Google、Amazon、Microsoftを連携させたアプリケーション構成の提案

クラウド技術センター
技術開発室
ITエンジニア

森本　淳司

Atsushi　Morimoto
atsushi-morimoto@exa-corp.co.jp

Google、Amazon、Microsoftの提供するクラウドサービスを利用したアプリケーション開発を行うとそれぞれのサービスは強みと弱みを持つことがわかる。本稿では各クラウドサービスの強みと弱みを明らかにして、強みを活かし弱みを補うために、複数のクラウドサービスを連携した３階層のアプリケーション構成を提案する。またアプリケーションを試作することでクラウドサービス同士の連携の実現性を検証し、実現することができた。

1. はじめに

企業におけるクラウドコンピューティング（以下、クラウド）の利用状況、認知度は年々確実に増加している。[1)] クラウドの特性、メリットが企業のニーズにマッチするためであろう。クラウドの代表的なメリットを示す。[2)]

- 情報システムを所有する形態から利用する形態に変えることで、システム投資の低減化を図ることができる。
- 従量課金制により、初期投資の負荷が少ない。
- 必要なときに必要な量だけ利用することができ、不用になった場合すぐにやめることができる。つまり、情報システムコストを変動費化することができる。

一般にクラウドを適用したアプリケーション開発ではGoogleやAmazon Web Services、Microsoft®などの企業が提供するGoogle App Engine™（以下、App Engine）やAmazon Web Services™（以下、AWS）、Windows Azure®（以下、Azure）などのサービス（以下、クラウドサービス）を利用する。

しかし、Google、Amazon Web Services、Microsoftが提供するクラウドサービスはすべて同じ機能ではなく、それぞれ強み弱みがある。各社クラウドサービスの強みを活かし弱みを補い合うことができれば、より最適なアプリケーションを開発できる。

そこで本稿では、クラウドサービスの代表的な特徴を整理し、強みと弱みを明らかにする。そしてその結果をもとにクラウドサービスの強みを活かし、弱みを補い合うことができる構成を提案する。また、本提案構成の検証を行うためにアプリケーションを試作した。検証においてクラウドサービス間の連携は、実績が少なく困難が予想されたので重点的に検証した。最後に課題と結論、今後の展望を示す。

なおアプリケーションの試作に用いたクラウドサービスは、一般的な開発技術の適用を考慮した結果、App Engine、Azure、Amazon EC2™、Amazon S3™を採用した。

2. クラウドサービスの強みと弱み

クラウドサービスの強みと弱みを明らかにするために、クラウドサービスの各社HPおよび文献を調査し、代表的な特徴を整理した。

2.1. 特徴一覧[3) 4) 5) 6) 7) 8)]

表１に各クラウドサービスの特徴を示す。

表１　クラウドサービスの特徴

	App Engine（Google）	Azure（Microsoft）	AWS（Amazon Web Services）
(1) 形態	PaaS	PaaS	IaaS
(2) データストア	KVS	RDB/KVS	RDB/KVS
(3) 制約	多数	少数	少数
(4) 開発言語	Java/Python	.NET（C#/VB）	OSに依存
(5) 開発環境	Eclipse	Visual Studio Windows Vista/7	ターミナル
(6) 無料利用	あり	なし	なし
(7) 料金体系	従量課金	従量課金	従量課金
(8) スケーラビリティ	自動	自動	手動

(1)　形態

App EngineとAzureはPaaS（Platform as a Service）である。PaaSはアプリケーションの開発・実行環境をサービスとして提供する形態である。AWSはIaaS（Infrastructure as a Service）である。IaaSはサーバーやネットワークをサービスとして提供する形態である。AWSは仮想マシン利用サービスとしてAmazon EC2、ストレージサービスとしてAmazon S3を提供している。

(2)　データストア

KVS（Key Value Store）と呼ばれスケーラビリティを活かすことのできるデータストアがある。ただしこれはスケーラビリティを確保するため、リレーショナルデータベース（RDB）がもつ厳密なACID特性を有さない。

App EngineではBigTableと呼ばれるKVSを利用できる。

Windows Azure Platformにおけるデータストアにはストレージサービスと RDBサービスの２つがある。１つ目のストレージサービスには４種類のサービスがある。大規模なバイナリデータを保存するためのBlob（Binary Large Object）とKVSであるTable、アプリケーション間の通信のためのQueue、そしてNTFSとして読み書きできるDriveが存在する。２つ目のRDBサービスは部分的な制

約があるもののクラウド上でSQL Server®を実現したものである。

AWSはAmazon Simple Storage Service™（S3）、Amazon Elastic Block Store™（EBS）、Amazon SimpleDB™など多数のデータストアを提供している。Amazon EBSはAmazon EC2のインスタンスから利用する形態であり、Amazon Simple DBはKVSである。

(3) 制約

App Engineは制約が多い。ここでは代表的な制約を紹介する。App Engineにはリクエストを処理し、レスポンスを返すまでの時間が約30秒以内でなければならない、リクエストタイマーと呼ばれる制約がある。制限時間を超えた場合、Exceptionが発生し割り込み処理が行われる。また、ファイルの読み込みはできるが、書き込みはできない制約もある。例えばアップロードされたデータをファイルとして保存することはできない。そして、アプリケーションが独自にTCP/IPのソケットを開くことができず、外部ホストと通信するためにはApp Engineが提供する機能を用いる必要がある。

一方、App Engineと比較してAzure、AWSの制約は少ない。Azureの代表的な制約事項はログ閲覧の仕組みがないことである。MicrosoftはAzure上のアプリケーションが出力するログを閲覧する仕組みを提供していない。よってその仕組みを自作するか別途ツールを利用する必要がある。

AWSではサービスによって利用できるコンピューティングリソースの上限が決定されている。例えばEC2ではインスタンス数の上限は20個と決まっている。ただし、AWSに申請することでこの上限を変更することは可能である。

(4) 開発言語

GoogleはJava®ランタイム環境とPython®ランタイム環境を提供している。Javaランタイム環境ではJavaプログラムをはじめ、JVM上で稼動するScalaプログラムなども実行できる。Pythonランタイム環境ではPythonプログラムを実行できる。

Azureは.NETランタイム環境を提供しているため、C#プログラム、VBプログラムを実行できる。その他にもAzureはFastCGIをサポートしているためPHPプログラムなども実行できる。

AWSはIaaSであるため、クラウド利用者が開発言語やOSなどを自由に選定でき、開発の自由度は高い。逆を言えば、クラウド利用者が準備することはPaaSより多い。

(5) 開発環境

App EngineおよびAzureの開発には統合開発環境を使用できる。App EngineではEclipse®を、AzureではVisual Studio®を利用して開発できる。ただし、Azureのアプリケーションを開発するためには、Windows Vista®またはWindows® 7が必要である。

AWSはIaaSであるため統合開発環境は特に存在しない。例えばAmazon EC2でLinux®の仮想マシンを利用する場合は、PuTTYなどのターミナルを使用することができる。そしてPuTTYからSSH（Secure SHell）でLinux仮想マシンにリモート接続してサーバー構築を実施できる。EC2でWindows仮想マシンを利用する場合はリモートデスクトップ接続（RDP）を利用できる。

(6) 無料利用

App Engineは一定のコンピューティングリソースを無料で使用できる。Azureにも無料利用プランはあるが、期間限定キャンペーンのため「無料利用」は「なし」とした。

(7) 料金体系

基本的にすべて従量課金制である。つまり、使用したCPU時間、保存したデータ容量、データ転送量などに応じて徴収される料金が決定される。App Engineは一定量を無料で使用できるが、課金を有効にすることでそれ以上のコンピューティングリソースを使用できる。Azureは実行環境の仮想マシンのスペックにより料金が異なる。Amazon EC2はサーバーの起動時間で課金され、Amazon S3はデータ保存量、データ転送量などに応じて課金される。

(8) スケーラビリティ

App Engineには自動スケーリングの仕組みがある。App Engine上のアプリケーションは利用するユーザー数などに応じて自動でスケールアウトする。Azureで自動スケーリングを実現するには「Dynamic Scaling Azure」などのツールを利用しなければならない。[9] 一方AWSでスケールアウトを行う場合、クラウド利用者が環境を構築しなければならない。具体的にはEC2を用いて必要な数の

仮想マシンを用意する。そして負荷分散を実現するためにロードバランサであるElastic Load Balancingを用いる必要がある。

2.2. 強みと弱みの一覧

2.1.で整理したクラウドサービスの特徴を強みと弱みに分類し、その結果を表２に示す。

表２　クラウドサービスの強みと弱み

	強み	弱み
App Engine	自動スケーリング 無料で利用開始できる	実行時間に制限があるなどの制約が多い
Azure	.NET環境を利用できる RDBを利用できる	Windows Vista/7が必要
AWS	構築自由度が高い さまざまなサービスがあるため用途に合ったものを利用できる	自由度が高い反面、設定、メンテナンスはユーザーが行う

3. 構成の提案

クラウドサービス間を連携する３階層の構成を提案し、それを実現するために必要な検証ポイントについて説明する。

3.1. 構成

３階層の構成を提案する。構成は図１に示すようにフロントエンド、バックエンド、データストアからなる。フロントエンドはクライアントとのやりとりを行うプレゼンテーション層であり、バックエンドはアプリケーションのロジック部分を担当するビジネスロジック層である。

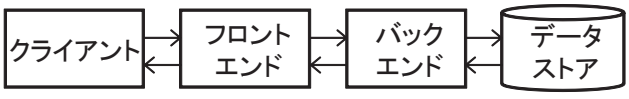

図１　クラウドサービス間連携

この構成のメリットの一例として、サーバーの負荷分散やシステムの変更容易性が挙げられる。具体的にはフロントエンドでスケーラビリティを活かして多数の処理を実行し、バックエンドで負荷の高い処理を実行することで負荷分散を実現できる。また、フロントエンドとバックエンドに分割することで変更時の影響範囲を小さくできる。

表２で示したクラウドサービスの強みと弱みから、フロントエンドとバックエンドの最適な組み合わせを考える。

フロントエンドはApp Engineが適している。これはApp Engineの自動スケーリング機能を活かすことができるためである。ただしリクエストタイマー制約は図１の構成全体に影響を及ぼしてしまうが、Googleが将来的にリクエストタイマーを延長すると表明しているため本稿ではフロントエンドにApp Engineを選択した。[10] バックエンドはAzureかAmazon EC2が適している。AzureはApp Engineのような制約はなくさまざまな処理を実行でき、EC2はIaaSであることから目的に合わせたサーバーを構築できるためである。この選択によりバックエンドではApp Engineが実現できないファイルの書き込み処理などできる。

3.2. 検証ポイント

図１の構成を実現するためにはクラウドサービス上のアプリケーション間で連携できなければならない。そこで本稿ではRESTを用いてフロントエンドとバックエンド上のアプリケーション同士が連携できるか検証した。一般的にアプリケーション間を連携するための技術にはSOAPとRESTがある。SOAPと比較してRESTはWebブラウザでURLを指定すると簡単にXMLを取得できるため利用しやすい。[11] このことから検証ではRESTを採用した。またRESTの定義はたびたび議論の的になるが、本稿では「HTTPを用いて指定されたURLに接続するとXMLが返却される通信形態」と定義する。

4. アプリケーション試作による検証

３章で提案した構成を検証するためにアプリケーションを試作した。この試作アプリケーションはオンライントランザクション向けシステムを想定しており、次の機能を提供する。

- 画像の登録

・　類似画像の参照

本稿では表３が示すように２つの構成について検証した。

表３　検証構成パターン

	フロントエンド	バックエンド	データストア
1)	App Engine	EC2	S3
2)	App Engine	Azure	S3

データストアは両構成ともAmazon S3を選択した。選択理由は、画像を保存するためにデータベースを利用するよりファイル形式でストレージに保存するほうが容易なためである。

4.1. 検証構成パターン1(App Engine - EC2 - S3)

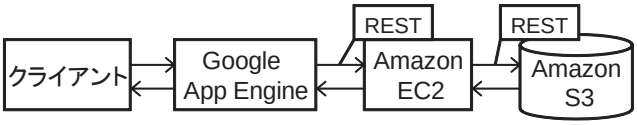

図２　App Engine - EC2 - S3連携

本節では、表３の検証構成パターン1）について解説する。利用したクラウドサービスは３つあり、フロントエンドにApp Engine、バックエンドにAmazon EC2、データストアにAmazon S3を利用した。検証した構成を図２に示す。

4.1.1. Google App Engineの役割

Google App EngineにはJava版とPython版の２種類がある。検証ではJava版であるGoogle App Engine for Javaを用いた。

App Engineはクライアントから画像登録を要求されたとき、画像をEC2に送信して登録を依頼する。そして画像登録の成功可否をクライアントに返却する。同様に類似画像の検索を要求されたときは、EC2に類似画像の取得を依頼し、その結果をクライアントに返却する。

画像処理は時間がかかるため、クライアントがWebブラウザで画像を参照した場合の画像取得ではEC2を経由せず、S3に保存されている画像のURLを直接参照させた。これより処理時間を節約することができる。

4.1.2.　Amazon EC2の役割

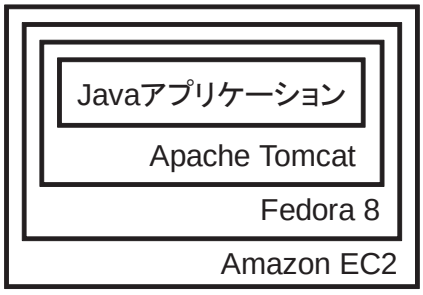

図３　EC2構成

Amazon EC2はApp EngineとAmazon S3の中継役をバックエンドで行う。EC2基盤上にはLinuxやWindowsなどのOSをインストールでき、その上で自由なサーバー構築を実現できる。検証ではLinux OSにFedora® 8を、WebアプリケーションサーバーにApache Tomcat™を選択した。この構成を図３に示す。Tomcat上で稼動するJavaアプリケーションは、App Engineからの要求に応じてS3に画像を保存する処理を行い、また類似度結果をS3から取得しApp Engineに返却する処理を行う。一般的にEC2は処理に時間のかかるビジネスロジックを担当することができる。

4.1.3. Amazon S3の役割

Amazon S3では画像データの保存および類似度結果の保存を行う。これら処理を行う際、AWSが提供している各種言語（Java、 C#、etc）用のREST、SOAPライブラリを使用できる。今回の検証で用いたS3操作用ライブラリは「Amazon S3 Library for REST in Java」である。

4.1.4. App Engine - EC2間連携

App Engine上のJavaアプリケーションとEC2上のJavaアプリケーションで通信を行った。しかし、App Engine上のアプリケーションはソケットを開くことができず、外部ホスト（ここではEC2上のJavaアプリケーション）と通信するにはURLフェッチJava APIを利用し

てHTTP、HTTPS接続する必要がある。URLフェッチJava APIの利用には２つの方法がある。

- Java標準ライブラリのjava.netの利用
- App Engineの低レベルAPIの利用

２つの方法の違いは例外を発生させることができるか否かである。URLフェッチサービスは送信するリクエストと受信するレスポンスのデータサイズを制限している。制限するデータサイズを超えた場合、Java標準ライブラリのjava.netは通知なしで切り捨てられる。一方、低レベルAPIを利用する場合は、通知なしで切り捨てるか、例外を発生させるかを選択できる。試作アプリケーションではユーザビリティの観点から例外処理を正しく行う必要があるため、App Engineの低レベルAPIを利用した。なお、java.net、低レベルAPIともにURLフェッチサービスは約５秒以内に外部ホストに接続できないとタイムエラーになる。

EC2上のJavaアプリケーションはApp Engineからのリクエストに応じるシンプルなサーブレットである。App EngineよりEC2上のサーブレットが指定するURLに画像登録のリクエストが来るとS3に登録を行い、そして成功可否をXMLに記述して返却する。リクエスト中にある画像のデータサイズは不確定のためGETを利用せず、POSTを利用した。また画像類似度取得のリクエストが来ると、S3から画像類似度を取得し、画像が保存されているS3のURLをXMLに記述して返却する。App EngineではJAXBを用いてXMLとJavaの変換を行った。

以上よりEC2の指定するURLにApp Engineが接続しHTTP通信することでクラウドサービス間の連携をRESTで実現できた。

4.2. 検証構成パターン2（App Engine - Azure -S3）

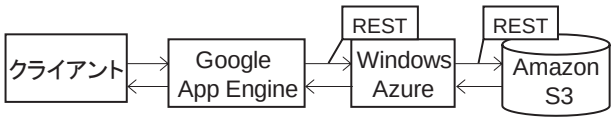

図４　App Engine - Azure - S3連携

本節では、表３の検証パターン2）について解説する。基本的な構成は4.1.と同じである。検証した構成を図４に示す。異なる点はバックエンドを担当するクラウドサービスにAzureを採用した点である。ここではEC2との相違点などAzureを中心に解説する。

4.2.1. Windows Azureの役割

Windows Azureは.NETアプリケーションなどを稼動させるための実行環境である。Azureの役割は4.1.2.と同様にApp EngineとAmazon S3の中継役である。

Azureはクラウドによるスケーラビリティを確保するために、Webロール、Workerロールと呼ばれる役割の異なるプログラムで構成される。WebロールはASP.NETで構築されるWebアプリケーション、またはWCF（Windows Communication Foundation）で構築されるWebサービスである。一方、Workerロールは非同期処理やバッチ処理などバックエンドで働く処理を行うアプリケーションプログラムである。

検証ではクラウド用のWCFである「WCF Service Web Role」を利用した。WCFの.NETアプリケーションがApp Engineからの要求に応じてS3の操作を行う。このS3の操作では、.NETアプリケーションからEC2やS3を操作するためのライブラリを含む「AWS SDK for .NET」を使用した。

4.2.2. App Engine - Azure間連携

App Engine上のJavaアプリケーションとAzure上の.NETアプリケーションで通信を行った。基本的なコンセプトは4.1.4.と同じである。つまりApp Engine側の仕組みはAzureが指定するURLに接続しHTTP通信を行う。そしてAzureが返却するXMLをJavaに変換する。

異なる点はAzure側で「WCF Service Web Role」を利用したことである。.NETが提供するライブラリを使用することでクラウドサービス間の連携をRESTで実現できた。

4.1.4.のEC2ではApp Engineに返却するためのXML応答の仕組みを独自に作成した。Javaアプリケーションで RESTを実現するためには、仕組みを独自に構築するだけではなくフレームワークを利用する方法もある。一般的にEC2はIaaSであるためサーバーの設定、構築をクラウド利用者が行わなければならないが、フレームワークの選定などにおいて自由度の高い開発を行うことができる。一方、PaaSであるAzureは実行環境を提供しているためIaaSと比較して開発自由度は低いものの、RESTなどの機能を実

現する場合には.NETのライブラリを活用し効率的な開発ができる。

5. まとめ

クラウドサービスの強みを活かし弱みを補い合うため3階層の構成を考案した。またその構成の中で必須技術となるクラウドサービス間（フロントエンドとバックエンド間）の連携を実現できた。

以下に試作アプリケーションに対する課題と対策、今後の展望、そしてクラウド利用における注意を述べる。

5.1. 課題とその対策

提案する構成を実用する場合は、次のような課題を解決しなければならない。

1) リクエストタイマーによる処理時間の制約

リクエストタイマー制約は将来的に延長されると仮定して3章の構成を提案した。しかし、現時点でリクエストタイマー制約を回避する方法を考えておかなくてはならない。回避策として、クライアントとバックエンドとの接続を非同期にする方法である。時間のかかる処理を行ったバックエンドは結果を一時的にキューと見立てたBigTableなどのデータストアに処理結果を保存する。クライアントはAjaxなどでポーリングしてそのデータストアにある処理結果を取得する。

2) URLフェッチサービスの接続制限時間

App Engineが外部ホストと通信するためのURLフェッチサービスには、規定された時間（約5秒）以内に接続できない場合はタイムアウトエラーになるという制約がある。したがって、タイムアウトエラーが発生したときにリトライを行い再接続する仕組みが必要である。

5.2. 今後の展望

1) 構成の有用性を検証

提案した構成をもとにアプリケーションを開発することはできたが、それぞれのクラウドサービスの強み（自動スケーリングなど)が有効に機能するかまでは検証できなかった。今後は構成の有用性を検証するためにApp Engineの自動スケーリングの効果などを測定していきたい。

2) 社内システム-クラウド間連携

クラウドの利用が活発化するにともない社内システムに保存している機密情報の活用方法や既存システムの活用方法が必要になる。つまり社内システムとクラウドサービスの連携は必須である。

画像類似度の計算はHadoop™クラスタの効果を測定するためバッチ処理として自社内で行った。ファイアウォールがあるためクラウドサービスから社内システムには接続できない。そこで社内システムからAmazon S3に接続して画像を取得し、Hadoopクラスタで画像類似度の計算を行い、その結果を再度Amazon S3に保存した。

今後は社内システムからクラウドサービスに接続するプル方式だけではなく、クラウドサービスから社内システムに接続するためのサービスであるWindows Azure AppFabricなどの技術を検証していきたい。

5.3. クラウド利用における注意

全体的な注意として、App Engine、Azure、AWS のSLA（サービスレベルアグリーメント）、目標復旧時間（Recovery Time Objective）を考慮すると、稼動できるアプリケーションはすべてノンミッションクリティカル向けになる。また、データストアとしてApp EngineのBigTableなどを利用する場合、リアルタイムなデータ更新を必要とするアプリケーションの開発は困難である。[12)] なぜならスケールアウトに強みのあるクラウドでネットワークの遅延なしにデータの一貫性を保つのは困難なためである。アプリケーションをクラウド化する場合、これらクラウドサービスの特性と実現するアプリケーション特性とを十分考慮する必要がある。

本稿のデータは2010年1月－9月時点のものである。クラウドサービスの変化は早いため、各クラウドサービスを利用する際にはデータ等の必要事項を注視しなければならない。

参考文献

1) インプレスジャパン『インターネット白書2010』インプレスコミュニケーションズ（2010）
2) IPA独立行政法人情報処理推進機構『クラウド・コンピュー

ティング社会の基盤に関する研究会報告書』、http://www.ipa.go.jp/about/pubcomme/201003/201003cloud.pdf、2010/08/13
3) Google Code『Google App Engine』http://code.google.com/intl/ja/appengine/、2010/09/10
4) AWS『Amazon Web Services』http://aws.amazon.com/jp/、2010/09/10
5) Windows Azure『Windows Azure Platform』http://www.microsoft.com/windowsazure/、2010/09/10
6) （株）グルージェント『Google App Engine for Java[実践]クラウドシステム構築』技術評論者（2009）
7) 株式会社日立システムアンドサービス 酒井 達明『Windows Azureアプリケーション開発入門』日経BP社（2010）
8) TIS株式会社SonicGarden 並河祐貴 安達輝雄『クラウドAmazonEC2/S3のすべて 実践者から学ぶ設計/構築/運用ノウハウ』日経BP社（2009）
9) msdn『Windows Azure Dynamic Scaling Sample』http://code.msdn.microsoft.com/azurescale、2010/09/09
10) Google App Engine『App Engine Product Roadmap』http://code.google.com/intl/ja/appengine/docs/roadmap.html、2010/09/10
11) XML コンソーシアム『エンタープライズ・システムのためのWeb2.0』http://www.xmlconsortium.org/wg/web2.0/teigensho/、2010/09/13
12) 伊藤孝行『クラウドコンピューティング技術』、http://pari.u-tokyo.ac.jp/policy/policy_issues/PI10_03_cloudcomputing.pdf、2010/08/23
