジュアル化(リアルタイムなグラフ生成など)を行う必要がある。

5.2 キャッシュミスペナルティの削減方法の研究

メモリプロファイリングツールによって容易にキャッシュミスの多発個所(ボトルネック)を発見できるようになったが、ボトルネック発見時の効果的な対処方法に関しては個別の対応が必要であった。今回一部のメモリアクセスがキャッシュミスの大半を占めていることを確認したが、一般にほとんどのプログラムでは、delinquent loads と呼ばれているごく一部のスタティックロードがキャッシュミスの大半を占めていることが知られている。

Wang[12]らが報告している delinquent loads のグラフと我々が得たグラフを比べてみると、我々のグラフの方が立ち上がりが滑らかである。滑らかということはキャッシュミス上位の累積度数(貢献度)が小さいので、delinquent loads が相対的に多いということになる。問題の質としては我々の方が難しいといえる。これは我々が実施したアプリケーションワークロード(TPC-B)の特性によるところが多いと推測できるが、どのようなアプリケーション特性がこのような差異を発生させたかをプログラムコードレベルで検討する必要がある。

具体的には、SPEC2000 および TPC-C(ないし TPC-B)についてアプリケーションごとに delinquent loads のトップ 10 位、トップ 1%を求め、各メモリアクセスを分類し、それに差異があるかどうか検討する。主な比較検討項目として次のものを考えている。1)アクセスメソッド(順アクセス(配列)、ランダムアクセス(ポインタ))、2)キャッシュミスのパターン(初期ミス、容量ミス、コンフリクトミス)、3)メモリトラフィック量、4)キャッシュミス回数。

6. まとめ

CPU とメモリ処理速度の向上率の差から、今後ますますメモリ構成を意識したプログラミングが重要になってくる。しかしながら従来はメモリアクセスコストを簡単に指摘するツールがなかったため、メモリコストの安いプログラムを書くことが難しかった。

今回開発したツールを利用すれば、簡単にユーザーはメモリ関連の動的特性に関する情報を入手でき、性能上のボトルネックを発見でき、性能向上を行える。

[謝辞]

本プロジェクトは平成 14 年度未踏ソフトウェア創造事業(プロジェクトマネージャー喜連川優東京大学教授)「OLTP性能向上を目的としたメモリプロファイリングツール」として支援を受けた。

また、今回開発したツール、スクリプト等は下記の URL で公開している。メーリングリストなどもあるのでご活用いただければ幸いである。

http://sourceforge.jp/projects/hardmeter/

[文献]

[1] Hennessy, J. L., and Patterson, D. A., Computer Architecture: A Quantitative Approach, 3rd Edition, Morgan Kaufmann Publishers, 2002

[2] Ailamaki, A. et. al, “DBMSs On A Modern Processor: Where Does Time Go?”, Proceedings of the 25th VLDB Conference, 1999

[3] 吉岡弘隆、平成14年度未踏ソフトウェア創造事業「OLTP性能向上を目的としたメモリプロファイリングツール」成果報告書、2003年2月

[4] 吉岡弘隆、“Intel系(IA－32)プロセッサのパフォーマンスモニタリングファシリティを利用したメモリプロファイリングツール”、第44回プログラミングシンポジウム、箱根、2003年

[5] 吉岡弘隆、“OLTP性能向上を目的としたメモリプロファイリングツール”、第14回データ工学ワークショップ、電子情報通信学会、2003年

[6] http://www.csd.uu.se/mikpe/linux/~perfctr/

[7] http://www.eg.bucknell.edu/~bsprunt/

[8] http://www.scl.ameslab.gov/Projects/Rabbit/

[9] http://icl.cs.utk.edu/projects/papi/

[10] Cantin, J. F., et al, “Cache Performance for SPEC CPU2000 Benchmarks”, http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/

[11] Gee, J. D., et al, “Cache Performance of the SPEC92 Benchmark Suite”, IEEE Micro (August) 17-27, 1993

[12] Wang, H., etal, “Speculative Precomputation: Exploring the use of Multithreading for Latency.” Intel Technology Journal, Volume 6 Issue 1, February 2002, http://developer.intel.com/technology/itj/2002/volume06issue01/

吉岡 弘隆 Hirotaka YOSHIOKA

ミラクル・リナックス株式会社取締役戦略担当。1984 年慶應義塾大学大学院理工学研究科修士課程修了。1984 年日本ディジタルイクイップメント研究開発センター株式会社入社、1994 年日本オラクル株式会社入社、2000 年ミラクル・リナックス株式会社設立、OSDL Board of Directors、情報処理学会会員、日本データベース学会会員、ACM 会員。