KVS, Flash を用いた高速データ検索、高速通信Webシステム

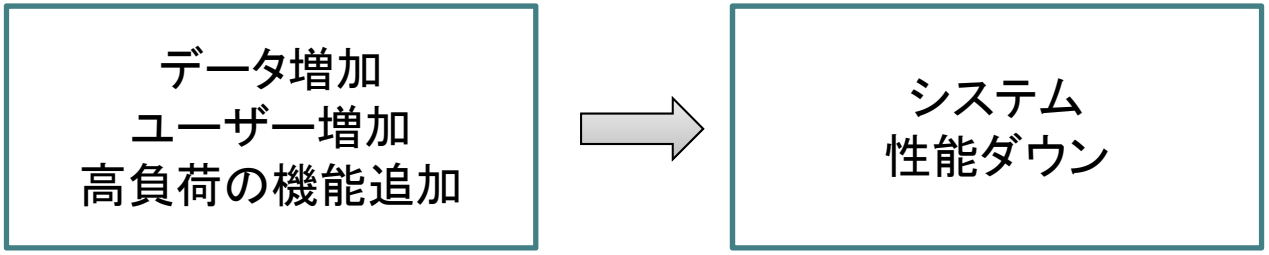
株式会社　トエツ･ジャパン

RDB を用いたシステムでのデメリット1

- システム性能がダウンするケース

  RDB を用いたシステムでは適切なチューニングを行えば高性能アプリの開発が可能である。

  しかし、システム投入以降、大幅なデータ増加やユーザー増加により、システムの処理能力が大幅に低下する場合がある。



RDBを用いたシステムでのデメリット2

- システム性能低下からの復旧

  性能低下システムはハードウェアの増強、アプリケーションやDBを修正することで元の性能に戻すケースが多い。

  しかし一般に、

  - ハードウェア増強行われない場合もある（予算の都合上）
  - アプリ、DBの修正には不具合発生が伴う
  - 性能低下→元の性能に戻すには高スキルが必要
  - 高スキルの担当者がすぐには見つからない状況もあり得る

  結果、性能低下したシステムはそのままとなり、
  業務が滞る、企業の信用低下に陥るケースも存在。

性能低下の回避策

- クラウドの利用もひとつ

  ハード増強やアプリ、DB修正を行うのではなく、
  簡易な方法で性能低下から復旧できればよい。
  →クラウド上にシステム構築することで実現可能。

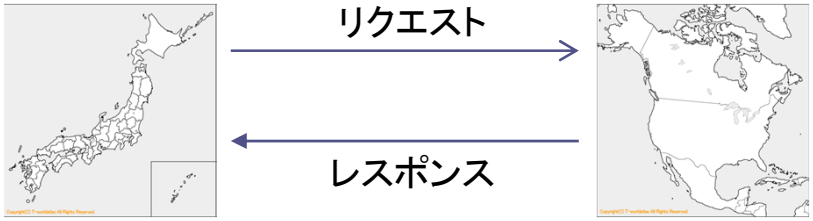
  クラウドでは性能低下時にサーバーを動的に追加したり
  メモリ、CPU、Diskなどの資源を簡易に追加できるため
  システムの性能低下が起きにくい。

  反面、クラウド向きではないケースも存在する。

クラウド向きではないシステム

- 通信速度

  メジャーなクラウドサービス（Google, Amazon, Azure）はサーバーが国外に配備されている。→そのためデータ通信が若干遅くなる。



大量データ通信となる場合、通信遅延が顕著になる場合がある。

システムで高速レスポンスが要求される場合、国外のクラウドを利用するのは難しい。
かといって、日本国内で提供されているクラウドサービスについてはメジャークラウド業者に比べて高コスト。

レスポンス悪化の発生メカニズム1

- DISK I/O による低速化

  システムが低速になるその理由

  1. 大量データ処理

     データ保持サーバー（RDBなど）で大量のDISK I/Oが発生する。

     →DISKはメモリより数万倍遅い

  2. ユーザー増加

     ユーザー1人の処理は低負荷であってもユーザー数が増加

     →OSキャッシュにデータが乗り切らない

     →スワップ発生、DISK I/O 多発

  3. 高負荷機能の追加

     システムに高負荷（=DISK I/Oが大量に発生）の機能が追加される。

     →OSキャッシュにデータが乗り切らない

     →スワップ発生、DISK I/O 多発

     低負荷処理も高負荷処理に引きずられ、レスポンス悪化。

システムにおいて、DISK I/O が頻発するとレスポンス悪化となる
特にメモリ不足によるスワップ発生は致命的。

レスポンス悪化の発生メカニズム2

- 通信量 による低速化

  システムが低速になるその理由

  1. 大量データ通信

     遠距離におけるサイズの大きなデータのやり取りはコスト高。

     →国内～海外サーバー間に高速通信網が存在していたとしても

     経路途中で DNS ホップが数回発生する。

  2. TCP/IP の乱発

     たとえば、Web上、小さな画像を大量にやり取りする場合、

     TCP/IPパケットが大量に発生する。

     Keep Alive 設定を行っていたとしても TCP/IP 通信コストは UDB等に比べコスト高。

     例として画像を挙げたが、通常のテキストデータも分割して取得するとコスト高。

遠距離データ通信、大量パケット通信はコスト高。

ただし、高速度レスポンスが不要の場合、通信コストは問題にはならない。

弊社開発事例について1

- 既存システム

  Local Access 上でデータ検索、データ入力を行っている。
  AP使用ユーザーは日本全国の在宅勤務者 100～150名
  Access 上の検索対象データは 2～5万件。
  日本語あいまい検索を行っている。
  （実際には SQL の like 検索：前方、後方あいまい検索）
  入力したデータはそのまま次回のデータ入力の検索対象となる。
  検索レスポンスは 1～3秒。

  Local Access 上、レコード数 2～5万件ならば
  　SQL : select * from 実績データ where 商品名 like '*検索語*'
  は確かに 1～3秒のレスポンスが可能。

新システムで Webシステムに移行

弊社開発事例について2

- 新システム業務要件

  Web システムとする。 入力データ、検索データは一元管理
  既存の入力データを流用。
  →Local Access 上のデータ 2～5万件 × 100名分をまとめる。重複データは少ない。

  初期データ数 300万件。 年間 150～300万件のデータ増加。
  最大 1000～1500万件のレコードを考慮。（今後、2倍～5倍になる可能性もあり）

  日本語あいまい検索。
  検索結果データは2階層のツリー構造。
  1階層目は最大100件程度。2階層目は検索該当データのサマリ化。
  サマリ化は地区別、分類別にサマリ化し、ヒット件数に応じてソート。
  検索該当件数は最大 10万件程度になる場合もある。
  Google検索のように、単語入力した瞬間に検索結果一覧が欲しい。
  レスポンスはAccess 程度の 1～3秒は欲しい。速ければ速いほど良い。

  とにかく低コストにしたい。サーバー台数も抑えたい。
  Oracle RAC など高価なサーバーは導入できない。
  SQL Server Enterprise も導入できない。
  現実的にはWebサーバー1台、DBサーバー1台

プロトタイプその１

- RDB の利用

  DB は OSS を考慮し、Postgres を採用。
  - 日本語全文検索ライブラリ MeCab を透過的に利用できる。
  - パーティション分割が可能
  - データファイル分割型なので、OSキャッシュに乗りやすい。

  検索AP 側でマルチスレッドによるDB検索を行う。

  ユーザーの検索リクエストは0.2～2秒間隔を想定。（英字入力時も考慮）
  データ件数 1000万件で試験。

  結果、ユーザー数 2～3名程度であれば、なんとか運用できる。
  ↓
  ユーザー数 5～10名シミュレーション時、
  スワップ発生。および、CPUコンテキストスイッチ頻発。

結果、レスポンス 5秒以上のケースが頻発

プロトタイプその２

- クラウドの利用

  データストア先を Google に変更。
  Google App Engine for java にて 平均レコードサイズ 1M のダミーデータを
  レスポンス。
  ↓
  結果、5秒越えの通信コストが発生。
  （高速に結果が返る場合もある）
  レスポンスコストのばらつきについては原因不明。

クラウドの場合、通信コストが問題なのでは？

プロトタイプその3

- KVS の利用

  検索対象データをオンメモリ KVS に配備。
  ↓
  KVS 上でデータヒットした場合。検索終了。
  データヒットしなかった場合、RDB に検索。

KVSヒット時の検索コスト=平均0.02秒以内。
KVS側で検索ヒットすればするほどRDB負荷も減少。
8割KVSヒットの場合、RDB検索コスト=平均 0.1 秒以内。

KVS について1

- KVSとは

  Key Value Store の略。
  →一意のKey に対してひとつの値 (Value) を返すだけの機能。

  RDB のような複雑な処理は行えない。
  →しかし、負荷分散、高可用性に強い。
  反面、集計処理や複雑な検索に弱い。

  近年発達しているクラウドの基盤技術のひとつ。
  Google , Amazon, Azure はKVSベースのサービスを多数提供
  → Google 検索, GMail, Google Map, YouTube, Aamzon Storage …

  クラウド企業以外でも、KVS を用いたシステムは多数存在。
  twitter, mixi , Facebook, 楽天, flicker, livedoor …

KVS について2

- 一般的なメリット
  - オンメモリKVSは 超高速。
  - 永続化対応KVS も高速。
  - 1台のKVSでRDB の数百倍の性能になる場合もある。
  - 分散スケールアウトが容易。

- 一般的なデメリット
  - RDB での JOIN が出来ない。
  - Key 1つに対して Value 1件のみのレスポンス
    （製品によっては範囲検索やあいまい検索のものもある）
  - トランザクションなし
    （製品によっては CAS 機能を持つものもある）

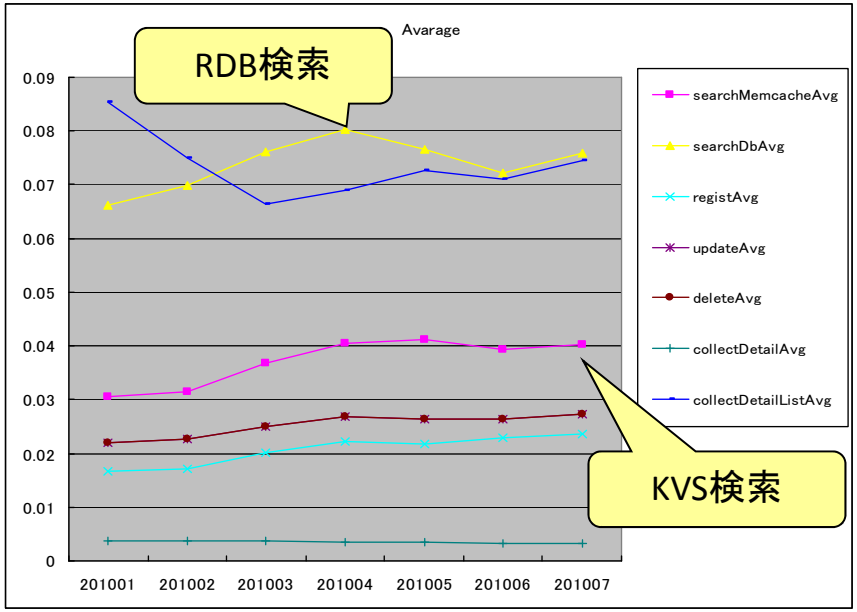
KVS について3

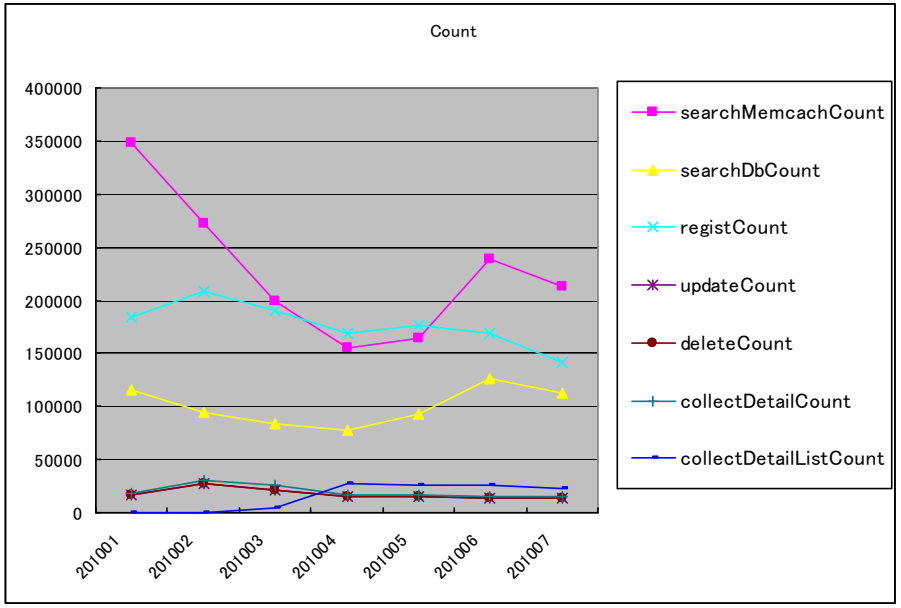
- 多種多用なサーバー製品
  - メジャーなKVSプロダクトとして memcached が存在する。
    - memcached はオンメモリKVS 。OSS で普及。
  - 他にもデータ永続化対応、あいまい検索可能、集計可能など特徴的な製品が多数存在。
    - memcached , Redis, Lux-IO, TokyoCabinet, Flare,
    - Cassandra, HBase, Groonga, kumofs .....
- どれが良いのか
  - それぞれの KVS の特性を理解し、
  - 仕様用途に合わせて KVS を選ぶ必要がある。
    - →新システムでは制約上、memcached を使用
      - （別途、TokyoCabinet, Groonga , kumofs も調査した）

実際にKVSを利用した感想

- 現実的なメリット
  - KVS1台だけで相当高速になる。
  - デフォルト設定の KVS と チューニング済みの RDB を比較
    - →KVS の方が 2～10倍高速。(RDB 使用メモリは 4G ： KVS は 1G)
- 現実的なデメリット
  - KVS 向きのデータ構造を考慮する必要がある。
    - →RDBの常識は通用しない。
  - メモリは貴重
    - →メモリサイズ以上のデータをmemcached には乗せられない。
    - →高速化したいデータのみをmemcachedに乗せる。
  - JOIN がほしくなる。
    - →メモリ資源が少なかったので一部正規化によりデータ量削減。

コスト計測





KVS検索コストはRDBの約半分。
KVS検索リクエスト数はRDBの2倍～3倍
→単純に約4～6倍の性能
参考までに：新システムでは月間80～100万リクエストが発生。

さらなる高速化

- KVS だけでは足りない

  新システムでは大量データをクライアントに返却するケースが考えられる。
  サーバー側を高速化したとしても通信速度が遅いと業務に支障が出る。

- エンドユーザー環境

  主たるエンドユーザーは在宅勤務者。
  　ADSL 30M～50M 程度 、無線LAN 接続も多い

  使用しているパソコンのCPUやメモリが低スペックであることも想定

通信について

- HTTP通信データ形式

  検索結果データについて

  テキストそのまま、JSON、XML、SOAP

  などに変換。

  Webサーバー側で deflate 圧縮を実施。keepAlive 設定ON。

  結果、確かに通信量は削減されるがクライアント側での高速化に限界がある。

  →特に XML形式データ の場合、DOM 展開が遅く、業務で使用するには遅すぎる。

  他にも数々のクライアント側高速化技法を使用。

  →データ量が多いと限界がある。業務で使用するには遅すぎる。

  クライアント側で非同期にデータやり取りを行ったとしても遅い。

  サーバー側でサイズ圧縮のためバイナリデータをBase64 変換を行った場合、

  クライアント側のデコードコストで使い物にならない。

テキストベースのデータ通信では限界がある

Flashについて

- 通信

  Flash は操作性、ユーザーインターフェスが優れている。
  その他に、Flash はネットワークAPIが充実している。
  バージョン9.0(ActionScript3.0)以降、バイナリ通信が可能。

- AMF

  Action Message Format の略。Flash APP で使用されるバイナリデータのフォーマット。
  AMF0, AMF3 の２種が存在。
  AMF3 は高圧縮のデータフォーマット。
  Flash Ver9.0 以降より AMF3 によるバイナリデータ通信が可能となった。
  HTTP 、HTTPSプロトコルで通信する。
  RPC可能。
  クライアント側は通信データをそのままネイティブ利用可能。
  (テキスト→バイナリ変換やXMLパースが発生しない)
  ネイティブデータのためクライアント側も高速。

通信コスト比較

- 比較サイト

  http://www.jamesward.com/census/

  上記サイトでは

  Ajax HTML, SOAP, XML. XML E4X, AMF3 等のフォーマット別の通信コスト比較が可能である。

  AMF3 の通信コストを 1 とした場合、

  XML E4X = 1～3

  XML = 3～5

  SOAP = 10～20

  HTML = 50～100

  の結果が得られる。

新システム結果

- データ量、ユーザー数
  - 既存システムの500倍のデータ量。ユーザー数 100倍。
  - 検索対象データ件数 2010/07時点 で 500万レコード
- データ検索コスト
  - サーバー側KVS検索+AMF3変換コスト
    - 平均 0.03sec
  - サーバー側RDB検索+AMF3変換コスト
    - 平均 0.08sec
- データ通信コスト
  - クライアント検索開始から結果取得コスト（通信時間含む）
  - ADSL 50M , 無線LAN 802.11g (理論値 54Mbps)の場合
    - 約 0.3～1.0 sec

RDBも速い

- RDB だけでも良い。

  実運用時、RDBの検索性能が思ったより良かった。
  ↓
  サーバーマシン、とくに SAS の性能が良かった。
  (SASのディスクキャッシュの恩恵を受けた)
  ↓
  KVS を使用しなくても問題ないかも？
  ↓
  不具合発生

不具合

- 2010/04/22 AM9:15 不具合発生

  関連業者が夜間バッチの大量データ取得を通常業務の時間帯に実行。
  →約 15分のシステム遅延が発生。
  RDBに接続する全機能、全画面で遅延が発生。
  ブラウザ側でコネクション切断など多発。
  通常 RDB検索平均コスト 0.07 sec → 60sec オーバー。
  →通常運用時より 1/800～1/1000の性能低下。

- KVS検索

  何も影響を受けず。

運用ミスによりシステムはダメージを受けたが、
逆にKVS の優位性を確認できた

ノウハウ1

- KVS のデータ構造

  RDBの常識は使えない。

  ↓

  Google 検索はなぜ速いか？

  あらかじめ任意の語による検索を実施し、
  検索結果を別領域に保持。
  検索語が入力された直後、該当する検索結果から表示ページに応じてレスポンスを返している。
  (Google の Blueprints や Google, Amazon の論文等)

  ↓

  RDBとは考え方が逆。

新システムでも類似した方法を利用

ノウハウ2

- KVSを利用する場合
  - 1：すべての業務、画面を KVS 対応にするにはメモリが足りなかった。
  - 2：KVS 用のデータ構造について考察が必要。
  - 3：KVS 用データを作成するためのバッチは複雑になりやすい。
  - 4：KVS 用フレームワークは発展途中。（汎用的なものは無い）
  - 5：高速性を求めない機能については通常の RDB システムでも良い。
    - →開発コストは低い。ただし RDB のデメリットは付きまとう。
    - →KVSの方が段違いに高性能。
  - 6：開発事例は RDB に比べ少ない。
    - →mixi, twitter, KLab 等の開発事例など非常にためになる。
    - （twitter では 300T のオンメモリ memcached を使用）

ノウハウ3

- Flash について
  - 1：非同期に連続リクエストを投げる場合、特殊な方法が必要。
  - 2：Flashでは日本語入力で潜在不具合が存在するため、回避が必要。
  - 3：クライアント端末上にキャッシュデータを作ることでより高速化。
  - 4：AIR アプリケーションにも応用できる。
  - 5：Flash の業務用AP開発者は少ない。
  - 6：HTML5に置き換えると低速になる。

今後の運用

- スケールアウト
  - データ数が増加したとしても単体KVS性能はほとんど変わらない。
  - ユーザー数が増加したとしても単体KVS性能はほとんど変わらない。
  - 性能低下時には KVS サーバーを追加すればよい。
  - →AP側で接続先に対する制御が必要（実装済み）
- 冗長化
  - 新システムではサーバー2台のみの運用であった（予算上の制限）
  - KVSサーバー、RDBともに冗長化が必要。
- クラウド化
  - KVS = クラウドの基盤技術。
  - →KVS 向けのシステムはクラウドにそのまま乗せやすい。

まとめ

- KVS を利用したシステムは高速。
- Flash を利用することにより通信も高速になる。
- KVS と RDB の運用が現実的。
  （KVS + RDB の併用は全世界的に流行中）
- KVS 、クラウドのアーキテクチャ、データモデルに精通する必要がある。
- KVS サーバー製品は多種存在。
  （それぞれ得意、不得意がある）
- クラウドAPに変更しやすい。
- 逆に言えばクラウドAPを高速化したい場合、KVS 利用に変更。