Web三層モデル(2)

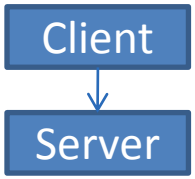
負荷とスケールアウト、アプリケーションの改善について

アジェンダ

1. 負荷の傾向分析
    1. Read負荷
    2. Write負荷
2. 負荷とスケーラビリティ
    1. Grainderについて
    2. スケールアウトの効果を実習する
3. よく知られた負荷対策
    1. 処理の効率化
    2. ステートレス
    3. キャッシュ
4. Teracottaについて
    1. 仕組みの概要
    2. インストールから効果の測定までを実習する
5. その他の負荷対策
6. CAP定理
7. Read/Write分散

Webシステムの負荷

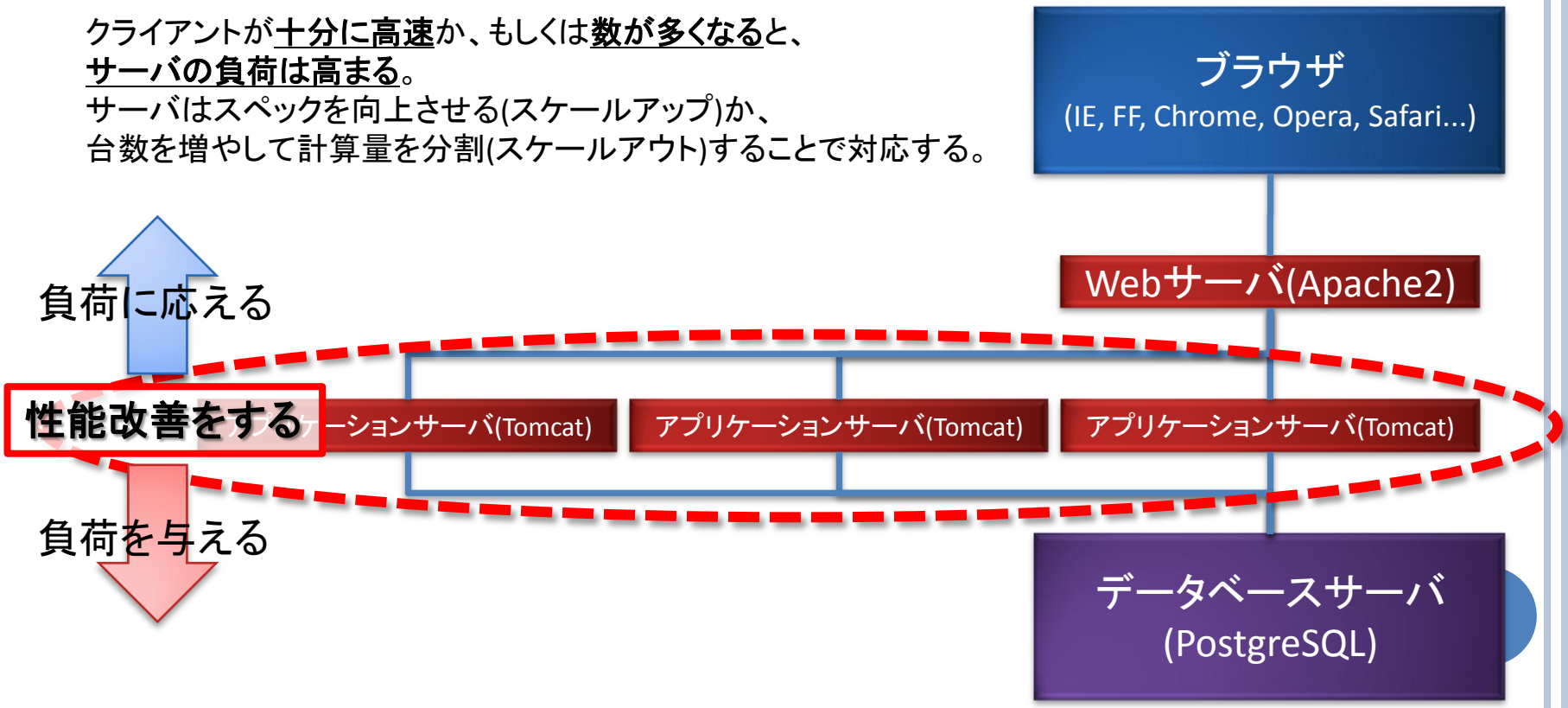
負荷とスケールアウト



負荷とは、クライアントからサーバへ処理を要求する計算量の総和の事。
サーバの性能は、そうして与えられた計算量をいかに高速に処理できるかを示すもの。

Webサーバから見れば、ブラウザがクライアントであるように、
データベースから見れば、アプリケーションサーバがクライアントになる。

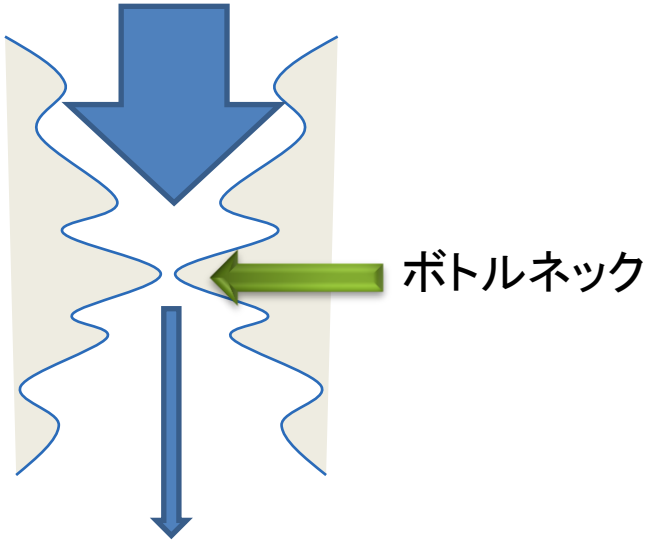
クライアントが**十分に高速**か、もしくは**数が多くなる**と、
サーバの負荷は高まる。
サーバはスペックを向上させる(スケールアップ)か、
台数を増やして計算量を分割(スケールアウト)することで対応する。



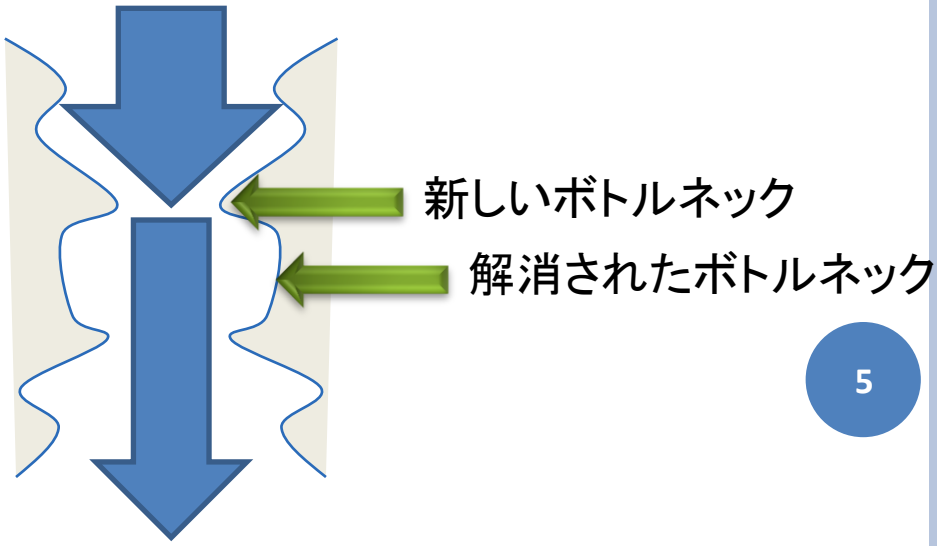
ボトルネックとボトルネックシフト

要求は砂時計のように流れて行き、
各ポイントで様々な処理を実行することで、
1リクエストの処理時間が決まる

サーバで処理できる量は、
最も遅いポイントに影響される



ボトルネックを解消すると
別のポイントが新たにボトルネックになる



READ負荷とは

- データの更新を必要としないクライアントからの要求で生じる負荷全般のこと
- 例
  - 掲示板の記事を閲覧する
    - 書き込みの無い間は常に同じ画面を見せていても良い
- 特徴
  - データが書き換わらない事が分かっていれば、DBに参照に行ったり、計算しなおしたりせずとも常に同じ値であることに期待できる。場合によっては常に同じ画面になる。
  - キャッシュと非常に相性が良い

WRITE負荷とは

- データの更新が必要なクライアントからの要求で生じる負荷全般のこと
- 例
  - 掲示板での発言
    - 誰かが発言をすれば、閲覧している人の結果も変わる
- 特徴
  - データを書き換える処理が動くため、他のクライアントが閲覧しているデータに影響する
  - データが共有されている物である場合は、どの程度の一貫性、可用性が求められるかで厳密さが決まる
    - 一般にこうした厳密さが求められれば求められるほどデータの更新はボトルネックになりやすい

R/W負荷の一般的特性

人間がブラウザを通じて利用する
Webシステムは一般的に
Write要求よりもRead要求が多くなる

データを変更する場合でも、
その操作の判断をするために
画面を閲覧する行為が多くなるため

Read要求によって高まるRead負荷への対処としては、
後述するキャッシュが効くので、比較的楽に対処できる。

Write要求によって高まるWrite負荷を下げるには
かなりの工夫と妥協が必要である。

演習:スケールアウトの効果を確認する

演習の概要

- 【目的】
  - スケールアウトをすると本当に全体の処理量は増えるのか
    を確認する
- 【方法】
  - 負荷生成ツール(Grinder)を用いて、
    サンプルプログラムに負荷(Read/Write)をかける
  - スケールアウトを実施した後、
    同じ負荷をかけて結果の違いを確認する

演習 【当演習参考資料1章】

1. サンプルプログラム「掲示板」に
   Read負荷をかけて測定してみよう
   - 「多くの人が掲示板を閲覧する」シナリオ
2. Write負荷をかけて測定してみよう
   - 「多くの人が掲示板に書き込みをする」シナリオ
3. Read/Write負荷の結果を比較してみよう
   - サーバ台数ごとに、
     ReadとWriteそれぞれの性能をグラフにしてみる

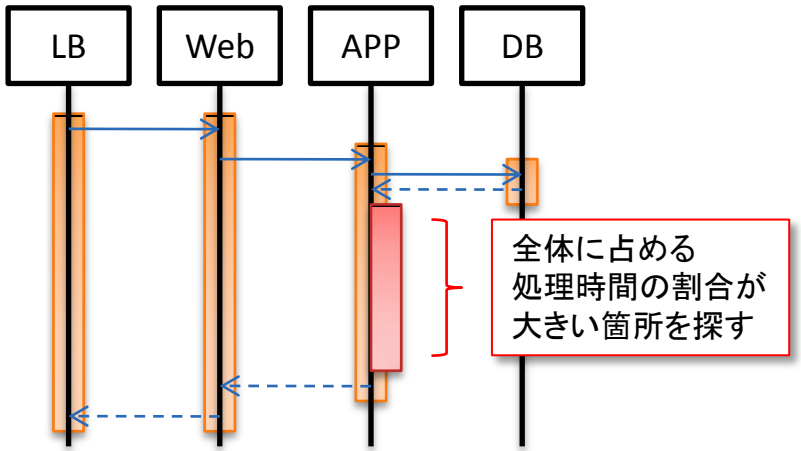
目安となる負荷

		Grinder Agent 7台 100 Threads
Web+App 各1台	Read負荷	TPS
	Write負荷	TPS
Web+App 各3台	Read負荷	TPS
	Write負荷	TPS

TPS = Transaction per second

性能を向上させるための対策

処理効率そのものを向上させる

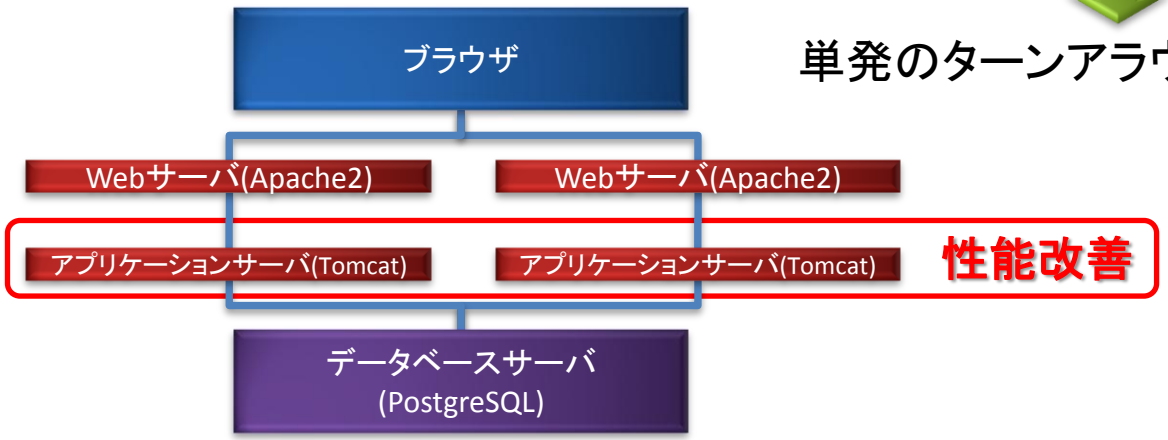


リクエストごとに、
各処理の実行時間などを計測する

一つの処理に時間をかければ
それだけ他の処理ができなくなる

【方法】
- ロジックの無駄をなくす(最適化)
- 使う言語を低級のものにする
  - 例) JavaからC言語を呼ぶ

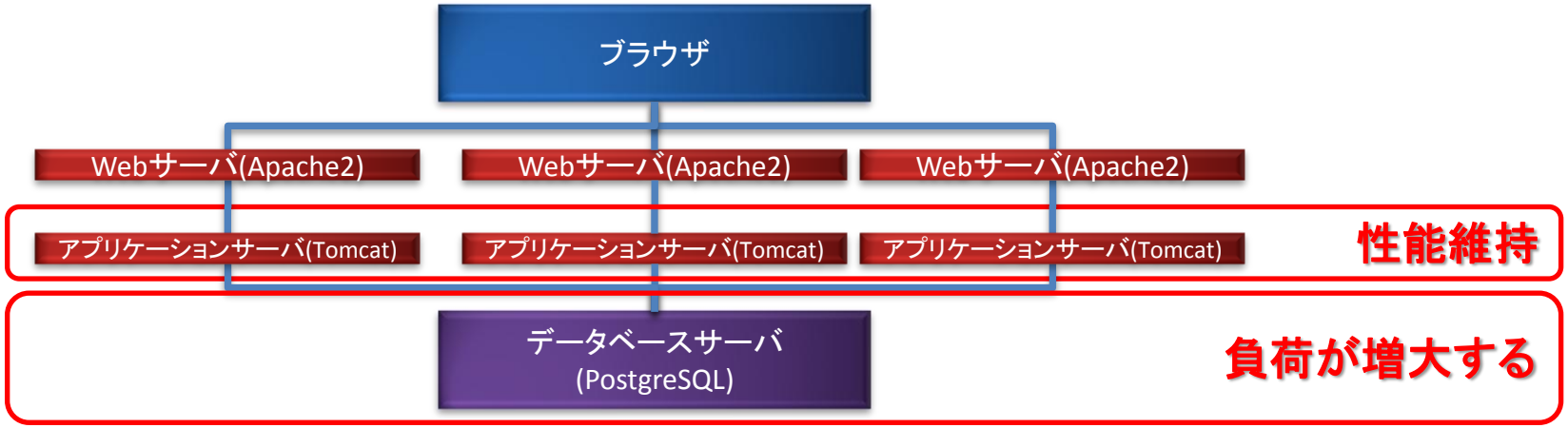
単発のターンアラウンドタイムが決まる



ステートレス化して水平分散を狙う

最初から「ある程度スケールアウトするアプリケーション」の開発が重要である
【参考】RESTアーキテクチャスタイル

ただし、スケールアウトは後段への負荷を向上させてしまう。
下図の場合は、データベースサーバの負荷が高くなり、
ボトルネックはそちらにシフトする。



キャッシュによる
データアクセスの効率化

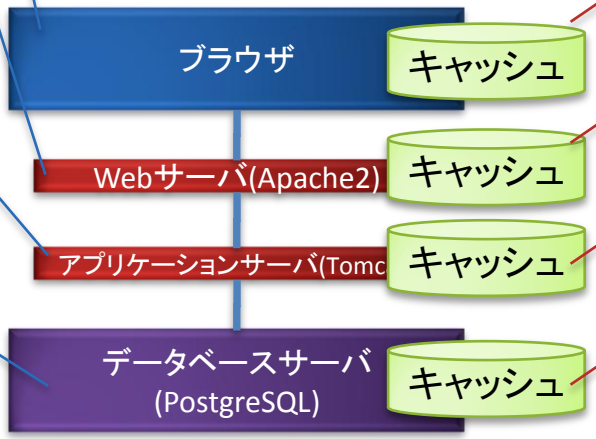
特にRead負荷への対処にはキャッシュが向いている。
データベースまでデータを確認しにいかなくても良くなるため、
要求のトラフィックや処理が減らせる。

キャッシュの処理

- •リロードをしても実質変化しないものは要求を出さない
- •Webサーバからキャッシュして良いと指示されたものを管理する
- •アプリケーションサーバで生成された動的コンテンツのうち、変化が無ければ生成結果だけをキャッシュしておくなど
- •DBへのアクセスをせずとも結果が同じであることが明らかなもの
- •DBサーバの機能としてのキャッシュ（クエリや結果セット）

キャッシュは、以下の考え方で各ポイントに実装できる。

1. 可能な限りクライアントにキャッシュを持たせる
2. キャッシュの保持は高速なI/Oに持たせる

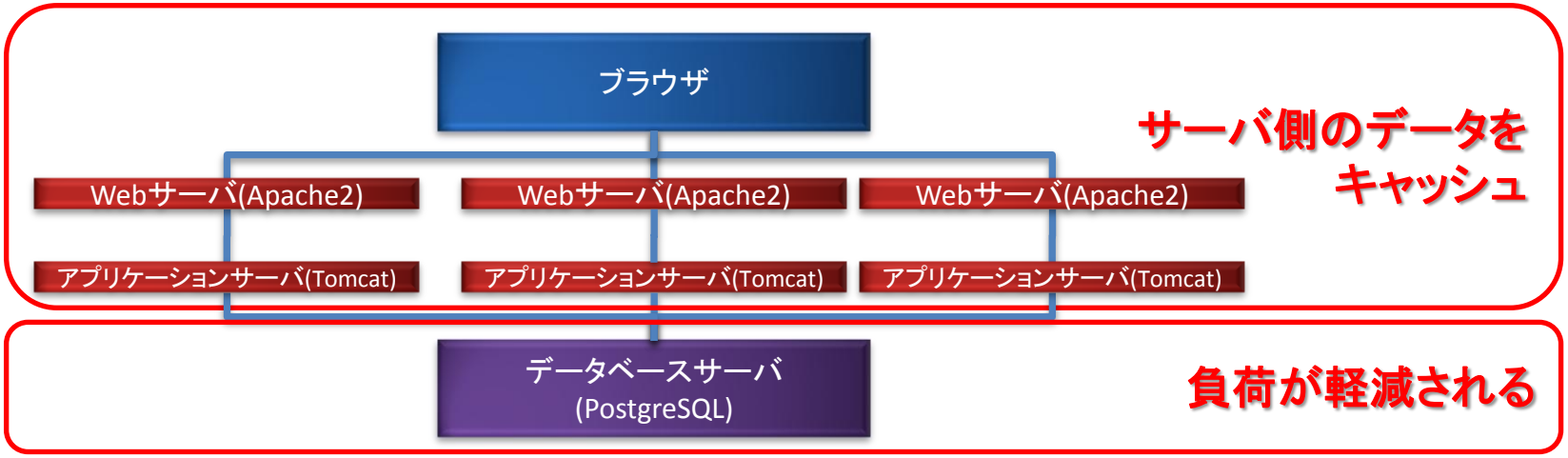


キャッシュされるもの

- コンテンツ全般
- コンテンツ全般
- 変化の無いデータや処理結果
- クエリ

キャッシュはサーバ側への
アクセスを減らす効果がある

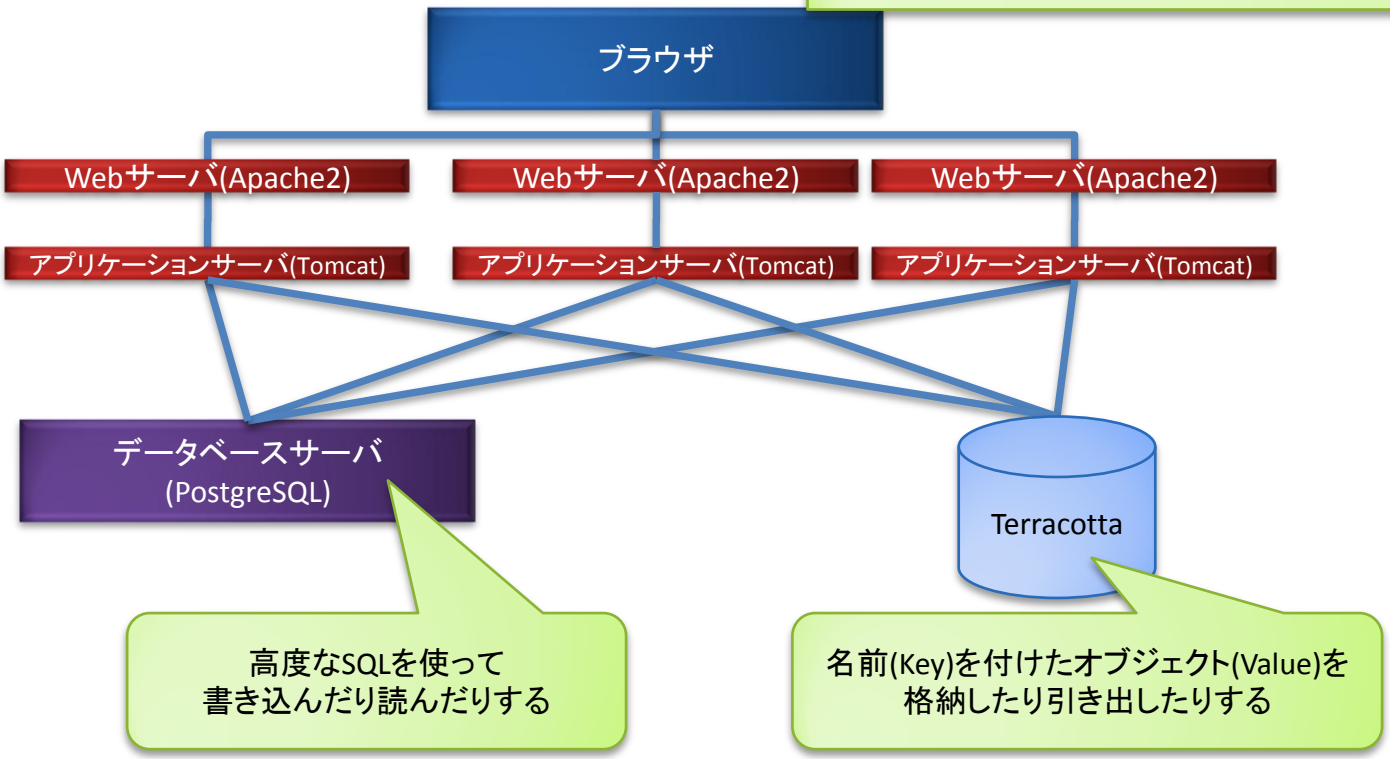
無駄な後段への要求を前段にて極力減らすアプローチ



TERRACOTTAの概要・仕組み
「全サーバに対する共有キャッシュを提供する」

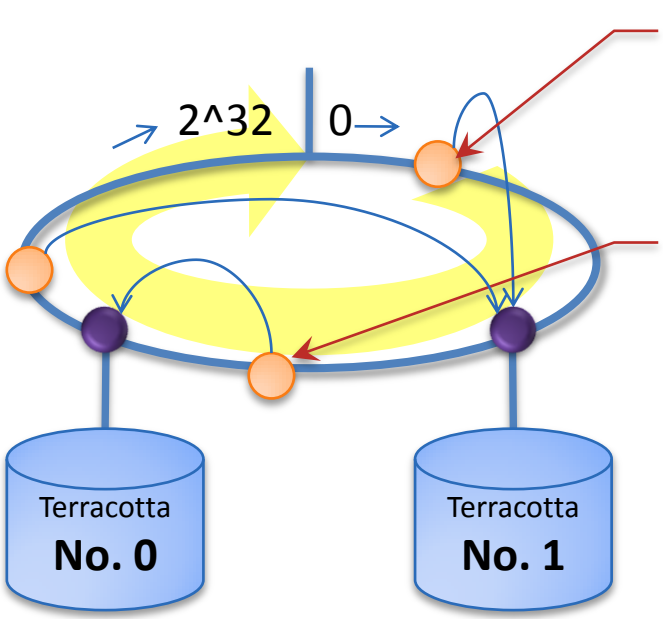
一般的なRDBMSと比較して
「一時的なデータの保存場所」と言う
目的に特化しているのが特徴

- 格納と引き出しのアルゴリズムが単純
  - 名前(Key)とオブジェクト(Value)の組を記憶するだけ
  - 名前(Key)からTerracottaサーバの場所を割り出せれば分散も用意
- キャッシュは消えても問題にはならない
  - ディスクに出力せずメモリだけで処理できる
  - I/Oが低く保てる



TERRACOTTAの分散

名前(Key)によるオブジェクト(Value)の格納場所決定アルゴリズム
Keyから“Consistent Hashing”アルゴリズムでサーバを決定する



Key-1のハッシュ値

Key-2のハッシュ値

Keyのハッシュ値から、
0～2^32の輪の上の位置を算出し、
ぐるっと回って次のTerracottaサーバを格納先とする

新しいTerracottaサーバがこの輪に追加された場合、
それまで格納先として存在していたサーバ分のデータが消える(キャッシュが揮発する)が、
このアルゴリズムにより失われるキャッシュを最小限に留め、格納先を増やすことができる。

演習：キャッシュを導入する

演習の概要

- 【目的】
  - キャッシュが後段の負荷を下げ、
    全体の性能が向上するかを確認する
- 【方法】
  - キャッシュサーバを導入してみる
    【当演習参考資料2章】
    - Terracottaを利用する
    - ※Terracottaのスケールアウトによる性能向上(2章4節以降)は
      時間のある人だけやってみてください
  - その効果を計測する
    【当演習参考資料1章】(再)

演習

1. サンプルプログラム「掲示板」をTerracottaに対応させてみよう
2. Read/Write負荷をかけて結果を比較してみよう
   - サーバ台数ごとにRead/Write負荷を計測する
   - 更にTerracottaの台数を変えて計測する
3. 【議論】
   - Writeの性能をさらに向上させるには何をすれば良いか
   - 自分でも実現できそうなものを考案しよう

目安となる負荷

Terracottaを利用しなかったケースとそれぞれ比較しても
性能向上しているのが確認できる

		Grinder Agent 7台 100 Threads
Web+App 各1台 Terracotta 1台	Read負荷	TPS
	Write負荷	TPS
Web+App 各3台 Terracotta 1台	Read負荷	TPS
	Write負荷	TPS

性能向上

TPS = Transaction per second

課題：議論したWRITE性能の向上案のどれかをプロトタイピングしてみよう

- 提出物
  - 動作するソースコード一式
- 例
  - MySQLの代わりに簡単なKVSを実装してみる

まとめ

クラウド基盤とWEBシステム

- クラウド基盤の効果
  - 【オンデマンド】サーバなどの資源を
    高速にシステム開発者・運用者へ提供することができる
  - 【プログラマブル】Web APIを利用してプログラムを組み、
    業務の自動化をすることができる
- Webシステムの開発と運用が大きく変わる
  - 【開発】スケールアウト後もシステムの性能を維持できるか
    はキャッシュのような技術的工夫が必要である
  - 【運用】スケールアウト・シュリンクインで
    システムの規模をいつでも素早く変化させることができる