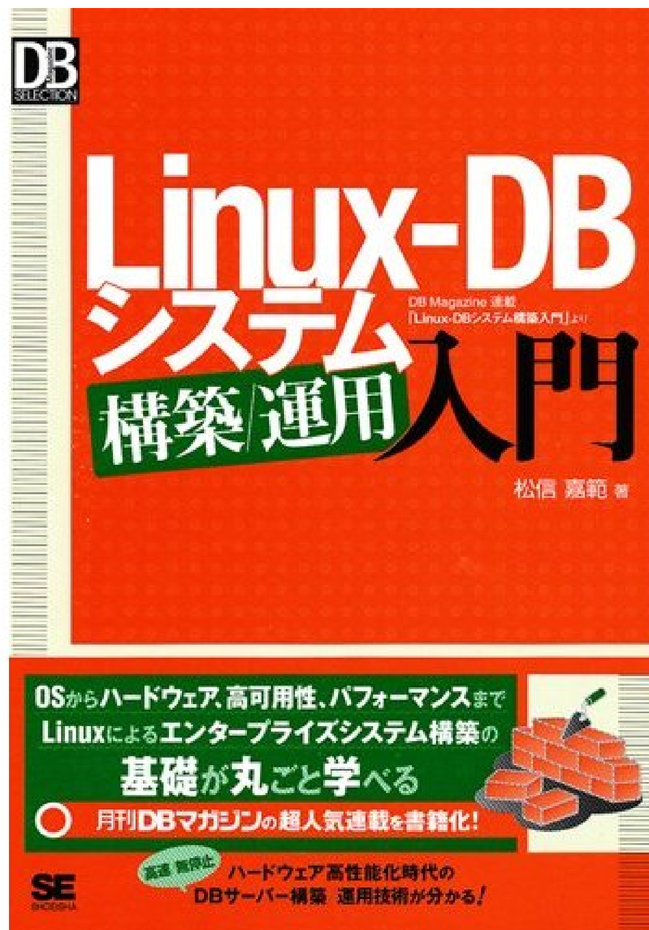
Linux/MySQLサーバーの パフォーマンスチューニング

松信 嘉範 (MATSUNOBU Yoshinori)
http://twitter.com/matsunobu
http://opendatabaselife.blogspot.com

Copyright 2009 Sun Microsystems inc

自己紹介



- Sun Microsystems所属 MySQLコンサルタント
- 2006年9月からMySQLコンサルタントとして勤務
- パフォーマンスチューニング、HA環境の構築、DBAトレーニング等お気軽にご相談ください

Copyright 2009 Sun Microsystems inc

今日のテーマ

- InnoDB(あるいはほかのDB)のブロックの内部構造や、列やインデックスの構造を理解して設計をする
- Linux上でのチューニングテクニックを理解する
- SSDの特性と注意事項を理解する

Copyright 2009 Sun Microsystems inc

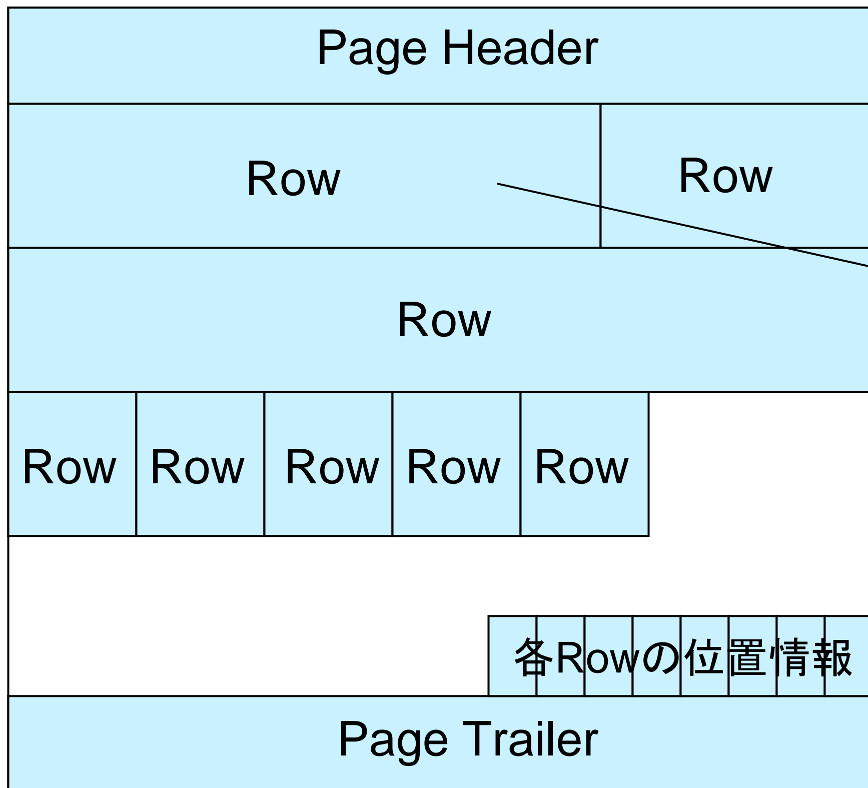
例：Blogエントリ用テーブル

```
CREATE TABLE diary (
  diary_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  user_id INT UNSIGNED NOT NULL DEFAULT 0,
  post_date TIMESTAMP NOT NULL,
  status TINYINT UNSIGNED NOT NULL DEFAULT 0,
  title VARCHAR(100),
  body TEXT,
  INDEX(user_id, post_date)
) CHARSET cp932 ENGINE=InnoDB;
```

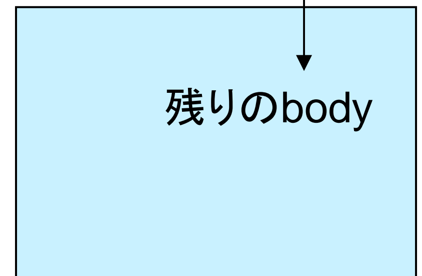
- **body列には日記の本文(1KB/行)**
- **それ以外の列は短い(50B未満/行)**
- **2000万レコード (20GB超)**

Copyright 2009 Sun Microsystems inc

InnoDBのブロック/レコード構造



diary_id | user_id | post_date
title | status | body (prefix) (768B)



1ブロック(ページ) = 16KB

I/Oの単位

Overflow Page(同じブロックに空きが無ければ別のブロックに格納)

Copyright 2009 Sun Microsystems inc

巨大なTEXT/BLOBはクエリ効率を悪化させる

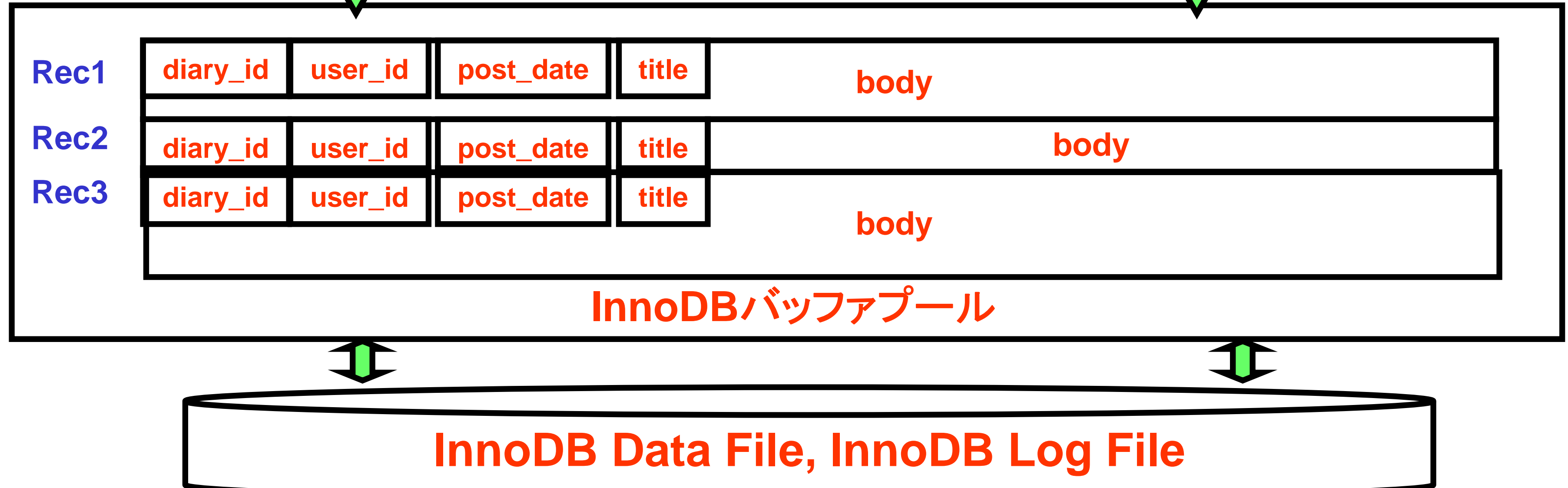
diary_id (BIGINT PK)	user_id (BIGINT, INDEX)	post_date (DATETIME, INDEX)	title (VARCHAR(100), INDEX(10))	body (TEXT)
1	5544321	2009/09/13 21:10:14	MySQL Clusterの新機能	…….(2000bytes)
2	5544321	2009/10/13 22:13:34	UEFA Champions League	…….(700bytes)
3	2345	2009/11/7 22:12:23	巨人・7年ぶりの日本一	…….(3000bytes)

SELECT user_id, post_date, title FROM diary WHERE diary_id=?

90% queries

SELECT body FROM diary WHERE diary_id=?

10% queries



- bodyを読まないクエリでも、そのレコードのbodyはInnoDBバッファプール上にロードされる
- そのbodyによって、キャッシュ済みのほかのレコード(ブロック)が追い出されてしまう

Copyright 2009 Sun Microsystems inc

1:1関連を考える

```
CREATE TABLE diary_head (
  diary_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  user_id INT UNSIGNED NOT NULL DEFAULT 0,
  post_date TIMESTAMP NOT NULL,
  status TINYINT UNSIGNED NOT NULL DEFAULT 0,
  title VARCHAR(100),
  INDEX(user_id, post_date)
) CHARSET cp932 ENGINE=InnoDB;

CREATE TABLE diary_body (
  diary_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  body TEXT
) CHARSET cp932 ENGINE=InnoDB;
```

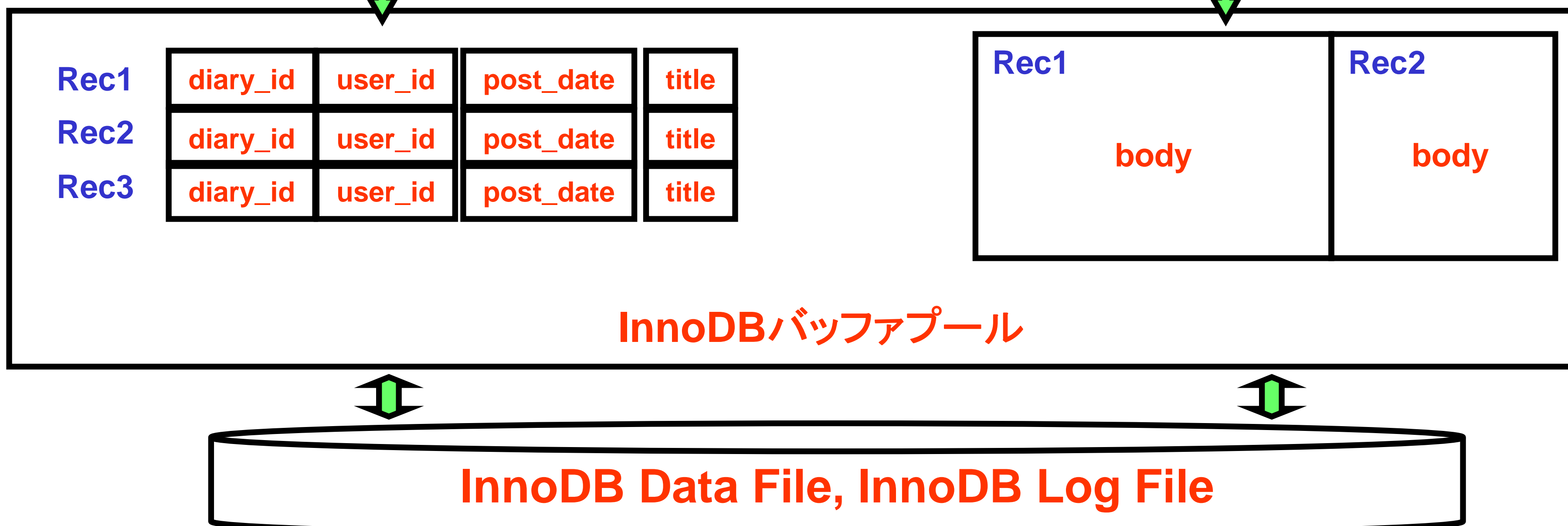
- diary_idを主キーとする2個のテーブル

- body列を持たないテーブル(diary_head)と、bobdy列だけを持つテーブル(diary_body)

- 正規化が崩れるので、一般的に非推奨

Copyright 2009 Sun Microsystems inc

1:1関連がなぜ効果があるのか



・bodyとそれ以外は異なるテーブルに属するため、bodyを読まないクエリを実行すればbodyはInnoDBバッファプールにはロードされない

・bodyが読まれる頻度が10分の1に低下するため、キャッシュから追い出されにくくなる

Copyright 2009 Sun Microsystems inc

テーブルサイズ

	レコードサイズ	主キー以外の インデックスサイズ
diary	23GB	620M
diary_head	930M	620M
diary_body	23GB	0

Copyright 2009 Sun Microsystems inc

クエリの実行効率

Body列を含むクエリの割合	通常テーブル (qps)	1:1関連 (qps)	比率
2%	227.54	3816.71	16.8
5%	235.04	2353.30	10.0
10%	246.87	1458.45	5.9
20%	276.47	832.89	3.0
33%	328.24	518.93	1.6
50%	435.17	353.29	0.8
100%	224.71	220.26	1.0

Copyright 2009 Sun Microsystems inc

1:1関連は一般的にどうなのか

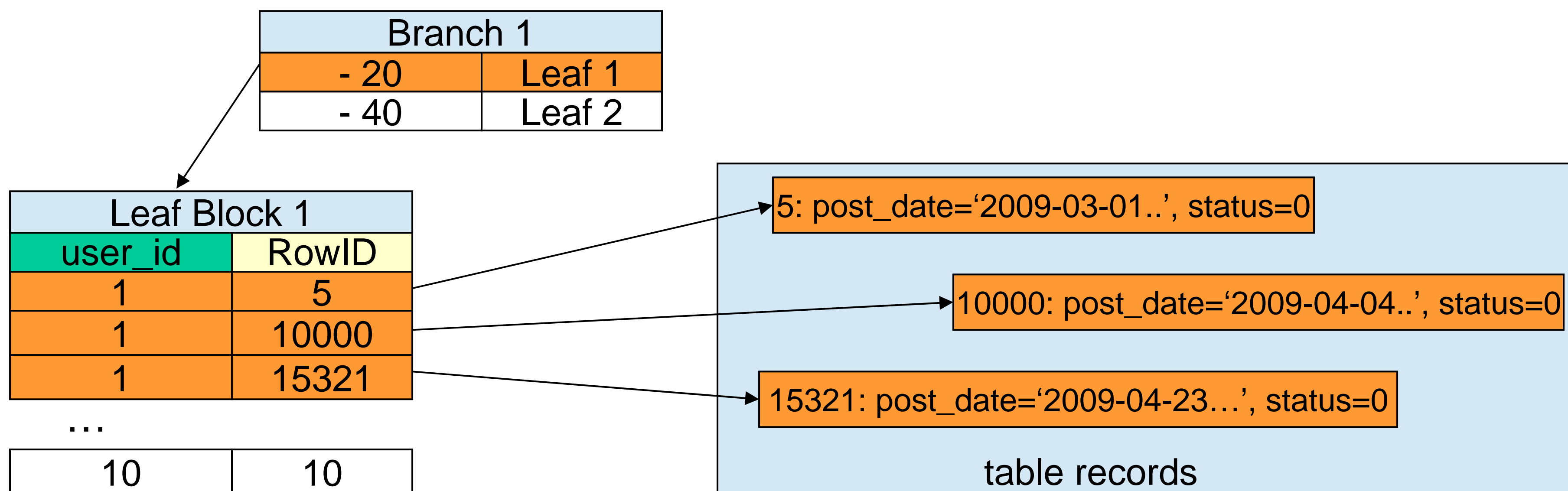
- 正規化を崩すので、必要ない限り使うべきではない
- 次に説明するCovering Indexでも同様の効果が得られる
  - (diary_id, user_id, post_date, title)でマルチカラムインデックス
  - 正規化を崩さないし余計なテーブルも要らないので効果的
- 巨大な列を除去するのは一般的に良い考え
  - テーブルサイズが小さくなればすべての性能が上がる
  - 巨大な列は
    - 別のテーブルに移す
    - Durable KVS (Tokyo Cabinet等)に移す
- 巨大な列の扱いを最適化してくれるストレージエンジンに注目
  - Falcon
    - TEXT型の列を別領域に保存
  - PBXT
    - 一定以上の長さの列を別領域に保存

Covering Indexを活用する

Copyright 2009 Sun Microsystems inc

非一意検索は意外と遅い

```
SELECT diary_id FROM diary WHERE user_id=1 AND status=0 AND
post_date >= '2009-03-01 00:00:00';
```



- user_id=1を満たすレコードが100個あれば、100回のランダムI/Oが発生しうる
- リーフブロックに対するI/Oは1回で済む
- I/O回数の見積もりとしては、リーフブロックに対して1回、データ領域に対してN回
- HDDでは、1秒あたりに処理できるランダムI/Oの回数はせいぜい数百回程度

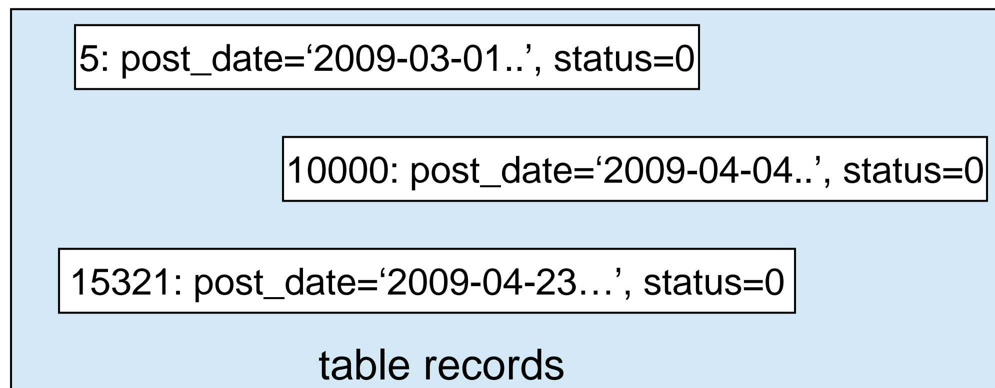
Copyright 2009 Sun Microsystems inc

Covering Index (インデックスだけを読む検索)

```
SELECT diary_id FROM diary WHERE user_id=1 AND status=0 AND
post_date >= '2009-03-01 00:00:00';
```

Branch 1	
20	Leaf 1
- 120	Leaf 2

Leaf 1			
user_id	post_date	status	RowID
1	2009-03-29	0	4
1	2009-03-30	0	10000
1	2009-03-31	0	5
1	2009-04-01	0	15321
1	2009-03-31	0	100
1	2009-03-30	0	200
1	2009-04-13	0	20000
..	…	..	400

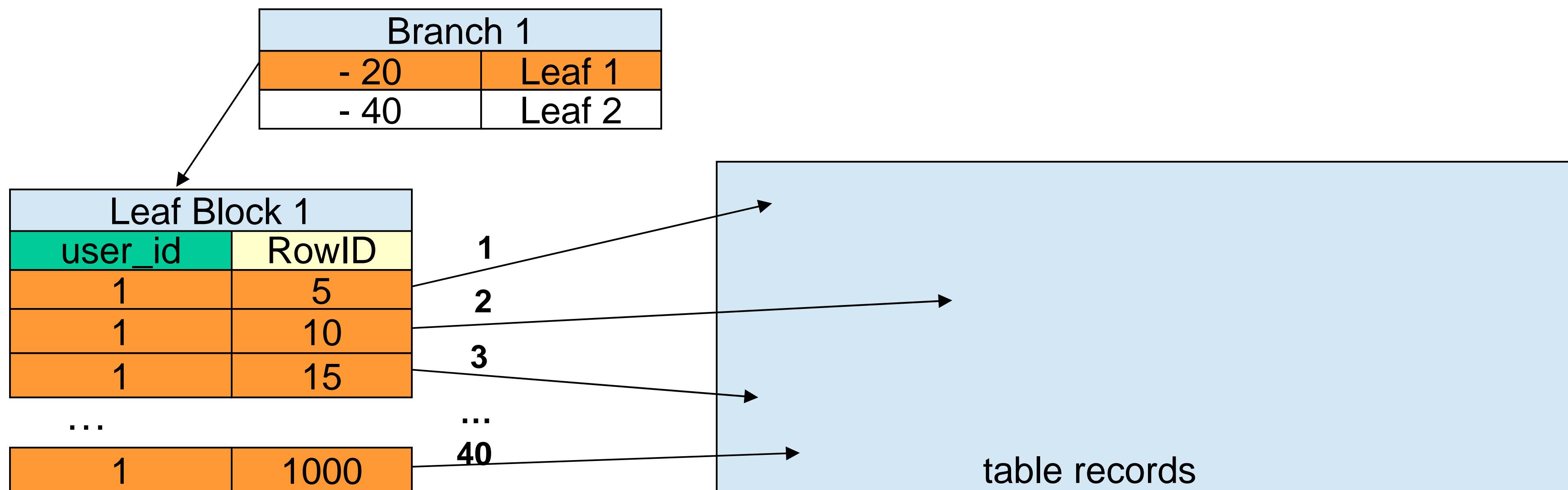


・そのクエリの実行に必要な列が、すべて1個のインデックスにおさまっている場合、インデックスだけを読めばSQL文の実行が完結する
・InnoDBではRowID=主キーなので、このクエリがCovering Indexになる
・データ領域へのランダムI/Oが発生しないので、非常に効率が良い
・status列は絞込みには役に立っていないが、これがインデックスに含まれていないとCovering Indexにはならない

Copyright 2009 Sun Microsystems inc

LIMIT句と範囲検索

```
SELECT diary_id FROM diary WHERE user_id=1 AND status=0 AND
post_date >= '2009-03-01 00:00:00' ORDER BY post_date LIMIT 30, 10;
```



・通常の範囲検索では、LIMITの値が大きくなると、ランダムI/Oの回数も比例して増えるのでその分実行時間がかかる

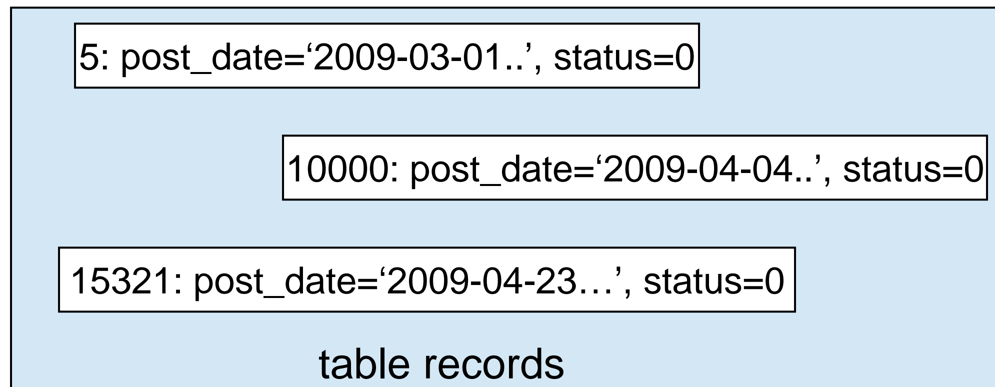
Copyright 2009 Sun Microsystems inc

Covering Index (インデックスだけを読む検索)

```
SELECT diary_id FROM diary WHERE user_id=1 AND status=0 AND
post_date >= '2009-03-01 00:00:00' ORDER BY post_date LIMIT 30, 10;
```

Branch 1	
20	Leaf 1
- 120	Leaf 2

Leaf 1			
user_id	post_date	status	RowID
1	2009-03-29	0	4
1	2009-03-30	0	10000
1	2009-03-31	0	5
1	2009-04-01	0	15321
1	2009-03-31	0	100
1	2009-03-30	0	200
1	2009-04-13	0	20000
..	…	..	400



・Covering Indexでは、LIMITが増えてもI/O負荷はほとんど変わらない

Copyright 2009 Sun Microsystems inc

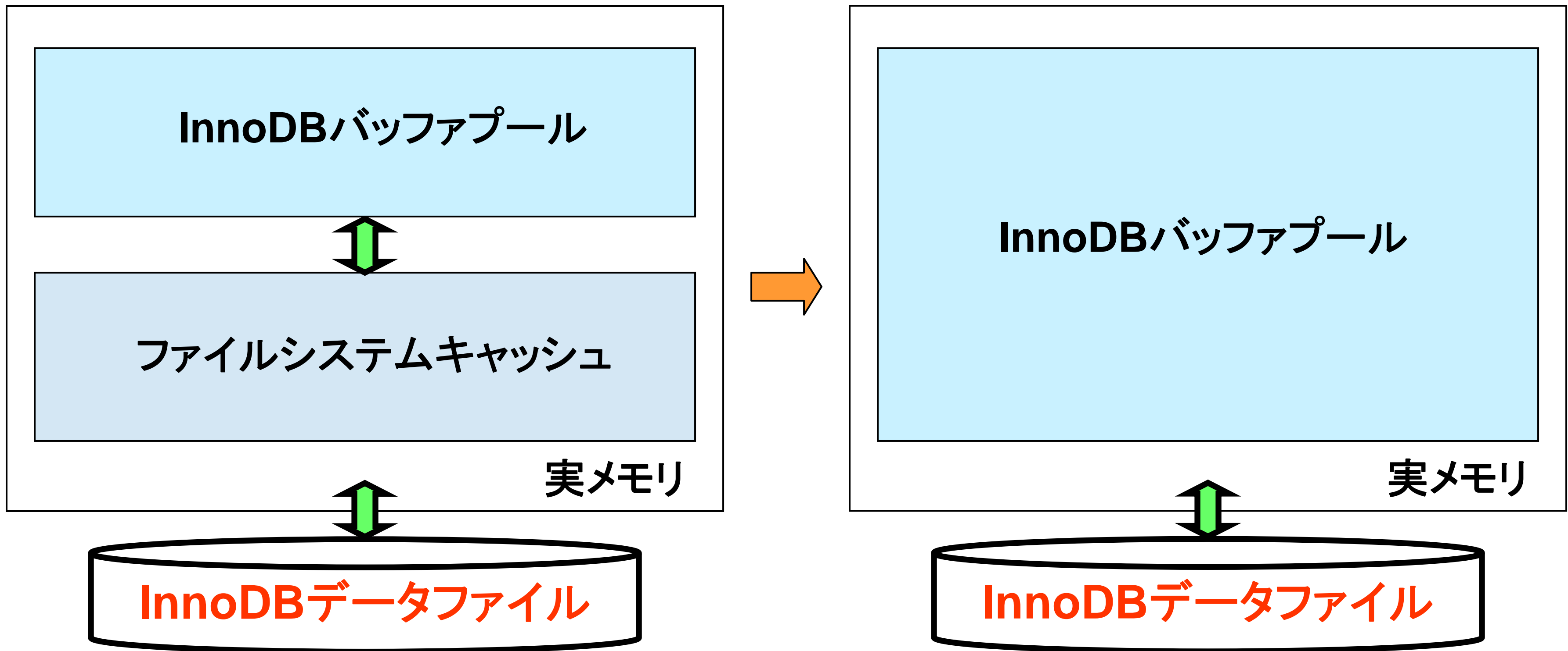
EXPLAINでの確認

```
> explain select count(ind) from t
           id: 1
  select_type: SIMPLE
        table: t
         type: index
possible_keys: NULL
          key: ind
      key_len: 5
          ref: NULL
         rows: 100000181
        Extra: Using index

mysql> select count(ind) from d;
+----------------+
|   count(ind)   |
+----------------+
|      100000000 |
+----------------+
1 row in set (15.98 sec)
```

```
> explain select count(c) from t
           id: 1
  select_type: SIMPLE
        table: t
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 100000181
        Extra:

mysql> select count(c) from d;
+------------+
| count(c)   |
+------------+
| 100000000 |
+------------+
1 row in set (28.99 sec)
```

Linux上でのチューニングテクニック

メモリを十分に取り、ダイレクトI/Oを活用する



innodb_flush_method=O_DIRECT

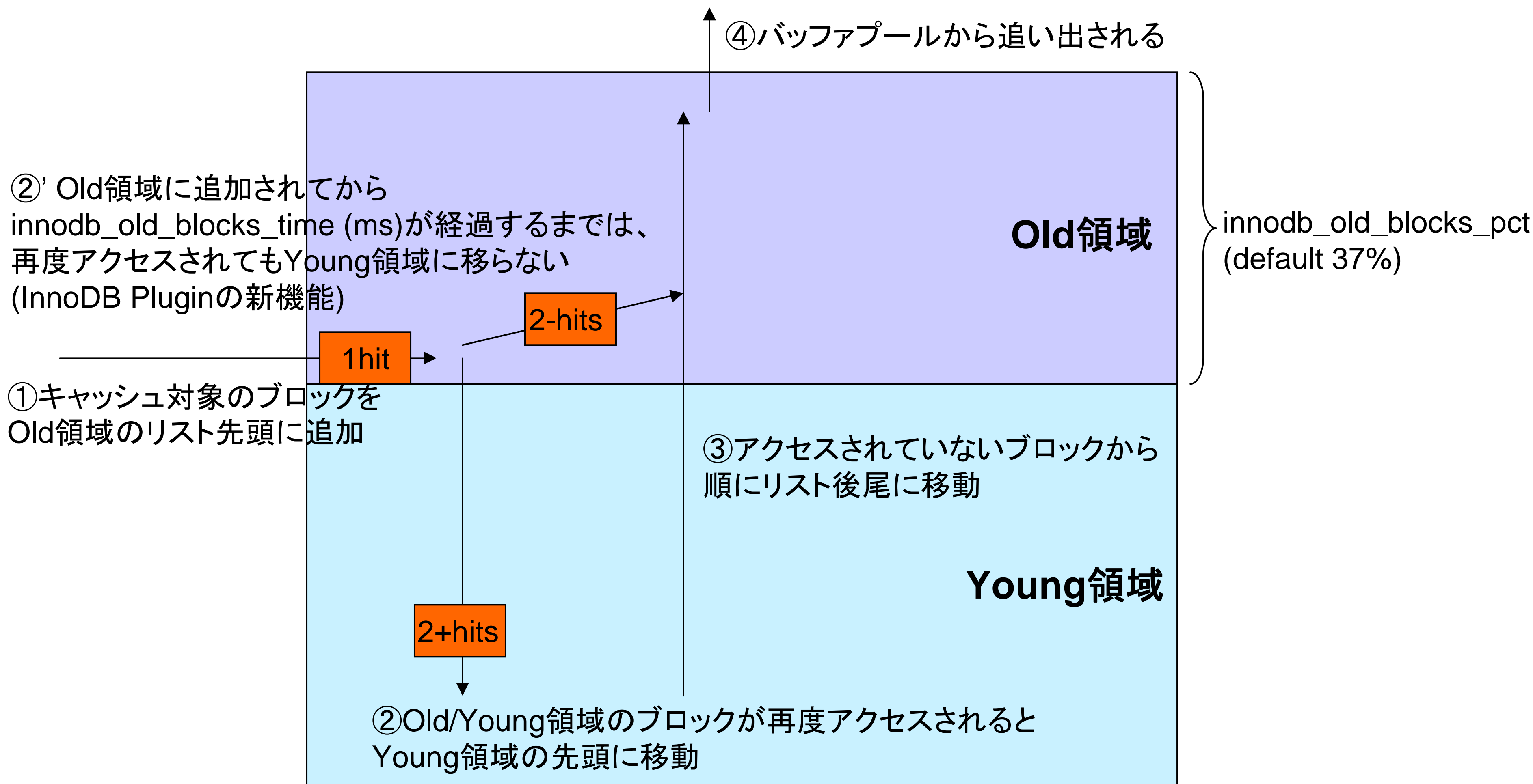
Copyright 2009 Sun Microsystems inc

メモリ上へのキャッシュ効率を意識する

- Innodb_flush_method=O_DIRECT
- Innodb_buffer_pool_size = 11G

- オンライン処理の後に、巨大なテーブルに対してフルスキャンをするのは問題がある
- InnoDB Plugin 1.0.5の新機能を生かす
  - SET GLOBAL innodb_old_blocks_time = 1000;
  - -- mysqldump等によるフルテーブルスキャン系の処理
  - SET GLOBAL innodb_old_blocks_time = 0;

InnoDBブロックのライフサイクル



OOM Killerに注意する

- Linuxではスワップサイズをゼロにできるが…
- 実メモリとスワップを両方使い切ると、OOM Killerが走る
- OOM Killerによって殺されるには、ある程度の時間がかかる
  - その間はアクセスをほとんど受け付けてくれない
- スワップサイズをある程度取って、OOM Killerを防ぐ

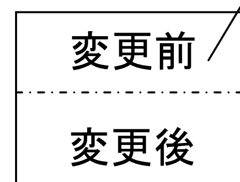
実プロセスがスワップアウトされることを防ぐ

- ダイレクトI/Oなら、実プロセス内にデータが置かれる
- 実メモリが足りなくなると…
  - A:プロセスをスワップアウトする
  - B:ファイルシステムキャッシュを縮小する
- A、Bのどちらが優先されるかはvm.swappinessで決まる
- 0ならB優先、100ならA優先 (デフォルト60)
- 当然、B優先にすべき
- # echo 0 > /proc/sys/vm/swappiness = 0

Copyright 2009 Sun Microsystems inc

ファイルシステム

- ext3
  - 最も使われている。安全策を取るなら最も良い
  - 巨大なファイルの削除に時間がかかる
  - ジャーナリング方式に3種類
    - writeback
    - ordered(デフォルト)
    - journal
    - データを書いている途中にクラッシュすると、ブロックが中途半端な状態になる可能性がある
    - InnoDBならデフォルトで防げる(doublewrite buffer)。PostgreSQLでもfull_page_writesによって防げる。つまりどのオプションでも安定性に大差無いが、journalだと遅いのでorderedかwritebackが良い
  - dir_index, noatime(relatime)



- xfs
  - 巨大ファイルの削除に時間がかからない
  - ダイレクトI/Oを使う場合、1個のファイルに並列で書き込みが可能

ファイルシステム

- ext2
  - ジャーナリングが無いため高速
  - fsckに非常に時間がかかる
  - 冗長化構成を組んでいる場合、あえてext2にして高速化を狙うことがある
- btrfs (zfs)
  - コピーオンライト
  - トランザクション対応なので、中途半端な状態で更新されることが無い
  - スナップショット・バックアップをオーバーヘッド無しで取れる

Copyright 2009 Sun Microsystems inc

統計ツールの使い方をおさえておく

- sar
- vmstat
- mpstat (CPUコア単位の負荷状況)
- iostat (IOPS、ビジー率)

- そのほかのツール
  - iotop (プロセス単位でI/O量を取る: kernel 2.6.20以降)
    - /proc/self/ioを読めばできる
    - Kernel 2.6.20未満でも以下の方法で取れる
      - echo 1 > /proc/sys/vm/block_dump
      - dmesg -c
    - 現実的にはあまり意味ない

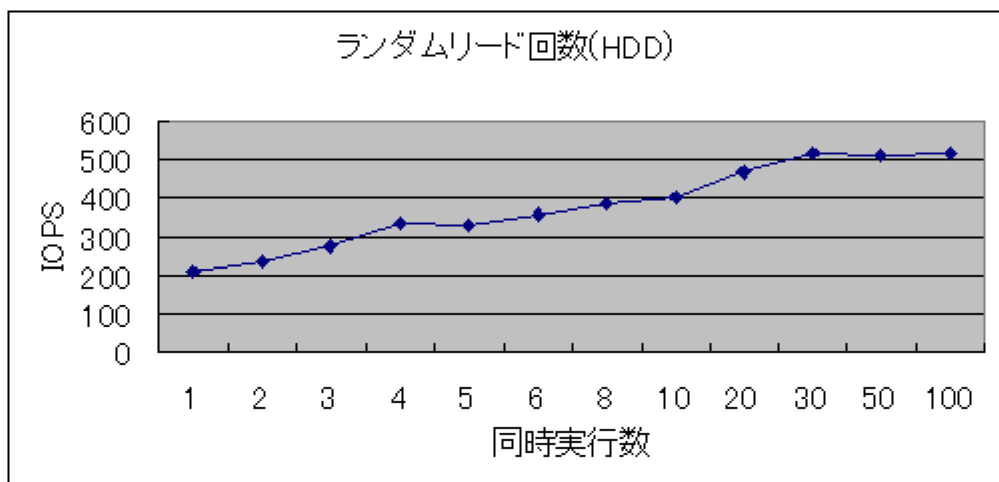
ネットワーク統計

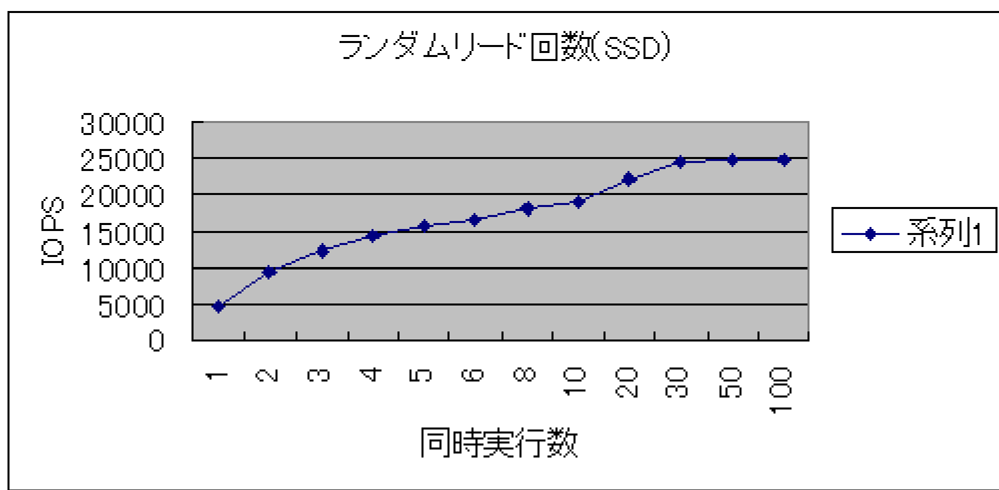
- /proc/net/dev にインターフェイスごとの
  転送量が出るので、ここを解析すれば良い

- mtstat
- 自作してもいい

SSDの時代

SSDはとても速い





SSD製品を選ぶ時に注意したいこと

- 書き込み性能は製品による差が激しい
  - ライトキャッシュ、ウェアレベリング、TRIM
  - プチフリーズ問題
- 基本的に、まだ地雷
  - Intel SSD: G2の最新ファームウェアでOSが起動しなくなるとか。。
  - ベンダー製サーバー向けSSDはかなりテストされています
- ライトキャッシュ必須
  - バッテリーで守られていることが必要
  - RAIDコントローラに任せるものと、SSD自身で持つ(キャパシタ)ものがある
  - RAIDコントローラに任せる場合、「RAIDコントローラがSSDに最適化」されていないといけない
    - HDDと同じ感覚で書き込んでしまい(ウェアレベリング無視)性能が伸びないという現象が。。

SSD製品を選ぶ時に注意したいこと

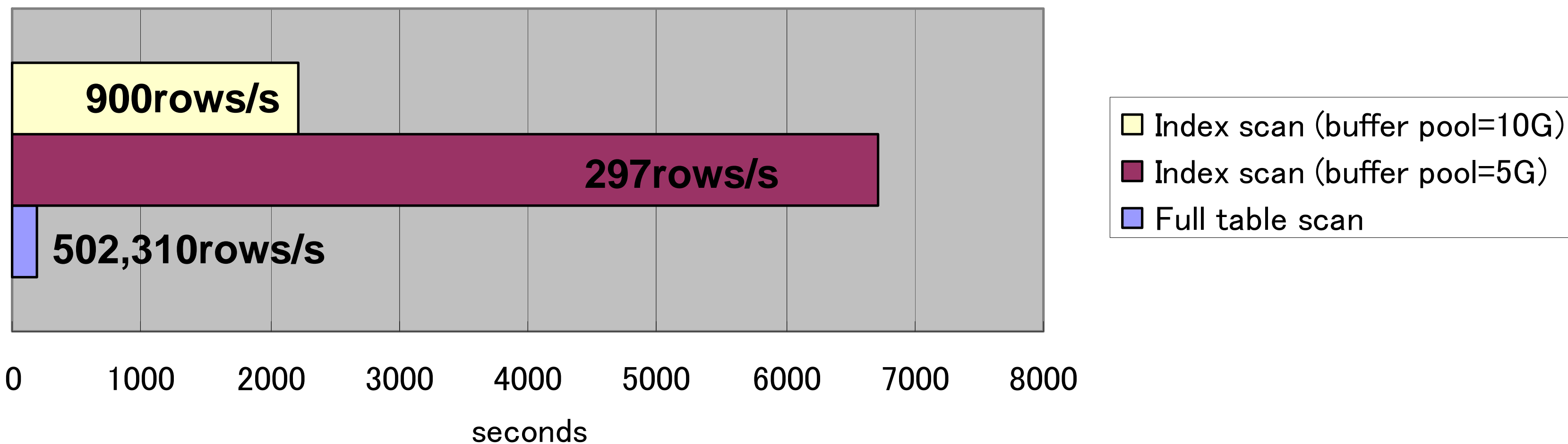
- 並列性が重要
  - SSDはフラッシュメモリを数多く搭載するという構造から、並列化が比較的容易
  - もちろん、並列性を活かせるようにI/Oコントローラが実装されていないとだめ
  - CristalDiskMarkなど、多くのベンチはシングルスレッドベースなので並列性の測定にはならない
- PCI-Express型SSDにも注目
  - 通常のストレージはSATAかSAS
  - PCI-Eに挿して使うタイプのSSDが出てきている (i.e. FusionIO)
  - インターフェイス速度が300MB/s -> 2GB/s
  - まだ高い
  - インターフェイス数が少ない

重いバッチ処理の性能

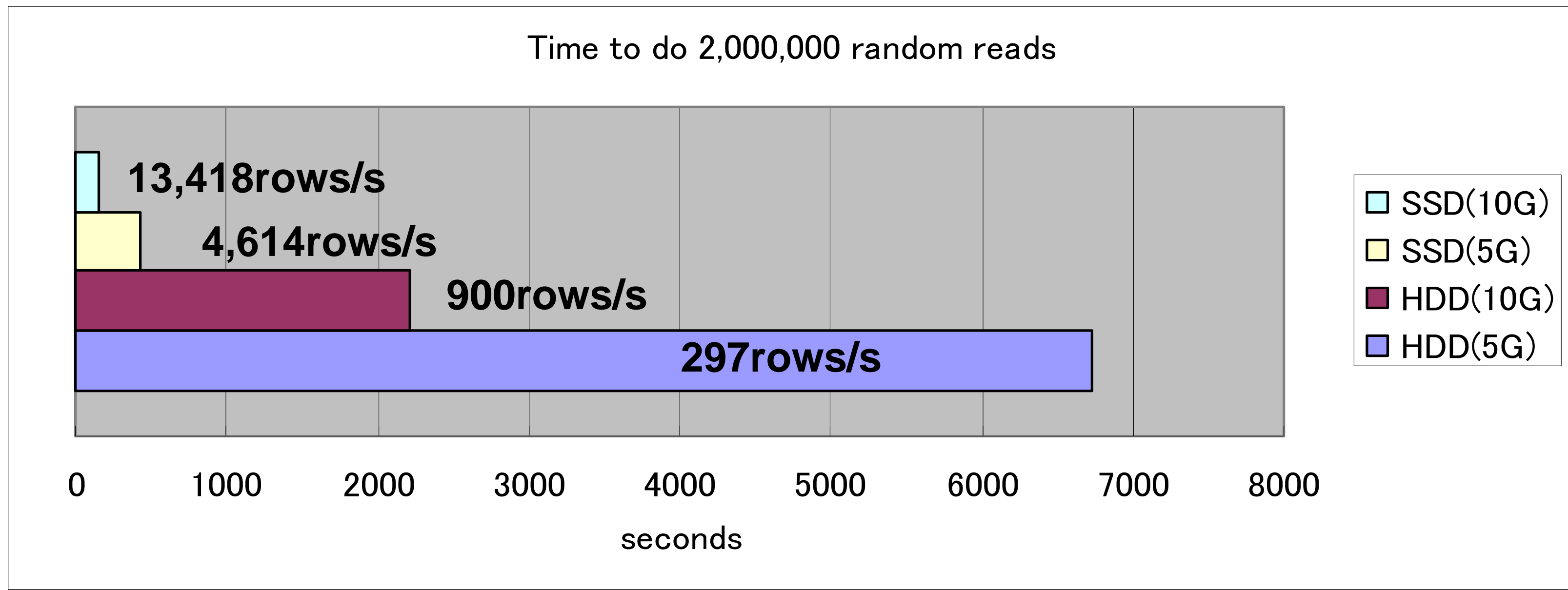
- 1億レコードのテーブルから200万レコードを取得
- Query: SELECT * FROM tbl WHERE seconday_key < 2000000
- 4GB index size, 13GB data (non-indexed) size
- Innodb_buffer_pool_size = 5GB, using O_DIRECT

Copyright 2009 Sun Microsystems inc

Benchmarks (1) : Full table scan vs Index scan (HDD)



Benchmarks (2) : SSD vs HDD, index scan



OS statistics

HDD, range scan

#iostat -xm 1

	rrqm/s	wrqm/s	r/s	w/s	rMB/s	wMB/s	avgrq-sz	avgqu-sz	await	svctm	%util
sdb	0.00	0.00	243.00	0.00	4.11	0.00	34.63	1.23	5.05	4.03	97.90

SSD, range scan

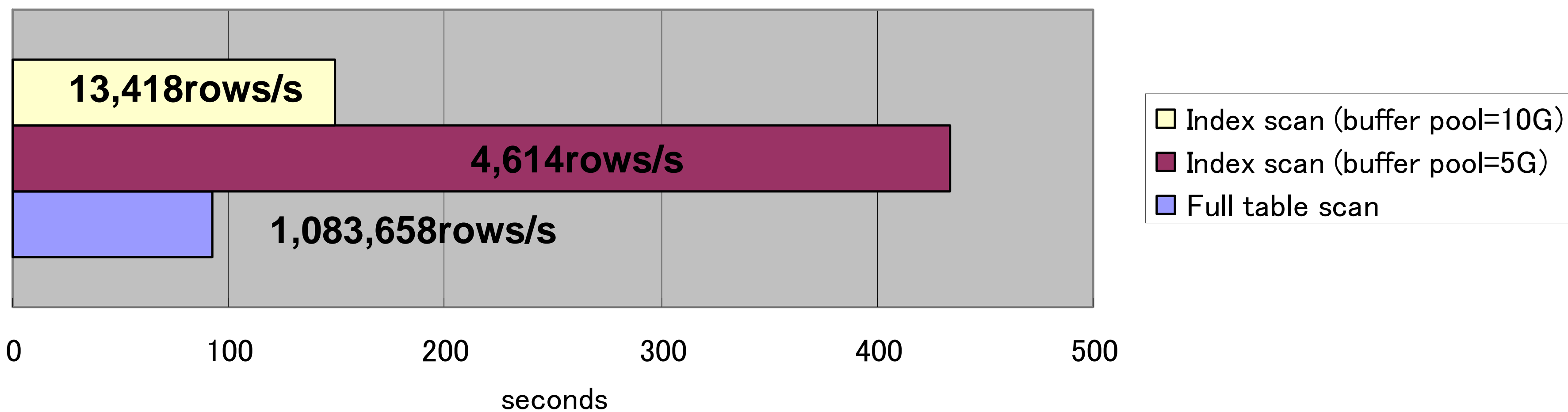
iostat -xm 1

	rrqm/s	wrqm/s	r/s	w/s	rMB/s	wMB/s	avgrq-sz	avgqu-sz	await	svctm	%util
sdc	24.00	0.00	2972.00	0.00	53.34	0.00	36.76	0.72	0.24	0.22	66.70

4.11MB / 243.00 ~= 53.34MB / 2972.00 ~= 16KB (InnoDB block size)

Copyright 2009 Sun Microsystems inc

Benchmarks (3) : Full table scan vs Index scan (SSD)



SSDに適したファイル配置を考える

- HDDはシーケンシャルリード/ライトが得意
- SSDはランダムリード/ライトが得意

- InnoDBの場合
  - ランダムI/O型
    - データファイル (ibd)
    - UNDOログ、Insertバッファ(ibdata)
  - シーケンシャルI/O型
    - バイナリログ
    - InnoDBログファイル
    - ダブルライトバッファ(ibdata)
    - その他ログファイル

Copyright 2009 Sun Microsystems inc

DBT-2ベンチマーク

	条件	スループット(NOTPM)
1	すべてHDD上に配置	3447.25
2	すべてSSD上に配置	14842.44
3	2)において、ライトキャッシュを無効にした場合	9877.06
4	REDOログをHDD上に配置	15539.8
5	REDOログとシステムテーブルスペースをHDD上に配置	23358.63
6	REDOログとシステムテーブルスペースをSSD上に配置	20450.78
7	REDOログとシステムテーブルスペースをtmpfs上に配置	24076.43
8	2)において、ダブルライトバッファを無効にした場合	22713.66
9	1)において、innodb_buffer_pool_size=10Gの場合	31927.45

まとめ

- ブロック構造、列/インデックスの構造を理解した上でDB設計をする
- Linux(OS)のチューニングポイントをおさえておく
- SSDは将来的に有望

Copyright 2009 Sun Microsystems inc

ありがとうございました