InnoDBのパフォーマンスチューニング (前編)

MySQL株式会社
シニアコンサルタント
ymatsunobu@mysql.com

今回のカバー内容

- パラメータ設定の指針
- テーブル設計の指針
- MySQL 5.1以降での強化内容
- ハードウェアの選定指針
- 様々なTips

※順不同です

Copyright 2007 MySQL AB

InnoDBをなぜ使うか

- **トランザクション**
  1. **コミット、ロールバック、セーブポイント構文**
  2. **行レベルロック**
  3. **オンラインバックアップ**
  4. **クラッシュリカバリ**
- 参照整合性制約(外部キー)

  – 「自分のアプリケーションではトランザクションは使わない」＝「InnoDBではなくMyISAMやMEMORYで十分?」
  – 「トランザクションが不要」という主張は、1だけを指していることが多い
  – 2-4はほとんどのアプリケーションで非常に重要な機能

Copyright 2007 MySQL AB

クラッシュリカバリ

- MyISAMはデータ量の増大とともに時間がかかる
  - REPAIR TABLE, myisamchk –rq
  - データロストの危険もある
- InnoDBはデータ量の増大との相関が無い
  - クラッシュ直前(最終チェックポイント以降)に行なわれていた更新の量に比例
  - InnoDBログファイルのサイズ
  - InnoDBバッファプールのサイズ
- 可用性(ダウンタイム)に直接的に影響する

Copyright 2007 MySQL AB

InnoDBチューニングの王道的アプローチ

- クエリを改善して全体的に処理効率を上げる
- データサイズをできるだけ小さくしてディスクI/O量を減らす(アプリケーション設計に大きく依存)
- メモリをできるだけ多く積んでディスクI/O頻度を減らす
- ライトキャッシュ(+バッテリーバックアップ)を搭載して同期書き込み性能を上げる

Copyright 2007 MySQL AB

コミット性能 (同期書き込み)

- innodb_flush_log_at_trx_commit=1, 0, 2
  - 1:コミットのたびにディスクに同期書き込み (デフォルト)
  - 0:1秒ごとにディスクに同期書き込み
  - 2:1秒ごとにディスクに同期書き込み。またコミットのたびにOSのファイルキャッシュに書き込み
  - マスターで1、スレーブで0～2にするのが一般的

同期書き込みのパフォーマンス

- fsync() (or O_SYNC + write, fdatasync()) の速度に依存
- ディスクに対する同期書き込み1回の所要時間
  - シーク待ち時間
  - 回転待ち時間 (平均で半回転)
  - 転送時間

シーク待ち時間・・・ 4ms程度
回転待ち時間・・・・ 4ms程度(回転数に大きく依存)
転送時間・・・・・・・・ 1ms程度(データ量に大きく依存)

せいぜい1秒あたり100～200回程度しか同期書き込みはできない。

Copyright 2007 MySQL AB

同期書き込みのパフォーマンス向上

- ライトキャッシュの使用
  - ディスクやRAIDコントローラ等が持つ機能で、ディスク本体の代わりにキャッシュ領域に書いて終わり(後で反映)となる
  - 毎秒数千回クラスのfsync()が可能。レスポンスタイムが大幅に改善
  - 後でまとめて(シークと回転を最適化して非同期に)ディスクに書くのでスループットも向上する。これは非常に大きなメリット
  - バッテリーバックアップなどの形で、クラッシュしてもデータが失われないようにすることは必須

- トランザクション間隔を適切に取る
  - 「1件ごとにコミット」は避ける(特にバッチ処理)

- グループコミット機能
  - 複数のセッションからのコミット命令をまとめて1回の同期書き込みに集約
  - MySQL5.0以降でのInnoDBは、現状ではバイナリログと併用するとグループコミット機能が効かないので注意 (innodb_support_xa=OFFでも効かない)

Copyright 2007 MySQL AB

コミット時の動作の違い

InnoDBログバッファ

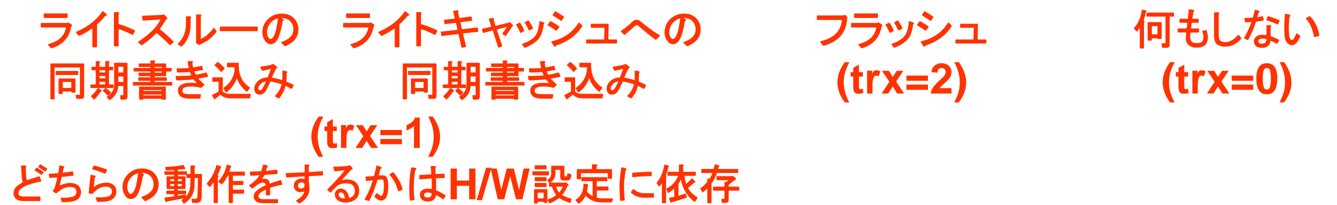
flush/fsync()

OSのファイルキャッシュ

デバイスドライバ

ライトキャッシュ

Disk



テーブル設計

テーブル設計やSQL文の記述は極めて重要

- SQL文の実行計画には注意する
  - インデックスの効果的な利用
  - 複雑なSQL文をシンプルにする
  - EXPLAINの結果に気を配る
    - Using temporary
    - Using filesort
    - Dependent Subquery
- サマリーテーブルの導入などは効果的
  - COUNT, SUM, AVGよりもWHERE idx_col=xx

よくある問題

- 主キーは重複しないようにUUID(VARCHAR(36))にした
- とりあえず多くの列にインデックスをつけた
- 全部で30カラムくらいのテーブルがある。そのうち数カラムには数KBの大きなデータが入るが、それらのカラムはほとんど参照しない

クラスタインデックスと主キー

- クラスタインデックス
  - 主キー値をキーに、残りの列値をすべて取得
- セカンダリインデックス
  - インデックス値をキーに、主キー値を取得
  - クラスタインデックスから、残りの列値を取得
- このため、主キー検索は高速になり、
  それ以外のインデックスによる検索の性能は落ちる

Copyright 2007 MySQL AB

クラスタインデックスの特性を生かす(1)

- 主キーは必ず定義する
  - 主キーを定義しない場合でも、内部で6バイトのクラスタインデックス用のキーが自動的に作られる。
  - アプリケーションから見ると全部のインデックスがセカンダリインデックスになってしまう
  - クラスタインデックスによる検索の高速化という恩恵を受けられなくなる。
  - 主キーが無いテーブルはテーブル設計上どこかおかしい
- 可能な限り主キーでlookupする
- INSERT系処理は、主キー値を昇順に割り当てる
  - ランダムに割り当てると再編成のオーバーヘッドが大きい。テーブルの断片化にもつながる。

Copyright 2007 MySQL AB

クラスタインデックスの特性を生かす(2)

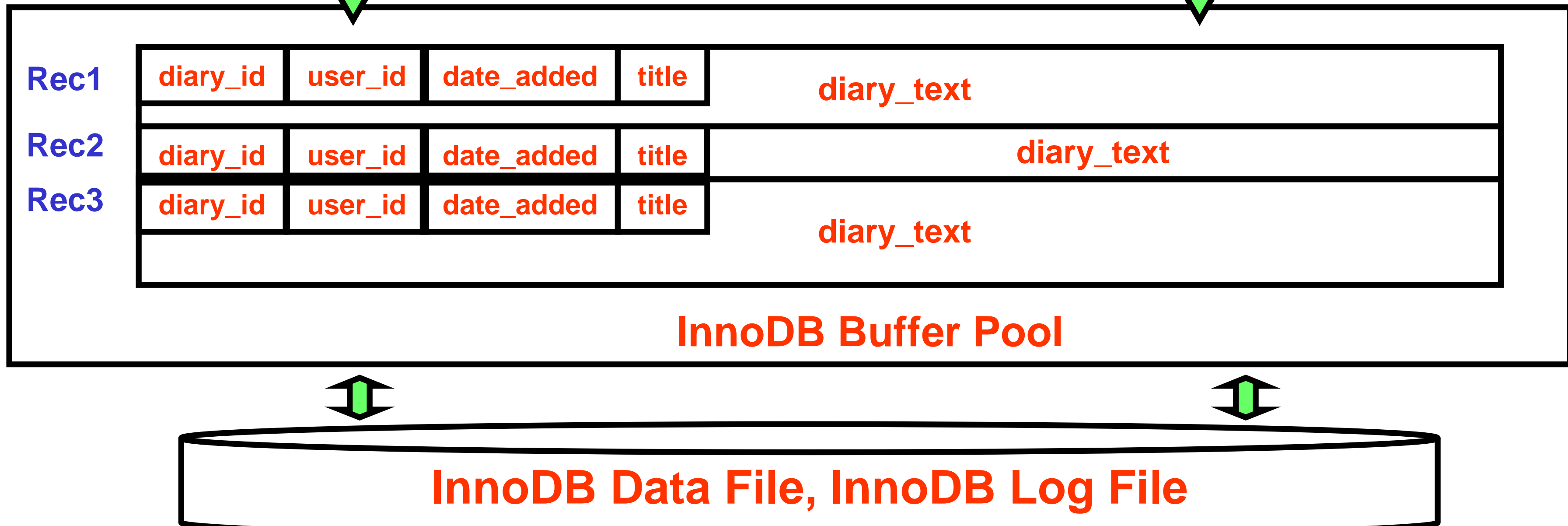
- 主キー値そのものの更新はコストがかかる
  - テーブル設計上も良くない
- 主キーのサイズは小さくする
  - セカンダリインデックスのサイズに影響する
  - UUIDよりも整数型
  - AUTO_INCREMENTについては注意が必要(後述)
- セカンダリインデックスだけで完結するクエリを書く
  - SELECT * FROM t1 WHERE second_key_part1=xxよりも SELECT second_key_part2 FROM t1 WHERE second_key_part1=xx
  - クラスタインデックスの取得→残りの列値の取得、の処理が不要になる。その分パフォーマンスが上がる

AUTO_INCREMENTとスケーラビリティ

- InnoDBのAUTO_INCREMENTの実装
  - テーブルロックを一時的にかける
    - 当該INSERT文だけ。実行が終わればロックを開放
  - 同時にINSERT数が増えれば競合により性能が落ちる
  - INSERT INTO … VALUES(null)に限らず、VALUES(100)のように明示的に値を割り当てる場合でも同様
- MySQL5.1で実装を変更
  - 同時INSERT数が300程度になってもスループットが落ちない
  - INSERT時に、複数のAUTO_INCREMENT値を予約して割り当てる
  - 理論上、番号が飛び飛びになる可能性がある

巨大な列の扱いには注意

diary_id (PK)	user_id (INDEX)	date_added (DATETIME, INDEX)	title (VARCHAR(100), INDEX(10))	diary_text (TEXT)
1	5544321	2007/06/13 21:10:14	Talking about Falcon	…….(2000bytes)
2	5544321	2007/06/13 22:13:34	UEFA Champions League	…….(700bytes)
3	2345	2007/06/14 01:12:23	Red Sox vs NYY	…….(3000bytes)



Copyright 2007 MySQL AB

巨大な列の扱い -- メモリ領域を有効に使う(ヒット率を上げる)

- SELECTで列名指定しなくてもInnoDBバッファプールには読み込まれる
  - MySQLサーバからクライアントには返されない

- 1対1関連が効果的な場合がある
  - 頻繁に読まれる列をまとめたテーブルと、めったに読まれない巨大な列をまとめたテーブルの1対1関連
  - {diary_id, user_id, date, title} と、{diary_id, diary_text(TEXT型)}
  - O/Rマッピングツールを使う場合は、1対1関連に十分に対応しているか注意が必要
    - 特にN+1 Select Problemを回避できる手段があるかどうか確認する

データ型と文字エンコーディング

- 内部的にMEMORY(またはMyISAM)のテンポラリテーブルを作ることがある
  - EXPLAINで"Using temporary"と出る場合
  - 主にグループ関数を使う場合にこのパターンになる
  - TEXT/BLOBを含む場合はMyISAMになる
  - tmp_table_size, max_heap_table_size (デフォルト16MB)
- MEMORYは実際の列値ではなく、定義したデータ型分だけメモリを確保する
  - VARCHAR(200) CHARSET eucjpmsは600バイト使う
- 文字コード変換はCPUをそれなりに使うので可能な限り避ける
  - --skip-client-character-set-handshake
  - 文字列型の列を多数含むテーブルのフルスキャンなどに注意
  - 1割～3割程度スループットが落ちるケースもある

メモリ領域の設定に関する勘違い

「InnoDBのメモリ領域の設定は特にしていません。
設定しなくても、OSのキャッシュを使ってくれるから
問題ないという噂を聞いたので。」

RDBMSとメモリ領域に関する一般論

- 全体論(特に読み取り)に関して
  - OSのキャッシュは汎用的なものである一方、RDBMSのキャッシュはRDBMSの性能を十分に引き出すために最適化して作られたもの
  - RDBMSのキャッシュ⇔OSのキャッシュのやり取りはコストのかかる処理なので、避けられるなら避けた方が良い
  - だからRDBMSのキャッシュを活用した方が速い(はず)

- 更新に関して
  - RDBMSのキャッシュサイズと、チェックポイントの頻度には強い相関がある
  - 高頻度のチェックポイントはディスク書き込みがボトルネックになる
    - 16KBを1000回fsync() vs 16MBを1回fsync()
    - ライトキャッシュの存在で大きく軽減はされるが差が無くなるわけではない
  - だからRDBMSのキャッシュを活用した方が速い
  - ※チェックポイントの間隔が長いとリカバリ時間が増えるので注意

innodb_buffer_pool_size

- パラメータの意味
  - InnoDBバッファプールのサイズ
  - デフォルト8MB
  - Dirtyな(更新された)領域の合計がこの値に近づくとチェックポイント
  - Dirtyでない(参照のみの)領域は、サイズが圧迫されると単に退避される(LRU)

- 設定指針
  - デフォルト値は少なすぎる
  - innodb_flush_method=O_DIRECTと併用することで、OSのキャッシュを迂回できるので、その分も割り当てられる
  - InnoDB専用構成であればRAMの7割～8割を割り当てて良い

- 参考
  - デフォルト値(8MB)のままだと、特に同時実行数が多く、バッファプールにおさまらない処理において劇的に性能が落ちてしまう (数十パーセント～数十分の1)
  - バッファプール退避の処理は、現行では内部的にGiant Lockを取るため同時実行性が下がる

チェックポイントに影響のあるパラメータ

- innodb_log_file_size (5MB), innodb_log_files_in_group(2)
  - InnoDBログファイルのサイズ
  - 積の値がベースになる (最大4GB)
  - Dirtyな領域の合計がこの値に近づくとチェックポイント
  - デフォルト値は少なすぎる
  - innodb_log_file_size=250MB, innodb_log_files_in_group=2程度が一般的

- innodb_max_dirty_pages_pct (90)
  - InnoDBバッファプールを占めるDirtyな領域の許容範囲を指定(パーセント単位)
  - この値を上回ると強制的にチェックポイント

- innodb_doublewrite (ON)
  - データブロックの中身をディスクに書くときに、1回だけではなく、2回書き込む方式
  - リカバリの確実性を上げる実装
  - fsync()でディスクに確実に書かれる場合にはfalseでも問題ない

InnoDBバッファプールとOSのキャッシュ(まとめ)

- 役割が重複している
- 一般的に、OSのキャッシュは使わず、全部InnoDBバッファプールに任せた方が良い

- innodb_buffer_pool_size=大きめの値
- innodb_flush_method=O_DIRECT

Copyright 2007 MySQL AB

InnoDBとロック制御

- 行レベルロック
- SELECTはロックをかけない
- ロックエスカレーションは発生しない
- Next-Key Lockingによる不整合防止

Next-Key Locking

- InnoDBログファイルとバイナリログの不整合を防ぐために必要な実装
- INSERT INTO t1 … SELECT … FROM t2
  - t2に対して共有ロックをかける
- UPDATE t1 SET xx WHERE non_index_column=x;
  - スキャンしたレコード全体(この場合はt1全体)に対して排他ロックをかける
- UPDATE t1 SET xx WHERE non_unique_index_column=x;
  - 当該インデックスと、前後に対して排他ロックをかける

Next Key Lockingの無効化

- innodb_locks_unsafe_for_binlog
  - Next Key Lockingを無効化する
  - Statementベースのバイナリログだと、不整合を起こす危険がある
  - セッション1：INSERT INTO t1 SELECT * FROM t2
  - セッション2：INSERT INTO t2 VALUES(…)
    - セッション2がセッション1よりも前に終わったとする
    - バイナリログの中身は、以下の順番になり逆転する
      - INSERT INTO t2 VALUES(…)
      - INSERT INTO t1 SELECT * FROM t2
    - INSERT INTO t1の結果が、実際の実行時と変わる

- MySQL5.1
  - 分離レベルをREAD COMMITTEDにすることと、
バイナリログを「Row Based」にすることで無効化できる

その他のパラメータ(1)

- innodb_thread_concurrency
  - 理論上の目安は(CPUコア数+ディスク本数)*2 (デフォルト8)
  - マルチコアCPUのスケーラビリティは5.0.30以降で大幅に改善
  - 0(無制限)に設定することで最大のスループットが得られることが多い
  - 接続数の増加によって急激にスループットが落ちるようであれば、4～20程度で調整すると良い
  - SHOW INNODB STATUSの結果が目安になる
    - OS Waitsが毎秒1000以上増加なら減らす
    - Mutex spin roundsが毎秒100以上増加なら減らす

- innodb_support_xa
  - バイナリログとInnoDBログファイルの同期を取る(2相コミット) (デフォルトtrue)
  - falseにすることで更新性能が上がるが、クラッシュ時にバイナリログがInnoDBログファイルに追いついていない可能性がある

- innodb_autoextend_increment
  - データファイルの自動拡張の単位(デフォルト8MB)
  - 単位時間当たりの更新量が多いようなら増やす

その他のパラメータ(2)

- innodb_file_per_table
  - テーブルごとにファイル(.ibd)を作成 (デフォルトfalse)
  - Ibdataにも依存する。.ibdだけバックアップしても動かない
  - 明示的にInnoDBのテンポラリテーブルを作る場合のオーバーヘッドに注意
- table_cache
  - 同時実行性を上げるための重要なパラメータ
  - 目安は、同時接続数×1接続あたりの同時使用テーブル数
  - Opened_tablesステータス変数に注目
  - 1024～2048程度が一般的
  - open_files_limitも大きくする
- thread_cache
  - 接続を切断した後に、再接続を軽いコストでできるようにするためのキャッシュ
  - コネクションプールを使わない場合に効果的
  - “Threads_created”ステータス変数に注目
  - 数百程度に設定することも珍しくない

Copyright 2007 MySQL AB

その他のトピック

バージョン5.0での主なパフォーマンス改善

- レコードの格納効率が改善された
  - row_format=COMPACT
  - 固定長型(INTEGER等)の長さ情報をレコードごとに持つかわりにメタ領域に持つなど、様々な面でレコードサイズのオーバーヘッドを縮小
  - 4.1以前のフォーマットはrow_format=REDUNDANT
  - 多くのケースで20%前後のサイズ圧縮になる。
I/O量が減ることによりパフォーマンスが向上する
- 5.0.30でマルチコア環境でのCPUスケーラビリティが大幅に改善された
- TRUNCATE TABLEが高速化された
  - 4.1まではDELETEと同じ動作をしていた

バージョン5.1以降での強化内容

- AUTO_INCREMENTの同時実行性の向上
- Next Key Lockingの無効化
  - 分離レベルREAD COMMITTED
  - Row based Binary Logging
- 透過的なテーブルの圧縮 (オプション。5.2からを予定)
  - row_format=COMPRESSED engine=InnoDB
  - CPU使用率が若干増加する
  - バッファプールをより効果的に使用できる
  - innodb_file_per_tableの指定が必要
- インデックスの追加/削除が高速になる (5.2からを予定)
  - 現在は、新しいインデックスを作るために、1行1行処理してテーブル全体を再編成する必要がある
  - 5.1でインデックス領域だけの再作成ができるようなインターフェースがMySQL上位層で追加された。ストレージエンジン側で実装すれば、テーブル全体の再編成は不要になる
  - InnoDB側では5.2から実装予定。削除も同様
  - 6.0からはオンラインで(更新をブロックせずに)再作成ができる予定

Copyright 2007 MySQL AB

シャットダウン時間の短縮

- シャットダウン時に行われる処理
  - InnoDBバッファプール上の、コミット済みのDirtyなデータをすべてディスクに書き込み

- シャットダウン時間の短縮方法
  - SET GLOBAL innodb_max_dirty_pages_pct=0;
  - しばらく待つ
  - SHOW GLOBAL STATUS like 'Innodb_buffer_pool_pages_dirty';
    - この値がゼロに近づく(十分小さくなる)のを待つ
  - mysqladmin shutdown (シャットダウン)

  - 計画停止などで効果的
  - 放っておいても更新が少なければInnodb_buffer_pool_pages_dirtyの値は減るが、innodb_max_dirty_pages_pctを低くすると更新量を減らす効果があるので、減り方がより急激になる
  - 0にした後は更新性能が落ちる

Copyright 2007 MySQL AB

ALTER TABLE, LOAD DATAの高速化

- SET GLOBAL innodb_flush_log_at_trx_commit=0;
- 処理
- SET GLOBAL innodb_flush_log_at_trx_commit=1;

- 失敗したらやり直せば良い
- やり直せない状況(オンラインアプリとの並行動作時など)では禁止

大量の更新処理

- InnoDBは、DELETEやUPDATEが行われても、古いレコードをすぐには削除しない
  - ほかのトランザクションから必要とされる可能性があるため
- Purge(クリーンアップ)用のスレッドが物理的な削除を行う
- 更新処理の負荷が非常に高いと、Purgeが追いつかないことがある
  - データファイルは肥大化する
  - 極端に性能が落ちることがある
  - SHOW INNODB STATUSの、"TRANSACTIONS"を見る
    - Purge done for trx's n:o , History List Length
- 多少性能を落として安定させることもできる
  - innodb_max_dirty_pages_pctを低い値にする
  - innodb_max_purge_lag
    - (History List Length/innodb_max_purge_lag - 0.5)*10秒だけ、各更新クエリを止めて更新負荷を減らす

マスターでは耐障害性、スレーブでは性能を追求

- innodb_flush_log_at_trx_commit
  - 1は最終コミット時までのリカバリを保証、0と2は保証しない
  - スレーブは壊れてもマスターから復旧できる
  - なので、ほとんどのケースで0で良い

- innodb_log_file_size
  - チェックポイント時間に影響
  - スレーブではリカバリに時間がかかっても問題ない
  - 大きめに取る (1GB程度で問題ない)
  - innodb_log_file_size * innodb_log_files_in_groupの上限は4GB

Copyright 2007 MySQL AB

マスターInnoDB、スレーブMyISAM/MEMORYの是非

- スレーブが途中で止まったときの対処に留意する

  1) START TRANSACTION;
  2) INSERT ..
  3) INSERT .. →スレーブで一意制約違反などが発生して止まった場合
  4) INSERT ..
  5) COMMIT;

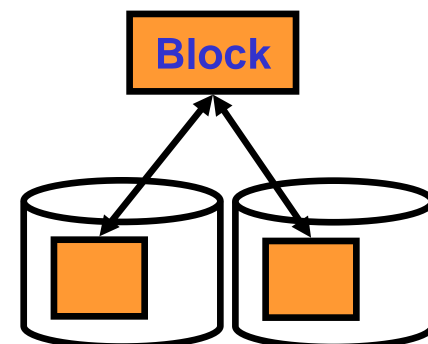
  ・マスターは1)～5)までコミットされているはず
  ・スレーブは非トランザクショナルなので2)まで書かれている
  ・だがレプリケーションを単に再開すると1)から再開される
  ・SET GLOBAL SQL_SLAVE_SKIP_COUNTER=4; 等で飛ばす必要がある

- SQLスレッドからの更新があるので、
  実際には参照オンリーではない
  - ALTER TABLEや巨大なUPDATEは、
    MyISAMだと参照をブロックしてしまう

ハードウェア選択

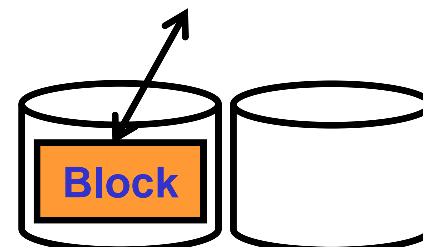
- メモリ
  - 最も重要
  - 64bit機がほとんど
  - 16GB RAMくらい搭載することも珍しくない

チャンクサイズ小：
のべ2回以上のI/Oが必要



- ディスク
  - RAID 1+0
    - RAID5は書き込み主体のアプリケーションに向かない
  - RAIDチャンクサイズ(ストライピングサイズ)
    - 最小I/O単位である、InnoDBのブロックサイズは16KB
    - 小さいチャンクサイズだと、1回の読み書きで
      複数のディスクにまたがった処理が必要
      →トータルのランダムI/O回数が増え、スループット低下
    - 256KB程度が典型的
  - ライトキャッシュ+バッテリーバックアップ

チャンクサイズ大：
1回のI/Oで取れる



その他のトピック

- アーキテクチャ設計
  - パーティショニング、レプリケーション
  - HAクラスタ
- SQL文のチューニング
- ベンチマークツール
- データベース管理
  - オンラインバックアップ、リカバリ、バージョンアップ
- その他のストレージエンジン
  - MyISAM, MEMORY, MySQL Cluster(NDB)
  - CSV, Blackhole, Archive, Memcached
  - DB2, Solid, Nitro, Infobright

MySQLコンサルティングサービス

- お客様の環境で発生した、
  MySQLに関するあらゆる技術的課題の解決を、
  短期間かつ高品質で行なう、有料のサービス
- オンサイトまたはリモートでのサービス提供
- MySQL World-Wideのコンサルティング・サポート・開発チームと直結体制での強力な支援
  - MySQLパフォーマンスチューニング
  - MySQLアーキテクチャデザイン
  - MySQLデータベース管理
  - MySQL Scale-out, HA and Replication Jumpstart
  - MySQL Cluster Jumpstart
  - MySQLマイグレーション
  - MySQL Time Hire
  - その他、MySQLの専門知識が必要なもの全般

- 詳細は http://www-jp.mysql.com/consulting/
- お問い合わせ先： consulting-jp＠mysql.com