〈クラウドの技術課題，将来展望〉

7

クラウドアプリケーションの分析と開発手法

萩原正義　マイクロソフト（株）

クラウドのアプリケーション開発は疎結合のスケールアウト設計を基本とする．すなわち，大規模分散配置したクラウドの物理ノード上にデータを分割して配置し，アプリケーションが並列実行して非同期通信により協調する．疎結合には，従来のサービス指向（Web サービスや REST☆1），キューイングが踏襲され，並列実行や非同期通信の実装には関数型パラダイムが一部で利用される．本稿では，従来のオブジェクト指向やコンポーネント指向によるアプリケーション開発，RDB を対象としていたデータ設計に加えて，こうしたスケールアウト設計の複雑さがどのように影響を与え，実際のアプリケーション開発がどのようになるか，開発手順と分析法を考察する．

クラウドのアプリケーション開発上の制約

クラウドの動作特性上，アプリケーション開発に対して考慮をすべき制約が存在する．以下の 4 つは代表的制約であり，アプリケーション開発に対して影響が大きい．

■非同期通信

クラウドは可用性を重視するため，部分的な障害を許容する作りであることをアプリケーションに求める．これまでの密結合アーキテクチャでは同期呼び出しが主であり，通信の参加者間でライフタイムの同期が必要となり，障害個所が全体の稼働を停止させてしまう．結果として，部分的な障害の許容には，非同期通信が本質的に必要である．一方，スケーラビリティの確保，ボトルネック解決のためのデータ分割やトランザクション分割が必要であるが，この分割データ間の更新の一貫性のための同期呼び出しが新たに必要となる．しかし，同期呼び出しは可用性の条件を満たすために利用できないので，一貫性に対する条件の緩和が必要となってくる．

■常時稼働

クラウド上のアプリケーションは原則として常時稼働であり，データ更新操作を常に受信可能としておくことが基本である．同時に，スケーラビリティの確保の観点で，アプリケーションが一旦読み取ったデータを更新できないように書き込みロックをしておくことは不可能である．したがって，読取りデータは常に古いデータとして扱うことが必要で，そのための設計が要求される．これを Postel's Law（RFC793）と呼ぶ．

■CAP（Consistency, Availability, network Partitioned）定理

可用性とスケーラビリティの確保を前提とするとデータの一貫性が確保できないことを提示する．したがって，データー貫性を必要とするアプリケーションでは，BASE（Basically Available, Soft state, Eventually consistent）トランザクションと呼ばれる，従来の ACID（Atomicity, Consistency, Isolation, Durability）トランザクションとは異なるデーター貫性の確保のためのアプリケーション設計が必要となる．BASE トランザクションを提供するための一手法が eventual consistency と呼ばれる consistency モデル☆2 である．

■キーバリューストア（KVS）

クラウドのデータサービスとしては，従来の OS レベルのファイルシステムに相当した，より抽象度の高いキーバリューストアが主流である．従来のオンプレミスの

☆1 REpresentational State Transfer の略．すべての情報（リソース）に URI による一意の識別子を持たせ，HTTP の CRUD 操作（POST/GET/PUT/DELETE）をステートレスに実行する．リソースへのリンクを使って，他のリソースへの参照，リソースの表現（状態）の転送による状態遷移を行う．

☆2 データの一貫性要求の強さを表現したモデル．

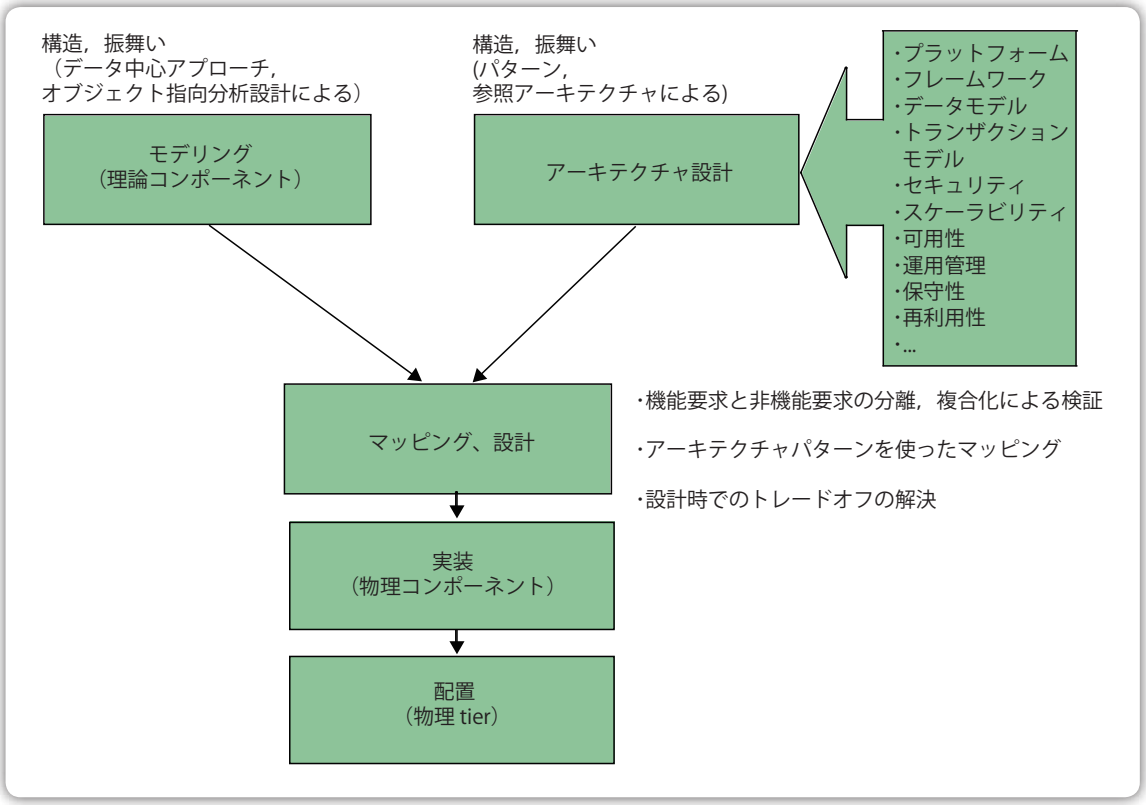

図 -1　アプリケーション開発手順

企業システムで主流の RDB はスケーラビリティの確保，XML やストリームなどの非構造化データへの対応の点でクラウドへの適用は困難である．クラウド上でも比較的小規模のアプリケーションや既存システムからの移行を重視する場合は RDB も利用可能であるが，一般的なクラウドのデータサービスはキーバリューストアであり，それとクラウド上の RDB，オンプレミスの RDB のハイブリッドな構成でデータサービスが稼働するのが現実的である．キーバリューストアは表形式のデータモデルを持つ RDB とは異なるデータ設計，実装法，管理方法が必要となる．キーバリューストアは行データごとに異なる物理ノードに配置され，行データ単位でトランザクションを実行するため，スケールアウト設計が基本となる．

クラウドのアプリケーション開発手順

クラウドのアプリケーション開発をするかどうかによらずアプリケーション開発手順には，（a）アーキテクチャ先行定義（Architecture-centric），（b）機能要求とアーキテクチャのすり合わせ，の原則が存在する（**図 -1**）．アプリケーション開発手順では，アプリケーション機能要求のモデリングの段階と，非機能要求を実現し複数のアプリケーションの機能要求を動作させる基盤となるアーキテクチャの設計の段階とを並行に開発し，アーキテクチャ設計に短期的な個別アプリケーションの要求を入り込ませず，汎用性と可変性（拡張性）を持たせることを考慮する．次の段階として両者の整合性の考慮を加え，機能要求を表現するモデルへの非機能要求や可変性のための「揺さぶり」を加えること，アーキテクチャのビジネス的な価値の正当性の評価などを通して両者をすり合わせていく．アーキテクチャは長期的に複数のアプリケーション開発での実現により検証し，再構築を経て成熟させていく．アーキテクチャは長期的な再利用性で投資効果を評価すべきである．

クラウドのアプリケーション開発ではこの開発手順の原則に従い，加えてスケーラビリティのためのスケールアウト設計に従って開発を進める．クラウドのアプリケーション開発手順の全体を**図 -2** で示す．すなわち，クラウドのデータサービスの利用を想定し，キーバリューストアの特徴である行単位のトランザクションの制約，遅延の大きなリモートアクセスの回避，異なる行間での eventual consistency による一貫性の確保，などを想定しなければならない．スケーラビリティはデータ層のボトルネックが大きく影響することを考慮すれば，データ設計を第一に考えたアーキテクチャを構築し，そこにアプリケーションの機能要求，つまり，データアクセス（ワークロード）を加味していく手順をとる．キーバリューストアのデータ単位がキーとバリューからなる行の非正規化データであることから，最初に論理レベルのデータ設計をデータの正規化で進め，次に物理設計でアプリケ

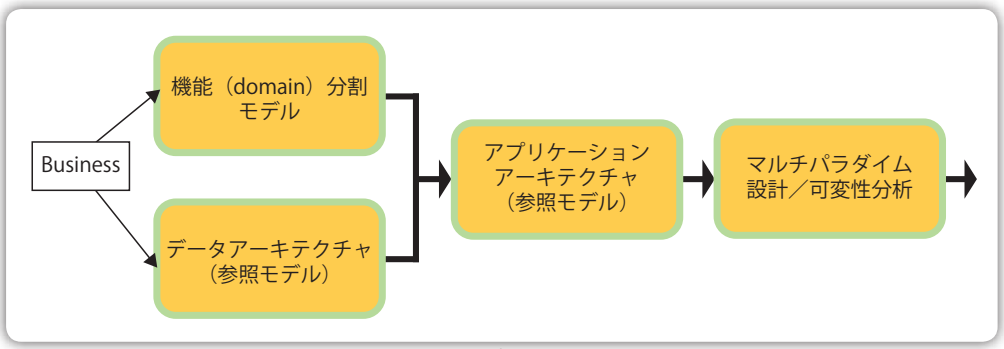

図 -2　クラウドのアプリケーション開発手順

ーションのデータアクセスを考慮してデータを非正規化し，リモートアクセスの最小化を考慮する．よって，開発手順としては，図 -2 に従った以下の各手順となる．

(1) 機能（domain）分割モデル：

スケーラビリティを左右するアプリケーションのデータアクセスを決定するため，アプリケーションの機能分割を行い，機能ごとのデータアクセス方法を明らかにする．機能分割にはサービス指向のサービス単位の分割を用いる．

(2) データアーキテクチャ（参照モデル）：

（1）と並行して，個別のアプリケーション機能に依存しないデータアーキテクチャを設計する．データアーキテクチャはアプリケーションの機能要求の考慮に先行して，マスタデータとトランザクションデータの論理レベルの配置に関する原則に従って設計する（**図 -3**）．データアーキテクチャの先行定義とそのための分析は既存のデータ中心アプローチ（DOA：Data Oriented Approach，正しい英語では，Data Centric Approach）の分析設計の基本的な考え方である．この原則は，データの保守性，再利用性，データアクセスなどの特性の違いからマスタデータとトランザクションデータを分離し，最適な論理レベルの配置を決定する．アプリケーション開発で物理レベルの配置や運用管理の方法としてクラウドを利用するかどうかにかかわらずデータ管理の基本となる．クラウドの特徴であるキーバリューストアの利用やそのデータの非正規化，水平パーティション化などの物理設計手法は，この論理レベルの配置の原則の上に成立し，データの記憶と管理，アクセスの最適化の実現法の一種となる位置づけである．以下に要点を示す．

（a）アプリケーションから共通性の高いマスタデータを外出しして定義，論理的に一元管理
（b）アプリケーションに依存するトランザクションデータはアプリケーションにローカルに管理
（c）マスタデータをトランザクションデータから一方向に参照
（d）トランザクションデータを集約，加工してデータウェアハウスを構成

(3) アプリケーションアーキテクチャ（参照モデル）：

アプリケーションの機能分割とデータアーキテクチャをすり合わせてデータの物理設計，配置を決めてスケーラビリティの基盤を作る．すり合わせではデータアクセス方法の方針を決定する参照アプリケーションアーキテクチャを参考にしてアプリケーションアーキテクチャを設計する．クラウドのアプリケーション開発での参照アプリケーションアーキテクチャには，非同期通信を主としたビジネスプロセスのロングトランザクション，クライアント／サーバ型で共通リソースを共有操作する協調アプリケーションなどが存在する．

より詳細には，データアーキテクチャの論理レベルの配置から物理レベルの最適な配置を決定する．物理レベルのデータは論理レベルのデータを分割または別の論理レベルのデータと複合化する．ただし，物理レベルの配置とはクラウドでは特定の物理ノードへの配置を意味せず，仮想化された物理ノードへの抽象化した配置となる．たとえば，構造化オーバレイで実現したクラウドの物理ノードは，障害時や運用上の都合で他の物理ノードに透過的に置換え可能な抽象化をクラウドが提供する．

ここでのデータ定義の区別は，論理レベルのデータ定義は正規化した保守用データとして設計時に利用し，物理レベルのデータ定義は非正規化，パーティション化したクラウドの運用管理用データとして実行時に利用する．このように論理レベルと物理レベルのデータ定義を分離して，分析設計段階で論理レベルの定義から物理レベルの定義を導き出す過程を踏むことが重要である．

さらにデータアクセスの実装の抽象化のために別のデータモデルを定義する場合がある．その結果，論理レベルの設計で使用する ER（Entity-Relationship）図，物理レベルで使用するキーバリューストアの行データ定義（Microsoft 社の Azure Storage Service と Google 社の Big Table では Entity，Amazon 社の SimpleDB では domain

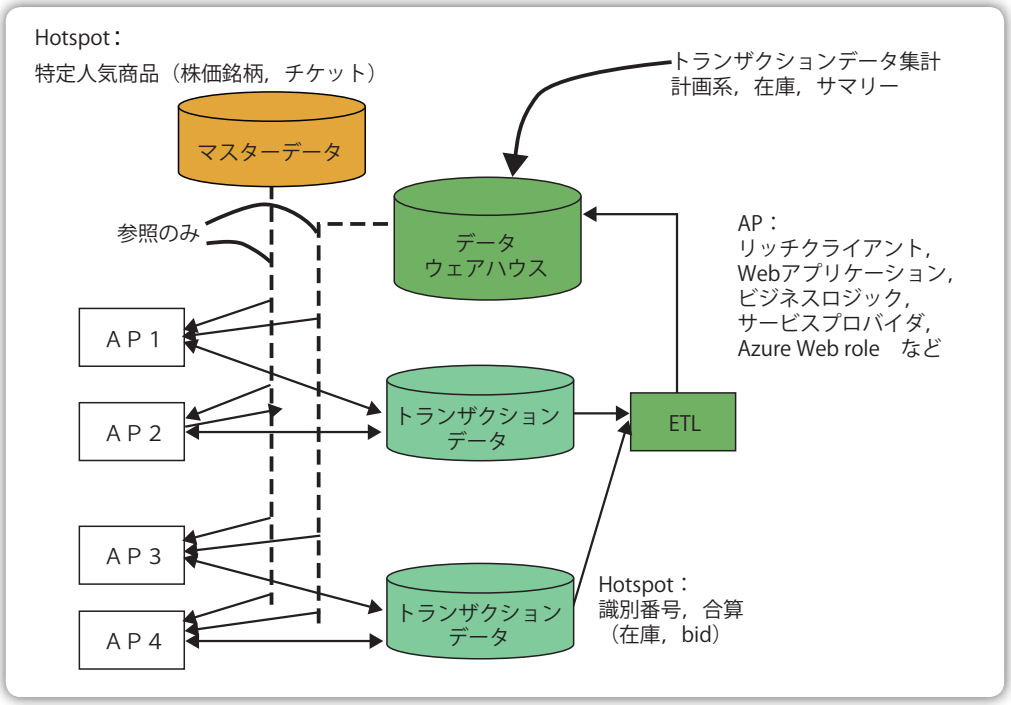

図 -3　データアーキテクチャ（マスタデータとトランザクションデータの論理レベルの配置）

の item と呼ばれる），クラウド上のプログラミングモデルで定義されるデータアクセス用のデータモデル（通常は物理レベル．Azure では ADO.NET Entity Framework の概念レベル）の代表的な 3 種類のデータモデル定義が存在することになる．これらはそれぞれの目的で区別し，分析設計段階で適切に利用していく．すなわち，資産としてデータ定義を保守する対象は論理レベルの ER 図であり，モデル駆動型開発では物理レベルのキーバリューストアの行データ定義を生成するために，この ER 図を使う．あるいは，プログラミングモデル上の抽象化したデータモデルから物理レベルの行データ定義を生成する．Azure では既存の RDB のスキーマ定義を読み取って抽象化したデータモデルである ADO.NET Entity を定義し，それを利用して RDB や Web サービスへのアクセス操作を行う．これはオブジェクト指向におけるモデル駆動型開発で，開発言語のクラス定義を保守の対象とせずに，開発言語のクラスに加えて Web サーバやアプリケーションサーバの関連構成定義を同時に生成する，コードと構成定義とを一体管理する UML や DSL モデルを保守の対象とするのと同じ考え方である．今後は，データ定義，コード，構成定義を統一的モデルで生成するための開発プロセスやツールが主流になるであろう．

☆3　ドメインごとに設計や実装のためのパラダイムの選択を変えることで，可変性に対して柔軟性を与える設計方法．たとえば，ロジックの実装，データ構造の振舞いの変化にはオブジェクト指向，分散バッチ処理の実行には関数型，変更個所の配置にはコンポーネント指向，機能の提供にはサービス指向というパラダイムの選択と組合せを使って設計を進める．

(4) マルチパラダイム設計／可変性分析：

（3）のアプリケーションアーキテクチャが決定した後，その上で動作するアプリケーションのコンポーネント化のための設計を行う．コンポーネント化の設計では，アプリケーション要求変更（可変性）に対して設計や実現法を選択するためのマルチパラダイム設計☆3 を適用していく．

なお，ここでの開発手順をプロジェクトで実際にどのように駆動するかはプロジェクト管理で採用する開発プロセスが決定する．

クラウドのアプリケーション開発の分析手順

クラウドのアプリケーション開発手順において，データアーキテクチャからアプリケーションアーキテクチャを決定する上で特に重要な分析は次の 3 項目である．

(a) データ－データ間の関係
(b) データ－トランザクション間の関係
(c) トランザクション－トランザクション間の関係

データおよびトランザクションの設計に必要な分析は，単独のデータやトランザクションで行うのではなく他との関係に注目して行う．またここでは，トランザクションでアプリケーション機能を捉えることが特徴であり，トランザクションはアプリケーション機能をユースケー

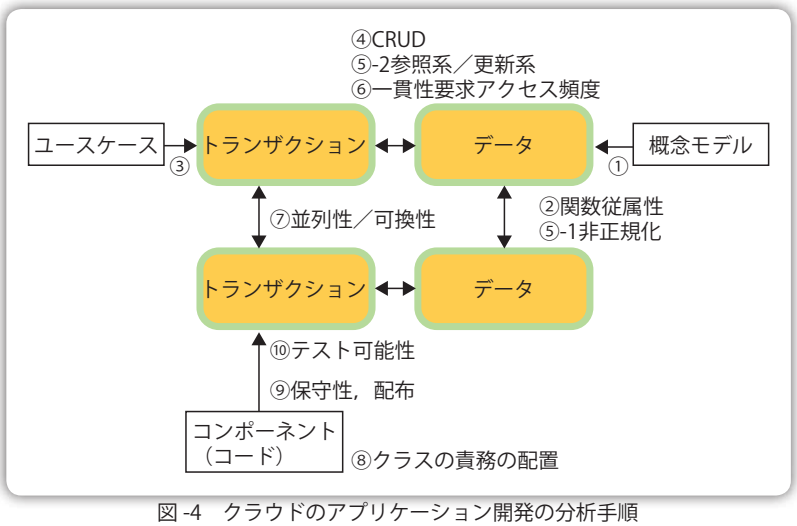

図 -4　クラウドのアプリケーション開発の分析手順

スのビジネス観点よりシステム寄りで捉え非機能要求の一部を取り入れる一方で，コンポーネントやクラスの実装レベルの分割では捉えていない点に注意する必要がある．なお，トランザクションをどのような構成要素単位で実現するかは，次の分析や設計の段階で考える．

上記の 3 項目の分析順序とそれらの関係を**図 -4** で示す．図 -4 に従い分析順序を説明する．

① 概念モデルから（論理レベルの）データを定義
② データ間の関係から関数従属性に従い正規化したデータを定義
③ （1）と（2）とは独立して，アプリケーションの機能要求を示すユースケース定義からトランザクションを定義
④ トランザクションのデータに対する操作を CRUD（Create/Read/Update/Delete）として識別
⑤ -1 データに対するトランザクションの操作から正規化データの非正規化を決定
⑤ -2 ユースケース（トランザクション）を参照系と更新系に分離
⑥ トランザクションの持つ一貫性要求，データへのアクセス頻度の要求を識別
⑦ トランザクション間の並列性，順序に関する可換性を識別☆4
⑧ トランザクションを実装するクラスのコードの責務の配置を決定
⑨ トランザクションを実装するコンポーネント単位を保守性，配布単位から決定
⑩ （テスト駆動開発のため）コードをテスト可能とする

クラウドのアプリケーション開発の分析手順でユースケース（トランザクション）を参照系と更新系に分離することは，従来の開発ではビジネスの要求がなければ必須ではなかったが，クラウドのアプリケーション開発では意図的な分離が必要となる．CAP 定理で説明したように，クラウドのアプリケーションはデータの一貫性に関して ACID トランザクションで実現していた即時一貫性を前提とすることはできない．つまり，アプリケーションでデータ更新した結果を読み取る場合，即時に更新が反映されない可能性がある．この可能性がある以上，更新系と参照系は明示的にユースケースの定義で分けておいた方がいい．ユースケース駆動で開発する場合は，プロジェクトの進捗管理やリスクの低減のためにそうすべきである．この参照系と更新系の分離はクラウドのアプリケーションの原則である．データ更新が生じるのは，ビジネス観点でのイベントが発生した場合であり，事実の記録が必要な場合である．一方，参照系は生じた事実を認識する時点で実行される．クラウドではこの 2 つは ACID トランザクションのときのように一体ではない．たとえば，注文というビジネス取引での売上が発生した場合，注文の事実（イベント）はトランザクションデータとして挿入される．一方，売上が実際に計上されたとの認識は，意思決定データとして集計される時点まで遅延可能である．これまでも，この 2 つのデータはデータウェアハウスでの分析などで見られるようにデータ加工のバッチ処理で遅延が許容されていた．参照系データの生成が即時に行われなくても問題がなかった．このようにビジネス上は多くの遅延処理がビジネスルールやバッチ処理などの要因で許容されている．したがって，eventual consistency がクラウド上のビジネスアプリケーションの要求を大きく損なうことは技術的には少ない．問題は従来の ACID トランザクションの信頼性に対する過度の期待と，ACID トランザクションを前提とした既存のフレームワークや開発環境のクラウドへの対応能力不足にある．むしろ eventual consistency は ACID トランザクションに伴う同時トランザクションのブロックによる同時処理能力の低下，同期処理による部分的障害への対応能力の不足の問題を改善する．

なお，データアーキテクチャを定義するための DOA の分析法と，その結果として定義されるデータを基準にして開発プロセスを進めるデータ駆動型開発は区別すべ

☆4 複数のトランザクション間で順序を入れ替えても実行結果が変わらない場合，それらのトランザクションは可換であるといえる．トランザクションが可換であれば，実行の順序を変えることで障害時の対応，排他制御などの改善をすることが可能である．

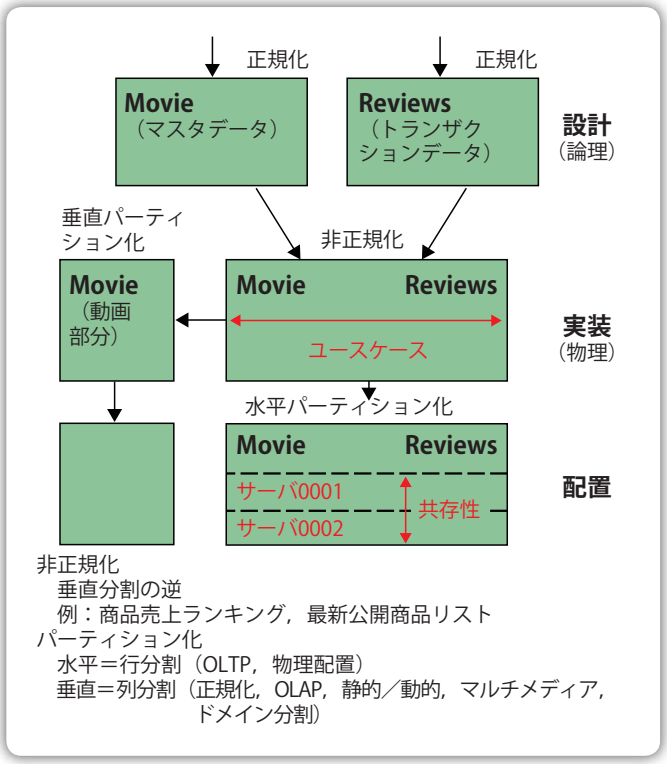

図 -5　クラウドのアプリケーション開発におけるスケールアウト設計

きである．前者は分析における区別すべきデータ特性，データモデル定義の位置づけや役割を決定し，後者は前者がどんな分析法を使ったかにかかわらず，定義された成果物の何を基準にして開発プロセスの全体を回すかのアプローチを与える．たとえば，DOA でデータを分析，定義し，ユースケースを基準にして開発プロセスを駆動する場合や，プロセス分析でビジネスプロセスを分析，定義し，サービスを基準にして開発プロセスを駆動する場合などがある．開発すべき対象，ドメイン，実行環境と技術に応じて分析法と開発アプローチを適切に組み合わせてリスクを軽減する．

ユースケースに関連する設計上の問題

クラウドのアプリケーション開発におけるスケールアウト設計では，アプリケーションのデータアクセス方法を決めるユースケースに従ってデータの非正規化，水平パーティション化を行うことが最重要である．その過程を図 -5 で示す．論理レベルでデータの正規化を ER 図で行い，次に物理レベルでのデータ非正規化，配置に関して水平パーティション化を実行する．水平パーティション化は，キーバリューストアが行データ単位で異なる物理ノードへデータ配置をするクラウドの特性を利用し，行データへのアクセスが並行処理になることを狙う．

ここでより詳細に設計段階を見ていくと，クラウドのアプリケーション開発ではユースケースが多くの問題に関連づけられていることに気づく．ここでは，ユースケースに関連する主要な 5 つの問題をまとめる．

(a) ユースケースがデータの非正規化，水平パーティション化の物理設計を決定
(b) 更新系と参照系のユースケースに分離し，eventual consistency によるデータ一貫性を実現
(c) 1 ユースケースはサービスの窓口になる boundary と，共通性の高いコアモデルの domain model の論理レベルのモデルとに分割
(d) 1 ユースケースは Web アプリケーションや Web サービスのためのフロントと，ビジネスロジックのためのバックのプログラミングモデルの物理レベルのモデルとに分割
(e) 1 ユースケースはクラウド上のトランザクションシステムのフロントと，オンプレミスのトランザクションシステムのバックとに配置

ここで，(a) (b) はユースケース単位で見た外部仕様である．(c) (d) (e) はユースケースの実装や配置に関する問題で内部仕様である．(c) は論理レベルの設計モデル，(d) は物理レベルの実装で利用するプログラミングモデル，(e) はクラウドのデータサービスやオンプレミスの RDB へのデータ配置やアプリケーション配置の問題である．

(c) の domain model は特定ユースケースに依存しないコアのオブジェクト構造で保守資産となる．(d) では (c) のできたモデルをクラウドの提供するプログラミングモデルで実現し，入出力や処理の役割を決定する．当然クラウドに実装依存である．(e) はトランザクション実行の役割分担をクラウドだけでなく，既存のオンプレミスのデータやアプリケーション資産と連携させて実行する配置を扱う．このデータ連携により，セキュリティや信頼性に関する SLA (Service Level Agreement) の確保，要求のカスタマイズ性，既存資産の有効活用など，クラウドだけでは十分な対応がとれない問題を改善する．

consistency モデルによる要求と concurrency モデルによる実装

アプリケーション開発の分析手順でトランザクションを利用するのは，アプリケーションの一貫性に関する要求をトランザクションの持つ一貫性要求に置き換えて表現するためである．トランザクションの持つ一貫性要求は consistency モデルによって表現する．consistency モデルの例としては，厳密な絶対時間に従ったデータ操作の順序関係を要求する strict，順序の相対的な関係

だけを要求する sequential，意味的に関連のあるデータの順序関係だけを要求する causal，データ（リソース）が解放されるまでに一貫性の解決を要求する release，次回のデータ利用を開始するまでに一貫性の解決を要求する entry などが存在する．これらは強い一貫性の要求から弱い一貫性の要求を表現しており，eventual consistency も弱い一貫性の要求の一表現である．アプリケーションの一貫性の要求には強い要求から弱い要求までのスペクトラムが存在し，1 アプリケーションであっても多くの一貫性の要求が混在している．すなわち，1 アプリケーションはさまざまな一貫性の要求を実現するための複数の consistency モデルを持ったトランザクションで表現される．

このような consistency モデルを実現する技術の選択肢が concurrency モデルである．代表的な例に楽観的ロックと悲観的ロックが存在する．ACID トランザクションは強い一貫性を保証するために悲観的ロックを利用する．しかし，ACID トランザクションにも隔離レベルと呼ばれる同時性制御に関する緩和条件が存在する．これを含めると concurrency モデルとして選択可能な実現技術は多様である．たとえば，非同期キューイング，ACID トランザクションの snapshot 隔離レベル，レプリケーションの各種トランザクションサポート，並列処理の reader/writer lock などである．これらの実現技術は consistency モデルの要求を満足する条件で適切に選択すべきである．クラウドのアプリケーション開発手順におけるアプリケーションアーキテクチャの設計では，concurrency モデルが適切に選択できることが重要な条件になる．

まとめ

クラウドのアプリケーション開発では，DOA の分析方法，SOA（Service Oriented Architecture：サービス指向アーキテクチャ）によるドメインや機能分割，オブジェクト指向分析設計によるオブジェクト指向開発言語を前提とした設計手法，コンポーネント指向による保守や配置，開発プロセス全体を駆動するユースケース駆動などを組み合わせる．それに加えて，並列実行や非同期通信の実装のための関数型パラダイム，モデル駆動型開発のための UML，DSL の抽象化したモデルによる保守とコード生成，可変性に対するドメイン分析とマルチパラダイム設計などを併用する．このように複雑に複合化した技術を用いるクラウドのアプリケーション開発では，技術の複合化に見られる妥協と合理性が問題ではなく，非同期通信とデータ非一貫性に根源的な問題があると思われる．クラウドという分散システムは信頼性 100% の仮定が成立せず部分的な障害の発生が前提であり，大規模なグローバルデータの一貫性の保証は本質的に不可能である．こうした“自然な”特性は，従来の同期通信や ACID トランザクションが逆に異常に制約が大きな仮説であったことを示している．従来の密結合の理想的な世界は，グローバリズム，成熟した社会での多様性，変化の加速化など社会的な要求に応えられなくなってきた．それがクラウド化の動機になったのではなかろうか．だとすれば，複合化した技術の複雑さに本質があるのではなく，従来の密結合システムの要素技術が陳腐化し，革新的な技術が誕生するところに本質があると見られる．たとえば，既存のスケールアップによるトランザクション性能が数万トランザクション／秒の段階に対して，スケールアウトによるデータ操作性能が数十万要求／秒の段階まで到達していることでも革新の一端を垣間見ることできる．

参考文献

1）萩原正義：アーキテクトの審美眼，翔泳社 (Mar 2009).
2）萩原正義：Windows Azure で理解するクラウド時代のアーキテクチャと開発手法，システム開発ジャーナル，Vol.10 (June 2009).
3）Helland, P.：Life Beyond Distributed Transactions (Jan. 2007).
4）Pritchett, D.：BASE An ACID Alternative, ACM Queue, Vol.6, No.3 (May/June 2008).
5）Stonebraker, M., et al.：The End of an Architectural Era（It's Time for a Complete Rewrite).

（平成 21 年 9 月 24 日受付）

萩原正義
masayh@microsoft.com
1993 年マイクロソフト入社．ソフトウェアアーキテクト．.NET Framework 関連，ソフトウェアアーキテクチャの研究開発と技術啓蒙に従事．クラウドコンピューティング，マルチパラダイム設計，ソフトウェアプロダクトライン，その他の開発方法論などが担当分野．早稲田大学大学院非常勤講師．