クラウドに適した分散 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

13T264 松田 正也（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分散 Web システムにおいて, キャッシュサーバ数を管理する機能に実装しているオートスケールアルゴリズムを改良したアルゴリズムを提案し, その実装と評価について述べる.

1 はじめに

クラウド技術が発展し, 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が, 負荷量に応じて動的にキャッシュサーバ数を増減させなければ効果が薄い. そこで, 当研究室ではクラウド環境において負荷量に応じて動的に仮想キャッシュサーバ (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]. 本研究では, 先行研究で開発された提案システムの再評価を行い, 結果から見えてきた問題点を解決するオートスケールアルゴリズムを提案する.

2 分散 Web システムの概要

図 1 に分散 Web システムの構成図を示す. 本システムはソフトウェアロードバランサに以下の機能を追加実装した拡張ロードバランサと, キャッシュ元の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.

A 負荷監視機能: サーバの負荷量を監視

B キャッシュサーバ管理機能: 負荷量に応じた VC サーバの起動・停止

C 振分先設定機能: VC サーバ数の増減に合わせたアクセスの振分先更新

負荷量の監視および VC サーバの増減は拡張ロードバランサの機能で行い, リクエストの制御はソフトウェアロードバランサの機能を用いて行う.

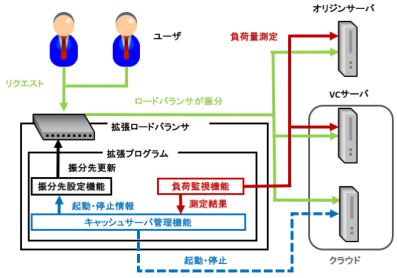

図 1: 分散 Web システムの概要

図 1 に示すキャッシュサーバ管理機能に実装されているオートスケールアルゴリズムでスケールアウトに用いる判別式を式 (1) に, スケールインに用いる判別式を式 (2) に示す. 式 (1) では, $AVGOR_t$, $AVGOR_{t-9}$, Th_{high} , N をそれぞれ現在の負荷量, 9 秒前の負荷量, スケールアウト時の閾値, 稼動台数として起動する台数 M を求める. 式 (2) では, $AVGORS_6$, Th_{low} , N , $Margin$ をそれぞれ最新 6 秒間の平均稼働率の合計値, スケールインの閾値, 現在のサーバ台数, 測定誤差の影響による停止直後の再起動を抑えるためのマージンとして, スケールインを行うかどうかを決定する.

$$M = \frac{(AVGOR_t + \frac{AVGOR_t - AVGOR_{t-9}}{9} \times S) \times N}{Th_{high}} - N \quad (1)$$

$$\frac{AVGORS_6}{6} < Th_{low} \times \frac{N-1}{N} - Margin \quad (2)$$

3 合計稼働率を用いるオートスケールアルゴリズム

先行研究ではオートスケールアルゴリズムの評価実験において, VC サーバを起動してから振り分けを開始するまでにかかる時間として設定している時間 S と同一である 30 秒間隔で, リクエストを増減させていた [1]. これに対して, 負荷増加率を倍にした 15 秒間隔で, リクエスト数を増減させた実験結果を図 2 に示す. リクエスト増加中において平均稼働率が上限である 1.0 に達している箇所が複数あり, 事前起動が間に合っていないといえる. 平均稼働率を用いたアルゴリズムでは, 9 秒前の平均稼働率と現在の平均稼働率の差分から負荷上昇率を求めるため, 9 秒前の稼働台数が無視されてしまうことにより, 負荷上昇率の計算に誤差が生じていると考えられる.

以上のことから平均稼働率の代わりに, 稼働台数を考慮に入れるために合計稼働率を用いたオートスケールアルゴリズムを提案する. 起動する台数 M は $TTLOR_t$, $TTLOR_{t-9}$, Th_{high}, N をそれぞれ現在の合計稼働率, 9 秒前の合計稼働率, スケールアウト時の閾値, 稼動台数として式 (3) で求める. 合計稼働率を用いることによって, 式 (3) の右辺の第 1 項の分子

で, S 秒後の負荷量を正しく計算することができると考えた.

$$M = \frac{TTLOR_t + \frac{TTLOR_t - TTLOR_{t-9}}{9} \times S}{Th_{high}} - N \quad (3)$$

スケールアウトアルゴリズムと同様にスケールインアルゴリズムも合計稼働率を用いる. $TTLORS_6$, Th_{low} , N , $Margin$ をそれぞれ最新 6 秒間の合計稼働率の合計値, スケールインの閾値, 現在のサーバ台数, 測定誤差の影響による停止直後の再起動を抑えるためのマージンとして以下の式 (4) によりスケールインを行うかどうか決定する.

$$\frac{TTLORS_6}{6} < Th_{low} \times N - Margin \quad (4)$$

図 3 に提案アルゴリズムの評価を行った結果を示す. 図 2 と比較して, 図 3 ではリクエスト増加中の区間において平均稼働率が 1.0 に達している区間が少ないことから, 平均稼働率に基づくオートスケールアルゴリズムと比較して, 合計稼働率に基づくオートスケールアルゴリズムの方が優れていることが確認できた.

4 台数依存オートスケールアルゴリズム

先行研究ではおおよそ期待通りの動作をしていたため, スケールアウトのトリガーとなる閾値 (負荷量の上限値) を 0.8 に固定して評価していた [1]. しかし, 閾値に依存して VC サーバの稼働台数や稼働率, クライアントへの応答時間, 運用コストなどが影響を受けると考えられるため, スケールアウトに用いる閾値とスケールインに用いる閾値を変動させて実験を行い, 応答時間と運用コストの観点から再評価を行った. 詳細は省くが, スケールアウトの閾値は小さく設定した場合に比べて, 大きく設定した場合の方がコストパフォーマンスが高いことがわかり, スケールインの閾値は応答時間やコストへの影響はあるが, スケールアウトの閾値と比べると影響は小さいことがわかった. このことと稼働台数がオートスケールに与える影響を考慮して, 現在のサーバ稼働台数に応じて各閾値を変動するオートスケールアルゴリズムを提案する. スケールアウトアルゴリズムは式 (3) を使用し, スケールインアルゴリズムでは式 (4) から, Th_{low} として Th_{high} から Th_{low} との差分である $Diff$ を引いた式 (5) を用いる.

$$\frac{TTLORS_6}{6} < (Th_{high} - Diff) \times N - Margin \quad (5)$$

最大で同時に起動できるサーバ台数に対する現在起動している台数の割合における Th_{high} の設定値を表 1 に示す. 最も低い Th_{high} を 0.5 に設定しているが, これはスケールアウトに用いる閾値の評価実験から, 0.4 以下に設定した場合のコストパフォーマンスは低いと判断したためである.

$Diff$ を 0.0~0.4 まで変化させてアルゴリズムを評価するための実験を行った結果をコストと平均応答時間についてまとめたグラフを図 4 に示す. 実験結果より,

表 1: 各台数割合における閾値

台数	~3	3~5	5~7	7~9	9~10
Th_{low}	0.5	0.6	0.7	0.8	0.9

$Diff$ を 0.2 より大きくしてもあまり効果が得られないことがわかり, 台数依存オートスケールアルゴリズムにおいて, $Diff = 0.2$ に設定した場合が最もコストパフォーマンスが良いとわかった. 今回の実験では, $Diff$ を固定値としたが稼働台数の変化に合わせて $Diff$ を変動させることでさらに不要なコストを削減できると考えられる.

5 まとめ

分散 Web システムに実装されているオートスケールアルゴリズムについて再評価し, 判明した問題点を踏まえて合計稼働率に基づくオートスケールアルゴリズムと台数依存オートスケールアルゴリズムを提案・評価し, 有効性を確認した. 今後の課題として, キャッシュサーバ管理機能の改良やヘテロなクラウド環境への対応, 利用時間料金に基づくコスト最小化などがある.

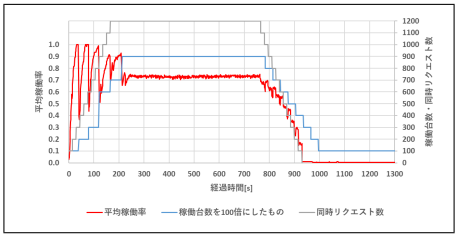

図 2: 平均稼働率 (平均稼働率方式, 15 秒間隔)

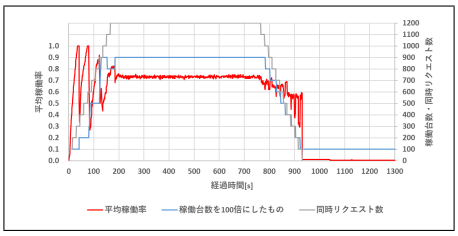

図 3: 平均稼働率 (合計稼働率方式, 15 秒間隔)

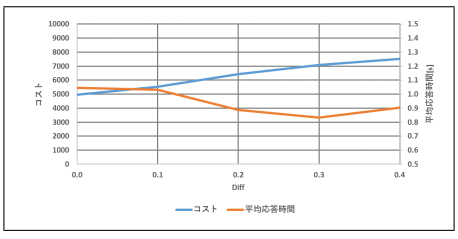

図 4: 台数依存オートスケールアルゴリズムの実験結果

参考文献

[1] 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