AWS Summit Tokyo 2015

Amazon Elastic MapReduce や Spark を中心とした社内の分析環境事例とTips

株式会社サイバード
データ戦略統括部
小松　裕一

CYBIRD®
モバイルでスマイル!

自己紹介

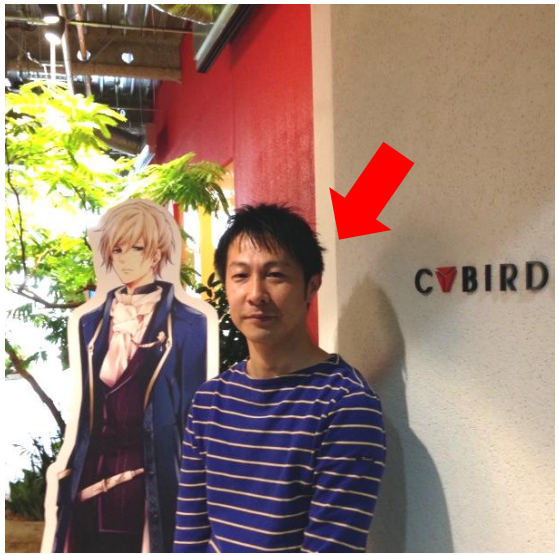
名前：小松　裕一

所属：株式会社サイバード
　　ビジネス戦略統括本部
　　データ戦略統括部

仕事：分析環境の構築
　　データ処理全般

出身：宮城県

好きなAWSサービス：S3



会社紹介

女性向け恋愛ゲーム -イケメンシリーズ-

- イケメン達との恋愛ストーリーを
  楽しめる恋愛ゲーム
- 7タイトル、7プラットフォーム、
  156カ国で配信
- 累計会員数1,100万人

※2014年10月時点

BFB 2015

- 3DCGのリッチな試合シーンが楽しめる本格派サッカークラブ育成ゲーム
- 世界140カ国配信、累計300万DL
- 香港ではAppStoreセールスランキングNo.1獲得

※2015年3月時点

情報コンテンツ

細木数子の占い
「六星占術」を
インターネット上で
唯一提供。

スマートフォンアプリTOP画面

スマートフォンアプリメニュー画面

電子書籍

発行部数世界一としてギネスブックに連続掲載されている細木数子の占い本を、電子書籍として、スマートフォン、フィーチャーフォンでも提供。お手軽価格でお楽しみいただけます

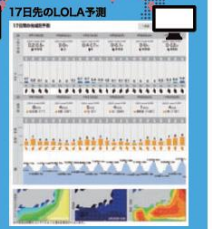
日本で唯一
「LOLA」システムを
採用 |1日最大|
17回更新の波情報

Pray for surf!
NAMI-ARU?

日本全国の波情報配信

約140カ所のサーフポイントの波情報を毎日更新。
日本全国のサーフレポーターが、朝から夕方まで実際に波をチェックし、波情報を鎌倉・七里ケ浜サテライトスタジオに集約。毎日最新の波情報を配信しています。

**世界基準の
波予測ツールを手に入れる**



いつでもどこでも
波情報をチェック！
アラーム機能搭載で
大事な情報逃さない！

LOLA Surfline

2013年3月よりアメリカ・サーフライン社との独占契約。それにより、日本で唯一、LOLAシステムを利用した17日先までの波予測が可能になりました。日本を含む全世界約4000カ所の予測データを提供しています。

なみある? BEACH HOUSE

海の家in逗子海岸！海の素晴らしさを伝えるリアルな発信拠点として、なみある?は、この夏もビーチシーンを盛り上げます。

Agenda

分析環境の概要

データ処理の詳細

現在の検証項目

まとめと今後の展望

Agenda

分析環境の概要

データ処理の詳細

現在の検証項目

まとめと今後の展望

分析環境の概要

- 背景
- 検討事項
- システム構成と用途

分析環境の概要

- **背景**
- 検討事項
- システム構成と用途

きっかけ

・2012年末からモバイルゲームがヒットし始める

⇒ 売上を向上させるべくデータ分析チーム結成

課題

・データの取得が困難

⇒ 重いクエリでサービスに負荷をかけられない

⇒ アップデートされるデータが残っていない

(e.g. 昨日のレベルが分からない)

⇒ タイトルはもちろん、プラットフォームごとにDBが異なる

制約

・限られた予算

・限られた人員

・大量データの処理の必要性

AWSで解決

・限られた予算
　⇒スモールスタートが可能

・限られた人員
　⇒運用が楽

・大量データの処理の必要性
　⇒EMRがある！

分析環境の概要

- 背景
- **検討事項**
- システム構成と用途

検討 1

オンプレミス

メリット

- 処理量が多い場合に費用対効果が高い
- 社内の他システムとの親和性が高い
- 機密情報を扱う場合に承認を得やすい

デメリット

- 初期投資が高額となる
- 運用面での学習コストが高い
- 運用リソースを多く必要とする

検討２

TREASURE DATA

メリット

- サーバ管理を考慮する必要がない
- Fluentdと連動したデータ収集が秀逸
- データを格納するだけですぐに集計が始められる

デメリット（2012年当時）

※デメリットというよりも要件を満たさなかったもの

- RDBのサービスがない
- 機械学習をプラットフォーム上で実行する方法がない
- Hiveでの集計方法しかない

当社の課題解決に対する評価

サービス	コスト（※1）	運用難易度（※2）	拡張性（※3）
オンプレミス	×	×	◎
TREASURE DATA	○	◎	△
AWS	◎	○	○

※1 スモールスタートを前提とした初期コストと月額運用コスト

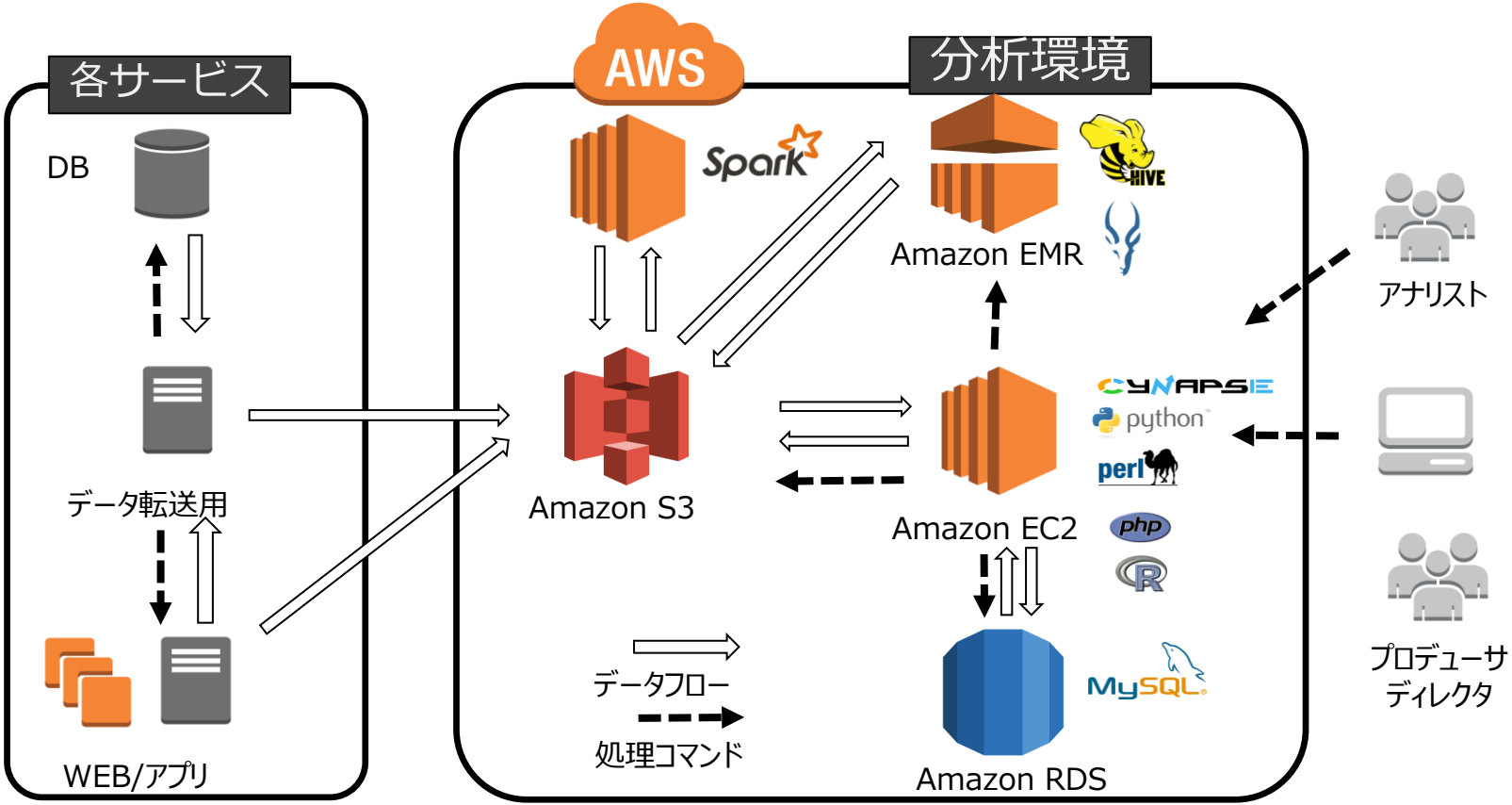
※2 サーバ管理を含めた運用の簡便さ

※3 必要とする機能の充実度、拡張性

分析環境の概要

- 背景
- 検討事項
- **システム構成と用途**

分析環境概要図



分析環境における最重要ポイント

S3にデータを集約

1. ディスクの増設やサーバ拡張などを気にせず使用できる
2. 低コストで利用できる

   ⇒これまで捨てていたデータを貯め続けることができる

3. AWSの各種サービスと統合されている

   ⇒EMRなど分析処理で利用するAWSサービスでのシステム構築が容易になる

分析環境の用途

- **アドホック分析**

  => ユーザごとの行動履歴から知見を得る

  (e.g. 特定のイベントや行動を経験したユーザの課金率は高い)

- **Cynapse（モニタリングツール）**

  => 毎日、KPIなどの指標をモニタリング

  (e.g. ログイン頻度別にDAUを切り分けることで、広告の成果を測る)

アドホック分析事例

- **回遊分析**

  タイトルをまたいでプレイしているユーザの動向を分析

  ＝＞ 改善施策や新規タイトルのアイデア

- **ユーザアンケート分析**

  ユーザのアクション情報をもとにクラスタリングし、

  クラスタごとの単語(N-gram)の頻度を比較

  ＝＞ 新規開発のアイデアや改修の優先順位付け

江戸風タイトル

20%

欧州風タイトル

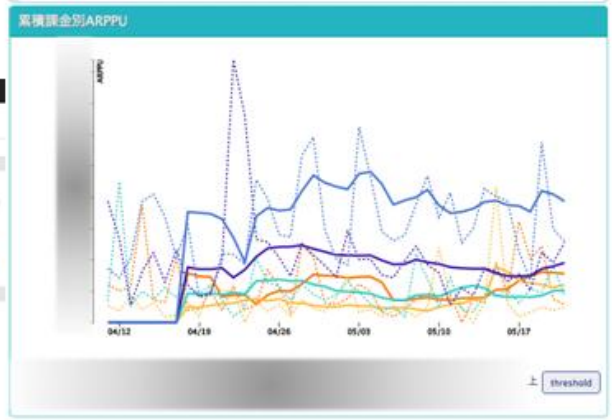
80%

幕末風タイトル

※数値は仮のものです

Cynapse (シナプス)

- 自社開発のモニタリングツール
- データを可視化し、複数のタイトルの状況を複数の指標で確認



Agenda

分析環境の概要

データ処理の詳細

現在の検証項目

まとめと今後の展望

データ処理の詳細

- バッチ処理
- アドホック分析
- Spark利用状況

データ処理の詳細

- **バッチ処理**
- アドホック分析
- Spark利用状況

バッチでのデータ処理フロー

S3にログデータを集約

EMR(Hive)でデータを加工

RDSに処理データを格納

状況に応じてSQL、スクリプトで処理

S3へのデータ格納タイミング

基本的には

前日の**更新**データを**毎日**

分析要件によっては

1時間毎

⇒イベント効果は日単位、もしくは時間単位で
十分に測定することができる

S3への格納データフォーマット

社内で独自に定めたフォーマット

を使用しています

・汎用的なバッチを作成することで新タイトルリリース時の対応がほぼ必要なくなる

⇒リリース直後から分析が開始できる

[よくある問題]
複雑なデータ構造のためKPIを取得する分析実装が間に合わず、施策投入のタイミングを逃してしまう

・必要なデータの所在や形式を特定するのが容易になる

⇒アドホック分析や、業務引き継ぎ等で無駄な作業がなくなる

[よくある問題]
分析要件に応じたデータを探すのが一苦労

データフォーマットの一例（ゲーム事業）

ユーザプロフィール

アイテム

ログイン

ガチャ

課金

項目
共通ID
ユーザID
レベル
課金通貨の保持量
非課金通貨の保持量
国コード
チュートリアルの位置
登録日時
更新日時
タイトルID
プラットフォーム
OSバージョン
機種
推奨機種種別

EMR（Hive）処理詳細

(1)**HiveQLを動的に作成し、S3に格納**

s3://xxxxx-backet/XXX/hql/20150501/retention.hql

(2)**ジョブフローを作成（EMRを起動）**

```
aws emr create-cluster --name "cluster_name" ¥
--ami-version 3.6 ¥
--applications Name=Hive ¥
--auto-terminate ¥
--use-default-roles ¥
--ec2-attributes KeyName=xxxxx ¥
--instance-type m3.xlarge ¥
--instance-count 6 ¥
--steps Type=Hive,Name="Hive Program",ActionOnFailure=CONTINUE,Args=[-f,s3://xxxxx-
    backet/XXX/hql/20150501/retention.hql]
```

(3)**ジョブフローにステップを追加**

```
aws emr add-steps --cluster-id j-XXXXXXXXXXXXX ¥
--steps Type=Hive,Name="Hive Program",Args=[-f,s3:// xxxxx-backet/XXX/hql/20150501/frequency.hql]
```

(4)**処理完了（S3に結果を出力）**

s3://xxxxx-backet/XXX/OUTPUT/retention/20150501/xxxxxxxxxxx

HiveQLファイル（参考）

入力テーブル作成（一部抜粋）

```
CREATE EXTERNAL TABLE IF NOT EXISTS login_log(user_id int,session_id string,create_date string)
PARTITIONED BY (login_dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '¥t' LINES TERMINATED BY '¥n'
STORED AS TEXTFILE LOCATION 's3://xxxxx-backet/XXX/login_log';
ALTER TABLE login_log ADD PARTITION (login_dt='201505') LOCATION 's3://xxxxx-
      backet/XXX/login_log/201505';
ALTER TABLE login_log ADD PARTITION (login_dt='201504') LOCATION 's3://xxxxx-
      backet/XXX/login_log/201504';
・
・
・
```

赤字部分を動的に変更

HiveQLファイル（参考）

出力テーブル作成（一部抜粋）

```
CREATE EXTERNAL TABLE IF NOT EXISTS dau_count (regist_date string,total int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '¥t' LINES TERMINATED BY '¥n'
LOCATION 's3://xxxxx-backet/XXX/dau_count/20150501';
・
・
・
```

赤字部分を動的に変更

HiveQLファイル（参考）

集計及び出力テーブルへの書き出し（一部抜粋）

```
INSERT OVERWRITE TABLE dau_count
SELECT SUBSTR(a.create_date,1,10),COUNT(DISTINCT(a.user_id))
FROM user_profile a JOIN login_log b ON (a.user_id = b.user_id)
WHERE b.create_date LIKE '2015-05-01%'
GROUP BY SUBSTR(a.create_date,1,10);
.
.
.
```

赤字部分を動的に変更

RDSへのデータ格納

処理時間を予測して、ある程度のバッファを見ながらCron設定

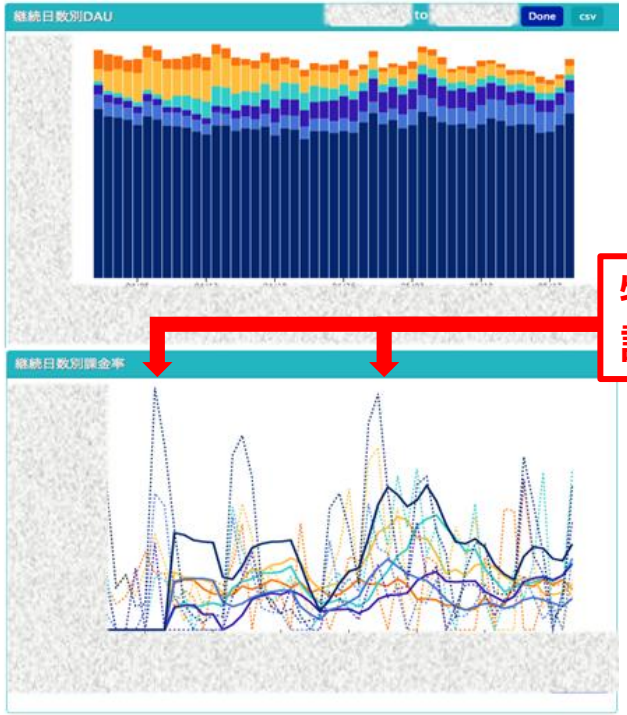
⇒script-runner、Lambda

データから得られる指標の一例

ユーザセグメント毎に**DAU**、**課金率**、**ARPPU**を取得

- レベル
- 累積課金
- ログイン頻度
- 継続日数

など



特定のセグメントの課金率が上昇している

RDSへの格納フォーマット

基本KPI

継続日数別KPI

レベル別KPI

ログイン頻度別KPI

累積課金別KPI

カラム
集計日　PRI
登録日　PRI
DAU
課金UU
売上
コンテンツコード　PRI
プラットフォーム　PRI
推奨端末　PRI

データ処理の詳細

- バッチ処理
- **アドホック分析**
- Spark利用状況

アドホック分析でのデータ処理

・Hive

⇒1度の処理で目的を達成できる場合

※S3のデータをそのまま入力データとして実行

・Impala(on EMR)

⇒高速で処理できるため複数回のクエリを実行して

試しながら処理する場合

※S3からEMR上のHDFSにデータをコピーして実行

データ処理の詳細

- バッチ処理
- アドホック分析
- **Spark利用状況**

Spark利用状況

目的	プロダクト
バッチ処理	Hive
アドホック分析	Hive
	Impala

Sparkとは

一言でいえば、

オンメモリで**高速**処理が可能な

分散処理フレームワーク

Sparkの特徴

- イミュータブル（不変）なRDDを基本としたデータ構造で、RDDを変換しながら処理
- オンメモリでの処理ができるため、繰り返し処理は高速
- HDFSやS3など様々なデータソースからRDDの作成が可能
- Java、Python、Scalaで記述可能
- Spark Coreを中心としたコンポーネント群（Spark SQL、MLlib、Spark Streaming、GraphX）を利用可能

使い勝手の良さ

- トライアンドエラーを繰り返しても苦にならない速さ
- S3のデータをHDFSにコピーせずそのまま利用できる手軽さ
- spark-shell、pysparkで気軽に試せる簡便さ

バッチ処理のフロー

- S3のファイルを読み込み、SparkSQLのテーブルを作成
  (繰り返し使うログイン・課金テーブルなどはメモリにキャッシュをし高速化)
- テンプレートSQLを、集計対象に合わせて書き換え実行
- RDSに処理済データを格納

※ Hiveの場合よりも、コスト1/6、処理時間1/3になっている。
(Hive: m3.xlarge × 3, Spark: m3.large × 2)

アドホック分析

iPythonNotebook上で、
PySparkを使っての分析が可能

jupyter ROCサンプル Last Checkpoint: Yesterday at 11:54 AM (autosaved)

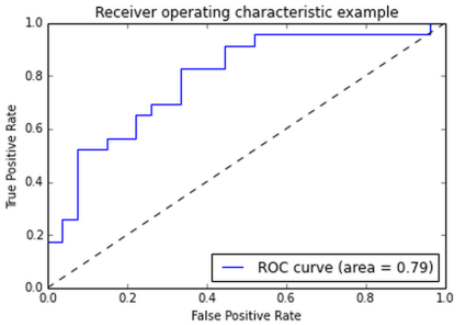
File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

```
roc_auc = auc(fpr, tpr)
print "Area under the ROC curve : %f" % roc_auc
```

```
Area under the ROC curve : 0.793881
```

In [11]:
```
# Plot ROC curve
pl.clf()
pl.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
pl.plot([0, 1], [0, 1], 'k--')
pl.xlim([0.0, 1.0])
pl.ylim([0.0, 1.0])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.title('Receiver operating characteristic example')
pl.legend(loc="lower right")
pl.show()
```



EC2へのSpark導入とiPythonNotebookの連携　http://goo.gl/gNE4mH

Agenda

分析環境の概要

データ処理の詳細

現在の検証項目

まとめと今後の展望

現在の検証項目

- Spark利用の効率化
- 共通IDをキーとした分析環境の構築
- 機械学習を使ったサービスの最適化

現在の検証項目

- **Spark利用の効率化**
- 共通IDをキーとした分析環境の構築
- 機械学習を使ったサービスの最適化

Hive on Spark

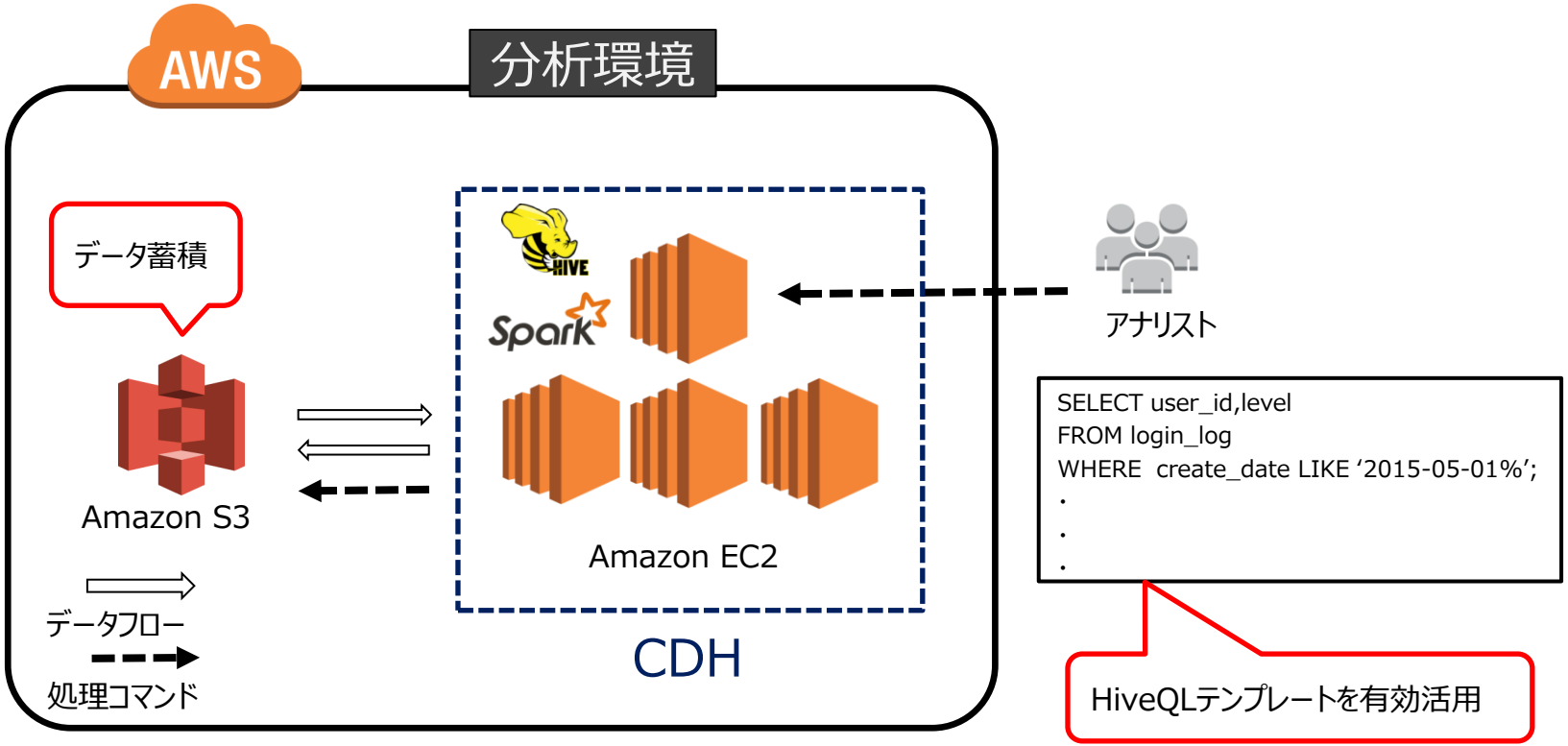
[概要]

Hiveの実行エンジンをSparkとするもの

[目的]

Hiveリソースを有効活用し、より簡易的かつ高速で集計を可能にする

Hive on Sparkの処理フロー



CDH(5.3.0)で検証（EC2上に構築、m3.large×4）

Percel: http://archive-primary.cloudera.com/cloudera-labs/hive-on-spark/parcels/latest/

[S3のアクセス設定]


```
<property>
  <name>fs.s3n.awsAccessKeyId</name>
  <value>XXXXXXXXXXXXXXXXXXX</value>
</property>

<property>
  <name>fs.s3n.awsSecretAccessKey</name>
  <value>XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX</value>
</property>
```

S3の認証キーを設定

クライアント設定の展開（設定の反映）を実行後にhiveを起動

[テーブル作成(例)]

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table (id int,level int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '¥t' LINES TERMINATED BY '¥n'
LOCATION 's3n://xxxxx-backet/XXX/20150501/20150501_level';
```

[実行エンジン切り替え]

```
set hive.execution.engine=spark;
※MapReduceは set hive.execution.engine=mr;
```

[テスト]

(条件)

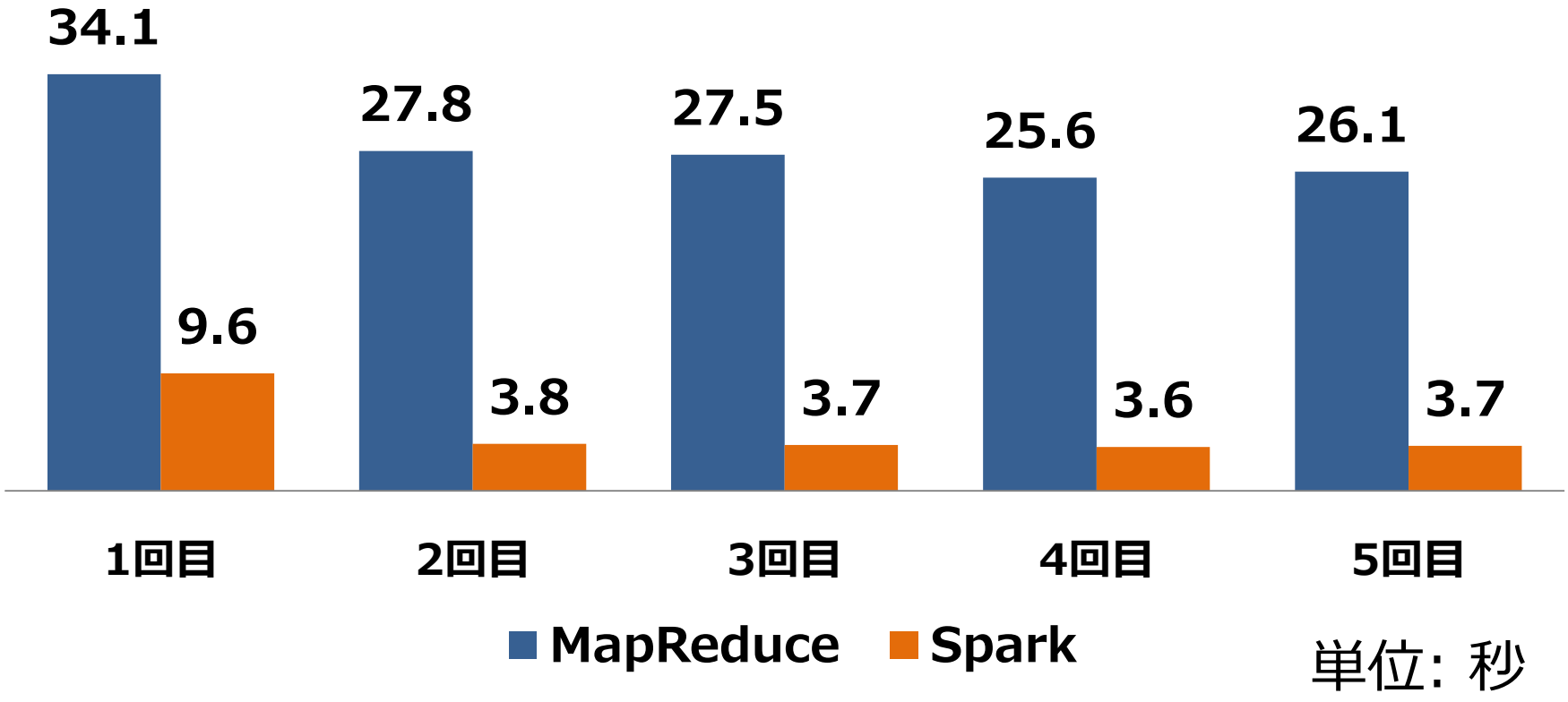
test_table(id,level)：20,723 レコード

test_table_2(id,update_time)：1,023,373 レコード

idでJOINしてid、level、update_timeを20件取得

```
SELECT a.id,a.level,b.update_time
FROM test_table a JOIN test_table_2 b ON (a.id = b.id)
LIMIT 20;
```

処理時間の比較



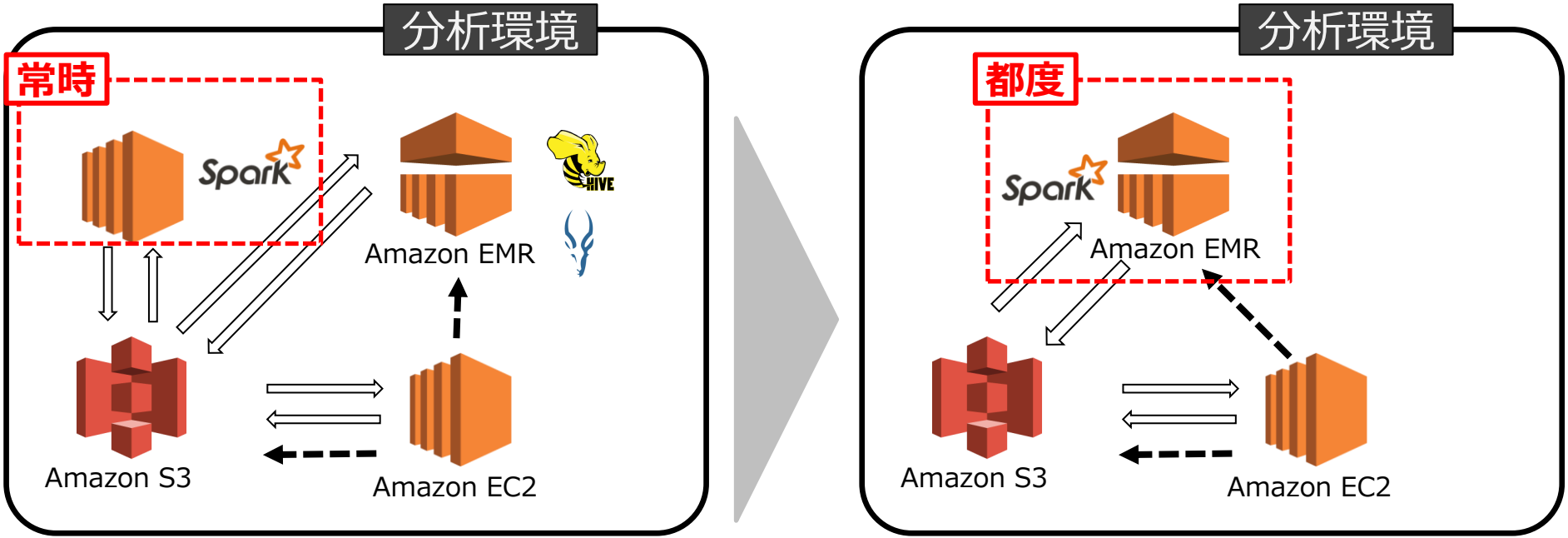
単位: 秒

Spark on EMR

[概要]
　EMR環境でSparkを実行する

[目的]
　必要な時に稼働させることによってコスト圧縮を図る

Spark on EMRの構成図



[EMR起動及びSparkインストールの実行例]

```
aws emr create-cluster ¥
--region ap-northeast-1 ¥
--ami-version 3.6 ¥
--name SparkTestCruster ¥
--no-auto-terminate ¥
--service-role EMR_DefaultRole ¥
--instance-groups
      InstanceCount=1,Name=sparkmaster,InstanceGroupType=MASTER,InstanceType=m3.large ¥
--ec2-attributes InstanceProfile=EMR_EC2_DefaultRole,KeyName=xxxxx ¥
--applications Name=HIVE ¥
--bootstrap-actions Path=s3://support.elasticmapreduce/spark/install-spark
```

[SSHログイン]

```
aws emr ssh --cluster-id j-XXXXXXXXXXX --key-pair-file ~/xxxxx.pem --region ap-northeast-1
```

[spark-shell起動]

```
./spark/bin/spark-shell
```

[動作テスト]

```
> var data = sc.textFile("s3://xxxxxx-backet/XXX/login/20150501/*.tsv")
> data.cache()
> data.count()
INFO scheduler.DAGScheduler: Stage 1 (count at <console>:24) finished in 2.180 s
INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
INFO scheduler.DAGScheduler: Job 1 finished: count at <console>:24, took 2.190953 s
res4: Long = 1023373

> data.count()
INFO scheduler.DAGScheduler: Stage 2 (count at <console>:24) finished in 0.071 s
INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
INFO scheduler.DAGScheduler: Job 2 finished: count at <console>:24, took 0.090554 s
res5: Long = 1023373
```

現在の検証項目

- Spark利用の効率化
- **共通IDをキーとした分析環境の構築**
- 機械学習を使ったサービスの最適化

共通ID

- ユーザ毎にユニークな管理IDを発行
- 共通IDと紐付く形でクライアント上からイベントログを取得

共通IDで実現できる分析

・タイトル間での横断的な分析

⇒プロモーションの参考指標

・より詳細な行動分析

（DB、アクセスログから判別できないもの）

⇒UIの改善、機能の改善

共通IDでの分析実現への課題

- イベント毎に収集される膨大なデータ処理
- 共通IDをキーにアナリストがアドホックかつ多角的に
  分析できる環境の構築

Redshift

現在の検証項目

- Spark利用の効率化
- 共通IDをキーとした分析環境の構築
- **機械学習を使ったサービスの最適化**

機械学習によるユーザの離脱(※)予測を行うことで、

離脱すると予測されたユーザに対して施策を行うことが可能に

※離脱の定義は、ゲームによって異なる

- 現状のステータス
  - レベル、アバター所持数　等
- アクションログ
  - 前日に使用したアイテム量、直近5日間のログイン頻度、継続日数　等

離脱と継続の2値分類であるためモデルにはSVM(※)を採用

※教師データが離脱者よりも継続者のほうが圧倒的に多い不均衡データであるため、
アンダーサンプリングを行い、さらにグリッドサーチを用いてパラメータを最適化

ユーザ毎に継続・離脱フラグがつけられる (再現率(※)は80%)

※離脱を防ぐのが目的のため適合率ではなく再現率を使用

Agenda

分析環境の概要

データ処理の詳細

現在の検証項目

まとめと今後の展望

まとめ

- 分析環境の概要
  - S3、EMRを中心としたシステム
  - アドホック分析、Cynapse(モニタリングツール)で利用
- データ処理の詳細
  - S3 ⇒ EMR ⇒ RDSの処理フロー
  - データの取得タイミングは1日もしくは1時間
  - 社内で定めた独自フォーマットを利用
- 現在の検証項目
  - Spark
  - 共通IDでの分析
  - 機械学習

今後の展望

・アナリストがアクセスしやすい環境の構築

・Spark利用の常態化

・機械学習のさらなる活用

発表は以上となりますが、

最後に人材募集のお知らせです。

多様なメンバーで構成されています

チームリーダ兼アナリスト
- 経営企画出身（M&A,事業戦略策定,等）
- 問題解決,ビジネス知識
- Excel,R,SPSS

アナリスト
- マーケティング部門出身
- Webマーケティングに関する知識
- Excel,SPSS,R

アナリスト
- ゲームプランナ出身
- ゲーム運営への深い理解
- SQL,R,Excel

アナリスト兼BI開発エンジニア
- 博士号（工学）保有
- 機械学習,統計学,フロントエンド開発
- R,Python,JavaScript

データ分析エンジニア
- BtoBソリューション部門出身
- システム設計,Hadoop関連技術,DB管理
- Perl,PHP,Scala

データ分析エンジニア
- ゲーム開発専門学校出身
- Hadoop関連技術,DB管理,機械学習
- Python

エンジニア募集中！

(アナリストも！)

http://www.cybird.co.jp/recruit/

ご清聴ありがとうございました