DeNA

ソーシャルゲームのためのデータベース設計

株式会社ディー・エヌ・エー
松信 嘉範 (MATSUNOBU Yoshinori)
Twitter: @matsunobu

自己紹介

DeNA

- MySQL/Linux周りのスペシャリスト
  - 2006年9月から2010年8月までMySQL本家(MySQL/Sun/Oracle)でAPAC/US圏のMySQLコンサルティングに従事
  - 主な著書に「現場で使えるMySQL」「Linux-DBシステム構築/運用入門」「Javaデータアクセス実践講座」

- DeNAでの主な役割
  - 安定化/パフォーマンス/運用周りの中長期的な改善活動
  - L3サポート/運用/トラブルシューティング
    - 難度の高いMySQL周りの問題の根本原因の特定と解決
  - 多くのプロジェクト支援
  - 社内勉強会/トレーニング
    - MySQLやデータベース周りのベストプラクティスを社内で共有し、技術スキルを底上げする
  - 技術マーケティング
    - 国内外のカンファレンスや、技術雑誌等

本日のテーマ

DeNA

- データベース的な観点でのソーシャルゲームの特徴
- データモデル
- ソーシャルゲームに従来型RDBMSを使うべきか、流行りのNoSQLで行くべきか
- 負荷対策 (アーキテクチャ面)
- 負荷対策 (ツール面)

ソーシャルゲームの主な特徴

DeNA

- 基本無料 + 都度/月額課金
- ユーザ数が基本的に多く、一気に増えることもある
  - 初動数日でユーザー数が100万人を超えることもある
  - ユーザ数の多さから、性能面での考慮が必要
    - データモデリングは実装とは切り離されるべきというのが原則だが、やばいところは工夫しないと破綻する
- クライアント/サーバ型
  - 主な情報はすべてサーバ側で持つ
    - 「どんなデータを持つか」というデータベース的な知見、「いかにして安定化・高速化するか」といったインフラ技術の知見などが求められる
  - 行動パターンを解析し、個々の機能の人気を分析し、改善していくという継続的なアプローチをする
  - 出してからの修正が比較的容易
  - 「出すまでが勝負」の家庭用ゲームと大きく異なる

ソーシャルゲームの主な特徴 (2)

DeNA

- ユーザIDをキーにした処理が多い

- 構造化されたデータ項目が多い
  - 長い文字列 (ミニメール等)は全体から見て少数派

- 永続的に必要とされるデータと、期間限定のデータがある
  - 永続的なもの: 所持金とか所持アイテムなど
  - 期間限定のもの: 時限イベント中の行動履歴など

- 可用性や整合性に関する要求が意外と高い
  - 課金をすることでサービスへの品質要求は飛躍的に上がる
    - 「購入したはずのアイテムが表示されない」
    - 「回復ツールを使用したのにHPが減った」
  - 長時間のダウンタイムは課金収入に大きな影響がある
  - 計画停止ができないわけではない
    - 早朝5時のトラフィックは、深夜11時のトラフィックの20%未満
    - 突然止まるのと、事前に告知があって止まるのとでは印象が大きく違う

データの構成要素

DeNA

- マスタデータ
  - 一度登録されたら頻繁には変更されないもの (例 : 商品マスタ)
- トランザクションデータ
  - 頻繁に変更されるもの (例 : 受注テーブル)
- ユーザデータ
  - すべてのゲームに共通なもの(ニックネーム等)と、
個々のゲーム独自のもの(所持アイテム等)がある
  - マスタデータとしての側面も、トランザクションデータとしての側面も持つ
  - 登録日などは一度入力したら変更されない
    - ただし、新規登録が殺到している場合は「膨大なINSERT」になる
  - ユーザの状態(HP、MP、所持金等)は頻繁に変更される

マスタ系のデータモデル

DeNA

- 敵
  - 敵ID, 名前, 最大HP, 最大MP, 攻撃力, 守備力, 経験値, お金, 登録日
- 職業
  - 職業ID、職業名、登録日
- データサイズが性能を圧迫するほど巨大になることは少ない
  - 1台のマスタ + 冗長化スレーブが典型的
  - 正規化をきちんとする
  - インデックス設計も重要

派生関係

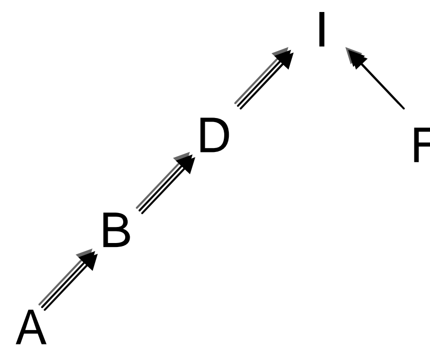
DeNA

- 「1 対 0..1」型のテーブル関連
- アイテムを例に取ると...
  - アイテム
    - アイテムID, 名前, アイテム種別, ゲーム内価格, 説明文, 登録日
  - 攻撃系アイテム
    - アイテムID, 攻撃力, 攻撃属性, 使用回数上限
  - 防御系アイテム
    - アイテムID, 防御力, 防御耐性, 使用回数上限
  - 補助アイテム
    - アイテムID, 効果範囲

階層関係

DeNA

- ミッションAをクリアするとミッションBが出現
- ミッションBをクリアするとミッションDが出現
- ミッションDとFをクリアするとミッションIが出現



1 ― 0..n

ID	ミッション名	上位ID
A	ミッションA	B
B	ミッションB	D
D	ミッションD	I
F	ミッションF	I
I	ミッションI	-

Dの上位をAにすると無限ループに陥る

1 ― 0..n

ID	ミッション名	階層区分	上位ID
A	ミッションA	0	B
B	ミッションB	1	D
D	ミッションD	2	I
F	ミッションF	2	I
I	ミッションI	3	-

上位の階層だけを上位ダンジョンに登録できるようにすれば登録ミスの可能性を大幅に減らせる

1個のミッションをクリアしたときに、2個以上新しいミッションを出すことができない

「階層型部品表」タイプ

DeNA

- 戦士(1) + 武闘家(2) → バトルマスター(8)
- 武闘家(2) + 僧侶(3) → パラディン(9)
- 僧侶(3) + 魔法使い(4) → 賢者(10)
- バトルマスター(8) + パラディン(9) → ゴッドハンド(12)
- 踊り子(5) + 笑わせ師(6) + 吟遊詩人(7) → スーパースター(11)

職業ID	職業名
1	戦士
2	武闘家
3	僧侶
4	魔法使い
5	踊り子
6	笑わせ師
7	吟遊詩人
8	バトルマスター
9	パラディン
10	賢者
11	スーパースター
12	ゴッドハンド

1

0..n
0..n

上級職ID	下級職ID	要レベル
8	1	-
8	2	-
9	2	-
9	3	-
10	3	-
10	4	-
11	5	-
11	6	-
11	7	-
12	8	-
12	9	-

生産管理システムの「部品表」がルーツのデータモデル

ユーザIDを主キーに持つテーブル

DeNA

- ゲーム共通情報 (モバゲーではOpenSocial APIで提供)
  - ユーザID、自己紹介、誕生日、趣味、性別、職業、血液型、プロフィール画像URL
- ゲームのメイン情報
  - ユーザID、登録日、更新日、レベル、経験値、所持金、HP、技P、術P
- イベント中のユーザ情報
  - ユーザID、勝利数、敗北数、引分数
- セッションデータ
  - ユーザID、最終ログイン時刻
- いろいろ
  - ユーザID、足あとを踏んだ回数、招待した回数、etc

■ ユーザIDだけを主キーに持ったテーブルのレコード数は限定的

- ユーザ数が上限 (数百万 – 1,000万単位にはなりうる)
- 1000万レコード×100バイト/レコード = 1GB

■ 派生関係を多用する

- 時限イベントの時だけ必要とする列がある

■ 1対1関連を使うこともある

- 極めてアクセス頻度の高いものは、専用のテーブルを用意してそこに集中
  - いざというときに他のサーバに移すことも容易

ユーザID+αで一意になるもの

DeNA

- 所持アイテム
  - ユーザID、アイテムID、個数、使用回数
- ダンジョン攻略
  - ユーザID、ダンジョンID、クリア回数
- 日記 (履歴系)
  - 日記ID、ユーザID、作成時刻、更新時刻、タイトル、本文、ステータス

- ユーザID+αで主キー/一意になるものはレコード数が飛躍的に跳ね上がる
  - 1000万レコード – 10億レコード超えは珍しくない
  - コメント/日記など、長い文字列を扱うテーブルのサイズは非常に大きくなる

- アクセス範囲に極端な局所性がある場合も
  - 日記は、最近のデータにアクセスが集中する

ソーシャル系

DeNA

- 人と人との関係を管理
  - 仲間リスト
    - ユーザID、仲間のユーザID、招待日、承認日、ステータス(承認待ち等)
  - 対戦成績
    - ユーザID、対戦相手のユーザID、対戦日, 戦闘結果
  - 足あと
    - 踏まれたユーザID、踏んだユーザID、踏んだ日にち、踏んだ回数
- ソーシャルゲームならではのテーブル
- レコード数は非常に多くなる
  - ユーザ数×平均友人数
  - 範囲検索やソートは注意して使う
    - ヘビーユーザほど友人数は多い
  - 登録可能な友人数に上限を設けることも検討する

RDBMSかNoSQLか?

DeNA

- MySQLのような従来型のリレーショナルデータベース(RDBMS)はソーシャルゲームには向かない?
  - そんなことはない
  - モバゲータウンではデータストアにMySQL、サマリ等のキャッシュにmemcachedを使用
  - ハードウェアの大幅な高速化(Nehalem CPU、大容量メモリ、SSD)により、必要十分な性能が出せることが多い
  - 複雑なクエリ、トランザクション、オンラインバックアップ、モニタリング、ユーティリティツール等はソーシャルゲームでも重宝する

- RDBMSは性能が悪い (秒間1000回の更新もさばけない) ?
  - モバゲータウンでのMySQLの更新トラフィックは1台あたり10,000 update/sを超えているものもある
  - 更新がさばけるかどうかの大きなポイントはランダムI/Oとインデックス設計
  - ディスクI/Oネックに陥れば、どんなNoSQLでも満足いく性能は出せない

- 「スキーマ」「トランザクション」「インデックス」はもっと評価されるべき

「スキーマが無い」とはどういうことか

DeNA

- 第3者からは、どこに何があるのか分からない
- 分かるのは、作った人だけ

ソーシャルゲームの負荷対策

DeNA

- 「初動ユーザ数が急増しうる」とは何を意味するか
  - 開始間もないうちはデータ量が少ない
    - よほど問題あるクエリを実行していない限り、どんなデータベースを用いても性能面では問題ない
  - データ量が激増する
    - 注意すべきは「ユーザID + α」をキーとしたテーブル
    - データベースサイズが1日数GB単位で増えることはざら
  - やがて急激に遅くなる
    - 分岐点は、アクセス範囲がメモリにおさまらなくなる段階
- 参照処理の負荷対策はそれほど難しくない
  - スレーブを追加し、アクセスを分散させれば良い
- 更新処理の負荷対策が難しい
- テーブルの性質や、INSERT / UPDATE / DELETEの頻度に応じて対策は変わる

INSERT主体のテーブル

DeNA

- アクセスログ、日記、対戦履歴などの履歴型テーブル
  - ひたすら蓄積(INSERT)し、SELECTもする
  - SELECT範囲に極端な偏りがある(最近のデータにアクセスが集中)
- INSERTの一般的な性質
  - B-Treeインデックスを更新するために、対象のブロックをメモリにロードする
  - 巨大なインデックスになると、メモリにおさまらずディスクからの読み込みが発生
  - インメモリで高速に終わっていたものが、ある時点からディスクバウンドに移行
    - 突如として大幅な性能ダウンに遭遇する
    - データ量の増加により、UPDATE/DELETE/SELECTも低速になる
  - どれだけ高速なストレージでも、インメモリの処理に比べて著しく落ちる
  - それでもInnoDBのINSERT性能は高い (Insert Buffering)

インメモリでINSERTを完結する

DeNA

- インメモリでのINSERT性能 (InnoDBの場合。以下同様)
  - 秒間15000 insert/s超えクラス (単一スレッドで)
- バッファプールを超え、ディスクreadが多発する場合のINSERT性能
  - 秒間2000 -4000 insert/sくらいまで落ちる (HDD、単一スレッドで)
- 高速ストレージでも実は状況はさほど改善しない
  - FusionIOでも、せいぜい5000 insert/s程度しか出ない
    - 実はInnoDBはメモリにおさまらないINSERTはCPUネックになる
    - チェックサム計算、高頻度のmalloc/free等に多くのCPUリソースを追加で消費してしまう
- ディスクreadが多発し始めた頃から、スレーブ遅延が深刻になる
  - 遅いからSSDで、で解決するわけではない
- INSERTをインメモリで完結する上で重要なことは「インデックスサイズを小さくすること」

時系列処理の高速化アプローチ

DeNA

- INSERTをインメモリで完結することが大切
  - 日付・時刻型の列でレンジパーティションを組む
    - MySQL5.1以降でサポートされた機能
    - パーティションそのもののオーバーヘッドはあるが、ディスクI/Oによる性能劣化を防げるメリットの方が圧倒的に大きい
    - INT型やTIMESTAMP型であれば秒単位の粒度まで設定可能
    - 古いデータをALTER TABLE .. DROP PARTITIONで高速削除可能
  - テーブルを日時別に分ける
  - 古いデータを削除するか、別のサーバに移す
  - インデックスを作らない
  - 無駄な列を作らない
  - スペース効率の良いデータ型を使う
  - 大容量メモリを搭載する

UPDATE主体のテーブル

DeNA

- 用途
  - ステータスの更新など
    - 現在HP、経験値、所持金、個々のアイテムの所持数、対戦成績、etc
    - ソーシャルグラフを扱うテーブルではUPDATEが最も頻発しやすい
- UPDATEの性質
  - UPDATE対象のレコードを読まない限り更新はできない
    - インメモリで完結すれば高速だが、データ量が多くなればディスクからのreadが頻発する
  - データサイズの増え方は緩やか(INSERT次第)
    - 性能が突然大きく落ち込むということは無いので、計画は立てやすい
  - HDDとSSDで大きな性能差がある
    - インメモリUPDATE: 12,000/s
    - HDD UPDATE: 300/s
    - SATA SSD UPDATE: 1,800/s
    - PCI-E SSD UPDATE: 4,000/s

その他の高速化アーキテクチャ

DeNA

- Sharding (水平分割)
  - データそのものを複数のサーバに分散し、1台あたりのデータサイズ/トラフィックを減らす
  - 重要な技術だが、これだけで1コマは必要な内容なので省略

- NoSQLを併用する
  - データ量はメモリにおさまる範囲だが、アクセス頻度が極端に高いものがある
    - 友人の最終ログイン時刻
    - 友人の足あと
  - SQL文の処理自体がオーバーヘッドになる
    - SELECT文は100,000回/秒程度が上限
    - INSERT/UPDATE/DELETE文は15,000回/秒程度が上限
  - NoSQLであれば、インメモリのデータアクセスを高速にさばける
    - 一意検索は200,000-750,000回/秒クラス
    - 更新系は30,000回以上/秒クラス
  - DeNAでは「HandlerSocket」というMySQLプラグインを自社開発し、NoSQL APIでInnoDBテーブルを高速アクセスしている
    - RDBMSのメリットを享受しつつ、NoSQL APIで高速アクセス

- 非同期に書きこむ
  - 瞬間最大風速が極端に大きいものに対して効果的
  - Q4Mが定番。「インデックスの無いテーブル」でも実現は可能

負荷対策の日常

DeNA

■ 問題を発見・追跡できるツールを整備

- パフォーマンス統計グラフの取得
  - CPU使用率、IOPS、CRUD別実行頻度、テーブルサイズ推移
  - InnoDBの各種統計情報など

- 各種エラー/異常数値の追跡
  - ハードウェア、OS、ミドルウェアの死活監視
  - エラーとなった接続/クエリ/トランザクション
  - レスポンスタイム
  - 同時接続数
  - レプリケーション遅延

    開発環境では意図的にレプリ遅延状態を起こすテストも実施

- 異常系のクエリ/トランザクションの追跡
  - 実行に一定時間以上かかったクエリ
  - 実行回数の極端に多いクエリ
  - 実行に一定時間以上かかったトランザクションと、その原因となったトランザクション

遅いクエリを改善する

DeNA

- インデックスに対する正しい理解が必要
  - インデックスが使われないのはどういう時かを理解する
  - 範囲検索とCovering Indexの関係性を理解する
  - ソートがなぜ重くなるのかを理解する
  - 更新性能がなぜ落ちるのかを理解する
- SQL文の実行計画を確認する
  - レコードを読み過ぎているクエリを直す
  - EXPLAINの読み方を習得する
- MySQLのオプティマイザは判断を誤ることがあるので気をつける
  - 特にソートが絡む場合
    - INDEX i1 (user_id, item_id, date), INDEX i2 (user_id, date)の2個のインデックスがあり、WHERE user_id=100 ORDER BY dateをした場合、i2ではなくi1が使われることがある
  - 自信がなければFORCE INDEX等で実行計画をコントロールすると良い
  - 多くのRDBMSはコストベース(実行計画は実際のレコード分布に依存)なので、テスト環境と本番環境では実行計画が変わることがある

実行頻度の多いクエリを減らす

DeNA

- 平均実行時間×実行回数 = トータル実行時間
  - 1回あたりの実行時間がいくら短くても、
    実行回数の極端に多いクエリは性能問題を起こす
- 典型的な例
  - 仲間(候補)一覧を抽出・表示
    - SELECT … WHERE target_user_id= N
    - 仲間(候補)の数だけ繰り返し実行すると、1リクエストあたり100-1000回ということもざらにある
    - IN句で1回で済ませると良い
- 有用なツール
  - mk-query-digest (Maatkit)
  - mprofile
  - mysqldumpslow

問題のあるクエリを探して直す

DeNA

pct	ts_cnt	Query_time_pct_95	sample query
25.7%	611	0.000925767	SELECT … WHERE user_id = 9460982
16.4%	382	0.000403909	SELECT … IN (1,2)
15.5%	40	0.00270811	SELECT … WHERE item_id= 100 ORDER BY xxx
7.90%	297	0.000467575	SELECT …
6.47%	71	0.00202083	SELECT …

- CPU使用率、iowait等から問題のあるサーバを特定する
- 全体に占める実行時間の割合の高いものからリストアップし、問題のあるものを直していく
- 健全なクエリであれば、平均実行時間は0.001s未満になるはず
- 実行回数の極端に多いものは、回数を減らすようなクエリを書けないかを検討する
  - IDによる一意検索の繰り返し→IN句の利用はその典型

トランザクションを正しく使う

DeNA

- 可能な限り一連の処理を1回のトランザクションで書く
  - MySQL(InnoDB)を使う最大の理由
    - 「ダンジョンAのクリアフラグを立てる→ダンジョンCの攻略開始可能フラグを立てる」
前半の更新だけで終わると、クリアしたが次のダンジョンに進めないというハマリ状態に
    - 1回のリクエストで1箇所しか更新しないということはまず無い
    - 1箇所しか更新しなくてもその途中でエラーが起こると中途半端な状態に
  - 例外をとらえたら適切にロールバックする
    - 握りつぶしてコミットしてはいけない(中間状態で確定してしまう)
  - トランザクション非対応のテーブルを更新→InnoDBを更新→ロールバック
    - InnoDBだけがロールバックされるので、矛盾した状態になる

- 楽観ロックによる整合性管理
  - 複数のトランザクションにまたがって整合性を管理したい場合に効果的
    - 例:チケットの在庫があることを確認→購入
  - Versionパターンとして知られる

遅いトランザクションを改善する

DeNA

- 「個々のクエリは速く、クエリ数も少ないが、トランザクション開始から終了までにやたら時間がかかっている」というケースがある
  - ロックを取った状態でコミット/ロールバックまでに時間がかかると深刻な問題になる
- 観測される症状
  - 同じレコードをロックしたいセッション全部が待ち状態になる
  - 高確率でLock Timeoutエラーとなる
- レコードの排他ロックを取ったまま、数秒間切断していないようなセッションを特定する
  - モバゲーでは実行に一定時間以上かかったトランザクションだけを記録するツールを用意し、問題箇所を特定している

ストールを防ぐ

DeNA

- すべてのクライアントが一定時間(1秒未満-数秒)何もできなくなることがある

- 観測される症状
  - 同時接続数が激増する、あるいは接続エラーになる(1秒間に1000リクエストの来るサービスで、2秒間止まれば接続数は2000になりうる)
  - レスポンスが一定時間以上返らない

- 主にRDBMSの実装に起因することが多い
  - 知っていれば防げるものが多いのでベストプラクティスに従う

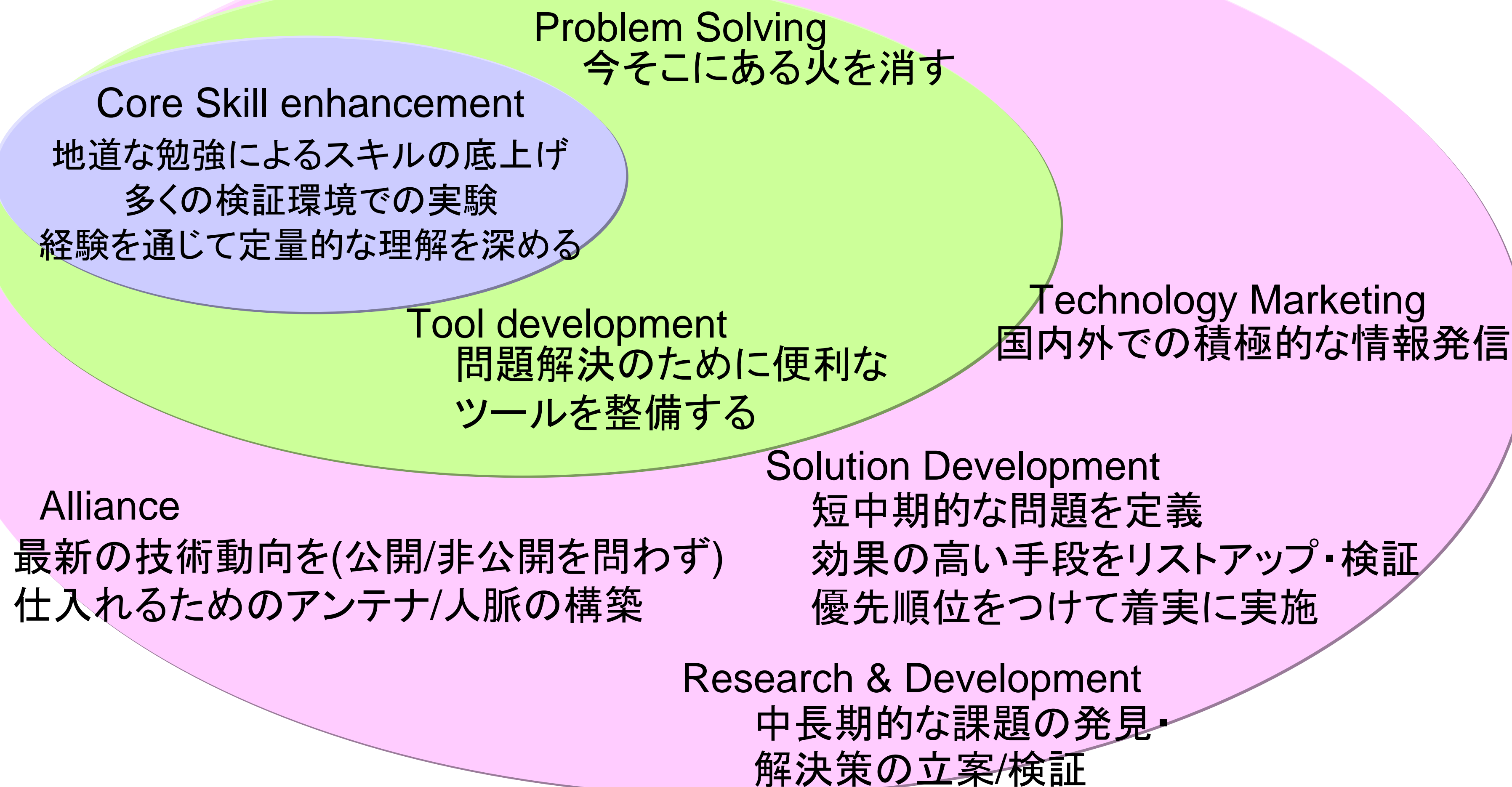
- MySQLの場合であれば..
  - pthread_create()/clone() (接続時にスレッドが生成される場合)
  - LOCK_open (巨大テーブルのDROP, FLUSH TABLE)
  - バーストライト(Log Preflush等)
  - Rollback Segments制限
  - Buffer pool mutex
  - などなど

より詳しい技術解説に興味のある方は。。

- 書籍「Linux-DBシステム構築/運用入門」(翔泳社)
- Developers Summit (デブサミ) 2011
  - 2月17日・18日 目黒雅叙園 (17日に登壇します)
  - 「大規模Webサービスのための データベース技術の現在・未来」
  - 17日には、DeNAのセッションが丸1日あります
- MySQL Conference & Expo 2011
  - 4月11日-14日 (米サンタクララ)
  - Linux and H/W optimizations for MySQL
  - Automated, Non-Stop MySQL operations and failover
  - Using MySQL as NoSQL – Introduction to HandlerSocket Plugin

インフラエンジニアのキャリア

DeNA



■ DeNAはこうしたキャリアを積み上げる上で非常に向いています

■ もちろん、ソーシャルゲーム本体の開発者も絶賛募集中です

メッセージ

DeNA

- データモデリングは重要
  - ゲームといえども、
データベースは単なるデータの置き場にはならない
  - ゲーム以外のシステムでの経験が無駄になることは無い
- 負荷対策のスキルを身につける
  - RDBMSをNoSQLにしたら解決するものではない
  - なぜ性能が出ないのかという根本部分での理解が大切(思考停止しない)
  - 状況を的確に把握できるためのツールを整備する
- 理論と経験のバランスが大切
  - 理論だけだと現実が見えず、経験だけだと最適解への到達が困難
  - 両方を積み上げていける場所は限られている
- 目を外に向ける
  - オープンソースは使うだけのものではない
  - 海外の技術カンファレンスで発信する日本の会社がどれだけあるか?
- DeNAは人材募集中です
  - 懇親会などでお気軽にご相談ください