**２０２１　年度　　卒業論文**

卒業論文題目

|  |
| --- |
| 検索システムを対象とした異種サーバにおける応答速度を考慮したロードバランサ |

　指導教員　　　　　　鷹野　孝典　教授

神奈川工科大学　情報工学科

　　　　　学籍番号　　　　１８２１０８６

　　　　　学生氏名　　　　　 松尾　祐介

　提出日　２０２１年　１２月２３日　指導教員　　印

　受理日　２０２１年　１２月２３日　情報工学科長　　印

# **論文要旨**

これから書きます。

# 目次

[**論文要旨** 2](#_Toc87721650)

[目次 1](#_Toc87721651)

[**1.** **まえがき** 1](#_Toc87721652)

[1.1 はじめに 1](#_Toc87721653)

[1.2 本論文の構成 2](#_Toc87721654)

[**2.** **関連研究** 3](#_Toc87721655)

[2.1 リバースプロキシによるロードバランシング手法 3](#_Toc87721656)

[2.2 応答速度評価付けシステムの評価手法 3](#_Toc87721657)

[2.3 Webサーバ計測システムの設計・開発 3](#_Toc87721658)

[**3.** **提案システム** 5](#_Toc87721659)

[3.1 提案システムの概要 5](#_Toc87721660)

[3.2 提案システムの実行手順 5](#_Toc87721661)

[3.2.1 評価付けシステム 6](#_Toc87721662)

[3.2.2 テーブル設計 6](#_Toc87721663)

[3.2.3 ロードバランサに使われる技術 7](#_Toc87721664)

[**4.** **実装システム** 8](#_Toc87721665)

[4.1 実装システムの概要 8](#_Toc87721666)

[4.2 評価付けの方法 9](#_Toc87721667)

[4.3 検索システムの構築 9](#_Toc87721668)

[4.4 実装した割り振り方法 12](#_Toc87721669)

[4.5 ロードバランサの構築 12](#_Toc87721670)

[4.5.1 開発環境 12](#_Toc87721671)

[4.5.2 計測評価システム 14](#_Toc87721672)

[4.5.3 コンフィグ書き換え 15](#_Toc87721673)

[4.5.4 コンフィグの適応 16](#_Toc87721674)

[4.5.5 データベースの構築と接続 16](#_Toc87721675)

[4.6 実装システムの実行 17](#_Toc87721676)

[**5.** **実験** 20](#_Toc87721677)

[5.1 実験目的 20](#_Toc87721678)

[5.2 実験環境 20](#_Toc87721679)

[5.3 実験方法 21](#_Toc87721680)

[5.4 実験結果 21](#_Toc87721681)

[5.5 実験の考察 23](#_Toc87721682)

[**6.** **むすび** 24](#_Toc87721683)

[**謝辞** 25](#_Toc87721684)

[**参考文献** 26](#_Toc87721685)

[その他 27](#_Toc87721686)

# **まえがき**

## はじめに

Webサーバは重要なライフラインになりつつある．サービス全体の応答性能や可用性を高めることができる「ロードバランサ」の需要は今後増加傾向になると予想される．

大企業だけでなく中小企業や個人のサイトでもサービスが拡大するにつれてWebの負荷分散が可能なサーバロードバランシングは重要視される．SEO(検索エンジン最適化)の観点から，競合サイトと比較し自身のサイトの表示速度が遅いとランキング評価で不利になるとされている6)為，負荷分散時にも応答速度に配慮する必要がある．通常ロードバランサのリバース先であるサーバ群は同性能であることが望ましいとされている．しかし，リプレイスによって導入された新しいサーバと旧式のサーバを混合して負荷分散に利用されることも個人や中小企業を中心に見受けられる．

こうしたサーバ間の性能や通信装置の性能にバラつきがある場合，応答速度が一定とは限らないが，導入のしやすさからロードバランサは頻繁にラウンドロビンと呼ばれる単純に空いているサーバへ割り振る方式が利用される．この状況では負荷分散をしているにもかかわらず，遅いサーバにつないでしまいWebの表示速度が返って落ちてしまうという課題がある．そうした課題を解決するには，異なる性能のサーバそれぞれの応答速度を計測し，ロードバランサが判断して動的に割り振りを行う必要がある．

本研究では，性能差のあるWebサーバからなるWebサイト環境において，応答速度によってサーバの割り振り先を決めるアルゴリズムの提案，Webサーバの応答速度を考慮したロードバランサの設計と開発を行った。提案システムでは，それぞれのWebサーバの応答速度を計測し，提案するアルゴリズムに基づいて評価付けを行う．応答の遅速に関しては個人によって異なる為,「Webサイトの反応時間の遅延と、それに対するユーザの反応」7)を参考に評価システムを設計した．また，この情報は評価済み応答速度としてデータベースへ保管される．データベースから現在と過去24時間の状態を抽出し，その情報をロードバランサに与えることで，不均一なサーバからなるWebサイトにおいて，応答速度を考慮しつつ負荷分散をすることができる．

Webサーバで負荷分散を行うロードバランサはプロキシ機能を応用している．このプロキシには不特定多数のクライアントから，特定のサーバへ通信を代わりに行うリバースプロキシとクライアントから不特定のサーバにアクセスする際，クライアントの代わりに通信を行うフォワードプロキシが知られている．Webサーバで負荷分散するには前者が使われる．NGINXというWebサーバはコンフィグを書き換えることで，リバースプロキシが行える．また，重みづけにも対応している為，応答速度を計測した後，自身の作成した評価アルゴリズムに基づいて重みを変更する。その後適応させる処理を行うことで，応答速度に基づく動的ロードバランサを開発可能である．通常コンフィグを適用させるにはWebサーバの再起動が必要だが,稼働率が落ちてしまう問題が発生する。そこで設定ファイルを再起動することなく適応できるreloadと呼ばれる機能を使用した．

実験では教育目的で開発されたシングルボードコンピュータであるRaspberry Pi 4B 4Gを4台使用する。ロードバランサ1台と検索システムを積んだWebサーバ3台を冗長的に配置する．OS（オペレーティングシステム）上でネットワークトラフィックをコントロールする「TCコマンド」やRAM，CPUの使用率を変更できる「stressコマンド」を使用し，異種のサーバを疑似的に再現する．

本研究では実装した提案方式のプロトタイプと一般的なロードバランサの割り振り方式である「ラウンドロビン」を応答速度に基づいて比較実験することで提案システムの有用性を示す．

## 本論文の構成

　本論文は全６章で構成されている．本章では研究の背景や動機，目的について述べる．第２章ではロードバランスの手法や計測システムの関連研究について述べる．第３章では本研究の提案方式とその特徴である本システムの内容，応答速度をどのように計測・評価し，ロードバランサが判断し割り振りを行うのか述べる．第４章では提案方式を構築した本システムの設計及び実験システムの実装について述べる．第５章では一般的なロードバランサと本システムで比較実験を行う．実験の概要と，比較実験後の結果を述べる．第６章では結論と今後の展望について述べる．

# **関連研究**

## リバースプロキシによるロードバランシング手法

HTTPセッションのハンドオーバによるWebサーバのロードバランスについて土居幸一郎,後藤滋樹らは既存のWebロードバランシング手法では，実行中のデータ転送を再配分することが出来ない.そこで，リバースプロキシ型のロードバランシング手法を元に，データ転送中であっても，相対的に負荷の高いサーバから負荷の低いサーバへと移し替えられる機構を提案した．4MB以下のファイルでは,ロードバランサを経由しないダウンロードの方が,ロードバランサを経由する場合に比べて,わずかに必要時間が短かった.しかし,4MB以上ではロードバランサを経由する場合の方のダウンロード時間が短くなる場合がある.したがって,音楽や映像のファイルのような比較的サイズの大きなファイルに対しては,はかの要素の寄与度に比べて,提案手法によるオーバヘッドは十分に小さいと言える.ロードバランサを経由しない場合とハンドオーバを行う場合とで,ダウンロードに必要な時間に違いは生じなかった、試行によっては,ハンドオーバを行った場合でロードバランサを経由しない場合のダウンロード時間を下回る場合もあった,よって,提案手法によるハンドオーバによるコストは十分に小さいことがわかった.が増加するに従って,サーバリソースの利用効率が向上す一方で,5接続を超えてからは利用効率が向上していない.その場合は,別の5接続数以下のサーバにデータ転送をハンドオーバすることによって,全体で利用効率を向上させることができる.図5の結果をもとに,2台のバックエンドサーバ同士でクライアント接続数が5台以下となるように調整した場合の平均転送速度と最低転送速度の増加率を表3に示す,平均速度と最低速度を,それぞれ最大で18.7%,31.1%向上させられることが分かった．

## 応答速度評価付けシステムの評価手法

複数のロードバランサによるWebシステムの応答時間最適化について，河野知行はバランサなし，定率分散，状態分散，応答時問分散の4つのケースを比較した．それらの振り分け技法の特徴を整理すると．パランサがなくとも，処理要求がパランサに偏りなく到着すれば，速い応答時間が確保される．しかし，偏りが生じれば，応答時間は指数的に悪くなる.定率分散は到着する処理要求に偏りが生じても，応答時間が指数的に悪くなる問題は解消できる.但し，偏りがない場合は，バランサがない場合より応答時間は長くなる.状態分散を行うと，応答時間は大幅に短縮される.M/M/n構成の待ち行列のような効果を得ることができる.応答時間分散は，ロジックの複雑さに比例した効果を得ることができない.定率分散の改善型と位置づけられる技法であろう.一般的なロードバランサが提供していると考える処理要求の振り分け技法を検証した.その結果により，状態分散が最も効果的な振り分け技法であるとの結論を得た.

## Webサーバ計測システムの設計・開発

クラウドに適したWebシステムにおいて堀内晨彦 , 最所圭三らはWebシステムの概要及び試作システムの評価について述べ，その負荷量の監視機能の改善方法を提案・評価を行った．i. Apacheのserver-statusで負荷量を測定　ii. 外部プログラムで負荷量を測定方法iでの測定結果を図4に,方法iでの測定結果を図5に示す．実験では,前の測定が終了してから1秒後に次の測定を開始する.そのため,プロットの間隔は負荷量の測定にかかった時間+1秒になる.どちらの方法でもスレッド数の増加に伴い,CPU 使用率及び稼働率が上昇しているが,図5のCPU 使用率はCGIを処理するため,すぐに100%に達している,図4では稼働率が1.0に達すると,次の測定までに1分以上かかっている．これはWebサーバデーモンの最大同時処理数以上のアクセスがあり,過負荷になっているためである.また,どちらの図でもアクセスが終わってから負荷量が低下するまでに20秒程度かかっているが,これはWebサーバが各スレッドの行った最後のアクセスを処理しているためである.以上の結果から,アクセス数に応じて増減する稼働率が負荷量に適していることが分かる．CPU使用率は同時アクセス数が少なくても100%に達するため,過負荷の判別ができない.また,稼働率の測定方法としては,Web サーバが過負荷になった場合でも短時間で測定が行える方法iが適しているが,過負荷を脱してから稼働率が低下するまでに時間がかかる.反対に方法iでは過負荷を脱すると直ぐに稼働率が低下するが,過負荷になった場合の測定に時間がかかる.

# **提案システム**

## 提案システムの概要

本研究では性能差のあるWebサーバからなるWebサイト環境において応答速度によってサーバの割り振り先を決めるアルゴリズム，応答速度を考慮したロードバランサの設計方法を提案する．

本提案システムでは常に応答速度を計測している為，サーバの故障やロードバランサとWebサーバ間でボトルネックが生じたとしても，ユーザがシステムの不具合を感じることはない．通常ロードバランサは性能が同一もしくはそれに近い性能で運用されることが前提とされているが，本提案システムの特徴として，異種で不均一な性能のサーバ環境において個人を対象とした負荷分散に着目している．また，ユーザは応答速度の評価を得ることでサーバの調子を確認することも可能になる．検索システムはWebサービスの中でも応答速度が重要視されている．また膨大な検索にも耐えられるためには負荷分散が重要であるため効果は高い．提案システムの流れ図は図 3‑1に，説明に関しては図 3‑1に示す．また評価済み応答速度を保管するデータベースのテーブル設計は表3-1に示す．

## 提案システムの実行手順

本提案システムではサーバの応答速度に変化が生じるとロードバランサが動的に割り振りを変更させるので負荷分散をしつつ最速のWebサーバへ多くのユーザをつなぐことが出来る．本提案システムの処理は大きく「計測」「評価」「処理」3段階に分けられ，5つのステップを踏んで実行される．STEP-1では想定しているWebサーバについて説明している．ロードバランサが応答速度に基づいて動的に割り振りを行うまでの流れに関しては，下記のSTEP-2からSTEP-5で説明する．提案システムにおける流れは図3-1に示す．

STEP-1: 冗長的で性能が不均一なWebサーバを用意．大量アクセスによるCPUやネットワークの負荷を分散させるため，仮想環境で作られるWebサーバではなく，ここでは物理的な別のサーバを数台用意する．

STEP-2: それぞれのサーバの応答速度を測るため，サーバにリクエストを送って応答速度を返す「応答速度計測プログラム」を作成し利用する．ここでは1分に1回各サーバの応答速度を計測する.

STEP-3: STEP-2で計測したデータは考案した応答速度評価アルゴリズムを用いてL1～Lnのn段階で評価を付ける．評価は主観的になりやすい為，「Webサイトの反応時間の遅延と，それに対するユーザの反応」6)と「RAILモデル」7)を参考に評価を行う.

STEP-4: 評価されたデータは評価済み応答速度としてデータベースへ保管される．

STEP-5: ロードバランサはこのデータベースへアクセスする．サーバの状態に応じて割り振り方法を動的に変化させることが可能になる．応答速度が最も早いサーバへの接続が優先される．

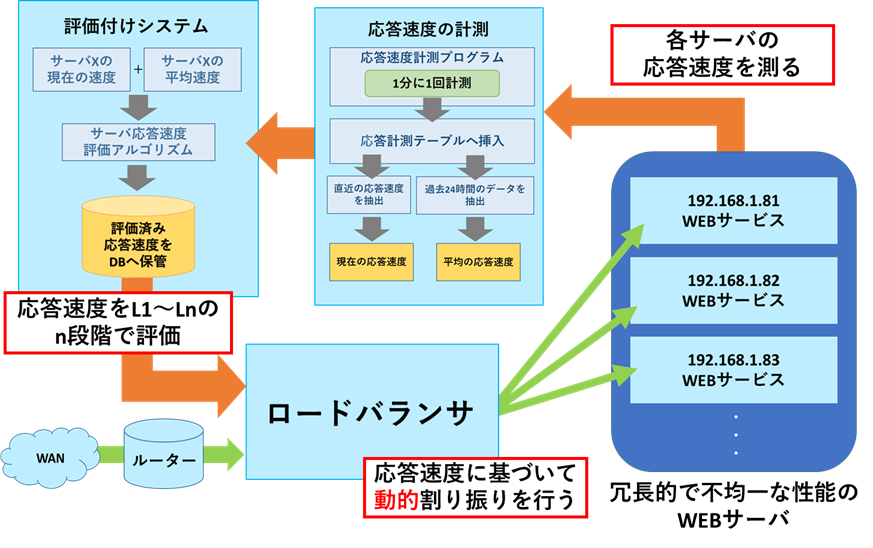


図 3‑1　提案システム図

### 評価付けシステム

実行手順のSTEP-3では評価付けを行う. L1～Lnのn段階で評価し表示することによってユーザは調子の悪いサーバを見つけ易くなる．評価付けに使う数式は下記に示す． は評価される値， は過去24時間の平均速度，は「Webサイトの反応時間の遅延と，それに対するユーザの反応」6)や「RAILモデル」7)に基づく応答速度の範囲を示す．

### テーブル設計

提案システムのテーブル設計内容を表3-2に示す．テーブルの設計では抽出した応答速度はfloat型，評価の格納はテキスト型とする．

表 3-1 提案システムのテーブル設計

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 名称 | 型 | デフォルト値 | 主キー |
| ID | 番号 | Integer | 1 | 〇 |
| now\_speed | 現在速度 | Float | NULL |  |
| now\_speed\_score | 現在速度評価 | Float | NULL |  |
| Ave\_speed | 平均速度 | Read | NULL |  |
| Ave\_speed\_score | 平均速度評価 | Read | NULL |  |
| datetime | 時刻 | Text | NULL |  |

### ロードバランサに使われる技術

Webにおいて，ロードバランサで負荷分散する場合に使われる技術として「リバースプロキシ」がある．これはプロキシ機能を応用した技術によってユーザからのリクエストを別のWebサーバへ転送している．動的，静的どちらのロードバランサを作成するときもこの機能は使われている．本提案システムでも，実行手順のSTEP-5でユーザからのアクセスを適宜Webサーバへ転送する為この技術を使う必要がある．

# **実装システム**

## 実装システムの概要

実装システムでは，物理的なWebサーバ，ロードバランサ機共に「raspberry pi 4b 4gb」を利用する。また，負荷分散するWebサービスは「検索システム」を対象とする。この検索システムは観光地のDBと接続され，場所をフォームに入力することで観光地として有名な場所を検索できる．図4-1では実装したシステムを説明し，流れは下記に示す．



図 4-1 実装システム図

STEP-1:異種環境を再現する為コマンドでraspberry pi 4b 4gbの性能を制限する。

STEP-2: 応答速度をPythonのHTTP通信ライブラリであるRequestsモジュールを利用し計測する．取得した値は現在の速度としてSQLITEデータベースへ格納する．

STEP-3: STEP-2:のデータベースへアクセスし過去24時間の平均速度を抽出する．表4-1はそのSQLの命令である．

STEP-4: 抽出した平均速度を設計した評価付けシステムへ渡し，S～Dの評価付けが行われる．

STEP-5: STEP-4:の評価を基にして，ロードバランサとして利用できるNginxのコンフィグを書き換える．最速のサーバ以外重みを下げる．

STEP-6: 変更した設定を適応させることでロードバランサは動的割り振りをはじめる．

表 4‑1過去24時間の平均速度を抽出するSQL文

|  |
| --- |
| select avg(speed) from response81 where datetime > datetime(datetime(), '-1 days', '+0 hours'); |

## 評価付けの方法

実装システムでは評価をS～Dの5段階評価とする．速度に対する評価の値は表4-2に示す．

表 4-2 応答速度と評価の対応表

|  |  |
| --- | --- |
| 応答速度 | 評価 |
| 0.000~0.016 | S |
| 0.017~0.099 | A |
| 0.100~0.999 | B |
| 1.000~9.999 | C |
| 10.000以上 | D |

## 検索システムの構築

本提案システムのロードバランサがリバースする場所は冗長的なWebサービスである．検索システムは応答速度や負荷分散が重要視される為，本提案システムのWebサービスでは「観光地検索システム」をRaspberry Pi上に構築する．観光地データベースに接続し，ユーザが場所を入力すると有名な観光地を返すシステムである．構築に使用する開発環境は表4-8に示す．データベースはMYSQLを利用し，WebサーバはApacheとする．データベースのスキーマは表4-15に，Apache2及びPHPのインストール手順は下記に示す．

**Apache**2**のインストール手順**

表4-3パッケージ一覧を更新

|  |
| --- |
| $ sudo apt update |

リポジトリ追加・削除時には必ず実行しなければならない。

表4-4パッケージを更新

|  |
| --- |
| $ sudo apt upgrade |

パッケージを更新しNGINXのインストール準備をする。

表4-5 apache2のインストール

|  |
| --- |
| sudo apt-get install apache2 |

**PHPのインストール手順**

表4-6 PHPのインストール

|  |
| --- |
| sudo apt install php |

表4-7 CLI，cURLモジュールのインストール

|  |
| --- |
| sudo apt install php-cli php-curl |

表4-8検索システムの開発環境

|  |  |  |
| --- | --- | --- |
| 名称 | バージョン | 利用用途 |
| Raspberry Pi | 4B 4G | Webサーバ |
| Apache | 2.4.38 | Webサーバ |
| MYSQL | 8.0.13 | 開発データベース |
| Google chrome | 94.0.4606.81 | Webブラウザ |
| Tera Term | 4.97 | SSH接続・ファイル転送 |
| Raspberry Pi OS | 9.4 | Raspberry PiのOS |
| Windows10 | 20H2 | OS |
| PHP | 8.0 | 開発言語 |

表4-9検索システムのプロジェクトファイルの説明

|  |  |
| --- | --- |
| ファイル名 | 機能説明 |
| pdo\_search\_form.html |  |
| pdo\_search.php |  |

表4-10検索システムへのアクセス方法

|  |
| --- |
| [http://192.168.1./](http://192.168.1.X/) |

Xはサーバの番号で本システムは81,82,83が存在する．ブラウザから各サーバへアクセスすると図4-1のように検索システムを利用できる．図4-2は検索語句を入力しサブミットした時の実行結果である．

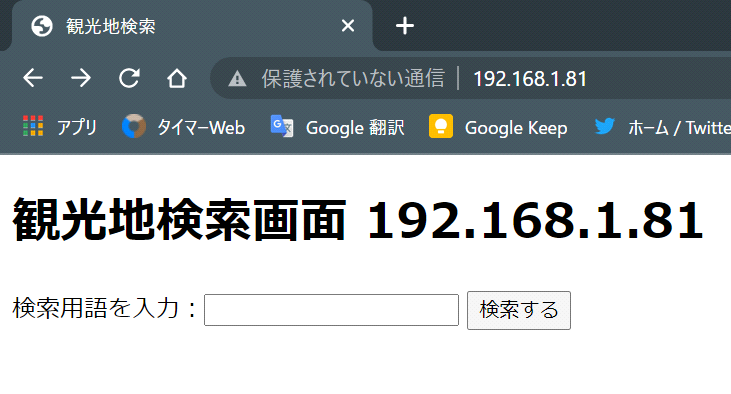
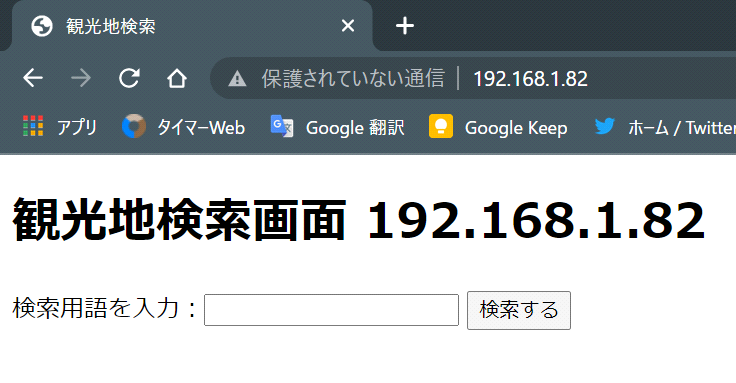
 



図4-1冗長的な検索システムのユーザインタフェース

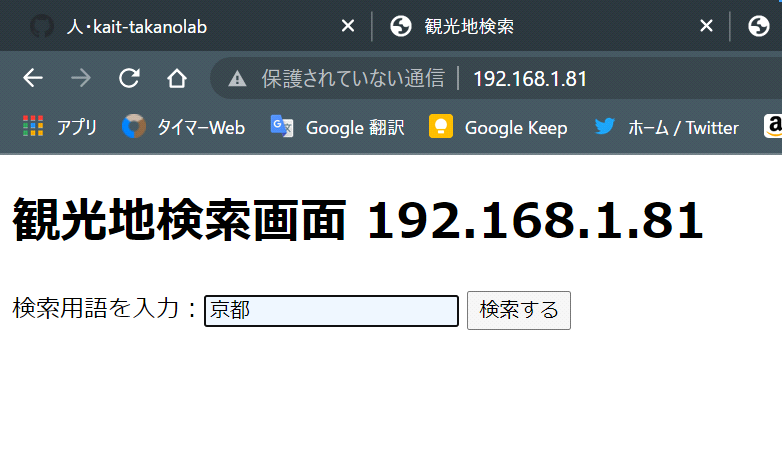
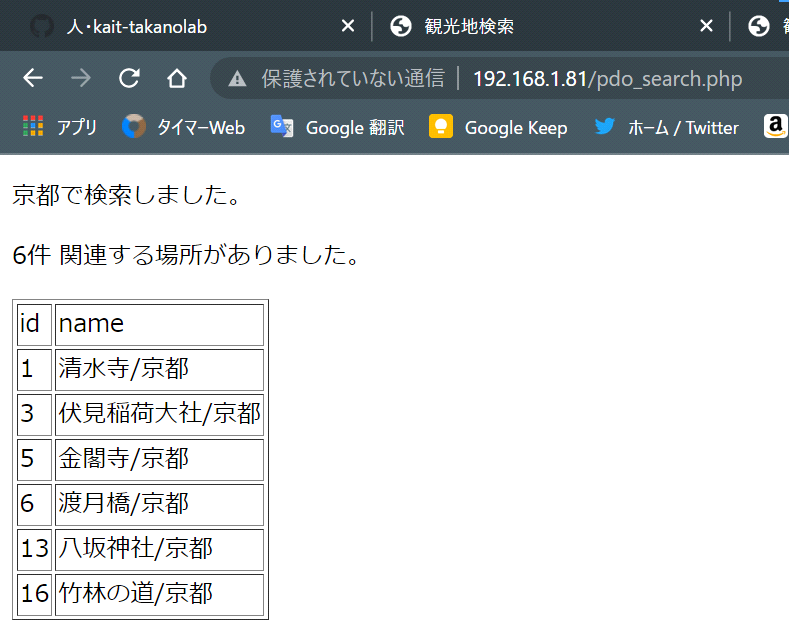
　

図4-2検索結果の例

表4-11フォームの処理(pdo\_search\_form.html)

|  |
| --- |
| <form action="pdo\_search.php" method="post">  検索用語を入力：<input type="text" name="name">  <input type="submit" value="検索する"> |

フォームに入力された値をPHPプログラムへpostを利用して送信する．

表4-12対象のカラムに対して文字列検索を行う(pdo\_search.php)

|  |
| --- |
| $statement = $dbh->prepare("SELECT \* FROM name WHERE name LIKE (:name) "); |

表4-13レコードを取得する(pdo\_search.php)

|  |
| --- |
| if($statement->execute()){  $row\_count = $statement->rowCount();  while($row = $statement->fetch()){  $rows[] = $row;  }  } |

表4-14データベースの接続

|  |
| --- |
| <p><?=htmlspecialchars($yourname, ENT\_QUOTES, 'UTF-8')."で検索しました。"?></p>  <p><?=$row\_count?>件 関連する場所がありました。</p> |

データベースMYSQLへの接続はPDOを利用する．

表4-15観光地データベースのスキーマ

(テーブル名：name)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 名称 | 型 | デフォルト値 | 主キー |
| ID | 番号 | Integer | 1 | 〇 |
| name | 観光地の名称 | text | NULL |  |
| Prefecture | 観光地の場所 | text | NULL |  |

## 実装した割り振り方法

本提案システムのロードバランサは過去24時間で評価が最も高いサーバに多く割り振る．評価の良いサーバが変化した場合，ロードバランサの割り振り方法も動的に変化する．また，現在の速度が10秒以上掛かるＤ評価の場合，計測した時点で何かしらのトラブルが生じている可能性が高い．したがって，そのサーバへの割り振りは行われないように設定している．この割り振りはNginx(エンジンエックス)というWebサーバを用いて，コンフィグの重みづけをプログラムによって書き換えることで実現させている．Nginxコンフィグの重みづけによる割り振り頻度は下記の数式で示す．はロードバランサの割り振り頻度， は重みづけの値である．

## ロードバランサの構築

本提案システムのロードバランサの構築を4.5.1から4.5.4で説明する。

### 開発環境

動的ロードバランサの構築にはNGINXというWebサーバを用いる。コンフィグデータを変更して重みづけをプログラムによって書き換え，再読み込みさせることによって実現させる。提案システムのプロトタイプを実装した開発環境は表4-16に示す．ロードバランサを作成するために用いたWebサーバのインストール手順は表4-18に示す．

表4-16実験システムの開発環境

|  |  |  |
| --- | --- | --- |
| 名称 | バージョン | 利用用途 |
| Raspberry Pi | 4B 4G | ロードバランサ |
| Python | 3.8 | 開発言語 |
| Spider(anaconda3) | 4.1.5 | 統合開発環境 |
| Google chrome | 94.0.4606.81 | Webブラウザ |
| Tera Term | 4.97 | SSH接続・ファイル転送 |
| SQLITE3 | 3.0 | 開発データベース |
| Raspberry Pi OS | 9.4 | Raspberry PiのOS |
| Windows10 | 20H2 | OS |

表4-17提案システム構築に用いたパッケージ

|  |  |  |
| --- | --- | --- |
| パッケージ名称 | バージョン | 備考 |
| SQLITE | 3.0 | 応答速度保管に使用 |
| Subprocess | 3.9.4 | プログラムからNGINX適応のシェルを呼び出す時に使用 |
| Datetime | 3.9.4 | DBへ日付挿入時に使用 |
| Requests | 2.21.0 | 応答速度取得に使用 |

実験システムのうちロードバランサはNGINXとラズベリーパイを用いて構築した．Raspberry Pi OSとNGINXのインストール手順を下記に示す．

**Raspberry Pi OSのインストール手順**

1. 公式サイトに配布されている「Raspberry Pi imager」をダウンロードする。

<https://www.raspberrypi.com/software/>

1. 「Raspberry Pi imager」を起動後，マイクロSDカードをPCに挿入する。「CHOOS STORAGE」　ボタンを選択するとSDカードを選択できる。
2. 「CHOOS OS」を選択し「Raspberry Pi OS(32-bit)」を選択する。
3. 「WRITE」を選択し，インストールを開始する。
4. 書き込み後Raspberry PiにSDカードを挿入し電源を入れて起動する。

**Nginxのインストール手順**

表4-18パッケージ一覧を更新

|  |
| --- |
| $ sudo apt update |

リポジトリ追加・削除時には必ず実行しなければならない。

表4-19パッケージを更新

|  |
| --- |
| $ sudo apt upgrade |

パッケージを更新しNGINXのインストール準備をする。

表4-20NGINXのインストール

|  |
| --- |
| $ sudo apt install nginx |

ロードバランサの設計・開発ではPythonの統合開発環境である「spider」をWindows上で利用し，TeraTermを用いてRaspberry Pi に転送した。本プロジェクトファイルの各種機能は下記に示す．

表4-21プロジェクトファイルの各機能

|  |  |
| --- | --- |
| ファイル名 | 機能説明 |
| measure\_evaluation\_InsertDB.py | 計測、評価、DB挿入プログラム |
| Response | データベースの |
| roop\_LoadBalancer.sh | 2つのプログラムを同時に動かしループさせる |
| nginxrestart.sh | NGINXの設定を再読み込みする |
| LoadBalancer.py | 設計したロードバランサ |

### 計測評価システム

応答速度を計測し評価するシステムはロードバランサとは別のプログラムで動いている.

4.5.2では計測システムの処理内容について説明する．

表4-22計測した値を評価する関数

|  |
| --- |
| def eva\_score(list):    #抽出した平均を評価  if 0<=list1[0] and list1[0]<=0.016:  evalution="S"  elif 0.017<=list1[0] and list1[0]<=0.099:  evalution="A"  elif 0.1<=list1[0] and list1[0]<=0.999:  evalution="B"  elif 1.0<=list1[0] and list1[0]<=9.999:  evalution="C"  elif 10.0<=list1[0]:  evalution="D"  return evalution |

表4-23応答速度を計測する処理

|  |
| --- |
| #応答速度計測  url = '[http://192.168.1.81](http://192.168.1.81/)'  res = requests.get(url)  time\_elapsed = res.elapsed.total\_seconds() |

requests.get()で要求を送信してから応答が到着するまでに経過した時間を計測している。ユーザがチェックしやすいよう秒に変換している

表4-24データベースＳＱＬＩＴＥに接続

|  |
| --- |
| #データベースＳＱＬＩＴＥに接続  dbname = 'response'  conn = sqlite3.connect(dbname) |

表4-25応答速度テーブルへ計測結果を挿入

|  |
| --- |
| setdb1 = (None, time\_elapsed,dt\_now)  cur.execute("insert into response81 (id,speed,datetime) values (?,?,?)", setdb1) |

表4-26応答速度テーブルから過去24時間の平均を取り出す

|  |
| --- |
| cur.execute("select avg(speed) from response81 where datetime > datetime(datetime(), '-1 days', '+0 hours');")  list1 = cur.fetchone()  evaluation\_ave=eva\_score(list1) |

### コンフィグ書き換え

コンフィグの重みづけを書き換えることでロードバランサの割り振り方法を変更している．表4-27では共通の条件をそのまま書き込み24時間のサーバ速度の平均で最も高ければ優先して接続されることになる．優先以外の接続は12回に1回接続されることになる．

表4-27コンフィグを書き換える処理

|  |
| --- |
| f = open('/etc/nginx/nginx.conf', 'w', encoding='UTF-8')  datalist1=['userwww-data;\n','worker\_processesauto;\n','pid/run/nginx.pid;\n','include /etc/nginx/modules-enabled/\*.conf;\n','events {\n',' worker\_connections 768;\n','}\n','http {\n',' include /etc/nginx/mime.types;\n','gzip on;\n','gzip\_vary on;\n','upstream backend1{\n']  f.writelines(datalist1)  if minmam == list1[0]:#81が平均最速なら  print('\n過去24時間で「サーバ81」が一番応答速度が速いので優先接続されます。')  datalist2 = ['server 192.168.1.81 weight=10;\n server 192.168.1.82 weight=1;\n server 192.168.1.83 weight=1;\n']  f.writelines(datalist2)    elif minmam == list2[0]:#82が平均最速なら  print('\n過去24時間で「サーバ82」が一番応答速度が速いので優先接続されます。')  datalist2 = ['server 192.168.1.81 weight=1;\n server 192.168.1.82 weight=10;\n server 192.168.1.83 weight=1;\n']  f.writelines(datalist2)    elif minmam == list3[0]:#82が平均最速なら  print('\n過去24時間で「サーバ83」が一番応答速度が速いので優先接続されます。')  datalist2 = ['server 192.168.1.81 weight=1;\n server 192.168.1.82 weight=1;\n server 192.168.1.83 weight=10;\n']  f.writelines(datalist2) |

### コンフィグの適応

reloadというNGINXを再起動しなくても変化させたコンフィグを適用させる機能を利用し表4-28では処理を適応させている．

表4-28nginxrestar.shの処理

|  |
| --- |
| #! /bin/sh  sudo systemctl reload nginx  exit 0 |

表4-28のシェルをロードバランサのプログラムで呼び出し実行させる。シェルを呼び出すにはsubprocessというモジュールを用いる。

表4-29nginxrestartのシェルを呼び出す

|  |
| --- |
| subprocess.run(['/home/pi/tools/nginxrestart.sh']) |

### データベースの構築と接続

本システムにおいて設計したデータベースについて説明する． 計測システムで測定した一連の応答速度はテーブルresponseで，評価システムで評価された応答速度はテーブルmonitoringで管理する。Webサーバごとに分けている為本システムでは下記のテーブルが3つずつ存在している。データベースの種類は軽量なSQLITEを使用して保管している．

表 4-30毎分計測した応答速度を格納するテーブル・スキーマ

(テーブル名：response81,82,83)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 名称 | 型 | デフォルト値 | 主キー |
| ID | 番号 | Integer | 1 | 〇 |
| now\_speed | 現在速度 | Read | NULL |  |
| datetime | 現在速度評価 | Text | NULL |  |

表4-31評価システムで評価された値を格納するテーブル・スキーマ

(テーブル名：monitoring81,82,83)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性名 | 名称 | 型 | デフォルト値 | 主キー |
| ID | 番号 | Integer | 1 | 〇 |
| now\_speed | 現在速度 | Read | NULL |  |
| now\_speed\_score | 現在速度評価 | Read | NULL |  |
| Ave\_speed | 平均速度 | Read | NULL |  |
| Ave\_speed\_score | 平均速度評価 | Read | NULL |  |
| Datetime | 時刻 | Text | NULL |  |

## 実装システムの実行

ロードバランサを実行するには下記の表4-3で示した「roop\_LoadBalancer.sh」を実行する．提案システムの動作には計測評価プログラムである「measure\_evaluation\_InsertDB.py」とロードバランス処理を行う「LoadBalancer.py」を同時に実行する必要がある．下記シェルではプログラムの同時実行とループ処理を担うことでロードバランサを動作させている．

表4-32プログラム同時実行とループ処理のシェル（roop\_LoadBalancer.sh）

|  |
| --- |
| #!/usr/bin/bash  for i in `seq 100`  do  sudo python3 LoadBalancer\_method2.py  sudo python3 measure\_evaluation\_InsertDB.py  sleep 20  done |



図4-3実装システムの診断、計測、評価を出力する様子（roop\_LoadBalancer.sh）

実装システムのプロセスは図4-3のようにコンソールで確認できる．図4-3ではロードバランサの診断結果を返している．この場合，過去24時間でサーバ81の応答速度が一番速い為，優先して接続される．また，現在速度や過去24時間の平均速度が評価と共に表示される為，ユーザは直観的にサーバの調子も確認することができる．



図4-4実装システムの動作の様子

図4-4ではロードバランサのIPアドレスである192.168.1.80へブラウザからアクセスしている．何度か更新ボタンを押すと異なるサーバへ転送される．表示されているタイトルのIPアドレスはリバース先のサーバを判別する為に付けている．図１で優先接続はサーバ81と書かれている．つまり他のWebサーバよりも多く192.168.1.81サーバ転送されることになる.

# **実験**

本章では，提案システムを利用した比較実験について述べる．

## 実験目的

設計・開発した動的ロードバランサと一般的に使用されている順に割り振る方式（ラウンドロビン）を比較し，応答速度が向上したのかを実験で調査する．

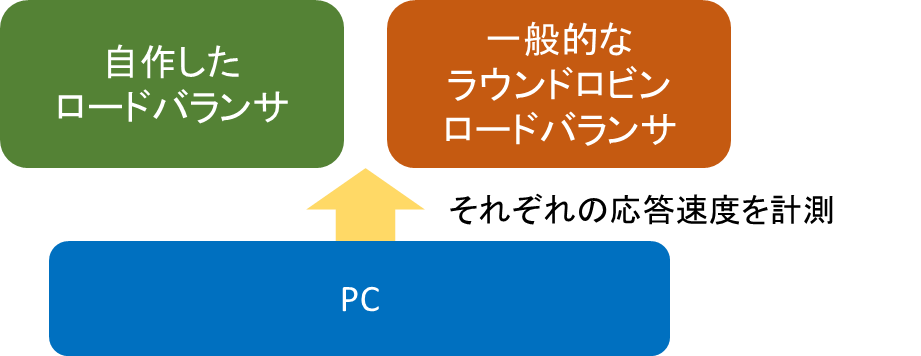


図 5‑1 実験目的のイメージ

## 実験環境

実験で使用するハードウェアは全てRaspberry Pi 4B 4Gを使用する．大量アクセスによるCPUやネットワークの負荷分散を想定し，ここでは物理的な別のサーバを数台用意する．表 5-1実験では使用するハードウェアの機器名用途を記している．ロードバランサと検索システムでWebサーバが異なるのは，設計・開発した動的ロードバランサがNginxの機能を利用している為である．異種のサーバ環境でのロードバランスを再現する為，stressコマンドやトラフィックに制限を設けるTCコマンドを利用する．表5-2ではそれぞれの変更点や使用するコマンドを示す．

表 5‑1実験で使用するハードウェア

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 機器名 | 台数 | 用途 | OS | Webサーバ |
| Raspberry Pi 4B 4G | 3 | 物理的に冗長化した検索システム | Raspbian | Apache |
| Raspberry Pi 4B 4G | 1 | 設計・開発した動的ロードバランサ | Raspbian | Nginx |

表 5‑2異種サーバ環境を再現するコマンド

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | IPアドレス | 変更点 | コマンド |
| Webサーバ１ | 192.168.1.81 | メモリ：4G→1Gへ(3Gのメモリを獲得) | stress -m 1 --vm-bytes 3221225472 --vm-hang 0 -q & |
| Webサーバ２ | 192.168.1.82 | CPU使用率を最大に | stress -c 4 |
| Webサーバ３ | 192.168.1.83 | ネットワーク トラフィック | sudo tc qdisc add dev wlan0 root tbf limit 1000b buffer 700b rate 50bps |

## 実験方法

5.1で示した通り実験では本提案システムとラウンドロビンロードバランサを比較実験する．単純にpingをかけて実験すると，ロードバランサまでの応答速度を計測してしまう．そこでPythonのモジュールであるrequests.getを利用する． httpのgetでリクエストを送るのでリバース先の速度も含めて計測できる．また，同モジュールのスクレイピング機能を利用し，割り振ったサーバの種類も観測する．具体的な実験手順は下記に示す．  
  
**実験手順**

1. Webサーバを不均一にする．使用するコマンドは表 5‑3に示す．
2. コンフィグの設定を変更し重み付けを等しくすることでラウンドロビンとして割り振られる．
3. 作成した実験用計測プログラムを用いて，「ラウンドロビン」での表示速度を測る．5分毎の計測を6時間行う．
4. (3)の計測終了後，提案システムのロードバランサに切り替える．本提案システムの起動方法は4章に示している．
5. 再び実験用計測プログラムを使い，「提案システム」での表示速度を測る．同様に5分毎の計測を6時間行う．
6. 表示にかかった時間の平均を求めて比較実験終了．

## 実験結果

5.3で示した流れで実験を行い， 5分毎6時間続けて計測を行った．ラウンドロビンの速度推移は図 5‑2で，提案システムの推移は図 5‑3で示す．



図 5‑2　ラウンドロビンの速度グラフ



図 5‑3　提案システムの速度グラフ

表 5‑4　サーバ毎の割り振り回数と頻度（ラウンドロビン）

|  |  |  |  |
| --- | --- | --- | --- |
| サーバ | 192.168.1.81 | 192.168.1.82 | 192.168.1.83 |
| 回数 | 25回 | 25回 | 24回 |
| 選択頻度 | 33% | 34% | 33% |

表 5‑5　サーバ毎の割り振り回数と頻度（提案システム）

|  |  |  |  |
| --- | --- | --- | --- |
| サーバ | 192.168.1.81 | 192.168.1.82 | 192.168.1.83 |
| 回数 | 50回 | 12回 | 11回 |
| 選択頻度 | 68% | 16% | 15% |

表 5‑6　測定した速度の平均を比較

|  |  |
| --- | --- |
| ラウンドロビンの平均速度 | 提案システムの平均速度 |
| 0.4304秒 | 0.1523秒 |

測定した速度の平均はラウンドロビンが0.4304秒，提案システムでは0.1523秒という結果が得られた．応答速度に関して本提案システムが優れている．

## 実験の考察

実験結果では，異種環境において一般的な割り振り方式であるラウンドロビンよりも本提案システムを利用する方が応答速度に関して速いことが確認できた．このことから，実験目的である「Webサーバの応答速度を考慮したロードバランサの実現」と「ロードバランサとWebサーバのボトルネック削減」を実現できており，本システムの有用性を証明できたと考える．

サーバ毎の割り振り回数と頻度を観るとラウンドロビンは33～34%と頻度がほぼ等しいが，本システムでは実験時の最速サーバであった192.168.1.81に68%という高い頻度で割り振りが行われたことが確認できる．このことからラウンドロビンは遅いサーバにも多く接続してしまうが，本システムはそのようなサーバを避けることが出来た為，応答速度の平均が向上したと推測できる．

応答速度に基づく割り振りは，性能差のあるコンピュータでも偏りが生じにくい．新しいサーバと旧式のサーバを混合して利用することも可能である為，比較的安価なロードバランサの構築が可能であると考える．

実装システムには各サーバの状態をS～Dという分かりやすい評価値で表示させる機能が組み込まれている為，ネットワークやサーバの知識に長けていないユーザでもトラブルに気づくことが出来ると考える．

ロードバランサは負荷分散することが最も大事だが，分散するサーバのうち1つでも故障やトラブルが生じてしまうと可用性が大きく低下する．本システムでは，応答速度を常に計測し応答が著しく遅い場合割り振りは行われない．したがって，こうしたサーバにつなぐリスクを回避することが可能である．このように可用性を維持したまま応答の速い状態で負荷分散を行うことも可能である為本実験のような異種環境だけでなく，同性能のサーバ群においても有用性は十分にあると考える．

# **むすび**

構成：

目的に対してどういう成果が得られたか書く  
実験結果や考察もすこしを盛り込んで成果を説明。

今後の展望なども書くみたい。

箇条書き

・異種環境の疑似的な再現ではなく，本当に性能差のあるコンピュータでテストしたい。

・実験では実際に大量のユーザが実際に利用した状況ではない．  
→実際に運用しているサーバ検索システムよりも負荷のかかるシステムで実験  
　　→違った結果が得られるかもしれない。

・電気やガス水道などと同じくWebサイトも重要なライフラインになりつつあるので  
こうした可用性とボトルが少しでも改善されることを期待している．

# **謝辞**

# **参考文献**

1. クラウドに適したWebシステムの負荷監視機能の改善について堀内晨彦 , 最所圭三第76回全国大会講演論文集,2014(1),437-438 (2014-03-11)
2. 複数のロードバランサによる Ｗｅｂシステムの応答時間最適化河野 知行Tomoyuki KAWANO情報処理学会研究報告システム評価（EVA）,2007(63(2007-EVA-021)),27-34 (2007-06-22)
3. 土居幸一郎,後藤滋樹HTTPセッションのハンドオーバによるWEBサーバのロードバランス (分散システム/インターネット運用技術・高品質インターネット)掲載誌 情報処理学会研究報告 = IPSJ SIG technical reports 2008(23) 2008.3.6・7 p.25～29
4. International Journal of Computer Networks & Communications (IJCNC) Vol.10, No.3, May 2018PROPOSED LOAD BALANCING ALGORITHM TOREDUCE RESPONSE TIME AND PROCESSING TIMEON CLOUD COMPUTING Nguyen Xuan Phi, Cao Trung Tin, Luu Nguyen Ky Thu and Tran Cong Hung  
   <https://www.researchgate.net/profile/NguyenPhi2/publication/325715539_Proposed_Load_Balancing_Algorithm_to_Reduce_Response_Time_and_Processing_Time_on_Cloud_Computing/links/5b5aef5b458515c4b24a9e98/Proposed-Load-Balancing-Algorithm-to-Reduce-Response-Time-and-Processing-Time-on-Cloud-Computing.pdf>
5. Scalable Load Balancing in the Presence of Heterogeneous ServersAuthors: Kristen Gardner, Jazeem Abdul Jaleel, Alexander Wickeham, Sherwin Doroudi  
   <https://arxiv.org/pdf/2006.13987.pdf>
6. ページの表示速度とSEOの関係  
   Daniel An. Find out how you stack up to new industry benchmarks for mobile page speed. (2017-02)
7. 反応時間の遅延と、それに対するユーザの反応  
   Paul Kinlan.What do people want from a news experience? (2014-12-8)
8. Webパフォーマンスの計測・最適化(RAILモデル)  
   GoogleInc.TheRAILPerformanceModel.  
   https://developers.google.com/web/tools/chrome-devtools/profile/evaluate-performance/rail, 2018.(2021/01/20)

# その他

* ソースコード等を記述するときは，表内に挿入し，行間を「固定値：12pt」に設定する．行間を制する者はwordを制する．
* 研究でやったことはすべて，基本的に「本研究では」と称する
* ファイル名は，「1421~-name-thasis.docx」で統一する．
* 目次では、謝辞や参考文献のページ番号は載せない
* 要旨には，背景，課題，提案，実験で確認したことを書く
* 図表番号とラベルの間は半角スペースを入れる
* 節と節の間は改行する．章と章は改ページする
* 本文は日本語：MS明朝　　英語：Cenutty
* ただし数式は「挿入→数式」から入力し，本文中で数式を用いる場合はフォントの形が似ているTimes New Romanを使う．
* 付録にあるプログラムの説明がしたいとき本文の方に，例えば，“（付録３参照）”，“……を付録３に示す．”等と参 照先を記しておく．
* 章の終わりに改ページを行う．
* 改ページはCtrl+Enter
* 今後の課題ではなく展望．修正点とか書かない