修士論文

多様なロードバランサーを提供可能な LBaaS(Load Balancer as a Service) の検討

2016 年 2 月 4 日

指導教員　相田 仁 教授

工学系研究科電気系工学専攻

37-146507　ダワージャルガル オリギル

目次

図目次

表目次

第1章

序論

1.1 はじめに

近年のクラウドコンピューティングの急速的な発展はICTに多大な恩恵をもたらしている。特に仮想化技術を用いてサーバやネットワークインフラをクラウドサービスとして提供しているIaaSの影響は多大なものである。大規模な分散システムを用いたサービスアプリケーションやコンピューティングシステムのインフラの構築がIaaSを使うことで従来よりずっと簡単になった。

多数のサーバから成る分散アプリケーションにおいてロードバランサーの役割は大きい。アプリケーションの負荷を効率的かつ効果的に分散することでシステムのパフォーマンスは向上する。IaaSのクラウドサービスではサーバなどの計算資源の他、ネットワークサービスの一つとしてLoad Balancer as a Service(LBaaS)が提供されている。他のIaaS同様にユーザーはロードバランサーをGUIやAPIを通じて手軽に構築することができる。このようにサーバ構築やロードバランサー構築がクラウドによって従来と比べ手軽にできるようになり、ユーザーはアプリケーションの開発にもっとリソースを裂けるようになった。

LBaaSについてのRahman[1]による論文ではLBaaSの現状と重要な機能について述べ、LBaaSの今後の課題について書かれている。その中のロードバランサー手法の選択性について本研究では注目し研究課題とした。世界的に様々なアプリケーションが作られる中で、多種多様なアプリケーションの特徴にあったロードバランサーを実装し負荷分散を行うことは重要である。LBaaSにおいてこのいろんな長所や短所をもったロードバランサー手法を提供し、ユーザーはその中から適切なロードバランサーを選択できることが求められている。しかし、現状の海外、国内のパブリック及びプライベートクラウドのLBaaSではRound RobinとLeast Connections手法などの限られた手法のみが提供されている。アプリケーションの特徴によってはこれらの手法では負荷分散のパフォーマンスが低い場合があり得る。

本論文では既存研究としてロードバランサー手法をいくつか取り上げる。そして、どうやったら現在のLBaaSでこれらの手法を提供できるかを検討する。多様なロードバランサーを提供可能なLBaaSを考える上で既存研究のロードバランサーがどのように実装され

ているかを大きく分類し、それぞれの種類のロードバランサーを提供するための機構を現状の LBaaS のアーキテクチャを踏まえて考察を行った。

既存研究では効率的、効果的なロードバランサーを実現するためにはリクエストによるアプリケーションへの負荷を知る必要があった。しかしリクエストから負荷量を知ることは難しいため、アプリケーションを開発したユーザー側にその情報を入力してもらう必要がある。LBaaS においてリクエストの負荷量の情報として負荷テーブル機構をどう提供するかについて様々な観点から考察した。ユーザーに負荷テーブルどのようにして入力してもらうかについて議論した。また負荷テーブルの正確性の問題についてクラウド基盤構築ソフトウェアの Openstack を用いて実環境に近い状態をシミュレーションによって検証した。

本研究では LBaaS において多様なロードバランサーを提供するための仕組みを検討し、シミュレーションによって負荷テーブルを用いたロードバランサー手法の有用性と問題点についての考察を行った。本研究では LBaaS におけるロードバランサー手法の課題解決のための検討を行い、一部の事象についてはシミュレーションや仕組みの提案をすることで深く議論した。今後の課題として他の事象についてもさらなる検討を行い LBaaS のロードバランサー手法の選択性の問題解決に取り組む必要がある。

1.2　本論文の構成

本論文は 5 つの章で構成されている。第 1 章では序論を述べた。第 2 章では本研究の背景としてロードバランサーについての説明や LBaaS(Load Balancer as a Service) について、関連研究について説明する。第 3 章では本研究の目的である LBaaS の課題を定義し、その解決法について検討する。第 4 章では第 3 章で検討した項目の中の一つを Openstack による実験によって検証し考察する。第 5 章では本研究のまとめと今後の課題について述べる。

第2章

研究の背景

本章では研究の背景として、ロードバランサーについての基礎的な知識と既存の研究について説明する。またクラウドコンピューティングにおいてロードバランサーを扱うための仕組みである LBaaS(Load Balancer as a Service) の説明とその仕組についてを述べる。

2.1 ロードバランサーについて

ロードバランサーとはネットワーク技術の一つで、ネットワークに掛かる負荷を複数のサーバ、リソースに分散し、システムの効率や性能、応答性を向上させる方法である。一般にロードバランサーと呼ばれるシステムを用いて負荷分散を行う。

大きなアプリケーションやシステムではサーバファームと呼ばれる複数のサーバ群の上でシステムを構築する。外部からの要求をこのサーバファームに分散して割り振るとき、各サーバに均等に負荷が割り振られるようにしなければならない。サーバファームの一つのサーバにリクエストが集中すればシステムの応答性が損なわれる。一般には同等な性能を持つサーバ群に均等に負荷分散すれば十分だが、サーバ能力にばらつきがある場合やサーバ間の通信速度などが異なる場合の複雑なシステムでは高度な機能を持つロードバランサが必要となる。

システムをサーバファームで構成する事によって、高い処理能力と応答性を得ると同時に、ひとつのサーバが故障してもサービスを停止しない可用性が得られる。また、システムの負荷が増え、もっとリソースが必要なときにサーバを追加してシステムを拡張することができ、システムを拡張することができる。ロードバランサを用いることで、クライアントから単一のアドレスでアクセスでき、システム側ではそれらのリクエストを負荷分散できる。

ロードバランサーの負荷分散方式は大きく 2 つに分類される [18]

1. 静的負荷分散 (Static Load Balancing)
2. 動的負荷分散 (Dynamic Load Balancing)

この 2 つの大きな違いはリクエストを分散させるサーバを選ぶとき静的分散はあらかじめ決められたルールに従ってサーバを選ぶのに対して、動的負荷分散はリクエストごとに

最適なサーバーを探しだし振り分ける。

i)　静的負荷分散

静的負荷分散はあらかじめ決めた順序、方法でサーバにリクエストを振り分ける。静的分散方式はサーバファームの性能が一定なとき、システムがリクエストの処理にかかる時間が一定のときに効果的である。各サーバに均等な負荷が分散されるため静的分散方式が適している。

静的分散方式は単純な方式なため実装が容易である。一般的な静的分散方式としてラウンドロビン方式がある。ラウンドロビン方式では外部からのリクエストをサーバファームの一つ一つのサーバに順番に割り振る。(図 2.1) リクエストごとの処理時間が一定である場合、各サーバにかかる負荷は平均的に一緒になる。従来のラウンドロビンの実装法として代表的なのが DNS ラウンドロビンという形の実装される。システムや Web ページの DNS 情報に複数の IP を登録して置き、DNS サーバが外部からのリクエストに対して順番に登録された IP を返していく。この他に異なる処理能力を持つサーバ群に負荷を分散させるために重み付きラウンドロビン (Weighted Round Robin, WRR) という手法がある。この手法では分散対象サーバの処理能力に応じてリクエストを分散比率を指定する。

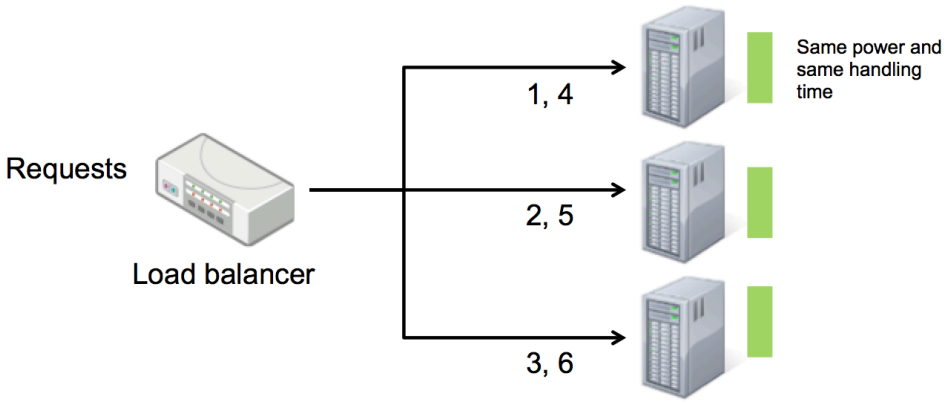


図 2.1: Round Robin loadbalancing method

逆にリクエストごとの処理時間が一定でない場合、例えばファイルのダウンロードを行うときやリクエストによって処理時間が変化するシステムで静的分散方式を使用するとサーバ負荷の偏りが大きくなってくる。このようなシステムには静的分散方式は適していない。(図 2.2)

このように静的分散方式はリクエストの処理時間に偏りがないシステムに向いているが、リクエストによって負荷が異なってくるようなより複雑なシステムだとうまく負荷分散が行われない。また分散先のサーバ情報はあらかじめ決められたものを所持しているため、サーバが何らかのエラーや故障で停止してしまったときの対処ができないなどの問題がある。そういった場合、動的負荷分散方式を考えなければならなくなってくる。

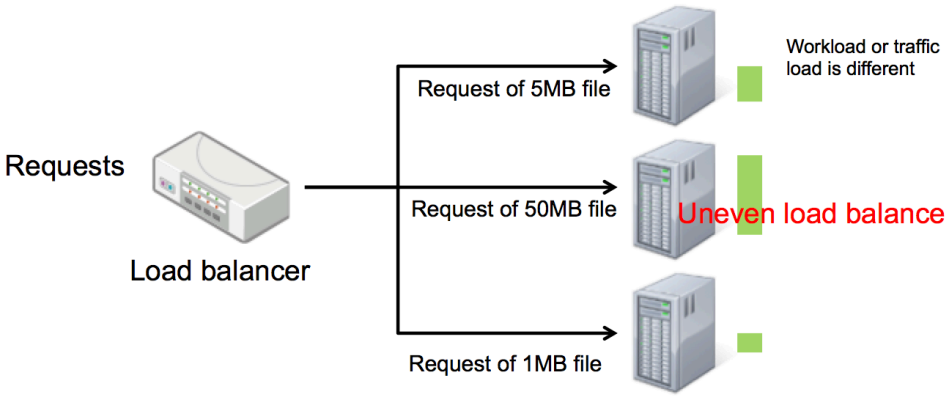


図 2.2: Static method does not work well when requests are not constant

ii)　動的分散方式

動的負荷分散はリクエストを振り分けるとき、リクエストが来た時に最適なサーバを選択する方式である。ロードバランサが常に各サーバの状態やサーバとの通信状態などを測定し、各サーバの状況を考慮して負荷を分散させる。動的負荷分散は静的分散方式が適していないシステムに向いている。リクエストごとに処理負荷が異なるようなシステムに効果的である。

動的負荷分散はシステム構成や観測できるサーバの状態などによって様々な方式が用いられる。システムによっては測定できる要素が異なってくる。また、どんなサービスを提供しているかによって適した状態を測定し、その結果を基に各サーバの負荷が均等になるように負荷分散を行う必要がある。動的負荷分散では各サーバのコネクション数や実際に利用しているクライアント数、サーバごとのデータ通信量、サーバの応答速度、サーバの CPU 使用率、ディスク IO 状況など様々な情報を用いる。アプリケーションによって適した情報を測定し負荷分散を行う。(図 2.3)

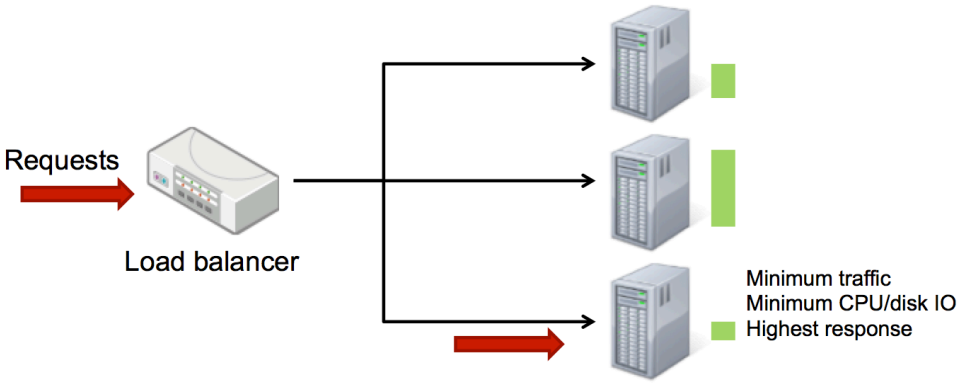


図 2.3: Dynamic loadbalancing method

リクエストによってサーバの CPU やディスク IO に負荷がかかるようなアプリケーションが走るサーバではシステム負荷をもとにサーバファームの中で一番負荷が低いサーバに割り振る最小サーバ負荷分散方式を用いる。

リクエスト毎で通信するデータの量が大きく異なる、FTP やメールサーバやストリーミングサーバなどではサーバの負荷はデータの転送量に依存する。この場合はデータ通信量が均一になるように最小データ通信量分散方式を用いたり、もしくは TCP/UDP コネクション数を測定し最小コネクション数分散を行う。

アプリケーションによって素早いレスポンスが求められるとき、Web サービスでユーザーにレスポンス性の高いサービスを提供したときはサーバの中から最もリクエストのレスポンスが早かったところを選択する最小レスポンス方式を用いる。レスポンスはサーバの負荷状況やトラフィック量による。

このように動的負荷分散ではシステムの特性を検証し、サービスに求められるポイントに合った分散方式を選択する。それぞれの方式で測定するサーバの状態は異なるため、それに適したシステムを組む必要がある。上記で述べた手法は一般的な動的分散方式の考え方で実際には様々なアルゴリズムや実装法で実装される。

上記の手法の中で最小コネクション数分散、つまりリーストコネクション (Least Connection) 手法はクラウドサービスで一般的に提供されている。リーストコネクション手法はデータの転送量にに限らずレスポンス性やサーバ負荷においてもパフォーマンスを発揮する。なぜならばおおよその場合は一番コネクション数が低いということはそのサーバが一番負荷が低いと考えられるからである。サーバ負荷やレスポンスを常に監視する必要がある最小サーバ負荷分散や最小レスポンス負荷分散に対してリーストコネクションはロードバランサーでリクエストを処理するときにコネクション数を監視するためサーバ監視やフィードバックのオーバーヘッドがなく、実装が比較的容易になっているためである。

2.2　クラウドコンピューティングにおける LBaaS

2.2.1　クラウドコンピューティング

クラウドとは文字通り Cloud=雲のような存在で、ネットワークの向こう側の雲の中にある伸び縮み可能なコンピューティングシステムである。ユーザーはネットワーク越しに雲の向こう側を意識することなくサービスを利用する。クラウドコンピューティングの公式な定義は存在しないが、一番知られている定義として米国の標準化団体 NIST(National Institute of Standard and Technology) によるものがある。NIST によるとクラウドコンピューティングは、どこでも、便利に、要求に応じてネットワーク経由により共有の計算資源 (ネットワーク、サーバー、ストレージ、アプリケーション、サービスなど) をサービスプロバイダーとのやり取り、もしくは最小限の管理コストで速やかに構成し、利用できる仕組みである。またクラウドコンピューティングが持つべき特徴として以下が挙げられている [2]。

On-demand self service　利用者からの要求にしたがって、必要となる計算資源を自動的に構築、設定し利用できること。

Broad network access　ネットワーク経由でパソコン、タブレット、携帯電話など様々な端末からサービスが利用できること。

Resource pooling　マルチテナント方式に基づき、複数の利用者に提供できるように計算資源は蓄えられ、物理的もしくは仮想的な資源を、利用者からの要求に応じて動的に割り当て、再割り当てできること。利用者から資源の存在する場所を意識することなく利用できるが、おおまかなレベル (国、地域、データセンター) での位置については特定することも可能な場合がある。

Rapid elasticity　利用者からの要求の変化に応じて、提供される資源量を柔軟に増やしたり減らしたりすることできる。利用者からは、使用できる資源量の上限があたかも存在しないかのように見えること。

Measured service　様々な計算資源の使用状況を自動的に監視でき、それに基づいて計算資源の利用を制御し、最適化できる。

クラウドサービスには 3 つの形態 SaaS、PaaS、IaaS が存在する。特に IaaS(Infrastructure as a Service) では、CPU やメモリやネットワークやストレージといったインフラを提供するサービスが仮想化技術の進歩により発展を遂げている。IaaS で仮想マシンを使用することによって伸び縮みできる、使いたい時に使いたい分だけ使うことができるスケーラビリティ性が構築できるようになったのである [3][4][5]。

クラウドサービスは大きく 3 つの形態をとる。IaaS(Infrastructure as a Service)、PaaS(Platform as a Service)、SaaS(Software as a Service) の 3 つである [2]。

SaaS(Software as a Service)　利用者が必要とするアプリケーションソフトウェアを、ブラウザやプログラムインターフェースによりアクセスできる形で提供する。CRM(顧客データベース管理ソフト) や ERP(企業経営管理ソフト) など事務アプリケーションなどをクラウドサービスにする。利用者はクラウド基盤の中身 (サーバー、OS、ストレージ、ネットワークなど) を意識することなく利用できる。

PaaS(Platform as a Service)　PaaS は利用者が作成したプログラムやソフトウェアの動作環境、フレームワークを提供するサービスである。例えば、Google App Engine は Python 環境や様々な API を提供している。利用者はクラウド基盤の詳細を意識する必要はないがアプリケーションの実行に必要な設定や制御を行うことができる。

IaaS(Infrastructure as a Service)　IaaS はサーバーなどのシステムインフラをサービスとして提供する。利用者はサーバー、ネットワーク、ストレージといった資源を利用する

ことができ、OS を含む任意のソフトウェアをインストールし、構築することができる。その代表的な例として Amazon Web Services があり、Amazon EC2(Elastic Computing Cloud) では仮想的なサーバー環境の時間貸し、S3(Simple Storage Service) では大容量ストレージを提供している。

2.2.2 LBaaS(Load Balancer as a Service) とは

LBaaS(Load Balancer as a Service) とはクラウドなどでロードバランサーをサービスとして利用するための仕組みのことである。従来、ロードバランサーは他のハードウェア、ソフトウェアと同様ユーザーが直接的に自分のアプリケーションに導入をしていた。クラウドコンピューティングの IaaS や PaaS などの発展により開発者はアプリケーションの開発に集中することができ、アプリケーションのインフラはクラウド上で仮想マシンを借りて組み立てることができるようになった。これらの仮想マシンに負荷を分散させるためのロードバランサーをクラウドベンダー側はユーザーにサービスとして使用できるよう LBaaS というサービスモデルを提供し始めたのである。ロードバランサー機能を他サービス同様にコンソール画面もしくは API コマンドなどを通じてセットアップでき、使った分だけ料金が発生するものとなっている。[1]

一般的な LBaaS のサービスモデルは次のようにユーザーにロードバランサーを提供している。

1. ユーザーアカウントの作成:
   ユーザーは LBaaS を使うベンダーのユーザーアカウントを作成する。アカウント情報には支払い情報や、サービス規約の同意が求められる。
2. 自分の使いたい規模や料金体系のものを選択:
   料金体系は一般的には使った分だけ料金が発生するもので、ロードバランサーを使った時間で発生したり、ロードバランサーが処理したネットワークリソースの分だけ発生したりする。
3. インスタンスの設定:
   負荷を分散させる対象のインスタンス (仮想マシン) を設定する。使用するプロトコルやポートなどの設定。
4. ロードバランサーの設定:
   ラウンドロビンや Least Connection などのロードバランサー手法の選択や、ロードバランサーのネットワークアドレスなどの設定を行う。
5. ヘルスチェックの設定: 負荷分散対象のインスタンスのヘルスチェック方法を設定する。ロードバランサーはダウンしているサーバーにリクエストを振らないように各サーバーが稼働しているかヘルスチェックを行う。その際のヘルスチェック手法を選択、設定する。

2.2.3　LBaaS の重要な機能

Rahman による LBaaS について調査研究 [1] によるとクラウドにおける LBaaS には次の機能が重要である。LBaaS を提供する上で次の事項を深く考慮したサービスづくりを考えるべきである。

ロードバランサー手法の選択性　LBaaS にはロードバランサー手法が幾つか用意されていて、それらの中からユーザーが自分にあったものが選べる必要がある。ロードバランサー手法の選択性はクラウドを使う様々なユーザーの特殊なニーズに応えるのに重要である。

通信インタフェース　様々なアプリケーション実装に対応するためには LBaaS が多様な通信手段をサポートする必要がある。HTTP、HTTPS、SSL、TCP、UDP など様々なプロトコルにおいての負荷分散ができる必要がある。

使用量の計測　IaaS などの他サービス同様に LBaaS の使用量を定量的で正確に測りユーザーに伝える機構が必要である。ユーザーは使用量を用いてアプリケーションのスケールアウトといった操作の判断をする。

料金体系　ユーザーが LBaaS を使用することによる料金の監視や予算の管理ができる。使った分だけ料金が発生する料金体系はユーザーの初期投資やリスクマネジメントが管理しやすくなるためサービスとして重要である。ユーザーは様々なロードバランサーを試し、アプリケーションに最適な手法をチューニングできる。

ベンダーロックインのリスク低減　一つのサービスベンダーに LBaaS を用いた際に他のサービスへの移行が困難になってしまうリスクを低減させる必要がある。LBaaS ベンダーたちはなるべく共通化されたサービス体型を組み立てる必要がある。

サービス品質保証 (SLA)　LBaaS 提供者がサービス品質保証を定める必要がある。ロードバランサーのセキュリティや耐障害性の品質が保証される。

データセンターをまたがったロードバランサー　クラウドサービスとして複数のデータセンターをまたがったロードバランサーを提供できる仕組みづくりが大切である。複数のリージョンにインスタンスを立ててサービスを提供するときユーザーに一番近いデータセンターにリクエストを送ることでレスポンス性が向上する。

ヘルスチェック　負荷分散対象のサーバーのヘルスチェック機能はリクエストがダウンしたサーバーに振り分けられないようするためにある。

2.2.4　既存の LBaaS

世界、国内の主要クラウドサービスではロードバランサーが LBaaS もしくはクラウドのネットワークの一つとして提供されている。例えば米 Amazon のクラウドサービスである AWS(Amazon Web Services) では Elastic Load Balancing という名でロードバランサーが提供されている。Elastic Load Balancing では Amazon EC2 で立てた複数のインスタンスに負荷を分散できるようロードバランサーを設定できる。負荷が高くなってきたら自動で追加のインスタンスを立ててくれる Auto Scaling 機能や AWS の複数のデータセンターに対してユーザーに一番近いインスタンスにリクエストを振り分けるマルチテナント機能などを持っている。

次の表 2.1 と表 2.2 では海外、国内の主要なクラウドサービスで LBaaS もしくはロードバランサーがどのように提供されているかを比較した。海外のサービスとして Amazon Web Services(AWS)[23], Google Compute Engine[24], Microsoft Azure[25], Rackspace[26] の 4 つ、国内のクラウドサービスとしてニフティクラウド [27]、ヤフージャパンの提供する IDCF Cloud[28]、GMO クラウド [29]、NTT コミュニケーションズの提供する Cloudn[30] を対象に、各クラウドサービスに対して前述した Rahman による LBaaS の重要な機能についての調べを行った。

Rahman[1] の述べるロードバランサー手法の選択性についてはほとんどのクラウドサービスが基本的な静的分散方式と動的分散方式であるラウンドロビンとリーストコネクションを提供している。また一部のクラウドでは分散対象サーバの処理能力に応じてリクエストを分散比率を指定する重み付きラウンドロビン (WRR) や重み付きリーストコネクション (WLC) が使用可能となっている。Google Cloud Engine では Content-based Load Balancing[24] と呼ばれるリクエストのコンテンツによって処理するサーバを分ける方式が用意されている。この手法では図 2.4 のようにリクエストを画像ファイル、動画ファイル、HTML、PHP などのように大別してそれぞれを処理するサーバ群に振り分ける。このようにすることで動画のようなネットワークトラフィック量が大きいものを別のコンテンツサーバから読み込ませて、他のデータベースとやり取りするようなリクエストなどを別のサーバで処理することができ、性質の異なるそれぞれのリクエストが互いに影響しないようにできる。

負荷分散を行うレイヤーとしてトランスポート層の L4 で行うものとアプリケーション層の L7 行うものがある。L4 ではリクエストのトランスポート層の範囲で負荷分散するためラウンドロビンやリーストコネクションなどの限られた手法でしか負荷分散ができない。L7 ではアプリケーション層で負荷分散を行うためリクエストの内容によって負荷分散ができ、L4 で行うときよりより柔軟な負荷分散ができる。通常の Web アプリケーションで負荷分散を行うとき大事になってくるセッション保持機能もロードバランサーの大事な機能の一つである。アプリケーションによっては一人のユーザーのリクエストを同じサーバに送り続けなければならない。L4 ではセッション保持を行うことが難しくなってくる。表 2.2 ではニフティクラウドが L4 での負荷分散を行うためセッション保持にソース IP を用いている。しかしユーザーが無線 LAN などを乗り換えるなどして IP アドレスが変わった場合にこの手法はうまくいかない。これに対して L7 ではアプリケーション層の情報を用いることができるため Web アプリケーションの Cookie 情報を用いて柔軟にセッション保持を行

表 2.1: 海外主要クラウドで提供されている LBaaS の比較

	AWS	**Google Cloud Engine**	**MS Azure**	**Rackspace**
対応レイヤー	L4/L7	L4/L7		
アルゴリズム	ラウンドロビン リーストコネクション	ラウンドロビン リーストコネクション Content-based LB	ラウンドロビン	ラウンドロビン リーストコネクション WRR*, WLC**, ランダム
HA 機能			あり	あり
マルチリージョン	あり	あり	あり	
アクセス制御	あり			あり
対応プロトコル	HTTP, HTTPS, TCP, SSL	HTTP, HTTPS		HTTP, HTTPS, TCP, UDP
セッション固定	Cookie セッション		ソース IP ハッシュ	あり
ヘルスチェック	ICMP, TCP, HTTP, HTTPS, 5 秒毎	あり	ICMP, HTTP, 15 秒毎	HTTP
ログ機能	あり、S3 へ保存			あり
料金体系	時間割+データ量	時間割+データ量	リクエスト量	
オートスケール	あり	あり		

*WRR: Weighted Round Robin, 重み付きラウンドロビン

**WLC: Weighted Least Connection, 重み付きリーストコネクション

表 2.2: 国内主要クラウドで提供されている LBaaS の比較

	ニフティクラウド	IDCF Cloud (Yahoo JAPAN)	GMO クラウド	Cloudn (NTT)
対応レイヤー	L4		L4/L7	
アルゴリズム	ラウンドロビン リーストコネクション	ラウンドロビン リーストコネクション ソース IP ハッシング	ラウンドロビン リーストコネクション WRR*	リーストコネクション
HA 機能		あり (アクティブ/スタンバイ構成)	あり	
マルチリージョン			なし	
アクセス制御	IP フィルタ			
対応プロトコル	IPv4, IPv6		HTTP, HTTPS	HTTP, HTTPS
セッション固定	ソース IP			Cookie セッション
ヘルスチェック	ICMP, TCP, 5 秒毎	TCP, 2 秒毎	HTTP, 5 分毎	TCP, SSL, HTTP, HTTPS
ログ機能			L4 の場合できない、L7 にする必要がある	
料金体系	月額+データ従量価格	無料	初期費用+月額	月額

*WRR: Weighted Round Robin, 重み付きラウンドロビン

うことができる。また L7 ではリクエストの内容を知ることができるため、Google Cloud Engine のようにリクエストの種類によって負荷分散を行う Content-based Load Balancing 手法 [24] を提供できる。

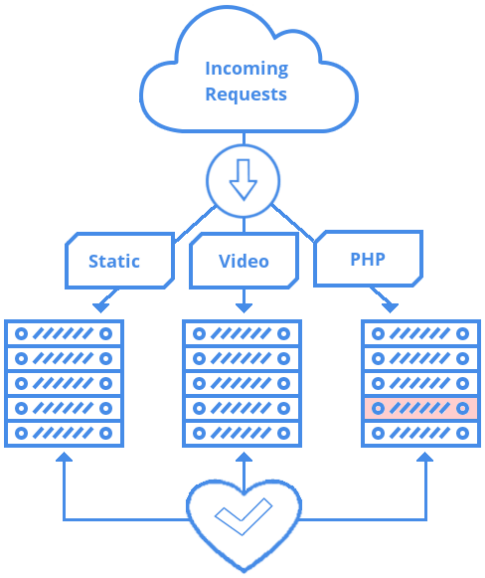


図 2.4: Google Cloud Engine: Content-based load balancing[24]

表 2.1, 2.2 にある HA 機能とは自動フェイルオーバー機能のことで HA は High Availability の略である。システムの冗長性を保持するための機能で、物理サーバなどに障害があってインスタンスが落ちてしまうときに自動で他の正常なサーバにインスタンスを起動しアプリケーションを停止させないようにする。この機能がロードバランサーに組み込まれているのは障害で停止したインスタンスに負荷分散を続けないようにする必要があり、フェイルオーバーで起動した新しいインスタンスをロードバランサーに設定しなければならないためである。ロードバランサーはヘルスチェック機能によりインスタンスを常に監視しているためすぐにフェイルオーバーを実行できる。

AWS や Google Cloud Engine や Microsoft Azure などのような世界各国にデータセンターを置くクラウドではデータセンター、リージョンをまたがったロードバランサーであるマルチリージョン機能が大事となる。世界規模でサービスを提供するときユーザーに一番近いデータセンターのインスタンスにリクエストを振り分けることがレスポンス性に影響する。

各クラウドにおいてヘルスチェック機能は様々な方法で行われている。インスタンスに ICMP で Ping を送って稼働状況を確認する方法や、特定の URL(index.html など) に HTTP/HTTPS リクエストを送ってアプリケーションが動いているかを確認するなどの方法がある。またヘルスチェックの間隔も各クラウドによって様々である。ヘルスチェック間隔は 2 秒、5 秒、15 秒などとなっていて、一部では 5 分間隔という比較的長い間隔でヘルスチェックを行うところもある。ヘルスチェックは対象のインスタンスが稼働しているかを

確認し、もしレスポンスがない場合は何らかの障害で停止していると考えられるため負荷分散対象から外す必要がある。そのためヘルスチェック間隔は短いのが望ましい。

料金体系において海外のクラウドサービスでは使った分だけ、使った時間と使用データ量に対して料金が発生するようになっている。これに対して国内のクラウドサービスは月額の料金体系が多く、長期間使用が前提となっている。これは国内のクラウドサービスを利用するユーザーがあらかじめ設計されたシステムを構築し運用するため短期間で運用体制が変わらないためだと思われる。しかし、クラウドを利用する様々なユーザーのニーズに応えるためには海外のサービスのように柔軟な料金体系が必要だと懐われる。

AWS と Google Cloud Engine ではオートスケール機能が備わっている。この機能はアプリケーション全体の負荷状況をモニタリングして、自動でインスタンスを立ち上げ下げするものである。

上記のようにクラウドサービス側から提供されているロードバランサーの他にサードパーティ社が提供しているロードバランサーがある。KEMP Technologies[31] や F5 Networks[32] などのようにハードウェア/ソフトウェアとしてロードバランサーを提供している業者がある。負荷分散機能を持った専用のハードウェアや仮想ルーターなどを用いてソフトウェアロードバランサーを提供している。

2.3　Openstack について

2.3.1　Openstack とは

Openstack とはクラウドサービスを提供するためのソフトウェアのことで国内、海外の様々なパブリック、プライベートクラウド構築に使われている。Openstack はオープンソースソフトウェアで IBM や Intel や Rackspace など国内では富士通、日立、NEC など 200 社以上の企業がこのオープンソースソフトウェアに貢献していて、自社のパブリック及びプライベートクラウドに Openstack を導入している。[20]Openstack は 2010 年に Rackspace Hosting と NASA によって始められたクラウド基盤ソフトウェアである。Openstack はクラウドサービスを提供するためのソフトウェアでデータセンター内のサーバやネットワーク管理、コンピューティング資源の管理、ユーザーの認証機能、料金体系機能、API、GUI の提供などすべての機能がそろったものである。開発言語は基本的に Python で作られていて Linux 系 OS 上で動く。2010 年 10 月 21 日に最初のリリースである Austin から Bexar, Cactus, Diablo とバージョンアップしていき 2015 年 10 月 16 日に 12 番目の Liberty がリリースされた。クラウド基盤構築ソフトウェアとして同時期に Eucalyptus や Cloudstack や OpenNebula などのソフトウェアが開発されていたが、Openstack のコミュニティが一番発展していて今ではほとんどのパブリック及びプライベートクラウドで Openstack が使われている。

クラウドを構築するときデータセンター上でたくさんのサーバ群のネットワーク管理、仮想マシンの作成、イメージ管理、ハードウェアリソース管理、ユーザーの認証機能、API、GUI など様々な機能が必要である。これらを全部提供するために Openstack はそれぞれの

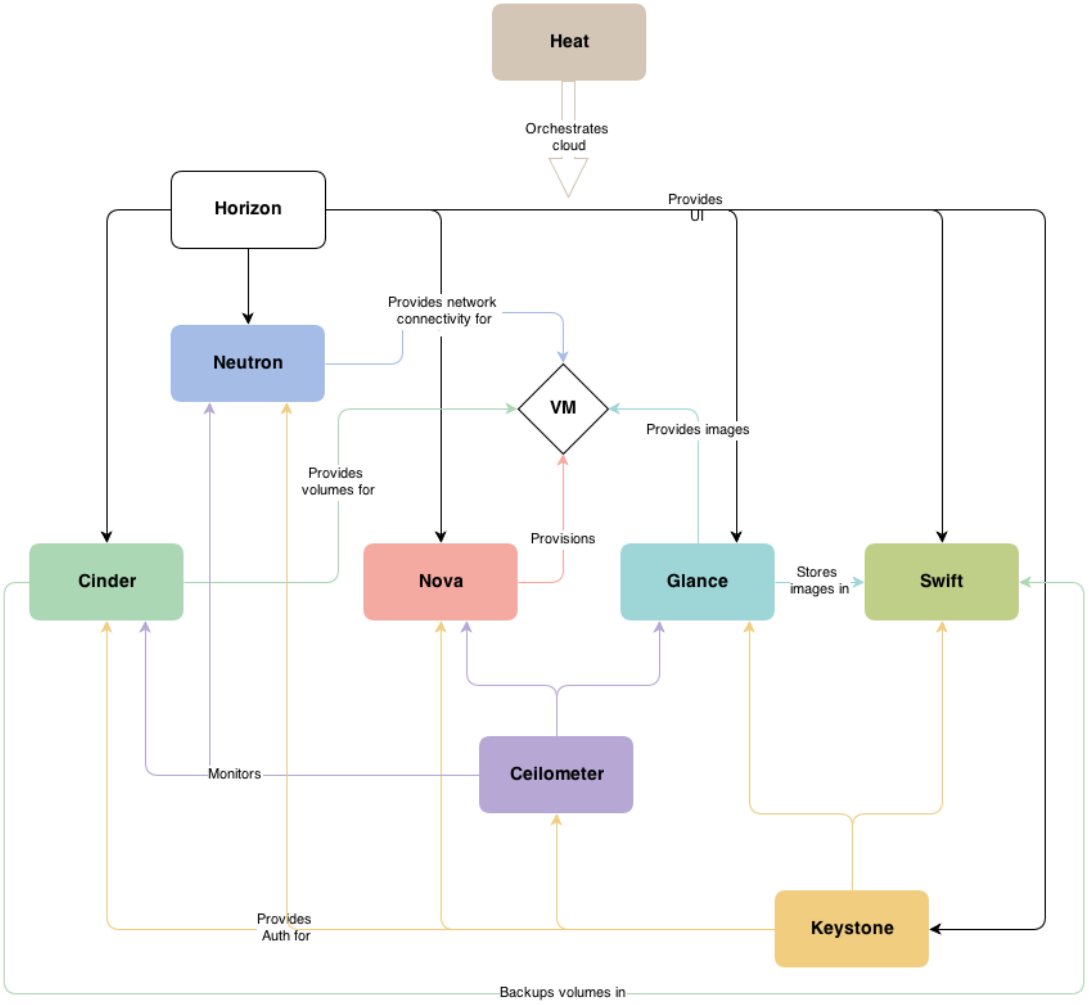


図 2.5: Openstack Architecture

機能をコンポーネント化して分けている。こうすることで大規模なクラウド構築においても必要なコンポーネントを必要な台数のサーバにインストールすることができスケーラビリティに優れている。Openstack のコンポーネントアーキテクチャは図 2.5 のようになっている。それぞれのコンポーネントが役割を持っていて互いに内部の API コールに必要なコマンドを呼び出して稼働している。例えばユーザーが Horizon から提供される GUI の Web 画面でログインし、そのログイン情報から Keystone という認証管理サービスがトークンを発行する。続けて仮想マシンを作成したときコンピュートサービスである Nova が仮想マシンイメージサービスの Glance を呼び出して仮想マシンテンプレートイメージを起動する。起動した仮想マシンに Neutron サービスがネットワーク設定をし、プライベート IP やグローバル IP などを割り振る。もしクラウドに計算資源がもっと必要になってきた時はコンピュートサービスである Nova がインストールされたサーバをもっと追加すれば CPU、メモリ、ストレージといった計算資源をスケールアウトさせることができる。

表 2.3: Rackspace によるロードバランサー手法の利用率

Algorithm	**Rackspace**
LEAST_CONNECTIONS	58.86%
ROUND_ROBIN	17.00%
RANDOM	13.88%
WEIGHTED_LEAST_CONNECTIONS	5.88%
WEIGHTED_ROUND_ROBIN	4.38%
MYSQL	3.94%

Openstack の各コンポーネントの役割について次の表 2.4 にまとめた。この中でコンピュートサービスの Nova とネットワークマネージャーの Neutron が仮想化技術を用いて仮想計算資源を提供してくれる。本研究の対象であるロードバランサーをサービスとして提供する LBaaS はネットワークサービスの Neutron に組み込まれている。ユーザーは GUI もしくは API コールによって Neutron でロードバランサーを設定することができる。海外主要クラウドサービスでの LBaaS(表 2.1) にあった Rackspace のクラウドサービスのなかではこの Openstack が稼働している。そのため Openstack の LBaaS で使用可能なロードバランサー手法はラウンドロビン、リーストコネクション、重み付きラウンドロビン、重み付きリーストコネクション、ランダムなどがある。Openstack の公式ドキュメントによると Rackspace では各ロードバランサー手法の利用率は表 2.3 のようになっている。リーストコネクション手法が約 59%で一番使用されている。Neutron は HAProxy[33] と呼ばれるオープンソースロードバランサーソフトウェアを用いている。HAProxy は 2001 年にリリースされており、高負荷に耐えられる信頼性の高いロードバランサーである。Neutron はこの HAProxy を LBaaS として使えるように API や設定項目を用意している。

表 2.4: Openstack の各コンポーネント説明

コンポーネント機能	サービス名	説明
Dashboard	Horizon	ダッシュボード。ユーザーに GUI 画面提供する。ユーザーは Web 画面からインスタンスを作成したり、ネットワークの設定などクラウドを使用することができる。
Compute	Nova	コンピュートサービス。仮想マシンを稼働させるサービス、Nova がインストールされたコンピュートノードを管理していてインスタンスの稼働や停止などの管理をする。KVM などの仮想化技術を用いてインスタンスを立てる。
Networking	Neutron	ネットワークサービス。クラウド内のネットワーク管理機能を担っている。仮想ネットワークを作成しインスタンスにネットワークを割り当てる。仮想ブリッジ、スイッチ、LAN、ルーター、IP アドレス、VM のネットワークを構築、LBaaS など様々なネットワーク管理する機能が組み込まれている。
Object Storage	Swift	オブジェクトストレージ。AWS の S3 のような機能でファイルなどを API を用いてオブジェクトとして管理することができる。
Block Storage	Cinder	ブロックストレージ。仮想マシンのブロックストレージを追加することができる。LVM などの機能によって仮想ストレージ管理ができる。
Identity service	Keystone	認証サービス。ユーザーの認証を行い、トークンを発行する。このトークンを用いて書くサービスに API コールを行い、サービスはそのトークンを再び Keystone に問い合わせることでユーザー認証機能を提供している。
Image service	Glance	イメージサービス。仮想マシンのイメージを管理する。ユーザーがインスタンスを起動するテンプレートイメージの管理や作成されたスナップショットなどを管理している。
Telemetry	Ceilometer	クラウドの使用量などを測るサービスで料金や監視のための機能である。
Orchestration	Heat	オーケストレーション。オーケストラの指揮者の意味で複雑な仮想マシンとネットワーク構成をスクリプトから自動で構築してくれるものである。

2.3.2　Openstack の LBaaS の仕組み

多様なロードバランサーを提供できる LBaaS を考える前に既存の LBaaS の仕組みとして Openstack のロードバランサーがどのように提供されているのかを考える。Openstack はクラウドサービスを提供する上で重要な機能をコンポーネント化し、各コンポーネントがそれぞれのプロセスとして走っており、互いに API コールを用いてつながっている。ロードバランサーは個の中のネットワークを管理するサービスである Neutron の中に組み込まれている。

Openstack でのロードバランサー提供サービスのアーキテクチャは図 2.6,2.7 のようになっている。クラウドユーザーが GUI を提供する Dashboard サービスもしくは自分自身で Openstack に直接アクセスできるクライアントアプリケーションを通して LBaaS と API 通信を行い、必要な設定事項や使用状況などを取得する。中央のロードバランサーサービスは Openstack 内の他のサービスと直接 API コールを通して認証機能やネットワーク機能にアクセスしている。そして HAproxy やその他のロードバランサーとは内臓の Driver を用いて通信を行い制御している。この Driver はそれぞれ異なる仕様や仕組みをもつロードバランサーとの橋渡し役となるドライバのことで、このドライバを用いてロードバランサープロセスと通信ができ、設定や制御ができるようになる。

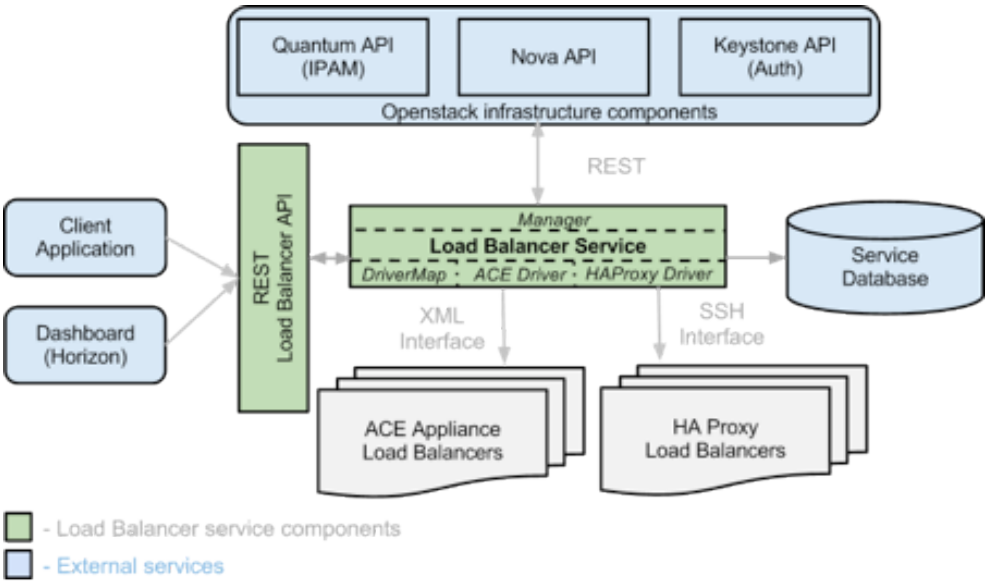


図 2.6: Openstack の LBaaS アーキテクチャ [21]

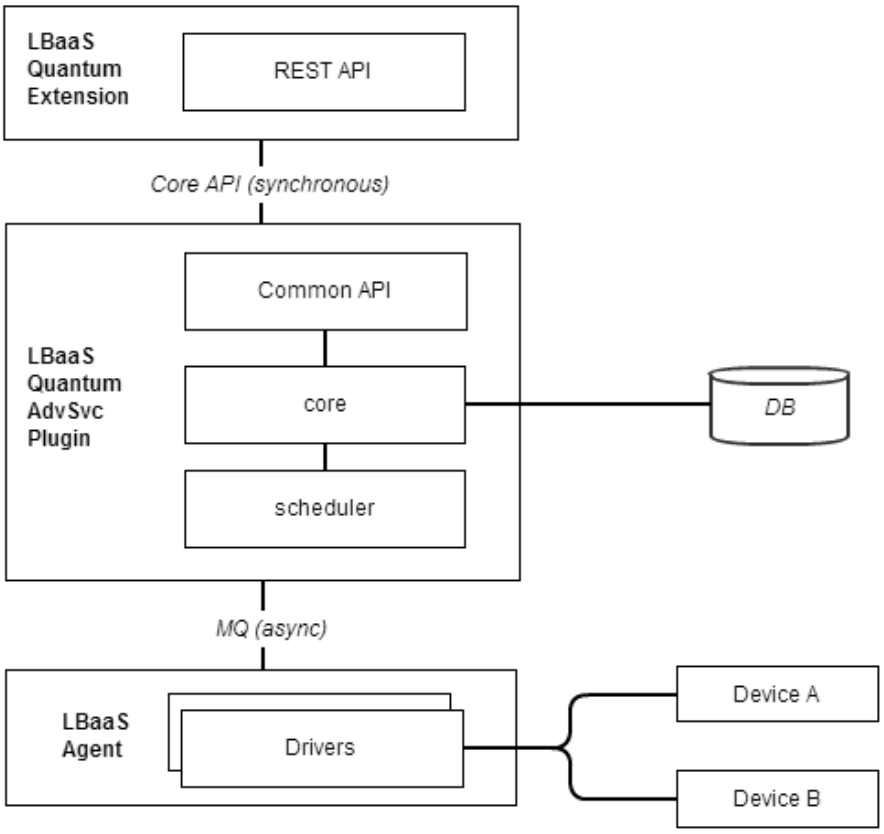


図 2.7: Openstack の LBaaS アーキテクチャ [21]

2.4　関連研究

これまでにクラウドコンピューティングについてとロードバランサーの仕組みについて、またクラウドコンピューティングにおいてロードバランサーをサービスとして提供する LBaaS について説明した。前述したように既存の LBaaS ではロードバランサー手法はラウンドロビンやリーストコネクションなど限られたものとなっている。これらは単純で実装が容易なものなのに対して、ミツバチコロニー最適化アルゴリズム [12][13] など複雑な実装法を持つロードバランサーが提案されている。この章では研究段階で提案されている様々な特徴を持つロードバランサーからいくつかを紹介する。今回はクラウドコンピューティングにおけるロードバランサーについての文献 [8], [7], [11] などで取り上げられている手法からいくつか紹介する。

2.4.1　Honeybee Foraging Behaviour

群知能を用いた最適化アルゴリズムをロードバランサーとしてミツバチコロニー最適化アルゴリズムがある [12][13]。このアルゴリズムはミツバチの採餌行動をモデル化した最適化アルゴリズムである。

ミツバチは採餌するときまず探索バチ (Scout Bee) が食料源を探す。そして食料源を発見すると巣に戻って食料の量と質、方向や距離を伝えるダンス (Waggle Dance) を行う。このダンスを元に適切な数の派遣バチ (Forager Bee) が食料を集めに飛んで行く。探索バチのダンスによって食料源に派遣されるハチの数が決まる。ミツバチ集団はこうして探索と派遣収集を役割分担しており、探索バチの適切なダンスによって最適な食料収集が行われている。

このミツバチの習性が探索アルゴリズムとして使われており、ロードバランサーとして考えるときサービスを走らせるマシンとサービスにアクセスしたリクエストキューを考える。このときサーバ群は一定の確率で探索バチと派遣バチの役目に別れ、探索バチはリクエストキューを探索しそれを Dance floor に変わるボードにその情報をポストする。探索バチの役割をもつサーバはボードを監視し、ポストされたリクエストの情報をもとに最適なリクエストをこなす。食料源の質と量と距離などはリクエストの処理時間やコストに置き換わりそれを元に適切な探索サーバがリクエストの処理に当たる。

ミツバチコロニーを用いたロードバランサーはシステムの多様性にパフォーマンスを発揮し、一定のスループットを保持できる (Fig.2.8)。しかしシステム全体の規模、サーバ群のサイズが大きくなると全体でのパフォーマンスは他のアルゴリズムと比べて劣る (Fig.**??**)。

2.4.2　Biased Random Sampling

この手法 [16] ではシステム全体をグラフとして捉える。各サーバはノードとしてマッピングされる。そしてノードはそれぞれのリソースの空きによってランダムに他のノードとエッジを作る。ノードは新しくリクエストを処理するとき他のノードとの繋がり、エッジ

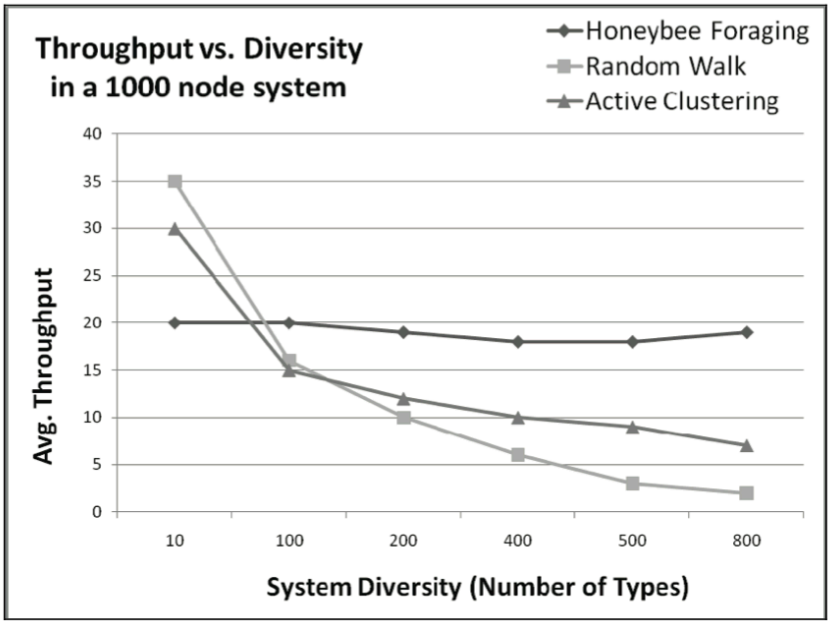


図 2.8: Throughput vs. Diversity in a fixed size system(adapted from [7])

を減らし自分のリソースが少ないことを伝える。そして処理を終えると再びエッジを作成する。つまりリソースに空きがあるサーバはエッジをたくさん持っていて、他のサーバから発見されやすい。システムにリクエストが来た時、Random Sampling が始まる。特定のノードからサンプリングは始まり、ランダムに他のノードを選んで進む。特定の w ステップでたどり着いたノードでリクエストを処理する。[16] では最適な w の値は $log(n)$(n はシステムのサイズ) であった。

Biased Random Sampling はシステムの多様性が大きくなるとスループットが落ちる (Fig.2.8)。しかし、システムのサイズが大きくなるとスループットは他のアルゴリズムと比べ上昇する。なのでこのアルゴリズムは大規模な、多様性の少ない処理を分散させるのに適している。

2.4.3 Stochastic Hill Climbing

この手法 [14] では高速な局所探索法として知られる山登り法 (Hill climbing) を用いたロードバランサーである。Stochastic hill climbing(SHC) は通常の Hill climbing 法と違い次のステップを選ぶときランダムに選ぶ。この手法ではシステムの仮想マシンの状態リストを監視していてリクエストが来た時に、リストからリソースが空いている仮想マシンを SHC 法で選び出す。

文献 [14] では基礎的なロードバランサー手法であるラウンドロビンと First Come First Serve 手法と比較シミュレーションを行っており、SHC 手法がほかよりレスポンス性が優れていた。

表 2.5: ロードバランサーの種類分け

	静的	動的	集中型	分散型	リクエストの処理時間	ノードの状態
Round Robin	○		○			
Least Connection	○		○			
Two Phase (olb + lbmm)	○		○		○	
Ant Colony[15]	○			○		
Honey bee		○		○	○	○
Biased Random sampling		○		○	○	○
Active Clustering[17]		○		○		○

2.4.4　ロードバランサーの種類分け

既存研究のロードバランサーをその仕組みによって大きく大別することができる。表 2.5 ではいくつかのロードバランサー手法を次の項目に当てはまるかを表した。前述したように静的分散方式と動的分散方式があり、研究されている手法のほとんどが動的分散方式である。単プロセスで負荷分散処理を行っているロードバランサーを集中型、分散対象の各ノードにロードバランサープロセスが稼働していて互いに通信しあいなが行うロードバランサーを分散型のロードバランサーと分別した。また負荷分散を行うときリクエストの処理時間やノードの負荷状態を監視しながら負荷分散を行うかどうかが様々なロードバランサーの共通点となる。[9][10]

第3章

LBaaSの課題検討

3.1 LBaaSの課題

ロードバランサーをサービスとして提供しているLBaaSでは様々な機能が求められている。多様なアプリケーションに対応した負荷分散手法を提供することはサービスとして重要な機能である。ロードバランサー手法の選択性が増すことによってユーザーが一長一短のあるロードバランサーの中から自分のアプリケーションに適した手法を選択することができる。しかし既存のクラウドサービスではLBaaSとして提供されているロードバランサー手法は限られている。ほとんどのクラウドサービスではラウンドロビンとリーストコネクションの手法のみが提供されている。これらの手法だと効率的な負荷分散が行えないケースがある。ユーザーのニーズによってはHoney Beeアルゴリズムのような多様性のあるシステム構築にパフォーマンスを発揮するロードバランサーが求められることが考えられる。ユーザーがこれらのロードバランサーを実装するには自分自身でシステム構築を行い、ロードバランサーを実装する他ない状態である。

クラウドサービスがLBaaSとして既存研究にある手法を提供していない最大の理由はクラウドサービス提供者とクラウドを用いるユーザーの間に隔たりがあるためである。クラウドを計算資源のみを提供していて、ユーザーはそれを使って自分のアプリケーションを動かしている。クラウド提供者側はユーザーのアプリケーションの中身や性質については知ることができない。またクラウドユーザー側はクラウド上での仮想資源を借りているため、その仮想資源より低レイヤーのネットワークなどの深い部分に鑑賞する権限がない。例えばクラウドユーザーが自らラウンドロビンロードバランサーを実装しようとするときアプリケーションが走るサーバの他にロードバランサーが走るインスタンスを別に立ててその上でラウンドロビンロードバランサーを動かす他ない。ラウンドロビンロードバランサーを仮想マシンの上ではなく、クラウド基盤ソフトのプロセスとして運用ができない。

3.2 分散型ロードバランサーの提供法

クラウドではリーストコネクションやラウンドロビンなどの限られたロードバランサー手法が提供されている。一方でクラウドのために研究されたロードバランサー手法が存在

する。それらの手法を実際に LBaaS として提供するための手法を検討する。第 2.4.4 章では既存研究のロードバランサーをその仕組みによって種類分けを行った。この種類分けでは集中型か分散型か、リクエストの処理時間またノードの状態を考慮したかが大きな特徴だった。これらの特徴のロードバランサーを提供できる LBaaS の仕組みが必要である。

既存研究のロードバランサーを大きく集中型と分散型に種類分け出来た。集中型ロードバランサーは Openstack の LBaaS アーキテクチャではすでに提供可能な状態である。Openstack の LBaaS アーキテクチャではロードバランサーとは専用のドライバを通じて設定や制御を行っている。Openstack に新しいドライバや必要な設定をインストールすることで現在提供されている HAproxy ロードバランサー以外のロードバランサーを提供することができる。さらに専用のルーターやスイッチを用いたハードウェアロードバランサーもすでにドライバやプラグインとして利用できるように開発されている。

分散型のロードバランサーは分散対象のインスタンスに専用のモジュールがくみこまて、インスタンスの監視や他のノードとの通信を行い負荷分散をしている。分散型のロードバランサーを提供するには各インスタンスにロードバランサープロセスを走らせる必要がある。このプロセスをユーザーの各インスタンスにどうやって実行させるかが課題である。既存の Openstack の構造を考慮すると分散型のロードバランサーのプロセスを実装するには次の 3 つの手法が考えられる。(図 3.1)

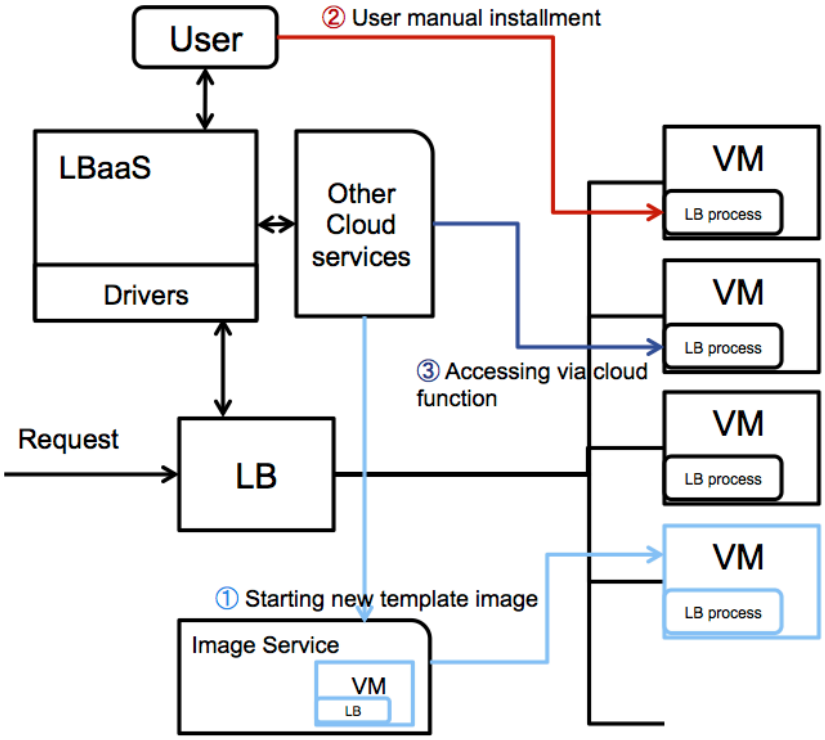


図 3.1: LBaaS において分散型ロードバランサーの提供法

1. すでにロードバランサーモジュールが組み込まれた仮想マシンイメージを新しく起動する。
2. 既存のインスタンスにユーザーがロードバランサーモジュールをインストールする。
3. 既存のインスタンスに Cloud-init などの仕組みによってクラウド側からモジュールをインストールする。

1 番目の手法ではユーザーに分散型ロードバランサーのモジュールがすでに組み込まれた仮想マシンテンプレートイメージを使用してもらう。クラウド側からあらかじめテンプレートイメージを作成してそれらをユーザーは起動できるようにする。ユーザーは起動したイメージを LBaaS に API や GUI を通じて設定を行うと、LBaaS はインスタンスに走るロードバランサープロセスと通じることができる。Openstack ではイメージ管理サービスである Cinder にロードバランサーモジュールが組み込まれたテンプレートイメージを登録しておいて、ユーザーはそのイメージを起動することになる。しかしこの手法ではユーザーはあらかじめ使用するロードバランサー手法を決めていて、選択したイメージを起動してからアプリケーションを実装する必要がある。そのためこの手法はすでに稼働しているアプリケーションでは手間がかかることになる。ユーザーは稼働中のインスタンスを停止させて、新しいインスタンスにアプリケーションを再稼働させなければならない。だが環境自動構築ツールなどを用いればアプリケーション環境の再構築は用意なものとなっており、この手法はそれらのツールとの連携が大事となってくる。

2 番目と 3 番目の手法はすでに稼働しているユーザーのアプリケーションにロードバランサープロセスを組み込む方式となる。一つはユーザーに手動で稼働中のインスタンスに必要なモジュールをインストールしてもらう方法と、もう一つはクラウドの機能を用いて LBaaS 側から自動でインストールする方法である。ユーザー側からロードバランサープロセスをインストールしてもらって、あとは LBaaS がドライバやユーザーの設定などを用いてロードバランサーを組み上げる。すでに稼働しているアプリケーションには LBaaS 側からプロセスをインストールするときはコンピュートサービスなどを用いる。Openstack の場合は LBaaS がコンピュートサービスである Nova を通じて仮想マシン内にロードバランサープロセスを組み込むことになる。この場合は Openstack 側と仮想マシンをつなぐ何かしらの通信手法があらかじめ用意されていることが前提となる。

3.3　リクエストの負荷を考慮したロードバランサーの提供法

3.3.1　リクエストの負荷テーブル用いた機構

ユーザーが実装するアプリケーションによってはリクエストによって掛かる負荷はリクエストの内容に大きく依存することが考えられる。またリクエストに負荷の大小の幅が広く、その頻度もバラバラな時は通常のロードバランサーで負荷分散を行うのは難しい。そのためリクエストの内容を考慮したロードバランサーが必要となってくる。このロードバランサーを使うためにはリクエストによってよって掛かる負荷の量を知る必要がある。しかし、ロードバランサーやクラウド側からはそれを測り知ることはとてもむずかしいもの

となってくる。アプリケーションを制作したユーザー側からあらかじめ入力してもらう必要がある。

実際のクラウドで提供されている LBaaS のなかでこの仕組一番近いものは Google Compute Engine の Content Based Load Balancing である。Content Based Load Balancing は第 2.2.4 章で紹介したように Web アプリケーションにおいてリクエストの URL からテキストリクエスト、画像ファイルのリクエスト、動画ファイルのリクエストなどに分けてそれぞれのサーバに負荷分散を行うものである。この手法の仕組みは図 3.2 のようになっている。ユーザーからの HTTP/HTTPS リクエストの URL 部分を見て、/static, /video, その他などのようにあらかじめユーザーに設定してもらった”URL Map”を用いて負荷分散を行っている。

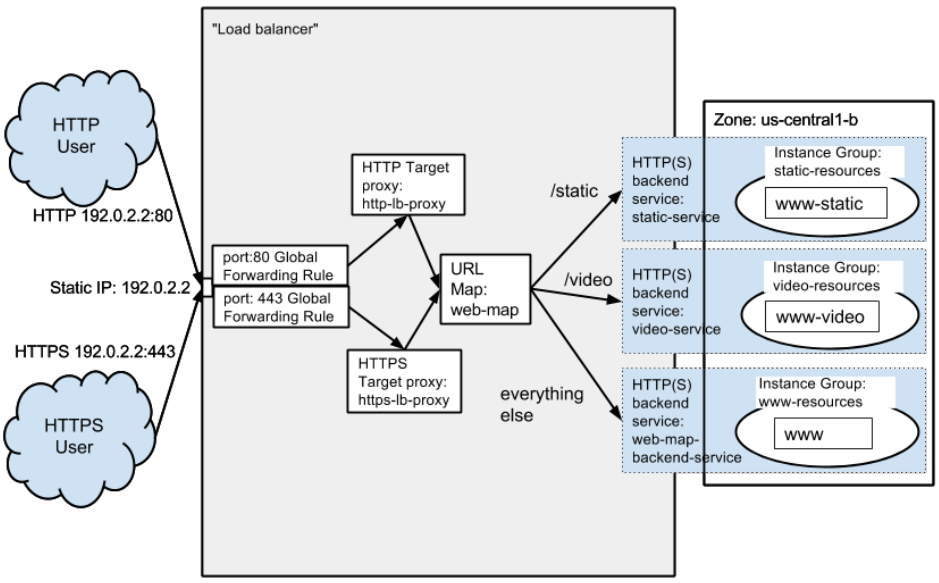


図 3.2: Google Compute Engine における Content Based Load Balancer の仕組み [24]

現在の LBaaS のアーキテクチャではリクエストの負荷量を管理する機構が必要である。図 3.3 のように LBaaS 内にリクエストの負荷テーブルを入力してもらい、その内容をドライバを通じて必要なロードバランサーに組み込んであげる機構を考察する。この負荷テーブルは LBaaS 内に実装されていて、外部から LBaaS の API を通じてアクセス可能なものとなる。負荷を考慮したロードバランサーはその負荷テーブルをもとにリクエストによる負荷を計算し負荷分散を行うことができる。LBaaS はロードバランサーの設定事項として負荷テーブルが必要なものだったとき対応するドライバによってロードバランサーに負荷テーブルを組み込む。

この負荷テーブルの中身がどんな内容であるべきかを考える。負荷テーブルはアプリケーションに来ると想定されるリクエストと、そのリクエストによって生じるサーバへの負荷の対応表の形をとる。負荷分散の対象が Web アプリケーションだった場合 Google のように URL によって掛かる負荷の一覧を作成することになる。ユーザーがリクエストの入力

として URL を正規表現などを用いて分別できるようにする。また URL 以外にも HTTP リクエストのヘッダの中身を見ることができるためその内容によって負荷が変わり得る場合においても、ユーザーが指定して入力できるように GUI や API と入力の規格を用意してあげることが必要とされる。

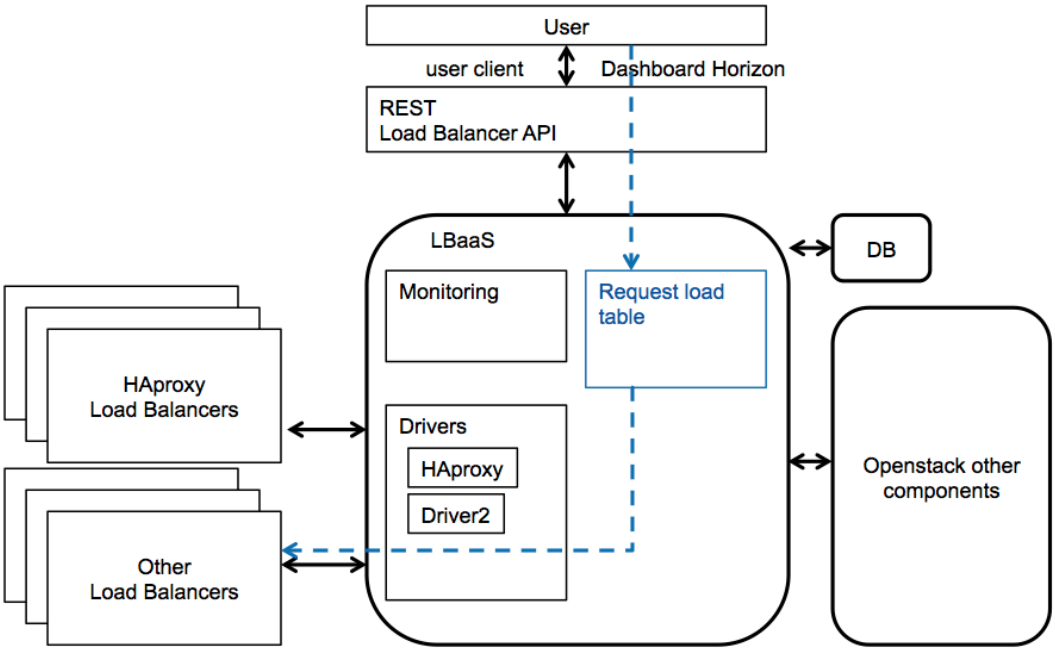


図 3.3: LBaaS においてリクエストの負荷テーブルを管理する機構

HTTP リクエスト以外のプロトコルの場合を考える。HTTP 以外のプロトコルで一般的に広く知られているプロトコルであれば上述したようにリクエストの中身で変わり得るパラメータを入力できる規格を作ることができる。しかし普及率が低いもしくはアプリケーションが独自のプロトコルを使用していて、ロードバランサーがリクエストの内容を判別することがむずかしいことが考えられる。また HTTP リクエストでも HTTP ヘッダの情報だけでは負荷の量が決まらない場合が考えられる。様々なアプリケーションに対応したリクエストの判別方法はないため、ユーザー側がロードバランサーに判別できるようなリクエストづくりを努めなければならない。ユーザーがアプリケーションを制作する際のデザインパターンにロードバランサーに適したリクエスト作りを考慮する必要がある。LBaaS 側も予めユーザーにどんなふうにリクエストの量を示せばいいかを規格化し、ユーザーに提示してあげる必要がある。例えば HTTP リクエストや TCP パケットの中に負荷の量を表すパラメータを挿入させる。ユーザーは LBaaS に負荷の量を示すパラメータを設定してあげると LBaaS はロードバランサーにその判別法をドライバを通じて個々のロードバランサー手法に伝えられる。また図 3.4 のようにリクエストの中身だけでは負荷量を測ることが難しい場合はアプリケーションを 2 段階に分け、1 段階目でアプリケーションがリクエス

トの負荷量を想定し、2 段階目の計算サーバに負荷テーブルで判別可能なリクエストに変えてリクエストを送るような設計法を実装することができる。

このようにリクエストの負荷量を考慮したロードバランサーを提供するにはユーザーに負荷テーブルを入力させてもらう必要がある。またアプリケーションによってリクエストは様々なものであるため LBaaS 側からリクエストの判別ができるように規格化された手法を提示する必要がある。またユーザー側はアプリケーションを設計するときリクエストの判別ができるよう努める必要がある。

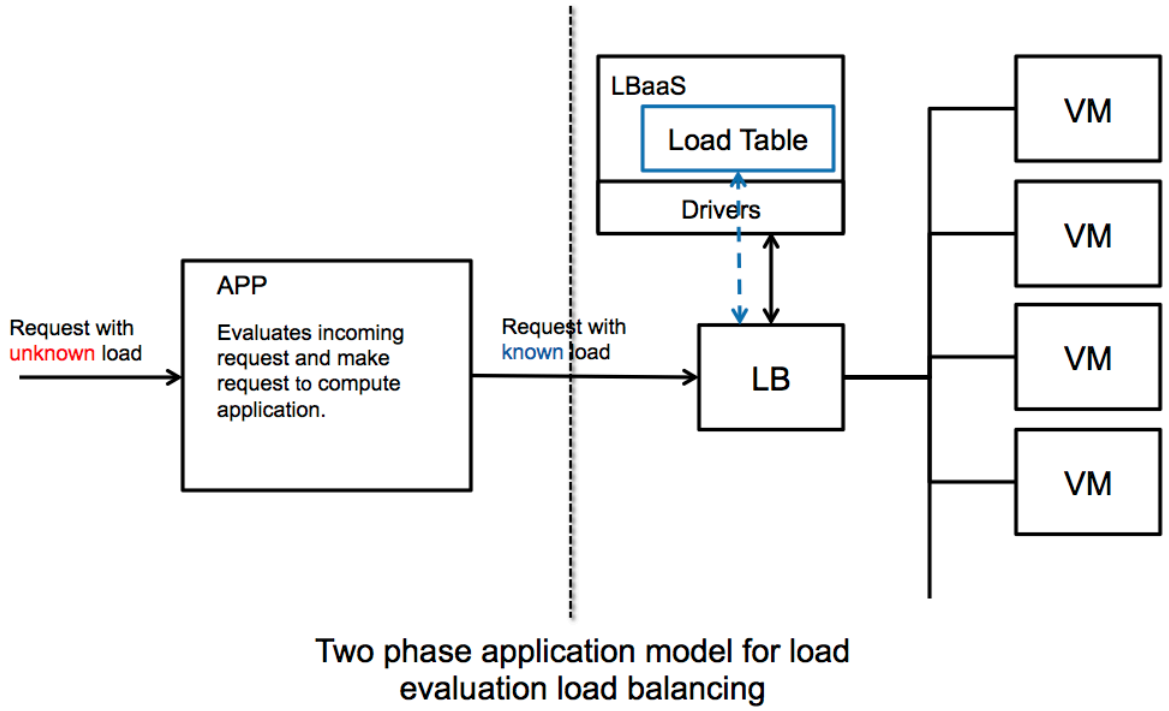


図 3.4: リクエストの負荷量を判別する機構をもつアプリケーション実装例

3.3.2　リクエストの負荷テーブルの修正機構について

リクエストの負荷を考慮したロードバランサーの提供法として LBaaS にリクエストの負荷テーブルを管理するモジュールを提案した。このモジュールは負荷テーブルをユーザーに入れてもらわなければならない。しかし現実にはアプリケーションには様々な形態があるためリクエストの内容だけでは負荷の量を判断することが難しい。またユーザーがリクエストによってもたらされる負荷の量を正確に測ることが難しい場合が考えられる。上述の図 3.4 のようにアプリケーションを 2 段階に分けることでユーザーがアプリケーション内の詳細なデータを用いてリクエストの負荷を推定することができる。しかしこの 2 段階デザインパターンはすべてのアプリケーションに適しているわけではないため、どうして

もリクエストの内容のみから負荷を推定する必要がある。またユーザーも負荷の量を測定して入力しなければいけない。

ユーザーから入力された負荷テーブルには誤差があることが考えられる。後述する負荷テーブルの検証実験では負荷テーブルの誤差や精度によるロードバランサーのパフォーマンスへの影響を測定している。ここではこの負荷テーブルの内容をあとから修正する機構について考える。負荷テーブルは LBaaS 内に一つのコンポーネントとして実装される。そのため負荷テーブルへの入力やロードバランサーへの出力などを API やドライバを通じてできるようになっている。そのため負荷テーブルの内容はいつでもアクセスすることができ、時によっては更新することができる。

負荷テーブルを修正するモジュールを図 3.5 のように LBaaS 内に組み込んでおく。負荷テーブル修正モジュールはロードバランサーなどから実際に負荷分散を行ったときのレスポンスタイムなどのログを収集する。収集したログから負荷テーブルによる誤差を修正するものである。ノードの状態を監視するロードバランサーからはノードの実際に負荷状態なども受け取ることができる。これらの入力から負荷テーブルを修正する機構を別途に作成すれば、負荷テーブルを非同期的に修正していくことが可能である。このような機構があればユーザーは最初の負荷テーブルの入力を大まかなものにでき、入力が容易になる。また修正された負荷テーブルからアプリケーションの監視や開発に役立つ情報を得られる。

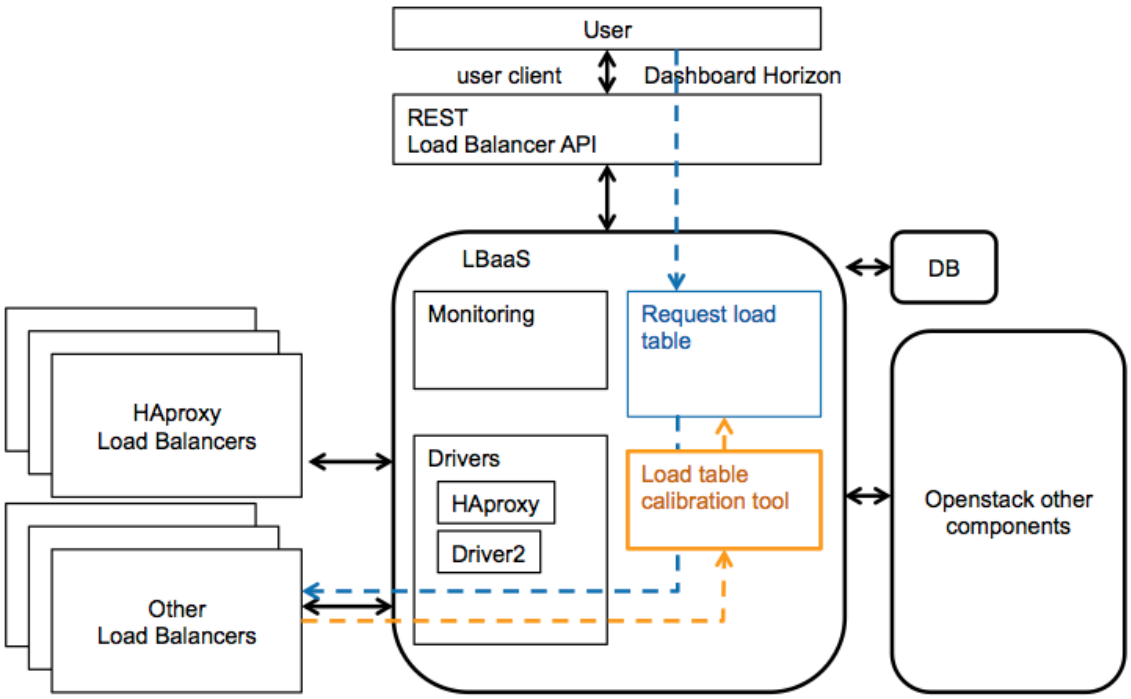


図 3.5: 負荷テーブルを修正する機構

第4章

提案手法の評価検証

4.1 実験環境

4.1.1 Openstack 環境

本研究では LBaaS の検証実験のため Openstack を用いてクラウド環境を用意した。Openstack はオープンソースのクラウド基盤構築ソフトウェアである。Openstack は実際に様々なパブリック及びプライベートクラウドで導入されている一番メジャーなオープンソースプロジェクトである。そのため Openstack を用いることで実際のクラウドに近い環境を構築することができる。

本研究で使用した Openstack はバージョン Juno(2014 年 10 月リリース) で、Xeon 8 コアサーバ 2 台で構成されている (図 4.1)。1 台のサーバは Openstack の all-in-one 構成となっていて Horizon, Nova, Neutron などのすべてのコンポーネントがインストールされており、ユーザーやデータ管理用のデータベースなど全部が入った構成のものとなっている。もう一台の方は compute-node となっていて Openstack の仮想マシンを管理するサービスである Nova のみがインストールされていて、Openstack の計算資源として稼働する。

サーバスペック:

- OS: CentOS 7
- CPU: Intel Xeon E3-1246v3
- Memory: 16GB (8GBx2) DDR3-1333
- Openstack Juno

4.1.2 シミュレーションの環境

ロードバランサーの検証のため Openstack 上で仮想マシンを立て、中に CPU 負荷を与えるアプリケーションを実装した。そのアプリケーションにポアソン到着によるリクエストを発生させロードバランサーのパフォーマンス計測を行った。

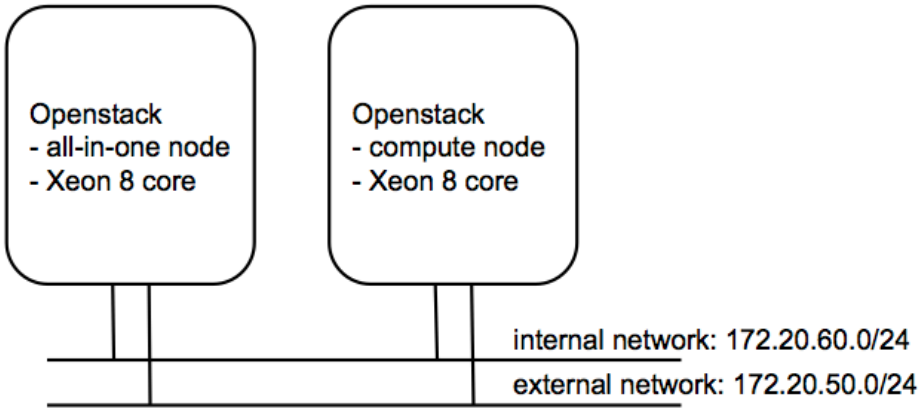


図 4.1: 実験で用いた Openstack 環境の構成

i)　アプリケーション構成

実験のためユーザーがクラウドで展開されるアプリケーションを作成した。アプリケーションは仮想マシン 10 台により稼働している。それぞれのインスタンスは最小構成の CPU 1 core, Memory 512MB で OS は Ubuntu 14.04 を使用している。

アプリケーションに来るリクエストの内容は CPU 負荷が掛かる複雑な計算処理だと想定した。実験用のアプリケーションを作成するとき CPU 負荷がかかる計算処理をするようにプログラミングされていて、リクエストには 1-100 までの値を送るようにした。1 のリクエストではおおよそ $11ms$ の CPU 負荷がかかりその結果が帰ってくる。2 のリクエストではその 2 倍、3 のリクエストではその 3 倍といったように 1-100 のリクエストに線形対応の CPU 負荷がかかるようになっている。図 4.2 のようにリクエスト x に対してアプリケーションは $t = 11 * x + 25$ms のレスポンスタイムを示すようなアプリケーションを構成した。

ii)　リクエストの生成モデル

アプリケーションは 10 台の仮想マシン上で走っており、この 10 台に負荷分散を行う。このときのリクエストをシミュレーションするとき、ポアソン到着を用いた。ポアソン到着とは待ち行列を考えるときもっともランダムな客の到着モデルをいう。ポアソン到着は独立性、定常性、希少性といった性質をもっており一人の到着が前後の到着によらないものとなっている。ここで客はアプリケーションを使うユーザーのリクエストに置き換えられる。平均到着率が λ[リクエスト/時間] の単位時間あたりの到着リクエストの離散確率分布は次のポアソン分布で表される。

$$P(N = k) = \lambda^k / k! * e^{-\lambda}$$

そして平均到着時間が $1/\lambda$[時間/リクエスト] の到着時間分布は次の指数分布の式で表される。

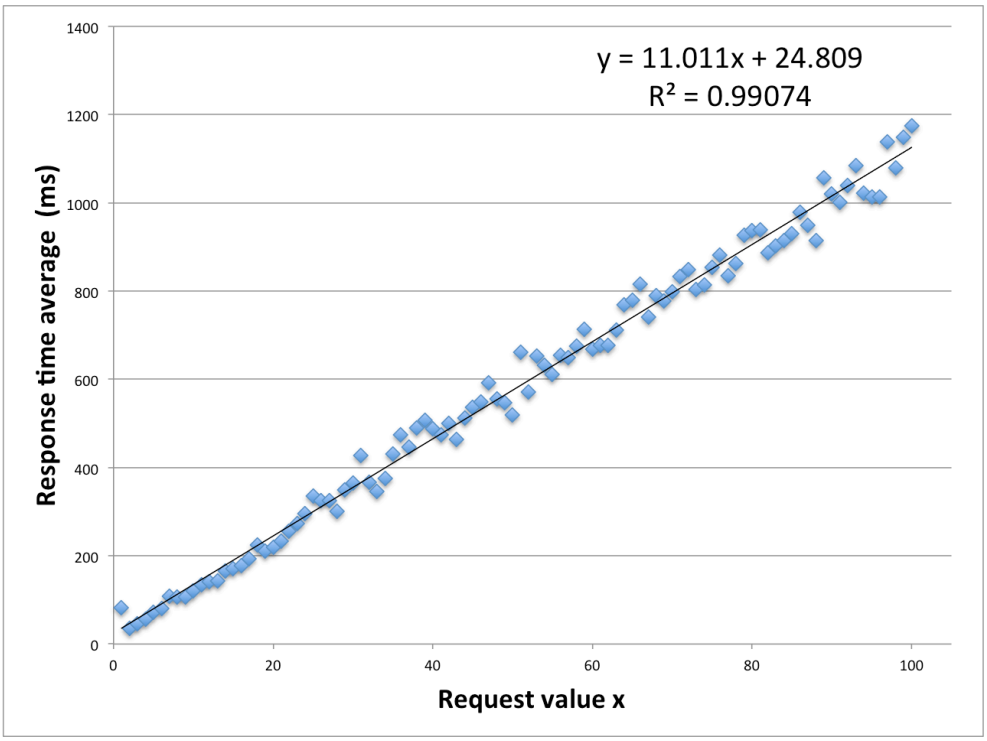


図 4.2: 実験アプリケーションのリクエスト x に対するレスポンスタイム (ms) の散布図とその近似線

$$f(t) = \lambda e^{-\lambda t}$$

アプリケーションのユーザーから上記のようにポアソン到着によってリクエストが来る。ユーザーのリクエストの内容は 1-100 までのパラメータのどれかになっていて、その値に対する CPU 負荷が生じることになる。実験ではこのリクエストのパラメータは 1-100 までの値が同確率で、つまり一様分布に従うまったくランダムなとき。また 1-100 までの値の生じる確率が正規分布になるようなリクエストが発生するとシミュレーションを行った。例えば平均 $\mu = 30$ で標準偏差 $\sigma = 1, 10, 20, 30, 50$ のときの 1-100 のリクエストがそれぞれ生じる確率は図 4.3 のようになる。シミュレーションでは乱数が 1 から 100 以外の数字の場合は乱数を再生成しているため分布は正確には正規分布よりちょっと異なる、図 4.3 の線図の面積の割合と同じになっている。リクエストパラメータの生成に正規分布を用いたのは $\sigma = 1$ のときほとんどのリクエストが平均 μ の値となりほとんどのリクエストが同じ量の負荷を掛けるものとなる。このようなときの負荷分散はラウンドロビンのような手法でも十分にパフォーマンスを発揮する。$\sigma = 10, 20$ くらいの値になると大部分のリクエストが $\mu \pm 10$ の値となり、ときおり $\mu \pm 30$ などのような大きなもしくは小さなリクエストが来るようなリクエストモデルとなる。そして $\sigma = 50, 70$ と大きくなるにつれ一様分布に近づいていく。このようなリクエストモデルを用いることで、ほとんどのリクエストが同様な負荷がかかるアプリケーションやリクエストの内容によって負荷が大きく変化するようなアプリケーションなどを想定できるようにした。

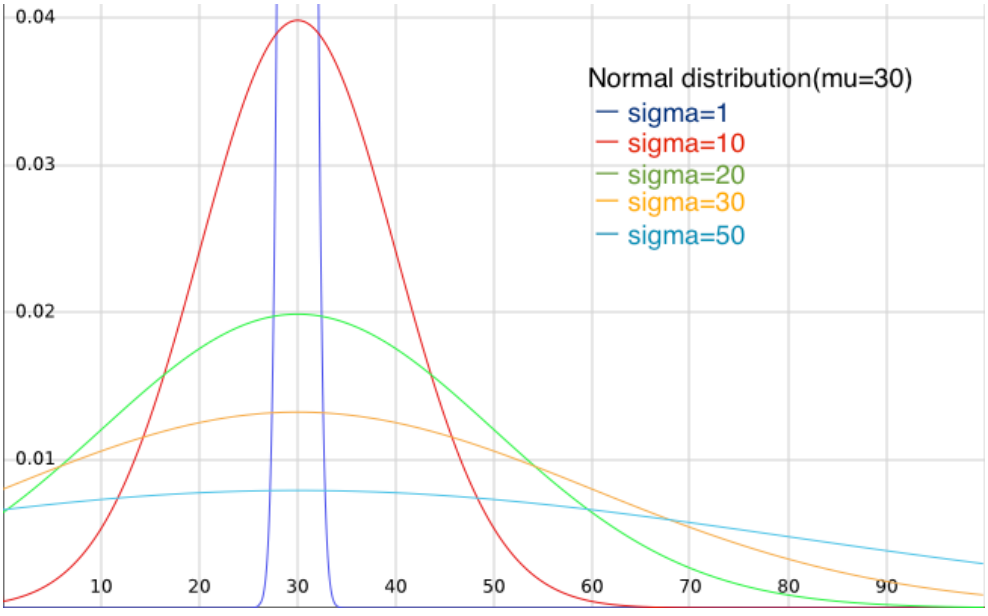


図 4.3: リクエスト x の生じる確率 (正規分布に近似)

iii)　ロードバランサーの構成

アプリケーションが走る 10 台のインスタンスに対して負荷分散を行うロードバランサーは次のものを用意した。既存の広く提供されているものとして Openstack の LBaaS で提供されている Round Robin 手法と Least Connection 手法を用いたロードバランサーを設置した。その比較対象として本研究で提案したリクエストの負荷評価を用いた Minw(Minimum workload) 手法を用意した。Minw 手法は Openstack のマスターノード上に走るプロセスとして稼働しており負荷評価の内容をロードバランサーサービスから得ているものだとしている。Minw の負荷分散手法は新しいリクエストが来たとき分散対象のサーバの中で一番少ない負荷を処理しているサーバに振り分ける手法である。この手法は集中型でリクエストの処理時間を考慮したものとなっている。ノードの状態を今までそのノードに振り分けた負荷の量から測っているため、直接的にノードの状態を監視していない。

4.2　リクエストの処理時間を考慮したロードバランサーの検証

4.2.1　実験内容

この実験では既存の広く提供されている負荷分散手法として Openstack で実装されている Round Robin 手法と Least Connection 手法とリクエストの内容を考慮した Minw 手法の比較を行う。このとき Minw 手法が用いている負荷評価の内容は正確なものとし、アプリケーションの 1-100 のリクエストに対して 1-100 の負荷量を正確にわかっているものとしている。実験ではポアソン到着で 500 リクエストを発生させ、リクエストの内容は上述した分布のものとなっている。正規分布の平均 μ と標準偏差 σ を変化させ、様々なリクエストモデルのときの各ロードバランサーのパフォーマンスを比較した。

比較するときの指標として、アプリケーションに負荷がかかっていない状態のときのリクエストの平均レスポンスタイム (図 4.2) と負荷分散を行ったときのリクエストのレスポンスタイムの倍率を用いた。低負荷時にはこの 2 つのレスポンスタイムはほとんど同じとなり倍率は 1 倍となる。高負荷時もしくは負荷分散がうまく言っていない場合はこの倍率が 1 より高くなってくる。シミュレーションで発生させた 500 リクエストの理想平均レスポンスタイムと比較した倍率の平均を求めることで各ロードバランサーの比較を行った。

ポアソン到着しているリクエストの負荷率を適度に調節したがらリクエストの分布を正規分布の平均 $\mu = 10, 30$ と標準偏差 $\sigma = 1, 10, 20, 30, 50$ と変化させ、3 つのロードバランサーをそれぞれ使用した時のレスポンスタイムを計った。

4.2.2　実験結果と考察

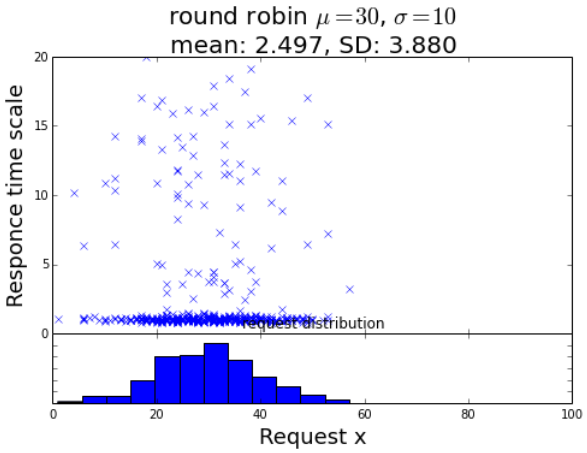
図 4.4、4.5、4.6 はそれぞれ Round Robin、Least Connection、Minw 手法を用いた時の結果の一部である。この図のあ上部の散布図は 500 リクエストの値とレスポンスタイムを理想の時間と比較した比率を表したものである。下部のヒストグラムは発生したリクエス

トの値の分布図である。リクエストの値の分布は想定通りおおよそ正規分布に従っているといえる。図の上部の散布図は負荷分散がうまくいっている場合は図 4.5a や図 4.6a のようにほとんどのリクエストが 1 倍率のレスポンスタイム、つまり理想平均レスポンスタイム通りに返事が帰ってきていることを意味する。図 4.4、4.5、4.6 の (a), (b), (c), (d) では、リクエスト分布が $\mu = 30$ と $\sigma = 10, 20, 30$ のときと一様分布 (ランダム) の時の結果を表している。

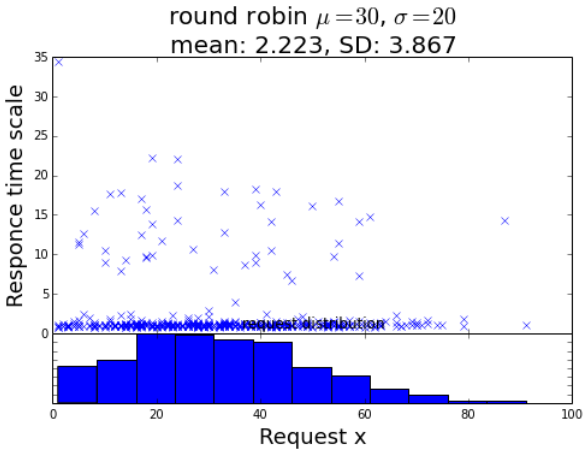
各シミュレーション結果を比較するためにレスポンスタイム比率の平均と標準偏差偏差を用いた。平均が 1 に近いほどリクエスト全体が理想のタイムで返ってきており負荷分散がうまくいっていることとなる。また標準偏差も大事な比較指標となる。標準偏差はレスポンスタイム比率のバラ付きを表している。全体の平均が 1 に近い結果となってもバラつきが高いということは負荷分散がうまくいかないリクエストの到着があったとき、ひとつのサーバに負荷が集中してしまいレスポンスタイムが伸びることになる。つまり局地的に負荷分散がうまくいかないケースが多いと分散が高くなることになる。

続いて図 4.7 では各ロードバランサーを色分けにし、平均レスポンスタイム比率の比較を表している。μ, σ を変化させた時の結果を表している。またエラーバーは標準偏差を表していてレスポンスタイムのバラ付きを示している。レスポンスタイム比率の分布は正確には正規分布ではないため厳密には上下のエラーバーは異なるはずだが、簡単のためこのように示した。また σ を変化させたときリクエストのポアソン到着の負荷率を調節しているため μ, σ がことなる結果は互いに相関関係はない。

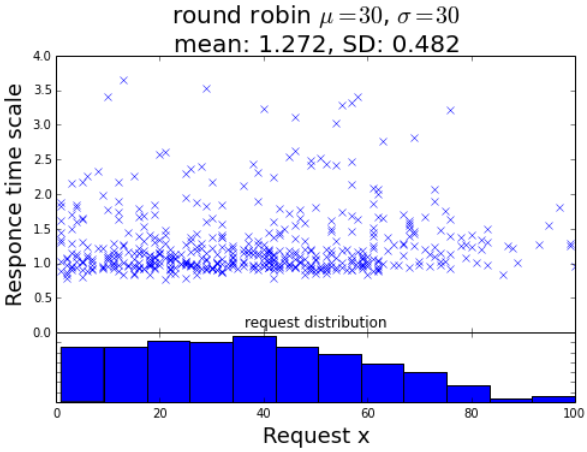
μ, σ を変化させることで様々なリクエストモデルをシミュレーションしている。例えば $\sigma = 1$ の時はアプリケーションに来るリクエストはほとんど同じ負荷がかかるものとなる、そのため Round Robin などの単純な手法でも十分に負荷分散を行うことができる。実際に図 4.7 では $\sigma = 1$ のときは 3 つの手法はほとんど同じパフォーマンスを発揮している。そして σ が大きくなっていくとリクエストの値のバラ付きは大きくなり、高負荷のリクエストと低負荷のリクエストが同じくらい起きることになる。この時は Round Robin や Least Connection などのリクエストの内容を考慮しない手法では同じサーバに高負荷のリクエストを集中させたり、低負荷のリクエストが集ったり、サーバがオーバーロードもしくは遊びができてしまう。$\sigma = 10, 20, 30$ と上がっていくに連れ Round Robin 手法は他の 2 つの手法より大きくパフォーマンスが落ちている。また σ が小さいとき Least Connection 手法は Minw 手法と比べ大きな差はないが、σ が大きくなってくると Minw 手法の方が Least Connection 手法より平均レスポンスタイム比率が下回っており、また標準偏差も小さいものとなっている。これはリクエストのバラ付きが大きくなってくるとコネクション数のみで負荷分散を行う Least Connection 手法では負荷を均等に振り分けることができなくなっている。



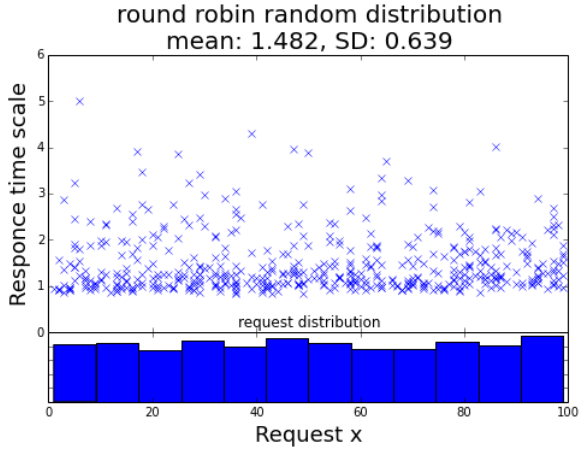
(a) $\mu = 30, \sigma = 10$ のとき



(b) $\mu = 30, \sigma = 20$ のとき

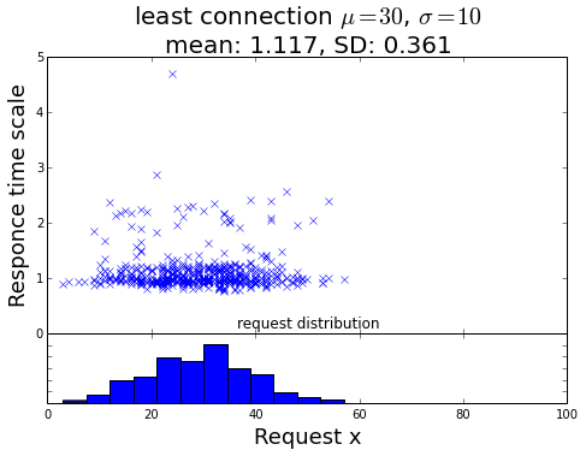


(c) $\mu = 30, \sigma = 30$ のとき

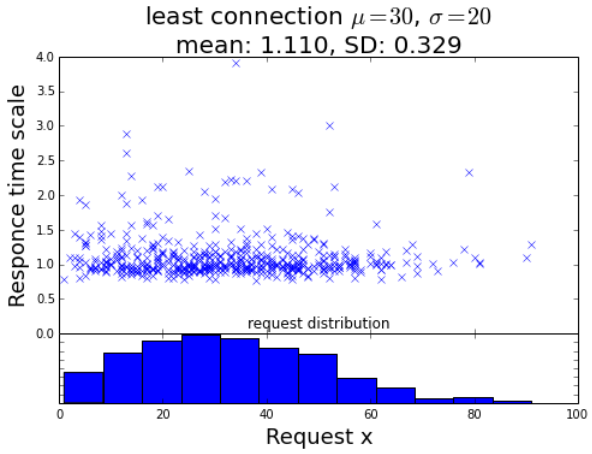


(d) ランダムのとき

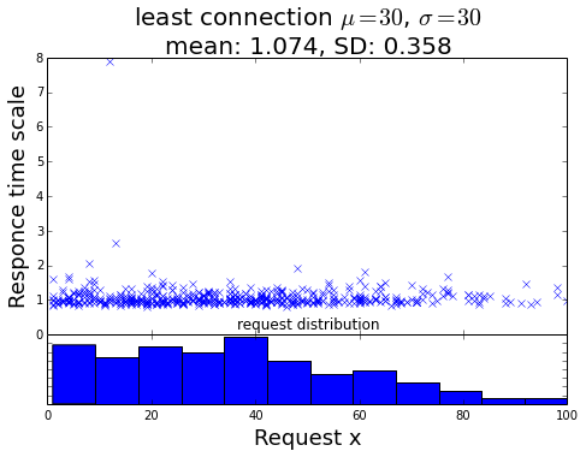
図 4.4: Round Robin 手法のリクエストとレスポンスタイム比率の散布図



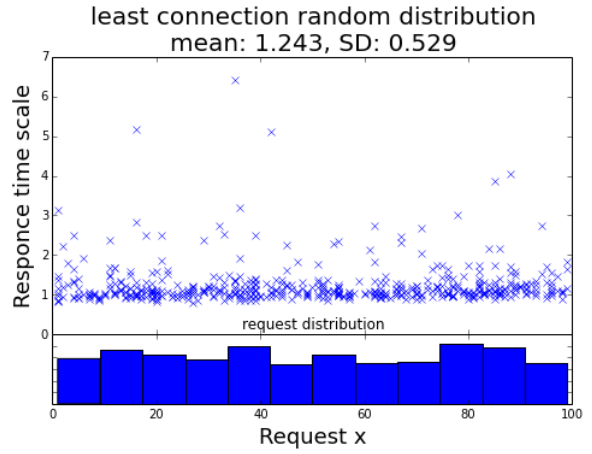
(a) $\mu = 30, \sigma = 10$ のとき



(b) $\mu = 30, \sigma = 20$ のとき

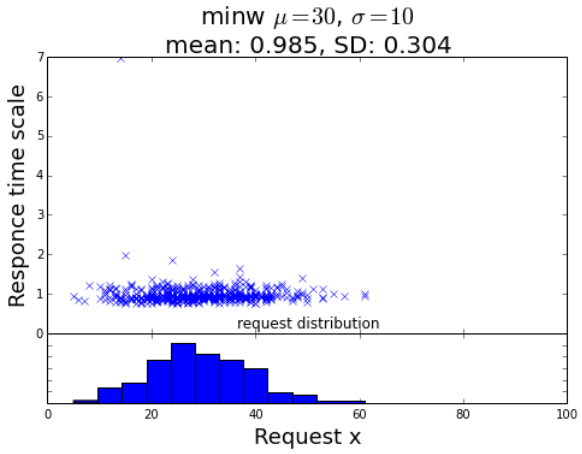


(c) $\mu = 30, \sigma = 30$ のとき

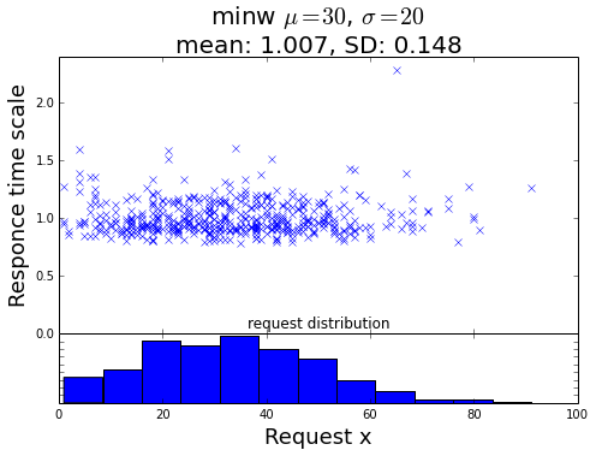


(d) ランダムのとき

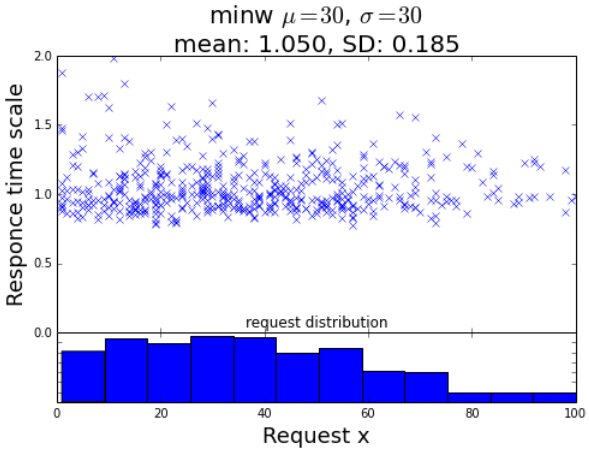
図 4.5: Least Connection 手法のリクエストとレスポンスタイム比率の散布図



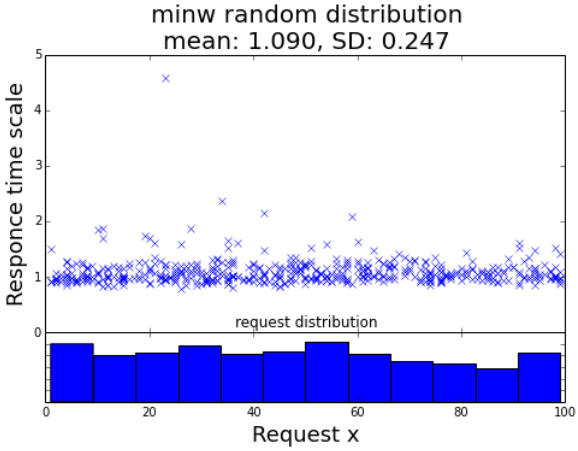
(a) $\mu = 30, \sigma = 10$ のとき



(b) $\mu = 30, \sigma = 20$ のとき

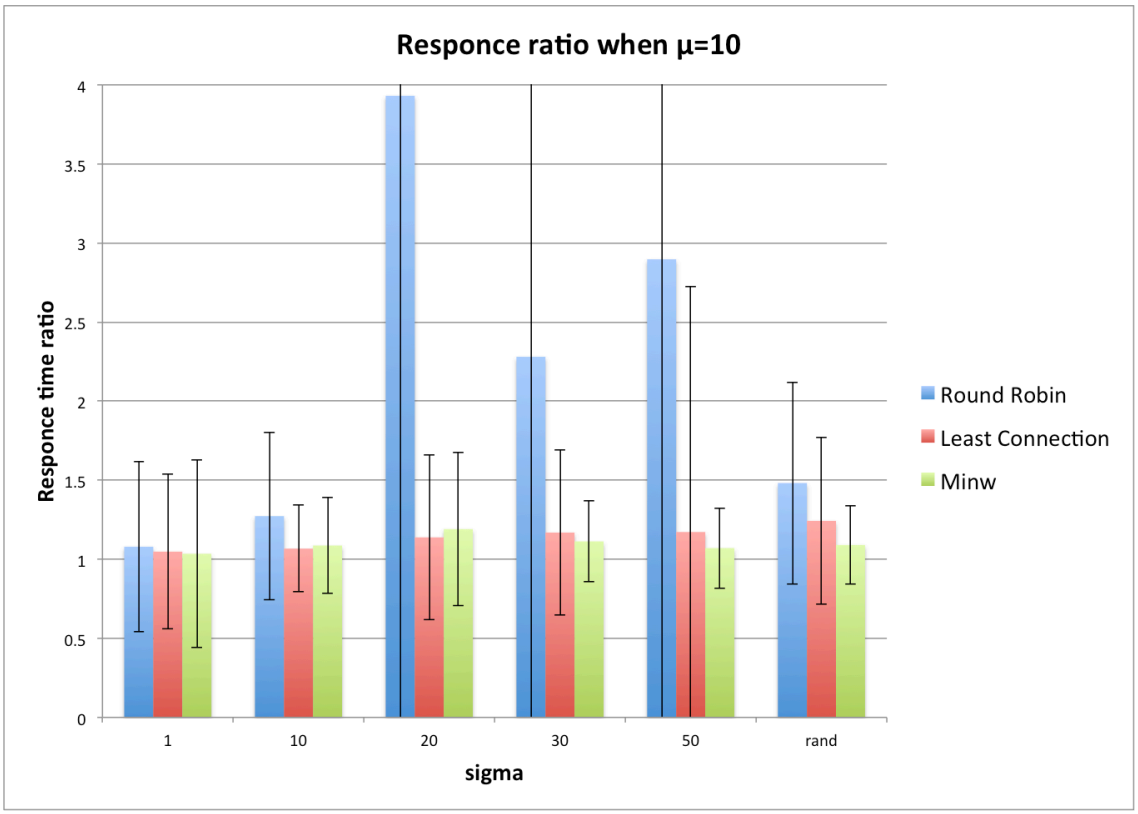


(c) $\mu = 30, \sigma = 30$ のとき

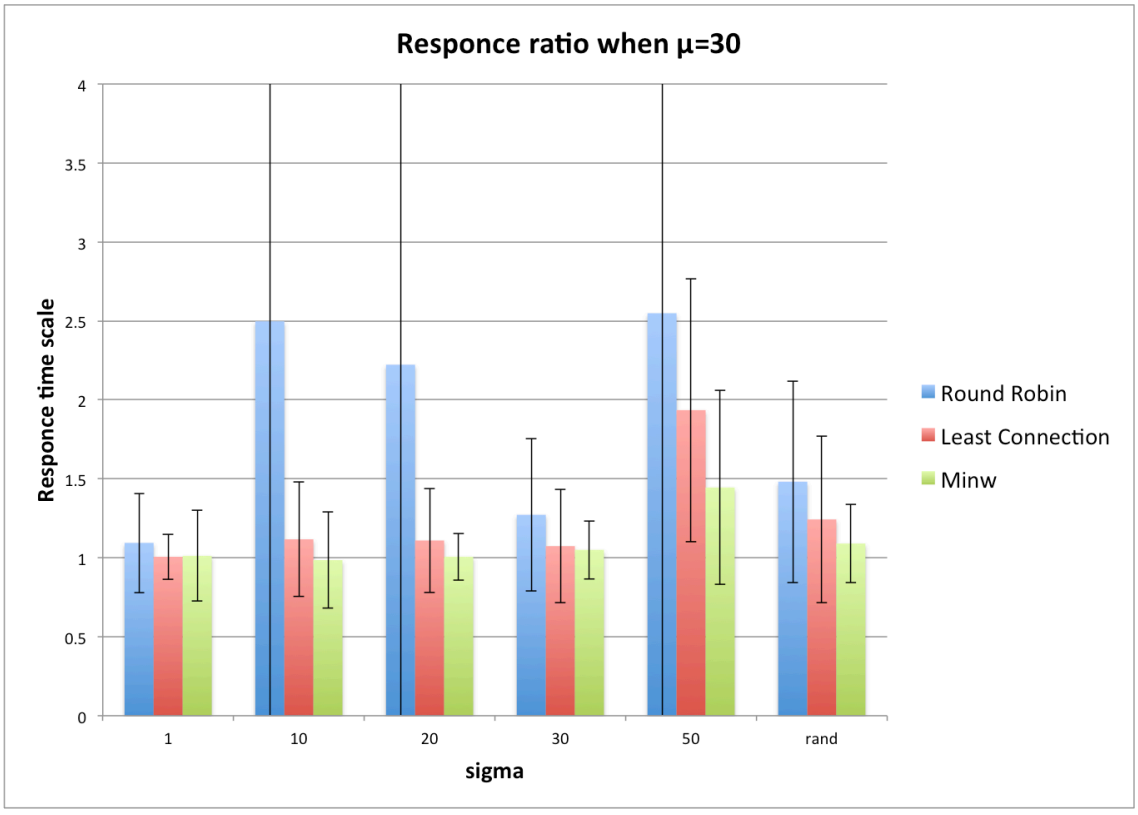


(d) ランダムのとき

図 4.6: Minw 手法のリクエストとレスポンスタイム比率の散布図



(a) $\mu = 10$ のとき



(b) $\mu = 30$ のとき

図 4.7: 各負荷分散手法のレスポンスタイム比率の平均とリクエスト分布の σ の関係 (エラーバーは標準偏差を示している)

4.3　負荷評価の精度による性能の検証

4.3.1　実験内容

ユーザーが負荷テーブルを入力するためにはリクエストによる負荷量を知る必要がある。前章の実験では 1-100 種類のリクエストの負荷量を正確に知っているときのシミュレーションとなっている。負荷テーブルが正確なためリクエストの負荷を考慮した Minw 手法はパフォーマンスを発揮していた。

この実験ではユーザーに入力してもらう 1-100 のリクエストの負荷量の精度について検証を行う。負荷全体を Precision の値だけ分割することができたとするときの負荷テーブルによって Minw 手法のパフォーマンスにどのような影響が出るかを実験した。Precision が 100 のときは 1-100 のリクエストを 100 分割することができたので $1:1, 2:2, 3:3, ...100:100$ と正確な対応付けができたとなる。Precision が 3 のときはユーザーは 1-100 の負荷を大、中、小の 3 つの値にしか分割できなかったことになる。実験では $Precision = 1, 2, 3, 4, 5, 10, 20, 30, 50$ と変化させた時、ランダム分布のリクエストのときの Minw 手法のパフォーマンスを測定した。

また上記の分割では 1-100 までのテーブルを均等に分割できている時の場合を想定した。しかし現実には高いもしくは低い負荷のリクエストをより詳細に分割できる可能性が考えられる。図 4.8 のように均等に分割するときと指数的に高いもしくは低い負荷の部分を詳細に分割した時のロードバランサーのパフォーマンスの影響をシミュレーションした。図では 5 分割しているが実験では 10 分割したときのシミュレーションを行った

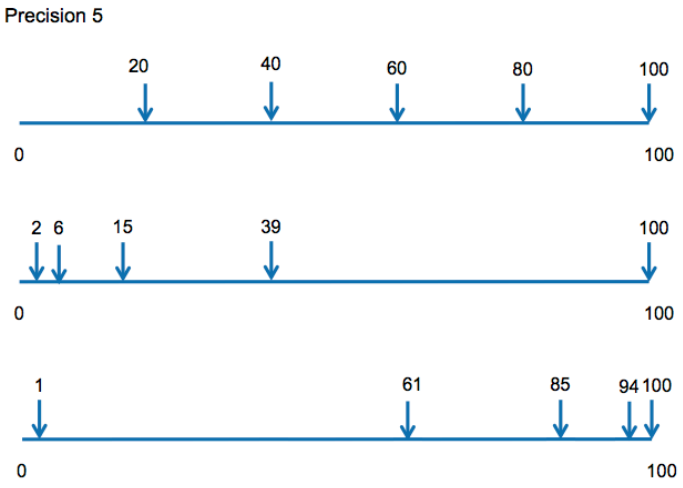


図 4.8: 指数的にリクエストを分割する

4.3.2　実験結果と考察

負荷テーブルの正確度についての実験の結果は 4.9 のようになった。Precision が 100 分割から 10 分割まで大きな差は見られなかった。しかし 3 分割、2 分割となると本来の負荷の量とテーブルの値との差が大きくなりパフォーマンスが落ちている。

また図 4.8 のようにリクエストを 10 分割したとき、均等に分割するのと、大きいもしくは小さい負荷をより詳細に分割したときの比較を行った。その結果、表 4.1 のようになった。大きい負荷を詳細に分けたほうが少しパフォーマンスが落ちているが全体的に優位な差は見られなかった。このようにユーザーは負荷テーブルを入力する際にある程度の分割精度が落ちてもロードバランサーのパフォーマンスが大きく落ちることがないと考えられる。

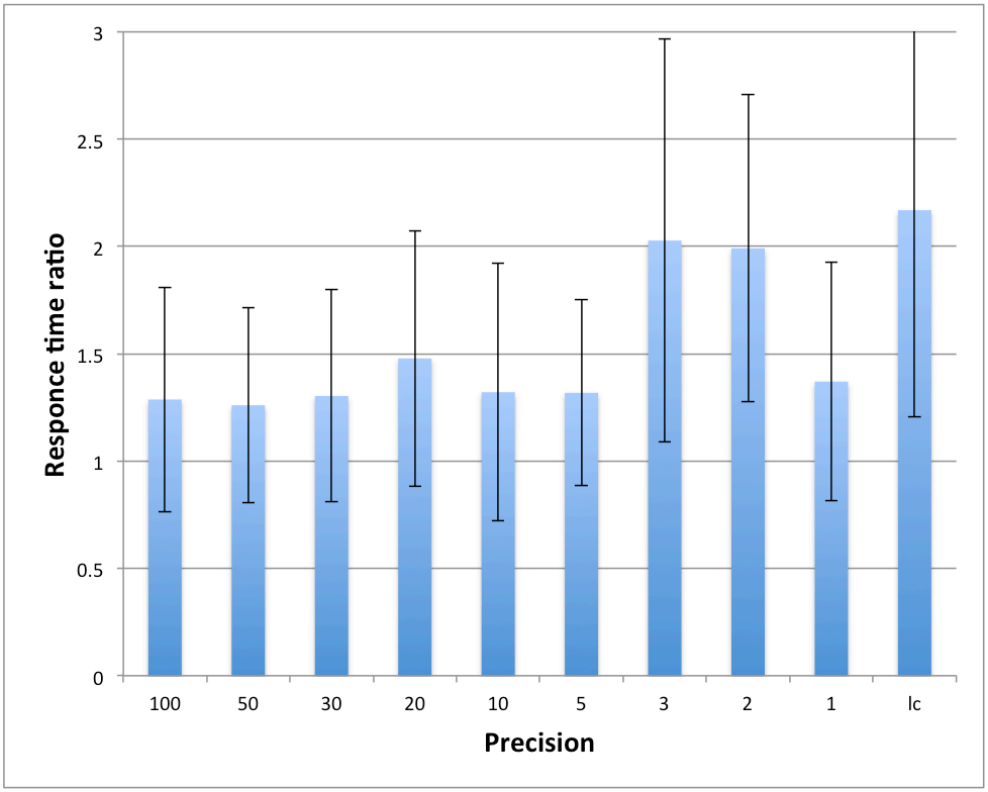


図 4.9: リクエストの負荷量の精度 (Precision) の変化によるレスポンスタイム比率の変化

表 4.1: リクエストを均等、大きいまた小さい負荷を詳細に分割した時のロードバランサーパフォーマンス

	平均	標準偏差
均等分割	1.321233962	0.598850561
小さい負荷が詳細	1.317571253	0.540736848
大きい負荷が詳細	1.386542701	0.554718156

第5章

結論

5.1 まとめ

クラウドを用いて分散システムを構築しアプリケーションを立ち上げるとき、効率的なロードバランサーを用いて負荷分散を行うことは大事である。様々な手法の中から自分のアプリケーションに適したロードバランサーを選んで負荷分散を行うことはアプリケーションのレスポンスの向上や、資源の有効活用に繋がる。クラウドではLBaaSという形でロードバランサーが提供されている。しかし既存のLBaaSではロードバランサー手法の選択性に欠けるもので、限られた手法のみが提供されている。本論文では研究段階にある様々なロードバランサーをLBaaSとして提供するにはどんな機構が必要かを議論した。。多様なロードバランサーを提供可能なLBaaSを考える上で既存研究のロードバランサーがどのように実装されているかを大きく分類し、それぞれの種類のロードバランサーを提供するための機構を現状のLBaaSのアーキテクチャを踏まえて考察を行った。

既存研究では効率的、効果的なロードバランサーを実現するためにはリクエストによるアプリケーションへの負荷を知る必要があった。リクエストから負荷量を知るために、アプリケーションを開発したユーザー側にその情報を入力してもらう必要がある。LBaaSにおいてリクエストの負荷量の情報として負荷テーブル機構をどう提供するかについて様々な観点から考察した。ユーザーに負荷テーブルどのようにして入力してもらうかについて議論した。また負荷テーブルの正確性の問題についてクラウド基盤構築ソフトウェアのOpenstackを用いて実環境に近い状態をシミュレーションによって検証した。

LBaaSの既存のアーキテクチャの中にリクエストの負荷情報を管理する負荷テーブル機構を設け、ロードバランサーやユーザーとのつなぎ役として稼働させる仕組みを検討した。様々なプロトコルに対応する負荷テーブルを実現するにはLBaaS側からは決まった規格を設ける必要がある。そしてユーザー側からはその規格に従った、リクエストの内容から負荷の量を知ることができるテーブル入力してもらう必要がある。場合によってはアプリケーションを2段階構成にし、リクエストの負荷を推定するフロントアプリケーションと、リクエストの負荷処理を行う計算ノードからなるデザインパターンの実装が考察された。

ユーザーから完全正確な負荷テーブルの入力を求めるのは難しい。アプリケーションによってはユーザーからの入力に性格で精度の高い情報を入力するのは難しいと考えられる。

本論文ではその精度によるロードバランサーのパフォーマンスの影響を小規模なクラウド環境で計測した。またロードバランサーなどから得られるログやノードの状態をもとに負荷テーブルの修正を行う機構の提案を行った。負荷テーブルを修正するモジュールを LBaaS 内に組み込むことでユーザーは容易に負荷テーブルを作成することができ、さらに修正されたデータを元にアプリケーションの監視や開発などに役立つ情報を得られる。

5.2　今後の課題

本研究では LBaaS において多様なロードバランサーを提供するための仕組みを検討し、シミュレーションによって負荷テーブルを用いたロードバランサー手法の有用性と問題点についての考察を行った。本研究では LBaaS におけるロードバランサー手法の選択性の課題解決のため、新たに負荷テーブルモジュールを LBaaS に組み込むことについて検討した。負荷テーブル作成の詳細な仕様などは様々なアプリケーションやロードバランサーを考慮し詳細な仕様をモデリングする必要がある。負荷テーブルのリクエストの内容の判別方法としてのリクエストの内容やリクエストのパラメータの読み込みなどの仕組みのさらなる検討が必要とされる。また負荷の量となる CPU やディスク IO やネットワーク幅などの入力をどのように取り込み、ロードバランサーがそれをどのように用いるかを考える必要がある。

負荷テーブルを修正するモジュールについてはロードバランサーから得られる情報を活用し、負荷テーブルを修正する機構を今後の課題とする。負荷テーブルを修正するモジュールには様々な手法を導入することができる。

謝辞

本論文を書き上げるにあたってたくさんの方々にお世話になりました。
研究のご指導いただいた相田仁教授には貴重な意見をいただき、研究方針を導いていただきました。
矢谷浩二准教授には毎週の打ち合わせでは研究の助けとなる貴重な意見をたくさんいただきました。
千葉新吾さんと古宇田フミ子さんと元岡みさ子さんには日頃の研究生活、研究室まわりの面でたくさんのサポートしていただき大変お世話になりました。
研究室メンバーの皆さんとは研究室生活を楽しく過ごすことができ、またたくさんのことを勉強させていただきました。
そして家族、友人たちのたくさんの応援のもと頑張ってこれました。
皆様には深く感謝しております。本当にありがとうございました。

参考文献

[1] Rahman, M., Iqbal, S., & Gao, J. (2014, April). Load balancer as a service in cloud computing. In Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on (pp. 204-211). IEEE.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," Nist Spec. Publ., vol. 145, p. 7, 2011.

[3] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, p. 50, Apr. 2010.

[4] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in 2008 10th IEEE International Conference on High Performance Computing and Communications, 2008, pp. 513.

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Futur. Gener. Comput. Syst., vol. 25, no. 6, pp. 599–616, 2009.

[6] Beloglazov, A., & Buyya, R. (2010, May). Energy efficient resource management in virtualized cloud data centers. In Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing (pp. 826-831). IEEE Computer Society.

[7] Randles, M., Lamb, D., & Taleb-Bendiab, A. (2010, April). A comparative study into distributed load balancing algorithms for cloud computing. In Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on (pp. 551-556). IEEE.

[8] Kansal, N. J., & Chana, I. (2012). Cloud load balancing techniques: A step towards green computing. IJCSI International Journal of Computer Science Issues, 9(1), 1694-0814.

[9] Agrawal, A., Manish, G., Milind, R. N., & Shylaja, S. S. (2014). A Survey Of Cloud Based Load Balancing Techniques. In Proceedings of Int. Conf. on Electrical, Electronics, Computer Science & Mechanical Engg., 27th April-2014, Bangalore, India.

[10] Randles, M., Odat, E., Lamb, D., Abu-Rahmeh, O., & Taleb-Bendiab, A. (2009, December). A Comparative Experiment in Distributed Load Balancing. In Developments in eSystems Engineering (DESE), 2009 Second International Conference on (pp. 258-265). IEEE.

[11] Nuaimi, K. A., Mohamed, N., Nuaimi, M. A., & Al-Jaroodi, J. (2012, December). A survey of load balancing in cloud computing: Challenges and algorithms. In Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on (pp. 137-142). IEEE.

[12] Nakrani, S., & Tovey, C. (2004). On honey bees and dynamic server allocation in internet hosting centers. Adaptive Behavior, 12(3-4), 223-240.

[13] Venkata Krishna, P. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. Applied Soft Computing, 13(5), 2292-2303.

[14] Mondal, B., Dasgupta, K., & Dutta, P. (2012). Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. Procedia Technology, 4, 783-789.

[15] Zhang, Z., & Zhang, X. (2010, May). A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on (Vol. 2, pp. 240-243). IEEE.

[16] Rahmeh, O. A., Johnson, P., & Taleb-Bendiab, A. (2008). A dynamic biased random sampling scheme for scalable and reliable grid networks. The INFOCOMP Journal of Computer Science, 7, 1-10.

[17] Saffre, F., Tateson, R., Halloy, J., Shackleton, M., & Deneubourg, J. L. (2009). Aggregation dynamics in overlay networks and their implications for self-organized distributed applications. The Computer Journal, 52(4), 397-412.

[18] Fujitsu,“負荷分散入門,”http://fenics.fujitsu.com/products/ipcom/catalog/data/1/1.html

[19] Amazon web services, http://aws.amazon.com/

[20] The Openstack Project, http://www.openstack.org/

[21] Openstack Neutron LBaaS, https://wiki.openstack.org/wiki/Neutron/LBaaS/

[22] Apache JMeter, http://jmeter.apache.org/

[23] Amazon web services,“Amazon Elastic Load Balancing,”http://aws.amazon.com/elasticloadbalancing/

[24] Google Developers, “Google Compute Engine Load Balancing,” https://developers.google.com/compute/docs/load-balancing/

[25] Microsoft Azure, “Traffic Manager,” https://azure.microsoft.com/en-us/services/traffic-manager/

[26] Rackspace US Inc,“Rackspace Cloud Loadbalancers,”http://www.rackspace.com/cloud/load-balancing/

[27] ニフティクラウド,“ニフティクラウドのロードバランサー,”http://cloud.nifty.com/service/lb.htm

[28] IDCF Cloud,“IDCF クラウド仕様・機能,”http://www.idcf.jp/cloud/spec/vr.html#lb

[29] GMO クラウド, “ロードバランサー,” http://private.gmocloud.com/server/option/loadbalancer.html

[30] NTT Communications Cloudn,“Cloudn Load Balancing Advanced,”https://www.ntt.com/cloudn/data/lba.html

[31] KEMP Technologies, https://kemptechnologies.com/

[32] F5 Networks, https://f5.com/

[33] HAProxy, http://www.haproxy.org/

発表文献

ダワージャルガル オリギル, 相田 仁, “多様なロードバランサーを提供可能なLBaaS(Load Balancer as a Service)の検討”, 電子情報通信学会 情報ネットワーク研究会, Mar. 2016(予定)