MySQLのパフォーマンスチューニングとよくある落とし穴

松信 嘉範 (MATSUNOBU Yoshinori)

Principal MySQL Consultant, Sun Microsystems

yoshinori.matsunobu@gmail.com

テーマ

- ハードウェア選定、バージョン選定
- ロードなどの更新処理のパフォーマンス改善
  - 今日のセッションのメイン
- レプリケーション
- 全文検索
- その他

ハードウェア選定・バージョン選定

- よくある勘違い
  - MySQLのバージョンは古くても良い
  - CPUコア数さえ多ければ、メモリサイズやディスクはどうでもいい
  - ディスクはSATA II 7200回転の1TBのディスクが1本あれば良い

MySQLのバージョン選定

- バージョンは5.0、5.1、5.4(beta)の中から選ぶ
  - たとえ4.0までの機能しか使わなくても5.0以降を選ぶ
  - 5.0と5.1は、8CPUコア程度までスケールする
  - 5.4は16CPUコア程度までスケールする
  - 4.1以下は、2CPUコア程度までしかスケールしない

- 4.0とかそれ以前のバージョンは化石
  - サポート対象外であり、バグフィックスも行なわれない
  - InnoDBのデータフォーマットの効率が悪く、より多くのデータサイズを消費する
  - 一貫性のあるバックアップを取る手段が限られている
  - データ消失を防ぐことのできる手段が限られている

ハードウェア選定

- 64ビット機が主流
- メモリサイズは極めて重要
  - 参照だけでなく、更新処理でも重要

- Linuxを選択する場合はI/Oスケジューラやファイルシステムに注意
  - noop, deadline, cfq, anticipatory
  - ext3, xfs (nobarrier)

ストレージ

- ハードウェアRAIDを使い、バッテリーバックアップつきライトキャッシュを搭載する
  - 書き込み性能が大幅に向上する
- IOPS (I/O per second)を意識する
  - SAS HDD 15,000回転で4～8本程度が主流
  - 1本のディスクではせいぜい数百iopsしか出ない
- RAID5はそれほど遅いわけではない
  - サイズに制約が無ければRAID1+0が推奨される
- SSDの効果は絶大
  - 2本のRAID 1 SSDで、HDD8本くらいの効果がある
  - 誰か人柱になってください

更新性能

よくある勘違い

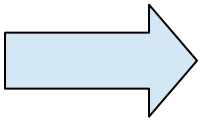
- INSERT件数(投入データサイズ)が10倍になれば、所要時間も10倍になると思っている
- INSERTは書き込みしかしないのでメモリサイズは小さくても良いと思っている
- インデックスが多くなると更新性能が落ちるが、たいしたことは無いので気にしなくて良いと思っている
- 履歴系・ログ系などの、蓄積主体のテーブルはMyISAMの方がInnoDBよりも良いと思っている

更新性能を高めるには、インデックスやストレージエンジンの特性をよく理解して、IOPSを減らす努力をすることが大切

INSERTすると何が起こるのか

INSERT INTO tbl (key1) VALUES (61)

Leaf Block 1	
key1	RowID
1	10000
2	5
3	15321
…	
60	431



Leaf Block 1	
key1	RowID
1	10000
2	5
3	15321
…	
60	431

Leaf Block 2	
key1	RowID
61	15322
Empty	

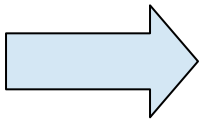
Leafがいっぱいになる

新しいブロックが割り当てられる

シーケンシャルなINSERT

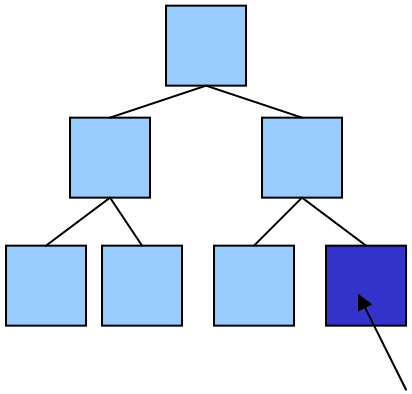
INSERT INTO tbl (key1) VALUES (current_date())

Leaf Block 1	
key1	RowID
2008-08-01	1
2008-08-02	2
2008-08-03	3
…	
2008-10-29	60



Leaf Block 1	
key1	RowID
2008-08-01	1
2008-08-02	2
2008-08-03	3
…	
2008-10-29	60

Leaf Block 2	
key1	RowID
2008-10-29	61
Empty	



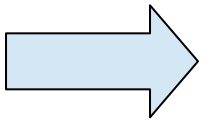
- auto_incrementやcurrent_datetimeなど
- インデックスに対してシーケンシャルに格納される
- フラグメンテーションが発生しない
- インデックスのサイズが小さくなる
- InnoDB PRIMARY KEYでは強く推奨される

すべてのアクセスがここに集中する: キャッシュされる

ランダムなINSERT

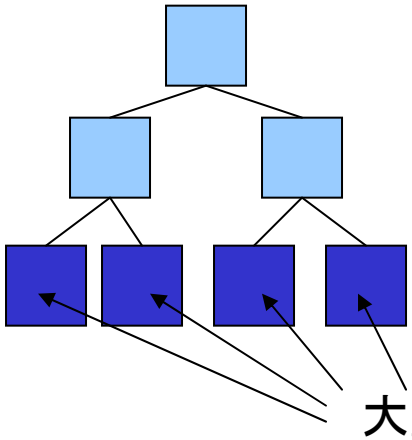
INSERT INTO message_table (user_id) VALUES (31)

Leaf Block 1	
user_id	RowID
1	10000
2	5
3	15321
…	
60	431



Leaf Block 1	
user_id	RowID
1	10000
…	
30	333
Empty	

Leaf Block 2	
user_id	RowID
31	345
…	
60	431
Empty	

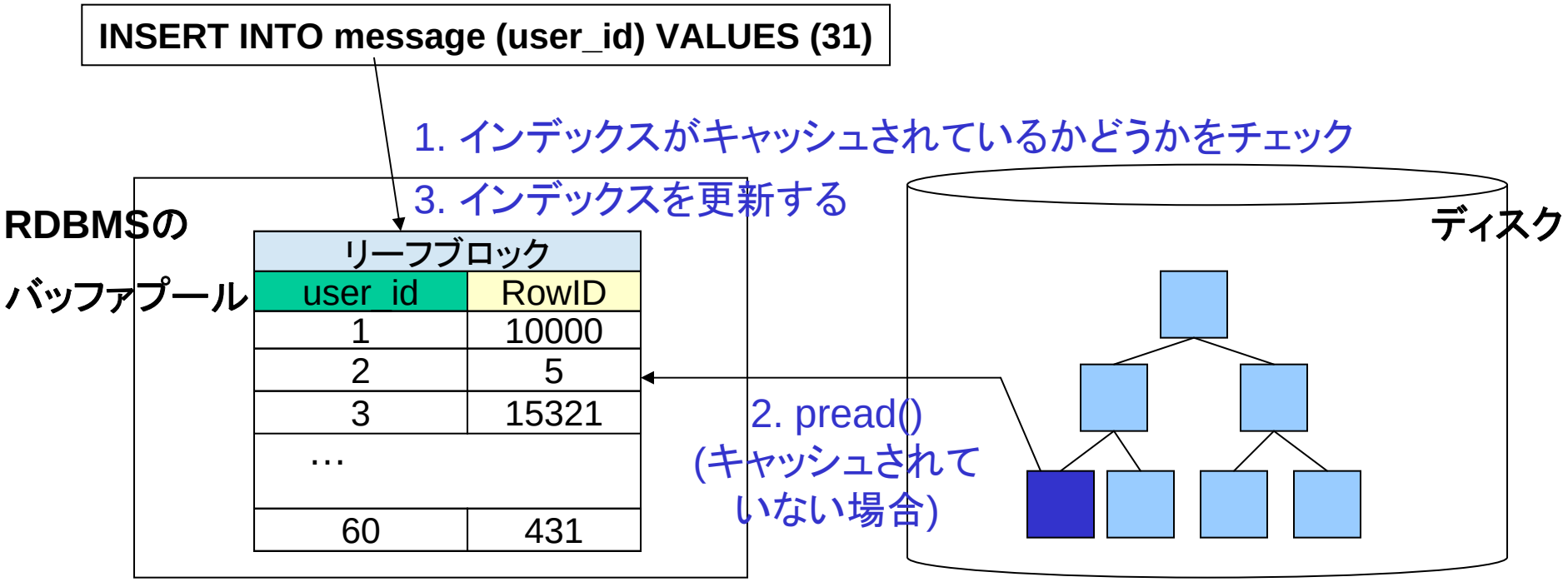


・通常、INSERTの順序はインデックスに対してランダム
(i.e. messageテーブルのuser_id)

・断片化しやすい
・各ブロックのエントリ数が少なくなる
・インデックスサイズが大きくなり、キャッシュされにくくなる

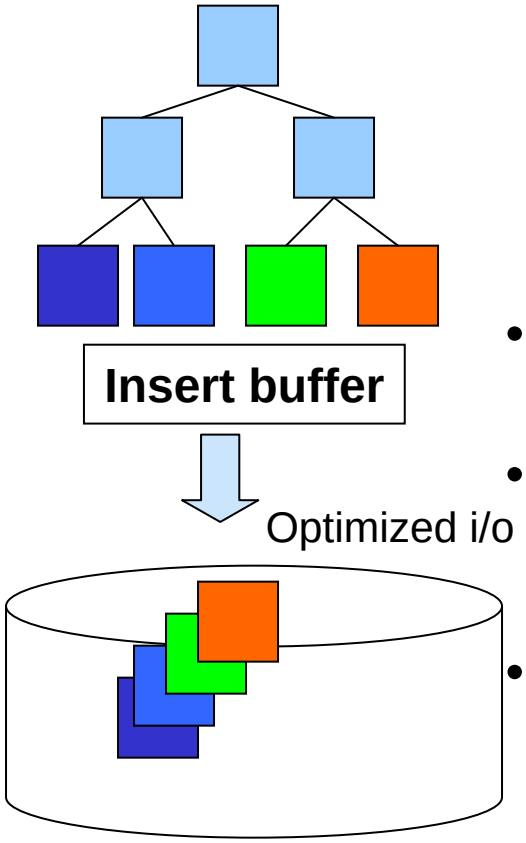
大量のリーフブロックが更新の対象になる

ランダムな INSERT はread()を必要とする



- バッファプールにキャッシュされていなければ、ディスクから読まなければいけない
- シーケンシャルなインデックス (AUTO_INC, datetime, etc) はこの影響を受けない
- メモリサイズを増やしたり、SSDを使うことで大きく改善される

InnoDB の独自機能: Insert Buffer

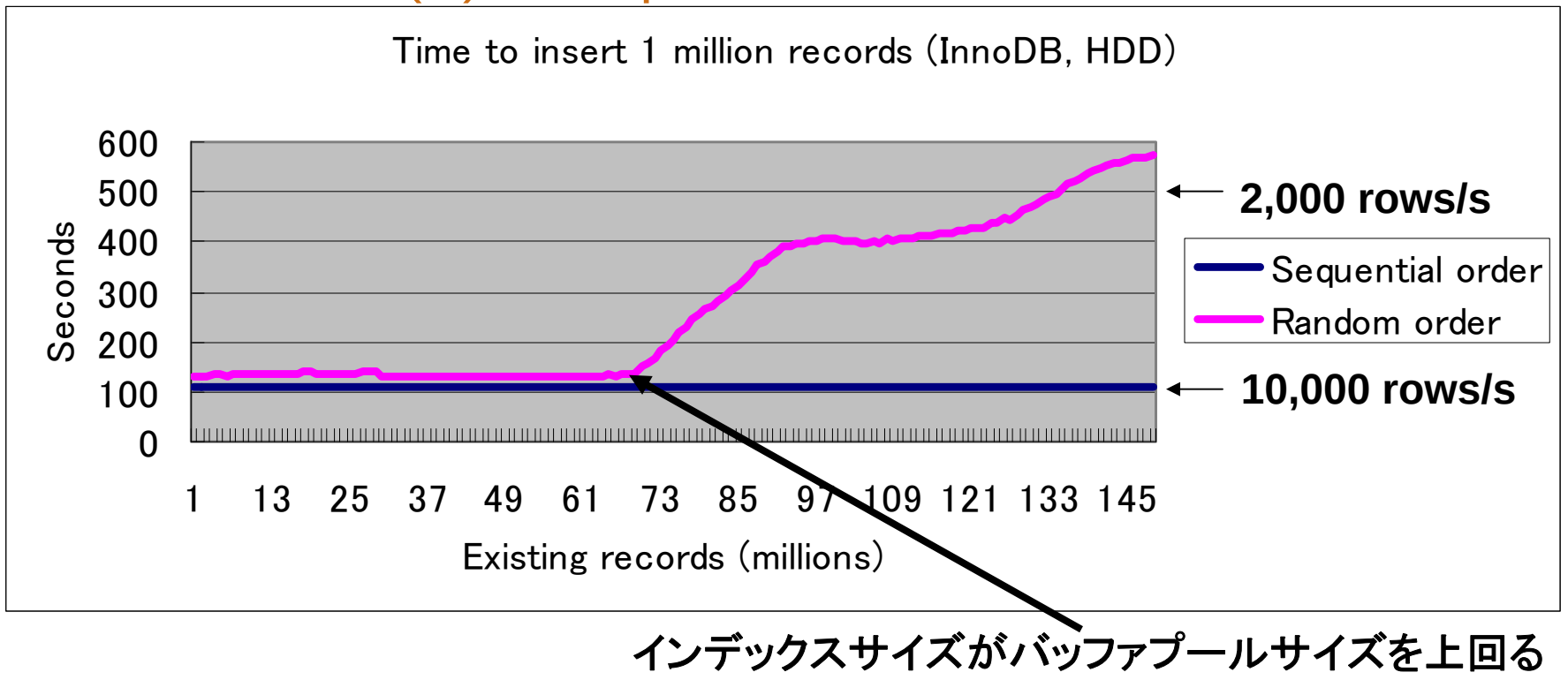


- 非一意インデックスにおいて、更新対象のブロックInnoDBバッファプール上に無い場合、InnoDBはディスクから対象ブロックを読むのではなく、「Insert Buffer」という専用領域に書き込む
  - Insert Bufferはメモリ上と、SYSTEMテーブルスペース上に作られる
- 段階的にインデックス本体に統合される(マージ)
- 長所 : I/Oオーバーヘッドの削減
  - ランダムI/Oの多くを(高速な)シーケンシャルI/Oにできる
- 短所 :
  - 検索処理は本体インデックスとinsert bufferの両方を読まなければいけないのでオーバーヘッドが増える
  - マージ処理に時間がかかる (再起動後も行なわれる場合がある)

インデックス戦略とINSERT性能の影響

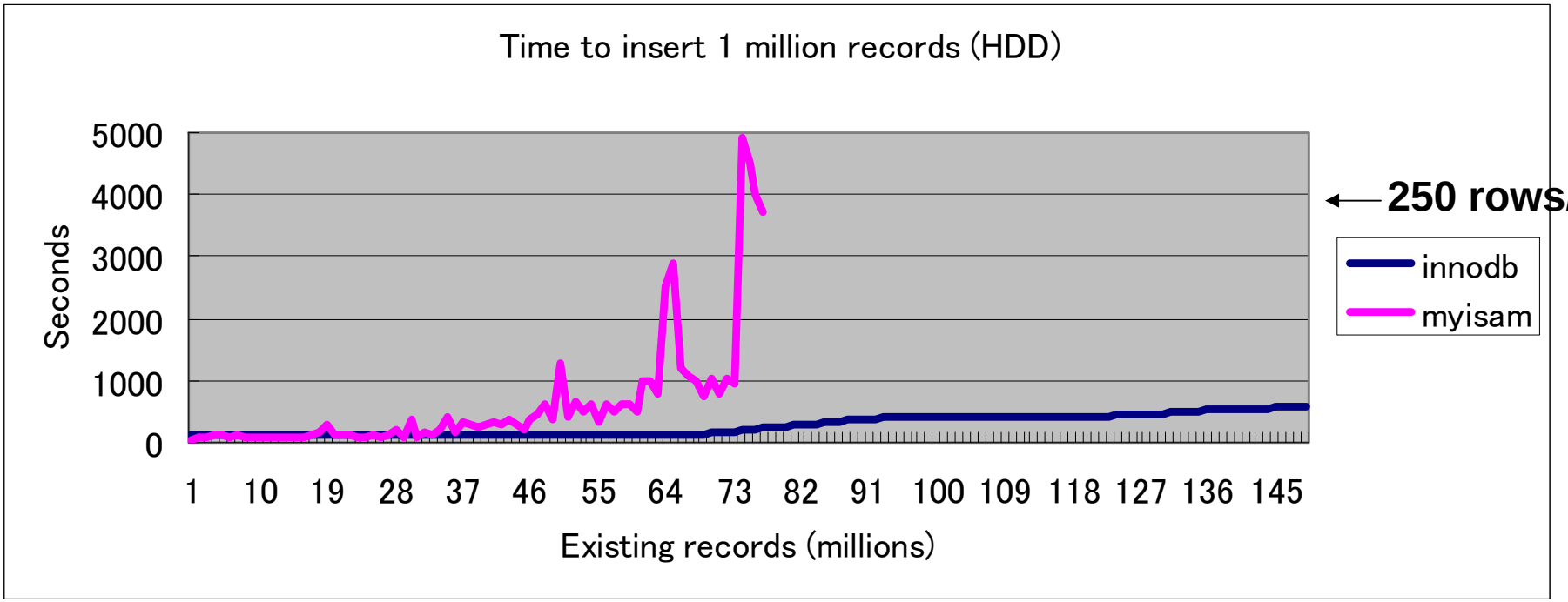
- 数億レコードをINSERTするベンチマーク
  - Twitterのメッセージなどを想定
- 100万レコードをINSERTする時間を測定
- 3つのインデックス
  - ランダムに入れる場合 vs シーケンシャルに入れる場合
    - Random: INSERT .. VALUES (id, rand(), rand(), rand());
    - Sequential: INSERT .. VALUES (id, id, id, id)
  - 主キーはAUTO INCREMENT
- InnoDB vs MyISAM
  - InnoDB: buffer pool=5G, O_DIRECT, trx_commit=1
  - MyISAM: key buffer=2G, filesystem cache=5G
- インデックス数 (3 vs 1)
- バッファプールサイズの変更
- MySQL 5.1のパーティショニング機能の活用

Benchmarks (1) : Sequential order vs random order



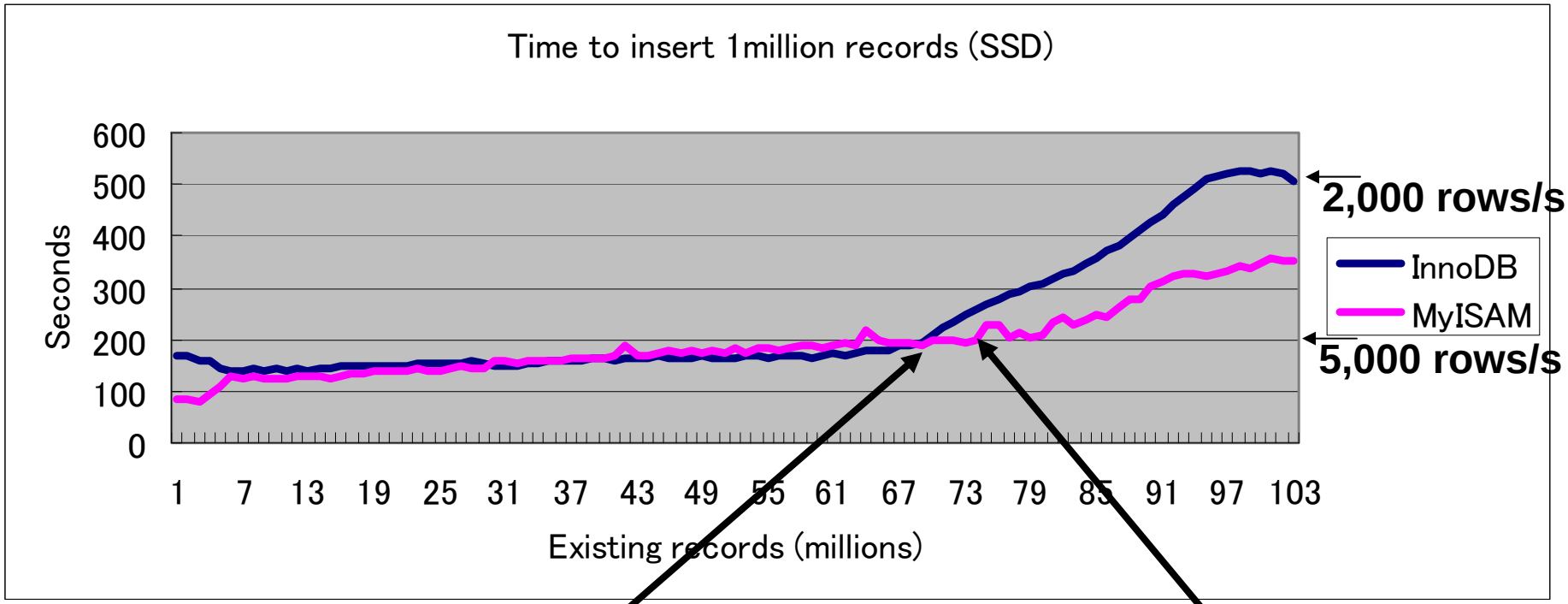
- バッファプールサイズにおさまっている状態では、INSERT時間が安定
- バッファプールを超えてからは徐々に時間がかかるようになる (read()時のヒット率が下がるため)
- シーケンシャルINSERTでは常にバッファプール内におさまるので安定
  - データ量が増えても更新性能が落ちないのは、すべてのインデックスがシーケンシャル順にINSERTされる場合だけ

Benchmarks (2) : InnoDB vs MyISAM (HDD)



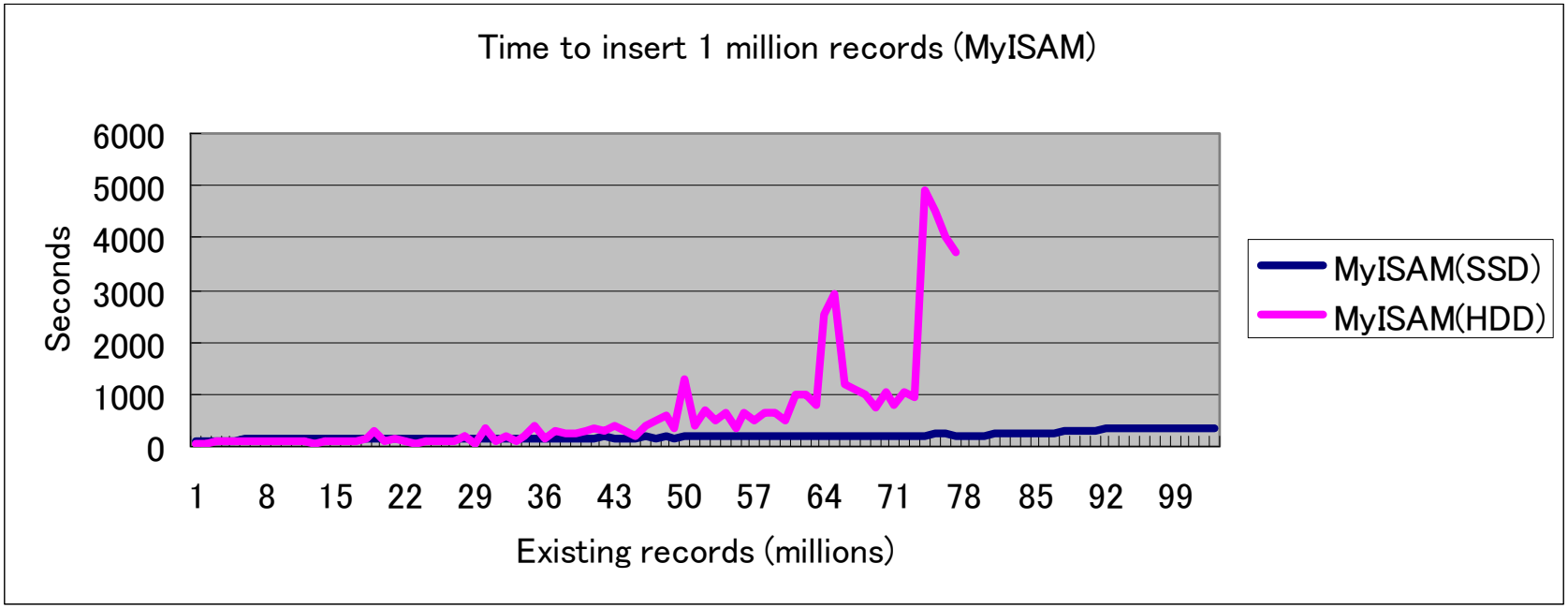
- MyISAMはInnoDBのinsert bufferに相当する最適化機構が何も無く、OSやストレージに強く依存する
- ディスクのシークや回転待ちはHDDでは深刻になる

Benchmarks(3) : MyISAM vs InnoDB (SSD)



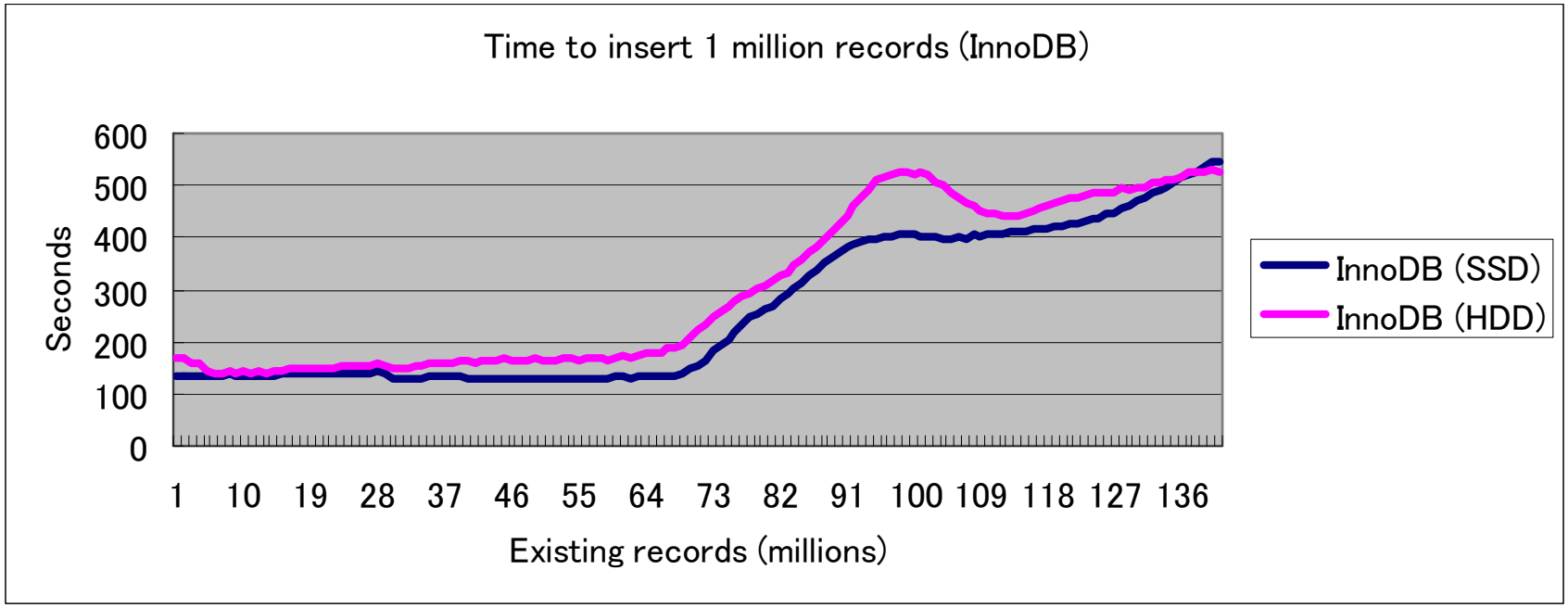
- MyISAMは、単にHDDをSSDに置き換えるだけでInnoDBよりも速くなったという例

Benchmarks (4) : SSD vs HDD (MyISAM)



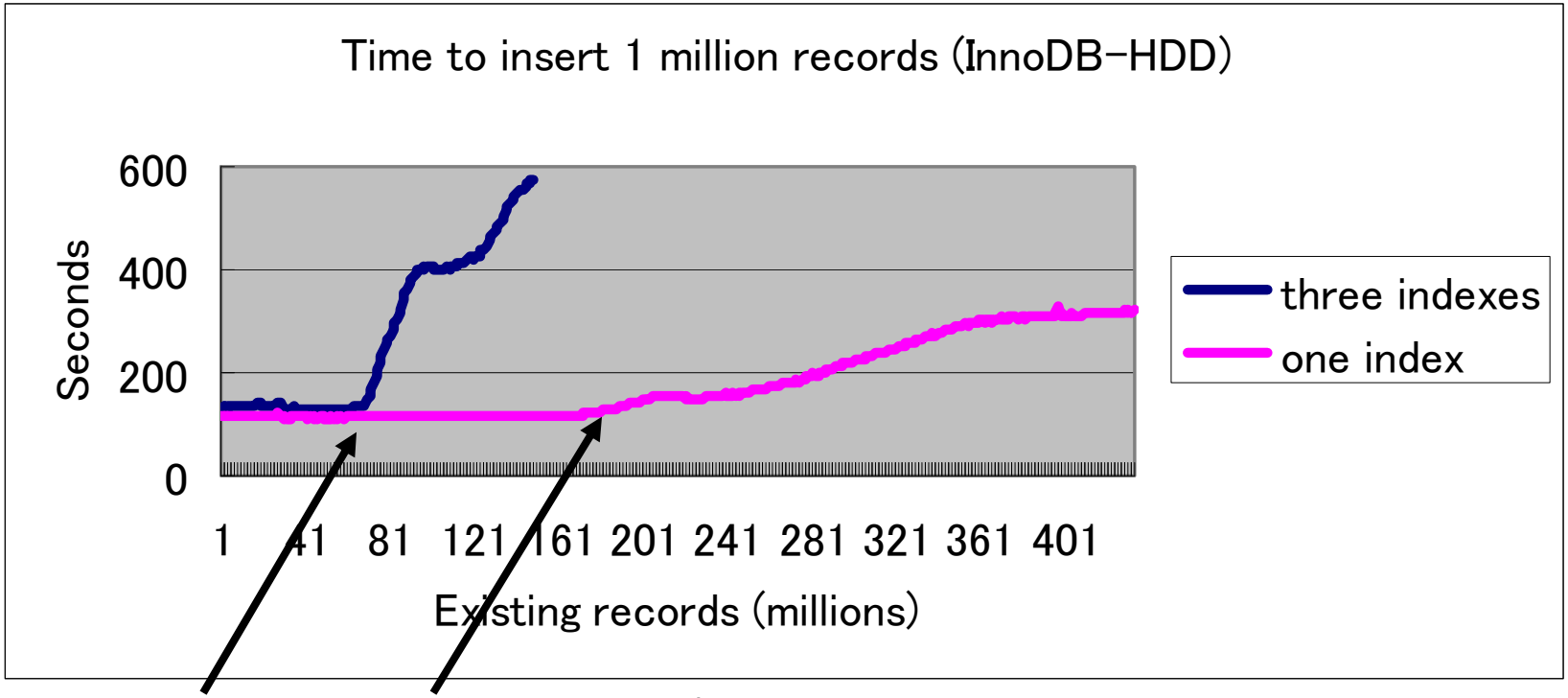
- MyISAMのINSERT性能は、SSDとHDDで非常に大きい
- ディスクシークや回転待ちはSSDでは発生しない

Benchmarks (5) : SSD vs HDD (InnoDB)



- MyISAMに比べると違いは大きくない
- InnoDBのInsert bufferによる影響が大きい
- Insert bufferのマージ処理に要する時間はSSDとHDDで相当に差があった (SSD:15min / HDD:42min)

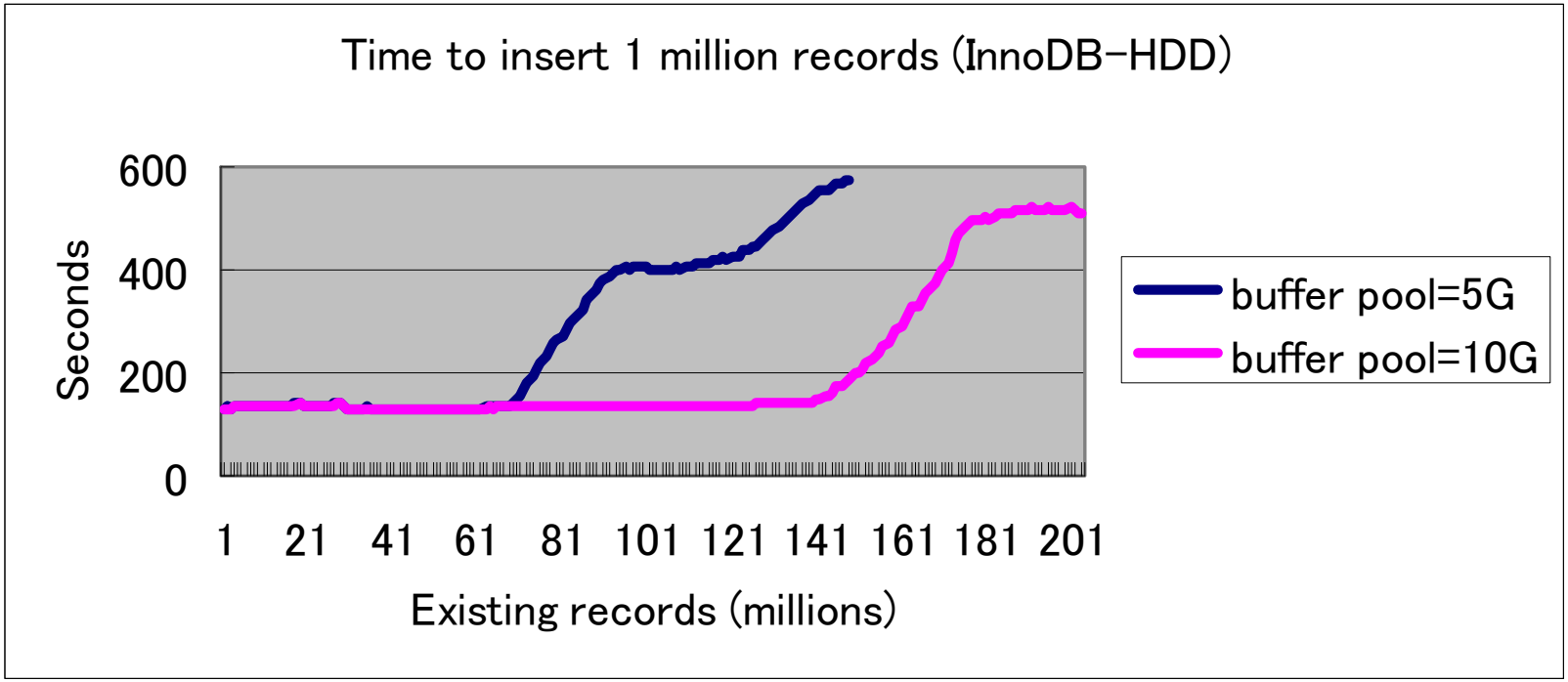
Benchmarks (6) : Three indexes vs Single index



インデックスサイズがバッファプールを超える

- インデックスが1個の場合は、インデックスサイズが小さくなるので
  パフォーマンスが遅くなる点も先延ばしにできる
- 1レコードのINSERTあたりに必要なランダムリード回数も小さくなる
- ベストプラクティス：インデックスの数は可能な限り小さくする

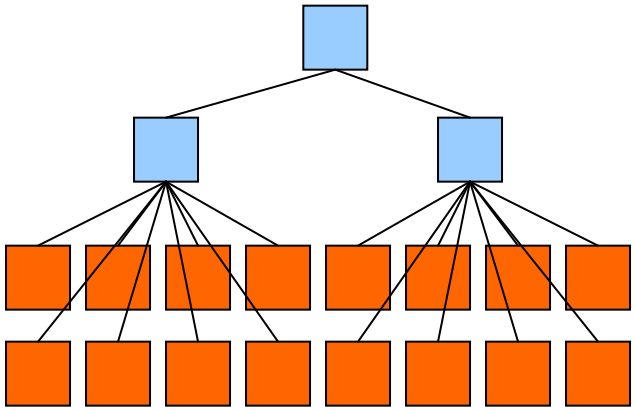
Benchmarks (7) : Increasing RAM (InnoDB)

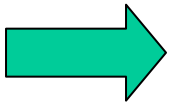


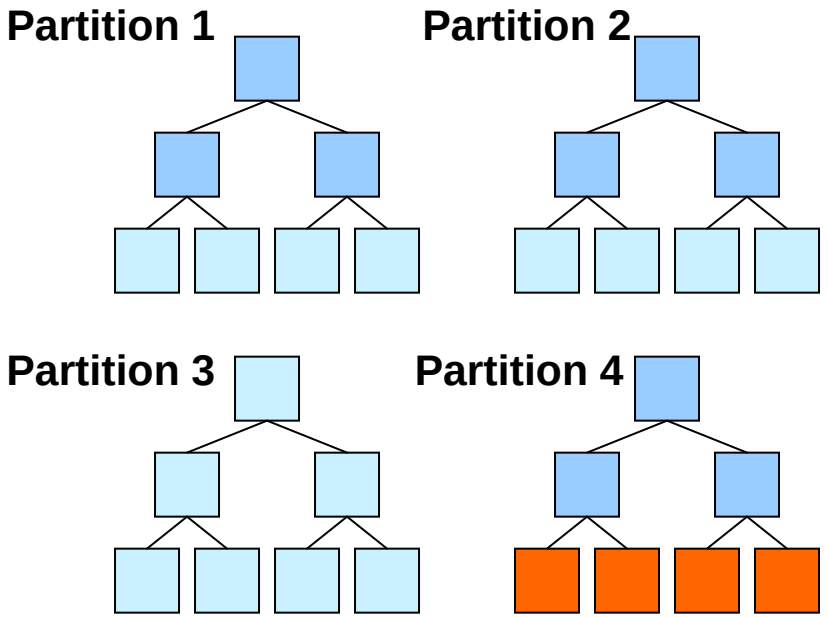
- メモリサイズの増大(より多くの領域をバッファプールに割り当てる)は、損益分岐点を高める効果がある
- ベストプラクティス：インデックスサイズは小さくする

インデックスサイズを小さくする

1個の巨大なテーブル(インデックス)

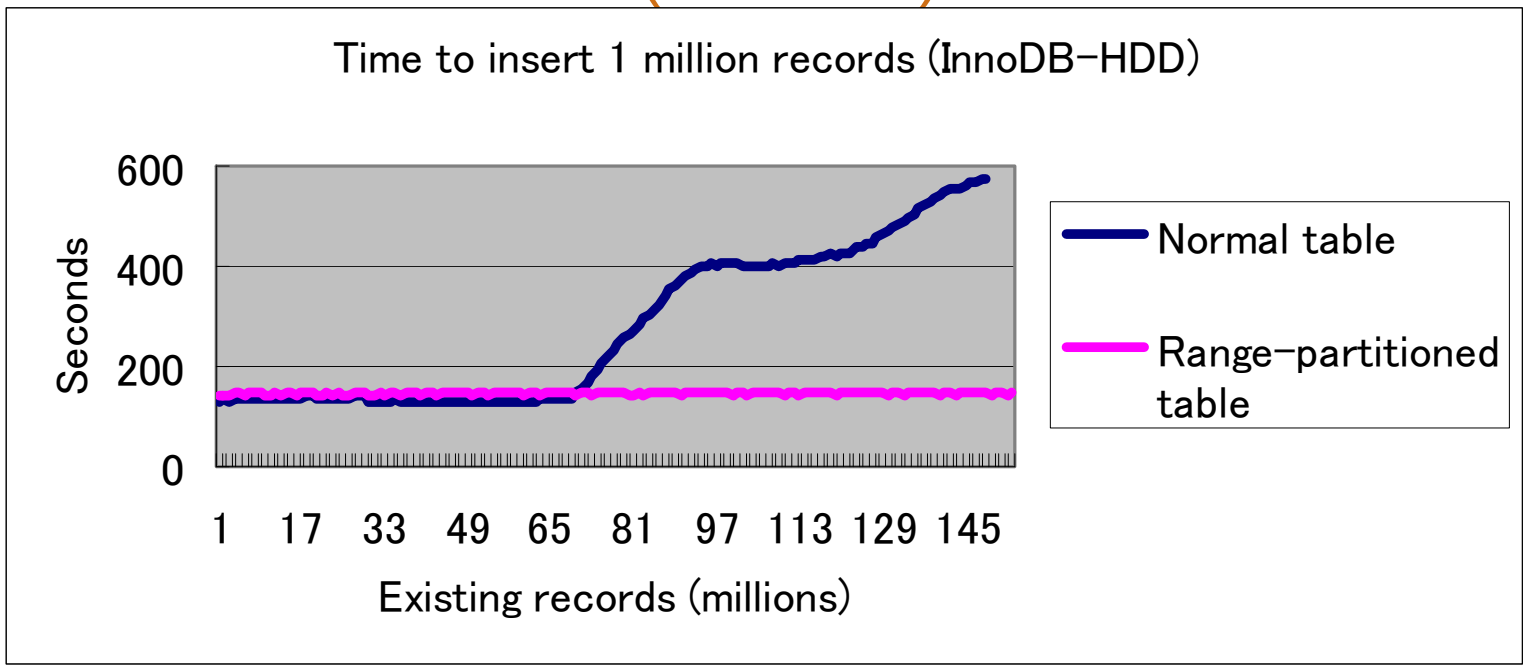






- すべての(アクセス対象の)インデックスがメモリにおさまる場合、INSERTは高速
- インデックスサイズを小さくするためのプラクティスに従う(データ型の最適化等)
- アプリケーションパーティショニング (Sharding)
- MySQL 5.1 のレンジパーティショニング
  - パーティショニングのキーを、シーケンシャルに投入される列にする(auto_incrementや登録時刻など)
  - インデックスもパーティショニングの対象になる (Local Index)
  - INSERT主体のテーブルでは、最新のパーティションだけがアクセス対象になる
  - SELECTが頻発する場合はパーティションプルーニングを考慮

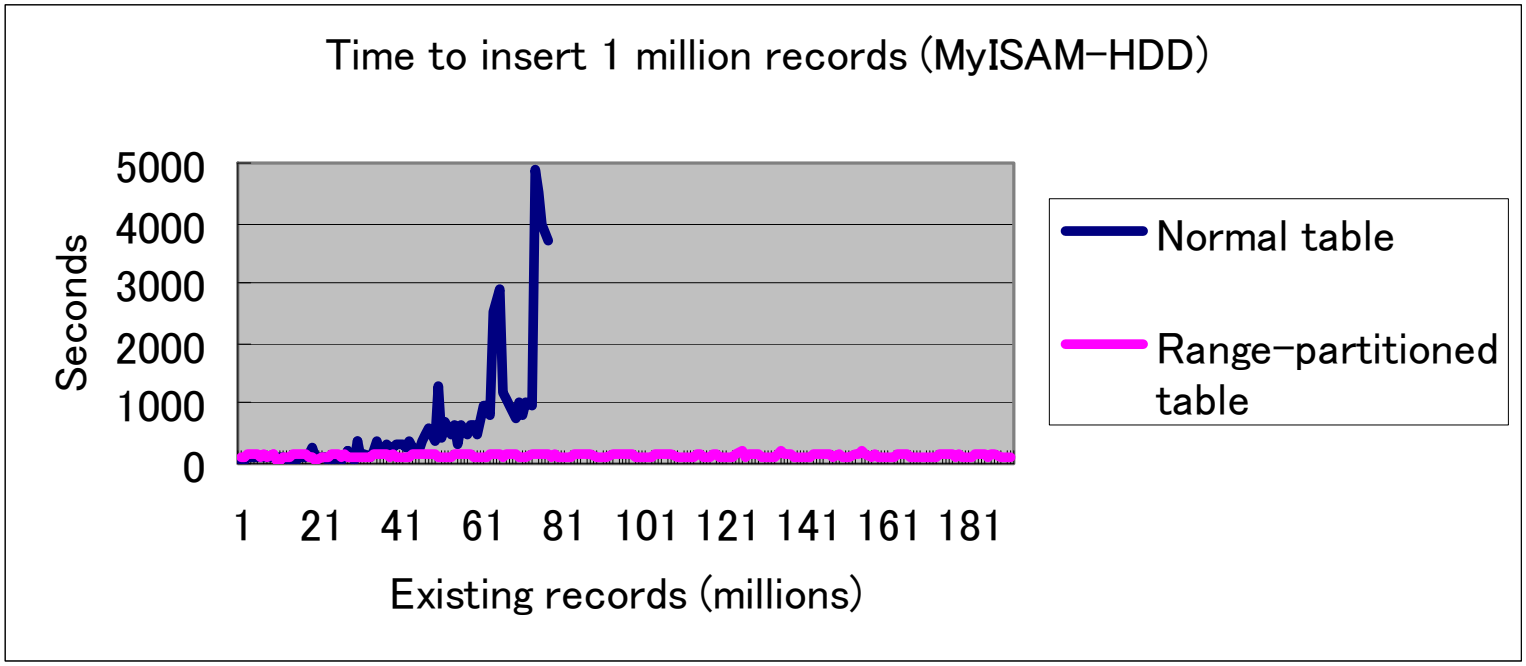
Benchmarks(8) : Using 5.1 Range Partitioning (InnoDB)




```
PARTITION BY RANGE(id) (
    PARTITION p1 VALUES LESS THAN (10000000),
    PARTITION p2 VALUES LESS THAN (20000000),
    ....
```

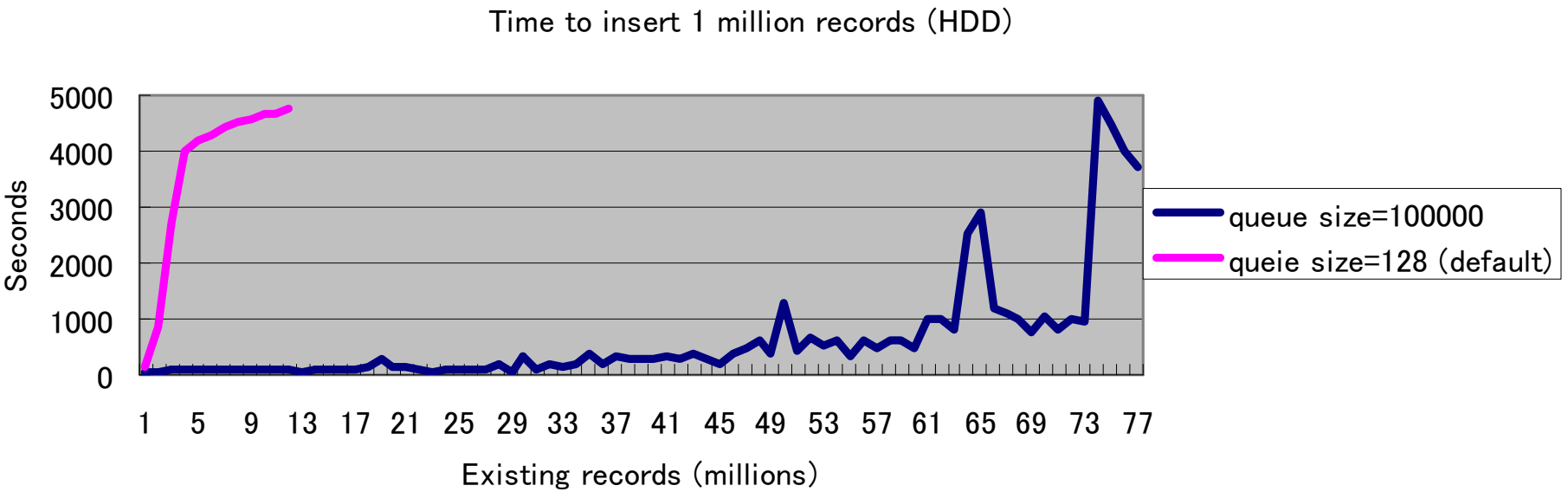
- INSERT主体のテーブルでは、直近のパーティションしか更新されないのでread()範囲が非常に限定される

Benchmarks(9) : Using 5.1 Range Partitioning (MyISAM)



- MyISAMでも同様

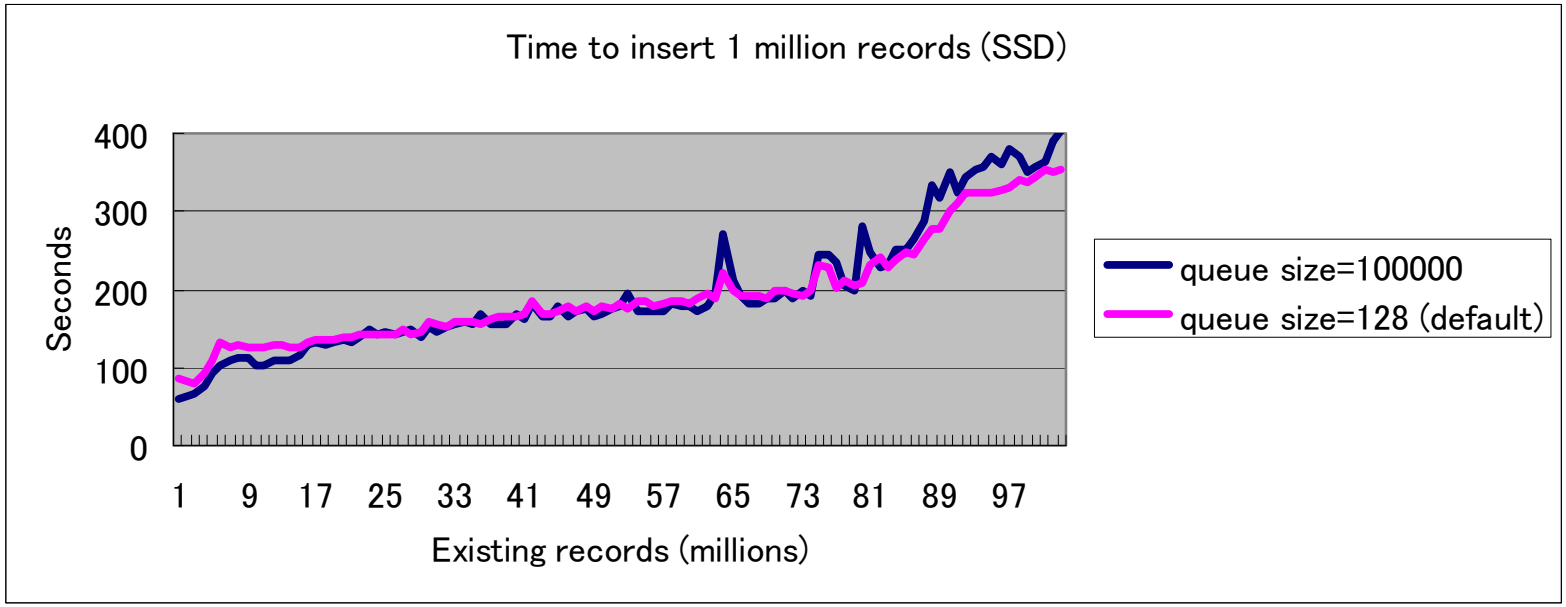
Benchmarks (10) : Linux I/O Scheduler (MyISAM-HDD)



- MyISAMは、InnoDBのようなI/O処理最適化のメカニズムが無い
  - OSとストレージに大きく依存する
- LinuxのI/Oスケジューラには、「I/Oキュー」というものがある
- キュー内のI/Oリクエストをソートし、最適になるように並べ替える
- キューサイズは変更可能

```
# echo 100000 > /sys/block/sdX/queue/nr_requests
```

Benchmarks (11) : Linux I/O Scheduler (MyISAM-SSD)

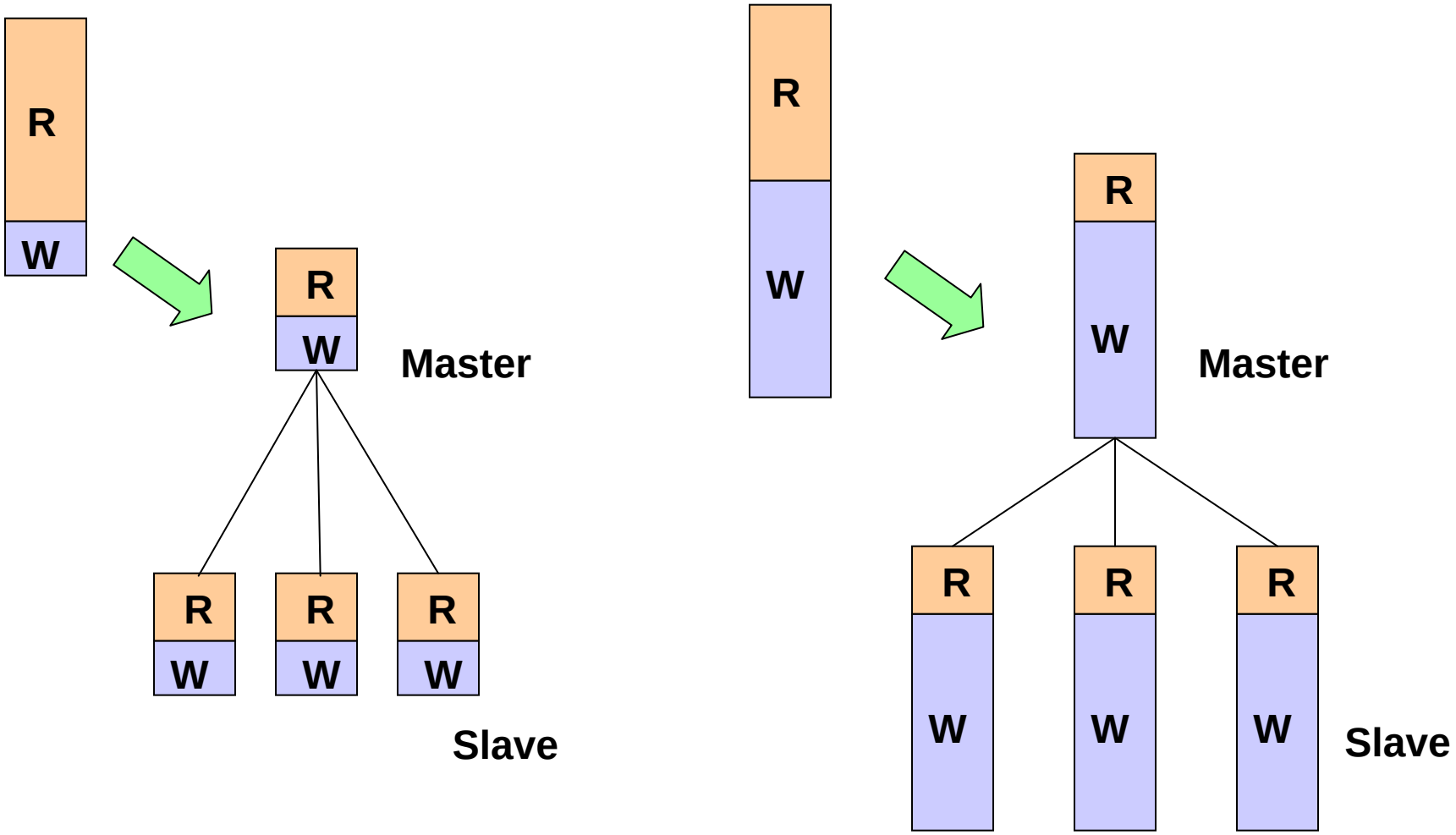


- SSDでは大きな違いが無い

レプリケーション

- スレーブの遅延をどのように防ぐか

レプリケーション



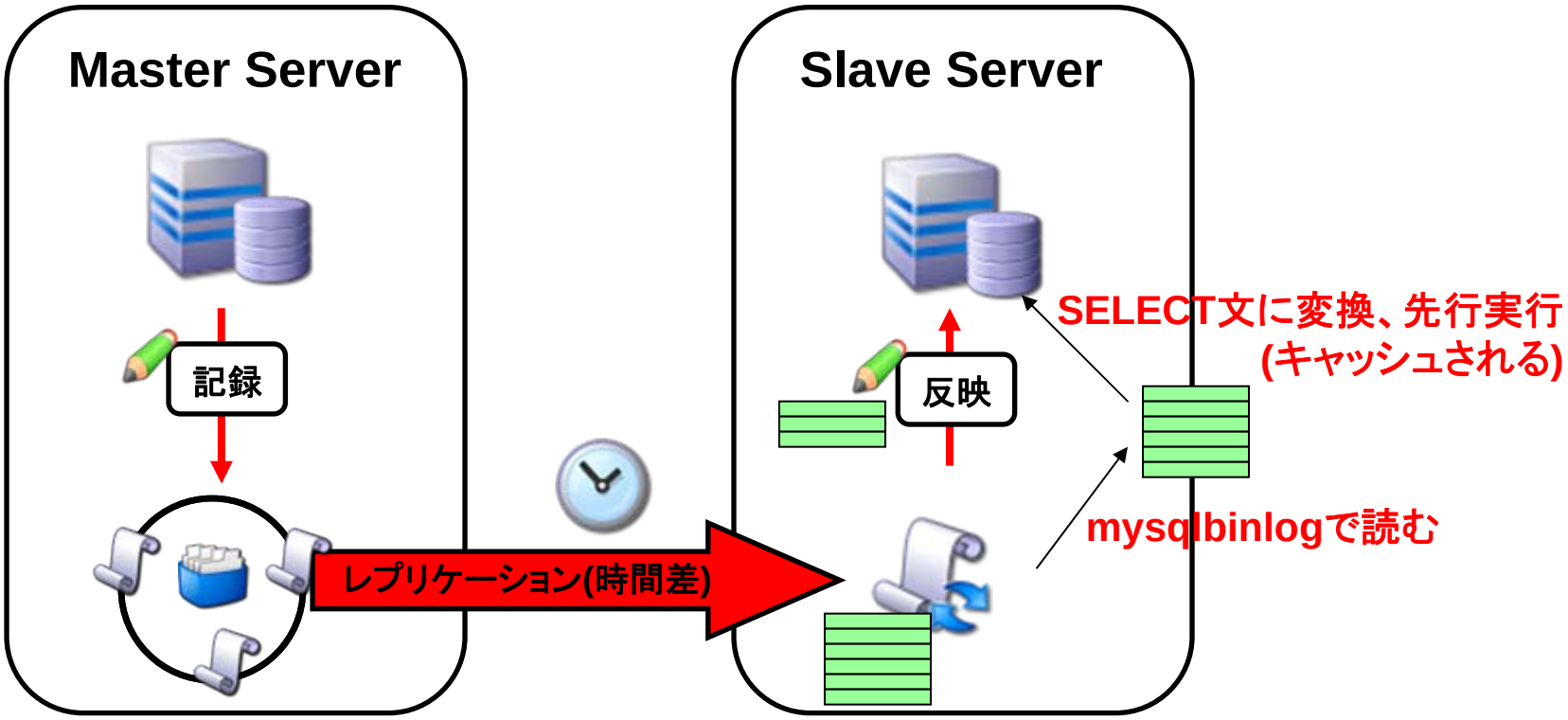
- 参照処理のスケールアウトに有効
- 更新処理のスケールアウトにはならない

スレーブの遅延にどう対処するか (1)

- レプリケーションはシングルスレッドで動く
  - 1本のI/Oスレッドと1本のSQLスレッド
  - マスターの負荷が高い場合やネットワーク回線が遅い場合、I/Oスレッドがボトルネックになることがある
  - どちらかというとSQLスレッドがボトルネックになる方が大半

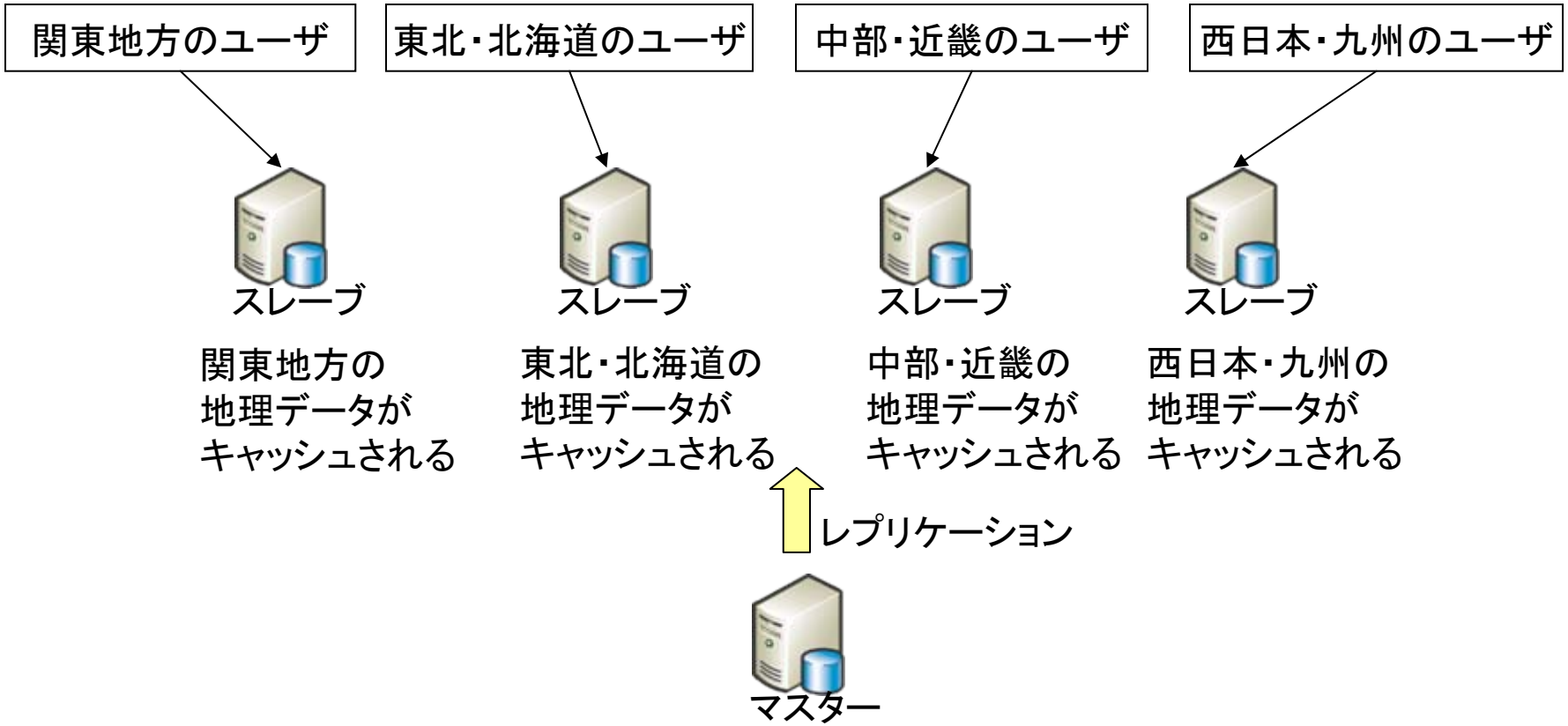
- レプリケーションはトランザクション単位
  - 1個のトランザクションが非常に長ければ、スレーブに伝達されるのが遅くなり、その分が遅延になる
  - LOAD DATAなどで大量にレコードを処理するような場合、コミット単位を小さく切るのが効果的
    - これはクラッシュ時のリカバリを短時間で終わらせる場合にも有効

スレーブの遅延にどう対処するか (2)



- I/Oスレッド、SQLスレッドともにシングルスレッドなので、複数のCPUコアを活かしきれない
- I/Oスレッドは高速に転送するが、SQLスレッドによる実行に時間がかかることが多い
- mysqlbinlogでリレーログファイルを読み、それをSELECT文に変換して実行
- 結果がキャッシュに乗るので、INSERT/UPDATE/DELETEが高速になる

キャッシュ効率を意識した振り分け



- 「関東のユーザは関東の地理データにアクセスすることが多い」など、アクセスパターンに偏りがあれば、単純にロードバランサーやラウンドロビンで振り分けるよりもキャッシュ効率がずっと良い
- 全スレーブが全データを持つようにすれば、関東と中部地方にまたがった検索などにも対処できる

スレーブのハードウェア・ストレージエンジン選定

- ハードウェア選定
  - CPUのコア数よりもクロック数を優先する
  - メモリサイズは相変わらず重要
  - ディスクは本数よりも、1本あたりのIOPSを重視
    HDDなら15,000回転が望ましい
    SSDは非常に有力な候補になる
  - マスターとスレーブ間のネットワーク回線はGbEを推奨

- ストレージエンジン
  - マスターInnoDB、スレーブMyISAMという形はよく見かけるが、
    ・MyISAMはそれほど高速ではないということと、
    ・MyISAMはテーブルロックかつ参照と更新が競合するので、
    集計クエリなどが長時間走ると、レプリケーション遅延が
    起こることに注意
  - マスターをBlackholeにするのは有効
    - マスターにはデータが残らず、制約違反等の検知もできない

全文検索

- バージョンごとに選択肢がある
- MySQLデフォルトの全文検索機能は、日本語に対応していない

- MySQL 5.0
  - **Tritonn/Senna**
    - 住商情報システムと未来検索ブラジルによるサポート提供
    - 最も実績がある

- MySQL 5.1以降
  - **Sphinx**
    - http://www.sphinxsearch.com/
    - MySQLのストレージエンジンとして動作し、UTF-8であればBi-gram方式により、日本語の全文検索が可能
    - 分散検索エンジン。非常に高速
    - Craigslistなど、海外の大規模サイトでの実績が多数
    - 利用方法がやや特殊
  - Fulltext Parser Plugin
    - http://mysqlftppc.wiki.sourceforge.net/Home-j
  - mroonga/groonga
    - Sennaの後継にあたる。現在開発中

その他の質問

Q：最近、PostgreSQLに負けつつあります

- MySQL技術者はPostgreSQLに勝っていると思っているし、PostgreSQL技術者はMySQLにリードを許したことは無いと思っているでしょう。
- すべては社内における権力争いに帰着されるのが実情ではないでしょうか。
- アプリケーション実装の選択肢は1つではないので、製品の性質をよく理解して、製品の機能を引き出せば、目的を達成できるかと思います。
- 重要性の高い分野についてはMySQLでも積極的な改善が進められています。

本体にはまだ取り込まれていないが、強力な機能 (パッチは完成済み)

- バイナリログのスケーラビリティ改善
  - バイナリログを有効にしているとき、同時に更新するスレッド数が増えても更新性能が伸びない(グループコミットがきかない)点の修正
  - 数十倍のレベルで高速化
- sync_binlog=1の高速化
  - 耐障害性の最も高いsync-binlog=1の性能改善
  - これも数倍のレベルで高速化
- 監査ログ機能
  - いつ、どこから誰がログイン/ログアウトし、どのクエリを実行したかを特定する
  - プラグイン形式になっており、任意にカスタマイズできる
  - General Logに比べて非常に高速
- UTF-8の4バイト文字のサポート

Custom Build / Early Adopter Program

- Custom Build
  - 「パッチは存在しているがまだ本家にはマージされていない」機能をバックポートし、プレミアム価格でサポートするメニュー
- Early Adopter Program
  - まだ社内のQA基準をクリアしていないが、非常に有望な機能を持ったバージョンについて、リリース前にそれを利用する機会を提供。希望した有償顧客に対して応相談

宣伝 (1)

- 新書籍「高性能・無停止 Linux-DB構築 (仮)」
  - 9月下旬発売予定
  - LVM, Heartbeat, DRBD, monによる高可用構成
  - MySQL Cluster、レプリケーション等による超高可用構成
  - RAIDやライトキャッシュなど、DBサーバの性能を引き出すハードウェア戦略
  - ファイルシステムやI/Oスケジューラの影響
  - パフォーマンスを引き出すインデックス戦略
  - DBサーバの負荷テストの要点やケーススタディ
  - SSDによるDBサーバへの性能の変化、またDBアーキテクチャはどのように変わるかの考察

宣伝 (2)

- DBマガジン新連載「Real World MySQL (仮)」
  - 10月発売号あたりから連載開始予定
  - 自分にとって、「現場で使えるMySQL (2006.3)」以来のMySQL長期連載記事
  - 現実世界のMySQLアプリケーションを例に取り、「このようなアーキテクチャ設計・テーブル設計が考えられる」という紹介。いくつかの選択肢を挙げ、トレードオフを考察
  - 典型的なWebアプリケーションでのMySQLデザイン
    - オーソドックスに参照処理が多いアプリケーションへの対処
  - オンライン更新負荷の高いアプリケーション
    - Twitterメッセージ、広告処理、ログ管理など
  - オンライン広範囲検索系のアプリケーション
    - 中野駅から徒歩5分以内にある家賃10万円以下の物件を探す
  - バッチ処理・集計処理でのMySQLデザイン

ありがとうございました