Case Study 1

ケーススタディ 1

東急ハンズ―ポイントシステム刷新

Redshiftのデータ連携が難航し設計やり直す

2015年12月に稼働させた会員顧客向け新ポイントシステムで、データウエアハウス「Amazon Redshift」とイベント駆動型のプログラムコード実行環境「AWS Lambda」を採用した東急ハンズ。両サービスの機能不足や特性から、2度にわたって設計をやり直した。

（八木 玲子）

「Amazon RedshiftとAWS Lambdaに、使って初めて分かる落とし穴があった」。東急ハンズのIT子会社、ハンズラボの小林亮介氏（イノベーショングループ シニアエンジニア）は、リーダーとして2015年12月に完了したポイントシステムの刷新プロジェクトをこう振り返る。

ポイントシステムは、東急ハンズの店舗やECサイト、スマートフォンアプリに共通する会員制ポイントサービス「ハンズクラブ」の中核を担うもの（**図1**）。店舗のPOSシステムやECサイトのシステムと連携し、会員顧客ごとの基本情報、ポイント残高、ポイント履歴、購買履歴を追加・更新する。データウエアハウス（DWH）に蓄積したデータは、ポイント追加・更新履歴が1億2000万件、購買履歴が2億3000万件に上る。

小林氏らは、オンプレミス（自社所有）環境で稼働させていた従来のポイントシステムを、Amazon Web Services（AWS）向けに刷新するプロジェクトを担当した。東急ハンズは、全システムのAWS移行を進行中で、その一環である。

小林氏らは、システム開発方針として、技術的な制約を課せられた。

まず、リレーショナルデータベース（RDB）ではなくNoSQLデータベース「Amazon DynamoDB」を使う必要があった。DynamoDBは、ポイント追加・更新の処理途中で障害が発生したときデータの整合を取る仕組みを備えていない。小林氏らは独自に

図1　新ポイントシステムの概要

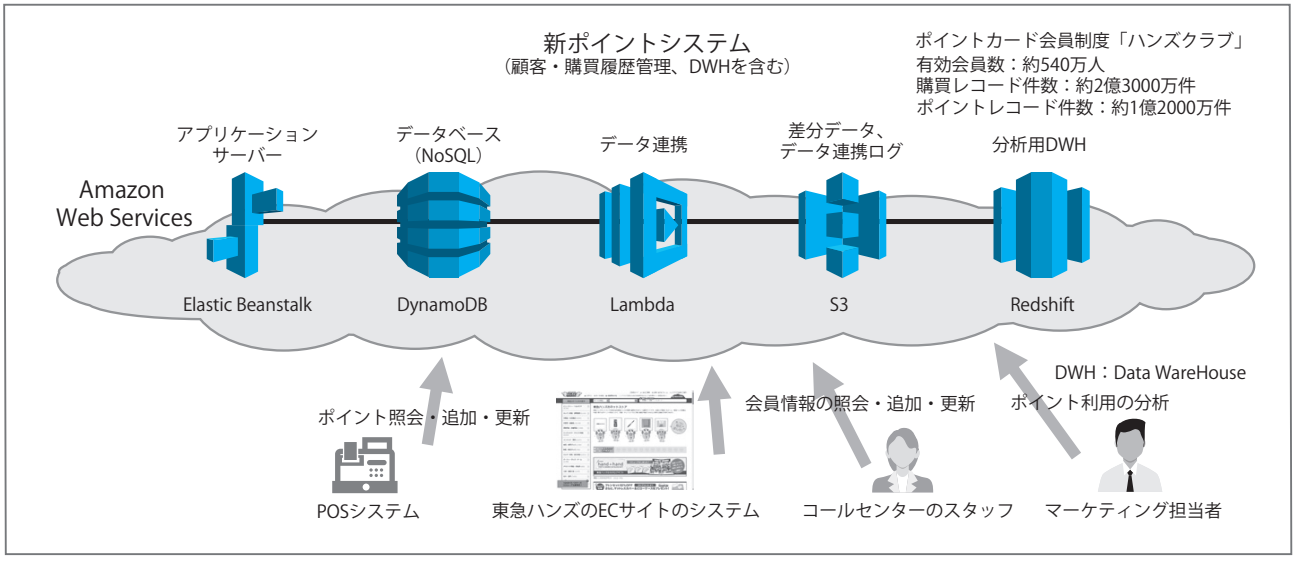


図2　新システム構築の流れと乗り越えた障壁

作らなければならなかった（**図2**）。

運用に手間が掛かる仮想マシンは使わない、という制約もあった。そのため、ポイントシステム内部でのDWHへのデータ連携用に、仮想マシン「Amazon EC2」ではなく、イベント駆動のプログラム実行環境AWS Lambdaを採用した。これに起因して、思わぬ障壁にぶつかり、2度にわたってアーキテクチャーを見直すことになる。

さらに移行フェーズでは、従来システム上の3億5000万件以上のデータを変換してAWS上の新システムへ移す、という壁に直面する。

これらの障壁を、小林氏らはアーキテクチャーを繰り返し見直すことで乗り越えた。プロジェクトの始まりから、具体的な経緯を見ていこう。

二人のチームで開発する

ポイントシステム刷新プロジェクトが始動したのは、2015年6月。プロジェクトチームは、リーダーの小林氏と、曽根努氏（ハンズラボ イノベーショングループ）の二人だ。

従来のポイントシステムは2012年に、ユニバーサル・シェル・プログラミング研究所が提唱するユニケージ開発手法を採用し、自社開発したもの（次ページの**図3**上）。UNIX系OSのシェルスクリプト（システムコマンドを並べたプログラム）を使うシンプルな仕組みで、「店舗からIT部門に異動してきた部員でも習得しやすかった」（小林氏）。

ただしポイントサービスの規模が拡大するにつれ、応答の遅さや検索性の悪さが顕在化。新システムでは処理性能と検索性が求められた。

新システムでは、前述したように、仮想マシンを使わないことも必要だった。仮想マシン上でアプリケーションやデータ連携プログラムを動かすのでなく、AWSが自動運用を行うマネージドサービスを使うことで、運用の手間を減らせる。

しかもAWSのマネージドサービスの一部は、システム負荷に合わせて、自動的に処理能力を高めたり下げたりする。システム負荷に合わせたコンピューティングリソースの見直しが不要になるうえに、柔軟に必要な処理性能を維持できる。

そこでアプリケーションの動作環境には、アプリケーションサーバーのサービス「AWS Elastic Beanstalk」を採用。データ連携には、イベント駆動でプログラムコードを実行するAWS Lambdaを選んだ。どちらもAWSによるフルマネージドサービスで、処理能力の上げ下げも含めて運用が不要だ。

①NoSQLの障壁
データの整合を保つ仕組み自前で作り込む

データの管理には、前述の通り、NoSQLデータベースのDynamoDB

を選択した。RDBを使わなかったのは、ハンズラボ全体の学習コストを重視したからだ。

従来システムではデータをテキストファイルで管理しており、RDBは利用していなかった。新システムを内製するに当たり、新たにRDBを導入するとなると、店舗勤務からシステム担当になった部員もSQLを習得しなくてはならない。新システムの開発には携わらないとしても、運用保守を担当する可能性がある。

NoSQLのDynamoDBなら、ユニケージ開発手法と親和性が高く、学習コストが抑えられる。しかも当時既に別システムの開発でDynamoDBを使い始めていた。

ただしNoSQLデータベースを使うことによる技術的な制約もあった。まず問題になったのが、複雑な集計処理に向かないこと。ポイントの精算や利用動向の分析に集計機能は欠かせない。そこで、集計機能に優れたDWHサービスAmazon Redshiftを併用することに決めた。

DynamoDBとRedshiftの間は、Lambdaを使ってデータ連携させる（図3下）。DynamoDBには、データの書き込み（追加・更新）があったことをイベントとして他のサービスに伝える機能「DynamoDB Streams」がある。Lambdaがこのイベントメッセージを受け取ると、Redshiftにデータを書き込む。

だが単純にDynamoDB、Lambda、Redshiftを連携させるだけでは不十分だった。「Lambdaはログが残らない仕様なので、そのままではRedshiftにデータが書き込まれた経過を追跡できない」（小林氏）。トラブル発生に備えて、データ書き込みのログを残す必要があった。

そこで、オブジェクトストレージサービス「Amazon S3」を利用し、どこまで処理が進んだかが分かるようにする構成を考えた。まず、DynamoDBからLambdaを通じてS3上に、Redshiftへの書き込みデータを記録する。さらに、S3からも書き込みイベントメッセージを発生させ、それを受け取ったLambda

図3　従来システムを基に考案した当初アーキテクチャー

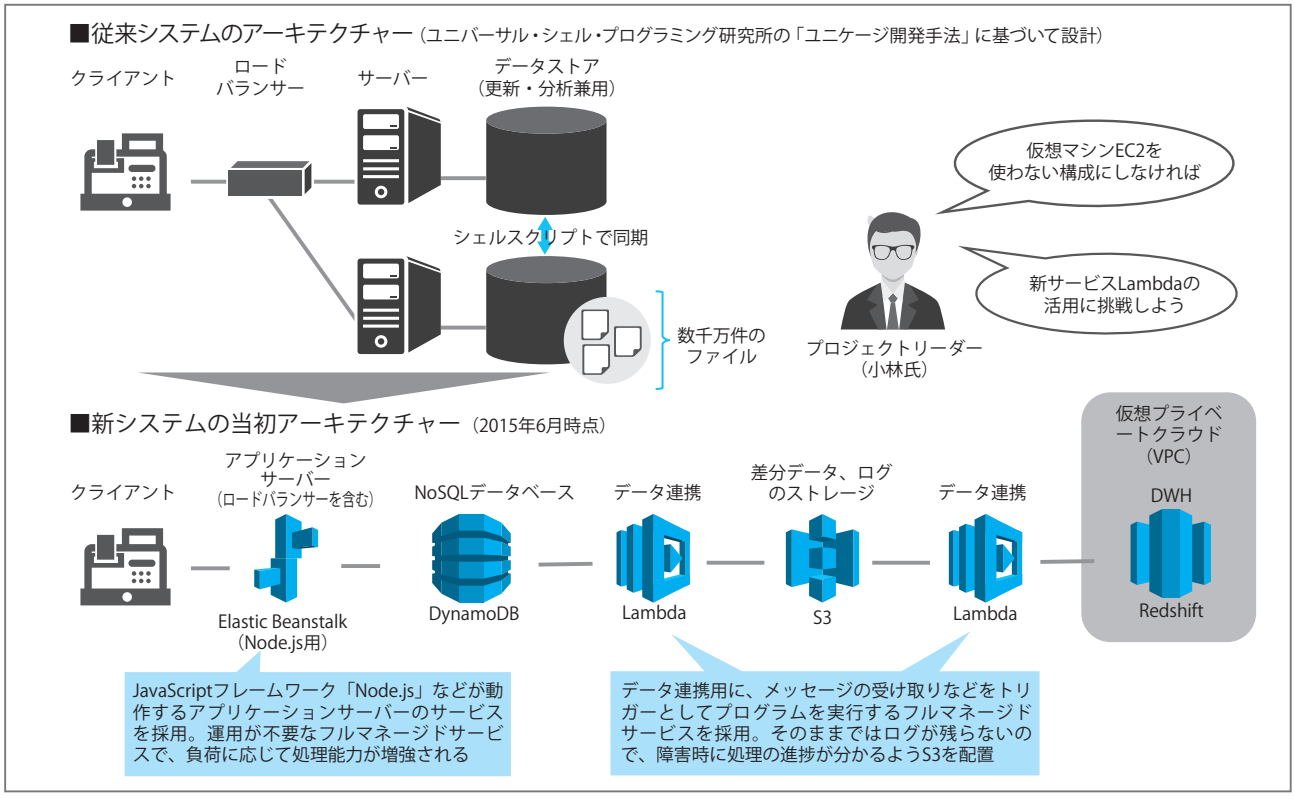
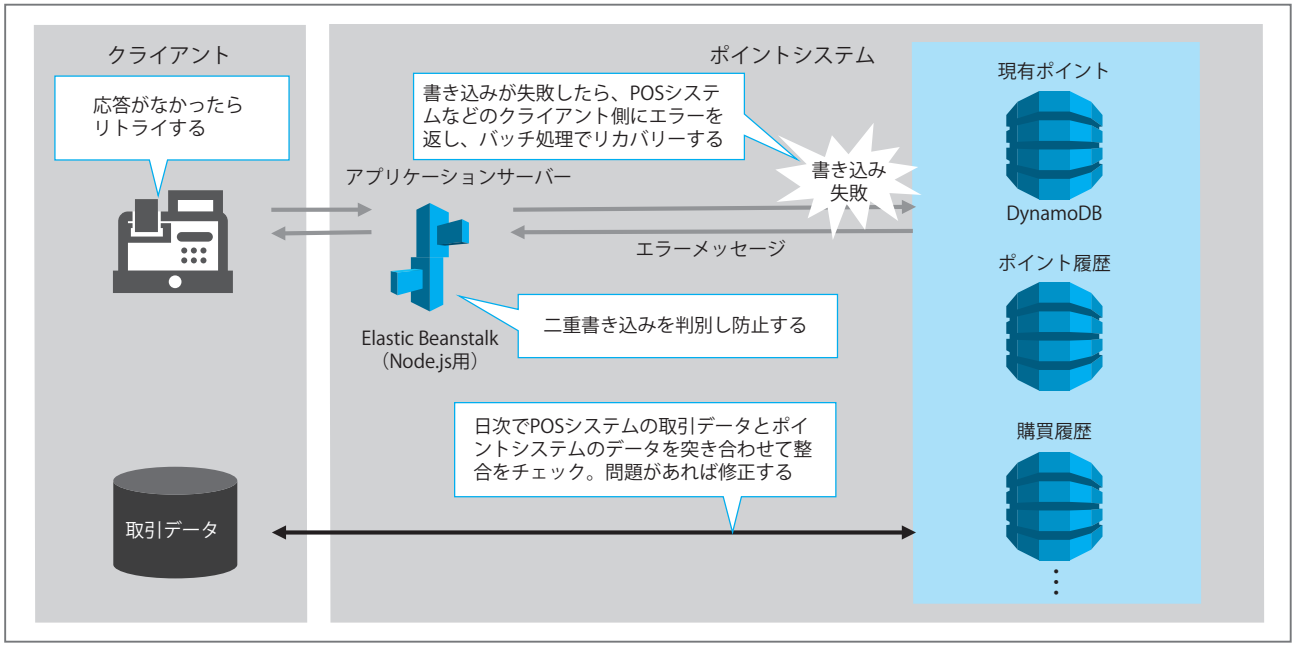


図4　データの整合を保つ仕組み



がRedshiftにデータを書き込む。Lambdaはこれらの動作をするたび、S3にログを書き込む。

NoSQLデータベースを使ったので、データの整合を保つ仕組みも必要だった（**図4**）。データの更新時に障害が発生すると、不整合が起こり得る。例えば何らかの原因でPOSシステムからポイント加算の同一リクエストが2度届いたとき、そのまま処理すると二重加算になる。

DynamoDBでは、そうしたデータの不整合を防ぐ仕組みを自前で作り込む必要がある。考えた仕組みは次のようなものだ。

「ポイントシステムで書き込みが失敗したら、POSシステムなどのクライアント側にエラーを返し、バッチ処理でリカバリーする」「クライアント側は応答がなかったらリトライする」「ポイントシステムは、二重書き込みを判別し防止する」「日次でPOSシステムの取引データとポイントシステムのデータを突き合わせて整合をチェックし、問題があれば修正する」。

これらの機能をアプリケーションとして実装する。

こうして、小林氏らは2015年6月中に当初のアーキテクチャーを設計した。

②データ連携の障壁

Redshiftへの書き込みでコネクション数が上限超過

「Lambdaで、思ったようなデータ連携ができない」。小林氏が障壁に直面したのは、実際にシステムを作り始めた2015年7月のことだった。

DynamoDBとS3は想定通りに連携できたが、S3とRedshiftをつなぐ部分で問題が発生した。Redshiftは、仮想プライベートクラウド「Amazon Virtual Private Cloud（VPC）」内に用意したが、LambdaはVPC内のリソースにアクセスできないことが判明した（次ページの**図5**）。当時のLambdaは、VPC内のリソースへの接続機能を持っていなかった（2016年2月11日、Lambdaにこの機能が追加された）。

小林氏らは、当初アーキテクチャーの再考を余儀なくされた。

代替策として、第2のアーキテクチャーを考え出したのは8月ごろ。Lambdaの代わりに、メッセージキューイングサービスの「Amazon SQS」と、それと組み合わせるジョブ実行環境の「Elastic Beanstalk worker tier」というサービスを利用することにした。worker tierは、SQSで受け付けたキューを順次実行する役割を担う。

これは、非同期処理を実現するうえで定番といえるアーキテクチャー。だが、テストしてみると

思わぬ壁にぶつかった。今度は、Redshiftの書き込み性能が足かせになった。

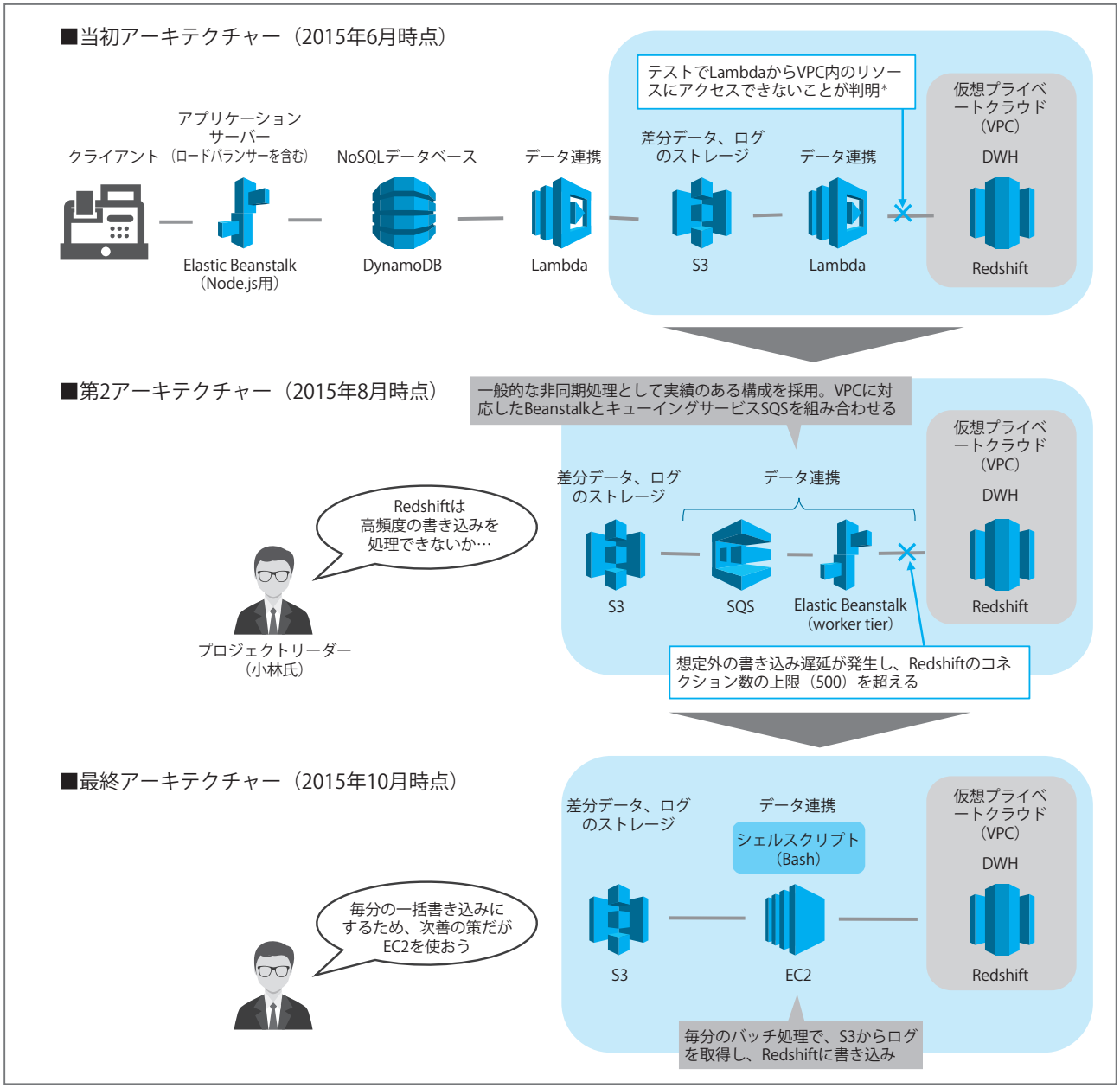
第2のアーキテクチャーでは、DynamoDBへの書き込みが発生するたびに、Lambda、S3、SQSを経由して、worker tierがRedshiftにデータを書き込む。その際、「Redshiftへの書き込みに時間が掛かることが分かった。1件当たり2～3秒掛かることもあった」（小林氏）。

Redshiftには次々と書き込み要求が届く。Redshiftの同時コネクション数は、上限の500を超えた。

バッチ処理で書き込み遅延を解消

「これではダメだ」。小林氏らは、またしても設計をやり直すことに

図5　アーキテクチャーの変遷



*2016年2月にLambdaの機能強化でVPC内のリソースにアクセス可能になった

なった。

問題を解決するには、Redshiftへの書き込み頻度を減らすほかない。そこで、書き込み要求を1分間ためておき、一括してRedshiftに渡す方式に変更した。

具体的には、S3とRedshiftの間に仮想マシンのEC2を設けて、1分おきのバッチ処理で書き込み要求をRedshiftに送る。

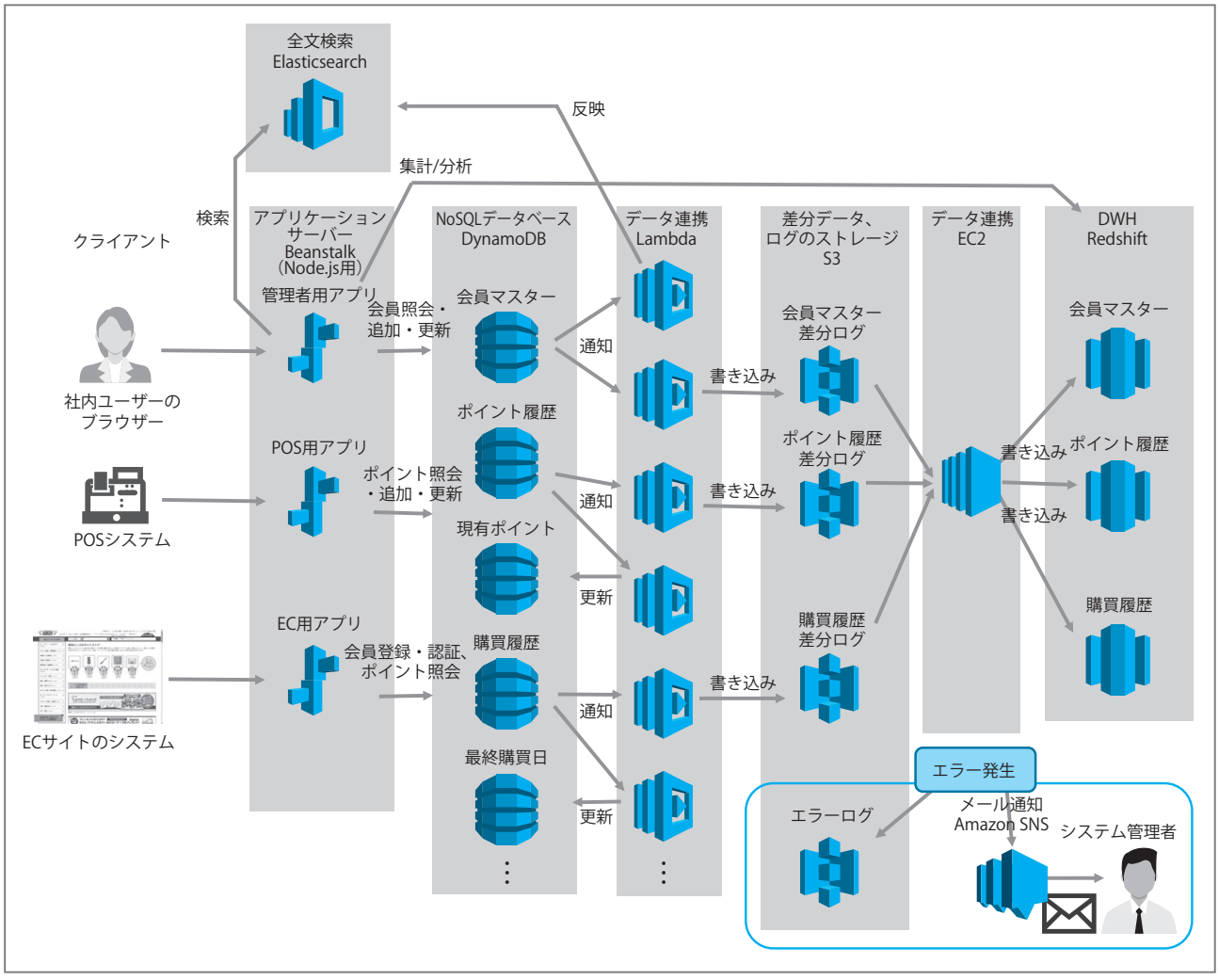
Redshiftにデータが反映されるまでに、1分程度の遅延が発生するが、「Redshiftのデータは、リアルタイムで確認する必要がない。翌日に売り上げデータを確認するといった使い方なので、この程度の遅延は問題にならない」(小林氏)。

次善の策としてEC2を使用するが、1分ごとのバッチ処理による書き込みに変えたことで、Redshiftの遅延を解消できた。

こうして新ポイントシステムの最終アーキテクチャーが固まったのは2015年10月のことだ(**図6**)。

JavaScriptフレームワーク「Node.js」で開発したアプリケーションが、Elastic Beanstalk上で稼働。データ格納用のDynamoDBから集計用のRedshiftへのデータ連携は、Lambda、S3、EC2を通じて行う。また検索機能として、全文検索サービス「Amazon Elasticsearch」を組み込んだ。

図6　完成したアーキテクチャー



③データ移行の障壁

当初方式では1カ月以上 分散処理で数日に短縮

小林氏らには、システム稼働までにもう一つ越えなければならない障壁があった。従来システムからのデータ移行である。

従来システムには、合計で3億件を超えるデータが保存されている。当初は、データ移行用にEC2を用意し、自作のシェルスクリプトを使ってDynamoDBに順次書き込む方式を考えた（**図7**）。DynamoDBには同時に25件を書き込めるため、データを25件ずつに分けて移行する。

具体的には、AWSが公開するSDK（Software Development Kit）と軽量言語のPHPを使って、DynamoDBへの書き込み用コマンドを作成。これをシェルスクリプトから呼び出す方式を採った。

試しに動作させてみると、時間が掛かりすぎた。「1度実行するごとに、SDKのモジュール読み込みだけで0.5秒掛かった」（小林氏）。購買履歴データの2億3000万件を例にした場合、SDKの読み込みだけで1カ月以上を要する計算になり、現実的ではない。

そこで小林氏らは、移行方式を再検討。並列処理する方式を考えた（**図8**）。移行プログラムをNode.jsのアプリケーションとして開発し、Elastic Beanstalk上で動作させる。Elastic Beanstalkの稼働基盤はEC2であり、そのインスタンスサイズや台数を手動設定できる。

小林氏らはCPUコアが1個の小規模なインスタンス「t2.micro」を30台設けて、それぞれ移行用プログラムを稼働させた。DynamoDB側では、「キャパシティー」と呼ばれる書き込み性能を一時的に上げる。こうして分散処理を行った結果、移行時間の大幅短縮に成功。最も件数の多い購買履歴でも、所要時間は7時間ほどだった。

データの移行は延べ数日間で完了。2015年12月に、新ポイントシステムをカットオーバーさせた。

Lambdaの適用を拡大へ

小林氏が今後の課題として挙げるのが、Lambdaの適用範囲の拡大だ。LambdaがVPC内のリソースにアクセスできないという制限から、今回はDynamoDBとRedshiftとのデータ連携にLambdaとEC2を併用した。

だが「EC2をLambdaで置き換えると、運用が楽になる」と小林氏は話す。Lambdaは、タイマー設定でプログラムコードを実行できる。2016年3月中をメドに、現在EC2で実現している1分間隔のバッチ処理を、Lambdaで置き換える考えだ。

図7　従来システムからの当初のデータ移行方法

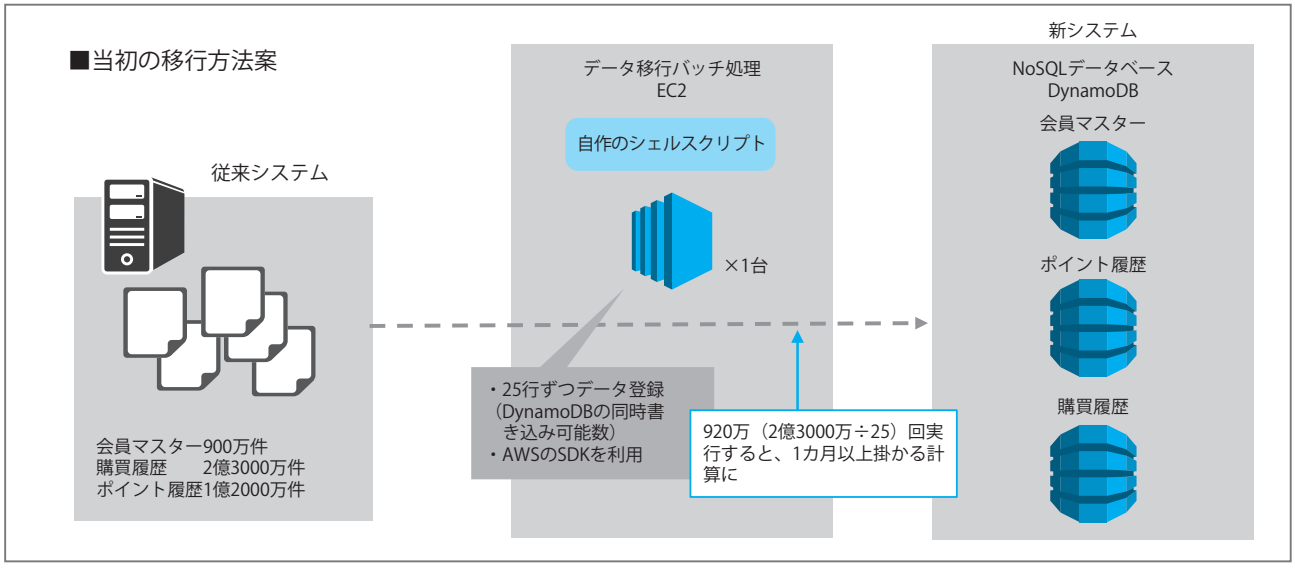
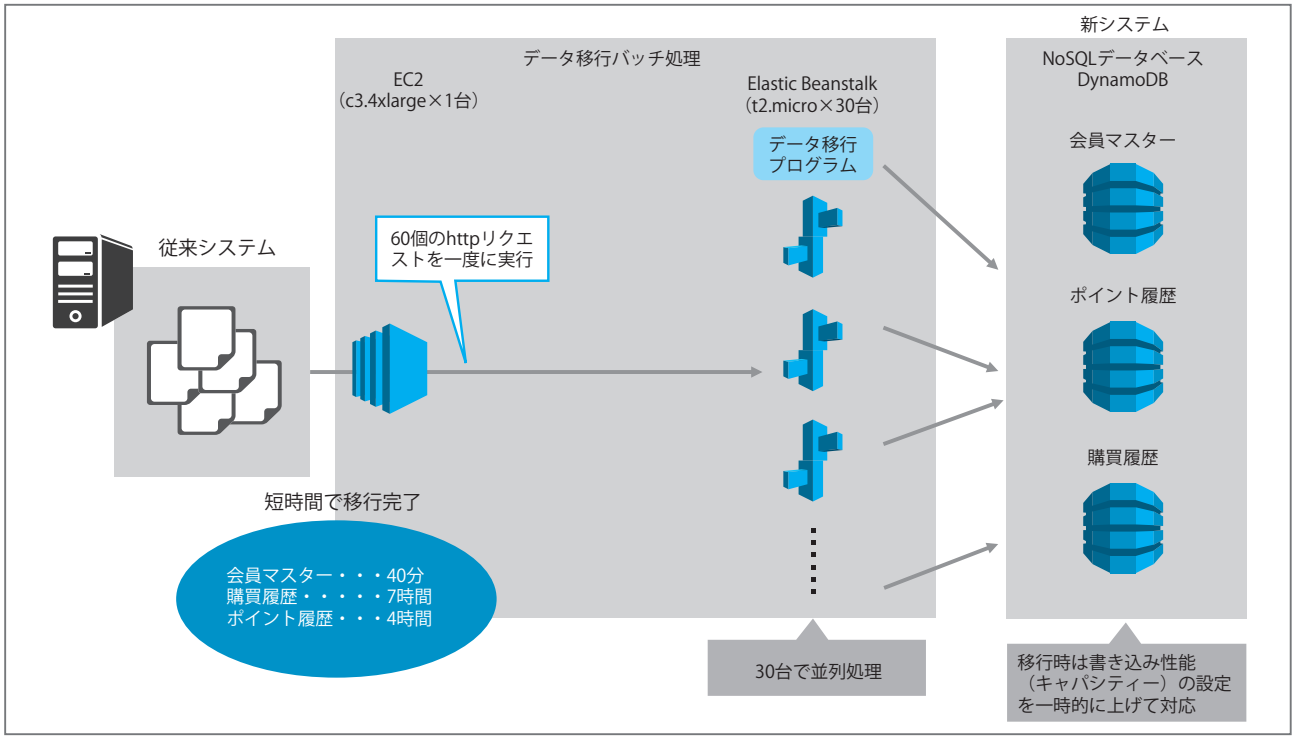


図8　改善したデータ移行方法



ケーススタディからの学び

Redshiftは逐次書き込みにせずバッチ処理でデータを移す

東急ハンズが直面した障壁の一つは、Redshiftの書き込み性能。POSシステムなどからDynamoDBへデータを追加・更新するたび、Redshiftに反映させるアーキテクチャーにしたところ、Redshiftの性能が追いつかずコネクションが超過した。

ただし、これはRedshiftに問題があるわけではない。野村総合研究所 副主任システムコンサルタント クラウド基盤サービス三部 瀬戸島敏宏氏によれば、「Redshiftは大量データの集計に特化した構造で、逐次書き込みとの両立は技術的に難しい」。トランザクションのたび書き込む設計によるコネクション超過は、Redshift導入のアンチパターンだという。

ハンズラボの小林氏らは回避策として、S3にためた差分データをEC2によってバッチ処理でRedshiftに書き込む仕組みにした。これは、Redshiftへのデータ転送の定石の一つ。

瀬戸島氏も、「Redshiftへの書き込みの受け皿として、S3、Amazon RDS、Amazon Elastic MapReduceを配置。ここから必要なデータだけをバッチ処理で抽出しRedshiftにコピーする使い方が良い」と勧める。