分散 Web システムにおけるロードバランサを用いたオートスケールアルゴリズムの改良と評価

16T259 畑 智裕（最所研究室）

クラウド環境において負荷量に応じてキャッシュサーバ数を動的に増減させることで，コストを低減しつつ応答性を高めるロードバランサを用いたオートスケールアルゴリズムの改良について述べ，その実装と評価について述べる．

1 はじめに

クラウドで提供されている仮想マシンをキャッシュサーバとして用いることで容易に負荷分散が行えるようになった．そこで，当研究室ではクラウド環境を対象にした負荷量に応じて動的にキャッシュサーバ (VC サーバ) 数を増減させることで，応答性を確保しつつ運用コストを低減する分散 Web システムを開発している [1][2]．先行研究では，負荷量として稼働率 (同時にサービスできる最大数に対する実際の同時サービス数) を用いる方法とスループットを用いる方法で比較実験を行い負荷量としてスループットを用いるほうが適切であると示した [1]. 本研究では，先行研究で開発されたオートスケールアルゴリズムの再評価を行い，結果から見えてきた問題点を解決するアルゴリズムを提案する．

2 分散 Web システム概要

図 1 に分散 Web システムの概要を示す．本システムは拡張ロードバランサ，コンテンツを提供するオリジンサーバ，オリジンサーバから取得したキャッシュを提供する VC サーバ群から構成される．拡張ロードバランサは，ソフトウェアロードバランサに，サーバの負荷量を監視する負荷監視機能，負荷量に応じて VC サーバを起動・停止するキャッシュサーバ管理機能，VC サーバ数の増減に合わせてアクセスの振分先を更新する振分先設定機能を持つ拡張プログラムを追加したものである．負荷量の監視および VC サーバの増減は拡張プログラムの機能で行い，リクエストの振り分けはソフトウェアロードバランサの機能を用いて行う．

スループットを用いる負荷量の計算式を式 (1) に示す．式 (1) では，一定区間 (i) の全稼働サーバの合計スループットの移動平均 (TP_{MAiAll}) をそれぞれのサーバでの上限スループットの合計値 ($TP_{HighAll}$) で割ることにより現在の全体処理能力に対して掛かっている負荷量 (LO) を算出している．

$$LO = \frac{TP_{MAiAll}}{TP_{HighAll}} \tag{1}$$

負荷量がスケールアウトの閾値 (Th_{high}) を超えると現在のサーバ数では処理することができないと判断し，VC サーバを起動し，振り分けを開始する．スケールインの閾値 (Th_{low}) を下回ると余分な VC サーバが動作していると判断し，振り分けを停止し，VC サーバを停止する．

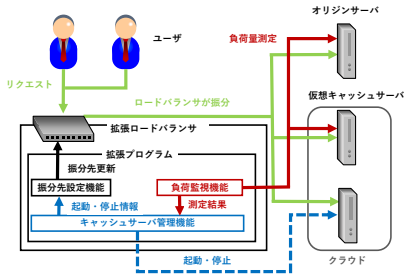

図 1: 分散 Web システムの概要図

3 動的閾値変更アルゴリズム

先行研究の評価実験では VC サーバの最大台数を 10 台，Th_{high} を 0.3，Th_{low} を 0.1，各サーバの上限スループットを 200 に設定し，秒間リクエスト数は 50 から 2000 まで増加させていた [1]．この時の最大リクエスト数を，最大台数でなければ処理できない値にしていたため，最大リクエスト数からの起動台数が理想値と一致していた．そこで，本研究では最大秒間リクエスト数を半分にして実験を行ったところ図 2 に示す結果が得られた．上限スループットと最大負荷量から起動されるサーバ台数は 6 台が適切であると考えられるが，図 2 から 10 台起動されていることが分かる．原因としては，Th_{high} が小さすぎたことであるが，負荷を確実に分散するために早期にサーバを起動しようとして小さな値になっていた．

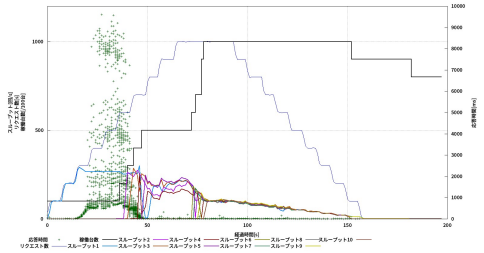

図 2: 先行研究のアルゴリズムを用いた実験の結果

現在の稼働台数に基づいて閾値を動的に変更すれば早期にサーバを起動しかつ余分なコストを減らすことができると考えた．この考えに基づき以下のアルゴリズムを開発した．現在の最大処理量から現在の負荷量を引いた差が上限のマージンより小さい場合スケール

アウトする．逆に，現在のサーバの 1 つが停止した時の最大処理量から現在の負荷量を引いた差が下限のマージンより大きい場合スケールインする．これらから，Th_{high} と Th_{low} を式 (2)，(3) で定義した．WS は現在のサーバの稼働台数，MG_{high} と MG_{low} はそれぞれ 1 台当たりの処理量で正規化した上限のマージンと下限のマージンである．稼働台数が変化するごとに閾値も変更されるので負荷に応じた数のサーバを起動できると考える．MG_{high} と MG_{low} の値の取る範囲を式 (4)，(5) に示す．MG_{low} はスケールイン後にスケールアウトがすぐに起きないように MG_{high} より大きくなくてはならない．

$$Th_{high} = \frac{WS - MG_{high}}{WS} \tag{2}$$

$$Th_{low} = \frac{WS - 1 - MG_{low}}{WS} \tag{3}$$

$$0 < MG_{high} < 1 \tag{4}$$

$$MG_{high} < MG_{low} < 1 \tag{5}$$

本研究では負荷増加率を 7 秒毎に 10 ずつ増える条件で，初期の閾値をそろえるため，MG_{high} を 0.7，MG_{low} を 1.0 に設定して実験を行った．結果を図 3 に示す．

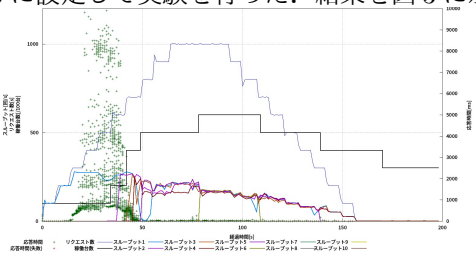

図 3: 提案したアルゴリズムを用いた実験の結果

結果から，固定値の場合最大 10 台サーバを起動していたが，稼働台数に基づいて動的に閾値を変更した場合最大 6 台と負荷に応じた台数を起動しており，余分なコストの削減が行えていることが確認できた．

4　停止待ち状態を追加したアルゴリズム

先行研究では，スケールインが発生するとサーバが停止され，スケールアウトが起きるとサーバが起動されていた．しかし，停止時には既に振り分けられたリクエストの処理を終えなければならないため，処理が終わるまで待ち (停止処理待ち期間)，停止を行っていた．処理が終わるまでにリクエストが増加し，スケールアウトが発生するとサーバが新しく起動を行っていた図 4．しかし，サーバを新しく起動すると起動までに時間がかかり迅速にリクエストの処理を行うことができない．停止処理待ち期間にスケールアウトが発生した時停止処理待ち期間中のサーバを稼働状態へと復帰させるようにした．これを実現するために図 4 の shutting を waiting と shutting に分割し，waiting から稼働状態への遷移を追加した (図 5)．このときの状態遷移図を図 5 に示す．

改良したアルゴリズムでの実験結果を図 6 に示す．図 6 より，丸がついている部分で改善が見られた．1 つ目の丸部分では，サーバの復帰が見られ応答時間の改善が見られる．2 つ目，4 つ目では，起動しすぎたサーバの削減によるコストの削減が見られる．3 つ目では，サーバの復帰が見られ応答時間の改善が見られる．以上のことから，応答時間の改善，コストの削減という改善が見られ，状態の追加の有効性を確認できた．

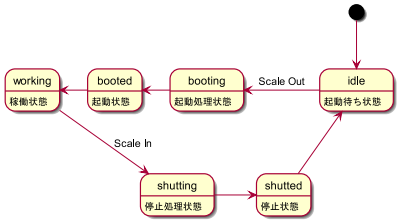

図 4: 先行研究における状態遷移図

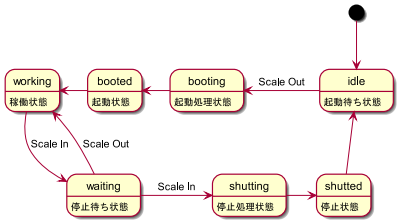

図 5: 停止待ち状態を追加した状態遷移図

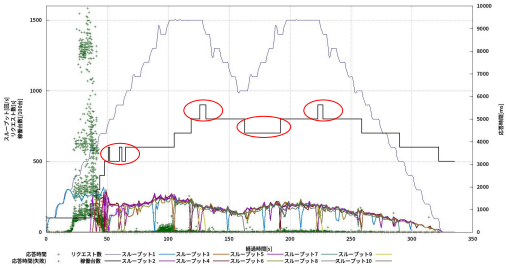

図 6: 停止待ち状態を追加した実験結果

5　まとめ

先行研究の分散 Web システムに実装されているアルゴリズムを再評価した結果判明した問題点を踏まえて，稼働台数に基づくオートスケールアルゴリズムと停止待ち状態を追加したオートスケールアルゴリズムを提案した．実験の結果，提案したアルゴリズムの有効性を確認できた．今後の課題として，性能の違うサーバに対してのアルゴリズムの検討，VC サーバを用いた評価実験，使用するサービスによって変動する上限スループットの推定などがある．

参考文献

[1] 松田正也，最所圭三，“クラウドを用いた分散 Web システムにおけるオートスケールアルゴリズムの改良と評価 “，香川大学修士 論文，2018.

[2] 堀内晨彦，“分散 Web システムにおけるオートスケールアルゴリズムの改良と評価 “，香川大学修士 論文，2016