社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

SSDを利用したリレーショナルデータベースにおける大規模意思決定支援クエリ処理性能の特性

鈴木　恵介†　早水　悠登†　横山　大作†　中野美由紀†　喜連川　優†,††

† 東京大学, 東京都
† 国立情報学研究所, 東京都

あらまし　SSD は HDD の 100 倍から 1000 倍の転送レートを持つ高速なストレージデバイスとして注目されている。データインテンシブアプリケーションの処理性能は、HDD を用いた環境では入出力コストが支配的なため、SSD により、その処理性能を大きく向上することが期待できる。一方で、処理コストに占める入出力コストが相対的に小さくなるため、HDD を使用していた際には気づかなかったアプリケーションの挙動が見えてくる。本稿では、関係データベースのハッシュジョインの処理性能を詳細に解析することで、HDD と SSD それぞれを用いた場合の挙動の違いについて考察し、 SSD の特性を考慮した処理コストモデルの必要性を明らかにする。
キーワード　データベースシステム, SSD, OLAP, 結合演算

Performance Characteristics of Large Analytical Query Processing on Relational Database with SSDs

Keisuke SUZUKI†, Yuto HAYAMIZU†, Daisaku YOKOYAMA†, Miyuki NAKANO†, and Masaru KITSUREGAWA†,††

† Kitsuregawa Lab., Center for Information Fusion, Institute of Industrial Science (IIS), Meguro-ku Komaba 4-6-1, Tokyo, Japan, 153-8505
† 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan, 101-8430

Abstract　The SSDs are expected new storage devices whose transfer rate are thousands faster than that of HDDs. By employing SSDs on behalf of HDDs, performance of data intensive applications is likely to be improved drastically since I/O cost of HDDs is dominant in execution time of data intensive applications. Then, different behavior of data intensive application may be observed in case of SSDs, since SSD's I/O cost is relatively small in total execution time. In this report, we analyze the difference in behaviors of hash join operation on a relational database with both HDD and SSD environment, and clarify that it is necessary to consider the cost model by taking into account characteristics of SSDs
Key words　DBMS, SSD, OLAP, Join

1. はじめに

現在の計算機では大規模データ処理性能は、二次記憶装置の転送速度に律速される場合が多い。従来の磁気ディスク (HDD) のスループットは、CPU の高速化と比較すると進展が少なく、より高速、広帯域なストレージデバイスが求められている。近年では、NAND flash や Phase Change Memory、FeRAM などの Storage Class Memory (SCM) と総称される、大容量、不揮発かつ HDD よりも高速なストレージデバイスの研究、開発が盛んになっており、HDD に置き換わる 2 次記憶装置として注目を集めている。中でも、NAND flash-based Solid State Drive (SSD) は、NAND flash chip の集積技術の進歩により、大容量化、低価格化が進み、データセンターなど大規模データ処理が必要な現場において導入が増加している。

SSD のアクセスレイテンシは HDD に対して 100-1000 倍程度高速であるため、従来の HDD を利用した場合と比較し、アプリケーションの挙動が大きく変化すると考えられる。

本稿では、データインテンシブなワークロードを持つ代表的なアプリケーションである、データベースシステム（DBMS）において、ストレージへの負荷の高い結合演算及び TPC-H [1]

ベンチマークを用い、SSD を利用した際の処理性能について詳述する。SSD による I/O の高速化によって、全体の処理性能を大きく向上できることを示すと同時に、HDD を利用した場合には見えなかった問題点を明らかにする。

本稿の構成は以下の通りである。2. 節では、今回計測の対象とするハッシュを用いた結合演算の処理方法と性質について述べ、3. 節では、実際の DBMS においてベンチマーククエリの性能計測を行い、結果について分析する。4. 節で SSD の活用のための DBMS の最適化の関連研究について触れる。5. 節ではまとめと今後の展望について述べる。

2. ハッシュを用いた結合演算

本稿では、大規模関係データベース演算として、ハッシュジョインを用いて SSD の有効性について論じる。ハッシュジョインでは、始めにハッシュテーブルを作成すれば、その後のタプルのジョイン処理を線形時間で行うことができ、データの選択率が高い時効率がよい。

2.1 Grace ハッシュジョイン [2]

例として、以下のクエリを考える。

```
SELECT R.a2, S.a3 FROM R, S
WHERE R.a1 = S.a1
```

テーブル R および S から R.a1 と S.a1 のカラムが一致するタプル同士の R.a2、S.a3 のカラムを抽出するといった操作になる。

R と S のテーブルがメモリ上に収まらない場合は、build phase と probe phase の 2 つの phase で処理される。まず、build phase では、R と S それぞれのテーブルを同じハッシュ関数で分割し、二次記憶に書き込む。このとき、全ての S のパーティションが、メモリに収まるサイズになるようにする。次に、probe phase ではまず S のパーティションをストレージから読み込み、メモリ上にハッシュテーブルを作成する。続けて、該当するハッシュ値の R のパーティションからタプルを読み込み、ハッシュテーブルを利用してジョイン条件のマッチングを行う。これを R と S の全てのパーティションに対して繰り返して処理が完了する。

2.2 ハイブリッドハッシュジョイン [3]

build phase において必要となるメモリは、n 個のパーティションに対しての書き込みバッファ分の n ページのみであるので、残りのページを利用して S の 1 つ目のパーティションのハッシュテーブルをメモリ上に保持することが可能である。この時 R の build phase では、R の 1 つ目のパーティションに属するタプルは二次記憶に書き込まれることなく、その場で S の 1 つ目のパーティションのハッシュテーブルを使って probing を行い、結果を返す。この操作により、S と R それぞれの 1 つ目のパーティション分の二次記憶へのアクセスを削減できる。

2.3 ハッシュジョインの処理コスト

HDD のアクセスコストが大きいため、従来の DBMS で用いられるネストループジョインは、大規模データの問い合わせでは処理負荷が高くなる。そこで、多くの DBMS では、Grace ハッシュジョインまたはハイブリッドハッシュジョインが採用されている。しかしながら、メモリが小さい場合、ハッシュジョインを用いても、その処理コストは大きくなる。図 1 に HDD を用いた DBMS における TPC-H クエリ 8 の実行時間とその内訳を示す。表 1 に計測に用いたシステムの環境、表 2 にデータベースの情報を示す。図 1 から分かる通り、用いる work_mem が小さいと、I/O wait の時間が全体の時間の中で大きな割合を占めている。これは、オリジナルのリレーションがシーケンシャルアクセス可能なのに対し、パーティションのアクセスはランダム I/O なので、処理コストが高くなるためである。従来の DBMS の実行時間モデルは、二次記憶の転送コストと主記憶の処理コストから構成されているが、二次記憶としては HDD の特性しか考慮されていない。SSD を用いることにより、転送速度が 100-1000 倍高速化されると同時に、read/write の性能差、シーケンシャル/ランダム I/O の性能差は HDD とは全く異なったものになる。つまり、SSD を用いたデータインテンシブアプリケーションではアプリケーションの処理コストは従来とは全く違ったものになると考えられる。そこで、3. 節以降、SSD を用いた場合のハッシュジョインの挙動について詳細に解析し、HDD に基づいた処理コストモデルとは異なるコス

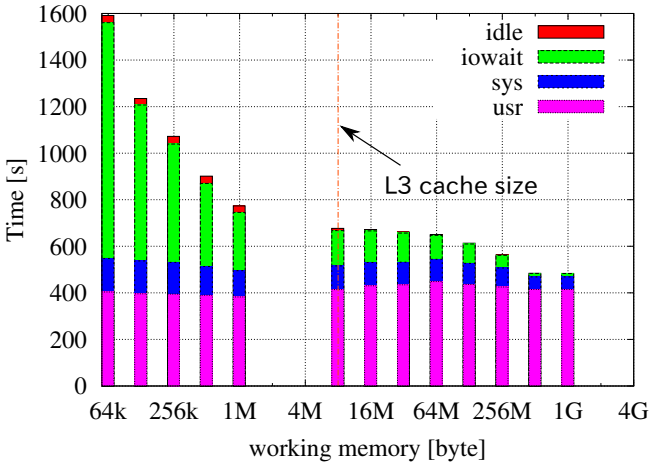


図 1 HDD 上での各 work_mem の値におけるクエリ実行時間 (Query2)

Fig. 1 Query Execution Time for each work_mem size on HDD (Query2)

表 1 計測に用いたシステムの環境（HDD）

Table 1 Experimental Platform of HDD

CPU	Xeon E5530 @ 2.40GHz x 2
L3 Cache	8MB
Main Memory	24GB
Storage	Hitachi HDS721010CLA332 x10 (RAID6 JCS VCRVAX-4F)

表 2 計測に用いたデータベースの設定

Table 2 Experimental Database Setup

DBMS	PostgreSQL 9.2.1 [4]
Shared buffer	8GB
work_mem	64KB - 2GB
TPC-H Scale factor	100

表 3　計測に用いたシステムの環境（SSD）
Table 3　Experimental Platform of SSD

CPU	Xeon X7560 @ 2.27GHz x 4
L3 Cache	24MB
Main Memory	64GB
Storage	ioDrive Duo x4 (8LU, Software RAID0)

表 4　各テーブルのサイズ及びタプル数
Table 4　Table sizes and number of tuples

Table name	Size	# of tuple
part	3.2GB	20,000,000
lineitem	86GB	600,037,902
orders	20GB	150,000,000
customer	2.7GB	15,000,000
supplier	173MB	1,000,000
nation	8kB	25
region	8kB	5

表 5　各記憶装置のレイテンシ
Table 5　Latency of CPU caches and storages

Storage	Latency
L3 cache	20 ns
DRAM (local)	100 ns
DRAM (Remote)	150 ns
SSD (512KB)	30 us
HDD (random, 512KB)	10 ms

トモデルの必要性を明らかにする。

3. ハッシュジョインの性能計測

SDD 上にデータベースを構築し、ジョイン演算を行うクエリについてメモリ使用量を変化させて実行時間を計測する。

3.1 計測環境

表 3 に計測に用いたシステムの環境を示す。また、実験環境における各記憶装置のアクセスレイテンシを表 5 に示す。データベースの設定に関しては、HDD のときと同様で、表 2 の通りであり、DB のベンチマークは TPC-H を利用する。各テーブルサイズは表 4 の通りである。

クエリ実行時のハッシュテーブルに用いるメモリ使用量 (work_mem) を 64KB-2GB の間で変化させる。また、クエリの実行時間とともに、キャッシュ参照回数、キャッシュミス回数、キャッシュミス率をプロファイリングツール perf を利用して取得する。

3.2 1 つのハッシュジョインを行うクエリ

1 回のハッシュジョインを行うクエリについて計測を行う。計測には Query1 の、lineitem 表と part 表をジョインするクエリを用いる。

Query 1　Join line item and part

```
SELECT count(*) FROM lineitem, part
WHERE l_partkey = p_partkey
```

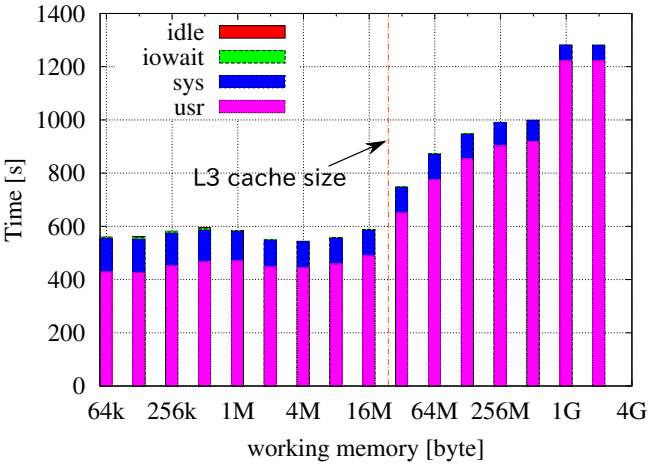


図 2　各 work_mem の値におけるクエリ実行時間 (Query1)
Fig. 2　Query Execution Time for each work_mem size (Query1)

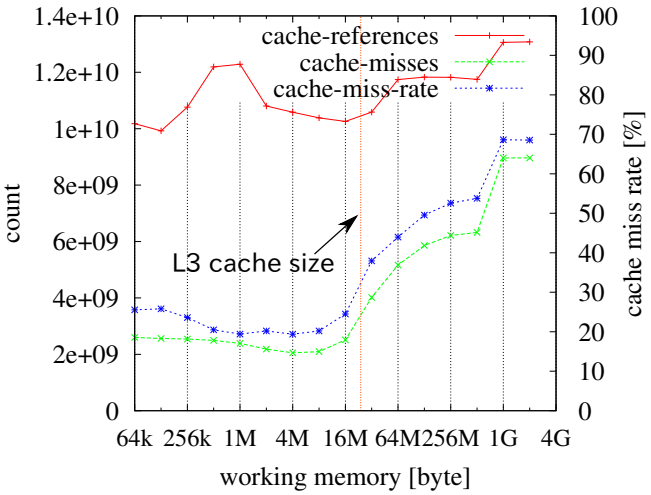


図 3　各 work_mem の値における L3 キャッシュ参照/ミスの回数とキャッシュミス率 (Query1)
Fig. 3　Number of cache references/misses and cache miss rates for each work_mem size (Query1)

図 2 は、work_mem の値を変化させた時のクエリ実行時間と user、system、iowait、idle の CPU 時間の内訳である。図中の usr が CPU コスト、sys と iowait の合計が I/O コストに相当する(注1)。work_mem の値が 16MB から 32MB と L3 キャッシュサイズの境界を越えた点から、CPU コストが大きく増加している。32MB 以降で実行時間が増加している原因としては、L3 キャッシュミス率の上昇が考えられる。

図 3 は、各 work_mem の値における L3 キャッシュ参照及びミスの回数とキャッシュミス率を示している。メモリ上のハッシュテーブルのサイズが L3 キャッシュサイズを越える 32MB 以降でキャッシュミス回数が増加している。メモリアクセスをローカルノードのみと仮定し、表 5 の値を利用してキャッシュ

(注1)：今回ストレージに使用した ioDrive は、システム内にデバイスのコントロールプロセスが存在しており、I/O が発行されると、このプロセスを介して処理される。コントロールプロセスが動作している時間は sys に含められており、今回の計測の場合は sys の時間はコントロールプロセスの消費時間に近似できた。

ミスのペナルティを計算すると、例えば、work_mem=4MB のとき $100(ns) \times 2 \times 10^9 = 200(s)$、work_mem=1GB のとき $100(ns) \times 9 \times 10^9 = 900(s)$ で 700 秒の差となり図 2 の CPU コストの差とおおよそ同程度の差になっている。

クエリ実行中のキャッシュ参照及びミス回数の推移をみると、work_mem=64kB-512MB （図 4(a)-4(e)）の間では、途中で傾向が変化している。図 4(a) の場合だと、経過時間が 300 秒の手前で、キャッシュ参照/ミスの回数、キャッシュミス率が低下している部分である。前の部分が build phase、後ろの部分が probe phase に相当する。

probe phase に関しては、work_mem の値が L3 キャッシュサイズより小さいときは、メモリ上のハッシュテーブルが L3 キャッシュに収まる大きさになるためキャッシュミス率が小さく、図から読み取ると work_mem=4MB のときで約 6%、work_mem=16MB のときで約 10%である。work_mem が大きくなると、ハッシュテーブルは L3 キャッシュに収まりきらなくなるため、キャッシュミス率が高くなり、work_mem=32MB のときで約 40%、work_mem=512MB のときで約 80%となっている。

build phase におけるキャッシュの参照について考えると、まず各リレーションのスキャンでは、work_mem の値に関わらず一定の回数キャッシュミスが発生する。アウターリレーションの build の際には、1 つ目のパーティションに属するアウターリレーションのタプルはその場で probing されるため、work_mem が大きいとき多くのタプルが probing されることになる。このとき、メモリ上のハッシュテーブルも大きくなるため、probe phase と同様の理由でキャッシュミス率は高くなり多くのキャッシュミスが発生する。キャッシュミス率は、work_mem=4MB のとき約 30%、work_mem=512MB のとき約 45%、work_mem=1GB のとき約 70%となっている。なお、work_mem=1GB 以降では、ハッシュテーブルは 1 つのパーティションに収まるため、probe phase はなくなり全てのタプルが build phase で処理される。

build phase と probe phase において発生するキャッシュミスのペナルティによって、work_mem が L3 キャッシュサイズ以上の値をとるとき、CPU コストが増加する。図 3 を参照すると、キャッシュミス回数は最大で 7×10^9 程度の開きがある。PostgreSQL では、ハッシュテーブルの 1 つのバケツに 10 タプル格納されることを仮定して分割を行っているため、probing によって発生するキャッシュ参照は、（アウターリレーション *lineitem* のタプル数）×（平均バケツ内タプル数）$= 6 \times 10^8 \times 10 = 6 \times 10^9$ となりオーダーとしてもおおよそ一致していると言える。

CPU コストと I/O コストの推移に着目すると、図 2 より例として、4MB のときと 1GB のときでは、I/O コストが 101 秒から 59 秒と 42 秒減少しており、CPU コストは 447 秒から 1221 秒と 774 秒増加している。CPU コスト増加の影響が I/O コスト減少の影響よりも大きくなっており、キャッシュミスペナルティによる CPU コストの増加が性能に大きく影響している。

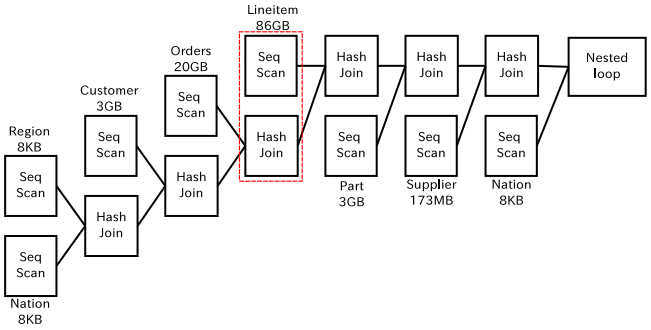


図 5　Query2 実行プラン
Fig. 5　Execution Plan of Query2

3.3　複数のハッシュジョインを行うクエリ

複数のハッシュジョインを行うクエリとして、TPC-H のクエリ 8 を用いて計測を行う。ハッシュジョインの性能を計測することが主眼であるので、クエリ中のジョイン演算を行う部分のみを実行対象とする（Query2）。テーブルの結合順序は図 5 の通りである。

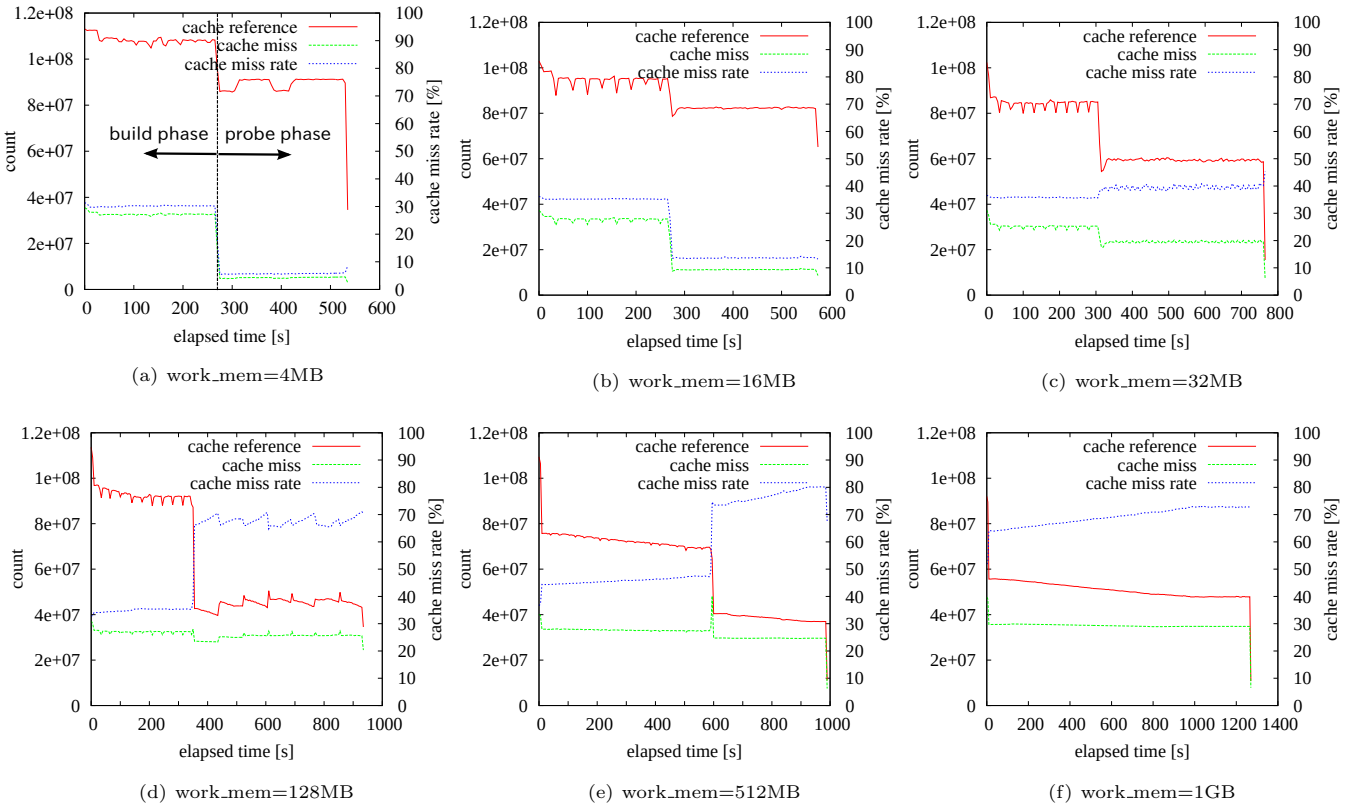
Query 2　TPC-H Query 8 (join part)

```
SELECT extract(year from o_orderdate),
       l_extendedprice * (1 - l_discount),
       n2.n_name
FROM part, supplier, lineitem, orders,
     customer, nation n1, nation n2, region
WHERE p_partkey = l_partkey
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'AMERICA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between
          '1995-01-01' and '1996-12-31'
      and p_type = 'ECONOMY ANODIZED STEEL'
```

図 6 は、work_mem の値を変化させた時のクエリ実行時間とその内訳である。1 つのジョインのみのときと同様に、L3 キャッシュサイズの境界付近で CPU コストが増加し、work_mem=2MB のときの 431 秒から work_mem=64MB のときの 547 秒まで 116 秒増えている。

図 7 は、各 work_mem の値における L3 キャッシュ参照及びミスの回数とキャッシュミス率を示している。work_mem=2MB のときの 30%から work_mem=512MB 以降の 50%まで増加している。L3 キャッシュサイズより手前の 8MB からキャッシュミス率が増加しているのは、ハッシュジョインが多段になっているため、複数のハッシュテーブルのデータがキャッシュに存在しているためである。

図 5 の実行プランのうち最も時間のかかる赤破線内のハッシュジョインでは、ハッシュテーブルサイズは 300MB 程度で



(a) work_mem=4MB　(b) work_mem=16MB　(c) work_mem=32MB

(d) work_mem=128MB　(e) work_mem=512MB　(f) work_mem=1GB

図 4　クエリ実行時の L3 キャッシュ参照/ミスの回数とキャッシュミス率の時間推移 (Query1)

Fig. 4　Timeline of number of cache references/misses and cache miss rates (Query1)

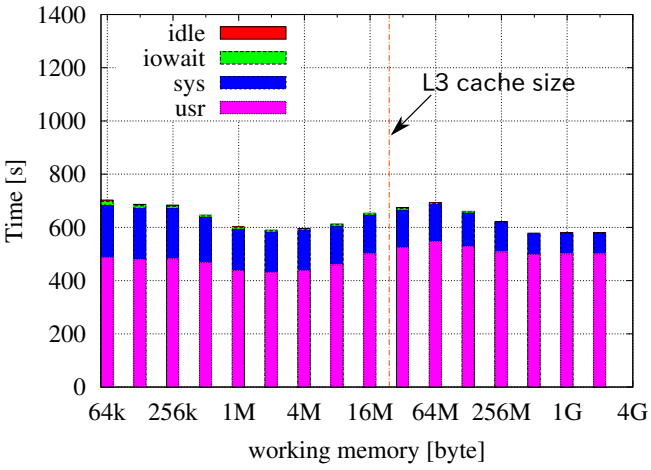


図 6　各 work_mem の値におけるクエリ実行時間 (Query2)

Fig. 6　Query Execution Time for each work_mem size (Query2)

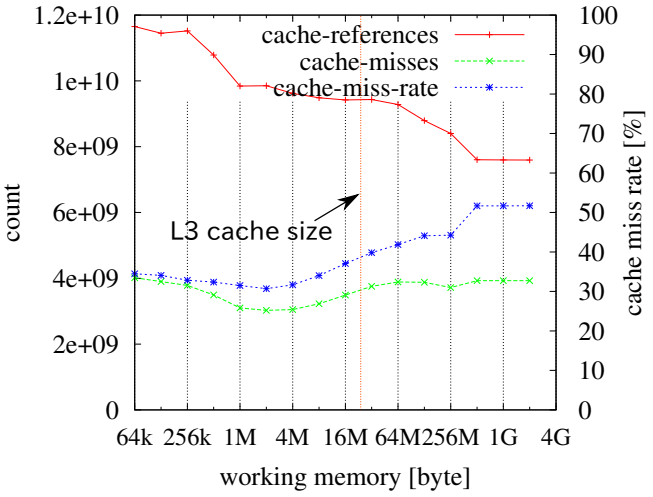


図 7　各 work_mem の値における L3 キャッシュ参照/ミスの回数とキャッシュミス率 (Query2)

Fig. 7　Number of cache references/misses and cache miss rates for each work_mem size (Query2)

あり、work_mem の値が大きい時メモリ上のハッシュテーブルは L3 キャッシュに収まらないサイズになっているにも関わらず、キャッシュミス率は最大で 50%と、1 つのジョインのクエリの場合と比較すると小さくなっている。これは、ジョイン条件となっている lineitem 表の l_orderkey アトリビュートがデータベース中で昇順にソートされており、さらに同じ値を持つタプルが平均 4 個存在(注2)しているため、同じ l_orderkey の値を持つタプルが複数存在するとき、probing の際に 2 つ目以降のタプルは全てキャッシュにヒットするためである。

3.4　考　　察

3.2、3.3 節の計測において見られたように、SSD を使用した場合、I/O コストのみではなく、CPU コストもクエリの実行性能に影響を与えていた。図 1 の HDD の処理コストの内訳比と、図 2、6 の SSD の処理コストの内訳比を比較すると、SSD では明らかに CPU コストが支配的である。これは、従来の HDD

(注2)：TPC-H ドキュメントの Test DataBase Data Generation 節参照。

を使用を前提としたDBで利用されていた、HDDの特性を基にしたコストモデルでは説明できない現象であり、SSDを使用したことによってストレージアクセスレイテンシとメモリアクセスレイテンシの差が小さくなったことによるものである。つまり、クエリプランナなどのコスト推定では、SSDに対する利用環境を特定し、キャッシュミスペナルティによるCPUコストの増加も考慮したコストモデルを利用する必要がある。

4. 関連研究

DBMSのアプリケーションレベルでの最適化に関する研究としては、バッファプールやSSDとHDDのハイブリッドストレージ管理、インデクシングに関するものが多い。SSDによるバッファプール拡張に関しては、Bhattacharjeeらの、temperature-aware caching (TAC) schema [5], [6] では、データのアクセスパターンをモニタリングし、アクセス頻度に応じたキャッシュ置換アルゴリズムを提案している。Kangらの提案するFaCEシステム [7] では、キャッシュ置換アルゴリズムにmultiversion FIFOを利用し、ランダム書き込みを削減している。また、キャッシュ追い出しの集約やSecond Chanceによってもスループットを向上させている。

SSDとHDDをハイブリッドストレージ環境で使う手法としては、Koltsidasらの研究 [8] では、モニタリングにより、ページ毎のワークロードを特定し、read-intensiveなページをSSDに、write-intensiveなページをHDDに配置する方法を提案している。hStorage-DB [9] では、DBMSのもつ、クエリに関するsemantic informationを利用したデータ管理を行っている。

インデクシングに関する研究では、FD-tree [10] は、データ更新時に書き込まれたデータをバッファリングし、SSDへの書き込みをシーケンシャルに行う最適化を実現している。PIO B-tree [7] では、SSDのinternal parallelismに着目した最適化を行っている。

これらのアプリケーションレベルでの最適化では、SSDの苦手なデータアクセス方法を避け、SSDの性能を十分に引き出すことを目的としているものが多い。SSDを導入したシステムの性能を最大限に活用するためには、SSDを使用することで、今までは見えていなかったCPUコストの影響も考慮に入れ最適化を行うことが必要となる。

5. おわりに

本稿では、DBMSにSSDを使用し、ハッシュジョインの実行性能について計測、分析を行った。

1つのハッシュジョインを含むクエリ、複数のハッシュジョインを含むクエリ共に、使用メモリ量がL3キャッシュサイズを越えるとキャッシュミスのペナルティによりCPUコストが増加し、1つのハッシュジョインを含むクエリの場合では、最大で2.35倍実行時間に差が生じた。これは、SSDによってストレージアクセスが高速化され、処理コストの割合としてI/Oコストが小さくなり、CPUコストが支配的になったことが原因である。CPUコストの増加が全体のクエリ処理の性能に影響を及ぼすことは、従来のHDDの使用を前提としたDBの、HDDの性能特性を基にしたコストモデルでは発生し得ないものであり、SSDを使用する際のコスト推定には、SSDの特性を考慮したコストモデルが必要になる。

今後の研究課題としては、今回の計測で得られた知見を基に、キャッシュミス率を考慮したコストモデルを作ることが挙げられる。このコストモデルを利用し、実行時間の正確な推定値が得られれば、SSDを利用したDBMSの最適化について検討が可能になる。

文献

[1] “Transaction Processing Performance Council, an ad-hoc, decision support benchmark”. http://www.tpc.org/tpch/.

[2] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, “Relational algebra machine grace,” Proceedings of RIMS Symposium on Software Science and Engineering, pp.191–214, Springer-Verlag, London, UK, UK, 1983.

[3] D.A. Schneider and D.J. DeWitt, “A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment,” Proceedings of the 1989 ACM SIGMOD international conference on Management of data, pp.110–121, SIGMOD ’89, ACM, New York, NY, USA, 1989.

[4] “PostgreSQL”. http://www.postgresql.org/.

[5] B. Bhattacharjee, K.A. Ross, C. Lang, G.A. Mihaila, and M. Banikazemi, “Enhancing recovery using an ssd buffer pool extension,” Proceedings of the Seventh International Workshop on Data Management on New Hardware, pp.10–16, DaMoN ’11, ACM, New York, NY, USA, 2011.

[6] M. Canim, G.A. Mihaila, B. Bhattacharjee, K.A. Ross, and C.A. Lang, “Ssd bufferpool extensions for database systems,” Proc. VLDB Endow., vol.3, no.1-2, pp.1435–1446, Sept. 2010.

[7] W.-H. Kang, S.-W. Lee, and B. Moon, “Flash-based extended cache for higher throughput and faster recovery,” Proc. VLDB Endow., vol.5, no.11, pp.1615–1626, July 2012.

[8] I. Koltsidas and S.D. Viglas, “Flashing up the storage layer,” Proc. VLDB Endow., vol.1, no.1, pp.514–525, Aug. 2008.

[9] T. Luo, R. Lee, M. Mesnier, F. Chen, and X. Zhang, “hstorage-db: heterogeneity-aware data management to exploit the full capability of hybrid storage systems,” Proc. VLDB Endow., vol.5, no.10, pp.1076–1087, June 2012.

[10] Y. Li, B. He, R.J. Yang, Q. Luo, and K. Yi, “Tree indexing on solid state drives,” Proc. VLDB Endow., vol.3, no.1-2, pp.1195–1206, Sept. 2010.