Vol.2016-OS-136 No.16
2016/3/1

IT システムに対する急激な負荷上昇の早期検出方式の評価

横山晴道[†1]　菅原智義[†1]

概要：今後は、数百億ものデバイスがインターネットに接続される Internet of Things (IoT) を想定した IT システムが求められる。IoT 用のサーバーは数秒間で急激に上昇する負荷に対応するための高いスケーラビリティを持つ必要がある。このような高いスケーラビリティを要求されるシステムでは、クラウドが利用されてきたが、クラウドのスケーラビリティは数時間～数分という長時間での負荷変化を対象にしており、IoT システムで求められるような数秒間で急激に上昇する負荷には対応できないという問題があった。我々は、この問題を解決するために、制御工学の分野で発展してきたカルマンフィルタを用いた負荷上昇検出方式を開発した。本方式を適用した場合、数秒以内の負荷上昇に追従することができ、システムの応答時間を保証することができた。

キーワード：リソース管理、負荷上昇検出、クラウド、IaaS

1. はじめに

将来、Internet of Things (IoT)の普及により、通信機能を持つデバイスの総数は指数的に増大し、2020 年には 500 億個もの IoT デバイスがインターネットに接続されると予測されている[6]。社会インフラなどで用いられる IoT システムでは、事故などの事象が発生すると、多くのデバイスが特定のサーバーに向けて一斉に処理を要求する。このため、IoT 用のサーバーは数秒間で急激に上昇する負荷に対応するための高いスケーラビリティを持つ必要がある。

従来、このような高いスケーラビリティを要求されるシステムでは、クラウドが利用されてきた。クラウドでは、利用者がみずから物理的なサーバーを用意する必要がなく、負荷に合わせてスケールイン・スケールアウトができるという利点がある。しかし、従来のクラウドのスケーラビリティは数時間から数分という長時間での負荷変化を対象にしており、IoT システムで求められるような数秒間で急激に上昇する負荷には対応できないという問題があった。

そこで、我々は、数秒間で急激に上昇するような負荷変化に対して、応答性能が劣化する前に負荷上昇を検出し、構成変更を行う負荷上昇検出方式を提案する。本方式の特徴は、制御工学の分野でロケットの軌道推定などに使われるカルマンフィルタを用いて、測定された負荷値をフィルタリングして、ばらつきの大きい測定値から負荷上昇の傾向を早期に正確に検出できることである。負荷上昇を想定したリクエストを与え、本方式を適用した場合、数秒以内の負荷上昇に追従することができ、システムの応答時間を保証することができた。

†1 NEC グリーンプラットフォーム研究所

2. 背景

2.1 クラウドにおけるシステム構成変更

クラウドサービスでは負荷の時間変化を検出して、システムの構成を自動的に変更する技術が存在する。例えば、Amazon Web Service (AWS) では、リソース使用量に合わせて起動するインスタンスの数を変更する auto scaling と呼ばれる仕組みが導入されている[7]。auto scaling において負荷の測定を行う Amazon CloudWatch は、基本モードでは 5 分、詳細モードでは 1 分の測定時間間隔が設定されている。また、新しい AWS インスタンスの起動には数分を要すると言われている。提供されているサービスの性質から、AWS の auto scaling では数時間から数分で負荷が増加するような状況を想定していると考えられる。

2.2 負荷分析の従来技術

AWS auto scaling においては測定した負荷が所定のしきい値を超えているか否かによって、構成変更を実施する。これは最も単純な方法である。これに対して、移動平均法では過去の値を平滑化して現在の負荷値を推定するために、突発的な変化に過敏に反応しないという特徴がある。移動平均は、クラウドでの構成変更を決定するときの負荷分析方法として頻繁に利用されている[8,9]。

2.3 急激な負荷上昇を検出するための要件

本研究では、システムへの負荷が数秒オーダーで増加するような状況で、早期に負荷上昇を検出し、構成変更を行うことを目的とする。

数秒で負荷が増加する場合、一般的に行われる分オーダーの時間間隔で負荷測定を行うと、数秒オーダーでの負荷上昇に即時に追従することができない 。そのため、秒もしくはサブ秒といった時間間隔でシステムの負荷を測定する必要がある。負荷測定の時間間隔を短くすると、結果として測定値の分散（ばらつき）が大きくなるという問題があ

る。負荷測定値の分散が大きいとき負荷上昇の判定をすると、負荷上昇が起こっておらずリソース追加の必要がないときでも負荷上昇と判断してしまう。したがって、負荷上昇を早期に検出するために、負荷の測定を短い時間間隔で行うことと不必要な構成変更を行う確率を低く抑えることを両立する負荷上昇検出方式が必要である。

3. 提案方式

3.1 提案方式の概要

上述の要件を実現するために、本研究では、負荷の測定を短い時間間隔で行うことと、不必要な構成変更を行う確率を低く抑えることを両立する負荷上昇検出方式を確立する。

負荷が上昇していると判断するためには、現在の負荷の値のみに基づいて判断するのではなく、負荷の変化傾向を推定して判断を行う必要がある。しかし、負荷の測定値のばらつきが大きい場合は、その変化傾向を推定することが難しい。そこで、我々は、制御工学で用いられているカルマンフィルタを元に、負荷変化を推定してその上昇傾向を判断する方式を提案する。

カルマンフィルタは制御工学分野で研究された信号処理技術であり[10]、誤差のある測定値から、時々刻々と変化するシステムの状態を推定するために用いられる。ばらつきのある負荷測定値の変化傾向を推定するという目的に適していると考えられるため、提案方式の一部に採用した。

本提案方式は、1 ステップごとの負荷測定値をカルマンフィルタの入力として、カルマンフィルタの内部状態を更新し、現在の負荷および変化量の推定値を出力する。出力値を組み合わせ所定のしきい値と比較することによって、負荷上昇をしているか、つまり、構成変更を行うか否かの判断を行う。

3.2 カルマンフィルタによる負荷上昇検出方式

本方式では、線形カルマンフィルタを用いて負荷測定値から推定値を計算する。

線形カルマンフィルタについて説明する。線形カルマンフィルタの状態方程式、観測方程式は下記の式であらわされる。観測方程式は内部システムの状態が測定値にどのように反映されるかを記述する式であり、状態方程式は内部システムの時間変化を推定する式である。

線形カルマンフィルタの観測方程式および状態方程式は、

$$Y_n = CX_n + v_n$$
$$X_{n+1} = AX_n + w_n$$

ここで、Y_nは n 回目の測定値、X_nは n 回目のシステム状態を表す値、v_nは測定に関わるノイズ、w_nはシステムに乗るノイズである。X_n, Y_n, v_n, w_n は測定値が 2 つ以上であれば、スカラーではなくベクトルである。そのときC, A は行列になる。また、ノイズv_n, w_nを平均が 0 の正規分布$N(0,\sigma)$に従うものとして考える。

$$v_n \sim N(0,R)$$
$$w_n \sim N(0,Q)$$

ここで、RとQはそれぞれv_n, w_nの分散の大きさを決めるパラメータである。

n 回目に測定値Y_nが得られた時の、推定値X_nは以下のように計算される。

$$\hat{X}_n = AX_n$$
$$\hat{P}_n = Q + AP_nA^T$$
$$K_n = \hat{P}_nC^T\left(C\hat{P}_nC^T + R\right)^{-1}$$
$$X_n = \hat{X}_n + K_n(Y_n - C\hat{X}_n)$$
$$P_n = (\mathbf{1} - K_nC)K_n$$

P_nはノイズの共分散行列である。$\hat{X}_n$は事前推定値とよばれ、測定値が得られるまえの予測値である。本方式では、カルマンフィルタの入力値Y_nを、現在の測定値および現在の測定値とその一つ前の測定値との差分の 2 次元ベクトルとする。つまり、システム負荷の測定値の系列 $U_1, \ldots, U_{n-1}, U_n$に対して、

$$Y_n = (U_n, U_n - U_{n-1})$$

とする。また、カルマンフィルタの出力値 X_n も現在の負荷の推定値および負荷変化量の推定値の 2 次元ベクトルとする。

$$X_n = (x_n, \Delta x_n)$$

ここで、x_n は現在の負荷の推定値、Δx_n は現在の負荷変化量の推定値である。X_n, Y_nをこのように見た時、式の係数A, Cは以下のように決まる。

$$A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

ただし、Δt は測定の時間間隔を表す。

負荷上昇検出方式は、上述の線形カルマンフィルタに基づき、負荷および変化量の推定値から将来の予測負荷値を計算し、しきい値と比較することによって負荷上昇をしていると判断する。構成変更が必要だと判定する負荷指標Sを出力値$X_n = (x_n, \Delta x_n)$から計算する。

$$S = x_n + \gamma \Delta x_n$$

ただし、γはあらかじめ決めておくパラメータである。そして、Sがあらかじめ定めたしきい値S_{thr}を超過した場合に負荷上昇していると判断して、構成変更を行う。

4. 実装

4.1 概要

上述のカルマンフィルタによる負荷上昇検出方式を、負荷測定機能および構成変更機能とともに実装した。実装したシステムの構成を図 1 に示す。

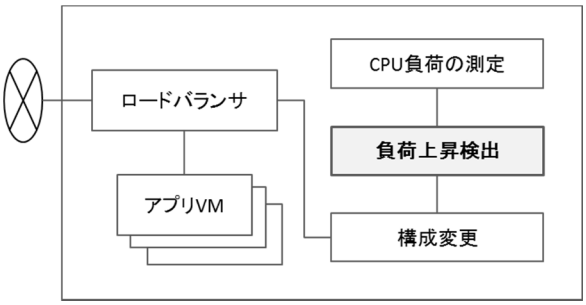

図 1　実装したシステムの構成

本実装システムではハイパーバイザ上に実装を行った。ハイパーバイザの仮想マシン（Virtual Machine, VM）は複数あり、VM 上ではステートレスな同一アプリケーションを実行させた。

本実装システムは、先の章で述べた負荷上昇検出方式を機能の中核に据える。一定の時間間隔で測定したゲスト OS の CPU 使用率をもとに、負荷上昇しているかどうかの判定を行い、判定がなされたときは構成変更を行う。

本実装システムは負荷上昇検出を行うためにゲスト OS の CPU 使用率を測定する。ハードウェアカウンタを利用する Linux perf を使用した。ハイパーバイザは一定の時間間隔で CPU 使用率の測定を行い、負荷上昇検出方式の入力値として使用する。

負荷が上昇していると判断されたあと、HTTP リクエストを処理するリソースを増やすために、構成変更を行う。本実装システムは外部からの HTTP リクエストを受け取り、ゲスト OS において処理を実行する。ハイパーバイザは外部のネットワークと接続されており、ひとつの仮想 IP アドレスを外部に見せている。ハイパーバイザはロードバランサを動作させている。ロードバランサは仮想 IP アドレスへの HTTP リクエストを、ゲスト OS のアプリケーションへと振り分ける。リクエスト振り分けは、ラウンドロビン方式で行われる。ハイパーバイザは設定されているゲスト OS をすべて起動させた状態で HTTP リクエストを受け付ける。リクエストの数が少なく低負荷のとき、ロードバランサは 1 つのゲスト OS に対してのみリクエストを振り分ける。

負荷が増加し、負荷上昇検出方式によって構成変更が必要と判断されたとき、ロードバランサの設定を変更して複数台のゲスト OS にリクエストを振り分ける。負荷の時間変化に応じて振り分け先のゲスト OS の台数を柔軟に変えることができる。

4.2 システムの負荷測定

本実装システムにおける、アプリケーションの負荷の測定方法について詳細に説明する。本研究ではアプリケーションの負荷を表す指標として、CPU 使用率を採用した。

CPU 利用率測定のための機能を今回の実装では Linux perf を用いて実装した。CPU 使用率を測定する方法としては、/proc 以下のファイルを read する方法がよく用いられるが、我々の実験の結果、測定のオーバーヘッドが大きくなるという問題が見られ、1 秒以下の時間間隔で測定を行うことが困難であった。

ハイパーバイザは、Linux perf を用いてアプリケーションが動作しているゲスト OS プロセスの task-clock を測定する。Linux perf では最低 0.1 秒、0.1 秒刻みで測定の時間間隔を設定することができる。CPU 使用率は、Linux perf で測定された task-clock を測定時間間隔で割った値として計算される。

Linux perf はハードウェアカウンタを利用しているために、オーバーヘッドが小さく、0.1 秒間隔で測定を行った時でもそのオーバーヘッドはアプリケーションの処理に影響を与えないことがわかったので、測定方法として Linux perf を採用した。

5. 評価

本論文で提案する負荷上昇検出方式について、測定を短い時間間隔で行うことと不必要な構成変更を行う確率を低く抑えることを両立することを確認するために、提案方式と既存方式の性能の比較評価を行った。

5.1 評価環境

評価環境は、図 2 で示されるようにクライアントとサーバーの 2 台の物理マシンによって構成される。クライアントマシンは、サーバーに対して HTTP リクエストを送り負荷をかける機能と、それぞれのリクエストに対して応答時間を測定する機能をもつ。サーバーマシンは、先に述べた負荷測定、負荷上昇検出、構成変更という機能をもったシステムを動作させる。

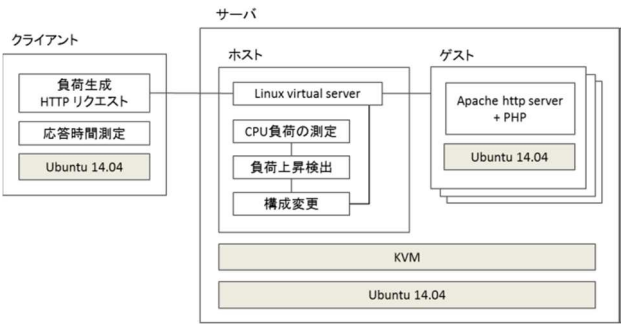

図 2　評価環境。負荷上昇検出機能を実装したサーバーマシンとクライアントマシン

評価に利用したマシンの仕様を説明する。クライアントマシンとサーバーマシンは共に NEC 製の Express5800/120Ei であり、CPU は Intel Xeon 3 GHz 4 コア，メモリ 8 GB，ディスクは HDD 450 GB である。OS は Ubuntu server 14.04,

Linux 3.13.0 を使用した。

サーバーマシンの仮想化環境について説明する。ハイパーバイザは KVM、ゲスト OS は Ubuntu server 14.04, Linux 3.13.0 を使用した。vCPU の個数は 1 つで物理マシン上の特定のコアに固定する。VM に割り当てたメモリは 1 GB, ディスク容量は 5 GB である。

サーバーマシンに実装したシステムの詳細について述べる。ロードバランサとして Linux virtual server (LVS)を使用した。ハイパーバイザが起動している VM の数は 3 つであり、低負荷時、ロードバランサはそのうちの 1 つにのみリクエストを振り分けている。サーバーが負荷上昇を検出したとき、ロードバランサのリクエスト振り分け先の設定を 1 台から 3 台に増加させる。ゲスト OS において、Apache http server と CPU を集中的に利用する PHP プログラムを用いてアプリケーション実行環境を構築した。

負荷上昇検出方式のパラメータの値について述べる。提案方式の測定ノイズv_n、システムノイズw_nの分散の大きさを決める行列Q, Rは

$$Q = \begin{pmatrix} 1.0 & 0 \\ 0 & 1.0 \end{pmatrix}$$

$$R = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}$$

と決めた。また、パラメータγは 0.3 と決めた。

5.2 負荷生成

自作した負荷生成プログラムにより、サーバーマシンに対して負荷をかけた。一般的には apache bench や jmeter、もしくは Rubis などの Web ベンチマークアプリケーションに付属したクライアントツールをもちいて、web アプリケーションの負荷テストを行うことが多い。しかしながら、本提案方式は数秒で増加する負荷を検出するため、その評価を行うためには負荷上昇の時間変化をある程度自由に調整できる必要がある。想定しているような負荷上昇を生成できるツールが存在しなかったため、プログラムを自作して評価に用いた。

負荷生成プログラムの特徴は、個々のリクエストを送出する時間差を指数分布により生成して、時間差の平均値(すなわち対応するポワソン分布でいえば、単位リクエスト数の平均値）を時間によって自由に変化させることができるという点である。負荷生成プログラムによるリクエストの時刻分布を図 3 に例示した。プログラムは個々のリクエストごとに TCP の接続を行う。負荷が急激に上昇する状況では、新規接続が多く発生することが考えられるので、個々のリクエストごとにセッションを開始するという方法を取った。また、それぞれのセッションはノンブロッキングに接続されるため、先のリクエストの応答を待たずに後続のリクエストを送る。

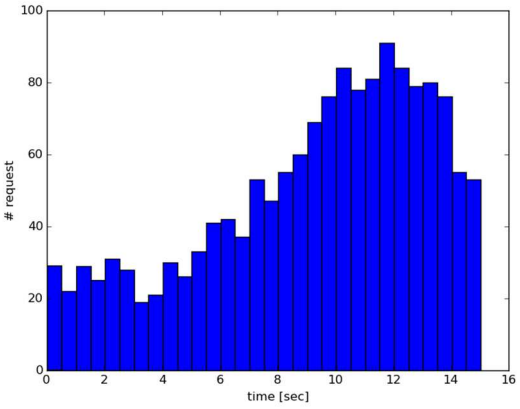

図 3 負荷生成プログラムにおける
HTTP リクエストの時刻の分布

5.3 既存方式

負荷の上昇を検出するための最も単純な方法は、単一のしきい値を設けて、それを超過した場合に負荷上昇と判断することである。AWS の auto scaling ではこのような方法でインスタンス増減がなされている。

ばらつきのある測定値を平滑化させる簡易な手法として、移動平均法がある。これはひとつ前の推定値と現在得られた測定値から、適切な重みづけを行って推定値を計算する手法である。U_nを n 回目の測定値、Y_nをU_nから計算された推定値、$a\ (0 < a < 1)$を計算するときの重み、とすると移動平均による推定値は、

$$Y_n = aU_n + (1-a)\,Y_{n-1}$$

と計算される。パラメータaは、今回の評価において$a = 0.4$とした。Y_nを構成変更が必要だと判定するときの負荷指標とし、所定のしきい値を超えた否かを判定の基準とする。

5.4 評価結果

急激な負荷変化を早期に検出できることを示すために、リクエスト数が増加してからサーバーが負荷上昇を検出し構成変更を始めるまでの時間を評価した。CPU 使用率を測定する時間間隔は 0.5 秒とした。構成変更を決定する負荷指標のしきい値は、提案方式と移動平均を用いた判定手法は 0.7（70%)、しきい値を用いた判定方式は 0.8（80%）である。負荷に関しては、20 秒までは平均毎秒 40 リクエスト、5 秒かけてリクエスト数を増加させ、25 秒の時点では平均毎秒 140 リクエストになるように、リクエストの時刻列を生成した。生成したリクエストの時刻列に従いサーバーのプログラムに負荷をかけた。

サーバーへの負荷かけを 100 回行い、構成変更を始めるまでの時間の平均値を算出した。表 1 はそれぞれの方式における時間の平均値を示している。また、提案方式および移動平均を用いた判定方式について、リクエストを増加させ

てから構成変更を行うまでの時間分布を、図 4 に示した。提案方式は、しきい値による判定方式より 3.6 秒、移動平均をもちいた判定方式より 0.8 秒早く構成変更を始めることができるとの結果を得た。

表 1　負荷上昇から構成変更を行うまでの時間の比較

しきい値による判定方式	7.3 秒
移動平均を用いた判定方式	4.5 秒
提案方式	3.7 秒

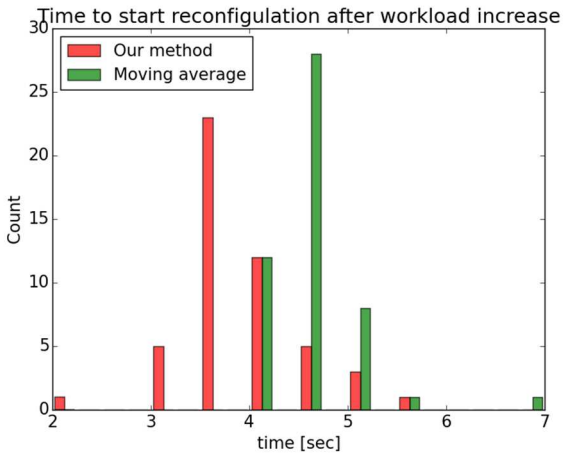

図 4　リクエスト増加後から構成変更を行うまでの時間分布

提案方式が測定値に対してより早く追随していることを示すために、負荷上昇を判定するための負荷指標について確認する。図 5 に、提案方式と移動平均を用いた判定方式で計算された負荷指標の時間変化の一例を示した。青色の線がサーバーにおける CPU 使用率の測定値を表している。測定の時間間隔は 0.5 秒である。時刻 20 秒の時点でリクエストを増加させ、4 秒後の時刻 24 秒には CPU 使用率は 80%程度に上昇している。緑の点が提案方式、赤の点が既存方式によって計算された負荷指標を示している。このグラフから、提案方式のほうが、負荷上昇に対してより早く、正確に追従していることが読み取れる。

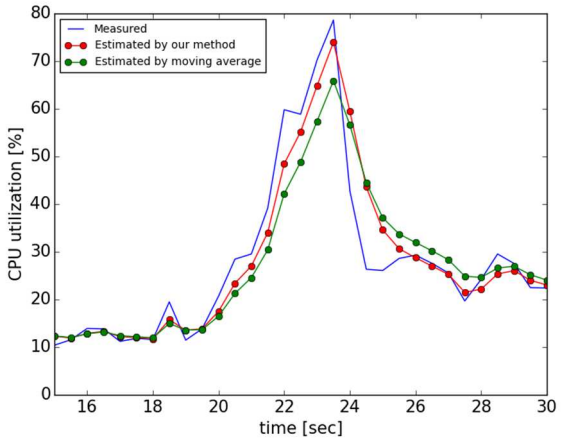

図 5　サーバーに負荷をかけた時の CPU 使用率の時間変化と各方式による負荷推定値の時間変化

負荷上昇検出の早さが応答時間の劣化を抑止していることを確かめるために、構成変更を行ったときの応答時間の変化を比較する。図 6、図 7 に、サーバーで測定された CPU 使用率、および提案方式および移動平均を用いた判定方式を用いて構成変更を行った時の応答時間の時間変化を示した。図 6 の赤線で示されたグラフが提案方式を用いたときの応答時間であり、図 7 の緑線で示されたグラフが移動平均による判定方式を用いた応答時間である。それぞれ構成変更を始めた時間を垂直の破線で示した。提案方式の場合は 23.5 秒、移動平均を用いた判定手法の場合は 24.5 秒に構成変更を始めた。提案方式の方がより早く構成変更しているために、過負荷時の応答時間の劣化をより小さく抑えられていることがわかる。

提案方式では、負荷検出が早いことに加えて、不必要な構成変更を行う確率が小さくなることを示す。そのために、負荷上昇後でも応答時間が保証されている割合 Service level agreement (SLA rate) および負荷上昇の誤検出率 False alert rate を用いて評価を行った。２つの指標について定義する。Service level agreement rate（SLA rate）は、負荷上昇が始まってからのリクエストの応答時間のなかで、所定のしきい値を超過しているものの割合である。SLA rate は、方式が負荷上昇を早くに検出し、構成変更を行えていることを表す指標である。今回、SLA rate を計算するときの応答時間のしきい値を 0.2 秒とした。

False alert rate は、構成変更が必要ない程度に低い負荷をかけたときに、誤って構成変更を行ってしまう割合である。False alert rate は測定に基づき負荷上昇だと判定された回数をすべての測定の回数で割った値である。評価する際のサーバーの測定間隔は、0.2 秒間隔とした。評価では、クライアントは毎秒 90 リクエストを生成する。リクエストを処理するサーバーの CPU 使用率の平均は 48%、標準偏差は（絶対値で）14%だった。False alert rate 評価時の、サー

バーで測定された CPU 使用率の時間変化の一例を図 8 に示した。

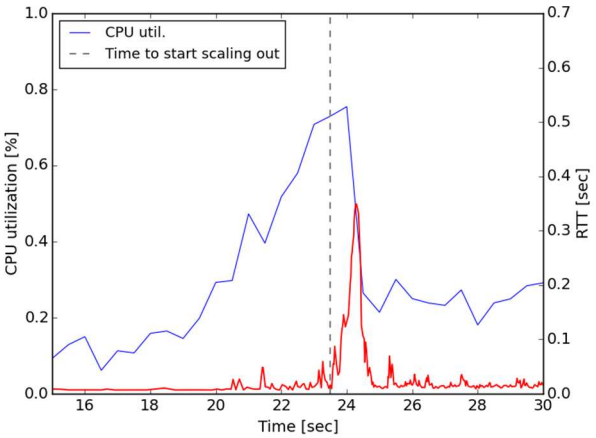

図 6　CPU 使用率と提案方式によって構成変更を行った時の応答時間の時間変化

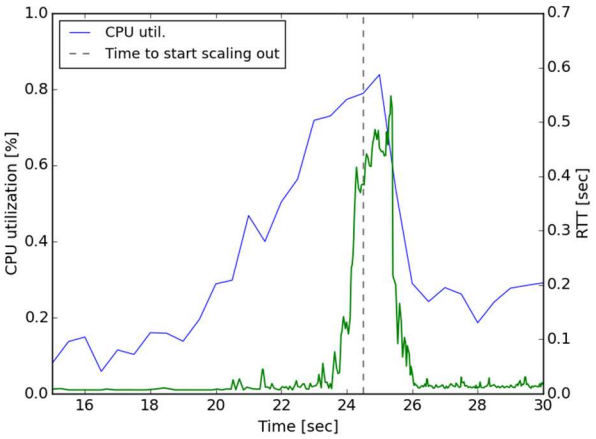

図 7　CPU 使用率と移動平均による判定方式で構成変更を行った時の応答時間の時間変化

図 9 には提案方式の SLA rate と False alert rate の評価結果を示した。赤線は提案方式であり、緑線は移動平均を用いた判定方式の結果である。図 9 では、構成変更を行う時の負荷指標のしきい値を 0.67, 0.7, 0.73 と変化させながら 3 点プロットした。しきい値を上げると、False alert rate が向上し SLA rate が悪化するというトレードオフの関係をみることができるが、いずれのしきい値の場合も提案方式のほうが、SLA rate は大きく、False alert rate の方が小さいという結果が得られた。今回の評価では、False alert interval が同一になるようにそれぞれのしきい値を調整する場合、3%から 7%程度 SLA が向上することがわかった。

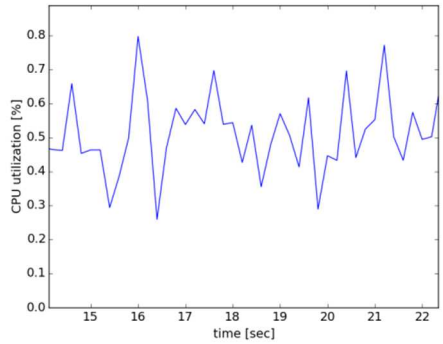

図 8　False alert rate 評価時のサーバーでの CPU 使用率

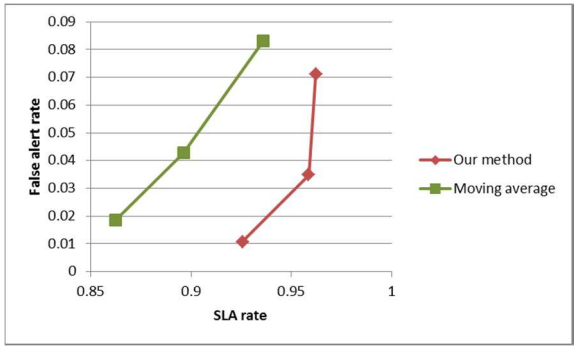

図 9　SLA rate と False alert rate のプロット

5.5 考察

提案方式を評価した結果について考察を述べる。カルマンフィルタを利用した提案方式は、0.5 秒間隔でシステムの負荷を測定したときに、既存方式よりも負荷上昇を早期に検出できることが判明した。カルマンフィルタにより負荷量および負荷変化量を推定し、負荷上昇後すぐに傾きが大きくなったことを検出したことによって、このような早期の負荷上昇検出が可能になったと考えられる。

提案手法では、負荷上昇をより早期に検出し、かつ、誤検出率を低下させることができた。先に述べた通り、提案方式で利用したカルマンフィルタは負荷変化量を推定しており、平均的に見れば負荷が変化していないときに負荷変化量を 0 に近い値に推定する。よって、提案手法は一過的な負荷変化に過敏に反応しないという性質を持つことが性能差の要因となっている。したがって、負荷の測定を短い時間間隔で行うことと不必要な構成変更を行う確率を低く抑えることを両立することができた。

6. 関連研究

自律コンピューティング分野では、IT システム上の環境変化を自動的に察知して、オペレータの介入なくシステムの構成を適切に変更する技術が研究されてきた[3,4,5]。こ

れらの研究では IT システムへの負荷と応答性能の関係をモデル化して、応答性能が劣化すると判断されたときにリソース追加を行う。しかしながら、想定している時間スケールが短くても分程度であるため、提案方式で行っているように負荷の変化傾向まで考慮してはいない。また、学習を使っている手法などは計算コストが大きく、秒オーダーの測定間隔では実用的ではないと考えられる。

過去の負荷の系列を学習して、将来の負荷傾向を予測する技術についても研究が盛んである。[1]では、ベイズ統計と過去データに基づいた適切な特徴量の選択によって、Google のコンピューティングクラスタの長期的な負荷変化を精度高く予測している。また、[2]では過去数分の負荷パターンをウェーブレット解析により独立成分に分解したうえで、先の時間方向へ外挿し重ね合わせることによって将来の負荷を予測している。さらに負荷の増加にあわせて VM を増減させ、効率的なリソース利用を図っている。これらの予測手法は、負荷に一定の規則性・パターンが存在する場合のみ有効であり、数秒で発生する負荷上昇では規則性・パターンが見いだせる可能性は低いように思える。ゆえに、本研究では長い過去データには依存しない制御工学の手法を採用した。

7. まとめ

IoT システムで求められるのは、急激に上昇する負荷に対応するための高いスケーラビリティである。クラウドサービスのスケーラビリティは数時間～数分という長時間での負荷変化を対象にしており、IoT システムで想定される負荷上昇には対応できないという問題があった。特に短い時間で負荷の測定を行うと、測定値のばらつきが大きくなり不必要な構成変更を行う確率が大きくなるという課題がある。我々はこの課題を解決するために、制御工学の分野で発展してきたカルマンフィルタを用いた負荷上昇検出方式を研究開発した。提案方式の特徴は、ばらつきのある測定値から負荷値とその変化量を推定して、負荷上昇の指標として用いることである。本方式を適用した場合、数秒以内の負荷上昇に追従し構成変更を行うことができ、システムの応答時間を保証することができた。

参考文献

[1] Di Sheng, Derrick Kondo, and Walfredo Cirne. "HOSt load prediction in a Google compute cloud with a Bayesian model." Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012.
[2] Nguyen Hiep, et al. "Agile: Elastic distributed resource scaling for infrastructure-as-a-service." Proc. of the USENIX International Conference on Automated Computing (ICAC'13). San JOSe, CA. 2013.
[3] Gong Zhenhuan, Xiaohui Gu, and John Wilkes. "Press: Predictive elastic resource scaling for cloud systems." Network and Service Management (CNSM), 2010 International Conference on. IEEE, 2010.
[4] Shen Zhiming, et al. "Cloudscale: elastic resource scaling for multi-tenant cloud systems." Proceedings of the 2nd ACM SympOSium on Cloud Computing. ACM, 2011.
[5] Han Rui, et al. "Lightweight resource scaling for cloud applications." Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International SympOSium on. IEEE, 2012.
[6] http://www.cisco.com/web/JP/tomorrow-starts-here/files/14.4_trillion_whitepages.pdf
[7] https://aws.amazon.com/jp/
[8] Londoño-Peláez Jorge M., and CarlOS A. Florez-Samur. "An Autonomic Auto scaling Controller for Cloud Based Applications." International Journal of Advanced Computer Science & Applications 4.9 (2013).
[9] Xiao Zhen, Weijia Song, and Qi Chen. "Dynamic resource allocation using virtual machines for cloud computing environment." Parallel and Distributed Systems, IEEE Transactions on 24.6 (2013): 1107-1117.
[10] Kalman Rudolph Emil. "A new approach to linear filtering and prediction problems." Journal of Fluids Engineering 82.1 (1960): 35-45.