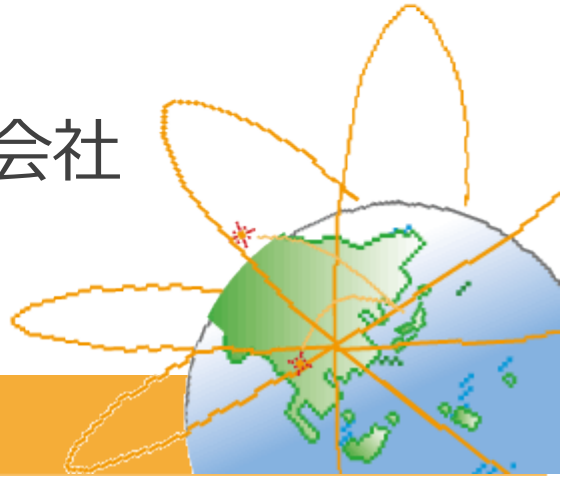
TC-01

ビッグデータだけじゃない！ Amazon DynamoDBの活用事例

CassandraからDynamoDBへの移行で見えたその特徴

2014.07.17

サイバーエリアリサーチ株式会社
中西　健



自己紹介

中西　健（なかにし　けん）

- ken@arearesearch.co.jp

サイバーエリアリサーチ株式会社

- 「どこどこJP」サービス担当データベースエンジニア

最近気になっているAWSサービス

- Amazon Kinesis

アジェンダ

Agenda

弊社サービスのご紹介
Cassandraシステム運用時の課題
DynamoDBについて
実装とチューニング
コスト比較
まとめ

弊社サービスのご紹介

サイバーエリアリサーチ株式会社

アクセスユーザーの地域認識技術であるIPジオロケーションデータベース「SURFPOINT」を提供する、国内オンリーワン企業

- http://www.arearesearch.co.jp/

IPジオロケーション技術

- インターネットに接続されたコンピューター等に割り当てられたIPアドレスから、アクセスユーザーの位置情報やインターネット接続環境を認識・マッピングするための技術。
- インターネット広告配信・コンテンツ配信制御・アクセス解析
- 銀行・証券取引におけるオンラインセキュリティ分野での利用も注目を集めています。

「どこどこJP」サービス

IPジオロケーション情報を提供するWebAPI

- IPアドレスに「位置情報・組織情報・回線情報・気象情報」など最大74種類の属性を紐付け
- 出力フォーマット：XML、JSON、JavaScript
- 用途：コンテンツ配信制御、アクセス解析、不正検出

バックエンドに「SURFPOINT」データ

- 全世界分のIPv4アドレスに対応
- **１IPアドレス単位で属性情報の修正に対応**
  １IP＝１レコード　でデータを保持しておきたい

「どこどこJP」サービス

http://api.docodoco.jp/v4/search?
key1=*APIキー1*&key2=*APIキー2*&ipaddr=*IPアドレス*

```
<?xml version="1.0" encoding="UTF-8"?>
- <docodoco>
    <IP>210.251.250.30</IP>
    <DomainName>arearesearch.co.jp</DomainName>
    <DomainType>.co.jp</DomainType>
    <LineCode>f</LineCode>
    <LineJName>フレッツISDN</LineJName>
    <LineCF>95</LineCF>
    <ProxyFlag>0</ProxyFlag>
    <TimeZone>+0900</TimeZone>
    <ContinentCode>3</ContinentCode>
    <CountryCode>JP</CountryCode>
    <CountryAName>Japan</CountryAName>
    <CountryJName>日本</CountryJName>
    <RegionCode>04</RegionCode>
    <PrefCode>23</PrefCode>
```

Cassandraシステム運用時の課題

旧Cassandraシステムの構成

Cassandraとは

- NoSQLのオープンソース製品
- USではメジャーな製品で、スケーラビリティと高可用性が特徴

構成

- オンプレ 6ノード、各ノード2TBx4のRAID0
- **42億レコードのIPアドレス属性情報**
- データベース中のデータ量は実効1.5TB程度

旧Cassandraシステムの課題

高負荷 / イマイチな安定性

- **ガベージコレクションのメモリ負荷&ディスク負荷**
- 結果、データ取得をミスする / ノードがDownする
- テーブル設計にも問題があった

焼津IDC~お客様システム間のレイテンシへの懸念

今後のシステム拡張におけるコスト面への懸念

× 別のIDCにCassandraシステムのコピーを構築

→ **解決にならない**

解決策

SURFPOINT™

amazon web services

速度面・安定性を重視したい

多くのアドテク界隈のお取引先が
すでにAWS環境上にシステム構築

でもDBはどうしたらいいだろう？

– CassandraをEC2上で構築する？　→　解決にならない

「DynamoDB, あります」

AWS

DynamoDB

レコード数40億以上なんですけど…

「へっちゃらです」

データ量500GB位になりそうなんですけど…

「へっちゃらです」

レスポンスが遅いと怒られるんですけど…

「問題ありません」

DynamoDBについて

…公式プレゼンから抜粋して
ざっくりとご紹介

DynamoDBの特徴

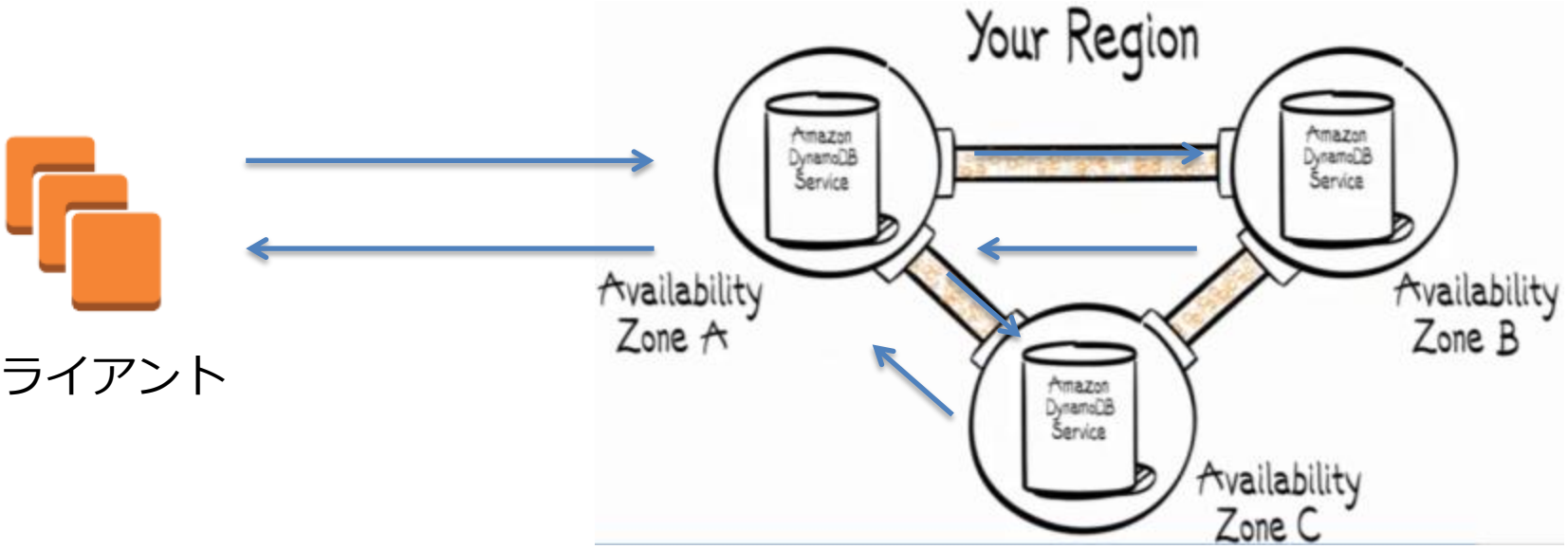
“NoSQL as a Service”

- 管理不要で信頼性が高い
- プロビジョンドスループット
- ストレージの容量制限がない

※「AWS Black Belt Techシリーズ Amazon DynamoDB」等より

特長１：管理不要で信頼性が高い

SPOFの存在しない構成
データは3箇所のAZに保存されるので信頼性が高い
ストレージは必要に応じて自動的にパーティショニングされる



特長２：プロビジョンドスループット

ReadとWrite、それぞれに対して必要な分だけのスループットキャパシティをプロビジョンする（割り当てる）ことができる

一般的なReadヘビーなDBなら

- Read：1,000、Write：100

WriteヘビーなDBなら

- Read：500、Write：500

この値はDB運用中にオンラインで変更可能

– ただし、スケールダウンに関しては日に４回までしかできない

特長３：ストレージの容量制限がない

使った分だけの従量課金制のストレージ

データ容量が増えてきたのでディスクを足したり、ノードを足したりという作業が不要

データの保存はSSDに対して行われる

DynamoDBのシンプルな料金体系

プロビジョンドスループットで決まる時間料金

- Read/Writeそれぞれプロビジョンしたスループットによって時間あたりの料金がきまる
- 大規模に利用するのであればリザーブドによる割引もあり

ストレージ利用量

- 保存したデータ容量によって決まる月額利用料金
- 計算はGBあたりの単価が適用される

実際の料金については下記を参照
http://aws.amazon.com/jp/dynamodb/pricing/

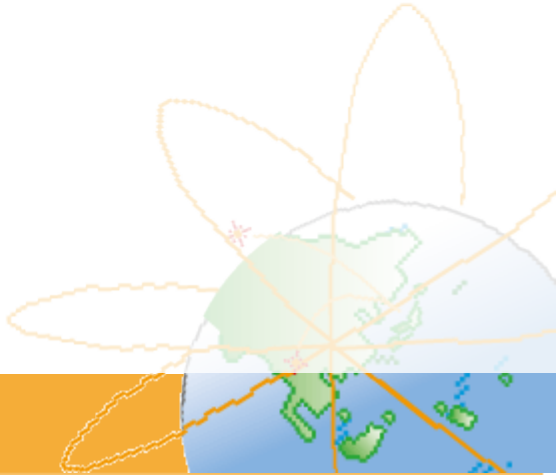
ちょっとブレイク

「ビッグデータ」と「どこどこJP」

「ビッグデータ」との比較

	「ビッグデータ」	どこどこJP
レコード数 データ量	レコード数は巨大 増大し続けるデータ	レコード数は巨大 データ量の変化は小
Write	常に書き込みが発生	定期的・比較的少量
Read	複雑なクエリや分析	クエリは単純 １レコードを返信

DynamoDBへの実装とチューニング



属性情報テーブルの分割

IP情報テーブル

ipv4_dec (IPアドレス十進)	3539728926
geo_hash_key	fcdf5e750a54cdc8829821949daf7668
org_hash_key	39afad0d1d4fb4fbf2c8f66269b5689e
timestamp	1401947252

地域情報テーブル

geo_hash_key	fcdf5e750a54cdc8829821949daf7668
xml	<DomainName>arearesearch.co.jp</DomainName><DomainType>.co.jp</DomainType><LineCode>f</LineCode><LineJName>フレッツ ISDN</LineJName><LineCF>95</LineCF><ProxyFlag>0</ProxyFlag>…

組織情報テーブル

org_hash_key	39afad0d1d4fb4fbf2c8f66269b5689e
xml	<OrgCode>100002637</OrgCode><OrgOfficeCode>0</OrgOfficeCode><OrgName>サイバーエリアリサーチ株式会社</OrgName><OrgPrefCode>22</OrgPrefCode><OrgCityCode>22206</OrgCityCode><OrgZipCode>411-0036</OrgZipCode><OrgAddress>静岡県三島市一番町18-22アーサーファーストビル4F</OrgAddress>…

- 36億レコード
- 500GB

- 60万レコード
- 500MB

- 120万レコード
- 800MB

チューニング（１）

公式 AWS SDK for PHP version 2 を使用

参考：“Performance Guide”

http://docs.aws.amazon.com/aws-sdk-php/guide/latest/performance.html

- PHPを5.4以上にアップグレード
- PHP5.5を使う / APCのようなOpcodeキャッシュを使う
- Classmap autoloader “Composer” を使う

速度追求部分には非公式PHPライブラリで対応

- **“php-simple-amazon-dynamodb”**

  https://github.com/ttsuruoka/php-simple-amazon-dynamodb

チューニング（２）

1183rps :C1.xlarge, Apache, 非公式ライブラリ

731rps：C1.medium, NginX, 非公式ライブラリ

419rps：C1.xlarge, Apache, 公式SDK, preloader有効

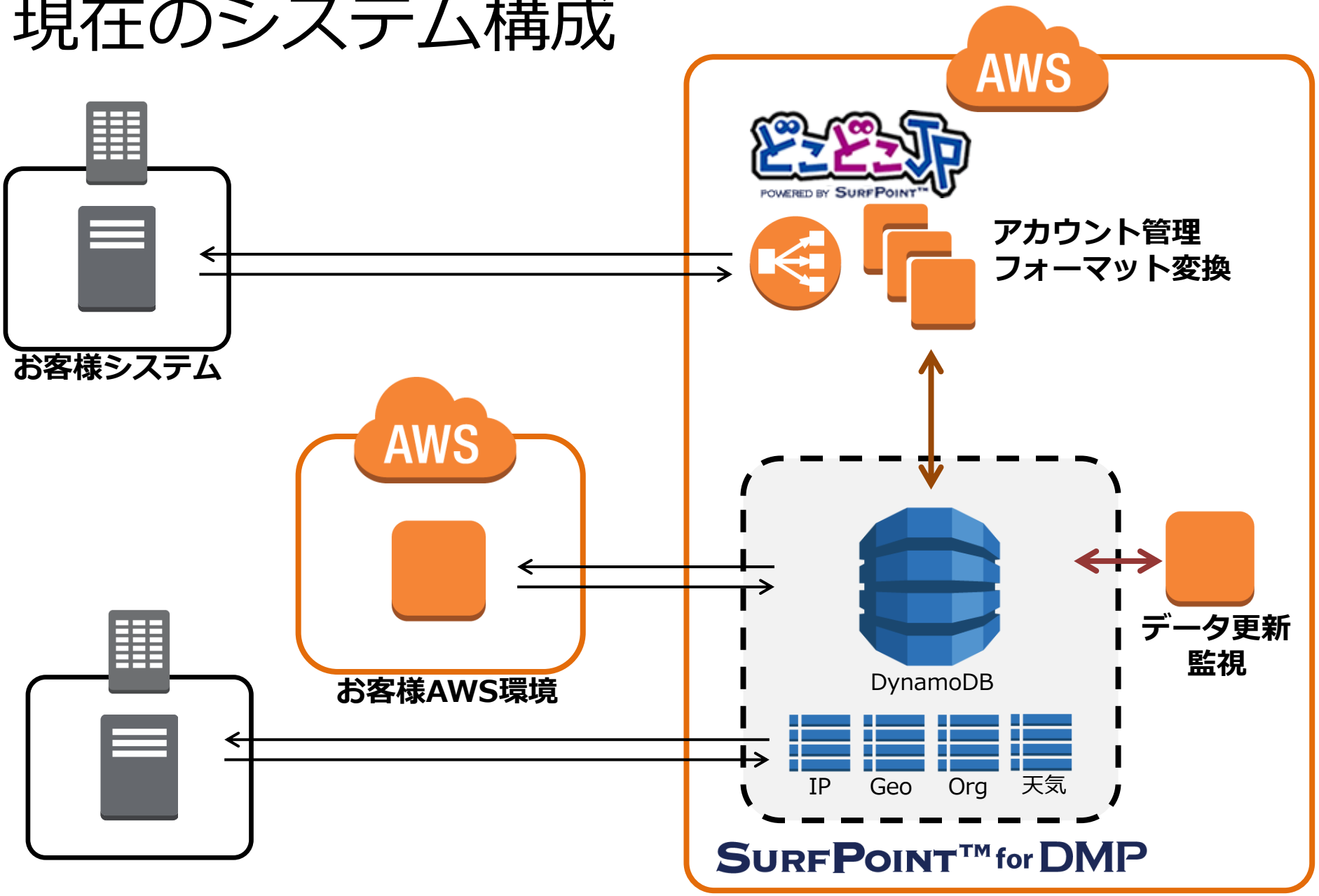
132rps：C1.medium, Apache, 公式SDK, preloader有効

131rps：C1.medium, NginX, 公式SDK, preloader有効

109rps：C1.medium, Apache, 公式SDK, preloader適用前

95rps：C1.medium, NginX, 公式SDK, preloader適用前

現在のシステム構成


AWS
どこどこJP
POWERED BY SURFPOINT™
アカウント管理
フォーマット変換
お客様システム
AWS
お客様AWS環境
DynamoDB
データ更新
監視
IP
Geo
Org
天気
SURFPOINT™ for DMP

DynamoDB：ここが不満

RDBMS的にはよくある処理が一発で完了できない

「全レコード一括削除」「テーブル名変更」

スループット消費し地道に処理するしかない場合も

速い？速くない？

HTTP通信のオーバーヘッドは無視できない

複数並列処理が大前提

処理の得意不得意はあります

キー / ハッシュの設計はよく考えないと！

DynamoDB以外のサービスと結合して実現する手も

アプリケーションの移植負荷は？

移植自体はそこまで手間はかからなかった

テストは少しハマった

- 負荷テストにApacheBenchを使用したが・・・

DynamoDBのスループットを可変にした

- プロビジョンドスループットはできるだけ抑えて**ケチりたい**
- 指定したスループットを超えた場合でもなんとかしたい

DynamoDBの負荷テスト

負荷テストに ab コマンドを使った結果

謎の挙動：最初の数十秒だけ高性能

http://localhost/rest.php?ip=220.216.104.1

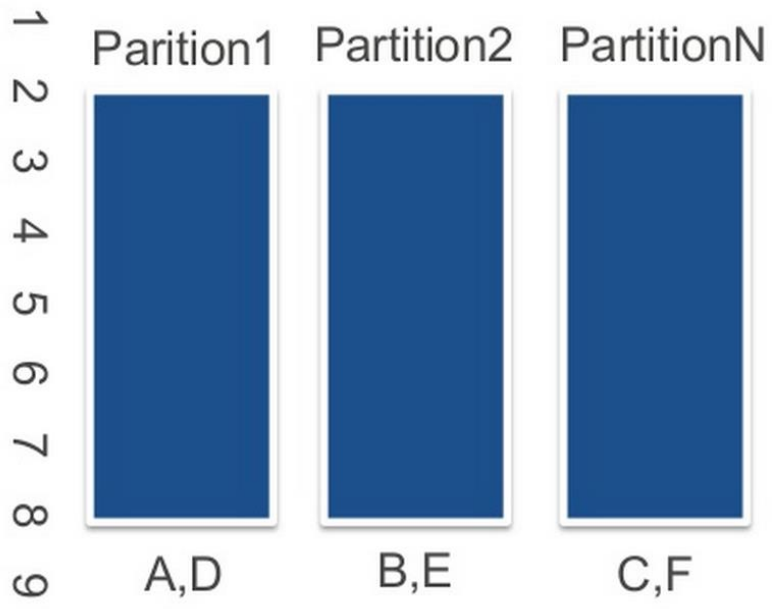
(結果の例)

- 1回目結果 = 1200req/s
- 2回目結果 = 240req/s　**エラー出まくり**
- しばらく時間をおくと、性能が復旧する ??

⇒ DynamoDBの内部データ構造に起因

DynamoDBの内部データ構造

Hash keyとRange keyの概念(2/3)



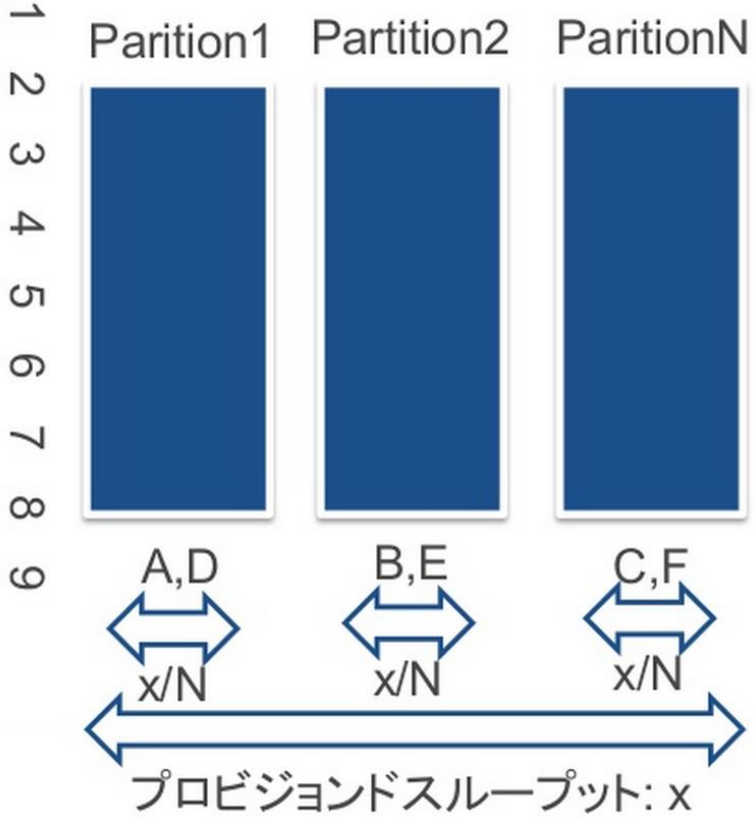
Partition

- DynamoDBはプロビジョンされたスループットを確保するためにデータを複数のPartitionに分散して格納している
- Partitionは格納データ容量やプロビジョンされたスループットによって自動的にリパーティショニングが実施される
- いま現在のPartition数を確認することはできない

※「AWS Black Belt Techシリーズ Amazon DynamoDB」より

DynamoDBの内部データ構造

Hash keyとRange keyの概念(3/3)



スループットはPartitionに均等に付与される

- プロビジョンしたスループットは各パーティションに均等に付与され、全体で合計値の性能が出るように設計されている
- アクセスされるキーに偏りが発生すると思うように性能がでないという自体を招くため、Hash keyの設計には注意が必要

DynamoDBの自動スケーリング

DynamoDBの自動スケーリング

公式SDKを使いCapacityUnitsを読み書き

```
$x = $dynamo->getreadCapacityUnits( $tableName );
$x = $dynamo->getwriteCapacityUnits( $tableName );
$x = $dynamo->getConsumedReadCapacityUnits( $tableName );
$x = $dynamo->getConsumedWriteCapacityUnits( $tableName );
```

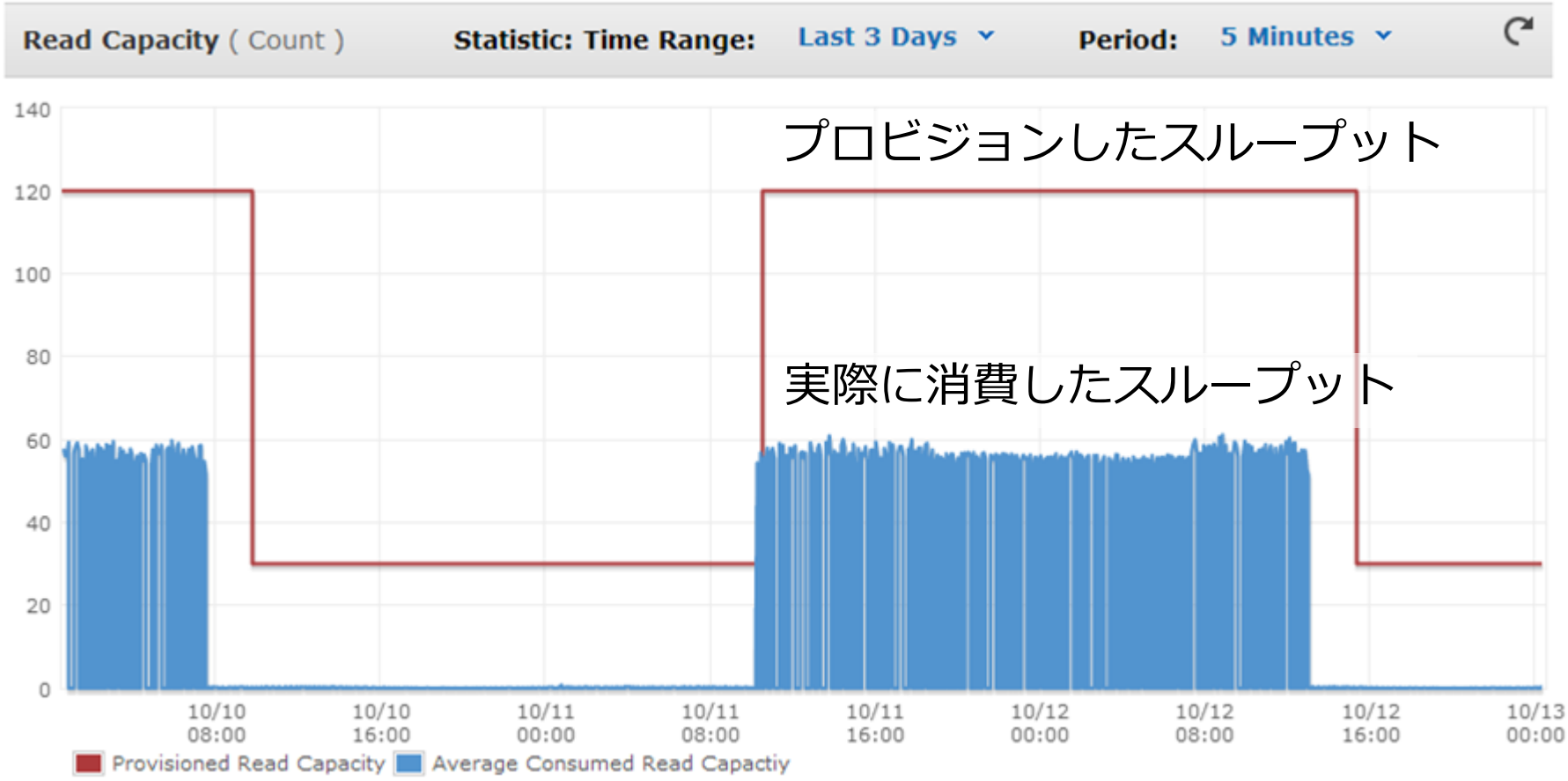
スループットを上げる条件

– 利用率が 80% を超えていたら
⇒ **CapacityUnit を現在の150%に上げる**

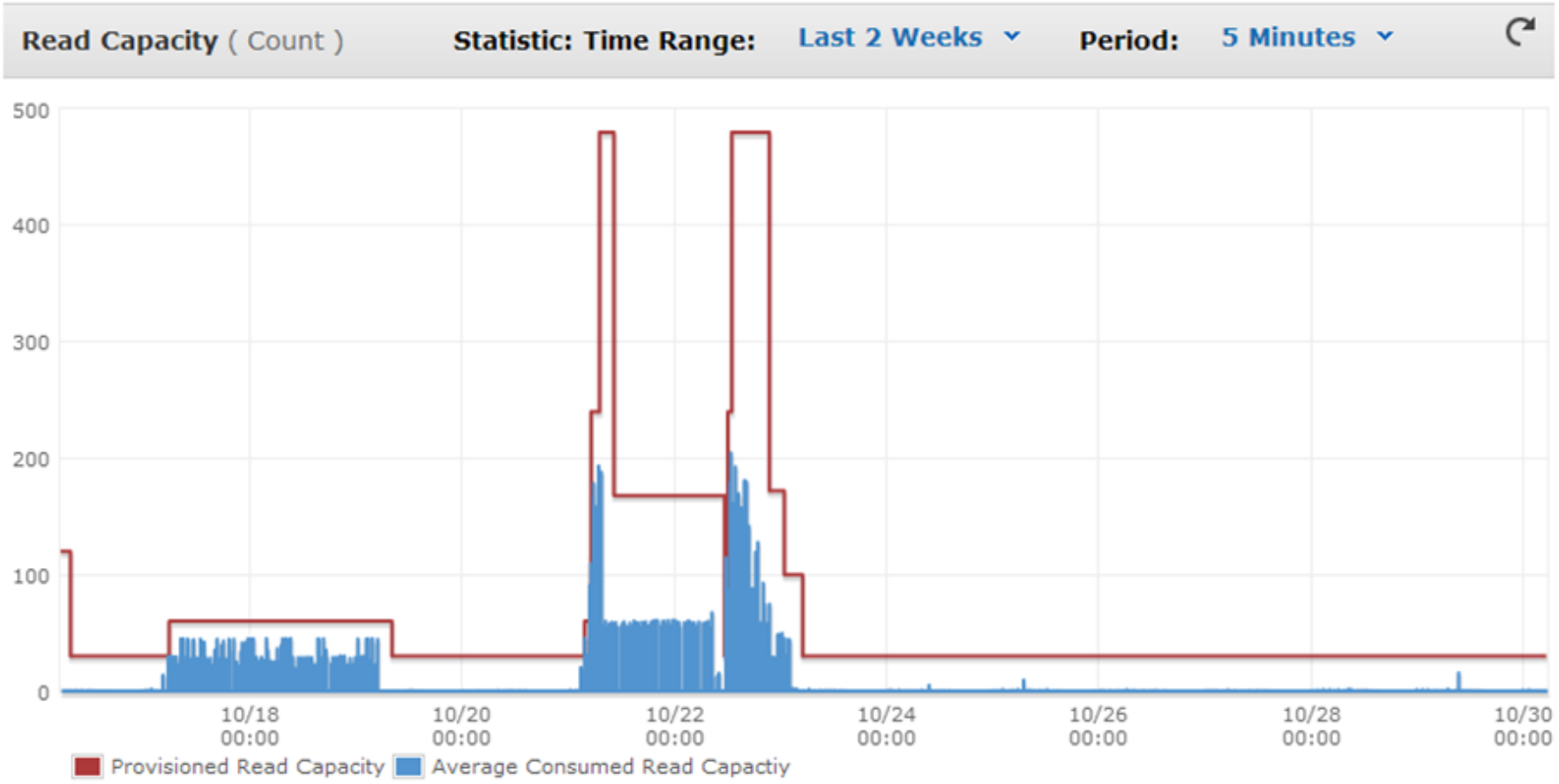
スループットを下げる条件

– 利用率が 25% 以下の状態が2時間(5分間隔×24回)継続
⇒ **CapacityUnit を現在の25%に下げる**

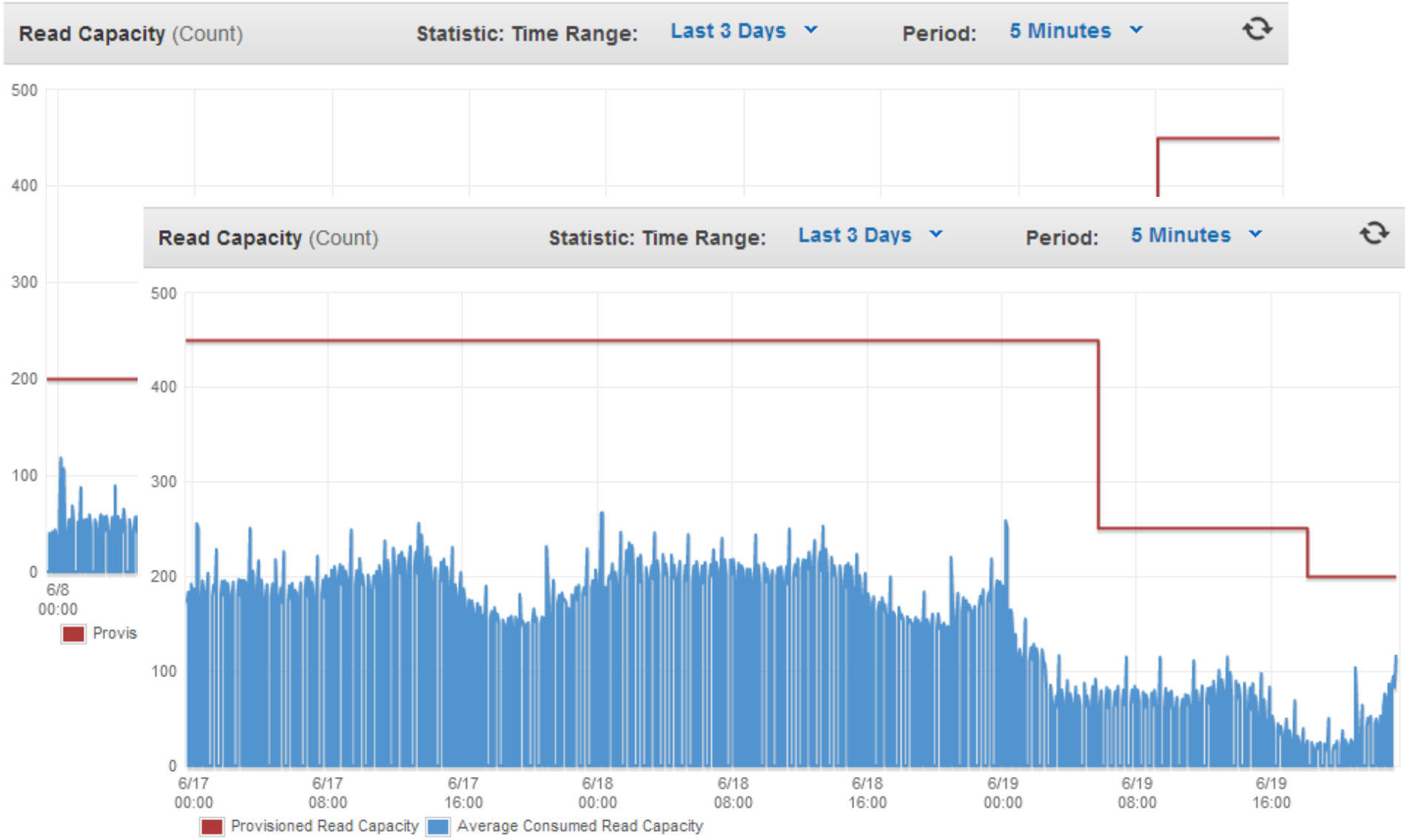
自動スケーリングの実行例



自動スケーリングの実行例

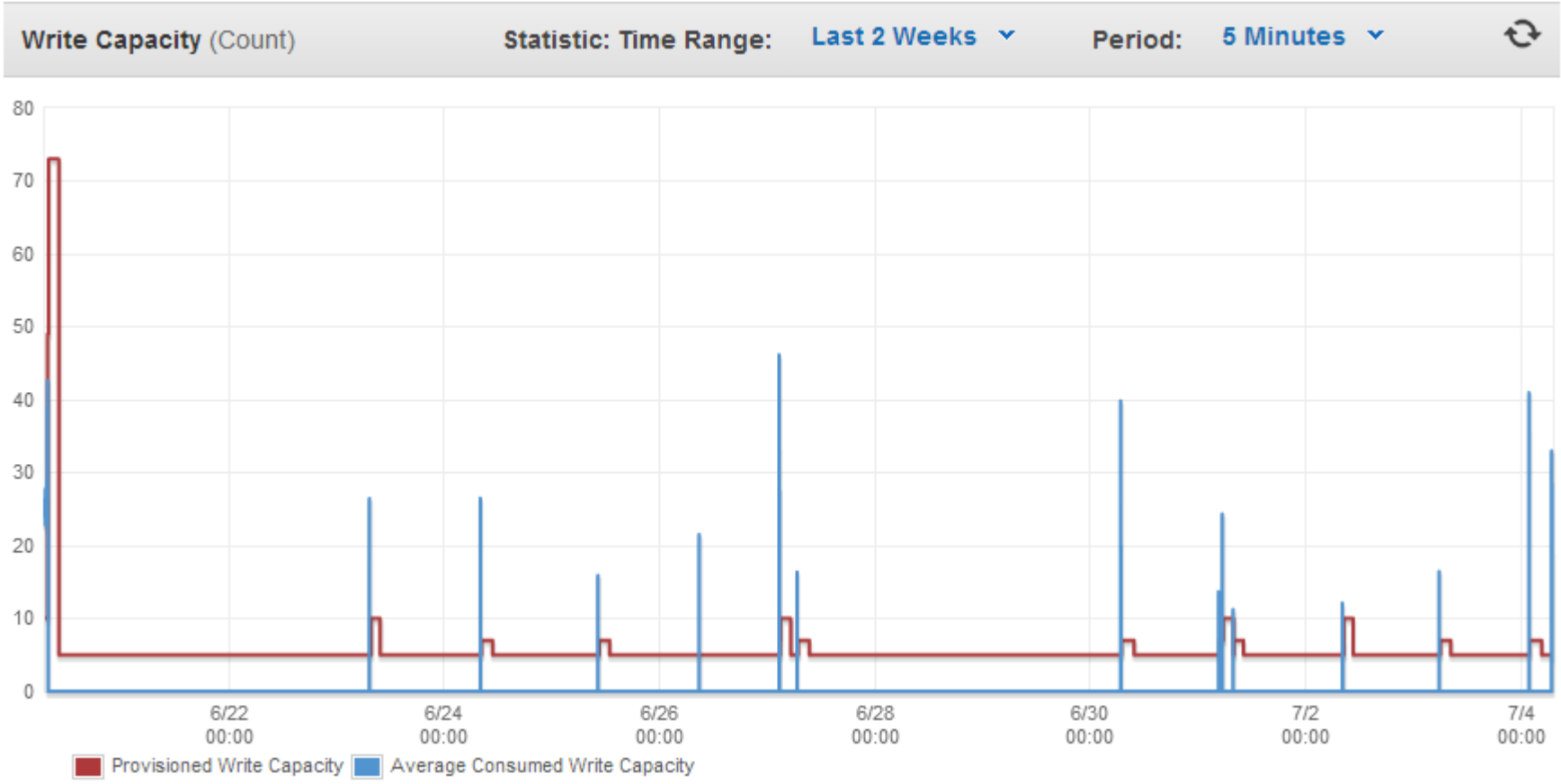


自動スケーリングの実行例



自動スケーリングの実行例

一方、書き込みスループットは



「ビッグデータ」との比較

	「ビッグデータ」	SURFPOINT
レコード数 データ量	レコード数は巨大 増大し続けるデータ	レコード数は巨大 データ量の変化は小
Write	常に書き込みが発生	定期的・比較的少量
Read	複雑なクエリや分析	クエリは単純 １レコードを返信

コスト↓

コスト↓

コスト比較

オンプレミス環境でのシステム構築
vs
AWS環境でのシステム構築

比較

	オンプレミス環境	AWS環境
開発期間	約18カ月	約6カ月
初期費用	約300万円	ほぼ０円
ランニングコスト	月額 約15万円	月額 約15万円

オンプレミス環境でのシステム構築に要したコスト

開発期間：約18カ月

- 本番サーバースペック選定のための実証テスト
- サーバー機の調達 / 200v電源は確保できるのか？

初期費用：約300万円

- Cassandraノード用 x86サーバー 6台
- API提供用 x86サーバー　4台など

ランニングコスト：月額 約15万円

- IDC費用（設置場所＋電源＋ネットワーク）
- ファイヤーウォールやロードバランサーのコストは別

AWS環境でのシステム構築に要したコスト

開発期間：約6カ月

- （注）旧システムから仕様/プログラムの一部を継承

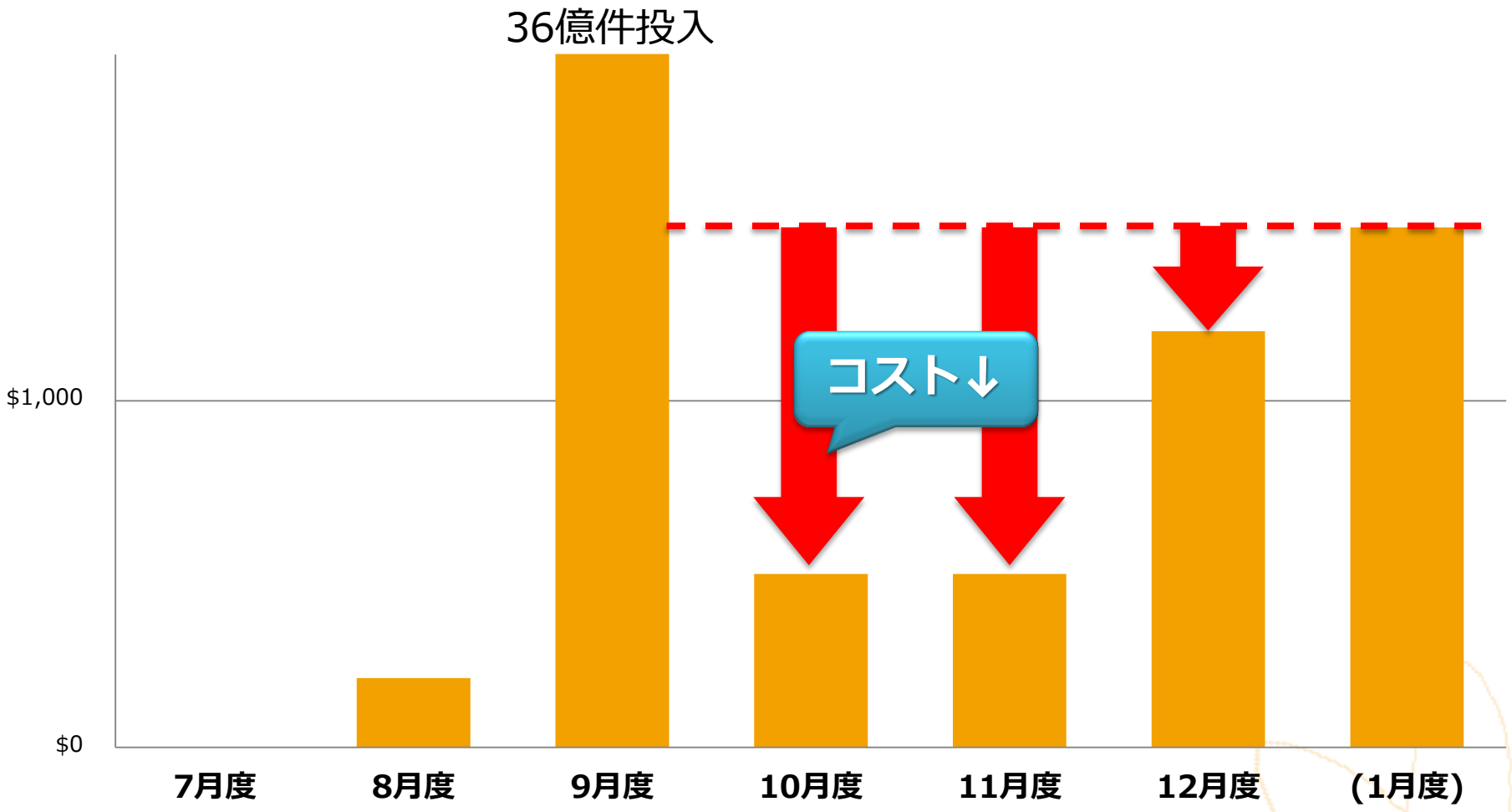
初期費用：ほぼ０円

- AWSサインアップ → 無料利用枠を活用

ランニングコスト：月額 約15万円

- 「どこどこJP」EC2インスタンスやELBの費用を含む
- DynamoDB単体での利用料金は **月額 5万円弱**

AWS環境でのシステム構築に要したコスト



仕様検討 開発 → 実データ投入 負荷テスト → チューニング テスト環境公開 → サービス開始

まとめ

AWSへのシステム移行で悩まなくて済んだこと

本番サーバースペック選定のための実証テスト

- スペック見積や電源確保がいらない
- microインスタンスでいきなりプロトタイプ開発を開始
- 様子を見つつインスタンスタイプを変更

システム構築完了から売上計上まで間のコスト

- 負荷テスト終了後、一旦 ”休眠状態” にさせておいた

インフラ調達に関わるコストやリスク

- 万が一の際(?)にもすぐにサービス撤収ができる安心感

まとめ

オンプレミス環境からAWS環境へ移行

- 要求事項を考慮し DynamoDB を選択
- 時間的・金銭的コストを大幅削減
- ビジネスの成長に伴うインフラ不安からの解放

「DynamoDBを導入したので、休日に安心して外出できるようになりました」

おまけ・・・新しい悩み

「有効活用を考えてよ!」

(^▽^;)

ご静聴ありがとうございました！