クラウドコンピューティングにおける
コスト推定アプローチ

２０２３年　３月

青嶋　智久

クラウドコンピューティングにおける
コスト推定アプローチ

青嶋　智久

システム情報工学研究科

筑波大学

２０２３年　３月

概要

今日クラウドコンピューティングは広く一般に利用されており、研究者からの注目も高まり続けている。クラウドコンピューティングは、ネットワークを介してサービス (クラウドサービス) を提供するコンピューティングモデルであり、クラウドサービス利用者はネットワーク経由でいつでも必要なクラウドサービスのコンピューティングリソース (クラウドリソース) を利用できる。クラウドコンピューティングには IaaS、PaaS、SaaS などのサービスモデルがあり、クラウドサービス利用者は目的に合ったサービスモデルを選択することで、構築するシステムの管理領域や QoS、システムのコスト特性をコントロールできる。

クラウドコンピューティングは仮想マシンやコンテナなどの仮想化技術により支えられており、近年ではサーバの概念を持たないサーバレスコンピューティングも積極的に活用されている。仮想化技術によりクラウドサービスプロバイダは柔軟なクラウドリソースの提供ができ、クラウドサービス利用者は必要なときに必要なだけクラウドリソースを利用できる。またクラウドサービス利用者はスケーリングの自動化をすることで、クラウドリソースの割当最適化ができる。スケーリングの自動化は、利用中のクラウドリソースをモニタリングし、クラウドリソースの過不足に応じてクラウドリソースを割り当てることで実現する。クラウドサービスのサービスモデルによりクラウドリソースのスケーリングの追従性に差があり、たとえばサーバレスサービスでは仮想マシンベースのサービスと比べて高いスケーリングの追従する。このようなクラウドコンピューティングの特徴は、クラウドサービス利用者のビジネス観点からも有用である。

一般にクラウドサービスの課金体系は従量課金を基本とする。必要なときに必要なだけリソースを利用するということは、クラウドリソースのコストを最適化できることを意味する。また従量制の料金形態はクラウドサービス利用者にとって、システム構築の初期に必要な資本コストを運用コストに転換する。このコンセプトの違いにより、クラウドサービスの利用者は初期コストを削減でき、ビジネスを素早く立ち上げることができる。

クラウドサービス利用者はコスト管理ツールを利用することで、現時点でどれだけのクラウドリソースを利用しているのか、どれだけのコストがシステム全体でかかっているのかを把握できる。しかしコスト管理ツールによるコスト把握は、主に現時点のシステムの

クラウドリソースの利用状況を確認することに主眼を置いており、 将来のクラウドサービスのコストを把握するためには、対象のシステムでどれだけのクラウドリソースを利用するかを推測する必要がある。従来のクラウドリソース利用量の推定方法には、対象システムのアーキテクチャから解析的にクラウドリソース利用量を推測する方法と、シミュレーションを用いてクラウドリソース利用量を推測する方法がある。しかしこれらの手法はクラウドサービスプロバイダの利用に主眼を置いた手法であることが多く、クラウドサービス利用者が知るのが難しいクラウドサービス内部の理解を求めるなど、ビジネスの立ち上げ時期に手早くクラウドリソース利用量を概算したい状況などでは利用しにくい部分がある。

本研究ではクラウドサービス利用者の立場から、解析的な手法やシミュレーションツールを使いこなす有識者がいなくても、ある程度許容可能なコストの概算を得るためのコスト推定手法を提案する。提案手法は、クラウドサービス利用者が簡便にコストを概算できることを重視しており、クラウドサービス利用者がシステム設計前の段階で将来のシステムコストを推定できるようになることを目標とする。

本論では、はじめに当提案手法の基本的な考え方、クラウドリソース利用量の推定プロセス、そして対象期間におけるアプリケーション全体のコストを算出する手続きを説明する。提案手法は対象システムのアーキテクチャをモデル化し、指定した計算手順に従うことで、対象システム内のコンポーネント間の関係性を導き、システム負荷に応じたアプリケーション全体のコストを求めるアプローチをとる。システムのアーキテクチャのモデル化では有向非巡回グラフ (Directed Acyclic Graph, DAC) と行列を用いて、隣接する 2 つのコンポーネント間の関係を表現する。またシステムを構成するコンポーネントごとのクラウドリソース利用量の推定では、モデル化した行列の演算により、システム負荷に応じた各コンポーネントで必要なクラウドリソースを推定する。そして推定を積み上げることで、対象期間におけるシステム全体のコストを求める。

さらに本論では実際のクラウドサービス上にシステムを構築し、提案するコスト推定手法を適用することで、実際にクラウドサービスを使って構築するシステムで、提案手法をどのように適用してコスト推定手順を進めていくかを実例する。サーバーベースのクラウドリソースを用いたシステムと、サーバレスベースのクラウドリソースを用いたシステムをそれぞれケーススタディとして構築して、当該システムのアーキテクチャのモデル化、クラウドリソース利用量の推定、アプリケーション全体のコストの算出プロセスを説明

する。ケーススタディでは広く一般に利用されている実際のクラウドサービスを用いる。サーバーベースのケーススタディでは、仮想マシンをベースにウェブアプリケーションとデータベースから構成されるアプリケーションを用い、当該システムに変動するシステム負荷をかけ、必要なクラウドリソースがシステム負荷に応じて割り当てられるかを検証した。そして仮想マシンサイズの粒度により推定誤差が生じるものの、許容可能なコストの概算を得られることを示した。サーバレスベースのケーススタディでは、サーバレスサービスを仮想マシンの代わりに用い、サーバーベースのケーススタディと同様の構成のシステムを構築し、変動するシステム負荷に応じてクラウドリソースが割り当てられるかを検証した。。そして許容可能なコストの概算を得られる可能性を示すとともに、サーバレスサービスで構築したシステムでは仮想マシンで構成されたシステムと比べて、よりシステム負荷に追従したシステムのコストを推定できることを示した。

目次

図目次

表目次

第 1 章

序論

クラウドコンピューティング (Cloud Computing) は、今日広く一般に利用されており、研究者からの注目も高まり続けている。近年においてもデジタル庁が Amazon Web Services, Inc. の Amazon Web Services (AWS) [Ama22i]、Microsoft Corporation の Microsoft Azure (Azure) [Mic22]、Google LLC の Google Cloud Platform (GCP) [Goo22]、Oracle Corporation の Oracle Cloud Infrastructure [Cor22p] を日本の行政機関が共同利用するガバメントクラウドに指定したことが記憶に新しい [デジ 22]。

クラウドコンピューティングとは、ネットワークを介してコンピューティングリソース (クラウドリソース) をサービス（クラウドサービス）として提供するコンピューティングモデルである [Nat22]。クラウドコンピューティングは、オンデマンドセルフサービス、幅広いネットワークアクセス、リソースプーリング、迅速な弾力性、測定サービスなどの特徴を備えており、クラウドサービスの提供者 (クラウドサービスプロバイダ) はクラウドリソースを操作するための管理画面および API（Application Programming Interface）を提供している。クラウドサービス利用者はクラウドサービスの管理画面あるいは API を利用して、ネットワーク経由でいつでもシステムに必要なリソースを構成（プロビジョニング）できる。

クラウドコンピューティングでは、IaaS (Infrastructure as a Service)、PaaS (Platform as a Service)、SaaS (Software as a Service) など、様々なサービスモデルが提供されており、利用者は自身の目的に合ったサービスモデルを利用できる。それぞれサービスモデルではクラウドサービスプロバイダより提供されるクラウドリソースの抽象度が異なり、クラウドサービスのインフラストラクチャの管理領域の違いや、スケーリング特性、コスト

モデルなどの違いがある。クラウドサービス利用者は、自らのニーズに合ったサービスモデルを選択することで、構築するシステムの管理領域や QoS (Quality of Service)、システムのコスト特性をコントロールできる。

クラウドコンピューティングの各種特性は仮想化技術により支えられている。クラウドコンピューティングで用いられる仮想化技術には、たとえば仮想マシン (Virtual Machine, VM) やコンテナ (Container) がある。クラウドサービスプロバイダは仮想化技術により、物理インフラストラクチャとは切り離した形で仮想化されたリソースを大量に準備 (リソースプール) できる。クラウドサービスプロバイダの管理するデータセンターが十分に大きいとき、クラウドサービス利用者側は、まるで無限のクラウドリソースをクラウドサービスのリソースプールから割り当てるような使い方ができる。

このような柔軟なプロビジョニングを可能とするクラウドコンピューティングの特徴は、ビジネスの観点からも有利である。なぜならば、柔軟なプロビジョニングが可能であるということは、クラウド利用者のコスト管理を容易にするからである。オンプレミスでシステムを構築する場合、本来であればシステム負荷のピークと満たすべき QoS (Quality of service) を基準に、物理インフラストラクチャを整備しなければならない。しかしクラウドコンピューティングであれば、必要なクラウドサービスを利用する同様に、不必要なクラウドサービスを利用しないことで、システムを構成するクラウドリソースのコストを最適化できる。

元来コンピューングリソースは物理的な機器であり、コンピューティングリソースのコストの概念は固定資産の価値に基づくものであった。一方でクラウドコンピューティングはサービスで提供されており、必要なものだけを使う、利用した分だけ支払う (Pay-As-You-Go) 従量課金を基本としている。これは物理的なインフラストラクチャに係るコストをクラウドサービスのコストに転嫁することにより実現している。これによりクラウドサービス利用者は“資本コストを運用コストに変換する”ことができる [AFG+10]。このコンセプトの違いによりクラウドサービス利用者は初期コストを削減し、ビジネスを開始するスピードを早めることができる。

またサービスモデルによるクラウドサービスの抽象度の違いによって、クラウドサービスプロバイダは様々な課金モデルを生み出している。クラウドサービスプロバイダは自らのクラウドサービスの課金モデルを任意に定義できる。一般にクラウドサービスのサービスモデルの抽象度が高ければ高いほど、物理インフラストラクチャから切り離された課

金モデルが設定している。たとえば抽象度の低い IaaS の場合は、サーバの台数やネットワーク転送量、ストレージの容量など、物理的なコンピューティングリソースを基準とした課金モデルを定義することが多い。一方で抽象度の高い SaaS の場合は、API リクエスト数やユーザー数など、よりクラウドサービスのサービスモデルに則した課金モデルを定義することが多い。

クラウドサービスの利用において、過不足なくクラウドリソースを利用するためには、スケーリングの自動化が不可欠である。利用しているクラウドリソースをモニタリングし、必要なクラウドリソースが不足しているときはクラウドリソースを自動的に割り当て、クラウドリソースに余剰があるときに自動的に解放することで、過不足ないクラウドリソースの割り当てが実現できる。クラウドサービスのスケーリング特性によりクラウドリソースのスケーリングの追従性に差があり、たとえばサーバレスサービス (Serverless Service) では仮想マシンベースと比べて高いスケーリングの追従性を有する。

クラウドサービスで割り当てたクラウドリソースの利用状況および掛かりうるコストは、クラウドサービスプロバイダが提供するコスト管理ツールで確認できる。クラウドサービス利用者はコスト管理ツールを用いて、現時点でどれだけのクラウドリソースを利用しているのか、どれだけシステム全体でクラウドリソースのコストを請求されるのかを把握できる。しかし注意すべき点としてはあくまで現時点におけるシステムの状況を確認できることである。これからクラウドサービスを利用しようとしている潜在的なクラウドサービス利用者が将来のクラウドサービスのコストを把握するためには、対象のシステムでどれだけのクラウドリソースを利用するかを推測する必要がある。

将来掛かりうるクラウドリソース利用量を把握するためには、どのようなシステムを構築するのか、そのシステムではどのクラウドリソースをどれだけ利用するのかを推定する必要がある。従来のクラウドリソースの推定法として、クラウドサービス上に展開するシステムのアーキテクチャから解析的にクラウドリソース利用量を推測する方法と、シミュレーションを用いてクラウドリソース利用量を推測する方法がある。しかしこれらの手法はクラウドサービスプロバイダを主眼を置いた手法であることが多く、クラウドサービス利用者の視点で見ると、いささか複雑で使いこなすには高い知見を必要とすることが多い。また手法によっては、クラウドサービス利用者からは知り得ない設定項目を必要とすることもある。

実際にクラウドサービスを利用する現場では、システム構成は大まかに定まっているも

のの、システムリリースまでの時間が少なく、解析的な手法やシミュレーションツールを使いこなす有識者がいないことも多い。また多くのクラウドサービスの関係者は、クラウドサービスの提供者ではなく利用者であり、クラウドサービス利用者のコスト推定を困難にしている。しかしビジネスにおけるシステム構築では、システムに係るコストの見積もりも、プロジェクト初期に考慮すべき課題の 1 つである。そこで本論ではクラウドサービス利用者の立場から、システムリリースまでの時間が少なく、解析的な手法やシミュレーションツールを使いこなす有識者がいなくても、ある程度許容可能なコスト推定手法を提案する。当コスト推定手法では、クラウドサービスの特性を数多く前提に置くことで、簡易かつ形式的なコスト推定プロセスを実現している。

本論では次の流れに従って議論を進める。第 2 章では、クラウドコンピューティングの定義や特徴を議論し、クラウドサービスのコスト見積もりに適用可能なアプローチを取り上げ、クラウドサービスのコストを見積もる際の課題を議論する。第 3 章では、本論で提案するコスト推定手法の基本的な考え方、クラウドリソース利用量の推定プロセス、そして最終的なコスト推定の算出方法を説明する。第 4 章では、仮想マシンベースのシステムを構築して第 3 章のコスト推定手法を適用し、実際にどのようにコスト推定プロセスを進めていくか説明する。そして当システムに負荷をかけ、当コスト推定手法により得られるクラウドリソース利用量の推定量と実際のクラウドリソース利用の間にどのような差があるか議論する。第 5 章では、ケーススタディの対象をサーバレスコンピューティングに広げ、サーバレスサービスで構築したシステムにおいて当コスト推定手法が効果的であることを示す。最後に第 6 章では、本論の研究成果により得られた知見をまとめる。

第 2 章

クラウドコンピューティングとそのコスト推定手法

クラウドサービスを用いるシステムのコストを見積もるには、システムの特性を考慮する必要がある。クラウドサービスを用いるシステムにおいて、システムのコストがどのように分析されてきたかを明らかにするため、クラウドコンピューティングのコスト推定に関連する研究をサーベイした。クラウドサービスを用いたシステムのコスト分析の関連分野に、クラウドサービスのコスト因子の分析、解析的な手法でクラウドリソース利用量を推定するアプローチ、シミュレーションに基づくクラウドリソース利用量を推定するアプローチなどがある。本章ではクラウドコンピューティングの概要と特徴をサーベイしたのち、コスト見積りに関する関連研究をサーベイする。

本章では、まずクラウドコンピューティングの定義や特性、サービスモデルについて説明する。次にクラウドサービスのコスト計算のプロセスと、先行研究におけるクラウドサービスのコスト計算の視点を説明する。最後に実際のクラウドサービスを用いるシステムのコスト推定にて、考慮するべきビジネス面での視点、たとえばクラウドサービスプロバイダによるコスト計算の支援ツールの提供や、クラウドサービスを用いるシステムのコスト推定において注目すべきポイント、そして先行研究における課題と本論文における考えを説明する。

2.1 クラウドコンピューティングの定義

クラウドコンピューティング (Cloud Computing) とは、サーバやネットワーク、ストレージなどのリソースをネットワーク経由でオンデマンドに提供するコンピューティングモデルである [Nat22, Kim09]。クラウドコンピューティングの利用者から見て、ネットワークの向こう側にまるで雲 (Cloud) のようにコンピューティングリソースがあることからクラウドコンピューティングと呼ばれている。クラウドサービスを提供する組織をクラウドサービスプロバイダと呼ぶ。

クラウドサービスプロバイダは自らのクラウドサービスの課金モデルを定義し、クラウドサービスのクラウドリソースやアプリケーションの開発や保守、運用を行う。クラウドコンピューティングの技術は、グリッドコンピューティング、ユーティリティコンピューティング、オートノミックコンピューティングに基づいている [MAJ+14]。

グリッドコンピューティング (Grid Computing) とは、複数のコンピューティングリソースを組み合わせ、協調的に処理を行うことで、大規模な処理を効率的に行うコンピューティングモデルである [JBF+05]。グリッドコンピューティングでは、コンピューティングリソースはユニット化され、共有のリソースプールで管理される。必要に応じてリソースプールから必要なコンピューティングリソースの割り当てをスケジューリングすることで、複数の計算処理が走るときに全体として効率的なコンピューティングリソースの利用を実現する。

ユーティリティコンピューティング (Utility Computing) とは、電気やガス、水道のようにコンピューティングリソースを利用した分だけ料金を支払うコンピューティングモデルである [YdAY+06]。ユーティリティコンピューティングでは個々の利用者に対して、共通のコンピューティングサービスを提供する。そのためユーティリティコンピューティングのサービス提供者は、より多くのサービス利用者にコンピューングサービスを提供でき、当コンピューティングサービスの運用コストを最小限に抑えることができる。またサービス利用者にとっても必要な分だけコンピューティングリソースのコストを支払えばよいため、不要なコンピューティングリソースを維持するためのコストを抑えることができる。

オートノミックコンピューティング (Automomic Computing) とは、人間による高度

なガイダンスに従って自律的にコンピュータが自らを管理するコンピューティングモデルである [PH04]。オートノミックコンピューティングは自律神経系などの生体システムに着想を得ている。オートノミックコンピューティングでは、計算性能、耐障害性、信頼性、セキュリティなどの要件を、人手を介さずに満たすことができる。そのためオートノミックコンピューティングでは、システム管理者のシステムへの介入を最小限に抑えながら、環境の変化に応じてシステムを再構成し、常に最適に近い性能でオペレーションを維持できる。

クラウドコンピューティング技術の基礎には仮想化技術がある [MAJ+14]。クラウドコンピューティングを支えるグリッドコンピューティング、ユーティリティコンピューティング、オートノミックコンピューティングの技術は、仮想化技術により実現されている。仮想化 (Virtualizatoin) とは、ハードウェアやソフトウェアのパーティション分割を用いてコンピューティングリソースを結合または分割し、1 つまたは複数の実行環境を提供する技術である [CB05]。仮想化の主な目的は、コンピューティングリソースを拡張可能、効率的、経済的にすることでワークロードを効率的に処理することである。仮想化技術には、オペレーティングシステム (Operating System, OS) 仮想化、ハードウェアレベル仮想化などがある。仮想化技術によって、クラウドサービスプロバイダは仮想化されたコンピューティングリソースをリソースプールから柔軟にクラウドサービス利用者に提供でき、クラウドサービス利用者は必要に応じてクラウドリソースを利用できる。

クラウドコンピューティングのコンピューティングリソースは仮想化されており、その実態は相互接続されたコンピュータの集合体である [HT12]。クラウドサービスプロバイダから見たとき、クラウドコンピューティングはコンピューティングリソースを仮想化し、抽象化することで、クラウドリソースをクラウドサービス利用者に動的に提供できる利点がある。そして多くのクラウドサービス利用者の需要を受け止めることで、リソースプールを維持するためのコストを安定させることができ、規模の経済性を追求できる [MBS12]。

今日の多くのクラウドサービスプロバイダは、大規模なデータセンターとクラウドリソースを管理するためのインフラストラクチャを所有している。たとえば実際に大規模なデータセンターを有しているクラウドサービスプロバイダおよびクラウドサービスには、Amazon Web Service, Inc. の Amazon Web Services (AWS) [Ama22i]、Microsoft Corporation の Microsoft Azure [Mic22]、阿里云の Alibaba Cloud [阿里 22b]、Google

Inc. の Google Cloud [Goo22] がある。多くのクラウドサービスプロバイダでは、クラウドサービスを構成する様々なコンポーネント (クラウドリソース) を日々開発しており、現在においてもクラウドサービスプロバイダの投資は積極的である [Res22]。

クラウドサービス利用者はクラウドサービスプロバイダとサービスレベル水準合意 (Service Level Agreement, SLA) に基づき契約を行いクラウドサービスを利用する。そしてクラウドサービス利用者はクラウドサービスプロバイダが提供するクラウドリソースを必要に応じて利用する。クラウドサービス利用者はクラウドサービスをオンデマンドに利用することで、本来資本コストだったコンピューティングリソースのコストを運用コストに変換できる [AFG+10]。これは多額な先行投資の必要性がクラウドサービス利用者のビジネスチャンスを妨げることを防ぎ、多くのクラウドサービス利用者が情報技術のビジネスメリットを享受できることを意味する。

まとめるとクラウドサービス利用者にとって、クラウドコンピューティングを利用することは以下のようなメリットがある [MLB+11]。

- クラウドサービスロバイダにより、クラウドサービスに必要なすべてのコンピューティングリソース (物理サーバ、ネットワーク、ストレージ、ソフトウェア、仮想サーバ、データセンター、消費電力など) は管理されており、クラウドサービス利用者はクラウドサービスプロバイダが提供するクラウドサービスに接続するだけでクラウドリソースをオンデマンドで利用できる。そのためクラウドサービス利用者は物理サーバ、ネットワーク、ストレージ、ソフトウェア、仮想サーバ、デンタセンター、消費電力などのコンピューティングリソースに多額の先行投資を行う必要がない。またこれらコンピューティングリソースを管理、運用するためのシステム管理者、ソフトウェア並びにハードウェア技術者、オペレーターなどの人的リソースの管理も不要である。
- リソースプールにより、クラウドサービス利用者はクラウドサービスおよびリソースプールが許す限り必要なクラウドリソースをいくらでも利用できる。これはクラウドサービス利用者が構築するシステム負荷に応じて、システムを構成するクラウドリソースを十分に割り当てることを意味する。
- 本来資本コストだったコンピューティングリソースのコストを運用コストに変換できることにより、クラウドサービス利用者がクラウドリソースにかける費用が、ク

ラウドサービス利用者が独自にコンピューティングリソースを 1 から用意する (オンプレミス) 場合に比べて低くなる可能性がある。オンプレミスでコンピューティングリソースを用意する場合、季節的な需要の変化や、一時的な注目による負荷など、日々変動するシステム負荷のピークに合わせて物理的なコンピューティングリソースを用意する必要がある。そのためシステム負荷のピーク以外では、オンプレミス上のコンピューティングリソースはクラウドリソースに比べて費用対効果が低くなる可能性がる。

- クラウドコンピューティングでは、クラウドサービス利用者はネットワークを通じてクラウドーサービスを利用することができる。そのためクラウドサービス利用者は法的な制約を除けば、地理的な制約、ネットワークの制約を受けずにクラウドリソースを調達できる。多様化および変化の早い現在のビジネス環境においては、マーケットや組織の状況変化に合わせて柔軟にコンピューティングリソースを調達できることは十分に魅力的である。

2.2 クラウドコンピューティングの特徴

クラウドコンピューティングはいくつかの固有な特徴を持つ。米国立標準技術研究所 (National Institute of Standards and Technology, NIST) は、2011 年にクラウドコンピューティングを特徴付ける性質として 5 つの基本特性を定義した [MG+11]。

オンデマンドセルフサービス (On-demand self-service)　クラウドサービス利用者はクラウドサービスプロバイダから提供されるクラウドリソースを、クラウドサービスプロバイダとの人的なやり取りを必要とせずにセルフサービスで利用できる。

幅広いネットワークアクセス (Broad network access)　クラウドサービス利用者はクラウドサービスの機能を、携帯電話やタブレット、ノートパソコンなどの様々なクライアントから標準的なネットワーク接続で利用できる。

リソースプーリング (Resource pooling)　クラウドサービスプロバイダはクラウドリソースをプールし、それぞれのクラウドサービス利用者にクラウドリソースを動的に割り当てる。

迅速な弾力性 (Rapid elasticity)　利用者は必要に応じてクラウドリソースをスケーリング

できる。場合によってはクラウドリソースの過不足に応じてクラウドリソースを自動的にスケーリングできる。

測定されたサービス (Measured service)　クラウドサービスプロバイダはクラウドリソースの計測機能を提供する。クラウドリソースのメトリクスは、クラウドサービスの利用状況のモニタリングや報告、クラウドリソースの制御などに用いることができる。

オンデマンドセルフサービスとは、クラウドサービス利用者がクラウドサービスプロバイダが提供するクラウドサービスの管理画面や Web API (Application Programming Interface) を通じて、自らクラウドリソースを操作して利用できることを指す。自らクラウドサービスの管理画面や Web API を操作して必要なクラウドリソースを割り当てることで、クラウドサービス利用者は必要なクラウドリソースを必要なタイミングですぐに利用できる。さらにクラウドリソースの管理を自動化することで、クラウドサービス利用者はシステムの構築や運用、管理に必要な管理者やスタッフ、構築や運用、管理にかかる時間を合理化することができる [iPR22]

クラウドサービス上に構築するシステムは、クラウドリソースをコンポーネントとして組み合わせることで 1 つのアプリケーションを構成する。システムの構築と運用には、ユーザーアクセスの制御、クラウドリソース割り当ての指定、システム利用の制御、システム利用状況のモニタリングなどのプロビジョニングを必要とする。プロビジョニングを自動化することで、これらのオペレーションにかける人的コストを減少させ、クラウドサービス利用者は自身のビジネスにとってより価値のある活動に人的リソースを割り当てることができる。

クラウドサービスプロバイダはプロビジョニングの自動化を支援する様々な機能をツールとして提供している。たとえばプロビジョニングの機能であれば、AWS では AWS CloudFormation [Ama22m]、Microsoft Azure では Azure Resource Manager [Cor22g]、Alibaba Cloud では Resource Orchestration Service [阿里 22j]、Google Cloud では Cloud Deployment Manager [LLC22j] が提供されている。

クラウドコンピューティングにおいて、幅広いネットワークアクセスとは、クラウドサービスを利用するために必要なネットワーク接続を提供することである。これはクラウドサービス利用者が携帯電話、ラップトップ、ワークステーション、タブレットなどのシ

ンクライアントやシッククライアントを含むさまざまなデバイスを用いて、クラウドサービスにシームレスに接続できることを意味する [Bas22]。現在では、在宅勤務の浸透による企業の管理外のネットワーク経路からの接続の増加や、従業員が自分で選んだ端末を業務に使用する BYOD (Bring Your Own Device) の増加によって、クラウドコンピューティングの幅広いネットワークアクセスの重要性はより増している。

クラウドサービスプロバイダからは、クラウドコンピューティングの幅広いネットワークアクセスを実現するために、管理画面機能や RESTful (Representational State Transfer) API を操作する CLI (Command Line Interface) ツールが提供されている。REST とはウェブ API のアーキテクチャスタイルであり、REST の原則 (統一インターフェース、ステートレス、キャッシュ可能、クライアント/サーバ、コードオンデマンド) を満たすウェブ API を RESTful API と呼ぶ [Sur16]。たとえば管理画面であれば AWS では AWS Management Console [Ama22e]、Microsoft Azure では Microsoft Azure portal [Cor22j]、Alibaba Cloud では Alibaba Cloud Management Console [阿里 22l]、Google Cloud では Google Cloud console [LLC22i] が提供されており、CLI ツールに関しては AWS では AWS Command Line Interface (AWS CLI) [Ama22j]、Microsoft Azure では Azure Command-Line Interface [Cor22d]、Alibaba Cloud では Alibaba Cloud CLI [阿里 22a]、Google Cloud では Google Cloud Command Line Interface (gcloud CLI) [LLC22e] が提供されている。

クラウドコンピューティングにおいてリソースプーリングとは、クラウドサービスプロバイダが提供するクラウドリソースを、複数の利用者が共有することを指す。クラウドサービスの仮想化されたサーバ、ストレージ、ネットワークなどのクラウドリソースは、その裏の大規模な物理的なサーバ、ストレージ、ネットワークから割り当てられている。たとえば 2012 年時点で AWS では世界中の 7 つのデータセンターで少なくとも 454,400 台のサーバを使用していたと推測されている [Mil12]。クラウドサービス利用者が、自身の利用しているクラウドリソースが実際にどの物理的なコンピューティングリソースにより提供されているか認識したり、利用しているクラウドリソースの実行環境を具体的に制御することはない。

クラウドサービスプロバイダがクラウドリソースをプールしていることで、クラウドサービス利用者のシステムで急激な負荷が発生した場合に、オンプレミスでシステム構築した場合と比べて、より多くのコンピューティングリソースをシステムへ一時的に割り当

てることができる。これは多数のクラウドサービス利用者でクラウドリソースを共有するからこそ可能な割り当てであり、ピークに合わせたコンピューティングリソースを事前に準備しなければならないオンプレミスのシステムと比較して企業にとって利点がある。

またクラウドサービス利用者から見て、クラウドコンピューティングのリソースプーリングは、リソースの冗長性の面でも利点がある [BA12]。クラウドサービスでは利用者が通常利用しているクラウドリソースに対して大きいリソースプールがあるため、クラウドサービス利用者が利用しているクラウドリソースに障害が発生した場合には、自動的に代替のクラウドリソースを割り当ててその影響を軽減することができる。

クラウドサービス利用者にとって、クラウドコンピューティングの迅速な弾力性の利点は、クラウドリソースの利用量をフレキシブルに調整できることである。ビジネスを目的とした Web アプリケーションであれば、Web アプリケーションの負荷増大に応じて素早くクラウドリソースを割り当てることは売上の最大化に貢献し、Web アプリケーションの負荷減少に応じて素早くリソースを解放することをコストの最小化に貢献する。

クラウドコンピューティングにおけるプロビジョニングには次のような手法がある [BG14]。

静的プロビジョニング (Static Provisioning)　クラウドサービス利用者とクラウドサービスプロバイダが契約し、あらかじめ固定のクラウドリソースを用意する。クラウドサービス利用者はそのクラウドリソースを定額料金で利用する。クラウドリソースの利用量が予測できる場合に効果的な手法である。

動的プロビジョニング (Dynamic Provisioning)　クラウドサービス利用者が指定した条件に応じて、クラウドサービスのスケーリング機能がクラウドサービス上のシステムの負荷に応じてクラウドリソースの割り当てを自動的に行う。クラウドサービス利用者は割り当てたクラウドリソースの利用量や利用時間のみに課金される。システム負荷およびクラウドリソースの利用量の予測が難しい場合に効果的な手法である。

ユーザーセルフプロビジョニング (User Self-Provisioning)　ユーザーセルフプロビジョニングでは、クラウドサービス利用者がクラウドサービスの管理画面や RESTful API などのクラウドサービスのインターフェースを通じてクラウドリソースの割り当てを行う。クラウドサービス利用者は割り当てたクラウドリソースの利用量や利用時

間のみ課金される。アドホックなクラウドリソースの割り当てを行いたい場合に有効な手法である、

以上のなかでクラウドサービス上のシステムの障害対応や急激な負荷変動に対する迅速な弾力性は、動的プロビジョニングにより効果的に働く。クラウドサービスプロバイダから動的プロビジョニングを支援する機能も提供されており、たとえば自動スケーリングの機能であれば、AWS では AWS Auto Scaling [Ama22b]、Microsoft Azure では Azure Autoscale [Cor22c]、Alibaba Cloud では Alibaba Cloud Auto Scaling [阿里 22d]、Google Cloud では Managed instance groups [LLC22b] の機能の一部として提供されている。

クラウドコンピューティングではクラウドサービス利用者は、必要に応じて短期的にクラウドリソースを利用し、そのリソースの利用量を支払うことができる。このようなクラウドコンピューティングのメリットを利用者が享受するためには、クラウドサービスの品質と信頼性が重要な要素になる [PRS09]。一般にクラウドサービスとクラウドサービス利用者の間ではサービス品質保証 (Service Level Agreement, SLA) が結ばれる [PRS09, VRMCL08]。クラウドサービス利用者はクラウドサービスプロバイダと SLA に基づく契約を結ぶことで、期待されるクラウドーサービスのサービスレベルを把握できる。

またクラウドサービス利用者が自らのクラウドリソース利用量を把握するために、モニタリングは重要なトピックである。クラウドコンピューティングにおいてクラウドリソースの割り当てを任意に増減することは利点ではあるが、利用者自身による操作ミスや、プロビジョニングの失敗、自身のキャッシュフローを超えたクラウドリソースの利用などの想定外の利用が発生する可能性も考えられ、想定外の利用への対応のためにもクラウドサービス利用者は現在の利用状況をモニタリングをしなければならない。加えてモニタリングでクラウドサービスのシステムが正常に動いているか確認したい、あるいはマーケティングやシステム投資戦略への活用のためにカスタマイズした指標を集めたいなども考えられる。

クラウドサービスプロバイダは、クラウドサービス利用者が自らのシステムに係るメトリクスを収集するのを支援するツールを提供している。たとえば AWS では Amazon CloudWatch [Ama22a]、Microsoft Azure では Azure Monitor [Cor22f]、Alibaba Cloud では CloudMonitor [阿里 22e]、Google Cloud では Cloud Monitoring [LLC22d] が提供

されている。クラウドサービス利用者はこれらのモニタリングツールを利用することで、サーバの CPU 使用率やメモリ使用量、ストレージ使用量、ネットワーク転送量といった基礎的なメトリクスだけでなく、クラウドサービスの利用者によるカスタマイズされた指標を収集することができる。

2.3 クラウドコンピューティングの分類

クラウドコンピューティングの分類には、デプロイメントメントモデルによる分類とサービスモデルによる分類がある。クラウドコンピューティングにおけるデプロイメントモデルとは、クラウドサービスの提供先、クラウドサービスの管理体制、クラウドサービスの利用目的に基づいた分類である。クラウドコンピューティングにおけるサービスモデルとは、クラウドサービスの提供形態に基づいた分類である。

2.3.1 デプロイメントモデルによる分類

クラウドコンピューティングのデプロイメントモデルはパブリッククラウド、コミュニティクラウド、プライベートクラウド、ハイブリッドクラウドの 4 つに分類できる [MG+11]。それぞれのデプロイメントモデルは、異なるレベルの管理、運用、柔軟性を有しており、用途に応じたモデルを選択することでクラウドサービス利用者は目的にあったクラウドサービスを利用できる。

パブリッククラウド (Public Cloud)　企業や個人などの不特定多数の消費者に向けてクラウドサービスを提供するデプロイメントモデルである。ネットワークを通じて多くのクラウドサービス利用者が利用できる。クラウドリソースはクラウドサービスプロバイダの施設内で管理、運用され、クラウドサービス利用者はクラウドサービスを利用することで、柔軟かつスケーラビリティの高いインフラストラクチャを簡単に構築できる。

コミュニティクラウド (Community Cloud)　関心を共有する複数の組織 (コミュニティ) に向けてクラウドサービスを提供するデプロイメントモデルである。対象のコミュニティがコミュニティクラウドのクラウドリソースを独占的に利用する。コミュニティクラウドのインフラストラクチャはコミュニティにより共同で管理される。パブリッククラウドと比べて開発や運用の負担がかかる一方、セキュリティ基準を業

界の事情に合わせたり、クラウドサービスプロバイダへの依存を減らすメリットがある。

プライベートクラウド (Private Cloud)　企業独自にクラウドサービスを構築するデプロイメントモデルである。企業の管轄内でデータセンターが運用されるために利用が企業内に限定され、セキュリティや運用管理が容易に行えるという特徴がある。外部からのアクセスが制限されていることから、不正アクセスなどのセキュリティリスクを最小限に抑えることができる。

ハイブリットクラウド (Hybrid Cloud)　パブリッククラウドやコミュニティクラウド、プライベートクラウドを組み合わせたデプロイメントモデル。プライベートクラウドで運用する必要のあるデータやシステムを保護しつつ、パブリッククラウドの柔軟性やスケーラビリティを活用できる。

パブリッククラウドとは、インターネットを介して、一般の利用者がアクセスできるクラウドコンピューティングのサービスモデルである。クラウドコンピューティングのデプロイメントモデルにおいて、パブリッククラウドは最も多くの種類のクラウドサービス利用者に開かれたデプロイメントモデルである。パブリッククラウドとは反対に、1 つの組織あるいは企業に属するクラウドサービス利用者のみがアクセスでき、管理するのがプライベートクラウドである。コミュニティクラウドではアクセスできるクラウドサービス利用者の範囲の意味でパブリッククラウドとプライベートの中間、ハイブリットクラウドはクラウドサービス利用者が同時に複数のデプロイメントモデルを利用する。

パブリッククラウドは多くのクラウドサービス利用者に開かれていることから、他のデプロイメントモデルと比べてクラウドサービスプロバイダは規模の経済を得ることができる。パブリッククラウドは規模の経済により得られる巨大なリソースプールにより、クラウドサービス利用者はそのクラウドリソースを安くそして柔軟に割り当てることができる。前述した AWS、Microsoft Azure、Alibaba Cloud、Google Cloud らはインターネットを介してクラウドサービスを広く一般に提供していることからパブリッククラウドの性質を有している。

パブリッククラウドと比べてプライベートクラウドの場合は、そのクラウドサービスの利用が 1 つの組織に限定されるために規模の経済は得にくい。そのためクラウドサービスプロバイダでもありクラウドサービス利用者でもある組織では、パブリッククラウドに

比べて同じクラウドリソースに対するコストが高くなる。一方でプライベートクラウドはパブリッククラウドと比べてデータのセキュリティとプライバシーの面で大きな利点を持つ [Goy14]。またプライベートクラウドは組織あるいは企業が自ら運用を行うため、仮想サーバやデータの配置場所などクラウドリソースの振る舞いをより詳細に把握できる利点がある。

2.3.2 サービスモデルによる分類

NIST によりクラウドコンピューティングはいくつかの提供形態 (サービスモデル) に分類される。NIST によるクラウドコンピューティングのサービスモデルには IaaS (Infrastructure as a Service)、PaaS (Platform as a Service)、SaaS (Software as a Service) がある [MG+11]。

IaaS (Infrastructure as a Service) IaaS は仮想マシン (Virtual Machine, VM) 技術を中心に、サーバやストレージ、ネットワークなどのハードウェアを仮想化したコンピューングリソースを提供する。

PaaS (Platform as a Service) PaaS はアプリケーションの土台となるプラットフォームを提供する。クラウドサービス利用者はハードウェアを仮想化したコンピューングリソースを制御せず、このプラットフォーム上にアプリケーションを展開することでシステムを構築できる。

SaaS (Software as a Service) SaaS はアプリケーションレベルの抽象化を行ったサービスを提供する。クラウドサービス利用者はハードウェアを仮想化したコンピューングリソースを制御せず、プラットフォーム上にアプリケーションの展開も行わない。クラウドサービス利用者は SaaS が提供するアプリケーションを設定することで自身が叶えたい機能を実現する。

IaaS はサーバを仮想化した仮想マシンを中心にサーバやストレージ、ネットワークなどの仮想化されたハードウェアを提供する。クラウドコンピューティングでは仮想化技術を用いて、サーバ、ネットワーク、ストレージといったクラウドリソースを、基盤となる物理的なコンピューングリソースから抽象化することでクラウドリソースを柔軟かつ俊敏に提供し、クラウドリソースの集約することで高い経済性を実現している。たとえば実際に市場に展開されている IaaS のサービスには、AWS の Amazon EC2 [Ama22n]、Microsoft

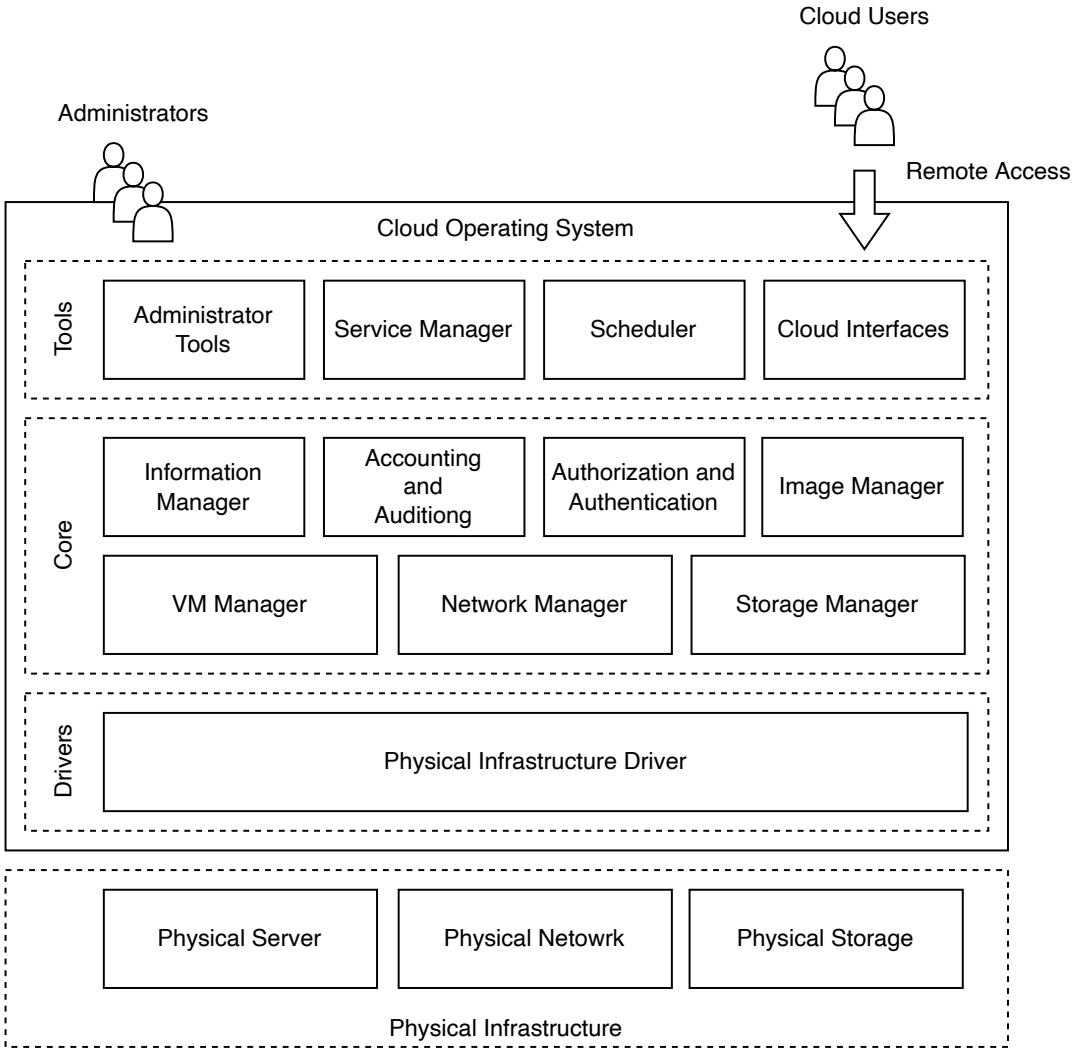

図 2.1　IaaS を構成するインフラストラクチャ

Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures [MVML12] より参考

Azure の Azure Virtual Machines [Cor22n]、Alibaba Cloud の Elastic Compute Service (ECS) [阿里 22f]、Google Cloud の Google Compute Engine [LLC22f] がある。

クラウドサービスプロバイダが IaaS の裏で仮想サーバや仮想ネットワーク、仮想ストレージなどを管理することで、クラウドサービス利用者はその実体を意識せずにあたかもサーバやネットワーク、ストレージを扱っているかのようにシステムを構築することができる。このように仮想化されたコンピューングリソースを取り扱い、管理するインフラストラクチャマネージャーをクラウドオペレーティングシステム (Cloud Operating System, Cloud OS) と呼ぶ [MVML12]。

図 2.1 は IaaS を構成するインフラストラクチャの概念図である。Cloud OS は物理インフラストラクチャと仮想インフラストラクチャを管理することで、クラウドサービス利用者のクラウドリソースのニーズに応じて仮想的なコンピューティングリソースの提供を制御する。Cloud OS は物理的なサーバ、ネットワーク、ストレージの CPU やメモリ、ネットワーク I/O、ディスク領域などの計算能力を集約し、仮想的なクラウドリソースに基づいて物理的なコンピューティングリソースを論理的にパーティショニングをすることで、クラウドサービス利用者に安全で分離された仮想マシンの実行環境 (仮想マシンインスタンス) を提供する。そして Cloud OS は仮想マシンインスタンスを取り扱うための管理画面や RESTful API などのインターフェースをクラウドサービス利用者に提供する。これによりクラウドサービス利用者はネットワークを介してクラウドリソースを自由に扱うことができる。

Cloud OS は仮想マシンを基本的なクラウドサービスの単位とする。VM Manager は物理サーバの CPU やメモリなどの計算能力を分割し、論理的に区切られた仮想サーバインスタンスを提供する。複数のコンポーネントから構成されるシステムを構築するためには、仮想マシンインスタンスの間で相互接続するためのネットワークも必要である。Network Manager はネットワークドライバを使用して物理ネットワーク上に論理的に区切られたプライベートな仮想ネットワークを設定する。ネットワークの分離には VLAN (IEEE802.1Q) [TFF+13] などが用いられる。Storage Manager は物理ストレージを集約し、論理的に分割することでクラウドサービス利用者に仮想ストレージを提供する。また障害発生時のデータアクセスの中断を回避するための高可用性と高信頼性を提供する役割も担う。これらの機能を実現する仮想化ソフトウェアに Xen [Fou22]、KVM (Kernel based Virtual Machine) [com22]、VMware [VMw22] などがある。

クラウドサービスを提供するためには、クラウドリソースの管理だけでなく、クラウドサービスを運営していく上で不可欠な各種機能を有していなければならない。Information Manager では仮想および物理インフラストラクチャの状態を監視し、これらのインフラストラクチャが正常に動作しているか情報を収集している。Information Manager が収集した情報から Accounting がコンピューティングリースの利用情報を記録する。Accounting は課金情報を生成するメカニズムを実装するためにも不可欠である。Auditing では、誰が、いつ、どのような操作でクラウドサービス内のコンピューティングリソースにアクセスしたかといったアクティビティに関する情報を記録する。こ

れらの情報は不正アクセスやリソースの不正利用などの脅威からクラウドサービスを守り、クラウドサービスのセキュリティを向上させるために不可欠である。Authentication and Authorization はクラウドサービス利用者および管理者を認証し、許可されたコンピューティングリソースへのアクセスのみを提供するメカニズムを提供する。認証認可の技術にはの LDAP (Lightweight Directory Access Protocol) [Zei06, Ser06] を介した認証メカニズムや、SAML (Security Assertion Markup Language) [HM05] や OpenID Connect [SBJ+14] などのインターネットベースの認証メカニズムがある。Image Manager では仮想マシンイメージを効率的かつ安全に管理するためのメカニズムを提供している。現在の仮想マシンを取り扱うクラウドサービスにおいて、仮想マシンイメージの管理は重要な課題であり、異なるクラウドサービス利用者の膨大な数の仮想マシンイメージを処理しなければならないからである。

Administrative tools はクラウドサービスの管理者が Cloud OS を管理するためのツール（ユーザー作成、変更、削除、ユーザー権限とアクセス制御ポリシーの管理）や物理インフラストラクチャの管理ツールを提供する。Service Manager は後述する Scheduller と連携し、クラウドサービス利用状況と物理インフラストラクチャの使用状況に応じて、仮想マシンインスタンスや仮想ネットワーク、仮想ストレージの配置を制御する。Sucheduler では仮想マシンインスタンスの初期配置を決定するとともに、最適化基準に従って物理サーバから別の物理サーバへ仮想マシンインスタンスを動的に再配置する。Sucheduler の配置条件には、ハードウェア (CPU やメモリ量) やプラットフォーム (ハイパーバイザや OS の種類)、アフィニティ (同じ物理サーバや同じ物理サーバのクラスタへの配置)、ロケーション (地理的制約)、サービスレベル契約の制約 (CPU 容量保証や高信頼性など) などの制約を加えることもできる。Cloud Interface ではクラウドサービス利用者に管理画面機能や RESTful API を提供する。

PaaS はクラウドコンピューティングのモデルの一つで、プラットフォームをサービスとして提供することを指す。PaaS ではクラウドサービスプロバイダはアプリケーション実行のためのプラットフォームを提供し、クラウドサービス利用者はこのプラットフォームを利用することでアプリケーションを実行する。IaaS と比較して PaaS ではランタイム環境までクラウドサービスプロバイダによりサポートされているため、クラウドサービス利用者は OS のインストールやアップグレードを自らの手で行う必要がない [SSKY16]。そのためクラウドサービス利用者は仮想マシンインスタンスをメンテナンスをすることな

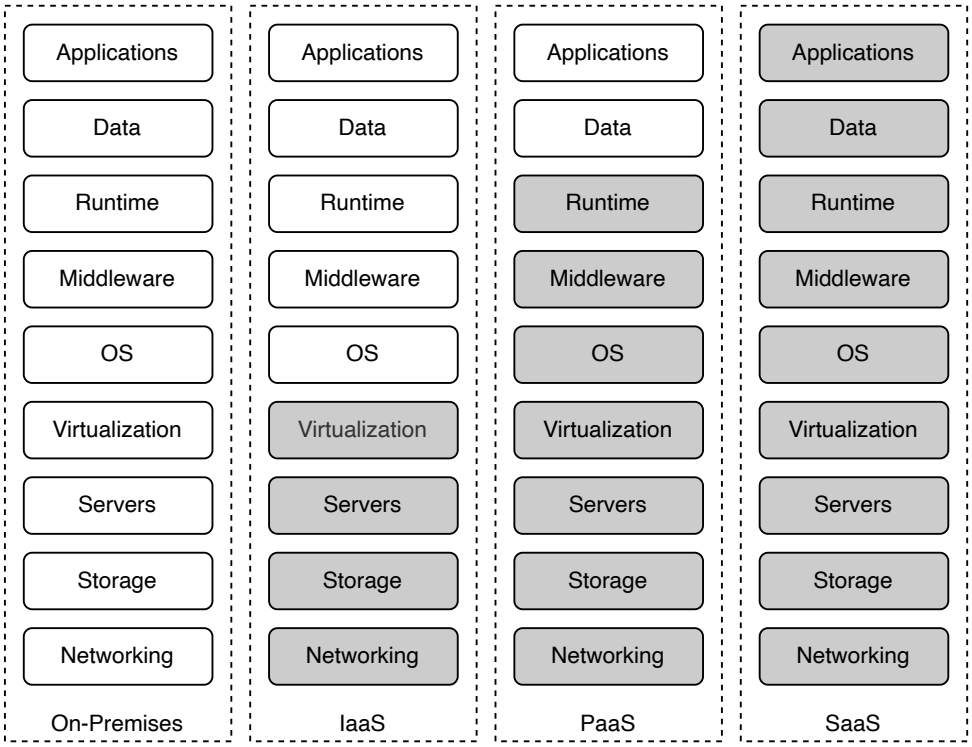

図 2.2　サービスモデルによる管理対象の違い

Cloud computing defined: Characteristics & service levels - Cloud computing news [Sch14] より参考

く、PaaS 上にアプリケーションを展開することができ、開発および展開サイクルを高速化できる [Son14]。PaaS のサービス例に AWS の AWS Elastic Beanstalk [Ama22p] や Microsoft Azure の Azure App Service [Cor22b]、Alibaba Cloud の Simple Application Server [阿里 22k]、Google Cloud の Google App Engine [LLC22a] がある。

SaaS はアプリケーションレベルの抽象化でサービスを提供する。クラウドサービス利用者は物理的なリソースや仮想マシン、プラットフォームなどを制御せず、アプリケーションの展開も行わない。PaaS との違いは、PaaS は提供されるプラットフォームを利用してアプリケーションを構築していくのに対して、SaaS ではアプリケーションそのもの、つまりソフトウェアがクラウドサービスとして提供されている点である。クラウドサービス利用者は利用している SaaS の設定をカスタマイズすることで自身が実現したい機能を実現する。SaaS のサービス例に Microsoft Corporation 提供の Microsoft 365 [Cor22i] や Google LLC 提供の Google Workspace [LLC22l]、Salesforce, Inc. 提供の Salesforce [Sal22] がある。

クラウドコンピューティングにおける IaaS、PaaS、SaaS のサービスモデルは、クラウドサービス利用者に提供するクラウドリソースの抽象度が異なる。IaaS より PaaS の方がクラウドリソースの抽象度が高く、PaaS より SaaS の方がクラウドリソースの抽象度が高い。クラウドリソースの抽象度が高くなればなるほど、よりクラウドサービス利用者に直接的な利点をもたらす機能を提供し、アプリケーションを実現するために必要な管理の多くをクラウドサービスプロバイダが負担する。図 2.2 は IaaS、PaaS、SaaS のサービスモデルにおいて、どれだけの層の管理項目をクラウドサービスプロバイダが担当しているかを表している。IaaS では仮想マシンまでクラウドサービスプロバイダが管理しており、クラウドサービス利用者がアプリケーションを構築するためには OS、Middleware、Runtime、Data、Applications までを設計して構築し、運用していかなければならない。SaaS では Applications の提供まで、すべての層がクラウドサービスプロバイダによって管理されており、SaaS の利用者は IaaS と比べて保守や運用、システムのサポートの負担無しにアプリケーションを利用することができる [KGP12]。一方で SaaS はインフラストラクチャのほとんどをクラウドサービスプロバイダが管理しているため、カスタマイズ性が IaaS や PaaS に比べて制限されるという特徴を持つ。そのためクラウドサービス利用者、特に企業においては、社内のセキュリティルールや独自に満たしたい要求に SaaS の仕様がマッチしないなどの状況がある。

クラウドコンピューティングのサービスモデルには IaaS, PaaS, SaaS の他にも様々なサービスモデルが提案されている。たとえば現在では、クラウドサービスプロバイダは XaaS (Everything/Anything as a Service) という形で様々なサービスモデルを提案している [DFZ+15]。データ分析ソフトウェアを提供する AaaS (Analytics as a Service) や仮想デスクトップ環境を提供する DaaS (Desktop as a Service)、コンテナ化されたアプリケーションやクラスタの管理機能を提供する CaaS (Container as a Service) などもあり、IaaS, PaaS, SaaS の分類は大局的な見方である。実際のビジネスの現場では、自社が満たしたい要件や制限に合わせてサービスモデルから得られるメリットとサービスモデルにより課される制約のバランスをとり、クラウドサービスの選択を行っている。

2.4 クラウドコンピューティングにおけるスケーラビリティ

クラウドサービス上にシステムを構築するとき、システムのスケーラビリティは外すことのできない考慮事項である。スケーラビリティとはリソースの需要 (あるいは要求) に対して追加のリソースを提供するプラットフォームの能力である [LEB15]。スケーラビリティを持つシステムは、たとえばリクエストやユーザー数が増加しても、正常に動作し続けてユーザーの満足を損なうことがないため、ビジネス上でも重要な特性である。スケーラビリティを持つシステムが、システムの負荷に応じてリソースの割り当てを調整することをスケーリング (Scaling) と呼ぶ。

2.4.1 プロアクティブスケーリングとリアクティブスケーリング

スケーリングのアプローチには、プロアクティブスケーリング (Proactive Scaling) とリアクティブスケーリング (Reactive Scaling) がある [FB14]。プロアクティブスケーリングとは、システムへの負荷の増加あるいは減少を予測し、あらかじめ予測したシステムの負荷をカバーできるだけのリソースを予測した期間に割り当てるアプローチである。クラウドサービスプロバイダのスケーリングサービス [Ama22b, Cor22c, 阿里 22d, LLC22b] では、スケジュールされたスケーリングとして機能が提供されており、クラウドサービス利用者は予測したシステムへの負荷の増減に合わせてスケーリングをスケジューリングすることで、システムの負荷に応じたリソースを自動的に割り当てることができる。

プロアクティブスケーリングを行っていても、ときにはシステムへの負荷の増減の予測と、実際のシステムへの負荷が異なることがある。あるいはシステムへの負荷の増減の予測は当たっていたが、一時的な流行りなどの予期しない外部要因によって、想定外のシステム負荷の変化が起こることもある。リアクティブスケーリングはこのような想定外のシステムへの負荷の変化 (システムの負荷の変動) に対応することができる。リアクティブスケーリングとは、システムの負荷や使用状況に応じて、リアルタイムで割り当てるリソースを増減させるアプローチである。リアクティブスケーリングでは、システムの負荷や使用状況を測定するためのモニタリングシステムと動的プロビジョニングを組み合わせることで、システムの負荷の変動に対してリアルタイムにリソースの追加と解放を実現する。

リアクティブスケーリングではモニタリングで収集したメトリクスから得るスケーリング指標を用いて、リソースのスケーリングを行う。スケーリング指標はクラウドサービス利用者のシステムの QoS (Quality of Service) を満たす直接的あるいは間接的な値が用いられる。QoS とはサービス品質、特にここではネットワークを介するサービスを安定して使えるよう保証する技術を指す。QoS を計る指標としては、たとえば直接的な指標であればレスポンスタイムや伝送速度、エラー率、間接的な指標であれば CPU 使用率やメモリ使用量、ディスク I/O などがある。クラウドサービスプロバイダのスケーリングサービス [Ama22b, Cor22c, 阿里 22d, LLC22b] では、自動スケーリングとしてリアクティブスケーリング機能が提供されており、クラウドサービス利用者はスケーリング指標に閾値やスケーリングのルールを設定することで、システムの負荷に応じてリソースのスケーリングを起動し、システムの負荷に対して過不足無いリソースの割り当てを実現できる。

2.4.2 垂直スケーリングと水平スケーリング

スケーリングの手法には大きく分けて垂直スケーリング (Vertical Scaling) と 水平スケーリング (Horizontal Scaling) がある [PS13]。垂直スケーリングとはシステムで使用するリソースのサイズ（CPU、メモリ量、ネットワーク帯域幅など）をシステムの負荷に応じて大きくする手法である。たとえば大きなメモリ容量を持つ大型サーバで、メモリに依存するステートフル (メモリ上に現在の状態を表すデータなどを保持し、その内容を処理に反映させる方式) アプリケーションの場合、使用するメモリ容量を随時拡大するアプローチなどが該当する。水平スケーリングとは、システムの負荷に応じて同じソフトウェアまたはハードウェアのリソースを追加する手法である。たとえばウェブアプリケーションなどで、ユーザー数やシステム負荷の増加（減少）に合わせてフロントエンドサーバを追加（または解放）することが該当する。水平スケーリングは垂直スケーリングと比べて、管理の複雑さは増すものの、特にウェブアプリケーションなどでコストと性能のバランスがとれる手法である [CVRM+10]。

クラウドコンピューティングにおいてスケーラビリティの手法は、以下の 3 つのクラウドコンピューティングの技術で成り立つ [CVRM+10]。

仮想化技術　仮想化技術によりスケーリングの際に仮想マシンインスタンスなどのリソースを追加する。仮想化技術により物理的なリソースの制約を受けずに割り当てるこ

とができる。またアフィニティ条件で異なる仮想マシンインスタンスを同じ物理サーバに配置するなど、計算能力の最適化に柔軟性を持たせることもできる。

リソースの共有　リソースを共有することで、リソースのプールに空きがあればスケーリング時にリソース割り当てる。垂直スケーリングでは物理マシンの最大容量まで、水平スケーリングではデータセンターの最大容量まで、リソースを割り当てられる可能性がある。

動的プロビジョニング　スケーリング時には動的プロビジョニングを用いてリソースを割り当てる。モニタリング機能と動的プロビジョニングを組み合わせることで自動的にスケーリングすることが可能である。

スケーラビリティを実現するクラウドコンピューティングの技術により、クラウドサービス利用者は垂直スケーリングと水平スケーリングを実現できる。垂直スケーリングにおいては、システム負荷に応じて元の仮想マシンインスタンスより高性能なインスタンスを割り当てることで、垂直方向にリソースを充足できる。水平スケーリングにおいては、システム負荷に応じて、既存の仮想マシンインスタンスのグループにインスタンスの複製を追加することで、水平方向にリソースを充足できる。アプリケーションが適切に設計され、そのアーキテクチャが適切なスケーラビリティを有していれば、システム負荷に応じて自動的に仮想マシンインスタンスがプロビジョニングされ、最終的にはシステムで必要なリソースと割り当てられたリソースでバランスがとれる。

2.4.3　スケーリングの実践例

クラウドサービス利用者がシステムのアーキテクチャを設計する際には、スケーラビリティを念頭に置いておく必要がある。アプリケーションのシステム負荷は刻一刻と変化するためクラウドサービス利用者がこれを完全に予想することは難しく、あらかじめクラウドサービス利用者が必要なリソースを期間ごとにプロビジョニングすることは現実的ではない。柔軟なプロビジョニングを実現するポイントは、監視システムと自動化された動的プロビジョニングを用いることである。各社クラウドサービスでは動的プロビジョニングをするためのツールと、リソースの状況を監視するためのツールが提供されている。これらのツールを組み合わせて、CPU 使用率やレスポンスタイムなど QoS に関連するインフラストラクチャの指標を監視し、動的にリソースを割り当てることで、システムに必要な

リソースのスケーリングを実現できる。クラウドサービスを用いるアプリケーションは、抽象度の高い指標に基づいて運用され、そのシステムの管理者に具体的な個々のスケーリングの対応を意識させないのが理想的である。

ビジネス状況下においてもクラウドサービス利用者がシステムのアーキテクチャを設計する際には、スケーラビリティを念頭に置いておく必要がある。たとえばサービス立ち上げ初期の段階では少数のユーザーを対象とし、短い市場投入期間や迅速なプロトタイプ開発が優先されていたとしても、市場の受け入れによってはシステムのユーザー数が数百人から数千人、さらには数百万人に増加する可能性があるからである。このようなケースにおいて、スケーラビリティを持たないシステムではシステム負荷に応じたリソースを即座に割り当てることは難しく、 QoS の低下及び機会損失、さらにはその後のビジネスの成否にまで影響を与える可能性がある。また稼働しているシステムにスケーラビリティを付与するには多くの時間と人的コストが必要であり、既存のシステムの設計によっては、スケーラビリティを付与するためにシステムを停止させなければならない可能性も考慮しなければならない。

さらにクラウドコンピューティングではクラウドサービスのリソースの利用量とコストが従量制により密接に結びついているため、動的プロビジョニングを行う際のアンダーバイイング (Under Buying) とオーバーバイイング (Over Buying) いずれかが発生する点にも考慮が必要がある。アンダーバイイングとは要求されたリソースに対して少ないリソースの割り当てを行うことであり、オーバーバイイングは要求されたリソースに対して多いリソースの割り当てを行うことである。ビジネス上ではアンダーバイイングは QoS の低下及び機会損失であり、オーバーバイイングはコストの増加による利益減少を意味する。対策としては小型のサーバを用いてリアクティブに水平スケーリングすることで、スケーリング時のアンダーバイイングとオーバーバイイングを抑える方法がある。

2.5 クラウドサービスのコスト因子

ビジネスにおいてクラウドコンピューティングのコストの把握は重要な関心事である。コストの把握で知られているアプローチに、総所有コスト（Total Cost of Ownership, TCO）のアプローチがある [LLL+09]。TCO とはリソースのコストや、メンテナンスなどの人的コスト、その他すべてのコストの総額を意味する。クラウドサービスプロバイダ

におけるリソースの費用は、サーバ、ストレージ、ネットワーク、電力、冷却、機器や、そのハードウェア上で動かすソフトウェアの導入費用が含まれる。またクラウドサービスプロバイダにおけるメンテナンスのコストには、導入時および導入後の運用後の維持費や管理費、人件費などが含まれる。クラウドサービスプロバイダはこれら TCO の考え方を用いることで、自身が提供するクラウドサービス全体のコストを把握している。一方でクラウドサービス利用者から見たとき、TCO はリソースのコストと利用するクラウドサービスを用いたシステムの運用管理の人的コストを合わせたコストである。

第 2.3.2 節で説明したように、クラウドサービスは仮想化技術によって物理インフラストラクチャとは異なる仮想化されたリソースを提供している。仮想化技術によって、クラウドサービスプロバイダによって仮想化されたリソースのコスト構造は、クラウドサービスプロバイダ自らが任意に設定することができる。クラウドサービスプロバイダによる柔軟なコスト構造は、クラウドサービスプロバイダの TCO を構成するコストの転嫁により成り立っている。クラウドサービスにおけるコストの転嫁について、Simonet 他 [SLO16] が大規模な分散型クラウド基盤データセンターにおけるコスト分析のなかで説明している。彼らはクラウドサービスをコントロールレベルに応じて分類し、完全なアウトソーシングであるレベル 1 から、完全なコントロールであるレベル 5 まで分類することで、コントロールレベルの違いによってコストの転嫁が行われ、クラウドサービスプロバイダとクラウドサービス利用者の間でのコスト構造の違いが生まれることを説明した。アウトソーシングの割合が高い (クラウドサービスプロバイダがクラウドサービス利用者の代わりに管理する項目が多い) クラウドサービスであるほど、物理インフラストラクチャやそのメンテナンスのコストがクラウドサービスに転嫁されており、クラウドサービスのコスト因子の数が減少している。これはクラウドサービス利用者はクラウドサービスを利用することで、物理インフラストラクチャと比べて少ないコスト因子数でリソースのコストを把握できることを意味する。

クラウドサービス利用者はクラウドサービスプロバイダによって定義されたリソースのコスト構造に基づき、利用したリソースのコストを支払う。多くのクラウドサービスプロバイダは、自らが提供するクラウドサービスのリソースに時間単位または使用量依存のコスト構造を設定している [MWT12]。たとえば AWS の Amazon EC2 [Ama22n] の料金は、利用する仮想マシンの性能をどれだけの時間利用したか、ネットワークであればどれだけのデータを転送したか、ストレージであればどれだけのサイズのストレージをどれだ

けの時間利用したかで請求額が定まる [Ama22n]。また PaaS や SaaS のように、より抽象化されたサービスであれば、コストの転嫁はさらに進み、より単純化されたコスト構造が設定されていることが多い。たとえば Salesforce であれば、ユーザー数ごとの月額料金により請求額が定まる [Sal22]。

まとめるとクラウドサービス利用者から見たとき、クラウドサービスのリソースのコスト構造は以下の要因に依存する。

利用するクラウドサービスプロバイダ　クラウドサービスはそれぞれ個別のクラウドサービスプロバイダにより管理、運営され、クラウドサービス利用者へサービスが提供される。仮想化技術により、各クラウドサービスプロバイダが提供するクラウドサービスの料金および料金体系はクラウドサービスプロバイダによって異なる。各クラウドサービスプロバイダは提供するクラウドサービスの価格設定を公開しており、クラウドサービス利用者がいつでも参照できるようにしている。たとえば AWS、Microsoft Azure、Alibaba Cloud、Google Cloud においても、それぞれのクラウドサービスの価格表 [Ama22h, Cor22m, 阿里 22c, LLC22m] が公開されている。クラウドサービス利用者は、公開されているクラウドサービスの価格表を参考にして、どのクラウドサービスプロバイダを採択するか、どのリソースを利用したらどれだけのコストがかかるか、どのリソースが自分たちにとってコストベネフィットがあるかなどの確認プロセスや意思決定を行う。

利用するクラウドサービスの提供形態　クラウドコンピューティングのサービスモデルには、IaaS や PaaS、SaaS、他にも様々な XaaS があり、クラウドサービスプロバイダにより数多くのクラウドサービスが提供されている。クラウドサービスのサービスモデルの抽象度の違いにより、物理的なリソースのコストやメンテナンスの人的コストが、クラウドサービスプロバイダの定めたクラウドサービスのコスト構造に転嫁される。一般にサービスモデルの抽象度が高ければ高いほど、より単純化されたコスト構造が設定されていることが多い。またクラウドサービスのデプロイメントモデルには、パブリッククラウド、コミュニティクラウド、プライベートクラウド、ハイブリットクラウドがある。より多くの利用者へ開かれているデプロイメントモデルほど規模の経済性と市場の競争性が働き、クラウドサービス利用者から見てリソースのコストが抑えられることを期待できる。

利用するクラウドサービスの種類　クラウドコンピューティングでは、仮想化されたサーバやネットワーク、ストレージンなど、様々なリソースをクラウドサービスとして利用できる。より抽象化されたアプリケーションといった形でリソースがクラウドサービスとして提供されることもある。同じクラウドサービスプロバイダ、同じクラウドサービスの提供であっても、リソースの種類が異なれば、異なるコスト構造が適用される。たとえば仮想マシンであれば、仮想マシンの種類ごとに設定された CPU、メモリ量、ストレージなどの性能が異なり、高性能であればあるほど時間単位の利用料に高い価格付けが行われている。仮想マシンの CPU のアーキテクチャーの違いや、ストレージの種類 (SSD や HDD) など、仮想マシンの仕様に基づいて異なる価格付けが行われる。アプリケーションであればユーザー数あたり価格付けや、リクエスト回数あたり価格付けなど、アプリケーションの種類によって異なる価格付けが行われる。

利用するクラウドサービスの量　利用するクラウドサービスのリソースの単価にリソースの利用量をかけ合わせることで単位時間あたりのコストが計算される。リソースの利用量の単位は、リソースのサービス設計を元に、クラウドサービスプロバイダにより仮想マシンの種類であったり、リクエスト数であったり、サービスを利用するユーザー数であったり様々である。リソースの単位時間においても、リソースのサービス設計を元に、クラウドサービスプロバイダにより 1 秒、1 時間、1 ヵ月など様々である。

利用するクラウドサービスが提供されている地域　クラウドサービスプロバイダによっては世界中にデータセンターを提供しており、クラウドサービス利用者が利用するクラウドサービスは提供地域ごとにコストが異なる。多くはコスト構造を維持したまま、日本の東京都や大阪府、アメリカのオレゴン州、アメリカの北バージニアなどのリージョンという区切りで、異なるリソースの単価が設定されている。これはリージョンごとにデータセンターを構成する物理的なリソースの調達コストや、メンテナンスを行う人的コストが異なることに起因する。クラウドサービスプロバイダ各社がクラウドサービスを展開しているリージョンは公開されており、たとえば AWS、Microsoft Azure、Alibaba Cloud、Google Cloud ではウェブサイト上にクラウドサービスを提供しているリージョンの最新情報を掲載している [Ama22l, Cor22h, 阿里 22i, LLC22h]。

クラウドサービスを利用する期間　クラウドサービスを利用すると、一定期間内に利用したコンピューティングリースの利用量に応じて、クラウドサービスプロバイダからクラウドサービス利用者に請求が発生する。多くのクラウドサービスプロバイダでコンピューティングリースの利用量を計上する対象の期間には 1 ヵ月が設定されている。各クラウドサービスプロバイダはコンピューティングリースの利用量の計上期間の締日後に、モニタリングにより集めたコンピューティングリースの利用状況のデータから自身が設定したクラウドサービスの価格体系のコスト因子となる指標を抽出し、あらかじめ定義したコスト計算の手順に従って当 1 ヵ月間におけるクラウドサービスの各利用者への請求額を計算する。

2.6　サーバレスコンピューティングの浸透

近年ではサーバレスコンピューティングへの注目が高まってきている [BCC+17]。たとえば近年の導入事例では Netflix や Coca-Cola などの利用例 [Das18] が記憶に新しい。サーバレスコンピューティングとは、クラウドサービスプロバイダがクラウドーサービス利用者に代わってサーバ (仮想サーバも含む) を管理するコンピューティングモデルである [FIMS17]。サーバレスコンピューティングの概念は、サーバを管理せずにクラウドリソースを利用できるという意味で PaaS や SaaS と重なる部分がある。サーバレスコンピューティングでは、利用者側から見てサーバの存在が隠蔽されている。サーバの管理に伴うキャパシティプランニング、構成管理、メンテナンス、スケーリングなどの運用はクラウドサービスプロバイダにより行われ、クラウドサービス利用者はサーバの存在を意識せずにクラウドリソースを利用できる。

サーバレスコンピューティングのサービス (サーバレスサービス) には様々なサービス例がある。たとえばコンピューティングのサービスであれば FaaS (Function as a Service)、データベースのサービスであれば DaaS (Database as a Service)、モバイルアプリケーション向けのバックエンドサービスであれば MBaaS (Mobile Backend as a Service) がある。FaaS はインフラストラクチャの管理や仮想サーバの存在を意識することなく、アプリケーションの機能を開発・実行できる環境を提供するものである。FaaS ではアプリケーションをトリガーとアクションに分解したソフトウェアアーキテクチャとして定義し、シームレスなホスティングとサービスのプラットフォームを提供す

る [Raj20]。FaaS のサービス例としては、AWS の AWS Lambda [Ama22o] や Microsoft Azure の Azure Functions [Cor22e]、Alibaba Cloud の Function Compute [阿里 22g]、Google Cloud の Cloud Functions [LLC22c] が挙げられる。MBaaS は、モバイルアプリや Web アプリの開発者に向けて、ユーザー管理、プッシュ通知、ストレージ機能などのバックエンドサービスを Web API で提供する。MBaaS のサービス例としては、AWS の AWS Amplify [Ama22k] や Microsoft Azure の Azure Mobile Apps [Cor22k]、Alibaba Cloud の Mobile Platform as a Service (mPaaS) [阿里 22h]、Google Cloud の Firebase [LLC22g] が挙げられる。

サーバレスコンピューティングの最大の特徴は、クラウドーサービス利用者から見て仮想サーバインスタンスの存在が隠蔽されていることである。サーバレスコンピューティングではインフラストラクチャの基礎となるロジックを管理する必要がないため、クラウドサービス利用者は本質的なサービスの価値となるアプリケーションのロジックのみに集中できるメリットがある。このような利点は、サーバレスコンピューティングのサービス (サーバレスサービス) の以下特徴により支えられる [FIMS17, ESVE+20]。

- 基盤となるインフラストラクチャの管理削減
- きめ細かいスケーリング
- 秒単位で実際の使用量に応じた課金

クラウドサービスプロバイダによりランタイム以下の階層のコンピューティングリースが管理されることにより、サーバレスサービスではアプリケーションのロジックとランタイムの間に境界が設けられている。アプリケーションのロジックとランタイムの間に管理の境界があることにより、クラウドサービス利用者はアプリケーションのロジックを実行するためのインフラストラクチャの管理を削減できる。サーバレスコンピューティングでは機能 (Function) の実行環境とバックエンドサービスを提供する [WCJL22]。

クラウドサービスの基礎となるクラウドリソースは仮想マシンである。仮想マシンは物理的なハードウェアを仮想化し、仮想マシンインスタンスの間に境界を設けて、コンピューティングリースの分離を実現する。しかし仮想マシンはハードウェアを仮想化しているため、仮想マシンインスタンスの稼働開始時には OS 起動のオーバーヘッドがかかり、仮想マシンインスタンス上のアプリケーションの稼働に数十秒から数分の時間を要する。サーバレスサービスでは仮想マシンよりも早く立ち上げることができるコンテナ技術

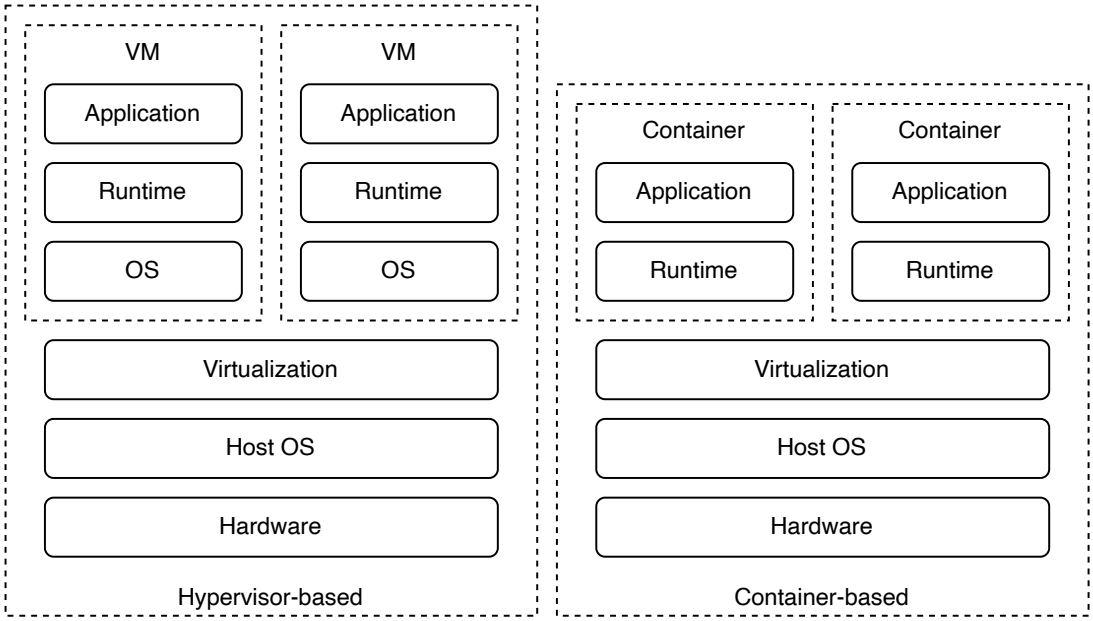

図 2.3　仮想マシンとコンテナの違い

を用いることが多い [WCJL22]。コンテナ (Container) とは仮想化技術の 1 つであり、ホスト OS 上にアプリケーションの実行環境を論理的に区切り、その区切られた環境内でアプリケーションを実行させる [KS17, PL15]。コンテナ技術を用いたソフトウェアには Docker [Doc22] や Linux Container (LXC) [lin22] などが挙げられる。

図 2.3 は仮想マシンとコンテナの実行環境の違いを表したものである。仮想マシンインスタンスの実行環境ではホスト OS の他に、各ゲストインスタンスで OS がインストールされている。一方でコンテナの実行環境では、ホスト OS 上でアプリケーションの実行環境を区切っているため、各ゲスト環境 (コンテナ) に OS がインストールされていない。コンテナの実行環境ではホスト OS のカーネルを共有することでアプリケーションの実行環境を揃えている。コンテナの実行環境では OS のカーネルを共有しているため、仮想マシンインスタンスの実行環境よりも早く立ち上げることができる (数十ミリ秒から数百ミリ秒)。またコンテナのイメージには一般にアプリケーションを実行するためのバイナリとライブラリが含まれるため、コンテナを実行する立場から見ると、ハードウェアやソフトウェアから独立して実行されているように見える。

コンテナ技術により仮想化されたコンピューティングリソースにより、サーバレスサービスでは仮想マシンベースと比べてきめ細かいスケーリングをすることができる [VETT+18]。きめ細かいスケーリングにより、従来の仮想マシンベースと比べてより

多くの最適化が可能になる。従来の IaaS など仮想マシンベースで構築されたシステムでは、アプリケーションは仮想マシンインスタンスで構成されており、スケーリング時にはプロビジョニングに数分かかる。サーバレスサービスにおいてはコンテナ技術によるフットプリントの少なさによって、クラウドサービス利用者はアプリケーションのロジックを実行するためのインフラストラクチャの管理を削減したまま数秒から数十秒でスケーリングを行える。

サーバレスコンピューティングの特徴であるきめ細かいスケーリングによって、クラウドサービス利用者は秒単位でシステム負荷に応じてクラウドリソースを割り当てることができ、システム負荷に応じた支払いのみをするメリットをより享受できる。これによりクラウドリソースのオーババイイングとアンダーバイイングを最小限に抑え、クラウドサービスのコストを削減したいクラウドサービス利用者と、データセンターのコンピューティングリソースを効率的に管理して TOC を抑えたいクラウドサービスプロバイダのニーズを両方満たすことができる。

2.7 クラウドサービスのコスト把握

2.7.1 利用しているクラウドサービスのコスト確認

クラウドサービス利用者にとって、現時点でどれだけのクラウドリソースを利用しているのか、どれだけのコストが構築したシステムでかかっているのかは重要な関心事である。クラウドサービスの可観測性により、クラウドサービスプロバイダはクラウドサービス利用者に対して、クラウドサービスのコスト管理ツールを提供している。クラウドサービス利用者は、このコスト管理ツールを用いることで、現状のシステムでどれだけのコストがかかっているかを把握することができる。クラウドサービス利用者がクラウドサービスのコストを把握する方法には、クラウドサービスプロバイダからの請求書を確認する方法と、 クラウドサービスの管理画面から現在のクラウドリソース利用量を確認する方法がある。

クラウドサービス利用者が、自身のクラウドサービス利用量を把握する最も単純な方法は、クラウドサービスプロバイダからの請求書を確認することである。クラウドサービスプロバイダは、定められた期間のクラウドリソース利用量を計測して収集し、集計することでクラウドサービス利用者への課金額を計算する。一般にクラウドサービスプロバイダ

図 2.4　クラウドサービスのコスト確認ツール

AWS Cost Explorer FAQs - Amazon Web Services [Ama22d] より引用

からの請求書には、対象となる期間、リソースの種類、リソースの利用量、そして請求金額が記述されている。

請求書を確認するのでは使いすぎの防止などを防げず、一般的にはクラウドサービスプロバイダが提供するコスト管理ツールを用いることも多い。図 2.4 は、AWS が提供するコスト管理ツール [Ama22c] の確認画面の例である。

図 2.4 では、月あたりのすべてのクラウドリソースのコストと、利用したクラウドリソースごとのコストが棒グラフで表示されている。またクラウドサービスプロバイダによっては、1 ヵ月間の累積だけなく使用状況を 1 日ごとにチェックしたり、グループ化を行うことでシステムのなかの特定のサービスに属するリソースのコストを把握することができる。

表 2.1　クラウドサービスプロバイダのコスト計算ツール

クラウドサービスプロバイダ	コスト計算ツール
Amazon Web Service	AWS Pricing Calculator [Ama22f]
Microsoft Azure	Azure Pricing Calculator [Cor22l]
Alibaba Cloud	Alibaba Cloud Price Calculator [Clo22]
Google Cloud Platform	Google Cloud Pricing Calculator [LLC22k]
IBM Cloud	IBM Cloud Cost Estimator [Cor22a]
Oracle Cloud	Oracle Cloud Cost Estimator [Cor22o]

2.7.2　利用しているクラウドサービスのコスト計算

アプリケーションを提供するビジネスでは、システムのインフラストラクチャに用いるクラウドリソースを決定する前に、そのクラウドリソースを追加することでどれだけのコストがかかるかを見積もりすることが一般的である。しかしクラウドサービスプロバイダは、クラウドリソースごとに様々な価格設定をしており、クラウドサービス利用者がクラウドサービスプロバイダが提供するクラウドリソースの価格設定を、すべて把握しながらリソースのコストを見積もることはその数が多すぎて困難である。クラウドサービスプロバイダは、事前にシステムに用いるクラウドリソースのコストを見積もるクラウドサービス利用者に向けて、クラウドサービスのコスト計算を支援するツールを提供している。

表 2.1 はクラウドサービスプロバイダ各社が提供しているコスト計算ツールの一例である。クラウドサービス利用者がコストを計算しやすくするために、クラウドサービスプロバイダはこのようなコスト見積りツールを無料で提供している。クラウドサービス利用者は、クラウドサービスプロバイダが提供するこれらのコスト見積もりツールを使うことで、クラウドサービス上のシステムのインフラストラクチャを維持するためのコストを見積ることができる。しかしこれらのツールはクラウドサービスプロバイダが提供するクラウドサービスのコストを正確に見積もることができるが、対象のクラウドサービスのコスト因子を事前に用意した上で入力する必要がある点に注意が必要である。

図 2.5 は AWS が提供しているコスト見積もりツール (AWS Pricing Calculator [Ama22g]) のスナップショットである。図 2.5 では AWS の仮想マシンのサービスである Amazon EC2 [Ama22n] のコスト見積もりを行っている。AWS Pricing Calculator のウェブサイトにアクセスし、コストを見積もる AWS のサービスを選択して、利用する

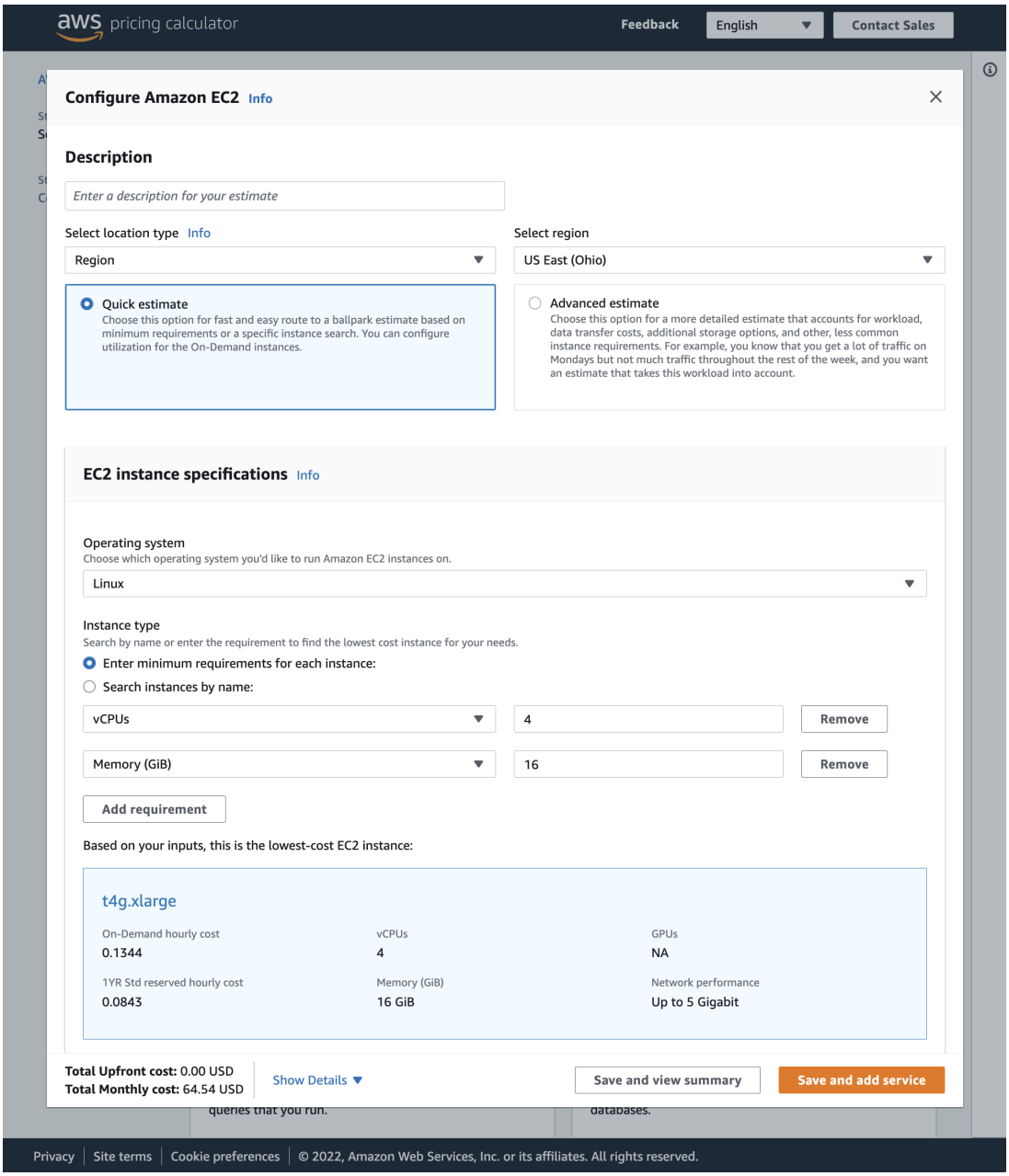

図 2.5　AWS Pricing Calculator を利用した Amazon EC2 のコスト計算

サービスの種類や数、利用期間、地域、使用量などを設定することで、クラウドサービス利用者はサービスの月額コストを計算できる。

一般にクラウドコンピューティング上のアプリケーションは、複数のクラウドリソースを組み合わせて 1 つのシステムが構築される。当コスト見積もりツールにおいても、各クラウドサービスの設定からクラウドリソースの月額コストを計算し、各クラウドリソースの月額コストを合計することでクラウドサービス上のシステムのコストを見積もれるよう

になっている。各クラウドサービスにおけるコスト見積もりの計算の元となるクラウドリソース利用量に関しては、クラウドサービス利用者が直接入力する必要があるため、当該アプリケーションの利用の増減に応じてクラウドリソースがどれだけ必要になるかはクラウドサービス利用者自身が推測する必要がある。

クラウドリソースを新しく利用する際にどれだけのコストがかかるかは、クラウドリソースの利用量がどれだけかの推測に依存する。しかし一般にクラウドサービス上に構築されるシステムは、複数のクラウドリソースが相互連携しながら、1 つのアプリケーションとして振る舞っている。クラウドサービス利用者が構築したシステムでどれだけのクラウドリソースが必要なのか、もっとも単純な推測方法は、システムを運用するなかでクラウドリソースをモニタリングし、システム負荷に応じたスケーリングをするなかで経験的にシステム内でのクラウドリソースの割合を把握する方法である。得られた経験からクラウドリソースの費用量を推測し、クラウドサービスプロバイダから提供されるコスト管理ツールに入力することで、クラウドサービス利用者はクラウドリソースのコストがいくらかになりそうかを把握することができる。

しかしシステムが複雑である場合や、ビジネスの立ち上げ段階である場合など、クラウドリソースの必要量を把握することが難しいケースの場合、クラウドサービス利用者が当該システムのなかでどれだけのクラウドリソースを必要とするかを把握することは難しい。クラウドコンピューティングではリアクティブなスケーリングにより、システムを構成するクラウドリソースはシステムの負荷に応じて動的にプロビジョニングされる。そのため、ある時点におけるクラウドリソース利用量が、将来的にも継続的に必要であるとは必ずしも言えない。またクラウドサービス上のシステムは、複数のクラウドリソースが相互通信することで成り立っており、構成するクラウドリソースが多くなると、ネットワークの複雑性から相互関係を把握しながら特定のクラウドリソース利用量を把握するのは困難になる。システムの負荷に応じたクラウドリソース利用量を把握するためには、システムの負荷に応じて各クラウドリソースがどれだけの量を必要とするかの関係性を把握する必要がある。

2.8 リソースの利用量の推定

2.8.1 解析的なアプローチ

システムの挙動を予測する方法のひとつに、解析的なアプローチがある [SKD16, AEMM13, BCS19, BCS19, AAP+13]。解析的なアプローチでは、グラフ理論や待ち行列理論などの解析手法が使われている。たとえば [SKD16, AEMM13, BCS19] ではグラフ理論が、[AAP+13] では待ち行列理論を用いた解析が行われている。第 3 章以降で提案する本論のコスト推定のアプローチにおいても、このグラフ理論を用いた手法を利用している。

ネットワークトポロジーにおけるノード間の関係の表現には行列表現を用いることができる。たとえば Sujan 他 [SKD16] はコンピューティング環境におけるスケジューリング機構の議論のなかで、クラウドサービスのネットワーク帯域幅を取り扱うために重み付き無向グラフ (Weighted Undirected Graph) を用いている。重み付き無向グラフとは、頂点を結ぶ辺に関係や情報の重要度を示す重みを持たせた無効グラフである。他には Abedifar 他 [AEMM13] が光ネットワークにおける RWA (Routing and Wavelength Assignment) 問題の定式化のなかで物理ネットワークの各ノード間のリンクの有無に無向グラフ (Undirected Graph) を用い、隣接行列を用いて表現している。

またアプリケーションはさまざまなノードから構成されるネットワークトポロジーを形成しており個々のノードが相互に関係している。複数ノードから構成されるプリケーションのコスト推定手法にはグラフ理論を用いたものがある。たとえば Brogi 他 [BCS19] はアプリケーションを構成するコンポーネントの依存関係を有向グラフ (Directed Graph) で表し、さまざまなタイプのコストをノードに関連付けることによって、複数のコンポーネントから構成されるアプリケーション全体のコスト見積もりを行っている。

ノード間のモデリングに用いられる理論に待ち行列理論もある [VST+14]。待ち行列理論はシステムやプロセスで待ち行列が生じる状況を分析するための数理理論である。待ち行列理論はハードウェアまたはソフトウェアの競合状態を記述するためにシステムモデリングで利用されてきた。待ち行列理論は、身近なところでは、たとえばサーバやルータなどのバッファ設計などに応用されている。代表的な待ち行列のモデルには、単一サーバで到着過程がポアソン過程 (M) でサービス時間が指数分布 (M) に従う M/M/1 待ち行列

や、k つのサーバで到着過程がポアソン過程 (M) でサービス時間が指数分布 (M) に従う M/M/k 待ち行列がある。クラウドコンピューティングの文脈においては、SLA を満たすためのパフォーマンス分析の文脈で用いられることがある。

複数のコンポーネントにより構成されるアプリケーションを待ち行列モデルでモデル化するときは、待ち行列ネットワークで表現できる [BGDMT06]。待ち行列ネットワークとは、複数の待ち行列が組み合わされたネットワークモデルである。たとえばシステムやアプリケーションでは、CPU やネットワーク帯域などの物理リソースまたはソフトウェアバッファや接続プールなどを、待ち行列ネットワークの要素 (待ち行列) として表すことができる。クラウドサービスの管理におけるクラウドリソース割り当てのポリシーに、待ち行列ネットワークを利用した例 [AAP+13] がある。

2.8.2 シミュレーションを用いたアプローチ

クラウドサービス上のワークロードに応じてクラウドリソースがどのように割り当てられるかを把握するアプローチに、シミュレーションを用いる方法がある。シミュレーションにもとづくアプローチでは、クラウドサービスにおけるクラウドリソースの割り当てメカニズムをモデリングし、シミュレーションを実行することでワークロード下における当該システムのパフォーマンスを観察している。クラウドシミュレーションを行うツールでは CloudSim [CRB+11] が有名である。他にも CloudSim を拡張する形で、様々なシミュレーションのアプローチが提案されている [SBA13, FFH12, PDCB17]。

CloudSim

CloudSim はクラウドサービスのインフラストラクチャをシミュレーションするためのフレームワークである [CRB+11] 。CloudSim ではトラフィックを生成し負荷テストのシミュレーションを行うことができるが、どちらかというと CluudSim はワークロード自体ではなく、様々な特性を持つクラウドサービス自体のシステムの検証に焦点を当てている。CloudSim は Java で実装されており [CRB+22]、クラウドサービスを用いたシステムをシミュレーションするための API を提供している。CloudSim はクラウドリソースの配置やスケジューリング、セキュリティなどをシミュレーションできる。CloudSim を使用することでクラウドサービスを用いたシステムにおけるコストやパフォーマンスを評価でき、クラウドサービスプロバイダとクラウドサービス利用者は CloudSim を用いるこ

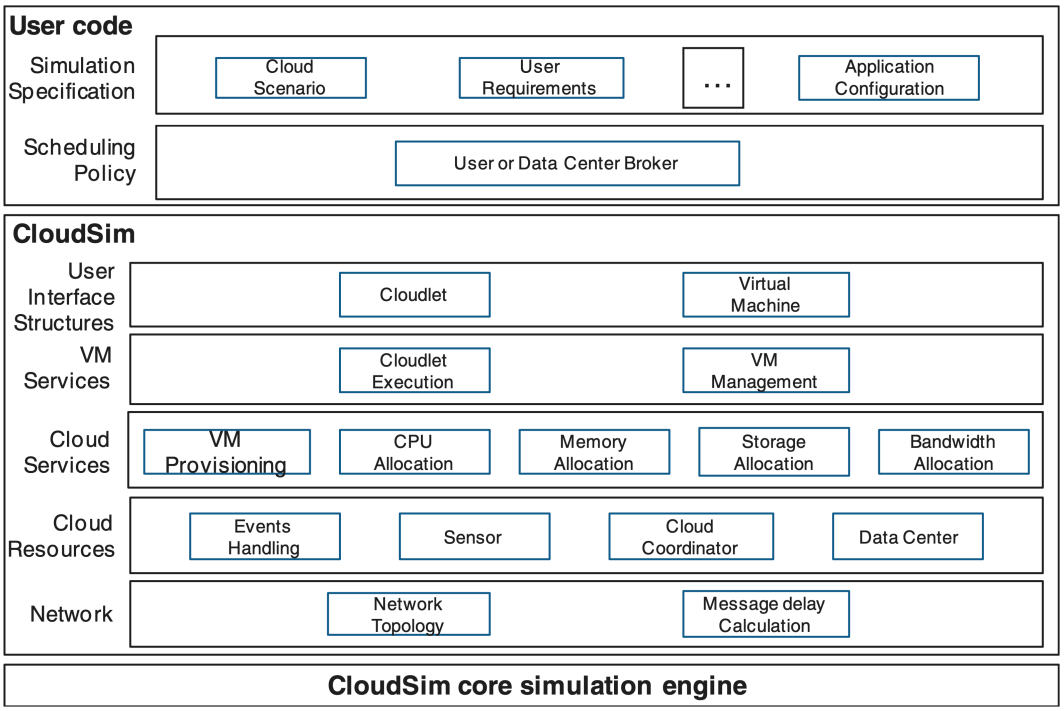

図 2.6　CloudSim のアーキテクチャー

CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms [CRB+11] より引用し抜粋

とで、システムの振る舞いが自身のビジネスにどのように影響するか検討できる。

図 2.6 は CloudSim のアーキテクチャ設計を表している。CloudSim の領域では仮想マシン、メモリ、ストレージなどの仮想化されたコンピューティングリソースのシミュレーションを行う。Cloud Services の領域は CloudSim におけるクラウドリソースの割り当てのコアとなる層であり、仮想マシンおよび仮想化されたネットワークやストレージの割り当てを行う。VM Services の領域は仮想マシンの管理タスクを実行する層であり、Cloudlet (タスクユニット) に基づいて Cloudlet Execution は VM Management と連携して仮想マシンの割り当てや解放、起動や停止といったタスクを実行する。User or Data Center Broker があるのは、CloudSim が単一の環境だけでなく、ネットワーク化されたクラウドサービス環境のモデリングとシミュレーションをサポートしているためである。これは CloudSim がデータセンター間の連携も考慮したシミュレーションのフレームワークだからである。

SimIC

SimIC はインタークラウド (Intercloud) におけるメタコンピューティング (Meta-computing) のシミュレーションを目的に、CloudCim を拡張したフレームワークである [SBA13]。インタークラウドとは複数のクラウドサービスを連携させることで、限られたクラウドサービス利用者のみが利用できる分散型および動的なコンピューティング環境である。メタコンピューティングとはデータセンターやクラウドサービスなど複数のコンピューティングリソースの集まりを組み合わせて、統合された仮想コンピューティングシステムを構築し、それらを効率的に使用することでメタコンピューティングの利用者の目的を達成するための方法を指す。

メタコンピューティングを実現するためには、複数のデータセンターやリソースを組み合わせるためのメカニズムが必要である。SimIC ではインタークラウドにおけるメタコンピューティングのシミュレーションを実現するため、SimIC では ICMS (Inter-Cloud Meta-Scheduling) を提案した。ISMS は相互運用可能なクラウドインフラストラクチャの制約のなかで、クラウドリソースの広範な提供を行う [SBKA13]。ISMS ではメタブローカー (Meta Brokers) と呼ばれる分散型のリソースの管理機能が、ヒューリスティックなパフォーマンス基準に基づいて、クラウドインフラストラクチャ間のコンピューティングリソースを調整する。

CDOSim

CDOSim はクラウド展開オプション (Cloud Deployment Options, CDO) を評価するために、CloudCim にて CDO のシミュレーションを支援するツールである [FFH12]。クラウドサービスプロバイダによるクラウドリソースの展開方法は無数にあり、クラウドサービス利用者はクラウドサービスへの移行の際に適切なプロファイルを、多くの CDO の選択肢を考慮して定義する必要がある。しかしクラウドサービスへの展開方法は無数にあるため、手動でこれらの CDO をクラウドサービスを用いて検証するのは困難である。CDOSim はこのような CDO の検証を CloudSim 上で容易にするためのツールを提供する。CDOSim を利用することで、クラウドサービスの潜在的な利用者は適切な CDO を選択する手続きを簡素化することができる。

CDOSim の基本的な使い方として挙げられるのは、既存のオンプレミス環境からクラ

ウドサービスへの移行で、適切な仮想マシンインスタンスの種類、ランタイム適応戦略の選択である。CDOSim では KDM に基づいて既存のシステムをリバースエンジニアリングし、当該アプリケーションをシミュレーションモデルに落とし込む。KDM (Knowledge Discovery MetaModel) [PCDGP11] は、既存システムに含まれる様々な物理的および論理的なソフトウェアアーティファクト (構成ファイルやランタイム、データベースなど) をメタモデル (XML など) 表現するフレームワーク (ISO/IEC 19506) である。KDM によりクラウドサービス利用者は既存システムのメタモデルを可能とし、仮想マシンの開始および停止する機能や、自動スケーリング機能、CPU 使用率など、シミュレーションを介した CDO の検証を可能とする。

2.8.3 待ち行列ネットワークモデルを用いたシミュレーション

第 2.4 節で説明したように、クラウドコンピューティングを用いたシステムを構築するときに、システムのスケーラビリティは外すことのできない考慮事項である。Vondra 他 [VŠ17] は自動スケーリングのアルゴリズムを評価するために、待ち行列ネットワーク (Queueing Netowrk, QN) モデルによる自動スケーリングのシナリオ専用のシミュレーターを実装し、シミュレーション環境のなかで自動スケーリングの検証を行えるようにした。QN モデルによる自動スケーリングのシミュレーターを実装した理由に、Vondra 他は CloudSim がインタラクティブなリアクティブスケーリングおよびレイテンシを扱うのが難しい点を指摘している。

当該シミュレーターでは使用率ベースのオートスケーラ、レイテンシベースのオートスケーラ、キューの長さベースのオートスケーラを検証できる。使用率ベースのオートスケーラはクラウドサービス一般に多く見られるオートスケーラである。使用率ベースのオートスケーラは CPU 使用率などの閾値に基づいてスケーリングを行う。レイテンシベースのオートスケーラは Google Cloud の App Engine [LLC22a] など、一部の PaaS サービスに見られるオートスケーラである。レイテンシベースのオートスケーラはレスポンスタイムなどのレイテンシに応じてスケーリングを行う。キューの長さベースのオートスケーラは Red Hat, Inc. の OpenShift [Red22] など一部の PaaS プラットフォームで見られるオートスケーラである。レイテンシベースのオートスケーラでは平均待ち行列の長さでスケーリングを行う。Vondra 他は QN モデルのシミュレーションによる検証のなかで、レイテンシに基づいてスケールアップし、キューの長さに基づいてスケールダ

ウンすることで、低コストで安定性の高いオートスケーラが得られると説明している。また Vondra 他は小規模な検証ではシミュレーションは不要で、ロードジェネレーターとパフォーマンス監視ツールを用いた検証の方が多くのシミュレーションよりも多くのさまざまな変数を観測できるとも説明している。

2.8.4 ContainerCloudSim

ContainerCloudSim は CloudSim の拡張機能であり、コンテナ技術を用いたクラウドサービスにおけるスケジュールおよび割り当てポリシーのパフォーマンスを評価するために実装された [PDCB17]。クラウドコンピューティングにおいてコンテナ技術は、クラウドサービスを構成する要素として主要な技術要素の 1 つである。第 2.6 節においても、コンテナ技術ににより仮想マシンベースと比べてきめ細かいスケーリングをすることができることを説明した。ContainerCloudSim を用いることで、ハイパーバイザベースの仮想化とコンテナベースの仮想化を並べて、双方の仮想化によるクラウドリソースの管理手法を検討することができる。

ContainerCloudSim ではコンテナの管理環境に CaaS (Container as a Service) を想定している。CaaS とはコンテナ技術を用いてアプリケーションの実行環境を抽象化し、CaaS の利用者にコンテナの実行環境と運用環境を提供するクラウドサービスである [HMA19]。ContainerCloudSim では仮想マシン上でコンテナを実行する配置モデルを採用している。ContainerCloudSim の先行研究 [PDCB17] ではコンテナのオーバブッキング、コンテナの統合、コンテナの配置のユースケースが検証されており、コンテナ技術を用いたクラウドサービスでのコンテナの管理手法の評価において ConotainerCloudSim が有効であることを示した。

2.9 本論におけるコスト推定のアプローチ

クラウドサービス利用者はクラウドコンピューティングを利用することで物理的なリソースの管理から解放され、必要に応じてクラウドリソースをクラウドサービスプロバイダが管理するリソースプールから割り当てることができる。これはクラウドコンピューティングの特徴である、オンデマンドセルフサービス、幅広いネットワークアクセス、リソースプーリング、迅速な弾力性、測定されたサービスが、仮想化技術により実現されて

いるためである。クラウドサービス利用者は、クラウドサービスプロバイダが提供するモニタリングの機能や動的プロビジョニングを組み合わせることで、システムの負荷に応じて割り当てるクラウドリソースをスケーリングすることができ、システムにかけるコストの最適化を図ることができる。

またクラウドコンピューティングでは IaaS、PaaS、SaaS、その他多くサービスモデルが、多くのクラウドサービスプロバイダにより提供されている。サービスモデルごとにサービスの抽象度が異なり、より抽象度が高いサービスであればあるほど、コンピューングリソースはより割り当てコストが低く、またコスト構造も物理インフラストラクチャとは切り離された柔軟なものになる。また近年ではコンテナ技術の活用によって、より細かい粒度でクラウドリソースの割り当てと解放ができるようになり、クラウドサービスの課金モデルに従量制を採用しているクラウドサービスプロバイダが多いことから、自動スケーリングを利用することでシステムにかけるコストの最適化をより効果的に図ることができるようになっている。

クラウドコンピューティングのコスト把握をする上でポイントとなるのが、システム負荷に対してクラウドリソースをどれだけ利用するのかを推定することである。すでに利用しているクラウドリソースに関しては、クラウドサービスプロバイダからの請求書や提供するコスト管理ツールを介して、クラウドサービス利用者はシステムおよびクラウドリソースのコストを把握することができる。これはクラウドサービス上でクラウドリソースの使用状況をモニタリングし、収集、計測することで実現している。しかしビジネスの立ち上げ時期など、システムの稼働履歴がない状況では、利用するであろうクラウドリソースを推定し、クラウドサービスプロバイダが定義した価格体系に基づき、アプリケーション全体のコストを求める必要がある。

先行研究ではクラウドサービス上のリソースの振る舞いを理解する方法として、解析的なアプローチとシミュレーションを用いたアプローチがある。解析的なアプローチではグラフ理論や待ち行列理論を使ったシステムの表現が挙げられる。システムのネットワークトポロジーをグラフとして解釈し、QoS への影響を確率分布を用いてモデル化している。シミュレーションを用いたアプローチでは、クラウド上に展開するシステムをシミュレーターに反映することで、クラウドリソースの割り当てや解放の振る舞いをシミュレーターから把握しようとする。

しかしビジネスの立ち上げ時期など、システムの稼働履歴がない状況でこれらの手法が

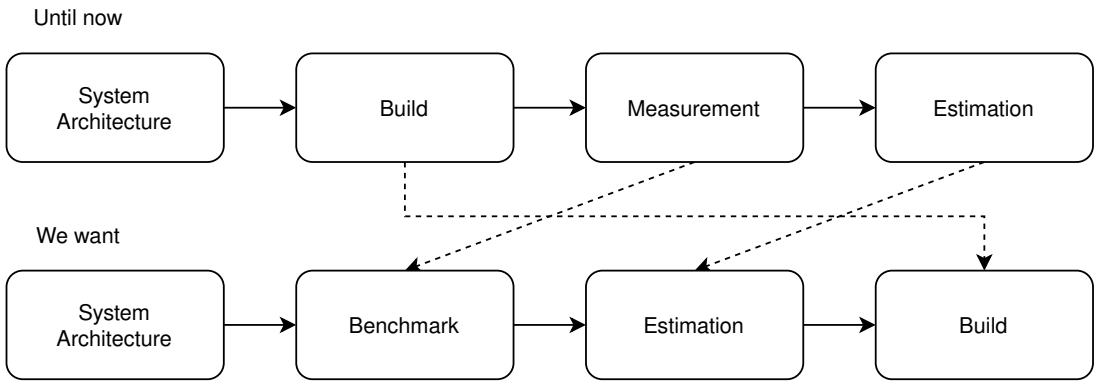

図 2.7　設計前段階のコスト見積り

そのまま用いられることは少ない。なぜならばシステムの稼働履歴がない状況では、アプリケーション全体でクラウドリソースがどれだけ利用されるかを把握していることは少なく、また多くのクラウドサービスではクラウドリソースの詳細な振る舞い (当該クラウドサービスのインフラストラクチャがどのようなロジックで動いているか) がブラックボックスになっているからである。またビジネスの立ち上げ時期などはシステムの規模も小さく、十分な検証時間が得られないことも多々ありえる。一方でクラウドサービスを利用している場合、本来物理インフラストラクチャを運用する際には考慮すべきであった物理的なコンピューティングの制約を意識しなくてよいというメリットがある。

そこで本論ではビジネスの立ち上げ時期など、クラウドコンピューティングの特性を踏まえた上で、システムの稼働履歴がない状況に短時間で妥当なコストを推定するための手法を提案する。本論が対象とするビジネスのステージは、図 2.7 で示すシステムの詳細設計前である。本提案手法は、解析的アプローチをベースに有向非巡回グラフ（Directed Asyclic Graph, DAG）を用いた、簡易なコスト見積手法である。当コスト推定手法で事前に用意する情報は、システムのネットワークトポロジーと隣接する 2 つのクラウドリソースの関係のみである。

第 3 章では DAG を用いたコスト推定手法を説明する。アプリケーションを構成するコンポーネントに負荷が生じたときに、クラウドリソース利用量が増加し、クラウドリソース利用量からコストが得られることを表現する。アプリケーションのコンポーネント間の関係をネットワークトポロジーとしてグラフ化し、行列表現を用いて表現する流れを説明する。アプリケーションのコンポーネント間の関係は 2 つのコンポーネントのみで完結しておらず、相互依存が連鎖している可能性がある。当コスト推定手法では行列の積和計

算を行うことで、任意の 2 コンポーネント間のクラウドリソースの関係を求める。システム負荷に対するクラウドリソース利用量を推定できたところで、クラウドリソース利用量から最終的にアプリケーション全体のコストをどのように計算するかを説明する。

第 4 章では、ケーススタディに第 3 章で提案したコスト推定手法を用いて、どのようにコスト推定プロセスを進めていくか説明する。第 4 章のケーススタディでは実際の IaaS を利用して議論を進める。ケーススタディでは実際にシステムをクラウドサービス上に構築し、システム負荷を変動したときに、前提としているクラウドサービスのスケーリングがどれほど適用できるかを確認する。そしてクラウドコンピューティングの特徴と実際の結果が、当コスト推定手法により得られたリソースの利用量に対してどのような誤差を生み出すが議論する。

第 5 では、サーバレスコンピューティングに提案手法を適用し、その有効性を示す。サーバレスコンピューティングで特徴的なのは、サーバの存在がサービスから、時にはコストからも隠蔽されていることである。そのためリクエストごとの料金など、あくまでサービスとして抽象化されたコストモデルのなかでコスト推定する必要がある。抽象化されたコストモデルにより、ユーザーの需要に応じたきめ細かいコスト推定が実現できることを示し、当コスト推定手法の活用シーンにおいて、どのような影響をもたらすかを議論する。

第 3 章

有向非巡回グラフを用いたコスト推定手法

クラウドコンピューティングを用いるシステムでは、複数のコンポーネントが組み合わさることで 1 つのシステムを構成する。これは構成要素に分離することで開発・管理・運用を容易するためである。複数のコンポーネント (マルチコンポーネント) で構成されたアプリケーションでは各コンポーネントが互いに連携している。そのためコスト推定ではこの相互連携するコンポーネントの関係を加味した推定プロセスが必要である。しかしコンポーネントが多くなればなるほど、計算プロセスは煩雑になり、全体を把握しながらシステム全体の負荷に応じたコストを推定することは困難になる。

本章で説明する提案手法では相互連携するコンポーネントの関係を式で表し、定まった計算プロセスを用いることで簡潔にコストを推定する事を目標とする。コンポーネントの関係を式で表す手法に有向非巡回グラフ (Directed Acyclic Graph, DAG) を用い、システムの使用量に応じた見積もりを可能としている。DAG を用いた計算プロセスにより得られるコンポーネント間の関係から、システムに与えられる使用量とシステム全体の負荷量の関係性を推定する。

3.1 イベント量とクラウドリソース利用量

現代のシステムにおいて、システムやアプリケーションは多数のコンポーネントから構成される。このように多数のコンポーネントから構成されるアプリケーションをマルチコンポーネントアプリケーションという。各コンポーネントは個別に開発され、独立して動

作する。各コンポーネントが組み合わさることで、アプリケーション全体を構成する。アプリケーションをコンポーネントに分離して管理することで、柔軟にアプリケーションを構築および運用できる。

クラウドサービス上に構築されるアプリケーションにおいても多数のコンポーネントから構成するのはオンプレミスと同様である。マルチコンポーネントアプリケーションでは、アドレスを付与されたコンポーネント (ノード) が相互通信することでアプリケーションを構成する。各ノードはそれぞれ特定の目的を持っており、アプリケーションを構成するリソースである。ノードは特定の目的を持ったリソースであり、スケールアウトする仮想マシン群などもノードとして扱うことができる。

ノードはイベントによって特定の計算処理を行う。イベントとは特定のアクションや状況の変化が発生したことを意味する。インターネットショッピングで購入ボタンを押したとき、インターネット上の Web API にリクエストするアプリケーションを仮定する。たとえば購入ボタンが押されることを条件に Web API へリクエストが行われたとき「イベント」が発生したと言う。

イベントによりノードで利用するクラウドリソースの量は変化する。ノードが仮想サーバであれば、仮想サーバの CPU 時間やメモリ使用量がイベント量に応じて変化する。仮想ストレージや仮想ネットワークであれば、イベント量に応じてストレージ書き込み量やネットワーク帯域などのリソースが利用される。本論ではイベントによりノードが利用するクラウドリソースを $\mathrm{med}(e_t)$ で表す。e_t は時間単位あたりのイベント量、med はイベント量 e_t からノードが利用するクラウドリソースを求める関数である。

$$\mathrm{med}(e_t) = \begin{bmatrix} \text{CPU 時間} \\ \text{メモリ使用量} \\ \text{ストレージ書き込み量} \\ \text{ネットワーク帯域} \end{bmatrix} \tag{3.1}$$

ノードが仮想マシンである場合、式 (3.1) のように右辺は仮想サーバの CPU 時間やメモリ使用量、ストレージ書き込み量やネットワーク帯域などになる。仮想マシンのようにリソースが提供する計算資源に上限がある場合、イベントの増減によりノードの負荷が高くなり、ノードの処理能力が限界に達することがある。一般にクラウドサービスを使ったシステムでは、自動スケーリングによりノードが高負荷になると追加のクラウドリソースを割り当てる (図 3.1)。またノードが低負荷になった場合には、不要なリソースを解放する。

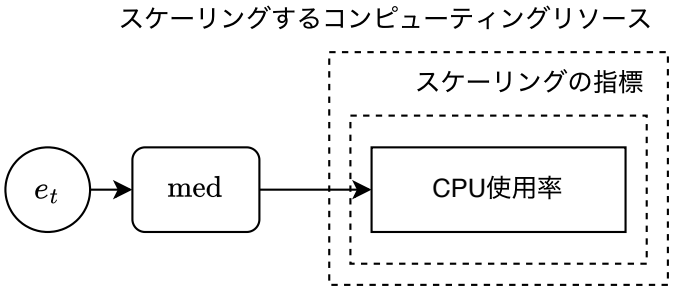

図 3.1　リソースの利用とリソースの割り当て

クラウドコンピューティングのスケーリングでは、CPU 使用率などをスケーリングの指標に用い、スケーリングの指標を基準に割り当てるリソースが増減する。たとえば仮想マシンの CPU 使用率 60% 以上をスケーリングの閾値にした場合、CPU 使用率が 60% を超えたときに仮想マシンインスタンスが 1 台追加される。またスケーリングの指標が複数ある場合、すべてのスケーリング指標を満たすようリソースが割り当てられる。たとえば仮想マシンで CPU 使用率とメモリ使用率をスケーリング指標にしている場合、CPU 使用率とメモリ使用率の双方を満たすリソースが割り当てられる。

ノードで利用するクラウドリソースの量に合わせて割り当てられるリソースを $\mathrm{res}(\mathrm{med}(e_t))$ で表す。ノードを仮想マシンで構成する場合、割り当てられるリソースは式 (3.2) で表すことができる。イベント量 e_t の増減に伴い、利用するクラウドリソース $\mathrm{med}(e_t)$ が得られ、割り当てられるリソースが変化する。

$$\mathrm{res}(\mathrm{med}(e_t)) = \text{仮想マシンインスタンスの種類} \times \text{単位時間あたりのインスタンス数} \quad (3.2)$$

3.2　ノードのコスト

クラウドコンピューティングのコストモデルは、クラウドサービスプロバイダによって定義される。各社のクラウドコンピューティングのサービス料金は、その特徴や利用目的に応じて異なる価格設定がされている。クラウドコンピューティングのコストは、基本的には仮想サーバの利用料やストレージ利用料、ネットワーク転送料などから構成される。クラウドコンピューティングのサービスの料金は、一般的にはそのサービスを提供する企業によって公開されている。そのためノードに割り当てるクラウドクラウドリソース利用量によって、ノードごとにかかるコストは求めることができる。ノードにおけるクラウド

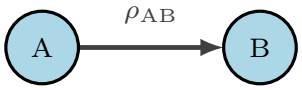

図 3.2　2 つのノード間の関係

$$
\begin{array}{ccc}
e_t & \dashrightarrow & \mathrm{med_A}(e_t) \longrightarrow \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\
 & & \downarrow \rho_{\mathrm{AB}} \\
 & & \mathrm{med_B}(e_t) \longrightarrow \mathrm{cost_B}(\mathrm{med_B}(e_t))
\end{array}
$$

図 3.3　2 つのノード間の関係式

リソース利用量からノードのコストが定り、関数 res を関数 cost に含め、ノードのコストを式 (3.3) で表す。

$$\mathrm{cost}(\mathrm{med}(e_t)) = \text{単位時間あたりのイベント量に伴うクラウドリソースのコスト} \quad (3.3)$$

式 (3.2) で得られるコンピューティング使用量と、クラウドサービスプロバイダが定義した価格設定により、単位時間あたりのクラウドリソースのコストが得られる。あるノードが仮想マシンで構成されている場合、利用するクラウドリソースが仮想マシンインスタンス 2 台であれば、仮想マシンインスタンスの料金 × 2 の利用料がノードのコストになる。

3.3　2 つのノード間の関係

マルチコンポーネントアプリケーションの基本単位はノードとノード間の関係である。一般にクラウドコンピューティングを用いたシステムの場合、複数のノードの組み合わせてシステムが構成される。そのためノード間の関係によっては、あるノードにかかるイベント量とクラウドリソース利用量が、他ののノードの影響を受けたものである場合がある。マルチコンポーネントアプリケーション全体のコストを把握するためにはノード間の関係を考慮しなければならない。

最も単純なノード間の関係は 2 つのノードで構成されるアプリケーションである。図 3.2 は 2 つのノードで構成されたアプリケーションのネットワークトポロジーを表している。ノード A からノード B への矢印は、2 つのノードの関係を表す。矢印の方向はイベントの方向を表しており、図 3.2 の場合、ノード A からノード B に向けて ρ_{AB} の関

係があることを表す。

図 3.2 の関係を、イベント量 e_t におけるクラウドリソース利用量および各ノードのコストの関係を図 3.3 で表す。e_t は単位時間あたりのイベント量である。$\mathrm{med_A}(e_t)$ はノード A におけるクラウドリソース利用量、$\mathrm{med_B}(e_t)$ はノード B におけるクラウドリソース利用量を表す。$\mathrm{cost_A}(\mathrm{med_A}(e_t))$ はイベント量 e_t あたりのノード A のコスト、$\mathrm{cost_B}(\mathrm{med_B}(e_t))$ はイベント量 e_t あたりのノード B のコストを表す。ρ_{AB} はノード A からノード B への関係、つまりノード A で利用するクラウドリソースに対してノード B でどれだけのクラウドリソースが利用されるかを表している。

ノード A で利用するクラウドリソース $\mathrm{med_A}(e_t)$ と、ノード A からノード B の関係 ρ_{AB} を用いて、ノード B で利用するクラウドリソース $\mathrm{med_B}(e_t)$ を次のように表す。

$$\mathrm{med_B}(e_t) = \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t) \tag{3.4}$$

式 (3.4) によりアプリケーションにかかるイベント量 e_t からノード A とノード B で利用するクラウドリソースを得る。たとえばイベント量 e_t に対してノード A の仮想マシンインスタンスが 1 台、ノード B の仮想マシンインスタンスが 2 台必要なネットワークトポロジーなら、ノード A からノード B の関係を $\rho_{\mathrm{AB}} = 2$ と定義することで、$\mathrm{med_B}(e_t) = 2 \times \mathrm{med_A}(e_t)$ と表す。

2 つのノードで構成されるアプリケーションのコストは 2 つのノードのコストの合計である。図 3.2 より、ノード A のコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t))$、ノード B のコストは $\mathrm{cost_B}(\mathrm{med_B}(e_t))$ であるため、2 つのノードで構成されるアプリケーションのコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t))+\mathrm{cost_B}(\mathrm{med_B}(e_t))$ である。式 (3.4) により、$\mathrm{med_B}(e_t)$ は $\rho_{\mathrm{AB}} \times \mathrm{med_B}(e_t)$ で置き換えられるため、アプリケーション全体のコスト $\mathrm{Cost}(e_t)$ は次のように表せる。

$$\mathrm{Cost}(e_t) = \mathrm{cost_A}(\mathrm{med_A}(e_t)) + \mathrm{cost_B}(\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \tag{3.5}$$

一例として 2 層の仮想マシン群で構成されるアプリケーションを式 (3.5) で表現する。1 層目の仮想マシン群がリクエストを受け取ると、2 層目の仮想マシン群に問い合わせをするアプリケーションである。そして各ノードの仮想マシン群のスケーリングの指標を CPU 時間にする。仮想マシンインスタンスの処理能力が不足する前にスケーリングの指標 (CPU 時間) によって仮想マシン群がスケーリングする。簡単のため、各ノードでは同じ仮想マシンインスタンスを利用し、同じスケーリング条件が設定し、ロードバランサの利用時間やネットワーク利用量は考えないものとする。図 3.2 の関係式に当てはめると、

単位時間あたりのイベント量と各ノードのクラウドリソース利用量、各ノードのコストは次のように意味付けられる。

e_t	: 1 分あたりのリクエスト数
$\mathrm{med_A}(e_t)$	: リクエスト数に応じたノード A の CPU 時間
$\mathrm{med_B}(e_t)$	: リクエスト数に応じたノード B の CPU 時間
$\mathrm{cost}(\mathrm{med_A}(e_t))$	: リクエスト数に応じたノード A の仮想マシンインスタンスの料金
$\mathrm{cost}(\mathrm{med_B}(e_t))$	: リクエスト数に応じたノード B の仮想マシンインスタンスの料金
ρ_{AB}	: ノード A の仮想マシンで利用する CPU 時間を 1 とした場合のノード B の仮想マシンで利用する CPU 時間の比率

各ノードの仮想マシン群が同じインスタンスかつ同一のスケーリング条件であることから、ノード A とノード B で同じコスト関数 cost になっている。そして概算として 1,000 リクエストあたりのノード A が利用する CPU 時間を 1.5 秒、ノード A とノード B で利用する CPU 時間の比が 1:2 であるとする。すると式 (3.5) により、$\mathrm{med_B}(e_t))$ は $\rho_{\mathrm{AB}} \times \mathrm{med_B}(e_t)$ で置き換えられるため、アプリケーション全体のコスト $\mathrm{Cost}(e_t)$ を次のように表現する。

$$\mathrm{Cost}(e_t) = \mathrm{cost_A}\left(1.5 \times \frac{e_t}{1,000}\right) + \mathrm{cost_B}\left(2 \times 1.5 \times \frac{e_t}{1,000}\right) \tag{3.6}$$

各ノードのコストは割り当てする仮想マシンインスタンスと、そのインスタンスの料金によって表現できる。ノード A とノード B は同じインスタンスを利用しているため、インスタンス 1 台あたりの価格は p である。そして関数 cost のスケーリング条件を CPU 利用可能時間 100 秒の CPU 使用率 60% とすると、次のように表現される。

$$\begin{aligned}\mathrm{Cost}(e_t) &= \frac{p}{0.6 \times 100}\left(1.5 \times \frac{e_t}{1000}\right) + \frac{p}{0.6 \times 100}\left(2 \times 1.5 \times \frac{e_t}{1000}\right) \\ &= 0.000075 p e_t\end{aligned} \tag{3.7}$$

3.4 3 つのノード間の関係

3 つのノードで構成されるネットワークトポロジーでは、複数のノード間の関係を取り扱わなければならない。ノードが 3 つある場合、2 つ以上のノード間の関係がある。そのため基本となる 3 つのノード間の関係を以下にまとめる。

- 直列な 3 つのノード間の関係

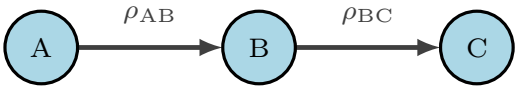

図 3.4　直列な 3 つのノード間の関係

$$
\begin{array}{ccc}
e_t & \dashrightarrow & \mathrm{med}_{\mathrm{A}}(e_t) \longrightarrow \mathrm{cost}_{\mathrm{A}}(\mathrm{med}_{\mathrm{A}}(e_t)) \\
 & & \downarrow \rho_{\mathrm{AB}} \\
 & & \mathrm{med}_{\mathrm{B}}(e_t) \longrightarrow \mathrm{cost}_{\mathrm{B}}(\mathrm{med}_{\mathrm{B}}(e_t)) \\
 & & \downarrow \rho_{\mathrm{BC}} \\
 & & \mathrm{med}_{\mathrm{C}}(e_t) \longrightarrow \mathrm{cost}_{\mathrm{C}}(\mathrm{med}_{\mathrm{C}}(e_t))
\end{array}
$$

図 3.5　直列な 3 つのノードの関係式

- 分岐する 3 つのノード間の関係
- 合流する 3 つのノード間の関係

3.4.1　直列な 3 つのノード間の関係

3 つのノードが直列な関係を持つ場合、あるノード間の関係がもう他方のノード間の関係に影響を及ぼす。図 3.4 は 3 つのノードが直列に配置されたアプリケーションのネットワークトポロジーを表している。ρ_{AB} はノード A からノード B への関係を表す。ρ_{BC} はノード B からノード C への関係を表す。注意すべきは、関係 ρ_{BC} が関係 ρ_{AB} の影響を受ける点である。

図 3.4 の関係を、イベント量 e_t におけるクラウドリソース利用量および各ノードのコストの関係に分解して図 3.5 に示す。e_t は単位時間あたりのイベント量である。$\mathrm{med}_{\mathrm{A}}(e_t)$ はノード A におけるクラウドリソース利用量、$\mathrm{med}_{\mathrm{B}}(e_t)$ はノード B におけるクラウドリソース利用量、$\mathrm{med}_{\mathrm{C}}(e_t)$ はノード C におけるクラウドリソース利用量を表す。$\mathrm{cost}_{\mathrm{A}}(\mathrm{med}_{\mathrm{A}}(e_t))$ はイベント量 e_t あたりのノード A のコスト、$\mathrm{cost}_{\mathrm{B}}(\mathrm{med}_{\mathrm{B}}(e_t))$ はイベント量 e_t あたりのノード B のコスト、$\mathrm{cost}_{\mathrm{C}}(\mathrm{med}_{\mathrm{C}}(e_t))$ はイベント量 e_t あたりのノード C のコストを表す。

直列な 3 つのノード関係は、2 つのノード間の関係を 1 ステップ伸ばしたものである。つまり式 (3.4) と同様に、ノード C が利用するクラウドリソースとノード B が利用する

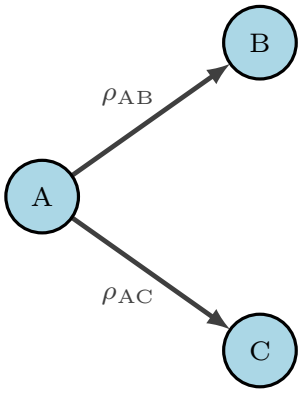

図 3.6　分岐する 3 つのノード間の関係

クラウドリソースの関係は $\mathrm{med_C}(e_t) = \rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t)$ となる。また式 (3.4) により、ノード B が利用するクラウドリソースは $\mathrm{med_B}(e_t) = \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$ であるため、ノード C が利用するクラウドリソース $\mathrm{med_C}(e_t)$ は次のように表せる。

$$\begin{aligned} \mathrm{med_C}(e_t) &= \rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t) \\ &= \rho_{\mathrm{BC}} \times \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t) \end{aligned} \tag{3.8}$$

直列な 3 つのノードで構成されるアプリケーションのコストは 3 つのノードのコストの合計である。図 3.4 より、ノード A のコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t))$、ノード B のコストは $\mathrm{cost_B}(\mathrm{med_B}(e_t))$、ノード C のコストは $\mathrm{cost_C}(\mathrm{med_C}(e_t))$ であるため、3 つのノードで構成されるアプリケーションのコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t)) + \mathrm{cost_B}(\mathrm{med_B}(e_t)) + \mathrm{cost_C}(\mathrm{med_C}(e_t))$ である。式 (3.8) により、$\mathrm{med_B}(e_t)$ が $\rho_{\mathrm{AB}} \times \mathrm{med_B}(e_t)$、$\mathrm{med_C}(e_t)$ は $\rho_{\mathrm{BC}} \times \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$ で置き換えられるため、アプリケーション全体のコスト $\mathrm{Cost_{sequenced}}(e_t)$ は次のように表現できる。

$$\begin{aligned} \mathrm{Cost_{sequenced}}(e_t) = &\ \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\ &+ \mathrm{cost_B}(\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \\ &+ \mathrm{cost_C}(\rho_{\mathrm{BC}} \times \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \end{aligned} \tag{3.9}$$

3.4.2　分岐する 3 つのノード間の関係

3 つのノード間で分岐する関係を持つ場合、あるノードは複数の関係を持つ。図 3.6 は 3 つのノードのうち 1 つのノードから 2 つのノードに関係があるアプリケーションのネットワークトポロジーを表している。ρ_{AB} はノード A からノード B への関係を表す。ρ_{AC}

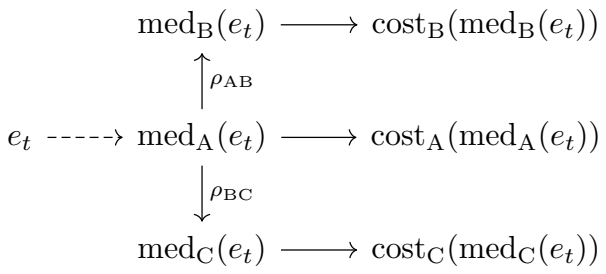

図 3.7　分岐する 3 つのノード間の関係式

はノード A からノード C への関係を表す。

図 3.6 を表現したイベント量 e_t におけるクラウドリソース利用量および各ノードのコストの関係式を図 3.7 に示す。e_t は単位時間あたりのイベント量である。$\mathrm{med_A}(e_t)$ はノード A におけるクラウドリソース利用量、$\mathrm{med_B}(e_t)$ はノード B におけるクラウドリソース利用量、$\mathrm{med_C}(e_t)$ はノード におけるクラウドリソース利用量を表す。$\mathrm{cost_A}(\mathrm{med_A}(e_t))$ はイベント量 e_t あたりのノード A のコスト、$\mathrm{cost_B}(\mathrm{med_B}(e_t))$ はイベント量 e_t あたりのノード B のコスト、$\mathrm{cost_C}(\mathrm{med_C}(e_t))$ はイベント量 e_t あたりのノード C のコストを表す。

分岐する 3 つのノード間の関係は、2 つのノード間の関係を複製したものとも言える。つまり式 (3.4) のノード B と同様に、ノード C が利用するクラウドリソースとノード A が利用するクラウドリソースの関係は $\mathrm{med_C}(e_t) = \rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t)$ となる。注意点としてはノード B で利用するクラウドリソース $\mathrm{med_B}(e_t) = \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$ とノード C で利用するクラウドリソース $\mathrm{med_C}(e_t) = \rho_{\mathrm{AC}} \times \mathrm{med_C}(e_t)$ では共通する 2 ノード間の関係が存在しないことである。

分岐する 3 つのノードで構成されるアプリケーションのコストは 3 つのノードのコストの合計である。図 3.6 より、ノード A のコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t))$、ノード B のコストは $\mathrm{cost_B}(\mathrm{med_B}(e_t))$、ノード C のコストは $\mathrm{cost_C}(\mathrm{med_C}(e_t))$ であるため、3 つのノードで構成されるアプリケーションのコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t)) + \mathrm{cost_B}(\mathrm{med_B}(e_t)) + \mathrm{cost_C}(\mathrm{med_C}(e_t))$ である。

ノード B で利用するクラウドリソース $\mathrm{med_B}(e_t) = \rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$ とノード C で利用するクラウドリソース $\mathrm{med_C}(e_t) = \rho_{\mathrm{AC}} \times \mathrm{med_C}(e_t)$ では共通する 2 ノード間の関係が存在しないことから、アプリケーション全体のコスト $\mathrm{Cost_{branched}}(e_t)$ は次のように表現

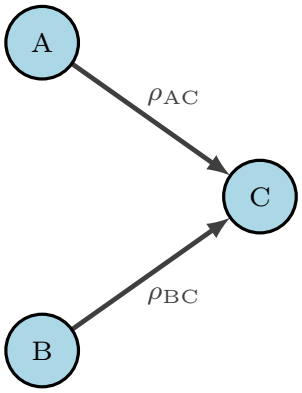

図 3.8　合流する 3 つのノード間の関係

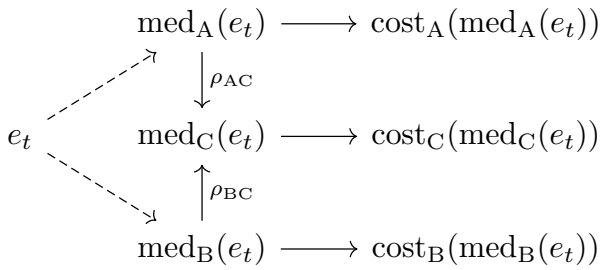

図 3.9　合流する 3 つのノード間の関係式

できる。

$$\begin{aligned}\mathrm{Cost_{branched}}(e_t) = {} & \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\ & + \mathrm{cost_B}(\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \\ & + \mathrm{cost_C}(\rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t))\end{aligned} \tag{3.10}$$

3.4.3　合流する 3 つのノード間の関係

3 つのノードが合流する関係を持つ場合、1 つのノードで 2 つの関係が合流する。図 3.8 は 2 つのノードから 1 つのノードに関係を持つネットワークトポロジーを表している。ρ_{AC} はノード A からノード C への関係を表す。ρ_{BC} はノード B からノード C への関係を表す。注意すべきは、2 つの関係 ρ_{AC} と ρ_{BC} がノード C にて混ざる点である。

図 3.8 の関係をイベント量 e_t におけるクラウドリソース利用量および各ノードのコストの関係式を図 3.9 に示す。e_t は単位時間あたりのイベント量である。$\mathrm{med_A}(e_t)$ はノード A におけるクラウドリソース利用量、$\mathrm{med_B}(e_t)$ はノード B におけるクラウドリソース利用量、$\mathrm{med_C}(e_t)$ はノード B におけるクラウドリソース利用量を表す。$\mathrm{cost_A}(\mathrm{med_A}(e_t))$

はイベント量 e_t あたりのノード A のコスト、$\mathrm{cost_B}(\mathrm{med_B}(e_t))$ はイベント量 e_t あたりのノード B のコスト、$\mathrm{cost_C}(\mathrm{med_C}(e_t))$ はイベント量 e_t あたりのノード C のコストを表す。

合流する 3 つのノード関係は、2 つのノード間の関係が重なったものであると言える。つまりノード A からの関係のみに限定したノード C が利用するクラウドリソースと、ノード B からの関係のみに限定したード C が利用するクラウドリソースを重ねる手続きが必要である。ノードとは特定の目的を持ったリソースであるため、複数の経路からのイベントを同一のものとして取り扱うことができる。そのためノード A からの関係のみに限定したノード C が利用するクラウドリソースと、ノード B からの関係のみに限定したード C が利用するクラウドリソースを合計を、ノード C が利用するクラウドリソースとして表現できる。

図 3.9 により、ノード A からの関係のみに限定したノード C が利用するクラウドリソースは $\rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t)$、ノード B からの関係のみに限定したード C が利用するクラウドリソースは $\rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t)$ である。ノード C が利用するクラウドリソースはノード A からの関係のみに限定したノード C が利用するクラウドリソースと、ノード B からの関係のみに限定したード C が利用するクラウドリソースの合計であるため、ノード C が利用するクラウドリソース $\mathrm{med_C}(e_t)$ は次のように表せる。

$$\mathrm{med_C}(e_t) = \rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t) + \rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t) \tag{3.11}$$

合流する 3 つのノードで構成されるアプリケーションのコストは 3 つのノードのコストの合計である。図 3.8 より、ノード A のコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t))$、ノード B のコストは $\mathrm{cost_B}(\mathrm{med_B}(e_t))$、ノード C のコストは $\mathrm{cost_C}(\mathrm{med_C}(e_t))$ であるため、3 つのノードで構成されるアプリケーションのコストは $\mathrm{cost_A}(\mathrm{med_A}(e_t)) + \mathrm{cost_B}(\mathrm{med_B}(e_t)) + \mathrm{cost_C}(\mathrm{med_C}(e_t))$ である。式 (3.11) により、ノード C が利用するクラウドリソース $\mathrm{med_C}(e_t)$ は、ノード A からの関係のみに限定したノード C のクラウドリソース利用量 $\rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t)$ と、ノード B からの関係のみに限定したード C のクラウドリソース $\rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t)$ 利用量の合計に置き換えることができるため、合流する 3 つのノードで構成されるアプリケーションのコストは次のように表現できる。

$$\begin{aligned} \mathrm{Cost_{merged}}(e_t) = {} & \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\ & + \mathrm{cost_B}(\mathrm{med_B}(e_t)) \\ & + \mathrm{cost_C}(\rho_{\mathrm{AC}} \times \mathrm{med_A}(e_t) + \rho_{\mathrm{BC}} \times \mathrm{med_B}(e_t)) \end{aligned} \tag{3.12}$$

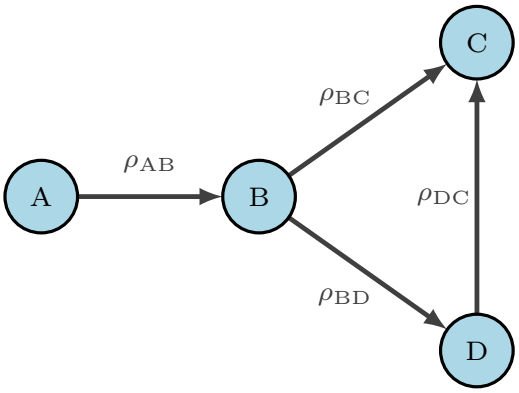

図 3.10　多ノード間の関係

3.5　多ノード間の関係

3.5.1　有向非巡回グラフを用いたネットワークトポロジーの表現

複数のノードで構成されるアプリケーションのネットワークトポロジーの表現に有向非巡回グラフ (Directed Acyclic Graph, DAG) を用いる。DAG とは、有向グラフの一種で、閉路がない有向グラフのことである。有向グラフとは頂点と辺から構成されるグラフの一種で、頂点から頂点へ向きを持つ辺で構成されている。閉路とは出発点に戻るような経路であり、巡回している経路を意味する。

複数のノードで構成するアプリケーションは、一方のノードが他方のノードに要求する関係の集まりで構成される。リクエストを受け取ったフロントエンドの Web アプリケーションが、バックエンドのデータベースに問い合わせする関係などが代表として挙げられる。そのため複数のノードで構成するアプリケーションは、ノードを点、ノード間の関係をエッジとした有向グラフで表現できる。

また複数のノードで構成するアプリケーションは閉路を持たない。なぜならば、閉路を持つアプリケーションでは、巡回している経路上のノードに対して巡回する回数だけリソースの要求が発生するからである。閉路を持つアプリケーションとは、つまり巡回している経路上のノードに対して、無限回のリソースの要求が発生するアプリケーションである。企業が構築するアプリケーションにかける予算は有限であるため、閉路を持つアプリケーションが存在することは考えにくく、本研究では扱わない。

アプリケーションのノードを点、ノード間の要求関係を辺とすることで、N つのノード

で構成されるアプリケーションを出発点を含めて $(N+1)\times(N+1)$ の重み付き隣接行列で表すことができる。重み付き隣接行列はグラフ理論において、グラフを行列で表す手法の 1 つである。重み付き隣接行列は正方行列であり、各行と各列が、グラフの点に対応している。隣接する点 i から点 j への辺が存在するとき、重み付き隣接行列では (i,j) 要素に辺の重みを用いる。隣接する点 i から点 j への辺が存在しない場合は、重み付き隣接行列の (i,j) 要素は 0 である。

重み付き隣接行列を用いることで、図 3.2 を式 (3.13) で表現できる。R は重み付き隣接行列であり、出発点を 1 行目と 1 列目に、ノード A を 2 行目と 2 列目に、ノード B を 3 行目と 3 列目に対応させる。また出発点からノード A への関係を $\rho_{\bullet\mathrm{A}}$ で表し、$R_{(1,2)}=\rho_{\bullet\mathrm{A}}$ とする。ノード A からノード B への関係は ρ_{AB} であるため $R_{(2,3)}=\rho_{\mathrm{AB}}$ である。ほかの隣接する 2 つのノード間には関係がないため、$R_{(1,2)}$ と $R_{(2,3)}$ 以外の各要素は 0 である。

$$R=\begin{bmatrix}0 & \rho_{\bullet\mathrm{A}} & 0\\ 0 & 0 & \rho_{\mathrm{AB}}\\ 0 & 0 & 0\end{bmatrix} \tag{3.13}$$

図 3.4 は重み付き隣接行列を用いて式 (3.14) に表現できる。図 3.4 は 3 つのノードを有するため、4×4 の重み付き隣接行列を用いる。ノード A からノード B の関係 ρ_{AB} とノード B からノード C の関係 ρ_{BC} があるため、各要素は $R_{(2,3)}=\rho_{\mathrm{AB}}, R_{(3,4)}=\rho_{\mathrm{BC}}$ となる。

$$R=\begin{bmatrix}0 & \rho_{\bullet\mathrm{A}} & 0 & 0\\ 0 & 0 & \rho_{\mathrm{AB}} & 0\\ 0 & 0 & 0 & \rho_{\mathrm{BC}}\\ 0 & 0 & 0 & 0\end{bmatrix} \tag{3.14}$$

図 3.7 は重み付き隣接行列を用いて式 (3.15) で表現できる。ノード A からノード B の関係 ρ_{AB} とノード A からノード C の関係 ρ_{AC} があるため、各要素は $R_{(2,3)}=\rho_{\mathrm{AB}}, R_{(2,4)}=\rho_{\mathrm{AC}}$ である。

$$R=\begin{bmatrix}0 & \rho_{\bullet\mathrm{A}} & 0 & 0\\ 0 & 0 & \rho_{\mathrm{AB}} & \rho_{\mathrm{AC}}\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix} \tag{3.15}$$

図 3.8 は重み付き隣接行列を用いて式 (3.16) に表現できる。出発点からノード A とノード B へ関係があるため、$R_{(1,2)}=\rho_{\bullet\mathrm{A}}, R_{(1,3)}=\rho_{\bullet\mathrm{B}}$ である。ノード A から

ノード C の関係 ρ_{AC} とノード B からノード C に関係 ρ_{BC} があるため、各要素は $R_{(2,4)} = \rho_{\mathrm{AC}}, R_{(3,4)} = \rho_{\mathrm{BC}}$ である。

$$R = \begin{bmatrix} 0 & \rho_{\bullet\mathrm{A}} & \rho_{\bullet\mathrm{B}} & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{AC}} \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.16}$$

複数のノードで構成するアプリケーションでも、隣接するノード間の関係を重み付き隣接行列で表す。図 3.10 は多ノードの関係を表した例である。図 3.10 を重み付き隣接行列で表すと式 (3.17) である。出発点からはノード A に関係を持ち、ノード A からノード B、ノード B からノード C、ノード B からノード D、ノード D からノード C の間に要求関係が存在する。そのため隣接行列 R の各要素は $R_{(1,2)} = \rho_{\bullet\mathrm{A}}, R_{(2,3)} = \rho_{\mathrm{AB}}, R_{(3,4)} = \rho_{\mathrm{BC}}, R_{(3,5)} = \rho_{\mathrm{BD}}, R = (5,4) = \rho_{\mathrm{DC}}$ となり、そのほかの各要素は 0 である。

$$R = \begin{bmatrix} 0 & \rho_{\bullet\mathrm{A}} & 0 & 0 & 0 \\ 0 & 0 & \rho_{\mathrm{AB}} & 0 & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} & \rho_{\mathrm{BD}} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{DC}} & 0 \end{bmatrix} \tag{3.17}$$

3.5.2 任意の 2 ノード間の関係

ホップ数が 2 となる 2 ノード間の関係、つまり一方のノード間の関係がもう一方のノード間の関係に影響を及ぼす場合、ホップ数 2 の 2 ノード間の関係を重み付き隣接行列の 2 乗で求める。式 (3.18) の $R^2_{(i,j)}$ はホップ数が 2 となるノード i からノード j への関係を表す。ホップ数が 2 であるため、2 つのノード間には最大 N つの中間ノード n が 1 層存在する。

第 3.4.1 項により、直列な 3 つのノードで構成されるアプリケーションにおけるホップ数 2 のノード間の関係は、隣接するノード間の関係の積で表す。ホップ数が 2 となる 2 ノード間の経路上に隣接する 2 つのノード間の関係が存在しないときは、隣接する 2 つのノード間の関係は 0 であるため、ホップ数 2 のノード間の関係は 0 である。ノードは単一の役割を想定するため $R^2_{(i,j)}$ は $R_{(i,n)}$ と $R_{(n,j)}$ の積和となる。

$$R^2_{(i,j)} = \sum_{n=1}^{N} R_{(i,n)} R_{(n,j)} \tag{3.18}$$

ホップ数が 3 となる 2 ノード間の経路数は、隣接行列の 3 乗で求める。式 (3.19) はホップ数が 3 となるノード i からノード j への経路数を表している。中継地点が定められた 2 点間の関係は要求元ノードから中継ノードの関係と中継ノードから要求先ノードの関係の積である。そのためホップ数が 3 となる 2 ノード間の関係は、要求元ノードから中継ノードの関係 $R^2_{(i,n)}$ と中継ノードから要求先ノードの関係 $R_{(n,j)}$ の積和である。

$$R^3_{(i,j)} = \sum_{n=1}^{N} R^2_{(i,n)} R_{(n,j)} \tag{3.19}$$

式 (3.19) の考え方を再帰的に適用することで、任意のホップ数を有するノード間の経路数を求められる。式 (3.20) はホップ数が k となるノード i からノード j への関係を表している。$R^k_{(i,j)}$ はホップ数が k となるノード i からノード j への関係である。ホップ数が 2 であれば $k = 2$ にすることで式 (3.18) が、ホップ数が 3 であれば $k = 3$ にすることで式 (3.19) が式 (3.20) から得られる。また複数のノードで構成されるアプリケーションの最大ホップ数が K であるとき、経路が存在しないため行列 R^{K+1} はゼロ行列である。

$$R^k_{(i,j)} = \sum_{n=1}^{N} R^{k-1}_{(i,n)} R_{(n,j)} \tag{3.20}$$

ホップ数を問わない任意の 2 ノード間の関係は、すべてのホップ数下における 2 ノード間の関係の和である。式 (3.21) は、2 ノード間の最大ホップ数が K であるネットワークトポロジーのノード i からノード j への関係を表している。$\hat{R}_{(i,j)}$ はノード i からノード j への関係であり、$\hat{R}_{(i,j)}$ は 1 〜 K のホップ数におけるノード i からノード j への関係 $R^k_{(i,j)}$ の和である。

$$\hat{R}_{(i,j)} = \sum_{k=1}^{K} R^k_{(i,j)} \tag{3.21}$$

より多数のノードを有するネットワークトポロジーでも式 (3.21) を用いることができる。式 (3.22) は図 3.10 において任意の 2 ノード間の関係を求めたものである。式 (3.21) の R に式 (3.17) の重み付き隣接行列 R を代入する。図 3.10 より、出発点からはノード A に関係は $\rho_{\bullet \mathrm{A}}$、ノード A からノード B の関係は ρ_{AB}、ノード A からノード D の関係は $\rho_{\mathrm{BD}}\rho_{\mathrm{AB}}$、ノード A からノード C の関係は $\rho_{\mathrm{BC}}\rho_{\mathrm{AB}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}}\rho_{\mathrm{AB}} = (\rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}})\rho_{\mathrm{AB}}$

である。式 (3.22) においても同じ 2 ノード間の関係が得られる。

$$
\begin{aligned}
\hat{R}_{(i,j)} &= \sum_{k=1}^{4} \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & 0 & 0 & 0 \\ 0 & 0 & \rho_{\mathrm{AB}} & 0 & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} & \rho_{\mathrm{BD}} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{DC}} & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} & (\rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}})\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} & \rho_{\mathrm{BD}}\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} \\ 0 & 0 & \rho_{\mathrm{AB}} & (\rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}})\rho_{\mathrm{AB}} & \rho_{\mathrm{BD}}\rho_{\mathrm{AB}} \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}} & \rho_{\mathrm{BD}} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{DC}} & 0 \end{bmatrix}
\end{aligned}
\tag{3.22}
$$

3.5.3 多ノード構成におけるクラウドリソース利用量

第 3.3 節と第 3.4 節により、各ノードのクラウドリソースのコストは、各ノードのクラウドリソース利用量によって定まる。多ノードで構成するアプリケーションにおける 2 つのノード間の関係 $\hat{R}$ は式 (3.21) によって与えられる。出発点から隣接するノードとの関係を起点となるノードのクラウドリソース利用量とすると、あるノードのクラウドリソース利用量は、出発点に対応する行とあるノードに対応する列が交わる要素の合計である。

図 3.2 を式で表すと、2 つのノード間の関係 $\hat{R}$ は式 (3.23) によって与えられる。出発点に対応する行とあるノード j に対応する列が交わる要素は $(1, j)$ である。式 (3.23) より、出発点に対応する行とあるノードに対応する列が交わる要素の合計は、ノード A が $\rho_{\bullet \mathrm{A}}$ であり、ノード B が $\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}}$ である。出発点とノード A の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}(e_t)$ とすると、ノード A のクラウドリソース利用量 $\mathrm{med}(e_t)$、ノード B のクラウドリソース利用量 $\rho_{\mathrm{AB}} \times \mathrm{med}(e_t)$ が求まる。

$$
\begin{aligned}
R &= \sum_{k=1}^{2} \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & 0 \\ 0 & 0 & \rho_{\mathrm{AB}} \\ 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} \\ 0 & 0 & \rho_{\mathrm{AB}} \\ 0 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{3.23}
$$

図 3.4 を式で表したとき、3 つのノード間の関係 $\hat{R}$ は式 (3.24) によって与えられる。出発点に対応する行とあるノード j に対応する列が交わる要素は $(1, j)$ である。式 (3.24) より、出発点に対応する行と各ノードに対応する列が交わる要素の合計は、ノード A が $\rho_{\bullet \mathrm{A}}$ であり、ノード B が $\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}}$、ノード C が $\rho_{\mathrm{BC}}\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}}$ である。出発点とノード A

の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}(e_t)$ とすると、ノード A のクラウドリソース利用量 $\mathrm{med}(e_t)$、ノード B のクラウドリソース利用量 $\rho_{\mathrm{AB}} \times \mathrm{med}(e_t)$、ノード C のクラウドリソース利用量 $\rho_{\mathrm{BC}}\rho_{\mathrm{AB}} \times \mathrm{med}(e_t)$ が求まる。

$$
\begin{aligned}
R &= \sum_{k=1}^{3} \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & 0 & 0 \\ 0 & 0 & \rho_{\mathrm{AB}} & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} & \rho_{\mathrm{BC}}\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} \\ 0 & 0 & \rho_{\mathrm{AB}} & \rho_{\mathrm{BC}}\rho_{\mathrm{AB}} \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{3.24}
$$

図 3.7 を式で表したとき、3 つのノード間の関係 $\hat{R}$ は式 (3.25) によって与えられる。出発点に対応する行とあるノード j に対応する列が交わる要素は $(1, j)$ である。式 (3.25) より、出発点に対応する行と各ノードに対応する列が交わる要素の合計は、ノード A で $\rho_{\bullet \mathrm{A}}$ であり、ノード B で $\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}}$、ノード C で $\rho_{\mathrm{AC}}\rho_{\bullet \mathrm{A}}$ である。出発点とノード A の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}(e_t)$ とすると、ノード A のクラウドリソース利用量 $\mathrm{med}(e_t)$、ノード B のクラウドリソース利用量 $\rho_{\mathrm{AB}} \times \mathrm{med}(e_t)$、ノード C のクラウドリソース利用量 $\rho_{\mathrm{AC}} \times \mathrm{med}(e_t)$ が求まる。

$$
\begin{aligned}
R &= \sum_{k=1}^{3} \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & 0 & 0 \\ 0 & 0 & \rho_{\mathrm{AB}} & \rho_{\mathrm{AC}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}} & \rho_{\mathrm{AC}}\rho_{\bullet \mathrm{A}} \\ 0 & 0 & \rho_{\mathrm{AB}} & \rho_{\mathrm{AC}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{3.25}
$$

図 3.8 を式で表したとき、3 つのノード間の関係 $\hat{R}$ は式 (3.26) によって与えられる。出発点に対応する行とあるノード j に対応する列が交わる要素は $(1, j)$ である。式 (3.25) より、出発点に対応する行と各ノードに対応する列が交わる要素の合計は、ノード A が $\rho_{\bullet \mathrm{A}}$ であり、ノード B で $\rho_{\mathrm{AB}}\rho_{\bullet \mathrm{A}}$、ノード C で $\rho_{\mathrm{AC}}\rho_{\bullet \mathrm{A}} + \rho_{\mathrm{BC}}\rho_{\bullet \mathrm{B}}$ である。出発点とノード A の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}_{\mathrm{A}}(e_t)$、出発点とノード B の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}_{\mathrm{B}}(e_t)$ とすると、ノード A のクラウドリソース利用量 $\mathrm{med}_{\mathrm{A}}(e_t)$、ノード B のクラウドリソース利用量 $\mathrm{med}_{\mathrm{B}}(e_t)$、ノード C のクラウドリソース利用量 $\rho_{\mathrm{AC}} \times \mathrm{med}_{\mathrm{A}}(e_t) + \rho_{\mathrm{BC}} \times \mathrm{med}_{\mathrm{B}}(e_t)$ が求

まる。

$$
\begin{aligned}
R &= \sum_{k=1}^{3} \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\bullet \mathrm{B}} & 0 \\ 0 & 0 & 0 & \rho_{\mathrm{AC}} \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \rho_{\bullet \mathrm{A}} & \rho_{\bullet \mathrm{B}} & \rho_{\mathrm{AC}}\rho_{\bullet \mathrm{A}} + \rho_{\mathrm{BC}}\rho_{\bullet \mathrm{B}} \\ 0 & 0 & 0 & \rho_{\mathrm{AC}} \\ 0 & 0 & 0 & \rho_{\mathrm{BC}} \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}
\tag{3.26}
$$

より多数のノードを有するネットワークトポロジーでも式 (3.23)(3.24)(3.25)(3.26) と同様にクラウドリソース利用量が求まる。式 (3.22) も図 3.10 において任意の 2 ノード間の関係を求めたものである。出発点に対応する行とあるノード j に対応する列が交わる要素は $(1, j)$ であるため、出発点とノード A の関係を $\rho_{\bullet \mathrm{A}} = \mathrm{med}(e_t)$ とすることで、次のように各ノードのリソースを得ることができる。

ノード A : $\mathrm{med}_{\mathrm{A}}(e_t)$
ノード B : $\rho_{\mathrm{AB}} \times \mathrm{med}_{\mathrm{A}}(e_t)$
ノード C : $(\rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}})\rho_{\mathrm{AB}} \times \mathrm{med}_{\mathrm{A}}(e_t)$
ノード B : $\rho_{\mathrm{BD}}\rho_{\mathrm{AB}} \times \mathrm{med}_{\mathrm{A}}(e_t)$

3.5.4 多ノード構成におけるアプリケーションのコスト

クラウドコンピューティング上に構築するアプリケーションのコストは、アプリケーションを構成するノードのクラウドリソースのコストの合計である。各ノードのクラウドリソースのコストは、各ノードのクラウドリソース利用量で定まる。第 3.5.3 項により、重み付き隣接行列の冪乗計算を用いて、各ノードのクラウドリソース利用量を計算する。クラウドコンピューティングではクラウドサービスプロバイダがリソースの価格を明確に定義しているため、クラウドリソース利用量に対してコストは一意的である。

クラウドコンピューティング上のアプリケーションはネットワークを通じて利用される。たとえば Web アプリケーションや、Web API などが挙げられる。アプリケーションの各ノードのクラウドリソース利用量は、アプリケーションに対する要求イベント、つまり HTTP のリクエストに伴う計算処理などによって増減する。クラウドコンピューティングの特性により、各ノードのクラウドリソースは、要求される計算処理に応じてスケーリングする。

第 3.5.3 項から、要求元である出発点からの要求に応じてアプリケーション内の各ノードのクラウドリソース利用量が得られる。各ノードのクラウドリソース利用量は、出発点に対応する行と各ノードに対応する列が交わる要素、つまり $\hat{R}$ の 1 行目の要素 $\hat{R}_{1,j}$ を用いる。1 行目の要素にクラウドサービスプロバイダが提供する価格設定を適用することで各ノードのコストを計算する。式 (3.27) は $\hat{R}$ の 1 行目の要素から、ある時点 t におけるアプリケーション全体のコストを表している。

$$\mathrm{Cost}(e_t) = \sum_{j=1}^{N} \mathrm{cost_j}(\hat{R}_{(1,\mathrm{j})}) \tag{3.27}$$

たとえば図 3.4 であれば、式 (3.24) により、イベント e_t ごとにノード A のクラウドリソース利用量は $\rho_{\bullet\mathrm{A}}$、ノード B で $\rho_{\mathrm{AB}}\rho_{\bullet\mathrm{A}}$、ノード C で $\rho_{\mathrm{BC}}\rho_{\mathrm{AB}}\rho_{\bullet\mathrm{A}}$ のコンピューティングリースを利用する。つまり $\rho_{\bullet\mathrm{A}} = \mathrm{med_A}(e_t)$ とすれば、イベント e_t ごとにノード A のクラウドリソース利用量は $\mathrm{med_A}(e_t)$、ノード B で $\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$、ノード C で $\rho_{\mathrm{BC}}\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)$ のコンピューティングリースを利用する。

説明のため受注管理システムの Web アプリケーションを例に挙げる。当受注管理システムは HTTP サーバ、Web API サーバ、データベースの 3 ノードで構成される。HTTP サーバは受注管理システムを受け付け、Web API サーバは受注管理システムの操作に伴う API リクエストの受け付け、データベースは Web API サーバからのデータ取得要求を受け付ける。当受注管理システムをグラフで表現すると図 3.4 になる。図 3.4 に対応するコストは式 (3.28) のように計算する。ノード A は HTTP サーバ、ノード B は Web API サーバ、ノード C はデータベースである。

$$\begin{aligned}
\mathrm{Cost}(e_t) =& \mathrm{cost_A}\left(\hat{R}_{(1,2)}\right) \\
&+ \mathrm{cost_B}\left(\hat{R}_{(1,3)}\right) \\
&+ \mathrm{cost_C}\left(\hat{R}_{(1,4)}\right) \\
=& \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\
&+ \mathrm{cost_B}(\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \\
&+ \mathrm{cost_C}(\rho_{\mathrm{BC}}\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t))
\end{aligned} \tag{3.28}$$

DAG の表現とこれまで説明してきた基本式を組み合わせることで、さまざまなアプリケーションの、さまざまなコスト構造を表現可能である。図 3.10 においても式 (3.28) と同様に計算できる。式 (3.29) は図 3.10 からコストを計算したものである。アプリケー

ションの負荷に応じたクラウドリソースの利用が各ノードで行われ、クラウドサービスプロバイダが設定した価格によりコストが算出できる。

$$\begin{aligned}\mathrm{Cost}(e_t) = &\ \mathrm{cost_A}\left(\hat{R}_{1,2}\right) \\ &+ \mathrm{cost_B}\left(\hat{R}_{1,3}\right) \\ &+ \mathrm{cost_C}\left(\hat{R}_{1,4}\right) \\ &+ \mathrm{cost_D}\left(\hat{R}_{1,5}\right) \\ = &\ \mathrm{cost_A}(\mathrm{med_A}(e_t)) \\ &+ \mathrm{cost_B}(\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \\ &+ \mathrm{cost_C}((\rho_{\mathrm{BC}} + \rho_{\mathrm{DC}}\rho_{\mathrm{BD}})\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t)) \\ &+ \mathrm{cost_D}(\rho_{\mathrm{BD}}\rho_{\mathrm{AB}} \times \mathrm{med_A}(e_t))\end{aligned} \tag{3.29}$$

コスト $\mathrm{Cost}(e_t)$ は時間単位のイベント e_t ごとのコストである。一般に AWS や Microsoft Azure などのクラウドサービスでは、月ごとのクラウドリソース利用量にもとづき請求する。クラウドサービス上に構築されたアプリケーションは負荷に応じてクラウドリソース利用量が増減する。そのためクラウドサービス上のアプリケーションの月額コストは、時間単位のイベント量に応じたコストの総和である。式 (3.30) は時間単位のイベント量に応じたコストの総和を表している。

$$\mathrm{Cost_{total}}(\boldsymbol{e}) = \sum_{t=t_0}^{T} \mathrm{Cost}(e_t) \tag{3.30}$$

e_t は時間単位あたりのイベント、$\boldsymbol{e}$ は時間単位あたりのイベントを時系列に並べたものである。イベント量が増えると各ノードの関係からクラウドリソース利用量が増加し、アプリケーションのコストが増加する。たとえば e_t を 1 時間あたりのリクエスト数、期間 T を 24 時間としたとき、$\boldsymbol{e}$ は $\boldsymbol{e} = \{e_1, \ldots, e_t, \ldots, e_{24}\}$ となる。各 e_t ごとのコストが図 3.11 のように変動しているとする。図 3.11 の場合、期間 24 時間のコストはヒストグラムの面積である。

3.6 ストック型のコンピューングリソース

前述したコスト構造をフロー型 (フロー型コスト) としたき、一部のクラウドリソースにはストック型のコスト構造 (ストック型コスト) を採用しているものがある。ストック型のコスト構造を採用しているクラウドリソースにはストレージの使用容量などが挙げ

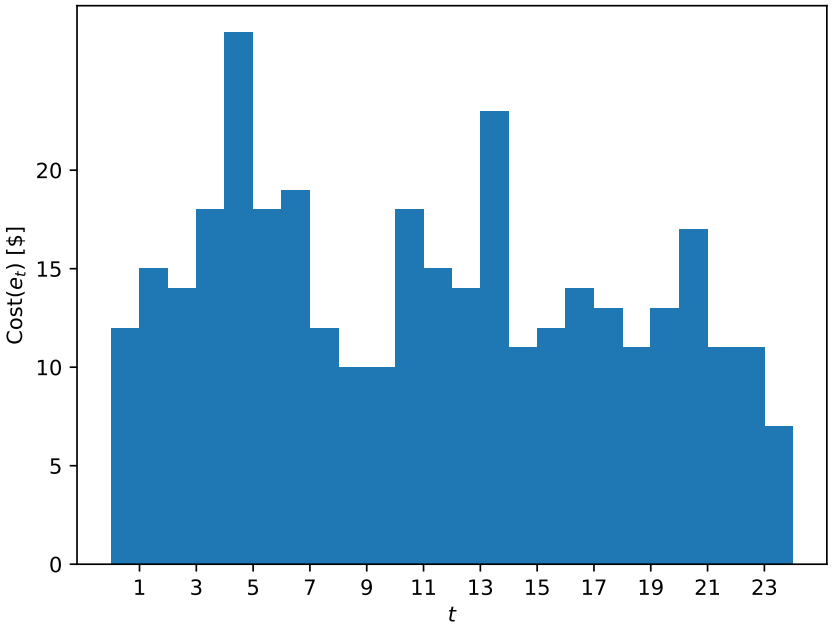

図 3.11　時間あたりのコストの変動とトータルコストの例

られる。ストック型のコスト構造はフロー型とは異なり、ある時点のコスト因子だけでなく、日々蓄積されるコスト因子によって請求額が決定される。たとえば日々積み重なるシステムのログを保存するストレージのコストは、これまでシステムが稼働し蓄積してきたデータサイズ、つまりストレージの使用量に依存するなどがある。そのためストック型のリソースのコストを算出するためには、リソースのコスト因子の使用量を積算する必要がある。

ストック型のクラウドリソース使用量の推定では、これまでと同様に任意の 2 ノード間の関係を用いる。フロー型コストとストック型コストの違いは、ノードに適用するコスト関数の違いである。ストック型コストでは、コスト関数内のイベント量に基づくクラウドリソース利用量の計算の際に積和演算を追加で行う。ストック型コストが適用されるノードのコスト関数を式 (3.31) で表す。

$$\mathrm{Cost}_{\mathrm{stock}}(e_t) = \sum_{j=1}^{N} \mathrm{cost}_{\mathrm{stock}_j} \left(\sum_{\tau=t_0}^{t} \hat{R}_{\tau,(1,j)} \right) \tag{3.31}$$

式 (3.31) の右辺が利用期間中の開始時点 t_0 からの積和になっている。例えば時点 τ におけるストック型コストのクラウドリソース $\hat{R}_{\tau,(1,j)}$ をイベント量を固定とし、毎時点

100MB のデータがストレージに出力されるとする。このとき時間の経過とともに 100MB ずつストレージ利用量が増加するため、たとえば $t_0 + 1$ の場合は $\mathrm{cost}_{\mathrm{stock}_j}(e_{t_0+1}) =$ 100MB 分のコスト が、$t_0 + 10$ の場合は $\mathrm{cost}_{\mathrm{stock}_j}(e_{t_0+10}) = 1$GB 分のコスト がストレージに発生する。

3.7 コスト推定手法のまとめとリリース後の対応

当コスト提案手法はビジネスの立ち上げ時期など、クラウドコンピューティングの特性を踏まえた上で、システムの稼働履歴がない状況に短時間で妥当なコストを推定するための手法である。本章では当提案手法について基本的な考え方、クラウドリソース利用量の推定手続き、そして対象期間におけるアプリケーション全体のコストを算出するまでの手続きを説明した。コスト推定対象のアプリケーションのモデル化ではアプリケーションのシステムを DAG でモデル化して隣接行列で表現することを説明した。コンポーネントのコスト因子となるクラウドリソース利用量の推定では、前述の隣接行列を積和計算することにより、イベント量を引数に各コンポーネントのリソース利用量を求める関数を得られること説明した。そしてクラウドリソース利用量から、アプリケーションの各コンポーネントのコストを求め、合計することでアプリケーション全体のコストを算出することを説明した。

本章のコスト推定手法を用いることで、システムの稼働履歴がない状況に短時間で妥当なコストを推定することができる。さらに本章ではシステム構築前のコスト推定手法に加えて、システムのリリース後に本コスト推定手法で用いたパラメータを更新することを推奨する。なぜならばシステムの稼働履歴がない状況では、その時点で妥当な推定は得られるものの、サービスリリース後にはクラウドサービス利用者は実際に稼働するシステムスから、精緻なシステムの振る舞いを直接確認することができるからである。クラウドサービスの利用者は、リリース後の運用から得られる情報から、2 ノード間の関係のパラメータなどを更新できる。多くのクラウドサービスではクラウドサービス上に構築しているリソースをモニタリングするための監視ツールを提供している。クラウドサービスの利用者は、これらの監視ツールを活用し、パラメータを更新することで、より正確にワークロードに応じたコストを推定できる。

コスト関数の見直しを行うタイミングとして、クラウドサービスプロバイダによるクラ

ウドサービスの価格体系の見直しがある。クラウドサービスはクラウドサービスプロバイダが、自らのビジネス戦略に基づいて定義している。そのためクラウドサービスプロバイダのビジネス状況あるいはクラウドコンピューティングのマーケットの変化に応じて、クラウドサービスの価格体系を変更することがある。価格体系の変更で想定すべきパターンは 2 つあり、1 つは単価の変更である。単価の変更が行われた場合、クラウドサービスの利用者はコスト関数の単価を更新することがで、アプリケーション全体のコスト関数を更新できる。2 つ目はコスト構造の変更である。可能性は低いがクラウドサービスプロバイダが、クラウドサービスですでに提供しているリソースの定義やコスト因子の変更、コスト計算のプロセスを変更するケースである。この場合、クラウドサービスの利用者はコスト関数の定義を見直す必要がある。

第 4 章

ケーススタディによる検証

本章ではクラウドコンピューティングを用いたアプリケーションへの当提案手法の有効性について、実際のクラウドコンピューティング環境を用いてケーススタディを行う。クラウドコンピューティング環境には Google Cloud Platform (GCP) を用いる。当ケーススタディに用いるアプリケーションは、ロードバランサと Web API、データベースで構成されたシンプルなアプリケーションである。各クラウドリソースの構築には主に仮想マシンインスタンスを用い、ロードバランサ、 Web API、データベースにはすべて GCP のクラウドリソースのサービスを用いる。

4.1 アプリケーションの構成とワークロードのシナリオ

当ケーススタディでは、始めにアプリケーションへの負荷シナリオをベースに提案手法によりコストを推定する。その後実際に実装したアプリケーションにワークロードの負荷をかける。負荷の増減に応じて各リソースをスケーリングさせた場合のコストを事前に推定したコストと比較する。当ケーススタディではクラウドリソースの追従性を検証することで、当提案手法がシステム構築前のコスト見積もりにおいて一定の有用性を持つことを示す。

4.1.1 仮想マシンをベースとしたシステム構成

図 4.1 にケーススタディで使用するアプリケーションのシステム構成を示す。クラウドコンピューティングのプラットフォームは GCP を利用し、IaaS ベースでアプリケーションを構築する。当アプリケーションは読み出し中心の Web アプリケーションである。

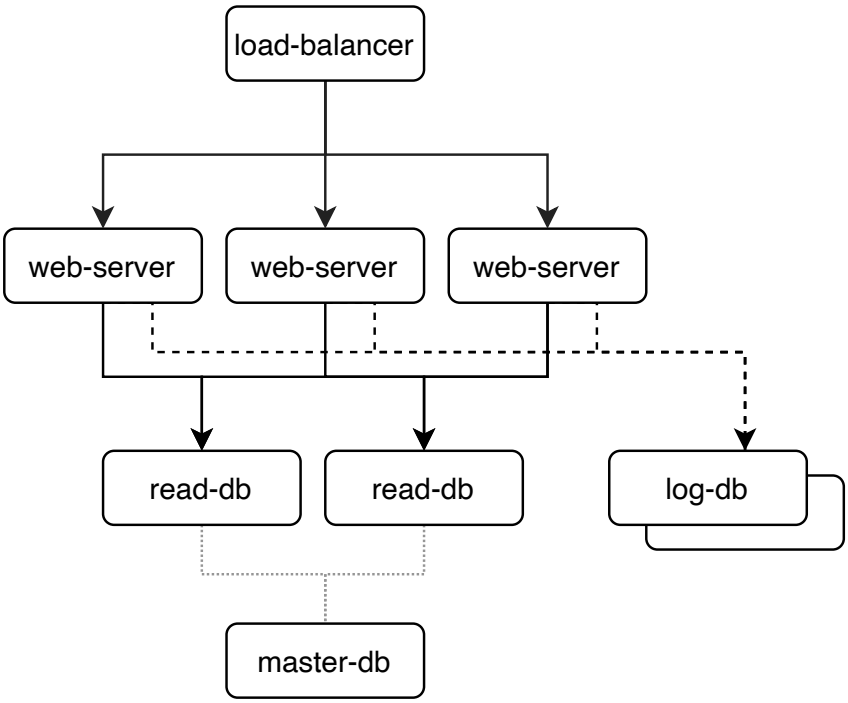

図 4.1　仮想マシンをベースとしたシステム構成

図 4.1 の `load-balancer` はロードバランサを表す。図 4.1 の `web-server` は Web API を提供する Web サーバを表す。図 4.1 の `read-db` と `master-db` はデータベースを表す。`read-db` と `master-db` の 2 つの役割がそろってデータベースの機能を有する。図 4.1 の `log-db` はログサーバである。`log-db` はアプリケーションの機能には貢献しない。

当アプリケーションではデータベース層の複数の `read-db` に格納されたデータを、コンピューティング層の複数の Web サーバを介して取得する。当アプリケーションにおけるロードバランサ層は以下を想定する。

- ロードバランサは外部からのリクエストをコンピューティング層の複数の `web-server` へ振り分ける。
- ロードバランサは利用量に応じて課金される。

当アプリケーションにおけるコンピューティング層は以下を想定する。

- 外部からのリクエストによる負荷増加に応じて仮想サーバインスタンスを水平スケーリングする。
- Web サーバは外部からリクエストを受け取るとデータベースにの読み取りリクエ

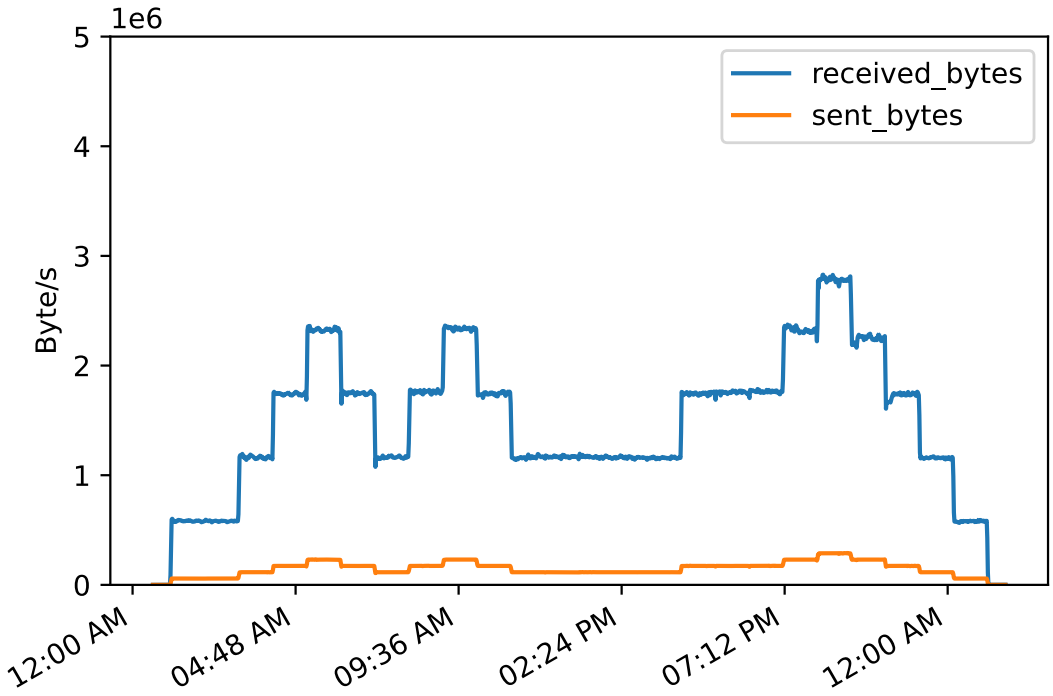

図 4.2　ケーススタディの負荷シナリオ

ストを投げ、その結果を返却する。

- Web サーバのアクセスログは `log-db` に保存する。

当アプリケーションにおけるデータベース層は以下を想定する。

- `read-db` に保存されているデータは、`master-db` に保存されているデータの複製である。
- データの更新は `master-db` を通じて行われる。実験シナリオではデータの更新は行わず静的なデータとする。

4.1.2　ケーススタディの負荷シナリオ

当ケーススタディではリクエスト量を 3 時間の間に変動させる。図 4.2 はケーススタディで想定するワークロードを示したものである。負荷をかける時間を 24 時間とし、1 時間ごとにワークロードを変化させる。

ノード	クラウドリソースの利用量
$\mathrm{med_A}$	Network egress traffic: 10 GB/hour/e_t
$\mathrm{med_B}$	Load balancer ingress traffic: 1 GB/e_t
$\mathrm{med_C}$	VM CPU usage (n1-standard-1(1vCPU)) : 6.2VMs/e_t
$\mathrm{med_D}$	VM CPU usage (n1-standard-2(2vCPU)) : 3.1VMs/e_t
$\mathrm{med_E}$	VM CPU usage (n1-standard-1(1vCPU)) : 0.062 VMs/e_t

表 4.1　各ノードにおけるクラウドリソースの利用量

図 4.2 は一般的な Web サービスの負荷シナリオを想定している。青色の線 (`received_bytes`) がクライアントから見た受信のトラフィックであり、橙色の線 (`sent_bytes`) がクライアントから見た送信のトラフィックである。午前 4:00 にワークロードの負荷を開始し、通勤時間帯である午前 8:00 に 1 度目の負荷のピークがある。昼休みである午後 12:00 に 2 度目のピークがあり、就寝前の午後 11:00 に 3 度目のピークがある。3 度目のピークが過ぎた後は、徐々に負荷が減少する。読み取り専用のアプリケーションを想定しいるため、クライアントから見て受信のトラフィックが支配的である。

4.2　2 ノード間の関係の推定

4.2.1　隣接する 2 ノード間の関係の表現

図 4.3 は、図 4.1 における隣接する 2 ノード間の関係を表したものである。図 4.3 の各ノードは、図 4.1 のネットワーク境界、コンピューティング層、データベース層の `read-db`、データベース層の `master-db`、そしてログサーバに対応する。隣接する 2 ノード間の関係は、ノード間のコスト因子となるクラウドリソースの利用量の比に対応する。たとえばノード C とノード E の関係 ρ_{CE} であれば、$\rho_{\mathrm{CE}} = \mathrm{med_E} \,/\, \mathrm{med_C}$ である。

各クラウドリソースについて詳細を表 4.1 に表す。各ノードのクラウドリソース利用量は、それぞれイベント量の 1 単位、つまりクライアントが 10GB/時 のトラフィックを受け取る場合におけるクラウドリソース利用量である。データベース層の `master-db` は常に 1 で固定であるため、ノード M は表 4.1 に含めていない。

図 4.3 において、隣接する 2 ノード間の関係は行列を用いて表現できる。具体的には行列 R の各行と各列に、それぞれノード A, B, C, D, E と出発点を対応させ、それぞれノードが対応する行と列を組み合わせた要素に隣接する 2 ノード間の関係を表現させること

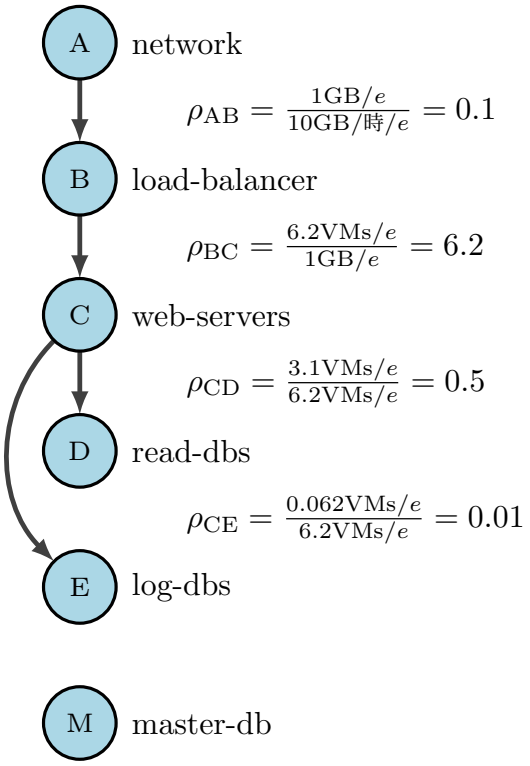

図 4.3　隣接する 2 ノード間の関係

ができる。たとえばノード A からノード B への関係であれば、2 行目と 3 列目が重なる要素 $(2,3)$ にノード A からノード B への関係が入る。

$$
\begin{aligned}
R &= \begin{bmatrix}
0 & \rho_{\bullet\mathrm{A}} & 0 & 0 & 0 & 0 \\
0 & 0 & \rho_{\mathrm{AB}} & 0 & 0 & 0 \\
0 & 0 & 0 & \rho_{\mathrm{BC}} & 0 & 0 \\
0 & 0 & 0 & 0 & \rho_{\mathrm{CD}} & \rho_{\mathrm{CE}} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \\
&= \begin{bmatrix}
0 & 10 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.1 & 0 & 0 & 0 \\
0 & 0 & 0 & 6.2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0.01 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{aligned}
\tag{4.1}
$$

4.2.2 任意の 2 ノード間の関係の推定

隣接する 2 ノード間の関係を式で表したことにより、任意の 2 つのノード間の関係 $\hat{R}$ は式 (4.2) によって与えられる。

$$\begin{aligned}\hat{R} &= \sum_{k=1}^{4} R^k = \sum_{k=1}^{4} \begin{bmatrix} 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.01 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^k \\ &= \begin{bmatrix} 0 & 10.0 & 1.0 & 6.2 & 3.1 & 0.062 \\ 0 & 0 & 0.1 & 0.62 & 0.31 & 0.0062 \\ 0 & 0 & 0 & 6.2 & 3.1 & 0.062 \\ 0 & 0 & 0 & 0 & 0.5 & 0.01 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\end{aligned} \tag{4.2}$$

4.3 フローベースのコスト推定

クラウドリソースの利用量が決まればコスト計算が可能になる。式 (4.2) から、フローベースのコストを式 (4.3) のように組み立てられる。

$$\begin{aligned}&\mathrm{Cost}_{\mathrm{flow}_{\mathrm{webaccess}}}(e_t) \\ &\quad = \mathrm{cost}_{\mathrm{flow}_{\mathrm{A}}}(\mathrm{med}_{\mathrm{A}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{B}}}(\mathrm{med}_{\mathrm{B}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{C}}}(\mathrm{med}_{\mathrm{C}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{D}}}(\mathrm{med}_{\mathrm{D}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{E}}}(\mathrm{med}_{\mathrm{E}}(e_t)) \\ &\quad = \mathrm{cost}_{\mathrm{flow}_{\mathrm{A}}}(\mathrm{med}_{\mathrm{A}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{B}}}(0.1 \times \mathrm{med}_{\mathrm{A}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{C}}}(0.62 \times \mathrm{med}_{\mathrm{A}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{D}}}(0.31 \times \mathrm{med}_{\mathrm{A}}(e_t)) \\ &\qquad + \mathrm{cost}_{\mathrm{flow}_{\mathrm{E}}}(0.0062 \times \mathrm{med}_{\mathrm{A}}(e_t))\end{aligned} \tag{4.3}$$

また負荷によるスケーリングが行われないノード M は固定のクラウドリソースである。式 (4.4) は `master-db` のフローベースのコストを表す。コスト関数 $\mathrm{cost}_{\mathrm{flow}_{\mathrm{med}}}(\mathrm{med}_{\mathrm{M}})$ は常に 1 台の仮想サーバインスタンス分のクラウドリソース利用量を返すため、関数 $\mathrm{cost}_{\mathrm{flow}_{\mathrm{med}}}(\mathrm{med}_{\mathrm{M}})$ は定数関数である。さらに当アプリケーションでは固定のクラウドリ

ソースを消費するノードはノード M のみになるため、関数 $\mathrm{Cost}_{\mathrm{flow}_{\mathrm{master}}}(e_t)$ も定数関数である。

$$\mathrm{Cost}_{\mathrm{flow}_{\mathrm{master}}}(e_t) = \mathrm{cost}_{\mathrm{flow}_{\mathrm{M}}}(\mathrm{med}_{\mathrm{M}}(e_t)) \tag{4.4}$$

アプリケーション全体のフローベースのコストはすべてのフローベースのコストの総和である。式 (4.5) は当アプリケーションにおけるフローベースのコストの総和を表す。Web サーバへのリクエストに伴うフローベースのコストを表す関数 $\mathrm{Cost}_{\mathrm{flow}_{\mathrm{webaccess}}}$ と固定のクラウドリソースのコストを表す関数 $\mathrm{Cost}_{\mathrm{flow}_{\mathrm{master}}}$ の合計になっている。

$$\mathrm{Cost}_{\mathrm{flow}}(e_t) = \mathrm{Cost}_{\mathrm{flow}_{\mathrm{webaccess}}}(e_t) + \mathrm{Cost}_{\mathrm{flow}_{\mathrm{master}}}(e_t) \tag{4.5}$$

4.4 ストックベースのコスト推定

ストックベースのコスト計算においても、フローベースの計算と同様に式 (4.2) から、ストックベースのコストを式 (4.6) のように組み立てられる。

$$\mathrm{Cost}_{\mathrm{stock}}(e_t) = \mathrm{cost}_{\mathrm{stock}_{\mathrm{E}}} \left(\sum_{\tau=t_0}^{t} (0.0062 \times \mathrm{med}_{\mathrm{A}}(e_\tau)) \right) \tag{4.6}$$

図 4.1 のログサーバ、つまり `log-db` のみストックベースのコストがかかると仮定しているため、式 (4.6) ではノード E のコストのみを計算している。たとえばノード E 上のログデータが 1 イベント/時あたり約 10MB 増加するとする。このとき $\mathrm{med}_{\mathrm{A}}(e_t)$ に対応する時間あたりのストレージ利用量の増加に伴うコストは式 (4.7) で表す。

$$\begin{aligned} \mathrm{cost}_{\mathrm{E_{inc}}}/\mathrm{med}_{\mathrm{A}} &= \frac{\text{ストレージの月額料金}}{\text{月あたりの時間}} \times \frac{\text{イベントごとの追加データ}}{\mathrm{med}_{\mathrm{E}}/\mathrm{med}_{\mathrm{A}}} \\ &= \frac{\$0.040/\mathrm{GB}}{24\,\text{時間} \times 31\mathrm{days}} \times \frac{0.01\mathrm{GB}}{0.0062} \\ &= \$0.000087 \end{aligned} \tag{4.7}$$

4.5 トータルコストの推定

クラウドリソース利用量の関係が定まるとアプリケーション全体のコスト推定が可能になる。当ケーススタディのアプリケーションのコストにはフローベースのコストとストックベースのコストが存在する。式 (4.8) はフローベースのコストとストックベースの和で

コスト因子	価格
Egress network traffic	$0.12/GB
Load Balancer transfer rules	$0.038/h
Load Balancer ingress traffic	$0.012/GB
Compute engine (n1-standard-1(1vCPU))	$0.043/h (effective)
Compute engine (n1-standard-2(2vCPU))	$0.085/h (effective)
Standard persistent disk (pd-standard)	$0.040/GB/month

From `https://cloud.google.com/products/calculator/`

表 4.2　GCP におけるクラウドリソースの価格表

ある。式 (4.5) のフローベースのコスト $\mathrm{Cost}_{\mathrm{flow}}(e_t)$ と、式 (4.7) のストックベースのコスト $\mathrm{Cost}_{\mathrm{stock}}(e_t)$ がそれぞれ展開されている。

$$
\begin{aligned}
\mathrm{Cost}(e_t) &= \mathrm{Cost}_{\mathrm{flow}}(e_t) + \mathrm{Cost}_{\mathrm{stock}}(e_t) \\
&= \mathrm{cost}_{\mathrm{flow_A}}(\mathrm{med_A}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{flow_B}}(0.1 \times \mathrm{med_A}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{flow_C}}(0.62 \times \mathrm{med_A}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{flow_D}}(0.31 \times \mathrm{med_A}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{flow_E}}(0.0062 \times \mathrm{med_A}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{flow_M}}(\mathrm{med_M}(e_t)) \\
&\quad + \mathrm{cost}_{\mathrm{stock_E}}\left(\sum_{\tau=t_0}^{t}(0.0062 \times \mathrm{med_A}(e_\tau))\right)
\end{aligned}
\tag{4.8}
$$

式 (4.8) により、イベントあたりにどれだけのコストがアプリケーション全体でかかるかを関数 $\mathrm{Cost}(e_t)$ で得ることができる。関数 $\mathrm{Cost}(e_t)$ が得られることで時点 t におけるシステム負荷に応じたアプリケーション全体のコストを得ることができる。簡単のため当アプリケーションに 1 ヵ月間常時 10 GB/時 の受信トラフィックが発生した場合のコストを推定する。表 4.2 は当アプリケーションで用いる GCP のクラウドリソースの価格である。関数 $\mathrm{Cost}(e_t)$ に表 4.2 を適用することで以下のようにコストを見積もることができる。

式 (4.9) は 1 ヵ月間に常時 10GB/時の受信トラフィックが発生した際の、フローベースのコストである。1 ヵ月のイベント数を 744 時間と仮定し、表 4.2 のクラウドリソース

コスト因子	月額費用
フローベースのコスト	
Egress network traffic for $\mathrm{med_A}$	\$892.8/月
Load balancer transfer rules and ingress traffic for $\mathrm{med_B}$	\$37.2/月
n1-standard-1(1vCPU) for $\mathrm{med_C}$	\$198.35/月
n1-standard-2(2vCPU) for $\mathrm{med_D}$	\$196.044/月
n1-standard-1(1vCPU) for $\mathrm{med_E}$	\$1.984/月
Rough estimation for $\mathrm{med_M}$	\$31.992/月
ストックベースのコスト	
pd-standard for $\mathrm{med_E}$	\$1.495/月

表 4.3　アプリケーションの月額費用内訳

の費用を代入することでフローベースのコストを求めている。

$$
\begin{aligned}
&\mathrm{Cost_{flow}}(24\text{時間} \times 31\text{日}) \qquad \because \mathrm{med_A}(e_t) = 10\mathrm{GB/時} \\
&\quad = 10 \times \$0.12 \times 744 \\
&\qquad\quad + (\$0.038 + (10 \times 0.1 \times \$0.012)) \times 744 \\
&\qquad\quad + 10 \times \$0.043 \times 0.62 \times 744 \\
&\qquad\quad + 10 \times \$0.085 \times 0.31 \times 744 \\
&\qquad\quad + 10 \times \$0.043 \times 0.0062 \times 744 \\
&\qquad\quad + \$0.043 \times 744 \\
&\quad \simeq \$1358.37/\text{月}
\end{aligned}
\tag{4.9}
$$

式 (4.10) は 1 ヵ月間に常時 10GB/時の受信トラフィックが発生した際の、ストックベースのコストである。1 ヵ月のイベント数を 744 時間と仮定し、表 4.2 のストレージの料金を代入することでストックベースのコストを求めている。

$$
\begin{aligned}
&\mathrm{Cost_{stock}}(24\text{時間} \times 31\text{日}) \\
&\quad = \sum_{t=1}^{744} \sum_{\tau=1}^{t} (10 \times \$0.000087 \times 0.0062) \\
&\quad \simeq \$1.495/\text{月}
\end{aligned}
\tag{4.10}
$$

式 (4.11) は 1 ヵ月間に常時 10GB/時の受信トラフィックが発生した際のアプリケーション全体のコストである。1 ヵ月のイベント数を 744 時間と仮定し、表 4.2 のストレージの料金を代入することでストックベースのコストを求めている。アプリケーションの月

額費用内訳は表 4.3 となる。

$$
\begin{aligned}
&\mathrm{Cost}_{\mathrm{total}}(24\text{ 時間} \times 31\text{ 日}) \\
&= \mathrm{Cost}_{\mathrm{flow}}(24\text{ 時間} \times 31\text{ 日}) + \mathrm{Cost}_{\mathrm{stock}}(24\text{ 時間} \times 31\text{ 日}) \\
&\simeq \$1358.37/\text{月} + \$1.495/\text{月} \\
&= \$1359.865/\text{月}
\end{aligned}
\tag{4.11}
$$

イベントに応じたアプリケーション全体のコストの関数を得ることで、イベントの変動、つまりシステムの負荷に応じてどのようにアプリケーション全体のコストが変動するかあらかじめ把握できるようになる。前述で得られたコスト関数をケーススタディのシステム負荷に適用すると式 (4.12) のようにアプリケーション全体のコストが得られる。

$$
\begin{aligned}
\mathrm{Cost}_{\mathrm{total}}(\boldsymbol{e}) &= \sum_{t=t_0}^{T} \mathrm{Cost}(e_t) \\
&= \sum_{t=t_0}^{T} \mathrm{Cost}_{\mathrm{flow}}(e_t) + \sum_{t=t_0}^{T} \mathrm{Cost}_{\mathrm{stock}}(e_t) \\
&\simeq \sum_{t=t_0}^{T} \$25.3294 + \$0.0009 \\
&\simeq \$25.33/\text{実験期間}
\end{aligned}
\tag{4.12}
$$

本節で説明した事例は単純なものであり、事業計画の段階ではより複雑なネットワークトポロジー、隣接する 2 ノード間の関係の設定が必要になるだろう。しかし以上説明してきた手順には、行列表現と形式的な計算手順により、新規システムのコストの概算を手早く算出できる利点がある。

4.6 仮想マシンの水平スケーリングの追従性

本章で説明したコスト推定手法は、クラウドコンピューティングの特性であるスケーラビリティにより、システム負荷に対して推測可能なクラウドリソースの割り当てが行われることに依存する。提案するフレームワークが実際のクラウドサービスでどのように動作するかを示すために。実際のクラウドサービス上にサンプルアプリケーションを実装した。サンプルアプリケーションは第 4.1 節で説明したとおりである。

当アプリケーションでは、コンピューティング層、データベース層、ログサーバは水平スケーリングする仮想マシンインスタンスを想定している。そのため実際のクラウドサービスである GCP 上で自動スケーリングするよう仮想マシンインスタンス群の設定を行っ

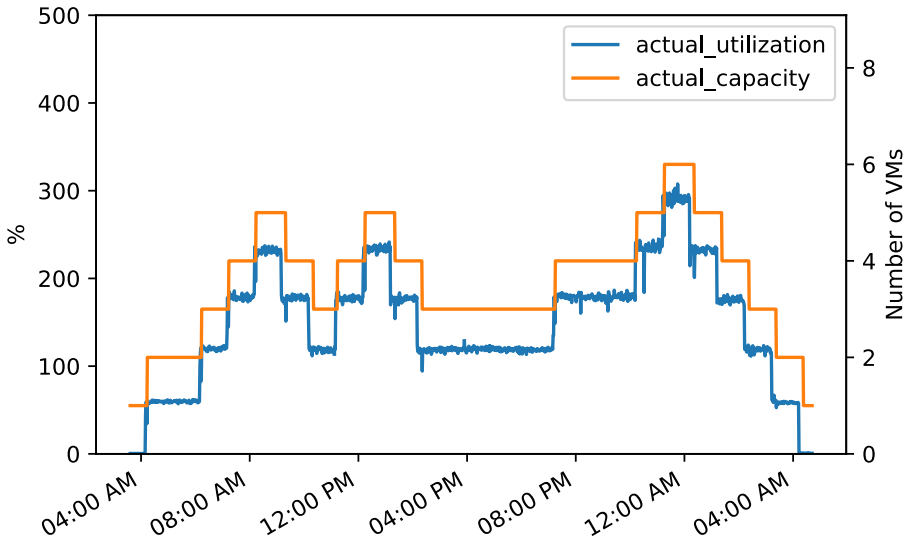

図 4.4　コンピューティング層の CPU 使用率と仮想マシンインスタンスの台数

た。ロードバランサは GCP のデフォルトのロードバランサを用いた。ワークロードに関しては図 4.2 のような負荷シナリオを設定し、各ノードで推定通りにクラウドリソースの割り当てが行われているか確認する。

図 4.4 は図 4.2 の負荷シナリオにおけるコンピューティング層の CPU 使用率の変動と割り当てられた仮想マシンインスタンスの台数の変化を表している。青色の線 (`actual_utilization`) は負荷シナリオによりコンピューティング層で発生した CPU 使用率を表している。橙色の線 (`actual_capacity`) は割り当てられた仮想マシンインスタンス、仮想マシンインスタンスの処理能力を 55% とした際のコンピューティング層全体で割り当て可能な処理能力 (CPU 使用率換算) を表している。なお CPU 使用率が 55% で表示されているのは、当仮想マシンインスタンス群のスケーリングの指標が CPU 使用率 55% だからである。

コンピューティング層で必要とされる CPU 使用率に対して割り当て済みのクラウドリソースが不足している場合、コンピューティング層では仮想マシンインスタンスが追加される。一方で CPU 使用率に対して、割り当て済みのクラウドリソースが過剰な場合は、コンピューティング層の仮想マシンインスタンスが減らされる。つまりコンピューティング層ではコンピューティング層で必要とされる CPU 使用率を満たす最小限のクラウドリソース、つまり仮想マシンインスタンスが割り当てられる。

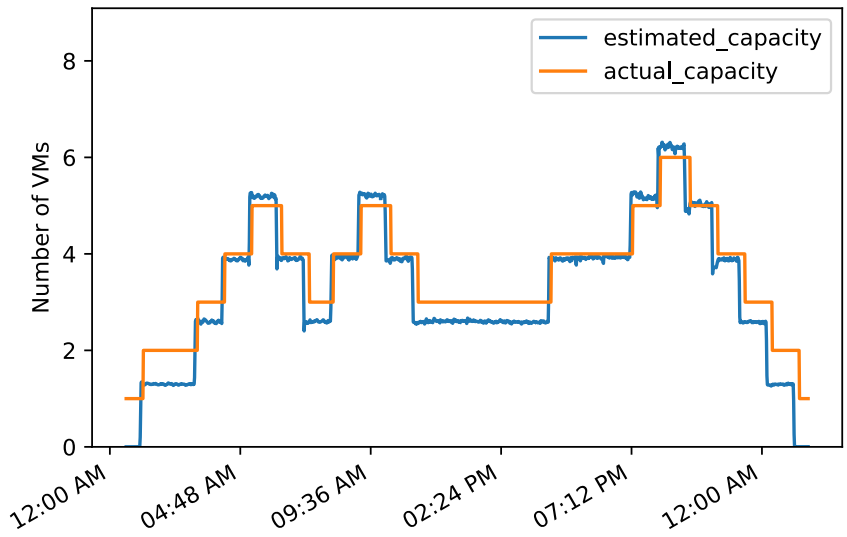

図 4.5　コンピューティング層の仮想マシンインスタンスの台数の予測値と実測値

図 4.4 を見ると、負荷シナリオに応じて仮想マシンインスタンスが増減していることが見て取れる。通勤時間帯である午前 8:00 と昼休みである午後 12:00、就寝前の午後 11:00 のすれぞれのピークに対して、クラウドリソースが適切に割り当てられ、そのほかの谷となる時間帯ではクラウドリソースの解放されている。コンピューティング層で必要とされる CPU 使用率に対して、割り当て済みのクラウドリソースの変化がより大きな階段上になっているのは、クラウドリソースが仮想マシンインスタンス単位、つまりサーバ単位で扱われているためである。

図 4.4 において、CPU 使用率と割り当て済みの仮想マシンインスタンスの台数に差があるのは、仮想マシンのインスタンスサイズの違いによるものである。クラウドコンピューティングにおいて、オーバーバイイングとアンダーバイイングは仮想的なコンピューティングリソースの単位で発生しうるものである。仮想マシンインスタンス 1 台あたりの計算能力の単位でクラウドリソースの割り当てと解放が行われているため、コンピューティング層で要求される計算能力と仮想マシンの割り当てにより確保された計算能力との間に差が生まれている。

図 4.5 は、式 (4.2) で得られた出発点とコンピューティング層との関係 ρ_{AC} から推定した仮想マシンインスタンスの台数 $\rho_{\mathrm{AC}} \times \mathrm{med}_{\mathrm{A}}(e_t)$ と、実際にワークロードにて割り当てられた仮想マシンインスタンスの台数の比較を表している。青色の線 (`estimated_capacity`)

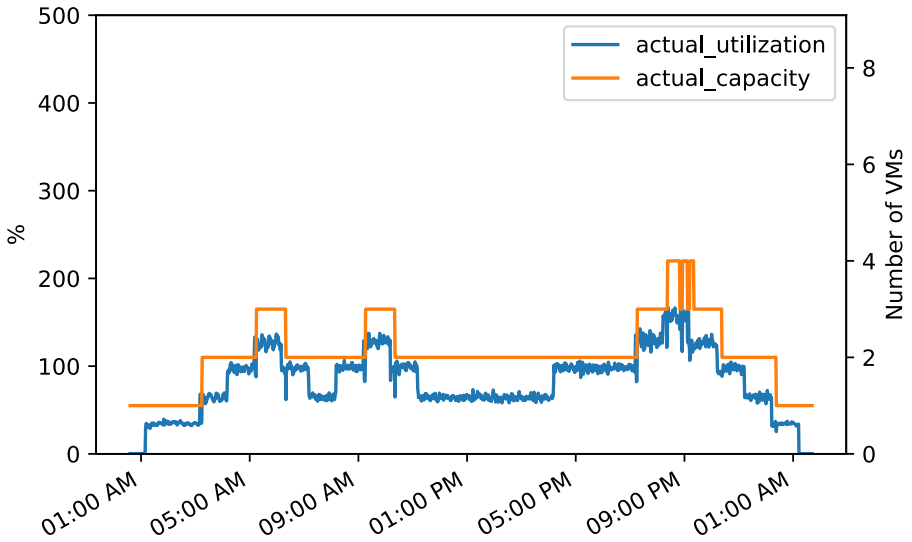

図 4.6　データベース層の CPU 使用率と仮想マシンインスタンスの台数

が推定された仮想マシンインスタンスの台数、橙色の線 (`actual_capacity`) が実際に割り当てられた仮想マシンインスタンスの台数である。提案したコスト推定手法により推定されたコンピューティング層の仮想マシンインスタンスの数と、実際にワークロードにて割り当てられた仮想マシンインスタンスの台数が、ワークロードに対して同様に増減していることが見て取れる。

仮想マシンのサイズによるオーバーバイイングとアンダーバイイングの振れ幅は、ワークロードによるシステム負荷の規模と 仮想マシンインスタンス 1 台が提供する計算能力の比に影響する。ワークロードによるシステム負荷の規模に対して、スケーリングに用いる仮想マシンインスタンス 1 台分の計算能力が相対的に十分小さければ、オーバーバイイングとアンダーバイイングの振れ幅は許容できるほどのものとなる。

データベース層 (`read-db`) で必要とされる CPU 使用率に対して、割り当て済みのクラウドリソースが不足している場合、データベース層では仮想マシンインスタンスが追加される。一方で CPU 使用率に対して割り当て済みのクラウドリソースが過剰な場合は、データベース層 (`read-db`) に割り当てる仮想マシンインスタンスの台数は減らされる。つまりデータベース層 (`read-db`) 層では、必要とされる CPU 使用率を満たす最小限のリソース、つまり仮想マシンインスタンスが割り当てられる。

図 4.6 を見ると、データベース層においても、負荷シナリオに応じた形で仮想マシン

インスタンスを割り当て、解放していることが確認できる。通勤時間帯である午前 8:00 と昼休みである午後 12:00、就寝前の午後 11:00 のすれぞれのピークに対してクラウドリソースが適切に割り当てられ、そのほかの谷となる時間帯ではクラウドリソースが解放されている。データベース層 (`read-db`) では MySQL を使用しており、実験中に使用される主な計算資源は CPU 時間である。データベース層 (`read-db`) で必要とされる CPU 使用率に対して、割り当てられているクラウドリソースの変化がより大きな階段上になっているのはクラウドリソースの増減が仮想マシンインスタンス単位、つまりサーバ単位で扱われているためである。

データベース層 (`read-db`) におけるクラウドリソースの割り当ての増減が、コンピューティング層と比べて大きな階段上になっているのは、イベントあたりに消費するクラウドリソースの違いによる。図 4.3 により、コンピューティング層からデータベース層 (`read-db`) への関係は 0.5 であるため、データベース層 (`read-db`) の仮想マシンインスタンスの台数の増減は 2 倍ほどの粒度で行われる。これにより、`read-db` の仮想マシンインスタンスの台数は Web サーバの約半分になっている。

また就寝前の午後 11:00 ごろに割り当てられているクラウドリソースの増減が頻繁に行われているのは、データベース層で必要とされる CPU 使用率が、仮想マシンインスタンスのスケーリング指標の閾値付近で上下しているためである。クラウドリソースのスケーリングが仮想マシンインスタンス単位で行われているため、仮想マシンインスタンスの計算処理能力単位でクラウドリソースの割り当てが上下している。

図 4.7 は式 (4.2) で得られた出発点とデータベース層との関係 ρ_{AD} から推定した仮想マシンインスタンスの台数 $\rho_{\mathrm{AD}} \times \mathrm{med}_{\mathrm{A}}(e_t)$ と、実際にワークロードにて割り当てられた仮想マシンインスタンスの台数の比較である。青色の線 (`estimated_capacity`) が推定された仮想マシンインスタンスの台数、橙色の線 (`actual_capacity`) が実際に割り当てられた仮想マシンインスタンスの台数である。提案したコスト推定手法により推定されたデータベース層の仮想マシンインスタンスの台数と、実際にワークロードにてデータベース層で割り当てられた仮想マシンインスタンスの台数が、ワークロードに対して同様に増減していることが見て取れる。

仮想マシンインスタンスのサイズによるオーバーバイイングとアンダーバイイングの振れ幅は、ワークロードによるシステム負荷の規模と仮想マシンインスタンス 1 台が提供する計算能力の比に影響するため、コンピューティング層に比べてスケーリングに用いる仮

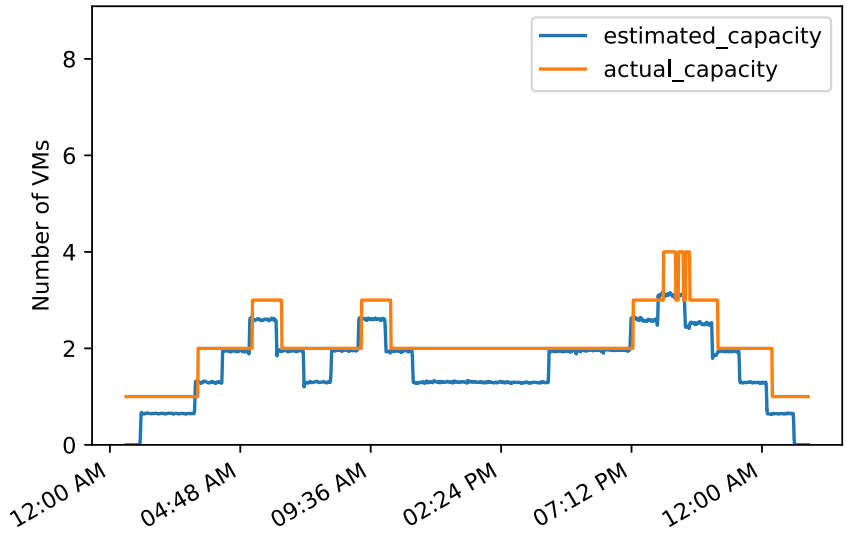

図 4.7　データベース層の仮想マシンインスタンスの台数の予測値と実測値

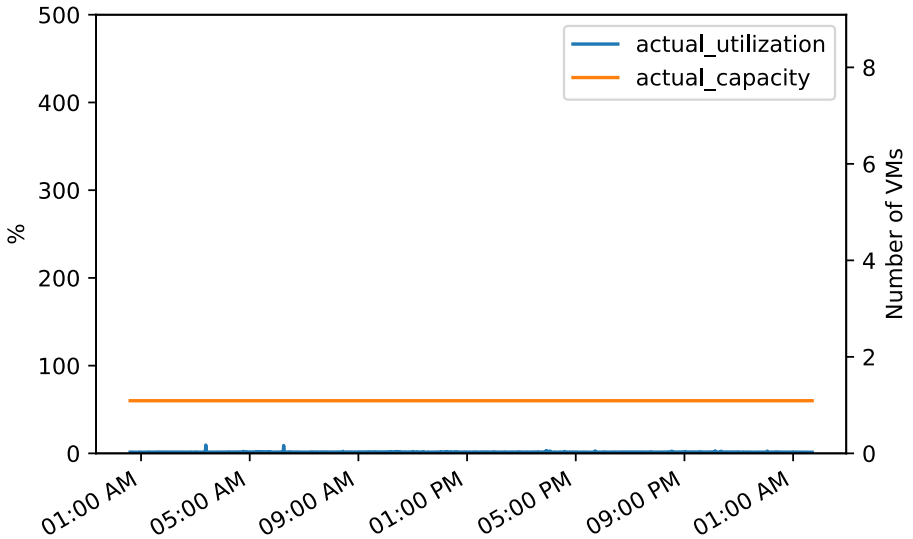

図 4.8　ログサーバの CPU 使用率と仮想マシンインスタンスの台数

想マシンインスタンス 1 台分の計算能力が相対的に大きいデータベース層では、オーバーバイイングの振れ幅が大きくなっている。

ログサーバにおける負荷シナリオに応じた仮想マシンインスタンスの割り当ては図 4.8 になる。青色の線 (`actual_utilizatoin`) がログサーバに必要とされた計算能力 (CPU

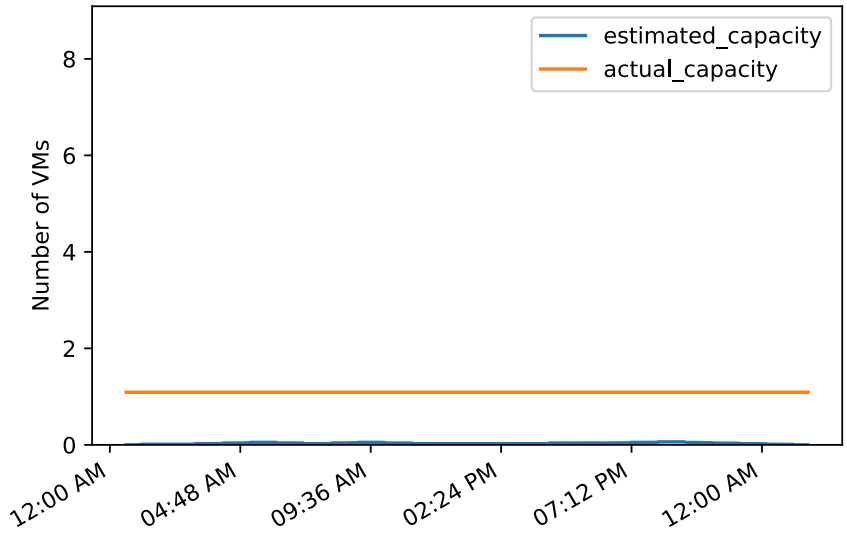

図 4.9　ログサーバの仮想マシンインスタンスの台数の予測値と実測値

使用率)、橙色の線 (`actual_capacity`) が実際に割り当てられた仮想マシンインスタンスの台数である。図 4.8 を見ると、負荷シナリオに応じた形で仮想マシンインスタンスを割り当ておよび解放が確認できない。これはコンピューティング層で必要とされる CPU 使用率の変動に対して、スケーリングに用いられる仮想マシンインスタンスの計算処理能力の単位が大きいためである。コンピューティング層で必要とされる CPU 使用率が、仮想マシンインスタンス数のスケーリングを行う閾値に達しないため、仮想マシンインスタンスの台数が 1 台のままになっている。

図 4.9 は式 (4.2) で得られた出発点とログサーバとの関係 ρ_{AD} から推定した仮想マシンインスタンスの台数 $\rho_{\mathrm{AE}} \times \mathrm{med}_{\mathrm{A}}(e_t)$ と、実際にワークロードにて割り当てられた仮想マシンインスタンスの台数の比較である。青色の線 (`estimated_capacity`) が推定された仮想マシンインスタンスの台数、橙色の線 (`actual_capacity`) が実際に割り当てられた仮想マシンインスタンスの台数である。仮想マシンインスタンスのサイズによるオーバーバイイングとアンダーバイイングの振れ幅は、ワークロードによるシステム負荷の規模と 1 台の仮想マシンインスタンスが提供する計算能力の比に影響するため、仮想マシンインスタンスの計算能力がシステム負荷の規模と比べて大きいログサーバでは推定したクラウドリソース利用量との差が大きくなっている。

ログサーバでは仮想マシンインスタンスのコストのほかに、保存するデータの増加に

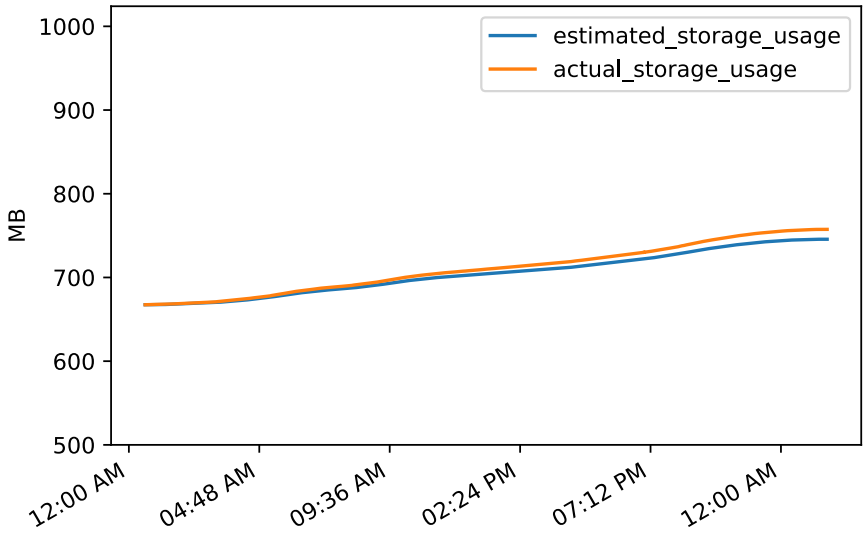

図 4.10　ログサーバのストレージ利用量の予測値と実測値

伴うストレージのコストがある。図 4.10 は式 (4.7) から計算したストレージ利用量の変化と、実際にワークロードの中で計測されたストレージ利用量の変化である。青色の線 (`estimated_capacity`) が推定したストレージ利用量、橙色の線 (`actual_capacity`) が、実際にワークロードの中で計測されたストレージ利用量である。図 4.10 を確認すると、提案したコスト推定手法により推定したストレージ利用量と、実際にワークロードの中で計測されたストレージ利用量が、システム負荷の変化の中で同様に増加していることが見て取れる。

4.7　ケーススタディのまとめ

本章では提案するコスト推定手法を、システムに対してどのように適用していくかをケーススタディに沿って説明し、実際にクラウドサービス上にシステムを構築することでワークロードの増加に応じたコスト推定ができることを示した。前述の図に示すように，クラウドサービスのスケーリング機能によって，ワークロードの変化に対応するためのコスト，すなわち割り当てられるクラウドリソースが変動していることが確認できた。これはクラウドサービスのスケーリングを適切に設定することで、ワークロードの変化に応じたコストを推定できることを示している。

第 4.5 節では隣接する 2 ノードから任意の 2 ノード間の関係を推定し、クラウドリソース利用量に伴うノードのコストを試算した。そしてイベント量に対応するアプリケーション全体のコストを求める関数を求め、コスト推定の対象期間のイベント量に適用することで、定めた期間内のアプリケーション全体のコストを得ることができた。これによりビジネス状況の変化に伴いアプリケーションのシステム負荷が変動したとしても、アプリケーション全体のコストを随時推定できる。

第 4.6 節では GCP 上に仮想マシンを用いたサーバベースアプリケーションを構築してワークロードの負荷シナリオに沿って実際にトラフィックを流し、各ノードで推測可能なリソースの割り当てが行われているか確認した。その結果、実用的な範囲で妥当なリソース割り当て量を推測できていることを確認された。式 (4.2) で得られる各ノードのクラウドリソース利用量、たとえばコンピューティング層であればノード B にあたる仮想マシンインスタンスの台数が、データベース層であればノード C にあたる仮想マシンインスタンスの台数が、式 (4.2) に近似する形でスケーリングしている。

一方で仮想マシンインスタンスというスケーリングの単位が大きいクラウドリソースを用いていることで、イベント量の変動に伴うクラウドリソース利用量に対して、割り当てされるクラウドリソースが仮想マシンインスタンスの単位になることを実験から確認した。クラウドコンピューティングにおいて割り当てされたクラウドリソースのコストモデルは、クラウドサービスプロバイダによりオープンな形で定義付けられるため一意である。そのためクラウドリソースの要求に対して、どのサイズの仮想マシンインスタンスを利用するかは、アプリケーションを構築する際には考慮すべきポイントである。

当コスト手法により、イベント量の変化に応じたアプリケーション全体のコストの変化をシステム構築前に把握できる。しかし概算は概算であり、システム構築前の小さな誤差が、アプリケーション全体ではクラウドコンピューティングのスケーリング機能により拡大される可能性がある点に注意が必要である。このような誤差を最小化することは難しいが、推奨するアプローチとしては、提案手法のパラメータのを継続的に繰り返すことである。初期の段階では専門家によるパラメータの推定 (たとえば ρ) を行い、事業を成長させながらパラメータの実測値を収集し、クラウドリソースが実際にどのように割り当てられているか把握しながらパラメータを更新していくのである。そうすることで事業が成長する過程で将来のシステムのコストをより正確に推測できるようになる。

第 5 章

サーバレスコンピューティングへの展開

サーバレスコンピューティングを用いたアプリケーションに対する本提案手法の有効性について、実際のクラウドコンピューティング環境を用いてケーススタディを行う。クラウドコンピューティング環境には Amazon Web Services (AWS) を用いる。当ケーススタディに用いるアプリケーションは、ロードバランサと Web API、データベースで構成されたシンプルなアプリケーションである。ロードバランサ、Web API、データベースにはすべて AWS のサーバレスコンピューティングのサービスを用いる。

当ケーススタディではアプリケーションへの負荷シナリオをベースにコストを推定し、その後実際にワークロードの負荷をかける。サーバベースのアプリケーションと比べて、よりワークロードのイベントがコスト因子に直結するサーバレスコンピューティングでは、実際のクラウドリソースの追従性がそのままコスト推定の精度につながる。当ケーススタディではサーバレスコンピューティングにおけるクラウドリソースの追従性を検証することで、当手法がサーバレスコンピューティングにおいてより実務的に利用可能であることを示す。

5.1 サーバレスアプリケーションと負荷シナリオ

5.1.1 ケーススタディのシステム構成

図 5.1 は、当ケーススタディに用いるサーバレスサービスで構成されたアプリケーションである。当アプリケーションは 3 層に分けられ、ロードバランシング層、コンピュー

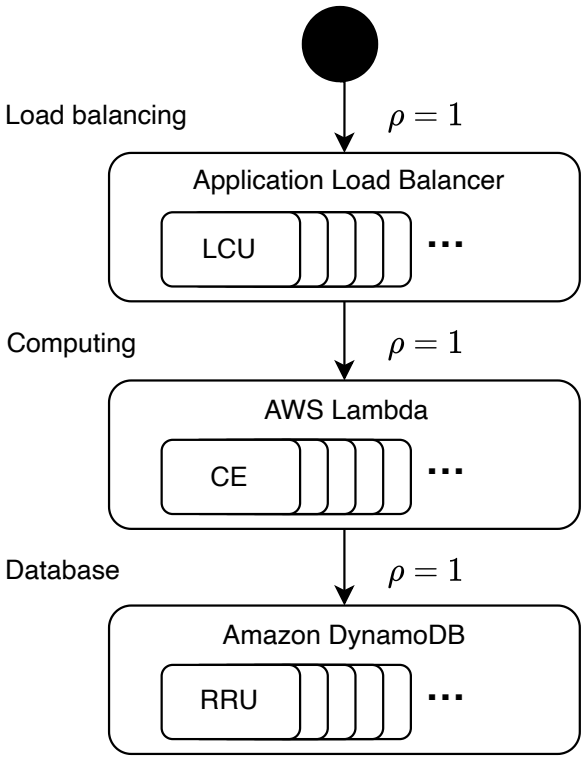

図 5.1　ケーススタディのシステム構成

ティング層、データベース層で構成される。当アプリケーションは AWS 上に構築される。ロードバランシグ層では Application Load Balancer (ALB) を利用し、コンピューティング層では AWS Lambda を利用し、データベース層では Amazon DynamoDB を利用する。当アプリケーションはプライベート環境に構築する。当アプリケーションはクラウドコンピューティングを用いた一般的な Web アプリケーションの構成である。クライアントは ALB にリクエストすることで当アプリケーションの機能を利用する。

ロードバランシング層の ALB は AWS 上で動作するアプリケーションレベルのロードバランサのサービスである。ALB は受け取ったリクエストを分散し、クライアントからのトラフィックをその他のサービスに転送する。ALB にはサーバの概念はなく、利用者は利用量に応じた料金を支払う。ALB では転送するリクエストの負荷に応じて内部的に自動でスケーリングをする。ALB では負荷が増加すると Load Balancer Capacity Units (LCU) という論理的なコンピューティングリソースが割り当てられる。負荷が減少すると、負荷の減少に応じて LCU は解放される。

コンピューティング層の AWS Lambda はサーバレスなコンピューティングサービスで

ある。AWS Lambda はサーバの概念が隠蔽されており、クラウドサービス利用者はサーバをプロビジョニング、管理、およびスケーリングをする必要なく、目的の計算処理を実行できる。AWS Lambda にソースコードをアップロードして、アップロード後に展開されたコードが呼び出されると、AWS Lambda がコードを実行する。AWS Lambda は転送するリクエストの負荷に応じて内部的に自動でスケーリングをする。AWS Lambda は負荷が増加すると Lambda 関数のインスタンスという論理的なクラウドリソースが割り当てられる。Concurrency Executions (CE) は Lambda 関数のインスタンスの数を指す。負荷が減少すると、 Lambda 関数の関数の未使用のインスタンスはアイドル状態へなったのちに解放される。

データベース層の Amazon DynamoDB は、自動的に容量を拡張する NoSQL データベースサービスである。Amazon DynamoDB はサーバが隠蔽されており、利用者はサーバをプロビジョニング、管理、およびスケーリングをする必要なく、テーブルの作成とデータを格納と取得ができる。Amazon DynamoDB はトラフィックやデータ量の変化に応じて内部的に自動でスケーリングをする。RRU (Read Request Units) は Amazon DynamoDB におけるスループットの指標である。RRU は Amazon DynamoDB が 1 秒間に実行可能な読み取り操作の数を意味する論理的なコンピューティングリソースである。Amazon DynamoDB への読み取り頻度が減少すると、RRU は自動的に減少する。

5.1.2 ケーススタディの負荷シナリオ

当ケーススタディではリクエスト量を 3 時間の間に変動させる。図 5.2 は当ケーススタディの負荷シナリオである。7:30 から 10:30 までクライアントからアプリケーションにリクエストを行い、8 時台と 10 時台にリクエスト量がピークになるようリクエスト量を変動させる。

5.2 サーバレスサービスのクラウドリソース利用量

サーバレスサービスは、サーバの存在を隠蔽することによりクラウドリソースの抽象化と柔軟な課金体系を実現している。サーバレスサービスはクラウドプロバイダによってクラウドリソースの処理基盤が管理されているため、利用者は仮想マシンや物理サーバの容量計画、設定、管理、メンテナンス、スケーリングを意識する必要がない。そのためサー

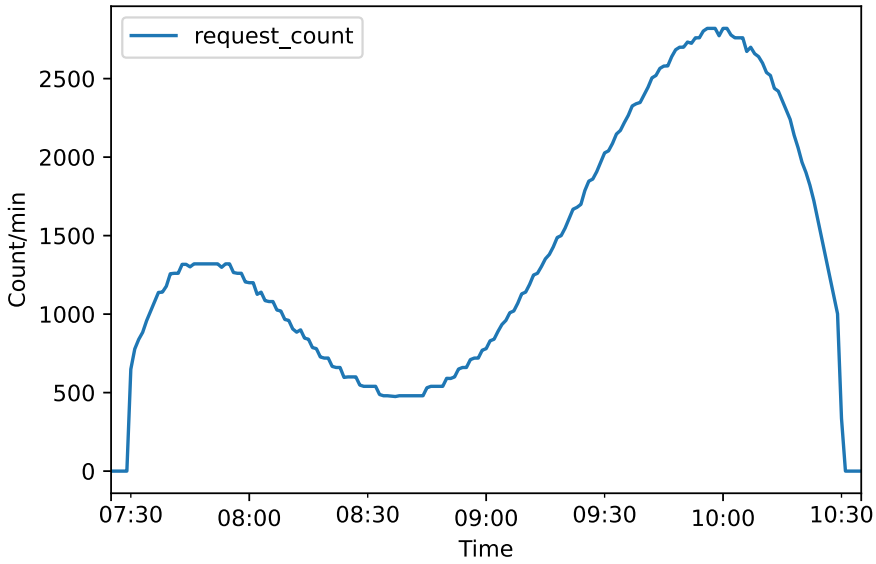

図 5.2　ケーススタディの負荷シナリオ

バレスコンピューティングにおいては、必要な時に必要なクラウドリソースを従量制で利用できる。

サーバレスコンピューティングでは課金の仕組みがサーバの概念から切り離されているため、サーバ単位のコスト構造から切り離されたコスト構造を持つことが多い。たとえば当ケーススタディのアプリケーションを構成する ALB, AWS Lambda, Amazon DynamoDB においても、独自のクラウドリソースの単位が定義されている。これら抽象化されたクラウドリソースは、LCU や Lambda 関数のインスタンス、RRU など、 仮想マシンベースのクラウドリソースの単位と比べて小さい単位である。またクラウドリソースのスケーリング指標においても、 仮想マシンインスタンスの CPU やメモリ、I/O などといったサーバ依存の指標ではなく、コネクション数や読み取り関数など、よりトラフィックから把握しやすい指標を用いていることが多い。

サーバレスサービスがサーバベースの IaaS と比べてより小さいクラウドリソースの単位でスケーリングすることにより、サーバレスサービスのクラウドリソース利用量はシステム負荷に対して高い追従性を持つ。たとえばケーススタディのシステム負荷に対して、コンピューティング層に仮想マシンを用いるとする。仮想マシンベースのスケーリングは AWS Lambda や Amazon DynamoDB といったサーバレスサービスほど細かいスケーリング特性を持たないため、仮想マシンベースのスケーリングでは図 5.3 のように階段状に

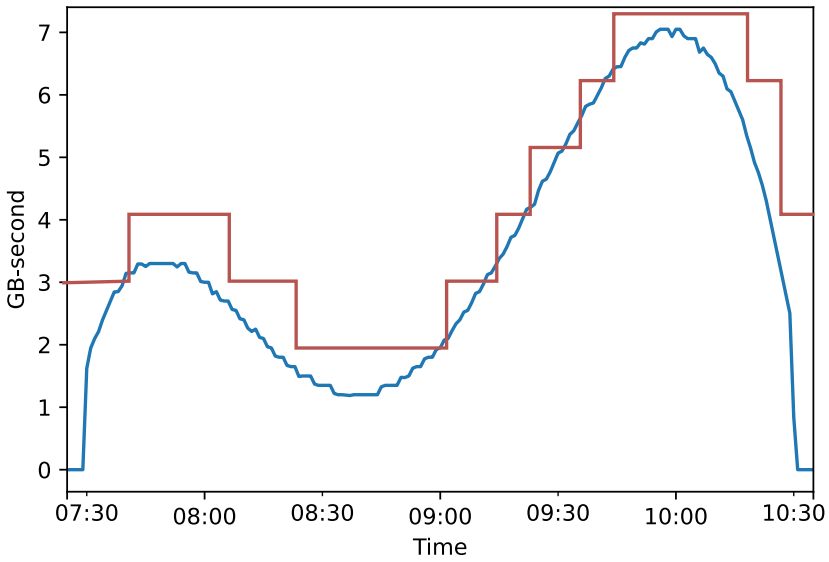

図 5.3　サーバレスサービスの特性を持たないスケーリングの例

スケーリングする。サーバレスコンピューティングでは、クラウドリソースの単位が小さく、また負荷への追従性が高いため、 仮想マシンベースのスケーリングを用いる場合と比べてより細かい階段を描く。

5.3　2 ノード間の関係の推定

マルチコンポーネントアプリケーションの基本単位はノードとノード間の関係である。第 3 章にて提案した手法を適用することで、アプリケーションを構成する 2 つのコンポーネント間の関係を推定することができる。2 つのノード間関係を得ることで、アプリケーションにシステム負荷 (イベント) が発生した際に、どのクラウドリソースがどれだけ利用されるかを推定できる。当ケーススタディではロードバランシング層、コンピューング層、データベース層が、当システムのネットワークトポロジーのノードとして 2 ノード間の関係を推定していく。

ロードバランシング層でトラフィック、つまりイベントが発生した場合、コスト因子であるクラウドリソースにどのような影響が出るかは式 (5.1) で表せる。式 (5.1) は、イベントごとに必要となる ALB の LCU 数を表している。

$$\mathrm{med}_{\mathrm{loadbalancing/LCU}}(e_t) = 0.000004 \times e_t \tag{5.1}$$

コンピューティング層でイベントが発生した場合のコスト因子の影響は以下の式で表せる。式 (5.2) は 1 分あたりの GB-second、式 (5.3) はイベントの数、つまり AWS Lambda 関数の呼び出し回数を表している。

$$\mathrm{med}_{\mathrm{computing/GB\text{-}second}}(e_t) = 0.0025 \times e_t \tag{5.2}$$

$$\mathrm{med}_{\mathrm{computing/request}}(e_t) = e_t \tag{5.3}$$

データベース層でイベントが発生した場合に、コスト因子がどのように影響するかは以下の式で表せる。式 (5.4) は、イベントごとに必要な DynamoDB の RRU(Read Request Units) 数を算出している。

$$\mathrm{med}_{\mathrm{database/RRU}}(e_t) = 0.5 \times e_t \tag{5.4}$$

図 5.1 により、クライアント (出発点) からロードバランシング層、ロードバランシング層からコンピューティング層、コンピューティング層からデータベース層の関係 ρ はそれぞれ 1 である。よってクライアントを含む任意の 2 ノード間の関係を式 (5.5) のように計算する。

$$\begin{aligned} R &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \hat{R} &= \sum_{k=1}^{3} R^k = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{5.5}$$

ケーススタディの固定リソース

クラウドリソースのなかには基本サービス料などのイベント量とは無関係に定額のコストとなるものがある。例えばロードバランサのアップタイムや当ケーススタディにおけるストレージのコストなどである。このような定額なクラウドリソースは、イベント量とは別に計算する。たとえば ALB の価格設定には利用量に基づくコスト以外に、負荷にかかわらず ALB を稼働する時間分 (アップタイム) のコストが含まれる。また今回のケーススタディではテストデータとして約 1.1GB のデータが DynamoDB に保存されている。DynamoDB の使用リソースを計算する際には、このデータのストレージ使用量を考慮する必要がある。したがってこの固定リソースは fixed(uptime for ALB) と fixed(1.1GB DynamoDB) で表現する。

表 5.1　Application Locad Balancer (ALB) の価格設定

コスト因子	費用
LCUs	$0.008 per LCU-hour (or partial hour)
Uptime	$0.0243 per ALB-hour (or partial hour)

表 5.2　AWS Lambda (ALB) の価格設定

コスト因子	費用
GB-second	$0.0000166667 for every GB-second
リクエスト数	$0.20 per 1M requests

5.4　トータルコストの推定

クラウドリソースの使用量が決まればコスト計算が可能になる。ALB のコスト計算では、コスト単位は時間であり、1 時間中の LCU のピーク値をコスト計算に使用する。コスト計算は、1 時間あたりの最大 LCU 数で行う必要がある。今回の実験では、7:30 から 10:30 まで負荷がかかっている。したがって、7:30-8:00, 8:00-9:00, 9:00-10:00, 10:00-10:30 に LCU として使用する資源は以下の通りである。

$$\begin{aligned}&\mathrm{res}_{\mathrm{loadbalancing}}(e_t)\\&= \begin{cases} \mathrm{med}_{\mathrm{loadbalancing/LCU}}(\max(e_1,\ldots,e_{30})) & \text{if } 1 \le t \le 30 \\ \mathrm{med}_{\mathrm{loadbalancing/LCU}}(\max(e_{31},\ldots,e_{90})) & \text{if } 31 \le t \le 90 \\ \mathrm{med}_{\mathrm{loadbalancing/LCU}}(\max(e_{91},\ldots,e_{150})) & \text{if } 91 \le t \le 150 \\ \mathrm{med}_{\mathrm{loadbalancing/LCU}}(\max(e_{151},\ldots,e_{180})) & \text{if } 151 \le t \le 180 \end{cases}\end{aligned} \tag{5.6}$$

また ALB 層が使用する固定的なクラウドリソースのコストも計算する必要がある。このコストは、1 分あたり $0.0243/60min である。負荷分散層のコストは表 5.1 を用いて以下のように表される。

$$\begin{aligned}&\mathrm{cost}_{\mathrm{loadbalancing}}(e_t) = \mathrm{res}_{\mathrm{loadbalancing/LCU}}(e_t) \times \$0.008/60 + \$0.0243/60\\&\sum_{t=1}^{180} \mathrm{cost}_{\mathrm{loadbalancing}}(e_t) \simeq \$0.073096\end{aligned} \tag{5.7}$$

コンピューティング層のコストを計算するためには、GB-second の量とリクエスト数を考慮する必要がある。コンピューティング層で必要とされるクラウドリソース $\mathrm{med}_{\mathrm{computing}}$ と割り当てされるクラウドリソース $\mathrm{res}_{\mathrm{computing}}$ はほぼ同じであるため、

表 5.3　Amazon DynamoDB の価格設定

コスト因子	費用
RRUs	$0.285 per million read request units
Data storage	First 25 GB-month is free
	0.285 per GB-month thereafter

式 (5.2), 式 (5.3), 表 5.2 からコストが計算できる (式 (5.8))。

$$\begin{aligned} \mathrm{cost}_{\mathrm{computing}}(e_t) &= \mathrm{res}_{\mathrm{computing/GB\text{-}second}}(e_t) \times \$0.0000166667 \\ &\quad + \mathrm{res}_{\mathrm{computing/request}}(e_t) \times \$0.20/1\mathrm{M} \\ \sum_{t=1}^{180} \mathrm{cost}_{\mathrm{computing}}(e_t) &\simeq \$0.062814 \end{aligned} \tag{5.8}$$

データベース層のコストを計算するには、固定コストだけでなく、ワークロードに伴うコストも考慮する必要がある (表 5.3)。しかし、DynamoDB の固定費は 25G バイト以下が無料であり、今回の事例では 1.1GB しか使っていない。したがって、データベース層のコストには、ワークロードに伴うコストのみが含まれる (式 (5.9))。

$$\begin{aligned} \mathrm{cost}_{\mathrm{database}}(e_t) &= \mathrm{res}_{\mathrm{database/RRU}}(e_t) \times \$0.285/1\mathrm{M} \\ \sum_{t=1}^{180} \mathrm{cost}_{\mathrm{database}}(e_t) &\simeq \$0.037039 \end{aligned} \tag{5.9}$$

まとめると、ケーススタディにおける期間中の総費用は以下になる。

$$\begin{aligned} &\sum_{t=1}^{180} \mathrm{cost}_{\mathrm{loadbalancing}}(e_t) + \sum_{t=1}^{180} \mathrm{cost}_{\mathrm{computing}}(e_t) + \sum_{t=1}^{180} \mathrm{cost}_{\mathrm{database}}(e_t) \\ &\quad \simeq \$0.073096 + \$0.062814 + \$0.037039 \\ &\quad \simeq \$0.172949 \end{aligned} \tag{5.10}$$

5.5　実環境におけるクラウドリソース利用量

当ケーススタディは小規模なものであるため、コスト推定により得られたコストはわずか $0.172949/3h と安価なものとなっている。しかしこのコスト推定により得られるコストが正確であれば、同じ手続きを用いて、より大きなケースのコストを計算できる。クラウドコンピューティングではクラウドプロバイダにより割り当てるクラウドリソースに対応する形で価格付けが行われる。つまりクラウドリソース利用量に応じてクラウドリソー

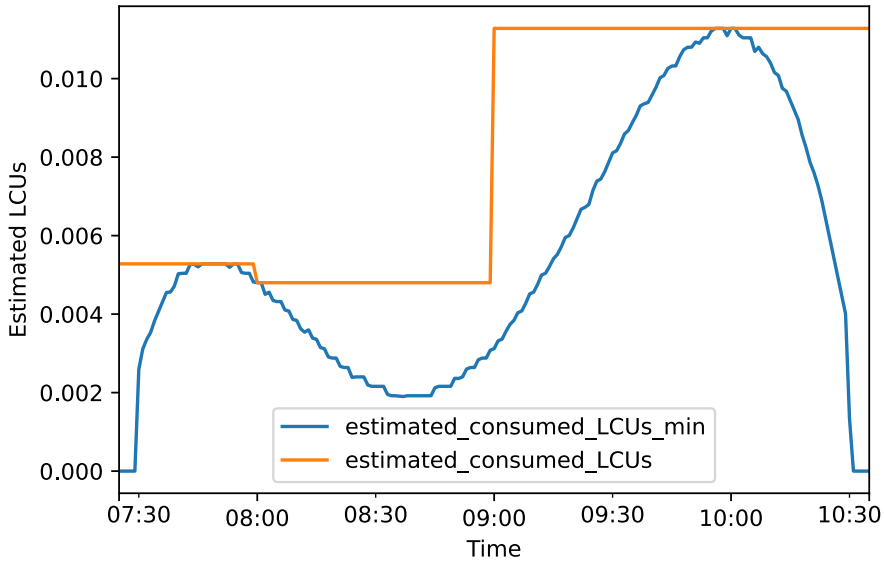

図 5.4　Application Load Balancer (ALB) の推定 LCU

スの割り当てが追従すればするほど、クラウドリソース利用量とずれのないコスト試算を得ることができる。

当ケーススタディではクラウドリソースの割り当ての精度を実証する。サーバレスサービスのコストはクラウドリソース利用量に基づくため、クラウドリソースの推定値の精度を示すことでこれを実現した。

5.5.1　ロードバランシング層におけるクラウドリソース利用量

ロードバランシング層ではシナリオの負荷に応じて LCU の数が変動する。図 5.4 は式 (5.1) から推定した ALB の LCU と、コスト計算するために切り上げた LCU を表している。青色の線 (`estimated_consumed_LCUs_min`) が推定した ALB の LCU であり、橙色の線 (`estimated_consumed_LCUs`) が切り上げた LCU である。

コストの計算では 1 時間単位ごとの内、最も高い LCU を計算に用いる。当ケーススタディでは 7:30 から 10:30 までの 3 時間を実験期間としているため、7:30〜8:00、8:00〜9:00、9:00〜10:00、10:00〜10:30 の LCU のピークがコスト計算に用いられる。LCU の変動に応じて切り上げ操作およびコストの算出は一意に定まり、切り上げ操作はコスト因子となるクラウドリソースの割り当てを意味する関数 res に対応する。なお当ケーススタ

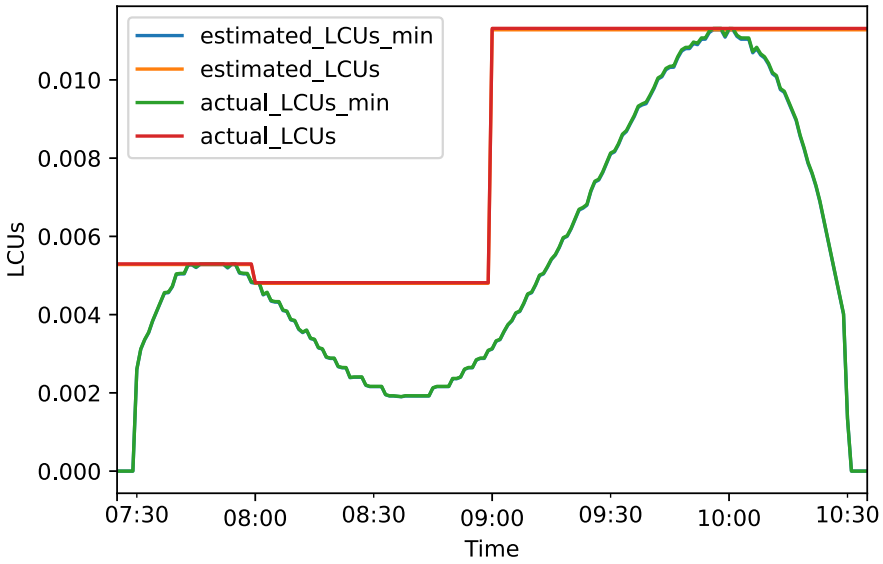

図 5.5　Application Load Balancer (ALB) の推定 LCU と実測 LCU の比較

ディでは割り当て関数 res をコスト関数 cost に含める。

図 5.5 は、推定した ALB の LCU を表す図 5.4 に実環境における実測値を重ねた図である。緑色の線 (`actual_consumed_LCUs_min`) が測定した ALB の LCU であり、赤色の線 (`actual_consumed_LCUs`) が測定した ALB を切り上げた値である。推定 LCU 数 (青線) と実 LCU 数 (緑線) がほぼ等しいため、図 5.5 ではまるで測定した ALB の LCU と測定した ALB を切り上げた値の線のみが描画されているように見える。

推定 LCU 数と実 LCU 数がほぼ等しいのは、ひとえに ALB のクラウドリソースの追従性によるものである。図 5.6 はリクエストごとの LCU の変動を表している。図 5.6 から事前に設定した推定 LCU に対して 1.5% ほどの変動しかないことが確認できる。これはつまりベンチマークから推定したリクエストあたりの LCU にほぼ対応する形で、実際の LCU が割り当てることを意味する。

5.5.2　コンピューティング層におけるリソースの利用量

コンピューティング層ではシナリオの負荷に応じて G-second の量が変動する。図 5.7 は式 (5.2) から推定した GB-second を表している。青色の線は推定した AWS Lambda の GB-second である。

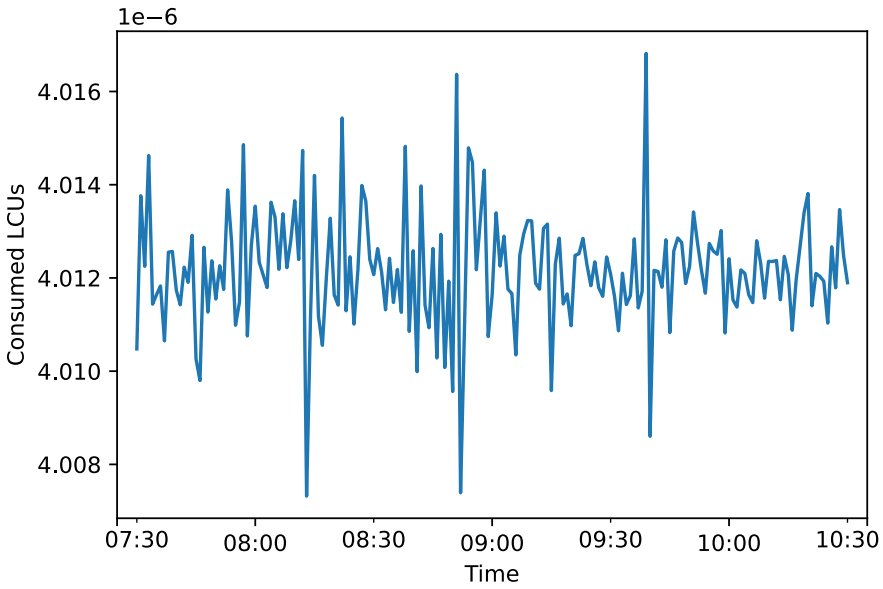

図 5.6　リクエストごとの Application Load Balancer (ALB) の実測 LCU

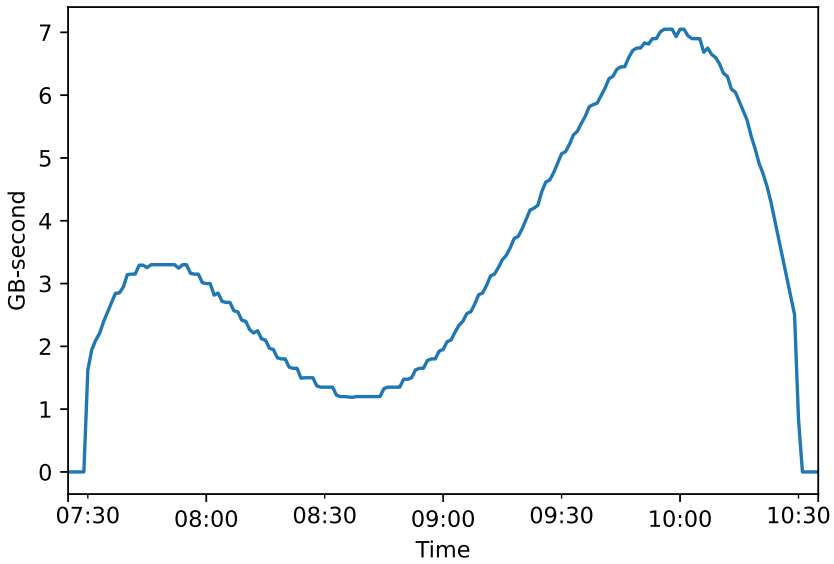

図 5.7　AWS Lambda の推定 GB-second

AWS Lambda のコストの計算では 呼び出し回数と、Lambda 関数へ割り当てるメモリに Lambda 関数の実行時間を掛けた値である GB-second を用いる。呼び出し回数はロードバランサの呼び出し回数と等しい。呼び出し回数は推定と実測で等しいことが明らかなため、コスト因子としてのリクエスト数に関しては言及しない。

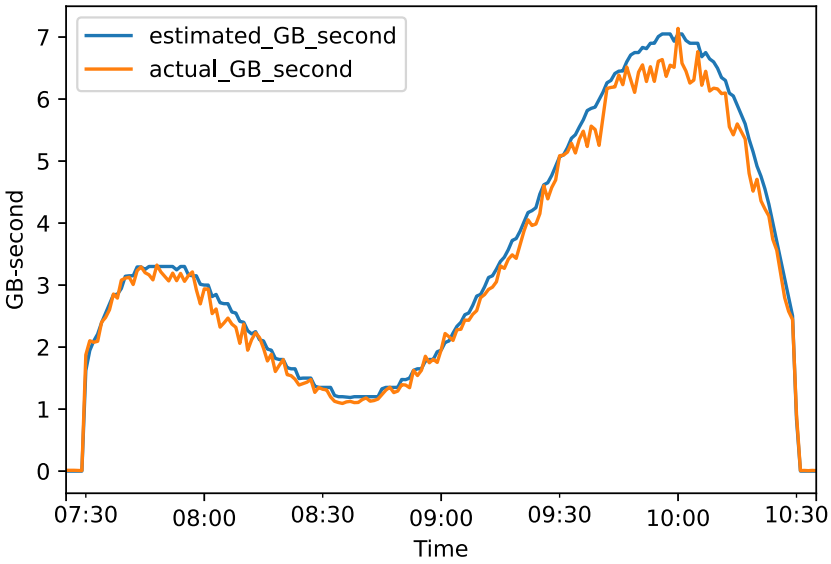

図 5.8　AWS Lambda の推定 GB-second と実測 GB-second の比較

図 5.8 は、推定した AWS Lambda の GB-second を表す図 5.7 に実環境における実測値を重ねた図である。青色の線 (`estimated_GB_second`) が推定した AWS Lambda の GB-second のであり、橙色の線 (`actual_consumed_LCUs`) が測定した AWS Lambda の GB-second の値である。図 5.8 から GB-seconds の推定値と実測値は、おおむね同じような変動を示していることが分かる。

ロードバランシング層の結果と比較してばらつきに違いがあるのは、Lambda 関数の GB-second の変動率が ALB の LCU の変動率と比べて相対的に大きいためである。Lambda 関数のばらつきのは処理時間の影響と相関がある。図 5.9 は Lambda 関数の呼び出しごとの処理時間のばらつきを表している。Lambda 関数に変動はあるものの、処理時間は 17〜20ms の間で安定している。この変動により GB-second の推定量と実測量にわずかな差が生じている。しかしこの差は許容誤差の範囲内であると考えられる。

5.5.3　データベース層におけるクラウドリソース利用量

データベース層ではシナリオの負荷に応じて RRU の量が変動する。当アプリケーションは読み出し専用であるため、ディスクサイズは実験中常に 1.1GB に固定されている。図 5.10 は式 (5.4) から推定した RRU を表している。青色の線はが推定した Amazon

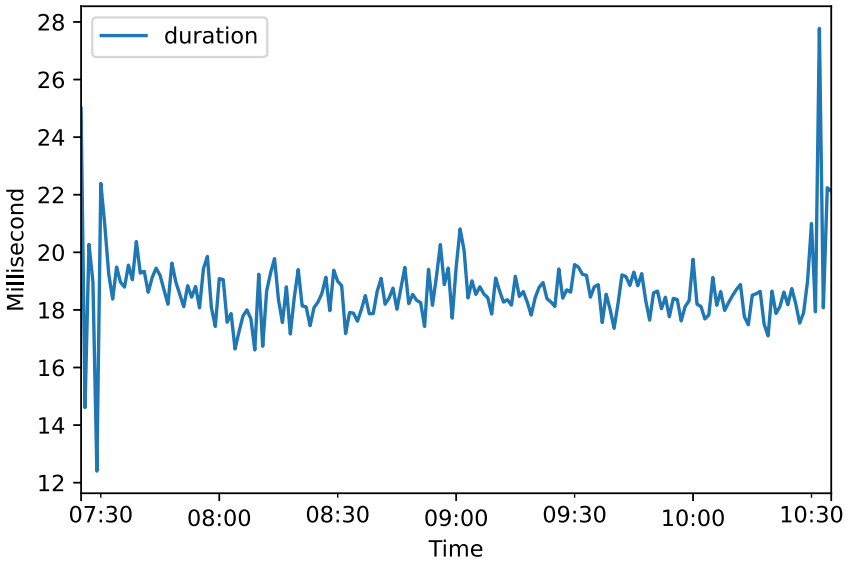

図 5.9　リクエストごとの AWS Lambda の実測 GB-second

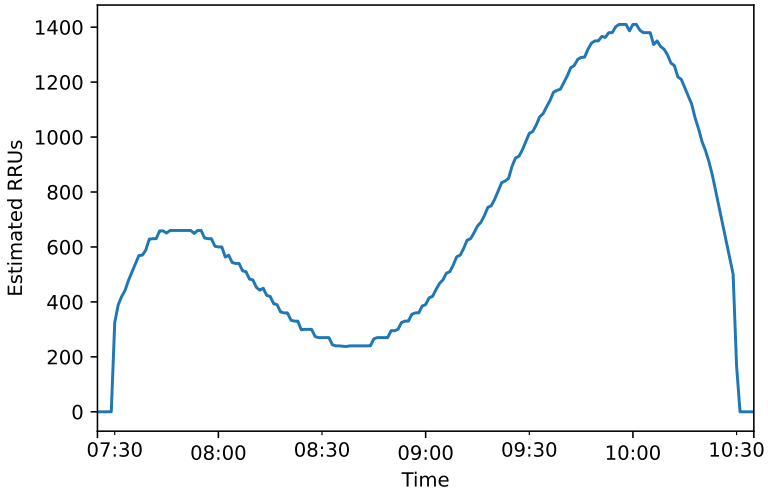

図 5.10　Amazon DynamoDB の推定 RRU

DynamoDB の RRU である。

データベース層の構成する Amazon DynamoDB のコスト因子は、ディスクサイズと RRU である。データベース層の読み出し回数はコンピューティング層の呼び出し回数と等しい。

図 5.11 は、推定した Amazon DynamoDB の RRU を表す図 5.10 に実環境における

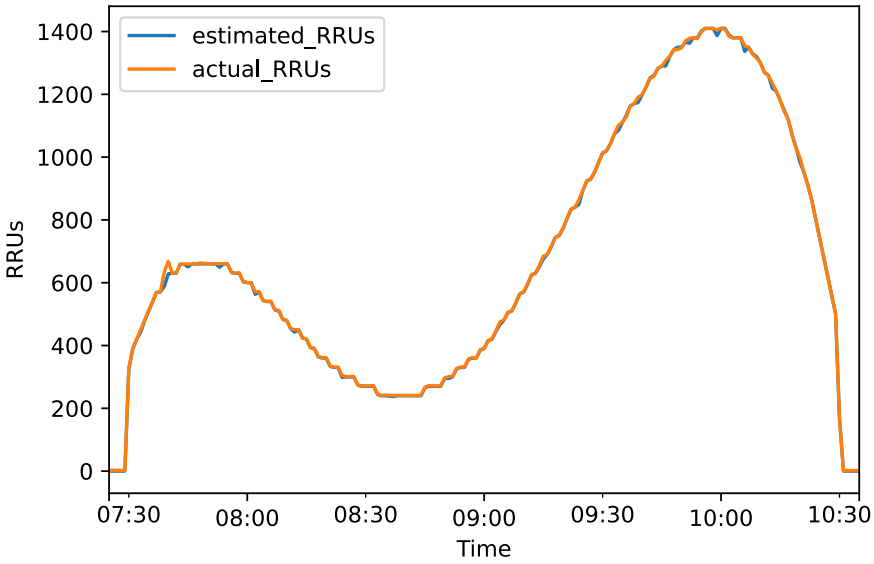

図 5.11　Amazon DynamoDB の推定 RRU と実測 RRU の比較

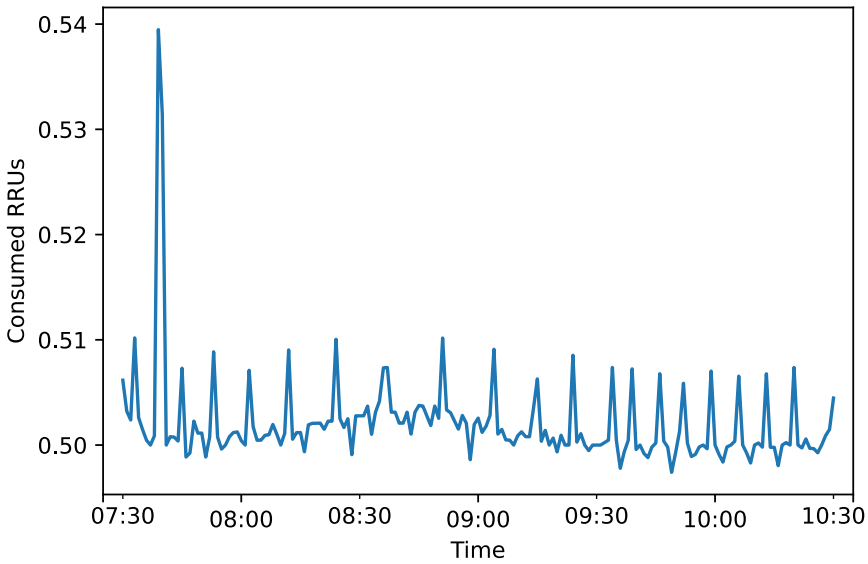

図 5.12　リクエストごとの Amazon DynamoDB の実測 RRU

実測値を重ねた図である。青色の線 (`estimated_RRUs`) が推定した Amazon DynamoDB の RRU であり、橙色の線 (`actual_RRUs`) が測定した Amazon DynamoDB の RRU の値である。図 5.11 から RRU の推定値と実測値はおおむね同じような変動を示していることが分かる。

ロードバランシング層やコンピューティング層と同様にばらつきに違いがあるのは、割り当てられる RRU に変動があるためである。図 5.12 はデータベース層の呼び出しごとに割り当てられる RRU のばらつきを表している。RRU の割り当てに変動はあるものの、割り当て数はおおむね 0.50 付近で安定している。この変動により RRU の推定量と実測量にわずかな差が生じている。しかしこの差は許容誤差の範囲内であると考えられる。

5.6 サーバレスコンピューティングのコスト追従性

以上のように、実験結果から、サーバレスコンピューティングにおいては計算資源の推定がより正確に行えることがわかる。これはつまり、計算されたコストにおいても正確に推定が可能であることを意味する。この大きな要因は、推定する粒度の違いとその追従性である。これまでの章では、仮想マシンをコスト単位とする IaaS のコスト推定を説明したが、本章ではサーバレスコンピューティングのコスト推定を提案してきた。IaaS とサーバレスコンピューティングはコスト構造が異なるため、IaaS とサーバレスコンピューティングを比較すると、IaaS の方が課金の単位が大きく、サーバレスコンピューティングの課金の単位が小さくなっている。課金の単位が異なること、負荷に応じてクラウドリソースの割り当てが行われる追従性の違いにより、サーバレスコンピューティングではより使用量に直接的にマッチしたクラウドリソースを消費できる。

第 6 章

結論

本論ではクラウドコンピューティングの定義から始め、現在のクラウドコンピューティングの特性に基づいたコスト推定手法を提案した。本論で提案したコスト推定手法は、クラウドサービスの利用者の立場から、リリースまでの時間が少なく、解析的な手法やシミュレーションツールを使いこなす有識者がいなくても、ある程度許容可能なコスト推定ができる手法である。当該コスト手法はクラウドリソースの使用量に制限を設けず、QoS を満たすことを前提に自動でスケーリングするなど、現在のクラウドサービスの仕様および活用の事例を前提としている。

第 3 章では、本論で提案するコスト推定手法の基本的な考え方、クラウドリソース利用量の推定手続き、そして対象期間におけるアプリケーション全体のコストを算出するまでの手続きを説明した。当コスト推定手法では、以下のコスト推定手順にもとづき、対象期間におけるアプリケーション全体のコストを推定する。

1. コスト推定対象のアプリケーションを DAG としてモデル化し、隣接する 2 つのコンポーネント間の関係を重み付き隣接行列で表現する。
2. 隣接する 2 つのコンポーネント間の関係を表す重み付き隣接行列の積和計算により、任意の 2 つのコンポーネント間の関係を得る。
3. 出発点とコンポーネント間の関係をすべて抽出し、イベント量を引数に各コンポーネントのリソースの利用量を求めるための関数を得る。
4. 各コンポーネントのリソースの利用量に各コンポーネントのコスト関数を適用し、イベント量を引数に各コンポーネントのコストを求めるコスト関数を得る。
5. イベント量から各コンポーネントのコストを求めるコスト関数を合計し、イベント

量を引数にアプリケーション全体のコストを求めるコスト関数を得る。

6. 対象期間におけるイベント量のリストに、イベント量を引数にアプリケーション全体のコストを求めるコスト関数を適用し、得られた時点ごとのアプリケーション全体のコストをすべて合計する。

第 4 章では、サーバベースアプリケーションのケーススタディに第 3 章のコスト推定手法を適用し、どのようにコスト推定プロセスを進めていくかを説明した。そしてサーバベースアプリケーションのケーススタディによる実験の結果から、サーバベースサービスの特徴により、当該コスト推定手法から得たリソースの利用量との間に仮想マシンインスタンスのサイズに伴う過大評価と過小評価が発生する可能性を示した。第 4 章で得られた成果を以下にまとめる。

- サーバベースアプリケーションのケーススタディにて、当該コスト推定手法を適用し、隣接する 2 つのコンポーネント間の関係から、イベント量を引数にアプリケーション全体のコストを求めるコスト関数を得られることを示した。
- サーバベースアプリケーションのケーススタディにて、イベント量を引数にアプリケーション全体のコストを求めるコスト関数から、対象期間におけるアプリケーション全体のコストが求められることを示した。
- サーバベースアプリケーションのケーススタディにて、サーバベースの特徴により仮想マシンインスタンスのサイズに伴う過大評価と過小評価が発生する可能性を示した。

第 5 章では、サーバレスアプリケーションに第 3 章のコスト推定手法を適用し、どのようにコスト推定プロセスを進めていくかを説明した。そしてサーバレスアプリケーションのケーススタディの結果から、サーバレスサービスの特徴により、サーバレスサービスで構成したアプリケーションにおいて、当該コスト推定手法がより効果的に活用できることを示した。第 5 章で得られた成果を以下にまとめる。

- サーバレスアプリケーションのケーススタディにて、当該コスト推定手法を適用し、隣接する 2 つのコンポーネント間の関係から、イベント量を引数にアプリケーション全体のコストを求めるコスト関数を得られることを示した。
- サーバレスアプリケーションのケーススタディにて、イベント量を引数にアプリ

ケーション全体のコストを求めるコスト関数から、対象期間におけるアプリケーション全体のコストが求められることを示した。

- サーバレスサービスの特性により、サーバレスサービスで構成したアプリケーションにおいて、当該コスト推定手法がより効果的に活用できることを示した。

本研究ではビジネスの立ち上げフェーズにおける簡便なコスト推定手法を提案した。クラウドサービスプロバイダのたゆまぬ努力により、クラウドコンピューティングのコスト構造は、クラウドサービス利用者にとってより直感的かつ直接的なものになっている。特にサーバレスコンピューティングにおいては、当該コスト推定手法における結果に近い実験結果が得られており、今後のクラウドサービスの進化によってはその精度はさらに高くなるものと期待している。一方で隣接する 2 つのコンポーネント間の関係にもとづく当提案手法において、リリース後のモニタリングから得られる推定値と実測値の差による 2 つのコンポーネント間の関係の見直しには議論の余地がある。今後の研究においては、クラウドコンピューティングの進化およびコスト構造の変化に追従するとともに、リリース後における 2 つのコンポーネント間の関係の更新プロセスについて取り扱っていきたい。

謝辞

本論文の作成にあたり終始適切な助言を賜り、また丁寧に指導して下さった吉田健一教授に感謝申し上げます。また研究のアプローチや考察の方法などについて適切な助言を下さりました津田和彦教授、木野泰伸准教授、岡島敬一教授、秋田県立大学の鈴木一哉教授にも心より感謝申し上げます。

参考文献

[AAP+13] Bernardetta Addis, Danilo Ardagna, Barbara Panicucci, Mark S Squillante, and Li Zhang. A hierarchical approach for the resource management of very large cloud platforms. *IEEE Transactions on Dependable and Secure Computing*, Vol. 10, No. 5, pp. 253–272, 2013.

[AEMM13] Vahid Abedifar, Mohammad Eshghi, Seyedali Mirjalili, and S Mohammad Mirjalili. An optimized virtual network mapping using PSO in cloud computing. In *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, pp. 1–6. IEEE, 2013.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, Vol. 53, No. 4, pp. 50–58, 2010.

[Ama22a] Amazon Web Services, Inc. Application and Infrastructure Monitoring – Amazon CloudWatch – Amazon Web Services. https://aws.amazon.com/cloudwatch/, 2022. accessed Dec. 30, 2022.

[Ama22b] Amazon Web Services, Inc. AWS Auto Scaling. https://aws.amazon.com/autoscaling/, 2022. accessed Dec. 30, 2022.

[Ama22c] Amazon Web Services, Inc. AWS Cost Explorer - Amazon Web Services. https://aws.amazon.com/aws-cost-management/aws-cost-explorer/, 2022. accessed Dec. 30, 2022.

[Ama22d] Amazon Web Services, Inc. AWS Cost Explorer FAQs - Amazon Web Services. https://aws.amazon.com/aws-cost-management/aws-cost-explorer/features/, 2022. accessed Dec. 30, 2022.

[Ama22e] Amazon Web Services, Inc. AWS Management Console. https://aws.amazon.com/console/, 2022. accessed Dec. 30, 2022.

[Ama22f] Amazon Web Services, Inc. AWS Pricing Calculator. https://calculator.aws/, 2022. accessed Dec. 30, 2022.

[Ama22g] Amazon Web Services, Inc. AWS Pricing Calculator. https://calculator.aws/, 2022. accessed Dec. 30, 2022.

[Ama22h] Amazon Web Services, Inc. AWS Product and Service Pricing — Amazon Web Services. https://aws.amazon.com/pricing/, 2022. accessed Dec. 30, 2022.

[Ama22i] Amazon Web Services, Inc. Cloud Computing Services - Amazon Web Services (AWS). https://aws.amazon.com/, 2022. accessed Dec. 30, 2022.

[Ama22j] Amazon Web Services, Inc. Command Line Interface - AWS CLI - AWS. https://aws.amazon.com/cli/, 2022. accessed Dec. 30, 2022.

[Ama22k] Amazon Web Services, Inc. Full Stack Development - Web and Mobile Apps - AWS Amplify. https://aws.amazon.com/amplify/, 2022. accessed Dec. 30, 2022.

[Ama22l] Amazon Web Services, Inc. Global Infrastructure Regions & AZs. https://aws.amazon.com/about-aws/global-infrastructure/regions_az/, 2022. accessed Dec. 30, 2022.

[Ama22m] Amazon Web Services, Inc. Provision Infrastructure as Code - AWS CloudFormation - AWS. https://aws.amazon.com/cloudformation/, 2022. accessed Dec. 30, 2022.

[Ama22n] Amazon Web Services, Inc. Secure and resizable cloud compute – Amazon EC2 – Amazon Web Services. https://aws.amazon.com/ec2/, 2022. accessed Dec. 30, 2022.

[Ama22o] Amazon Web Services, Inc. Serverless Computing - AWS Lambda - Amazon Web Services. https://aws.amazon.com/lambda/, 2022. accessed Dec. 30, 2022.

[Ama22p] Amazon Web Services, Inc. Website & Web App Deployment - AWS Elastic Beanstalk - AWS. https://aws.amazon.com/elasticbeanstalk/, 2022. ac-

cessed Dec. 30, 2022.

[BA12] Eric Bauer and Randee Adams. *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[Bas22] Chiradeep BasuMallick. What Is Broad Network Access? Definition, Key Components, and Best Practices — Spiceworks. https://www.spiceworks.com/tech/cloud/articles/what-is-broad-network-access/, 2022. accessed Dec. 30, 2022.

[BCC+17] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research advances in cloud computing*, pp. 1–20. Springer, 2017.

[BCS19] Antonio Brogi, Andrea Corradini, and Jacopo Soldani. Estimating costs of multi-component enterprise applications. *Formal Aspects of Computing*, Vol. 31, No. 4, pp. 421–451, 2019.

[BG14] BH Bhavani and HS Guruprasad. Resource provisioning techniques in cloud computing environment: a survey. 2014.

[BGDMT06] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[CB05] Susanta Nanda Tzi-cker Chiueh and Stony Brook. A survey on virtualization technologies. *Rpe Report*, Vol. 142, , 2005.

[Clo22] Alibaba Cloud. Pricing Calculator - Alibaba Cloud. https://www.alibabacloud.com/pricing-calculator#/, 2022. accessed Dec. 30, 2022.

[com22] The Linux Kernel community. KVM. https://www.linux-kvm.org/page/Main_Page, 2022. accessed Dec. 30, 2022.

[Cor22a] International Business Machines Corporation. IBM Cloud cost estimator — IBM. https://www.ibm.com/cloud/cloud-calculator, 2022. accessed Dec. 30, 2022.

[Cor22b] Microsoft Corporation. App Service — Build & Host Web Apps — Microsoft Azure. https://azure.microsoft.com/en-us/services/app-service/, 2022. accessed Dec. 30, 2022.

[Cor22c] Microsoft Corporation. Azure Autoscale — Microsoft Azure. https://azure.microsoft.com/en-us/products/virtual-machines/autoscale/, 2022. accessed Dec. 30, 2022.

[Cor22d] Microsoft Corporation. Azure Command-Line Interface (CLI) - Overview — Microsoft Learn. https://learn.microsoft.com/en-us/cli/azure/, 2022. accessed Dec. 30, 2022.

[Cor22e] Microsoft Corporation. Azure Functions – Serverless Functions in Computing — Microsoft Azure. https://azure.microsoft.com/en-us/services/functions/, 2022. accessed Dec. 30, 2022.

[Cor22f] Microsoft Corporation. Azure Monitor - Modern Observability Tools — Microsoft Azure. https://azure.microsoft.com/en-us/products/monitor/, 2022. accessed Dec. 30, 2022.

[Cor22g] Microsoft Corporation. Azure Resource Manager — Microsoft Azure. https://azure.microsoft.com/en-us/get-started/azure-portal/resource-manager/, 2022. accessed Dec. 30, 2022.

[Cor22h] Microsoft Corporation. Choose the Right Azure Region for You — Microsoft Azure. https://azure.microsoft.com/en-us/explore/global-infrastructure/geographies/, 2022. accessed Dec. 30, 2022.

[Cor22i] Microsoft Corporation. Microsoft 365 - Subscription for Office Apps — Microsoft 365. https://www.microsoft.com/en/microsoft-365, 2022. accessed Dec. 30, 2022.

[Cor22j] Microsoft Corporation. Microsoft Azure Portal — Microsoft Azure. https://azure.microsoft.com/en-us/get-started/azure-portal/, 2022. accessed Dec. 30, 2022.

[Cor22k] Microsoft Corporation. Mobile Apps — Microsoft Azure. https://azure.microsoft.com/en-us/solutions/mobile/, 2022. accessed Dec. 30, 2022.

[Cor22l] Microsoft Corporation. Pricing Calculator — Microsoft Azure. https://azure.microsoft.com/en-us/pricing/calculator/, 2022. accessed Dec. 30, 2022.

[Cor22m] Microsoft Corporation. Pricing Overview—How Azure Pricing Works — Microsoft Azure. https://azure.microsoft.com/en-us/pricing/, 2022. accessed Dec. 30, 2022.

[Cor22n] Microsoft Corporation. Virtual Machines (VMs) for Linux and Windows — Microsoft Azure. https://azure.microsoft.com/en-us/services/virtual-machines/, 2022. accessed Dec. 30, 2022.

[Cor22o] Oracle Corporation. Cloud Cost Estimator. https://www.oracle.com/uk/cloud/costestimator.html, 2022. accessed Dec. 30, 2022.

[Cor22p] Oracle Corporation. Cloud Infrastructure — Oracle. https://www.oracle.com/cloud/, 2022. accessed Dec. 30, 2022.

[CRB+11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, Vol. 41, No. 1, pp. 23–50, 2011.

[CRB+22] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudslab/cloudsim: CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services. https://github.com/Cloudslab/cloudsim, 2022. accessed Dec. 30, 2022.

[CVRM+10] Juan Cáceres, Luis M Vaquero, Luis Rodero-Merino, Alvaro Polo, and Juan J Hierro. Service scalability over the cloud. In *Handbook of Cloud Computing*, pp. 357–377. Springer, 2010.

[Das18] Dashbird. The Biggest Companies Using Serverless Right Now — Dashbird. https://dashbird.io/blog/companies-using-serverless-in-production/, 2018. accessed Dec. 30, 2022.

[DFZ+15] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. Everything as a service (XaaS) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pp. 621–628. IEEE, 2015.

[Doc22] Docker, Inc. Docker: Accelerated, Containerized Application Development. https://www.docker.com/, 2022. accessed Dec. 30, 2022.

[ESVE+20] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110*, 2020.

[FB14] Maram Mohammed Falatah and Omar Abdullah Batarfi. Cloud scalability considerations. *International Journal of Computer Science and Engineering Survey*, Vol. 5, No. 4, p. 37, 2014.

[FFH12] Florian Fittkau, Sören Frey, and Wilhelm Hasselbring. CDOSim: Simulating cloud deployment options for software migration support. In *2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, pp. 37–46. IEEE, 2012.

[FIMS17] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.

[Fou22] Linux Foundation. Home - Xen Project. https://xenproject.org/, 2022. accessed Dec. 30, 2022.

[Goo22] Google LLC. Cloud Computing Services — Google Cloud. https://cloud.google.com/, 2022. accessed Dec. 30, 2022.

[Goy14] Sumit Goyal. Public vs private vs hybrid vs community-cloud computing: a critical review. *International Journal of Computer Network and Information Security*, Vol. 6, No. 3, pp. 20–29, 2014.

[HM05] John Hughes and Eve Maler. Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, Vol. 13, , 2005.

[HMA19] Mohamed K Hussein, Mohamed H Mousa, and Mohamed A Alqarni. A placement architecture for a container as a service (CaaS) in a cloud environment. *Journal of Cloud Computing*, Vol. 8, No. 1, pp. 1–15, 2019.

[HT12] Mohammad Hamdaqa and Ladan Tahvildari. Cloud computing uncovered: a research landscape. *Advances in computers*, Vol. 86, pp. 41–85, 2012.

[iPR22] iMiller Public Relations. Self-Service Cloud: Benefits, Hurdles, and Solutions — Data Center POST. https://datacenterpost.com/self-service-cloud-benefits-hurdles-and/, 2022. accessed Dec. 30, 2022.

[JBF+05] Bart Jacob, Michael Brown, Kentaro Fukui, Nihar Trivedi, et al. Introduction to grid computing. *IBM redbooks*, pp. 3–6, 2005.

[KGP12] Gurudatt Kulkarni, Jayant Gambhir, and Rajnikant Palwe. Cloud computing-software as service. *vol*, Vol. 2, pp. 2–6, 2012.

[Kim09] Won Kim. Cloud computing: Today and tomorrow. *J. Object Technol.*, Vol. 8, No. 1, pp. 65–72, 2009.

[KS17] Zhanibek Kozhirbayev and Richard O Sinnott. A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, Vol. 68, pp. 175–182, 2017.

[LEB15] Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. Scalability, elasticity, and efficiency in cloud computing: A systematic literature review of definitions and metrics. In *Proceedings of the 11th international ACM SIGSOFT conference on quality of software architectures*, pp. 83–92, 2015.

[lin22] linuxcontainers.org. Linux Containers. https://linuxcontainers.org/, 2022. accessed Dec. 30, 2022.

[LLC22a] Google LLC. App Engine Application Platform — Google Cloud. https://cloud.google.com/appengine, 2022. accessed Dec. 30, 2022.

[LLC22b] Google LLC. Autoscaling groups of instances — Compute Engine Documentation — Google Cloud. https://cloud.google.com/compute/docs/autoscaler/, 2022. accessed Dec. 30, 2022.

[LLC22c] Google LLC. Cloud Functions — Google Cloud.

https://cloud.google.com/functions, 2022. accessed Dec. 30, 2022.

[LLC22d] Google LLC. Cloud Monitoring — Google Cloud. https://cloud.google.com/monitoring, 2022. accessed Dec. 30, 2022.

[LLC22e] Google LLC. Command Line Interface Gcloud Cli — Google Cloud. https://cloud.google.com/cli, 2022. accessed Dec. 30, 2022.

[LLC22f] Google LLC. Compute Engine: Virtual Machines (VMs) — Google Cloud. https://cloud.google.com/compute/, 2022. accessed Dec. 30, 2022.

[LLC22g] Google LLC. Firebase. https://firebase.google.com/, 2022. accessed Dec. 30, 2022.

[LLC22h] Google LLC. Global Locations - Regions & Zones — Google Cloud. https://cloud.google.com/about/locations, 2022. accessed Dec. 30, 2022.

[LLC22i] Google LLC. Google Cloud console - Web UI Admin. https://cloud.google.com/cloud-console, 2022. accessed Dec. 30, 2022.

[LLC22j] Google LLC. Google Cloud Deployment Manager documentation — Cloud Deployment Manager Documentation. https://cloud.google.com/deployment-manager/docs, 2022. accessed Dec. 30, 2022.

[LLC22k] Google LLC. Google Cloud Pricing Calculator. https://cloud.google.com/products/calculator/, 2022. accessed Dec. 30, 2022.

[LLC22l] Google LLC. Google Workspace — Business Apps & Collaboration Tools. https://workspace.google.com/, 2022. accessed Dec. 30, 2022.

[LLC22m] Google LLC. Pricing Overview — Google Cloud. https://cloud.google.com/pricing, 2022. accessed Dec. 30, 2022.

[LLL+09] Xinhui Li, Ying Li, Tiancheng Liu, Jie Qiu, and Fengchun Wang. The method and tool of cost analysis for cloud computing. In *2009 IEEE International Conference on Cloud Computing*, pp. 93–100. IEEE, 2009.

[MAJ+14] Lakshay Malhotra, Devyani Agarwal, Arunima Jaiswal, et al. Virtualization in cloud computing. *J. Inform. Tech. Softw. Eng*, Vol. 4, No. 2, pp. 1–3, 2014.

[MBS12] C Madhavaiah, Irfan Bashir, and Syed Irfan Shafi. Defining Cloud Computing in business perspective: a review of research. *Vision*, Vol. 16, No. 3, pp. 163–173, 2012.

[MG+11] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. 2011.

[Mic22] Microsoft Corporation. Cloud Computing Services — Microsoft Azure. https://azure.microsoft.com/en-us/, 2022. accessed Dec. 30, 2022.

[Mil12] Rich Miller. Estimate Amazon cloud backed by 450000 servers. *Data Center Knowledge*, 2012.

[MLB+11] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing—The business perspective. *Decision support systems*, Vol. 51, No. 1, pp. 176–189, 2011.

[MVML12] Rafael Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, Vol. 45, No. 12, pp. 65–72, 2012.

[MWT12] Benedikt Martens, Marc Walterbusch, and Frank Teuteberg. Costing of cloud computing services: A total cost of ownership approach. In *2012 45th Hawaii International Conference on System Sciences*, pp. 1563–1572. IEEE, 2012.

[Nat22] National Institute of Standards and Technology (NIST). Cloud Computing — CSRC. https://csrc.nist.gov/projects/cloud-computing, 2022. accessed Nov. 20, 2022.

[PCDGP11] Ricardo Pérez-Castillo, Ignacio Garcia-Rodriguez De Guzman, and Mario Piattini. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, Vol. 33, No. 6, pp. 519–532, 2011.

[PDCB17] Sareh Fotuhi Piraghaj, Amir Vahid Dastjerdi, Rodrigo N Calheiros, and Rajkumar Buyya. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, Vol. 47, No. 4, pp. 505–521, 2017.

[PH04] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *International workshop on unconventional programming paradigms*, pp. 257–269. Springer, 2004.

[PL15] Claus Pahl and Brian Lee. Containers and clusters for edge cloud architectures–a technology review. In *2015 3rd international conference on future internet of things and cloud*, pp. 379–386. IEEE, 2015.

[PRS09] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.

[PS13] Rajni Patil and R Singh. Scaling in cloud computing. *International Journal of Advance Research, IJOAR*, Vol. 1, pp. 21–27, 2013.

[Raj20] Arokia Paul Rajan. A review on serverless architectures-function as a service (FaaS) in cloud computing. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, Vol. 18, No. 1, pp. 530–537, 2020.

[Red22] Red Hat, Inc. Red Hat OpenShift makes container orchestration easier. https://www.redhat.com/en/technologies/cloud-computing/openshift, 2022. accessed Dec. 30, 2022.

[Res22] Precedence Research. Cloud Computing Market Size to Surpass USD 1,614.10 BN by 2030. https://www.precedenceresearch.com/cloud-computing-market, 2022. accessed Dec. 30, 2022.

[Sal22] Salesforce, Inc. Salesforce: The Customer Company. https://www.salesforce.com/, 2022. accessed Dec. 30, 2022.

[SBA13] Stelios Sotiriadis, Nik Bessis, and Nick Antonopoulos. Towards inter-cloud simulation performance analysis: Exploring service-oriented benchmarks of clouds in SimIC. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pp. 765–771. IEEE, 2013.

[SBJ+14] Natsuhiko Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. Openid connect core 1.0. *The OpenID Foundation*, p. S3, 2014.

[SBKA13] Stelios Sotiriadis, Nik Bessis, Pierre Kuonen, and Nick Antonopoulos. The inter-cloud meta-scheduling (ICMS) framework. In *2013 IEEE 27th Inter-*

national Conference on Advanced Information Networking and Applications (AINA), pp. 64–73. IEEE, 2013.

[Sch14] Edwin Schouten. Cloud computing defined: Characteristics & service levels - Cloud computing news. https://www.ibm.com/blogs/cloud-computing/2014/01/31/cloud-computing-defined-characteristics-service-levels-2/, 2014. accessed Dec. 30, 2022.

[Ser06] Jim Sermersheim. Lightweight directory access protocol (LDAP): The protocol. Technical report, 2006.

[SKD16] S Sujan and R Kanniga Devi. An efficient task scheduling scheme in cloud computing using graph theory. In *Proceedings of the International Conference on Soft Computing Systems*, pp. 655–662. Springer, 2016.

[SLO16] Anthony Simonet, Adrien Lebre, and Anne-Cécile Orgerie. Deploying distributed cloud infrastructures: Who and at what cost? In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pp. 178–183. IEEE, 2016.

[Son14] Mitesh Soni. Cloud computing basics—platform as a service (PaaS). *Linux Journal*, Vol. 2014, No. 238, p. 4, 2014.

[SSKY16] Akanksha Singh, Smita Sharma, Shipra Ravi Kumar, and Suman Avdesh Yadav. Overview of PaaS and SaaS and its application in cloud computing. In *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, pp. 172–176. IEEE, 2016.

[Sur16] Vijay Surwase. REST API modeling languages-a developer' s perspective. *Int. J. Sci. Technol. Eng*, Vol. 2, No. 10, pp. 634–637, 2016.

[TFF+13] Patricia Thaler, Norman Finn, Don Fedyk, Glenn Parsons, and Eric Gray. IEEE 802.1 Q, 2013.

[VETT+18] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uță, and Alexandru Iosup. Serverless is more: From paas to present cloud computing. *IEEE Internet Computing*, Vol. 22, No. 5, pp. 8–17, 2018.

[VMw22] VMware, Inc. VMware - Delivering a Digital Foundation For Businesses.

https://www.vmware.com/, 2022. accessed Dec. 30, 2022.

[VRMCL08] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition, 2008.

[VŠ17] T Vondra and J Šedivỳ. Cloud autoscaling simulation based on queueing network model. *Simulation Modelling Practice and Theory*, Vol. 70, pp. 83–100, 2017.

[VST+14] Jordi Vilaplana, Francesc Solsona, Ivan Teixidó, Jordi Mateo, Francesc Abella, and Josep Rius. A queuing theory model for cloud computing. *The Journal of Supercomputing*, Vol. 69, No. 1, pp. 492–507, 2014.

[WCJL22] JINFENG WEN, ZHENPENG CHEN, XIN JIN, and XUANZHE LIU. Rise of the Planet of Serverless Computing: A Systematic Review. 2022. accessed Dec. 30, 2022.

[YdAY+06] Chee Shin Yeo, Marcos Dias de Assunção, Jia Yu, Anthony Sulistio, Srikumar Venugopal, Martin Placek, and Rajkumar Buyya. Utility computing and global grids. *arXiv preprint cs/0605056*, 2006.

[Zei06] Kurt Zeilenga. Lightweight directory access protocol (ldap): Technical specification road map. Technical report, 2006.

[デジ 22] デジタル庁. ガバメントクラウド | デジタル庁. https://www.digital.go.jp/policies/gov_cloud/, 2022. accessed Dec. 30, 2022.

[阿里 22a] 阿里云. Alibaba Cloud CLI. https://www.alibabacloud.com/help/en/alibaba-cloud-cli, 2022. accessed Dec. 30, 2022.

[阿里 22b] 阿里云. Alibaba Cloud: Cloud Computing Services. https://www.alibabacloud.com/, 2022. accessed Dec. 30, 2022.

[阿里 22c] 阿里云. Alibaba Cloud Pricing — Flexible and Cost-effective. https://www.alibabacloud.com/pricing, 2022. accessed Dec. 30, 2022.

[阿里 22d] 阿里云. Auto Scaling: Automatically Adjusts Computing Resources - Alibaba Cloud. https://www.alibabacloud.com/product/auto-scaling, 2022. accessed Dec. 30, 2022.

[阿里 22e] 阿里云. CloudMonitor: Real Time Monitoring of Web Resources & Ap-

plications - Alibaba Cloud. https://www.alibabacloud.com/product/cloud-monitor, 2022. accessed Dec. 30, 2022.

[阿里 22f] 阿里云. Elastic Compute Service (ECS): Elastic & Secure Cloud Servers - Alibaba Cloud. https://www.alibabacloud.com/product/ecs, 2022. accessed Dec. 30, 2022.

[阿里 22g] 阿里云. Function Compute: Fully Hosted and Serverless Running Environment - Alibaba Cloud. https://www.alibabacloud.com/product/function-compute, 2022. accessed Dec. 30, 2022.

[阿里 22h] 阿里云. mPaaS: Mobile Platform as a Service - Alibaba Cloud. https://www.alibabacloud.com/product/mpaas, 2022. accessed Dec. 30, 2022.

[阿里 22i] 阿里云. Regions and zones. https://www.alibabacloud.com/help/en/basics-for-beginners/latest/regions-and-zones, 2022. accessed Dec. 30, 2022.

[阿里 22j] 阿里云. Resource Orchestration Service: Simplify Computing Resources O&M - Alibaba Cloud. https://www.alibabacloud.com/product/ros, 2022. accessed Dec. 30, 2022.

[阿里 22k] 阿里云. The Simple Application Server of Replacing Vps-Alibaba Cloud. https://www.alibabacloud.com/product/swas, 2022. accessed Dec. 30, 2022.

[阿里 22l] 阿里云. Use the Alibaba Cloud Management Console. https://www.alibabacloud.com/help/en/basics-for-beginners/latest/use-the-alibaba-cloud-management-console, 2022. accessed Dec. 30, 2022.

関連業績リスト

- 青嶋智久, 吉田健一, “クラウド利用における設計を考慮したコストモデル,” 研究報告インターネットと運用技術 (IOT), vol. 2018-IOT-40, no. 47, pp. 1-6, Feb. 2018.
- T. Aoshima and K. Yoshida, “Pre-Design Stage Cost Estimation for Cloud Services,” in 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), Jul. 2020, pp. 6-66. doi: 10.1109/COMPSAC48688.2020.00018.
- T. Aoshima and K. Yoshida, “Pre-Design Stage Cost Estimation for Cloud Services,” IEEJ Transactions on Electronics, Information and Systems, vol. 142, no. 6, pp. 679–688, 2022, doi: 10.1541/ieejeiss.142.679.
- T. Aoshima and K. Yoshida, “Cost estimates for modern e-business systems,” in 26th International Conference on Knowledge Based and Intelligent information and Engineering Systems (KES 2022), Sep. 2022