Vol.2018-IOT-40 No.47
2018/3/6

クラウド利用における設計を考慮したコストモデル

青嶋　智久†　　吉田　健一†

† 筑波大学経営システム科学専攻　〒 112–0012 東京都文京区大塚 3–29–1

あらまし　計算資源をインターネット経由でサービスとして利用できるクラウドは近年一般化しており，その需要は拡大を続けている．クラウドは オンデマンドで自給式のサービス・広範なネットワーク接続・リソースの共有・迅速な弾力性・サービスの測定等の特性を持ち，それらの特性から従量制の課金モデルが定着している．企業と開発者はクラウドにより初期費用の軽減と柔軟なプロビジョニングを得ることができ，ビジネスの機会を広げたが，想定外の利用に基づく予定外のコスト負担というリスクも抱えることになった．そのリスクを防止するためにはコストの早期予測が必要だが，コスト予測には手間と時間がかかり，しばしばビジネスでは時に正確さよりも迅速性が重要視される．この課題に対して本論文では，簡易さと再利用性を重要視したコスト算出手法を提案し，1) ワークロードとシステムアーキテクチャの関係 2) ワークロードとコストを関連づけることによるコストモデルの算出 3) パターンの概念を用いたコストモデルの再利用 の 3 つの視点から説明する．

キーワード　クラウドコンピューティング，コストモデル，デザインパターン，システムアーキテクチャ

Cost Model from System Architecture in Cloud

Tomohisa AOSHIMA† and Kenichi YOSHIDA†

† Graduate School of Business Sciences, University of Tsukuba　Otsuka 3–29–1, Bunkyo-ku, Tokyo, 112–0012 Japan

Abstract　Cloud computing, which can use computing resources as a service though the Internet, has prevailed in recent years. Its basic characteristics are On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, and Measured service. Based on these characteristics, Pay-Per-Use accounting model has been established. Companies and developers can gain initial cost reduction and flexible provisioning through the cloud and widen business opportunities, and reducing the risk of unexpected cost burden based on unexpected use. In order to prevent such risk, it is necessary to estimate the cost earlier by observing the business situation. In this paper, we propose a cost calculation method that emphasizes simplicity and reusability against this business problem and discuss it based on the following viewpoints. 1. Relationship between workload and system architecture, 2. Cost model calculation by relating workload, 3. Reuse of cost model based on concept of pattern.

Key words　Cloud computing, Cost model, Design pattern, System Architecture

1. まえがき

クラウドコンピューティング（クラウド）は提供形態やデプロイメントモデルによって分類されており，オンデマンドで自給式のサービス（On-demand self-service）・広範なネットワーク接続（Broad network access）・リソースの共有（Resource pooling）・迅速な弾力性（Rapid elasticity）・サービスの測定（Measured Service）の 5 つの特徴を持つ [1]．さらにクラウドでは「Pay-Per-Use」，つまり利用した分だけ支払いを行う従量制の価格モデルが定着しており，これらの特徴から利用者は必要に応じた動的かつ柔軟なプロビジョニングが可能である．ビジネスの展開に合わせて計算資源を随時調整することができる．この特徴は Armbrust et, al (2010) [2] でも触れられており，クラウドの経済的特徴は「資本コストを運用コストに変換できる」点であると説明されている．

さらに（特にパブリッククラウドにおいて）多数の利用者の計算資源の共有による規模の経済性の恩恵がある．十分に多くのシステムを内包するクラウドサービスの場合，各システムの負荷の変動は統計的な多重化によってクラウドサービス全体が負う負荷の変動を平滑化させる．そのため，相対的にプロバイダーは需要の変化に対して計算資源の調整を行いやすく，それが利用者・プロバイダー双方にとってより良い費用対効果をも

たらす．

このように利用者，特に需要の予測が困難なビジネス用途において利点があるクラウドだが，必ずしも利用しやすいかという点には検討の余地があると考えている．何故ならば多くの場合，企業ではほぼ必ず事業の予算，つまりそのシステムに今後どの程度のコストが発生するのかを事前に求められる．運用コストであるクラウドのコストは，売上に対してどの程度の割合で発生するかの観点が重要である．実際に利用量から発生するコストの軽視による，思いもよらない請求，俗に言う「クラウド破産」が発生することに注意喚起する声もある [3]．本論文では，このような課題に対して活用可能なクラウドサービス上のシステム構築におけるコスト推定手法について議論を行う．手法の提案では，ビジネスで利用されることを鑑み，簡易さと再利用性を重要視する．これはリリース後の需要予測が不確かな状況で，事前のコスト推定が必要となるケースを想定している．

2. 関連研究と研究目的

関連研究を参考するにあたり，1) 要素としてどのようなものを想定するべきか，2) クラウドのコストにはどのような特性があり考慮すべき事項があるのか，3) コストを意識し実運用で検討する際にどのような手段があるのか，4) クラウドの構成には構造のようなものはないのか の点に注目した．

クラウドがどのようなコスト要素によって構成されているかについては，コントロールレベルによる 5 つのコスト分類 [4] がある．これはデータセンターのコストが ストレージ・ネットワーク・電力・冷却・保守・設備 で構成され，各要素のコントロールをどの程度行うかで，利用者側のコスト要因が異なる点を説明している．コントロールレベルは完全にアウトソーシングを行うレベル 1 から，完全にコントロールするレベル 5 まで分類されており，例えばレベル 1 は Amazon Web Services (AWS) [5] などのフルマネージドサービス，レベル 5 は 東日本電信電話株式会社（NTT 東日本）などが提供するハウジングサービス [6] が近い．コントロールレベルによって限られた要因にコストは転嫁され，コントロールをクラウドベンターに大きく委任することで，コスト要因は仮想的な CPU・メモリ・ディスク・ネットワークにまとめられる ．これによりクラウドでは少ない要素で，インフラストラクチャーのコストを試算することができる．

プロバイダーが提供するクラウドサービス，特に多く利用されている IaaS では仮想サーバー（VM）単位で計算資源が提供されており，VM の種類によって性能が異なっている．これに対して，コンポーネントの配置戦略とアプリケーションの各プロファイルを利用することでワークロードと性能の関係が説明されている [7] ．その関係性は構成されるコンポーネントごとのリソース消費と，コンポーネント間の通信パターンから関数化することで表現されており，その際にコンポーネントのリソース需要をブラックボックスとして取り扱う手法を用いている．コンポーネントのリソース需要は計測によって把握され予測に用いられる．

ビジネスの初期段階でシステムの予想を行う場合，「どの程度の利用に応じてシステムのコストはいくらか」という視点で話が行われることが多い．しかしシステムは必ずしもワークロードと単純な線形関係ではなく，構成から直接コストを求めるには複雑で手数がかかる．システムの基礎となる VM はプロバイダーより，タイプ別で提供されており，VM 単位で性能上限があることから，そのコストは連続的ではなく階段状になる．そのためワークロードに対する VM の構成は，VM の選択と，その VM が最低どれだけ必要なのかに影響される．しかしこれだけではアプリケーション全体を直接マトリックス構造として表すことになり，より複雑な構成を考えるには複雑になり過ぎてしまう．また各構成要素の組み合わせを再利用することも難しい．実際のシステムでは数十台，もしくは数百台もの構成が行われることもあり，より理解しやすい構造化を元にコンポーネントの構成を考える枠組みが必要である．

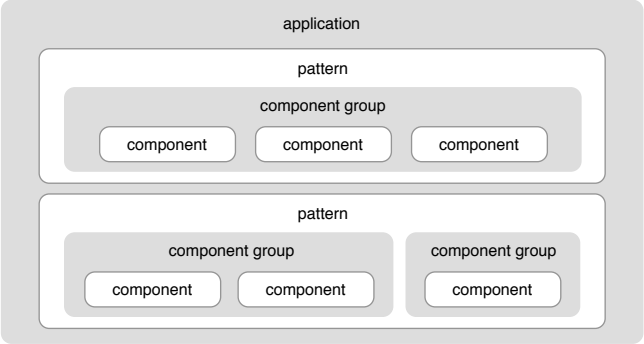

図 1　コストモデルの全体像

再利用性には設計手法として広く利用されているパターン化の概念 [8] がある．これは課題に対するプラクティスでもある抽象化された共通概念をパターンとして捉え，頻繁に訪れる課題に対し適用しようとする考えである．通常，システムのアーキテクチャはクラウドサービスの機能的な違いから一般的には交換可能にはできないが，頻繁に適用されるプラクティスのパターンは経験上利用されている [9] [10] [11]．当研究ではコストモデルにおいても，このパターンの概念がコストの試算にも適用できるのではないかと考え，パターンを前提としたコストモデルの算定手法の提案を行う．

3. コストモデルの提案

パターンを基軸とした構造化を検討するにあたり，その全体像（図 1）を説明する．提案する構造では階層構造を主な考えとし，大きい枠組みから「アプリケーション」「パターン」「コンポーネントグループ」「コンポーネント」に階層化する．インフラストラクチャーをこのような形で階層化するのは，近年のコンテナ技術を利用したオーケストレーションなどでも見られる構造であり，代表的なものでは Kubernetes [12] がある．

コンポーネントは，アプリケーションを分解していく先の構成要素の最小単位である．コンポーネントの集まりは，コンポーネントグループとして表され，個々のコンポーネントは，性能・属性に関して全て同じものであることを想定する．コンポーネントグループはコンポーネントが，他のコンポーネントグループや，パターンへのインプット・アウトプットに接続す

る際に代表となる概念であり，物理的な繋がりを表現するものではない．

パターンは複数のコンポーネントグループから構成される．プラクティスとして再利用できることを期待され，汎用的かつ適度な抽象度を持つ．パターンの組み合わせによってアプリケーションが構成される．アプリケーションは特定のワークロードの処理を行うシステム全体を表しており，パターンのコストモデルの組み合わせによって，そのコストモデルが定義される．

3.1 パターンのコンポーネントとワークロード

次式の有向グラフはパターン内のワークロードの分布を表し，それぞれの要素に対してどれだけの負荷がかかるかを表現する．パターンはコンポーネントグループの組み合わせで構成されることから，最終的なコンポーネントグループ間のワークロード $\hat{\lambda_{ij}}$ は，個別のワークロード比率の分布 $\Lambda_{\mathrm{pattern}}$ から導くことができる．パターンのワークロードを λ_1 にすることで，コンポーネントグループのワークロード $\lambda_j = \lambda_{1j}$ が得られる．

$$\Lambda_{\mathrm{pattern}} = \sum_{k=1} \begin{pmatrix} \lambda_{11} & \dots & \lambda_{1j} \\ \vdots & \ddots & \vdots \\ \lambda_{i1} & \dots & \lambda_{ij} \end{pmatrix}^k = \begin{pmatrix} \hat{\lambda}_{11} & \dots & \hat{\lambda}_{1j} \\ \vdots & \ddots & \vdots \\ \hat{\lambda}_{i1} & \dots & \hat{\lambda}_{ij} \end{pmatrix}$$

コンポーネントグループのワークロードの処理は，最小倍の同一コンポーネントのワークロード処理によって実現する，そのため，コンポーネントタイプの数量 n_{js} は，コンポーネントグループのワークロード処理量 $\hat{\lambda_j}$ と，指定したコンポーネントタイプのワークロード処理量 λ_{js} から導出できる．

$$n_{js} = \left\lceil \frac{\hat{\lambda}_j}{\lambda_{js}} \right\rceil, \quad n \in \mathbb{N}$$

ネットワークについてもワークロードの分布から求めることができる．各コンポーネントグループの入力ワークロードは $\hat{\lambda}$ によって判断できるため，$\hat{\lambda}$ を見ることでネットワーク全体でどれだけのワークロードが発生しているかを把握することができる．

3.2 パターンのコストモデル

ワークロードを処理するコンポーネントのコストを考える場合，そのコンポーネントがどのような制約があるかを考慮した上で検討しなければならない．コンポーネントを含むサーバーは，使用量モデルを適用し動的に計算資源が拡張されるクラウドサービスとは異なり，サーバー側の資源が固定的であるために，CPU・ディスク I/O・メモリ消費量・ネットワーク帯域などが飽和すると，それ以上ワークロードを処理し切れなくなってしまうからである．コンポーネントグループ内のコンポーネントは，役割を等しくするコンポーネントを同一タイプに設定した上で議論を行う．コンポーネントは必要なワークロードに対応する形で，選択されシステムアーキテクチャのコスト予測に用いられる．しかし，前述でも述べたように，コンポーネントのワークロードに対する性能は，何かしら一つの要素が飽和する時点で限界を迎えるため，コンポーネントを含むサーバー全体の性能はワークロードに対して線形に比例しない．そこで，プロファイルから設計者が選択する任意のコンポーネントを基準にすることで，ワークロードに対して最も少ない台数を導き出す．

前述ではコンポーネントの構成は，パターン配下の有向グラフの分布から得られることを述べた．パターンのコストがコンポーネントの構成から得られるとすると，次式の関係が得られる．ここでの $C_{pattern}$ はパターンのコスト，C_{js} は λ_{js} が処理性能の限界値として持つコンポーネントのコストである．

$$C_{pattern} \xrightarrow{\text{component}} \left\{ C_{1s} \left\lceil \frac{\hat{\lambda}_1}{\lambda_{1s}} \right\rceil, \cdots, C_{js} \left\lceil \frac{\hat{\lambda}_j}{\lambda_{js}} \right\rceil \right\}$$

ネットワークおいてもコンポーネント同様，ワークロードに応じてコストが計算される．

$$C_{pattern} \xrightarrow{\text{network}} \begin{pmatrix} C_{11}\hat{\lambda}_{11} & C_{12}\hat{\lambda}_{12} & \dots & C_{1j}\hat{\lambda}_{1j} \\ C_{21}\hat{\lambda}_{21} & C_{22}\hat{\lambda}_{22} & \dots & C_{2j}\hat{\lambda}_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ C_{i1}\hat{\lambda}_{i1} & C_{i2}\hat{\lambda}_{i2} & \dots & C_{ij}\hat{\lambda}_{ij} \end{pmatrix}$$

パターンのコストモデルは，これらコンポーネントタイプのコスト C_{js} とネットワークのコスト C_{ij} の合計とする．コンポーネントのコストモデルは従量制の場合，コンポーネントタイプのワークロード λ_{js} が 1 となり．天井関数が外される．その結果，コンポーネントのコストモデルは，ネットワークのコストモデルと同様にワークロードと線形関係となる．

$$C_{pattern} = \sum_{j=1} C_{js} \left\lceil \frac{\hat{\lambda}_j}{\lambda_{js}} \right\rceil + \sum_{i=1} \sum_{j=1} C_{ij}\hat{\lambda}_{ij}$$

3.3 アプリケーションのコストモデル

アプリケーションはパターンの組み合わせにより，そのコストが予測される．パターンの組み合わせによるコストは，コンポーネントグループの組み合わせと同様にパターンが相互にどう繋がっているかで，ワークロードの分布が変化する．アプリケーションのコストモデルは，前述のパターンのコストの予測式と構造は似通っているが，パターンのコストモデルの試算でコンポーネントタイプ指定が行われるため，新規にコンポーネントタイプ指定を行う必要はない．パターンのコスト試算と同様に，ワークロードの分布を受け止めるパターンごとのコスト C_p と，ワークロードが流れるネットワークのコスト C_{op} を合計したものになるため，そのコストモデルは次式のようになる．

$$C_{application} = \sum_{p=1} C_p\hat{\lambda}_p + \sum_{o=1} \sum_{p=1} C_{op}\hat{\lambda}_{op}$$

3.4 パターン定義

パターンを新規に作成した場合，その作成したパターンを再利用可能な形で定義する．定義したパターンを共有し，再利用することで，その後のコストモデルの試算を行う場面において，開発者及びビジネス主体の負担を軽減することができる．また，パターンに基づくコストモデルは，その元となるアーキテクチャーに強く依存しており，クラウド上にシステム構築を

行う上での重要な知見の一つとなり得る．新規作成したパターンとそのコストモデルは，フォーマット（表 1）に基づき再利用可能な形でパターン化する．パターン定義を行う際には，クラウド利用を前提としたスケーラビリティ・可用性・課金制・マルチテナントの有無などの点に考慮する．なおフォーマットは，クラウドアーキテクチャーのパターン化で既に適用されているもの [8] [9] を参考にしている．作成したパターンは，再利用時にコストモデルを参照され，同様の計算に用いられることが期待される．

表 1　提案するフォーマット

項目	説明
名称	パターン名
サマリー	パターンの概要
図式	パターンの構造の図式化
課題	解決する課題・適用する動機
クラウドでの解決	パターンの説明・クラウドでの解決手順
実装	どのように実装し，解決するかの説明
ワークロード分布	各コンポーネントグループの負荷
コストモデル	パターンを適用した場合のコストモデル
注意点	注意点・注釈などの付加情報

4. ケーススタディ

本章ではデータベースからデータを参照するウェブアプリケーションを想定し、前章で提案したコストモデルについてケーススタディを試みる．パブリッククラウド上にインフラを構築することを想定しており，顧客対象はコンシューマーである．利用するクラウドプロバイダーの提供するサービスは IaaS であり，イントラネットワークの通信料は無料だがグローバルネットワークの通信料はワークロードに伴う通信量に応じて発生する．ロードバランサーは，クラウドプロバイダーよりサービス化されたものが提供されているため，サービス利用者は個別にロードバランサーの機能をサーバー上に構築しない．またロードバランサーも，利用量に応じて課金される従量課金を採用している．IaaS で提供される仮想サーバーは，CPU・メモリ・ディスク・ネットワークのそれぞれの性能の組み合わせによって価格付けされており，それぞれの要素を自由に設定することはできない．それぞれ指定された性能の組み合わせがサーバータイプとして設定され，提供されているため，利用者はその中から必要なサーバーを選択する．

4.1 システム構成

ケーススタディに応じたシステム構成として図 2 の構成を定める．この構成はウェブシステムでは一般的な構成であり，ロードバランサー・ウェブサーバー・データベースの 3 つの要素から構成される．各要素間の接続は均等に分散され，ロードバランサーを経由して，クライアントが当ウェブシステムからデータを取得するのが，システムの提供する機能である．データベースはリードレプリカ構成を想定する．

当システムは，読み込みのみが非常に大きい状況を想定しており，書き込みが多くを占めるサービスである．具体的には，

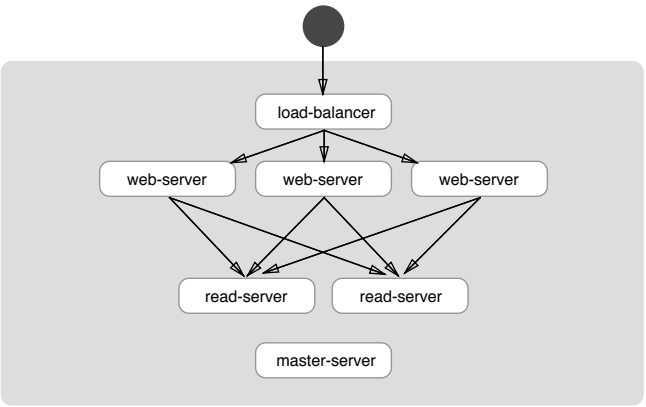

図 2　システム構成

クライアントからのコンテンツ取得，例えば位置情報取得サービスなどを想像すると理解しやすい．スケールを行う状況では，このウェブサーバーとリードレプリカが水平に拡張されることを想定する．ロードバランサーとリードレプリカ構成のマスターサーバーに関してはスケールアウト/アップは想定しない．議論をシンプルにするために当シナリオケースでは，モニタリングやスケーリングなどの付加的な設定は設けない．

4.2 ケーススタディにおけるパターンとコンポーネント

ケーススタディは，大きく分けて 2 つのパターンから構成されていると考えられる．1 つ目はロードバランサーからコンポーネントに負荷が分散されている構成である．この構成は入力で与えられるワークロードの負荷を，ロードバランサー配下のコンポーネントに分散させる役割を持ち「マルチサーバーパターン」とする．2 つ目は読み込み専用データベースを用意することで読み込み性能を向上させるリードレプリカ構成である．この構成はクラウドサービスだけでなく，多くのデータベース製品で採用されている構成でもあり「リードレプリカパターン」とする．

マルチサーバーパターンはロードバランサーの機能と複数のコンポーネントを持つという機能に分割して考えることができる．そのため，コンポーネントグループ間のワークロードの関係性を式で表すと以下のように表される．

$$\Lambda_{\text{multi-server}} = \sum_{k} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}^{k} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

マルチサーバーパターンのコンポーネント構成では，入力ワークロード $\lambda_{\text{multi-server}}$ とロードバランサー・複数サーバーのワークロードの関係から，各コンポーネントグループ内でどのようなコンポーネント単位を設定するかで関係性が定まる．ロードバランサーを従量制のクラウドプロバイダーの提供するサービス化されたコンポーネントグループで実現すると，各ワークロードとコンポーネントの関係は次式となる．

$$\hat{\lambda}_{\text{load-balancer}} = \lambda_{\text{multi-server}}$$

$\hat{\lambda}_{\text{load-balancer}}$ は，マルチサーバーパターンの入力ワークロード $\lambda_{\text{multi-server}}$ の 1 単位に対応するロードバランサー機能への

ワークロードである．ロードバランサーは従量制サービスとして提供されているため，$\lambda_{\text{multi-server}}$ にワークロード比率を掛け合わせた値がロードバランサーのサービスが処理するワークロードとなる．

次にロードバランサーから分散されたワークロードを受け取るコンポーネントグループ内の各コンポーネントで受け取る関係を次式に表す．

$$\hat{\lambda}_{\text{servers}} = \lambda_{\text{server}} \left\lceil \frac{\lambda_{\text{multi-server}}}{\lambda_{\text{server}}} \right\rceil$$

$\hat{\lambda}_{\text{servers}}$ は，ロードバランサーから分散されたワークロードを受け取るコンポーネント群（サーバー群）である．コンポーネント群にかかるワークロードが，設定したサーバータイプで処理できるワークロードが複数集まって処理されることを表している．

リードレプリカパターンは，マスターとリードレプリカの 2 つのコンポーネントグループによって構成される．当ケーススタディで想定しているワークロードは，データベースからのデータの読み込みを想定しているため，マスターに該当するコンポーネントグループに対してはワークロードが発生していない点に注意が必要である．

$$\Lambda_{\text{read-replica}} = \sum_k \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}^k = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

マルチサーバーパターンと同様にリードレプリカパターンにおいても，コンポーネントグループの構成からワークロードの分布を得ることができる．ワークロードに対応する分布であるため，マスターとなるコンポーネントからリードのコンポーネントへの反映などの負荷は含まれていない点に注意が必要である．そのため，マスターとなるコンポーネントへのワークロードの見積もりは 0 となる．

$$\hat{\lambda}_{\text{read-servers}} = \lambda_{\text{read-server}} \left\lceil \frac{\lambda_{\text{read-replica}}}{\lambda_{\text{read-server}}} \right\rceil$$

$$\hat{\lambda}_{\text{master-servers}} = 0$$

4.3 アプリケーションのコストモデル

ロードバランサーとコンポーネントのコストモデル，さらにパターン内でのネットワークにかかるコストを足し合わせることでマルチサーバーパターン全体のコストモデルを定義することができる．マルチサーバーパターンが適用されるシステム箇所は，イントラネットワークが想定されており，その基本的な価格付けを考慮する必要がある．当ケーススタディではイントラネットワークにかかるコストは，無料が想定されているため，パターン内のネットワークコストはワークロードの大小に関わらず 0 となる．そのため，パターンのコストとして試算できるモデル式からネットワークコストを省略することができ，次式でそのコストモデルを定義することができる．

$$C_{\text{multi-server}} = C_{\text{load-balancer}} \lambda_{\text{multi-server}} + C_{\text{server}} \left\lceil \frac{\lambda_{\text{multi-server}}}{\lambda_{\text{server}}} \right\rceil$$

リードレプリカはマスターとリードの 2 種類のコンポーネントグループから構成されるためリードとマスターのコストモデルを足し合わせることで，マルチサーバーパターンと同様にリードレプリカパターンのコストモデルを定義することができる．パターン内ネットワークのコストモデルも同じく省略することができ，そのモデル式は次式に表せる．マルチサーバーパターンと異なる点は，マスターコンポーネントグループがサーバータイプに依存はするもののワークロードの規模に対して固定的であるため台数を 1 台に固定することでコストモデルを表現することになる．

$$C_{\text{read-replica}} = C_{\text{read-server}} \left\lceil \frac{\lambda_{\text{read-replica}}}{\lambda_{\text{read-server}}} \right\rceil + C_{\text{master-server}}$$

得られたパターンのコストモデルを組み合わせることで，アプリケーション全体のコストモデルを定義付けることができる．パターン内のコンポーネント構造と同じく，その繋がりとワークロードの比率を定義付けることで，アプリケーションのワークロードを基準とした，パターンごとのワークロード分布を試算することができるが，パターン間のワークロードの関係を考えると，マルチサーバーパターンからリードレプリカパターンへのワークロードは，アプリケーションの仕様に依存しており，マルチサーバーパターンへの入力と等しい比率であるとは限らないことが分かる．このような状況の場合，次式の λ_{23} のように変数を設定し，計測によって比率を得ることに期待する．

$$\Lambda_{\text{application}} = \sum_k \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & \lambda_{23} \\ 0 & 0 & 0 \end{pmatrix}^k = \begin{pmatrix} 0 & 1 & \lambda_{23} \\ 0 & 0 & \lambda_{23} \\ 0 & 0 & 0 \end{pmatrix}$$

アプリケーションのコストモデルは，各パターンのコストモデルとパターン間のネットワークのコストモデルを足し合わせることで求めることができる．ネットワークのコストモデルに λ_{23} が含まれていないのは，λ_{23} のリンクがイントラネットワークに属するからである．イントラネットワークの流量にかかるコストは 0 である想定のため，コストモデルの式から省略している．次式は当ケースのワークロードに対するコストモデルである．ウェブサーバーのサーバータイプ・リードサーバーのサーバータイプ・マスターサーバーのサーバータイプ・仕様に基づくウェブサーバとリードサーバー間のワークロード分布の関係から定義されていることが分かる．

$$\begin{aligned} C_{\text{application}} =& C_{\text{load-balancer}} \lambda_{\text{application}} + C_{\text{web-server}} \left\lceil \frac{\lambda_{\text{application}}}{\lambda_{\text{web-server}}} \right\rceil \\ &+ C_{\text{read-server}} \left\lceil \frac{\lambda_{23} \lambda_{\text{rapplication}}}{\lambda_{\text{read-server}}} \right\rceil + C_{\text{master-server}} \\ &+ C_{\rightarrow \text{multi-server}} \lambda_{\text{application}} \end{aligned}$$

4.4 パターンの再利用

パターンを定義する際に注意が必要な点は，フォーマットの注意点項目への記述を漏らさないようにすることである．これは、そのパターンが課題と状況に合致していたとしても，注意点で述べられる点で条件を満たさないことがしばしばある事を考慮した注意点である．パターン利用者の立場であっても同様

表 2　マルチサーバーパターン

項目	説明
名称	マルチサーバーパターン
サマリー	ロードバランサーから複数サーバーに負荷分散する
図式	図 3 参照
課題	サーバーの可用性・冗長性・負荷の増大に対応する
クラウドでの解決	クラウドサービスで提供されるロードバランサーを用いて，仮想サーバーに対して負荷を振り分ける
実装	(例：AWS) ELB を利用し，EC2 を振り分け先に設定する．さらに ELB から EC2 のにヘルスチェックを行うことで正常稼働を確認し，障害時にサーバーを自動的に切り離すことで正常動作を維持できる
ワークロード分布	$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$ load-balancer, compoonent
コストモデル	$C_{\text{load-balancer}}\lambda_{\text{multi-server}} + C_{\text{server}}\left\lceil \frac{\lambda_{\text{multi-server}}}{\lambda_{\text{server}}} \right\rceil$
注意点	冗長性や可用性を得るためには最低でも N+1 のサーバーを適用しなければならない．また負荷分散の仕方によってはセッションの共有が困難な場合がある

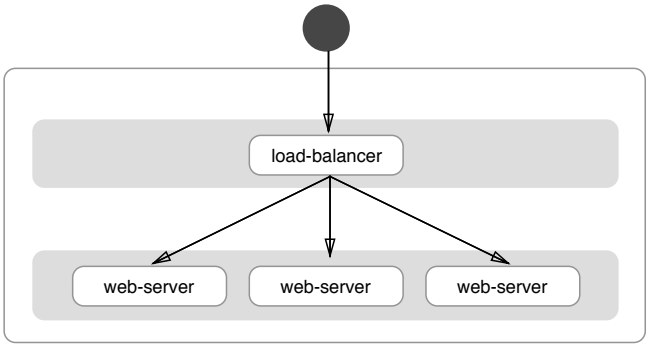

図 3　マルチサーバーパターン図

で，パターンを再利用する際には注意点を理解した上での利用を行うべきである．例えば，当ケースで定義を行うリードレプリカパターンの一貫性に関する注意事項などは，不慣れなうちは見逃しがちであり，気付いた時には運用・開発で大きな負担となってしまっている状況は珍しくない．

5. 結　　論

当研究では，パターンの概念を用いたコストモデル算出手法と，再利用性のためにパターンを残す必要性について議論し，1) ワークロードからコストを算出する関係を数式化する方法と，2) システムの利用形態から簡易にコストを導出する方法，を提案した．加えてパターンの再利用性の観点から，パターンに基づくコストモデルの定義付けについて議論を行った．このような手段を講じることで，ビジネスの初期段階で簡易にコストを試算することができ，また異なるシステムにおいてもコストモデルの知見を再利用できる．

パターンに限らず，プラクティスを再利用する，もしくはブラックボックスとして扱えるようにするアプローチの有用性は経験上知られており，同じソフトウェアに関わるものではアルゴリズムやそれを実装したライブラリなどを取り巻く環境など前例は多い．さらにクラウドコンピューティングの台頭によって，パターンで表現するような機能がサービスとして展開されている状況も増えており，その料金体系がオンデマンドの利用に応じるものが多いことからも，これからますます組み合わせの知見からコストの予測を行う場面は増えると思われる．当研究の成果がそのような場面の一助になれば幸いである．

文　　献

[1] P.M. Mell and T. Grance, “The NIST definition of cloud computing,” Technical report, National Institute of Standards and Technology, 2011. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing,” Commun. ACM, vol.53, no.4, pp.50–58, 2010.
[3] 株式会社日経 BP，“記者の眼 - 知らない間に高額請求、クラウド破産にご用心：ITpro,” http://itpro.nikkeibp.co.jp/atcl/watcher/14/334361/021500780/, 2017. Accessed: 2018-01-21.
[4] A. Simonet, A. Lebre, and A.C. Orgerie, “Deploying distributed cloud infrastructures: Who and at what cost?,” 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), pp.178–183, 2016.
[5] Amazon, “Amazon Web Services,” https://aws.amazon.com/, 2018. Accessed: 2017-12-08.
[6] 東日本電信電話株式会社，“ハウジングサービス | NTT 東日本データセンター | 法人のお客さま | NTT 東日本,” https://www.ntt-east.co.jp/business/solution/datacenter/housing.html, 2018. Accessed: 2018-01-21.
[7] C. Stewart and K. Shen, “Performance modeling and system management for multi-component online services,” Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, pp.71–84, 2005.
[8] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck, “An architectural pattern language of cloud-based applications,” Proceedings of the 18th Conference on Pattern Languages of Programs, pp.2:1–2:11, 2011.
[9] C.D.P.A. CDPA, “AWS-CloudDesignPattern,” http://aws.clouddesignpattern.org/, 2017. Accessed: 2017-12-08.
[10] Microsoft, “Cloud Design Patterns,” https://docs.microsoft.com/en-us/azure/architecture/patterns/, 2017. Accessed: 2017-12-08.
[11] Cloud Computing Patterns, “Cloud computing patterns,” http://www.cloudcomputingpatterns.org/, 2017. Accessed: 2017-12-14.
[12] T.L. Foundation, “Kubernetes,” https://kubernetes.io/, 2018. Accessed: 2018-01-03.