高速ストレージ環境における DBMS クエリの I/O 並列化に関する検討

鈴木 大地† 上田 高徳†,‡ 山名 早人††
†早稲田大学大学院 基幹理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1
‡早稲田大学 IT 研究機構 〒169-8050 東京都新宿区戸塚町 1-104
††早稲田大学理工学術院 〒169-8555 東京都新宿区大久保 3-4-1,
国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2

E-mail: {dsuzuki,ueda,yamana}@yama.info.waseda.ac.jp

あらまし 近年，半導体技術の進展により Solid State Drive (SSD）が広く利用されるようになった．SSD は HDD のヘッドやアークのような物理機構を持たないため, 高速なストレージ I/O 性能, 優れた耐障害性や低消費電力などの特徴を有する. DBMS においても SSD の高速な I/O により，クエリが高速に処理できることから SSD ベースの DBMS への期待が高まっている．しかし，ストレージが高速になるほど OS や DBMS においてストレージ I/O に関する処理に必要な単位時間あたりの CPU 時間が増える．将来的には，SSD などが発展した高速なストレージ環境において，OS や DBMS の I/O 処理に占める CPU 時間が増えボトルネックになることが予想される．現在，1 CPU コアあたりの性能は頭打ちになり，性能向上のためマルチコア化が主流となっている．OS や DBMS における I/O 処理でもボトルネックがストレージ I/O から CPU へとシフトしつつあり，I/O 処理の並列化が必要である．以上を踏まえ本論文では，OLAP のクエリによく見られるストレージへのシーケンシャルなアクセスを主に想定し，I/O 処理を並列化や非同期処理を考慮しながら，SSD 環境における DBMS のクエリ処理性能について検討及び性能を評価し，考察を行う．

キーワード DBMS，Solid State Drive，OLAP, マルチコア

1. はじめに

近年，半導体技術の進展により SSD (Solid State Drive) がベンダ・コンシューマを問わず広く利用されている．SSD は，HDD (Hard Disk Drive) と同様のインターフェースを採用していることや高速なディスク I/O 性能，優れた耐障害性や低消費電力などの特徴を有していることから，今後もさらなる普及が予想される．

また昨今は，特にビッグデータを契機として，企業で大規模な顧客データなどを対象として OLAP（オンライン分析処理）が盛んに行われている[1]．OLAP で扱うデータの規模は年々増大する傾向にあり，解析基盤としてのストレージの I/O 性能の向上は重要な要素となっている．このため SSD がストレージとして採用される機会も多くなっている．

SSD のような高速なストレージの台頭で I/O 性能が向上する一方，OS や DBMS においてストレージ I/O に関する処理に必要な CPU 時間は増える傾向にある．このため将来的には，SSD などが発展した高速なストレージ環境において，OS や DBMS の I/O 処理に占める CPU 時間が増えボトルネックになることが予想される．また現在，1 CPU コアあたりの性能は頭打ちになり，性能向上のためマルチコア化が主流となっている．OS や DBMS における I/O 処理でもこの傾向は顕著であり，ボトルネックがストレージ I/O から CPU へとシフトしつつあり，I/O 処理の並列化を検討する必要がある．

既存研究では，SSD の Read/Write 性能の非対称性や高速なランダムアクセスに着目し，SSD 環境における DBMS のコストモデルを議論・提案しているもの[5]や SSD と HDD を対象として，External Sort や Join に関するアクセスパターンの違いを比較・検討している研究[6]がある．しかしマルチコア CPU が今後さらに進んだ場合, CPU が DBMS クエリのコストモデルに及ぼす影響は無視できないものであり，高速なストレージ環境に適したアーキテクチャを OS 含めて考える必要がある．

そこで高速ストレージ環境を前提として，実際にクエリ処理に及ぼす影響を検討・評価する必要がある．

本稿ではまず SSD の特性を述べつつ，Linux OS や DBMS の I/O 処理における現状の課題について議論する．実際の TPC-H のクエリの一部を用いて，I/O 性能の評価し，クエリに与える影響を考察する．

本稿では以下の構成をとる．2 節で SSD 環境における CPU ボトルネックについて，DBMS のクエリや Linux OS の観点から議論をする．3 節においてストレージの I/O 性能および DBMS クエリに関する実験および結果を述べ，考察を行う．4 節で関連研究について述べる．最後に 5 節でまとめを述べる．

2. I/O 処理における CPU ボトルネックの考察

本節では OS のカーネルや DBMS の I/O 処理における CPU ボトルネックについて簡易的な実験の結果を示しつつ議論する．Linux カーネルの 2.6.32-X を対象としている．

2.1. OS の I/O に関する議論

2.1.1. Direct I/O と Linux カーネル

Linux OS 上でストレージからの読み込み，及びその際の CPU 負荷を計測するため，表 1 に示すコードを用いて計測を行った．スループットとカーネルの CPU 使用率の結果は，図 1 に，また Linux カーネルを含むプロファイルの結果を表 3 に示す．本節で使用した機器及び環境については表 2 に示すとおりである．

表 1 の処理では，256GB のファイルに対し 2GB のブロックサイズで，Direct I/O によるシーケンシャルな読み込みを行っている．Direct I/O とは，通常の Linux OS のページキャッシュを経由した I/O と違い，ユーザ空間とファイルの間でデータを転送する方式である．DBMS などのアプリケーションでは，独自のページキャッシュ機構を持つため，通常のページキャッシュ（ディスクキャッシュ）を利用した I/O では，キャッシュの処理が二重になり，無駄な遅延が発生することになる．そこで Direct I/O のような機構が必要になる．また OLAP においては，アドホックなクエリが用いられるため，OS のキャッシュを多用することは考えにくい．

実験結果の図 1 より Direct I/O を利用した読み込み処理におけるスループットは毎秒 5,000~5,500MB であり，その際のカーネルの CPU 使用率 (*% system*) も 70~80%ほどであることがわかる．実際のカーネルの処理では，システムコールの read に起因する処理が大半を占めている（表 3 の太字の symbol name 群）．

表 1 読み込み処理を行うコード

```
const long long BUFF_SIZE = 1024LL*1024LL*2048LL;
int main(int argc, char* argv[]) {
        int fd = open("output.file", O_RDONLY|
O_DIRECT);
        if(fd < 0)        return -1;
        char * buffer;
        posix_memalign((void**) &buffer, 512,
BUFF_SIZE);
        ssize_t ret = 0;
        while((ret = read(fd, buffer, BUFF_SIZE)) > 0)
        close(fd);
        free(buffer);
        return 0 ;
}
```

Linux OS において，ブロックデバイスからの読み込みは *generic_file_read* 関数がエントリとなっており，カーネルはキャッシュの検索 (*balance_dirty_pages*) やファイルの先読み処理 (*do_page_cache_readahead*, *read_cache_pages_invalidate_pages*) などを行なっている．プロファイルの結果より，RAID やバスでは，まだ帯域に余裕があることが確認できる．そこで I/O の処理に関して，将来的にさらに高速なストレージ環境では CPU でボトルネックが生じることが予想される．

I/O 処理における CPU のボトルネックの原因として，現在の Linux OS の I/O 処理では，システムコールから呼び出されたスレッドコンテキスト上で動作することになっていることが考えられる．結果として 1CPU コアあたりの負荷が増加することになり，I/O 処理に占める CPU 時間が増えボトルネックになることが予想される．将来的にさらに高速なストレージ環境において，このような CPU ボトルネックを避けるためには，Linux OS のシステムコールが呼び出したスレッドコンテキスト上以外でも I/O 処理が動作することが望まれる．

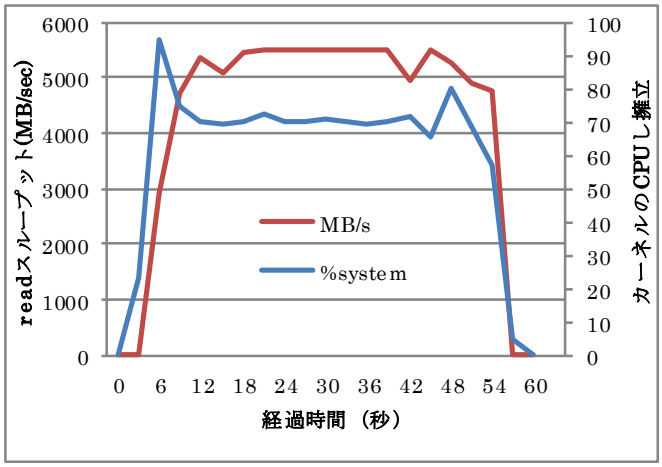

図 1 Direct I/O での read スループットとカーネルの CPU 使用率

表 2 実験環境

Server	Dell PowerEdge R910
CPU	Intel Xeon L7555 ×4 プロセッサ （計 32 物理コア，64 論理コア）
Memory	512GB
OS	CentOS 6.2 (2.6.32-279.22.1.el6.x86_64)
RAID Card	LSI MegaRAID SAS9286-8e
SSD	・Samsung 840 pro×8 (RAID0)×2 ・Intel SSD 520 series×8 (RAID0)×2 以上のソフトウェア RAID0
HDD	RAID0(×15)×RAID5(×7) 計 105 台

表 3 プロファイル結果

CPU: Intel Core/i7, speed 1861.95 MHz (estimated)			
samples	%	Image name	symbol name
103046	15.14	vmlinux	***read_cache_pages_invalidate_page***
36961	5.43	megaraid_sas	/megaraid_sas
36957	5.429	vmlinux	***__do_page_cache_readahead***
23113	3.395	vmlinux	sys_imageblit
21242	3.121	vmlinux	clockevents_notify
16033	2.355	vmlinux	***perf_read***
14048	2.064	vmlinux	***balance_dirty_pages***
13261	1.948	vmlinux	***file_read_actor***
12499	1.836	vmlinux	***__wake_up_bit***
11958	1.757	vmlinux	vbin_printf
11793	1.732	vmlinux	***__blockdev_direct_IO_newtrunc***
11783	1.731	vmlinux	***__zone_pcp_update***
10881	1.599	vmlinux	***__bio_copy_iov***
10395	1.527	vmlinux	***__kmalloc***
9672	1.421	vmlinux	scsi_eh_ready_devs
9563	1.405	raid0	/raid0
8434	1.239	vmlinux	scsi_send_eh_cmnd
8414	1.236	vmlinux	s_show
8330	1.224	vmlinux	***submit_page_section***
8262	1.214	vmlinux	scsi_try_host_reset
7647	1.123	vmlinux	***generic_file_aio_read***
6670	0.98	vmlinux	***dio_bio_add_page***
…	…	…	…

2.1.2. Linux OS におけるストレージアクセス

Linux OS におけるストレージへのアクセス方法およびシステムコール・ライブラリの関係について簡単に説明する.

アクセス方法には, O_SYNC や O_DIRECT のようなフラグを用いずにファイルを読み書きする方法（標準モード), 書き込み処理の同期をとる方法(同期モード), データ転送の要求により, プロセスの実行が中断しないようにする非同期 I/O（非同期モード）や先述した Direct I/O（Direct I/O モード）によるユーザ空間への直接転送にする方法が主に使用される.

同期モードは, 書き込み時に, ストレージにデータが書き込まれるまでに write()システムコールを発行したプロセスの実行を中断するために使われる.

非同期モードは, アプリケーション側の処理を中断させずに, バックグラウンドで I/O 処理が行われる(ノンブロッキング). 実装上, POSIX API と Linux カーネルのシステムコールが存在するが, システムコールを用いる場合, Direct I/O での転送となり, バッファのアドレスが 512B 単位でアライメントされている必要がある.

Direct I/O モードにおいては, 常にバッファのアドレスが 512B 単位でアライメントされている必要がある.

表 2 の実験環境における I/O 性能に関して, 表 4 にまとめておく.

表 4 Linux OS におけるストレージのアクセス性能

	Read 時のスループット(MB/s)
標準モード	1700MB/sec
非同期モード	1700MB/sec
Direct I/O モード	5000～5500MB/sec

標準や非同期モードに比べ, スループットに関しては, Direct I/O が最も性能が良い.

2.2. SSD のアクセスパターン毎の性能

SSD のアクセスパターンごとの性能について確認しておく. ストレージのベンチマークツールである fio[15]を用いて, SSD の性能を測定した結果をに示す.

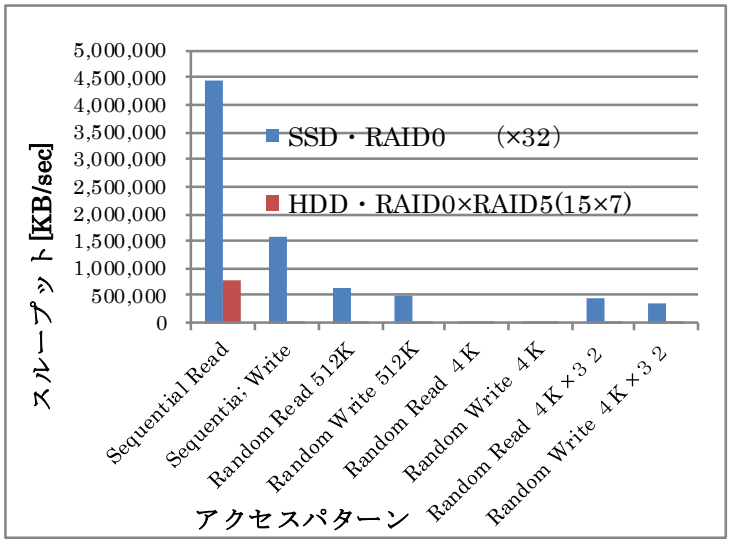

図 2 SSD・HDD のアクセスパターン及びスループット

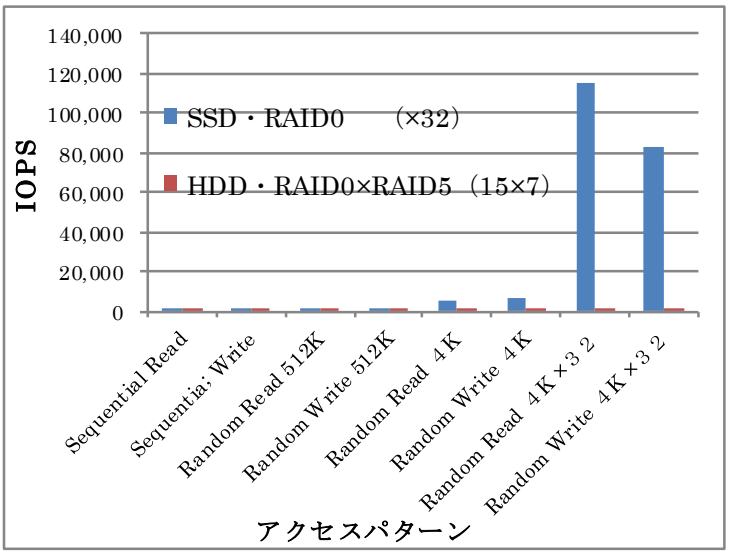

図 3 SSD・HDD のアクセスパターン及び IOPS

SSD の場合，ランダムアクセスの性能が HDD に比べ高速であるという特徴があるものの，シーケンシャルなアクセスパターンの方がランダムなアクセスパターンよりもスループットでは7~150倍ほど優れているという結果を得た．SSD において，シーケンシャルアクセスがランダムアクセスよりも効率が良い要因としては，SSD のコントローラの影響や，主にキャッシュを利用したプリフェッチが効果的に機能していることが考えられる．

図 2，図 3 および Linux カーネルの I/O 処理より，高速なストレージ環境においては，I/O のスループット性能が最も高いシーケンシャルアクセスで，CPU ボトルネックの問題が発生すると考えられる．

2.3. DBMS のクエリにおける議論

2.3.1. 実クエリの特性

前項において，SSD の性能においてシーケンシャルなアクセスパターンでの優位性を確認した．本項では，DBMS で用いられる具体的なクエリについて，簡単な実験結果とともに議論する．

昨今，企業などで盛んに行われているビッグデータの解析は，OLAP（オンライン分析処理）と呼ばれ，トランザクション処理が中心の OLTP（オンライントランザクション処理）とは異なり，読み込み処理が中心であり，特定のカラムのデータへの連続アクセスといった特徴がある．

表 5 に OLAP のベンチマークである，TPC-H のクエリ 1 を示す．このクエリでは，where 句においてカラムの属性が *l_shipdate* の値で比較を行い，最終的に請求額や出荷額の合計を計算する．*l_shipdate* はインデクスが張られており，インデクスのキーの値から絞ることも可能である．

しかし，表 6 に示すように，あえてインデクスを用いず *lineitem* テーブルをシーケンシャルアクセスでフルスキャンするほうが，インデクスを用いるよりもクエリの処理レイテンシははるかに短い．つまり，実クエリにおいてもシーケンシャルアクセスが有効な場合があることが確認できる．今回は where 句における単純な比較であるが，External Sort や Join といった DBMS クエリ中のアルゴリズムにおける read/write でも同様な事象が発生することが予想される．前項で述べた SSD の I/O の特性も考慮した上で適切なアルゴリズムを選択することが重要であると考えられる．

表 5　TPC－H クエリ 1

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as
sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) *
   (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
  from
        lineitem /*! ignore index (i_l_shipdate) */
  where
        l_shipdate <= date'1998-12-01' –
  interval :1 day (注：interval は 60~120)
  group by
        l_returnflag,
        l_linestatus
  order by
        l_returnflag,
        l_linestatus;
```

表 6 TPC-1 のクエリの結果(MySQL-5.6.8rc, スケールファクタ:4)

インデクスの指定	クエリレスポンスタイム
インデクスなし	4m21s
インデクスあり	40m 以上

2.3.2. DBMS の物理実行プランにおける議論

DBMS におけるクエリ実行は，SQL の解析後，オプティマイザによる論理･物理実行プランの作成を経て，実行エンジンによって実行される．実行エンジンでは，テーブル中のタプルを順に読んだり，特定のカラム属性の値に対し，インデクスを用いて比較したりするなどの Scan 処理をはじめ，Sort や Join 処理が含まれる．SQL クエリの実行にあたっては，個々の実行インターフェース及びその内部のアルゴリズムのコスト及び実装が大きく影響する．

従来の HDD を対象とした DBMS においては，その実行プランのコストに，ディスク I/O を最も重視してきた．これはメモリアクセスに比べて HDD の方が圧倒的に遅いためであり，その他のリソースからの影響は無視できる程度だったためである．

実行エンジンにおける，DBMS のオペレータの内，ここではソートマージ Join を例に考える．

- External Sort

ソートマージ join で用いられる External Sort についてアルゴリズムを述べる.

1. まずソートすべきファイルの先頭から，メモリに格納できるだけのデータをシーケンシャルに読み込む．このデータをメモリ上のソート手法で並べ替え，その分をサブリストとして別のファイルに先頭から書きだしていく．元のファイルになるまでこれを繰り返す．
2. 次に，書きだしたサブリスト（分割したファイル）の全部のデータを先頭から少しずつ読み込んで，1タプルずつ取り出し，そのうち一番先頭になるタプルを取り出す．このタプルが先頭となる．
3. 先頭をとりだしたサブリストは新たにレコードを読み込み，1から繰り返す．

- ソートマージ Join

今2つのリレーション（テーブル）$R(X,Y)$，$S(Y,Z)$において，（X,Y,Z はそれぞれのカラムの属性を示す）Y を結合キーとしてソートマージ Join を考える．簡単のため R,S にもそれぞれにおいて Y の値で重複するタプルはないものとする．

1. リレーション R をソートする．ソートにおけるキーは Y で，マージソートを行う．リレーション S も同様にソートする．
2. R と S をソートしたものから1つずつ順にタプルを取り出し，以下に従って調べていく．
   (ア) (R の Y の値) < (S の Y の値)ならば，R から次のタプルを取り出し2を最初からやり直す．
   (イ) (R の Y の値) > (S の Y の値)ならば，S から次のタプルを取り出し2を最初からやり直す．
   (ウ) (R の Y の値) ＝ (S の Y の値)ならば，join して結果として出力する．
   (エ) R または S のタプルの最後まで調べたら,これで全ての結果が出力されたことになる．

   ソートマージ Join に必要なストレージアクセスの回数は，各々のリレーションのブロックサイズをそれぞれ，B(R)、B(S)とすると，マージソートが高々2回のブロックアクセスであり，最後の書き込みが1回となるので，アクセスコスト：C_{acs}は

$$C_{acs} = 5(B(R) + B(S)) \qquad (1)$$

と表される．([14]より)

また，メモリの使用量:M は，R と S の最大値の平方根で表せる．

$$M = \sqrt{\max(B(R),\ B(S))} \qquad (2)$$

実際の DBMS においても，クエリのコストはストレージへのアクセスコストでおよそ決定されている[12][13].

3. 実験

本節では，DBMS クエリ及びオペレータの一部を Linux OS のシステムコールを用いて実装し，実験・測定を行い，ストレージの I/O 性能及び Linux OS の影響について考察する．

3.1. システムコールによる TPC-H クエリの実装

実験データとして TPC-H の LINEITEM 表を用いて，TPC-H のクエリ1（表5）を実装した．スケールファクタ（SF）を 10 から 100 まで変化させ，Linux OS のストレージアクセスに関するシステムコール毎に，クエリ処理性能を測定した．

ストレージへのアクセス方法は，以下のとおりである．なお全てのアクセスは Direct I/O で行う．

- read()による同期 I/O 処理
- Linux カーネルの非同期 I/O(aioread)
- pread による，ファイルオフセットを指定したマルチスレッド（pthread）による処理

いずれもアクセスのブロックサイズは 2GB である．

また pread による処理では，スレッドを 2,4,8,16 と調整し，計測を行った．

3.2. 実験結果・考察

実験結果を図 4 に示す．

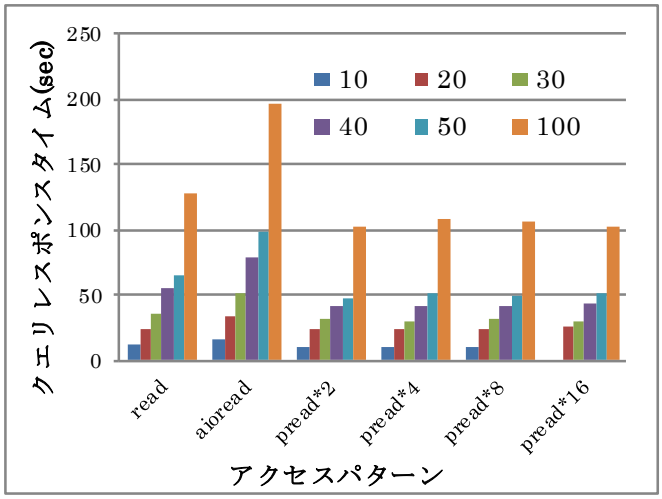

図 4 クエリレスポンスタイム

システムコールの read()は,1スレッドの処理に関わらず，pread による処理よりも高速なケース(SF=20,30)が存在した．1スレッドによる非同期 I/O の aioread については，クエリ処理のプロセス本体とは別に I/O 処理を行うが，今回のケースでは I/O スループットが，元々read()に比べ低いことが影響していると考えられる．またマルチスレッド下での pread での処理は，read()ほど先読みの効果が働かず，スレッドのオーバーヘッドが大きいことも影響していると考えられる．今回は，各スレッドのメモリ制限などは詳細には設定していな

いため，メモリなどのリソースの観点からも read による処理が効率的であると考えられる．

以上から，現在の Linux OS の実装上では，システムコールの read を用いて，非同期 I/O な処理をするスレッドと I/O 以外の DBMS クエリの処理を行うスレッドによる処理が効率的であると推察される．

3.3. 非同期 I/O 処理と DBMS 処理の関する実験

ここでは DBMS クエリ中のアルゴリズムとして，ソートマージ Join を対象とする．ソートマージ Join のアルゴリズム及び理論上のコストモデルは，2.3.2 で述べたものを想定している．本実験ではアルゴリズム中の各 I/O の性能を測定し，アクセスコストの妥当性を検証することが目的である．

ソートマージ Join の I/O に関する処理は，全部で 5 つに大別できる．

① 全データを含むファイルからメモリへの read（非同期）
② メモリ上の sort 後，サブリストへの write（同期）
③ メモリで mergesort を行う際，各サブリストから行う read（同期）
④ メモリ上の先頭データの write（同期）
⑤ 結合処理を行う際の read（非同期）

以上の I/O 処理について測定する．

なお，実験データは TPC-H の LINEITEM 表と ORDER 表（それぞれスケールファクタは１０）を用いて，l_orderkey と o_orderkey を結合キーとする．また，I/O バッファ：2GB（同期 I/O），2GB×4（非同期 I/O）とする．

3.3.1. 実験結果

実験結果を表 7 に示す．

表 7 アルゴリズム中の I/O 性能

I/O 処理	スループット MB/s（最高値）
①	**4000~5000**
②	2000
③	100
④	≒0
⑤	**4000~5000**

実験結果の表から，I/O 処理の①，⑤においては Linux の先読み機構が充分に機能して CPU バウンドな処理であり，残りの②，③，④は I/O バウンドな処理となっていることが確認できた．したがって DBMS の Sort や Join のアルゴリズムにおいても，将来的な高速ストレージ環境においては，従来のストレージへのアクセスコストのみを想定したコストモデル（2.3.2 の（２））だけでは不十分であると考えられる．今後は他のアルゴリズムを含め，I/O 処理に占める CPU 時間の増加の影響を考慮したコストモデルを検討する必要がある．

4. 関連研究

本節では，既存研究と本研究との関連についてまとめる．

デバイスとしての SSD については，Chen ら[2]の研究で実機の計測を含め詳細な説明がなされている．Chen らは SSD の性能及びその特性について，read におけるレイテンシとアクセスパターンの相関をはじめ，ランダム write の性能と負荷の関連などの実験を行っている．その際，SSD の read レイテンシにおいて，シーケンシャルなアクセスパターンの方が，ランダムやストライドなアクセスパターンよりも６５％程優れているという実験結果を得ている．この結果の要因としては，主に SSD のコントローラが低コストでデータをプリフェッチする点であると述べている．またシーケンシャルなアクセスの方が性能的に優位であるという点については，前節で行った簡易的な実験結果と一致する．

Lee ら[3]は，SSD 環境の DBMS における実アプリケーションレベルの性能を予測するため，TPC-B によるトランザクションの処理性能の測定し，トランザクション数が増加するにつれ，高速なストレージ環境における CPU ボトルネックの発生を指摘している．

ストレージに SSD を採用した DBMS について，そのクエリ最適化に関する研究では，Tsirogiannis ら[6]が DBMS の page scan において，SSD のランダムアクセス性能を生かして転送データを削減することで効率的な scan アルゴリズムを提案し，またこのアルゴリズムを用いて高速な multi-way join を実装している．Bausch[7]らの研究では，PostgreSQL[13]のクエリ中の Sort や Join アルゴリズムについて，HDD から SSD に変更した場合の影響について考察している．その際，単一レコードに対するランダムアクセスや種々の Join を用いたクエリでは性能改善が見られるが，全レコードから特定の条件でフェッチするクエリでは，それほど改善は見られないとしている．つまり，一様に全てのアルゴリズムが改善されるわけでないと結論づけている．また Bausch ら[5]は，SSD の read/write の非対称性に着目し，PostgreSQL 上のクエリの I/O におけるコストモデルを変更した上で TPC-H を測定し，従来の HDD を対象としたコストモデルを用いた場合とは異なり，概ね改善された結果を得ている．

具体的には，各 DBMS のクエリコストについて以下の２つについて変更を行っている．

- 単にシーケンシャル・ランダムなアクセスパターンによる２通りのコストモデルだけでなく，read/write の区別も含めた４通りの重みのコストモデルに拡張する
- DBMS クエリ中の Sort アルゴリズムのブロック

のアクセスパターンにおいて，シーケンシャル・ランダムの比を変更する．

クエリ最適化に関する研究のいずれも，主に HDD を対象とした従来のコストモデルに対し，SSD の I/O の特性を考慮した改良を行なっているが，将来的な高速なストレージ環境における CPU ボトルネックに関する議論については不十分である．特に 2 節で述べたような，OLAP で頻出であるシーケンシャルなアクセスのクエリに対処するためには，CPU の影響まで考慮し，並列化を含めたコストモデルの検討・改善の余地があると考えられる．

5. おわりに

本稿では，将来的に予想される高速ストレージを想定し，Linux OS を対象として，I/O 処理における CPU ボトルネックについて議論及び考察を述べた．現状の Linux カーネルの I/O 処理では，先読みやキャッシュが機能する場合に，DBMS のクエリ処理時も含め，I/O 性能が発揮され，効率的な処理が可能となっている．今後ストレージ性能が向上するにつれ，DBMS のクエリ処理の方法も変わることが予想され，OS 側の改善お必要となると考えられる．

今後の課題・展望として，より詳細なプロファイルやアクセスパターンの解析の実施をはじめ，DBMS のコストモデルの改善などが考えられる．

今回得られた知見をもとに，この課題に取り組んでいきたい．

謝辞

本研究の一部は，文部科学省「Web 社会分析基盤ソフトウェアの研究開発」および科学研究費（挑戦的萌芽研究 No.23650053）によるものである

参 考 文 献

[1] 日立ビッグデータ利活用 :http://www.hitachi.co.jp/products/it/bigdata/(2013.2.14 アクセス)

[2] Feng Chen, David A.Koufaty, Xiaodong Zhang: “Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives,” *In Proc. of the 11th Int’l joint Conf. on Measurement and modeling of computer systems, 2009.*

[3] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, Sang-Woo Kim: “A case for flash memory ssd in enterprise database applications,” *In Proc. of the the 2008 ACM SIGMOD Int’l Conf. on Management of data, June 2008.*

[4] S. Pelley, T. F. Wenisch, and K. LeFevre: “Do query optimizers need to be ssd-aware?,” *In Proc.of the ADMS'11, 2011.*

[5] Daniel Bausch, Ilia Petrov, Alejandro Buchmann, “Making cost-based query optimization asymmetry-aware,” *In Proc. of the 8th Int’l Workshop on Data Management on New Hardware, 2012,* pp.24-32.

[6] Dimitris Tsirogiannis, Stavros Harizopoulos, Mehul A. Shah, Janet L. Wiener, Goetz Graefe: “Query processing techniques for solid state drives,” *In Proc. of the 35th SIGMOD Int’l Conf. on Management of data, 2009.*

[7] Daniel Bausch, Ilia Petrov, Alejandro Buchmann: “On the Performance of Database Query Processing Algorithms on Flash Solid State Disks,” *In Proc.of the 22nd Int’l Workshop on Database and Expert Systems Applications, 2011,* pp.139-144.

[8] Transaction Processing Performance Council. TPC benchmark H---standard specification. : http://www.tpc.org/tpch. (2012.12.4 アクセス)

[9] OSDLDBT Project. Database test 3 (dbt-3): http://osdldbt.sourceforge.net. (2012.12.4 アクセス)

[10] J.Do and J. M. Patel:”Join processing for flash SSDs remembering past lessons,” *In Proc. of the 5th Int’l Workshop on Data Management on New Hardware, 2009.*

[11] Mehul A. Shah, Stavros Harizopoulos, Janet L. Wiener, Goetz Graefe: “Fast scans and joins using flash drives,” *In Proc.of the 4th Int’l workshop on Data management on new hardware, June 13-13, 2008.*

[12] MySQL : http://www-jp.mysql.com(2012.12.4 アクセス)

[13] PostgreSQL : http://www.postgresql.org/(2012.12.4 アクセス)

[14] Hector Garcia-Molina, Jeffrey D.Ullman, Jennifer Widom:”DATABASE SYSTEMS The Complete Book,” pp.619-758, Pearson Education, 2008.

[15] Fio : http://freecode.com/projects/fio(2012.12.4 アクセス)

[16] Daniel P.Bovet 著，高橋浩和 監訳：詳解 Linux カーネル，オライリー・ジャパン(2007)．