Title	ファジィエントロピーを用いたスライディングウィンドウ回帰に基づくコンテナオートスケーリングシステム
Author(s)	横山, 尚弥
Citation	
Issue Date	2023-09
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/18762
Rights	
Description	Supervisor: 田中 清史, 先端科学技術研究科, 修士(情報科学)

修士論文

ファジィエントロピーを用いたスライディングウィンドウ回帰に基づくコンテナオートスケーリングシステム

横山 尚弥

主指導教員　田中 清史

北陸先端科学技術大学院大学
先端科学技術研究科
(情報科学)

令和５年９月

Abstruct

When offering services such as e-commerce on the cloud, there is a necessity to modify the amount of server resources provided in accordance with the irregularly increasing and decreasing traffic. This need arises when there's a desire to maintain a constant level of service while, at the same time, doing one's utmost to restrain costs.

There is an abundance of prior research in fields like time series forecasting and load prediction. Many of these approaches rely on traditional time series forecasting, which necessitates that the data used for learning adhere to stationary or unit root processes, or they use deep learning approaches that involve using a vast amount of data and parameters. Given the high demand for data and time in learning and inference, it proves difficult to provide the necessary server resources for burst traffic, which can drastically increase or decrease within a span of minutes.

On the other hand, there are preceding studies that use approaches such as sliding window learning, which involve partitioning data finely and repeatedly conducting learning and inference in a sequential manner. Existing auto-scaling methods that apply this sliding window learning take into consideration not only the most recent load but also burst traffic, thus providing server resources needed in the near future.

In this study, we propose a traffic forecasting method that involves dynamic window size changes that can follow even slight trend changes. This method is based on regression estimation using sliding window learning and incorporates burst traffic detection using fuzzy entropy. Furthermore, we propose a new auto-scaling system that applies decision regression trees to multiple load metrics.

As a platform for operating this system, we adopted container virtualization technology. Container virtualization is a technology that allocates server resources independent from other processes for a single process. It enables the flexible allocation of computer resources on the server to processes as needed. Compared to the conventional method of building one application server per virtual machine, using container virtualization can reduce the start-up time of an application server to a matter of seconds.

In the evaluation, we conduct four comparative experiments using actual traffic rather than simulations. In the comparative experiments, we use traffic data publicly available on the web to reproduce traffic patterns with a load generator and output to the system under experiment. As baseline methods, we compare with two prior studies and the auto-scaling feature of Kubernetes, known as Horizontal Pod Autoscaling, which is the de facto standard as a container orchestration tool.

As a result, compared to the baseline methods, the proposed method reduced the

number of request failures and improved the Mean Squared Error (MSE) between the ideal container count and the actual container count by an average of 150.17 points.

概 要

E コマースサービスなどをクラウド上で提供する際に， サービスレベルを一定に保つ一方でコストをなるべく抑えたい場合， 不規則に増減するトラフィックに合わせて提供するサーバ資源量を変化させる必要がある．

時系列予測や負荷予測といった分野の研究は数多くの先行研究はあるが、多くは学習させるデータが定常過程や単位根過程である必要がある古典的な時系列予測や, 非常に多くのデータやパラメータを用いる深層学習を用いるアプローチであり, 学習と推論に多くのデータや時間を要するため数分程度で急激に増減するバーストトラフィックに対して必要なサーバ資源を提供することは困難である．一方, スライディングウィンドウ学習など細かくデータを区切り, 逐次的に学習と推論を繰り返すようなアプローチの先行研究もある．このスライディングウィンドウ学習を応用した既存の自動スケーリング手法では， 直近の負荷に加えバーストトラフィックを考慮し， 近未来に必要なサーバ資源を提供している．

本研究ではスライディングウィンドウ学習による回帰推定をベースに, ファジィエントロピーを用いたバーストトラフィック検知を取り入れることで, わずかなトレンドの変化にも追従できるようにした動的なウィンドウサイズ変更を伴うトラフィック予測法と, 複数の負荷指標に対する決定回帰木により推定された負荷から必要なコンテナ数を適用する新たなオートスケーリングシステムを提案する．

このシステムを動作させる基盤としてはコンテナ仮想化技術を採用した．コンテナ仮想化技術は一つのプロセスに対して他のプロセスとは独立したサーバ資源を割り当てる技術であり, 必要に応じたサーバ上のコンピュータ資源を柔軟にプロセスに割り当てることができる．また従来のように 1 つの仮想マシンに対して 1 つのアプリケーションサーバを構築するのに比べ, コンテナ仮想化技術を用いることでアプリケーションサーバの起動時間を数秒程度に短縮させることができる．

評価ではシミュレーションではなく実トラフィックを用いた 4 つの比較実験を行う．比較実験ではウェブ上に公開されているトラフィックデータを用いて, 負荷生成器にてトラフィックパターンを再現し, 実験対象であるシステムに対して出力する．ベースライン手法としては先行研究 2 つと, コンテナオーケストレーションツールとしてはデファクトスタンダードである Kubnerets のオートスケーリング機能である Horizontal Pod Autoscaling との比較を行う．

結果， 提案手法がベースライン手法と比較してリクエスト失敗数を減少させ, 理想的なコンテナ数の推移との平均二乗誤差（Mean Squared Error, MSE）が平均 150.17 ポイント改善した．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　導入

近年，Web トラフィックの流量は増大し続けており， 将来どれほど増えるか事前に推定することはますます困難となっている．以下は UCI Machine Learning Repository が公開している E コマースサービスのトラフィックである．

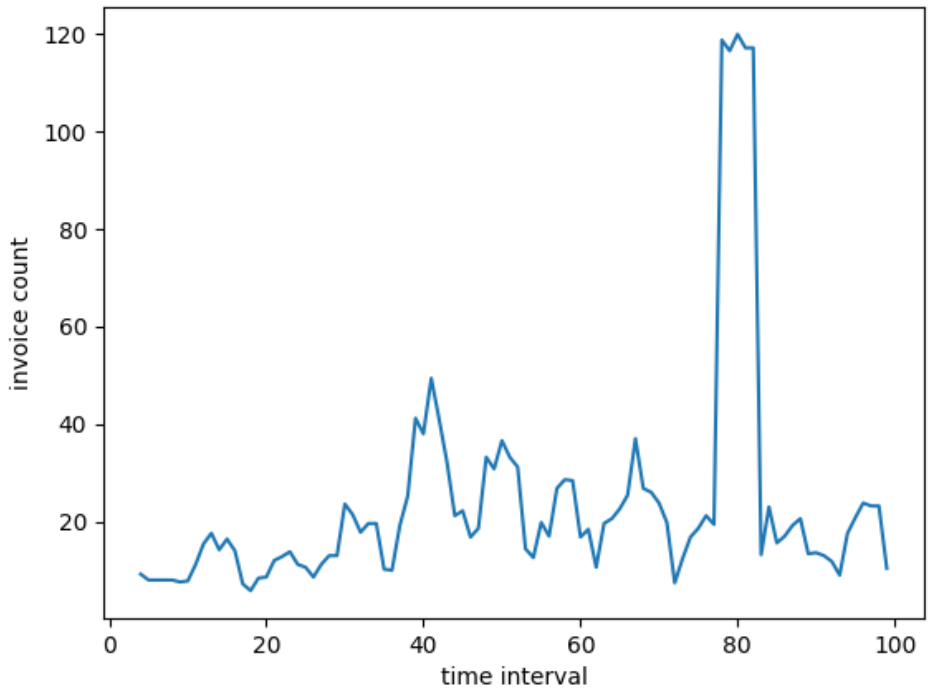

図 1.1: E コマースサービスのトラフィックデータの一例 (https://archive.ics.uci.edu/ml/datasets/Online+Retail+II)

Web サービス， 特に E コマースやメディアのようなサービスでは図 1.1 で表されるような数分の間に突発的なトラフィックの増減（以下このような事象をバーストと呼ぶ）が生じる．加えてこのような Web サービスは最近ではオンプレミスのサーバではなく，Amazon Web Service といったパブリッククラウド事業者上のサーバを利用することが多い．理由としてはサービスの成長や顧客の急激な増減に応じてオンデマンドにサーバ台数を増減できることが挙げられる．また秒単位

の課金体系であることが多く, リソースの要求に対してサーバ台数を細かく正確に変えていくほどサーバにかかるコストを最適化することができる.

従来，こういった時系列データに対して傾向を学習, 推定を行う場合, 時系列分析として著名な ARIMA [3] や SARIMAX [4] などが用いられることが多かった. それらは定常もしくは単位根過程であること, 半年や 1 年間といった長期間のデータを準備できることを前提条件としている. しかし, 本研究で注目するデータはバーストを生じる傾向のある Web トラフィックであり, 従来の長大な時系列データを必要とするような学習方法で的確にバーストの傾向を推定し再現することは難しい.

さらにパブリッククラウドのようなサービス上にサーバを展開したとしても, 起動するサーバが従来使用されてきたハイパーバイザ型仮想マシン上の単なるプロセスである場合，サービスの起動までに数分を要するため，サーバの増減により直後のバーストに対応することは困難である.

以上の課題に対して, 本研究にて行なったアプローチは次の通りである. 従来使われてきたハイパーバイザ型仮想マシン上にアプリケーションを起動するのではなく，コンテナ型仮想化を施したアプリケーションサーバを起動する事で, 1 つの独立したコンピュータ資源を持つアプリケーションを数秒で複数個同時に起動することができる（以下この独立したコンピュータ資源を持つアプリケーションサーバをコンテナと呼ぶ）.

バーストを検知し未来の負荷を推定する場合, その推定器における学習や推定時間を少しでも短くする為に素早く計算を実行し, より直近のデータの傾向に集中して学習することが重要である. そこで本研究では直近 50 秒間の負荷データを用いて学習を行い 10 秒後に生じるであろう負荷を推定する. また負荷の推定には ElasticNet 回帰を用いて線形回帰することで十分な推定精度を保ちつつ高速な負荷推定を実現する. またその推定の過程でバーストを検知した場合, バーストを検知した後のデータのみで学習を実行することで, バーストが生じる前の傾向から独立した推定結果を出力する. このバースト検知器として, 筋電図や心電図の傾向種別などに用いられており短いデータに対しても十分なバースト検知能力を持つ Fuzzy Entropy [5] を採用する. Fuzzy Entropy を Web トラフィックのバースト検知器として使用し, その有効性を示した研究は本研究を除いて他にない.

提案する手法により推定された負荷に対して必要なコンテナ数を事前に学習済みの決定回帰木を通じて推定し, 推定された負荷に応じてコンテナ数を増減させるコンテナ型システムを構築した.

1.2 論文構成

本稿の構成は以下の通りである．第 1 章では本研究の背景や概要部分を述べた. 第 2 章では本研究を理解する上で重要な前提知識について述べる. 第 3 章では関連研究について述べる．第 4 章にて本研究の提案手法について記述する．第 5 章で

は提案手法の有効性を先行研究との比較実験により示す． 第 6 章にて本稿のまとめ， および今後の課題を示す.

第2章　準備

2.1　情報エントロピー

2.1.1　データの複雑性とエントロピー

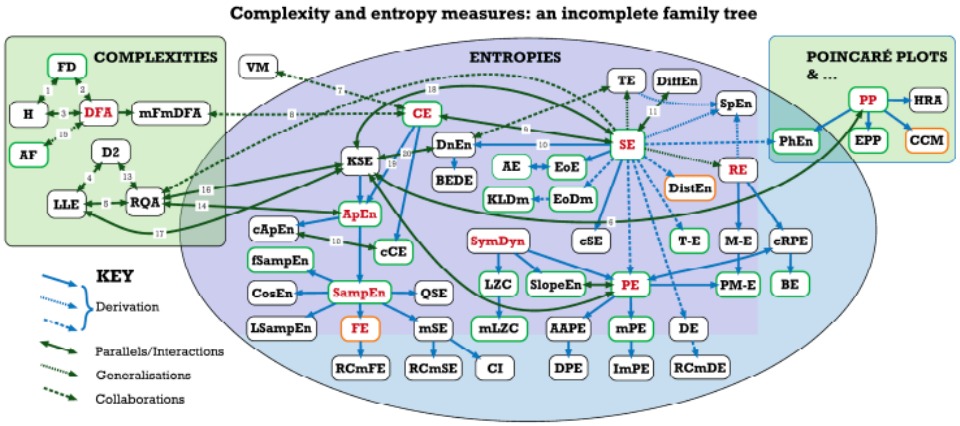

図 2.1: Approximate Entropy とその派生系 [1]

心電図や気候変動, 金融市場などから生じる不規則な時系列データの複雑性を測定する代表的な方法として Shannon Entropy がある. Shannon Entropy は Shannon が提唱した, 情報理論における情報量を定量化しシステムのランダム性の度合いを測定する数学的手法である [6].

X が値 x_1, .. x_n を取り得るとき, それぞれの値に関連する確率を $p(x)$ として表現すると, Shannon Entropy は以下のように定義される.

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{2.1}$$

このようなデータの複雑性を情報理論から定量的に測定する方法は Shannon Entropy の他にも多くの先行研究があり, 特に現代科学において脳波測定における傾向分析やそれに付随する研究には Approximate Entropy と Sample Entropy がよく利用されている [7]. またその用途により図 2.1 にあるような Approximate Entropy（ApEn）から派生したエントロピーが数多く存在する.

2.1.2 Approximate Entropy

Pincus は時系列における規則性の割合を定量化するために Approximate Entropy [8] を開発した. Approximate Entropy は相関性, 持続性, 規則性を測定する数学的手法である. Approximate Entropy が任意の与えられた数列のエントロピーを算出するアルゴリズムは以下となる.

長さ N の数列 u を与えられたとする. ただし N は非負の整数とする.

$$u = \{u_1, u_2, ...u_N\} \tag{2.2}$$

次に非負の整数 m, $m \leq N$ を持つブロック $x(i)$ および $x(j)$ を定義する.

$$x(i) = \{u(i), u(i+1), ..., u(i+m-1)\} \tag{2.3}$$

$$x(j) = \{u(j), u(j+1), ..., u(j+m-1)\} \tag{2.4}$$

この 2 つのブロック同士の Chebyshev 距離 $d[x(i), x(j)]$ について定義する.

$$d_{ij}^m = d[x(i), x(j)] = max_{k=1,2,...,m}(|u(i+k-1) - u(j+k-1)|). \tag{2.5}$$

この長さ d_{ij}^m を用いて、$C_i^m(r)$ を定義する. ただし $C_i^m(r)$ の分子は $j \leq N - m + 1$ の時のみカウントされる.

$$C_i^m(r) = \frac{d_{ij}^m \leq r \text{ を満たす } j \text{ の数}}{N - m + 1} \tag{2.6}$$

算出した $C_i^m(r)$ から更に対数を取った後, その平均値を $\phi^m(r)$ として定義する.

$$\phi^m(r) = \frac{1}{N - m + 1} \sum_{i=1}^{N-m+1} \log C_i^m(r) \tag{2.7}$$

式 2.7 から $ApEn(m, r, N)(u)$ を以下のように定義できる.

$$ApEn(m, r, N)(u) = \phi^m(r) - \phi^{m+1}(r), \tag{2.8}$$

Approximate Entropy が低い値を示す時系列データは, 一貫性があり, 反復性が高く, 予測可能で, そのパターンは時間を経ても繰り返される. 一方 Approximate Entropy が高い値を示す時系列データは, データ間の独立性, 反復パターンの希薄さ, そしてランダム性を示す [7]. この特性は過去のトラフィックにないような突発的に上昇するバーストトラフィックの検知に対して機能する.

またこの後の節で紹介するエントロピーについても, Approximate Entropy で考案された手法をベースとして計測を行うため自己相関性, 持続性, 規則性を測定する特徴は継承されている.

2.1.3 Sample Entropy

Approximate Entropy は発明以来様々な研究に応用 [9–11] されてきたが, 一方で 2 つの課題があった. 1 つ目は相対的整合性が保証されておらず, r の値によっては結果が異なる可能性があることである. 2 つ目は Approximate Entropy の値がデータ系列の長さに依存することである [7]. この 2 つの問題を回避するために, Richman と Moorman [12] はベクトルの距離を比較する際に, 自身同士の比較を行わない統計量である Sample Entropy を定義した.

Sample Entropy は, m 個の点の 2 つの類似した列が次の点 m+1 でも類似したままである条件付き確率の対数の負の値で, 各ベクトルを自分自身を除く他のすべてのベクトルについて数えたものである. これは Sample Entropy が相対的な一貫性を維持し, また系列の長さにほとんど依存しないことを意味する. Sample Entropy のアルゴリズムは以下の通りである.

Approximate Entropy で用いた数列 u（式 2.2）および 2 つのブロック（式 2.3 および式 2.4）は同様に使用するが, ブロック同士を比較して得られる Chebyshev 距離（式 2.5）に条件として $i \neq j$ を加えた Chebyshev 距離 $d[x(i), x(j)]_{x \neq j}$ を定義する. この比較結果を用いて, 以下のように Sample Entropy を定義できる.

$$SampEn(m, r, N) = -\log \frac{A}{B} \tag{2.9}$$

ただし, A は $d[Xm+1(i), Xm+1(j)]_{x \neq j} \leq r$ となるような長さ $m+1$ のテンプレートベクトルの数であり, B は $d[Xm+1(i), Xm+1(j)]_{x \neq j} \leq r$ となるような長さ m のテンプレートベクトルの数である.

また Sample Entropy は A または B がゼロとなる場合は定義されない [12] という性質を持つ.

2.1.4 Range Entropy

Approximate Entropy と Sample Entropy は信号の振幅変化の影響を受けやすいという課題から Amir Omidvarnia 氏は非定常な信号変化に対してより頑健な Range Entropy を提案した [13]. RangeEn は Approximate Entropy と Sample Entropy が用いている Chebyshev 距離を以下の距離関数 $d_{range}(X_i^m, X_j^m)$ に更新したものである.

$$\begin{cases} d_{range}(X_i^m, X_j^m) = \frac{\max_k |x_{i+k} - x_{j+k}| - \min_k |x_{i+k} - x_{j+k}|}{\max_k |x_{i+k} - x_{j+k}| - \min_k |x_{i+k} - x_{j+k}|} \\ k = 0, ...m-1 \end{cases} \tag{2.10}$$

本論文では Chebyshev 距離を距離関数 $d_{range}(X_i^m, X_j^m)$ に更新した Approximate Entropy および Sample Entropy を以下 $RangeEnA(m, r, N)$, $RangeEnB(m, r, N)$ とする.

2.1.5 Distribution Entropy

Li らは N=50 のような非常に少ないデータ数に対しても安定しており, データ分散に依存しない Distribution Entropy [14] を考案した. Distribution Entropy も Sample Entropy と同様, 数列 u（式 2.2）および 2 つのブロック（式 2.3 および式 2.4）, またそれらのブロック同士の Chebyshev 距離（式 2.5）を使用する.

Distribution Entropy ではブロックを M 個のビンと捉え, Chebyshev 距離（式 2.5）の経験的確率密度関数を推定する. よって Distribution Entropy は以下のように定義できる.

$$DistEn(m) = -\frac{1}{\log_2(M)} \sum_{t=1}^{M} p_t \log_2(p_t) \tag{2.11}$$

2.1.6 Fuzzy Entropy

Approximate Entropy および Sample Entropy は表面筋電図（EMG）信号の特徴抽出と特徴分類に従来から広く使用されてきたが，Approximate Entropy は類似度を過大に評価する傾向があり [7]，Sample Entropy はパラメータの値が極端に小さい場合に出力値が定義できない欠点があった．このような問題点を解決するために，2007 年，Chen らによって Fuzzy Entropy [5] が提案された.

Approximate Entropy と Sample Entropy では, 本節で説明してきた通り 2 つのベクトル間の誤差が r という閾値内に収まるかどうかで分類する伝統的な二値分類を行う. しかしながら現実世界ではクラス間の境界はあいまいであり, 入力パターンが完全に特定のクラスに属するかどうかを判断することは困難である.

1965 年に Zadeh により導入されたファジー集合の概念 [15] は, この種の入出力関係を不確かな状況下で特徴づける手法を提示している. Fuzzy Entropy はこの Zadeh の理論で提唱された「メンバーシップ度」を導入することにより, 前述したような近似度を 0 か 1 で算出する代わりに $[0, 1]$ の範囲の連続する実数値へと関連付けて表現し, 対象の時系列データのパターンがどの程度ランダム性を持つかを評価する手法である [5].

以下に文献 [5] から引用した Fuzzy Entropy の定義を記す. Fuzzy Entropy についても比較を行うベクトルの定義については Sample Entropy と同様, 数列 u（式 2.2）および 2 つのブロック（式 2.3 および式 2.4), またそれらのブロック同士の Chebyshev 距離（式 2.5）を使用する.

ここで Fuzzy Entropy は，Chebyshev 距離 d_{ij}^m，実数 n,r に対するファジィ関数 $\mu(d_{ij}^m, n, r)$ により $x(i)$ と $x(j)$ の類似度 D_{ij}^m を算出する.

$$D_{ij}^m = \mu(d_{ij}^m, n, r) \tag{2.12}$$

ファジィ関数 $\mu(d_{ij}^m, n, r)$ のメンバーシップ関数として指数関数を採用する．

$$\mu(d_{ij}^m, n, r) = \exp\left(-\frac{(d_{ij}^m)^n}{r}\right) \tag{2.13}$$

類似度 D_{ij}^m から ϕ^m を以下のように計算する．

$$\phi^m = \frac{1}{N-m} \times \sum_{i=1}^{N-m}\left(\frac{1}{N-m-1}\sum_{j=1, j\neq i}^{N-m} D_{ij}^m\right) \tag{2.14}$$

同様に ϕ^{m+1} についても以下のように計算する．

$$\phi^{m+1} = \frac{1}{N-m} \times \sum_{i=1}^{N-m}\left(\frac{1}{N-m-1}\sum_{j=1, j\neq i}^{N-m} D_{ij}^{m+1}\right) \tag{2.15}$$

最後に，ϕ^m，ϕ^{m+1} それぞれの自然対数をとり，それらの差を $FuzzyEn(m, n, r, N)$ と定義する．

$$FuzzyEn(m, n, r, N) = \ln \phi^m - \ln \phi^{m+1} \tag{2.16}$$

2.1.7 本論文での位置付け

本章で触れたシャノンエントロピー, およびその派生エントロピーはNが50-2000程度の時系列データに対しても高速に動作し, 平均回帰性, 持続性, 規則性を測定できる. 本稿ではその中でもより優れたものを採用する. それら評価については5節にて記述する.

2.2 コンテナ型仮想化とコンテナスケジューリング

2.2.1 コンテナ型仮想化

コンテナ型仮想化を施したアプリケーション（以下コンテナと呼ぶ）は, Docker [16] などのコンテナ型仮想化管理ツールがインストールされたサーバ（以下ノードと呼ぶ）上で, 他のプロセスとは独立したサーバ資源（サーバに積載されているアプリケーションの実行に必要な資源. CPU, メモリ, ネットワーク等を指す）を利用するプロセスとして実行される. コンテナは, クライアントツール（以下クライアントと呼ぶ）からノードに対して発行される実行開始命令を介してノード上で実行される.

クライアントの実行開始命令には実行すべきコンテナの種類が記載されている. ノードは実行すべき種類のコンテナイメージ（コンテナを保管するために加工されたバイナリファイル）のノード上での存否を確認し, 存在しなかった場合はノー

ド自身がコンテナイメージを保管しているHTTPサーバ（以下コンテナレジストリと呼ぶ）に対してリクエストを行い, コンテナイメージを取得でき次第コンテナを実行する. 存在する場合は速やかにキャッシュされているコンテナを実行可能である.

2.2.2 コンテナオーケストレータとコンテナスケーリング

1台以上のノードで構成されたサーバ群（以下クラスタと呼ぶ）に対してKubernetes [17] 等, 専用のクラスタ管理ツール（以下コンテナオーケストレータと呼ぶ）を用いることで, クラスタの中から特定のノードを選んでコンテナの配置を行うことができる. その際予め定められた規則に従い特定のノードに対してコンテナの配置を行う. このようにクラスタ内の特定のノードに対して, ある規則に従ってコンテナを配置することをコンテナスケーリングと呼ぶ.

2.2.3 本論文での位置付け

本研究では特に数十秒といった非常に短い時間においてスケーリングを行いたいニーズに応えるシステムのため, 仮想マシンを用いるより数秒で起動および停止させることが可能なコンテナ型仮想化を採用するに至った. またコンテナスケーリングを行うコンテナオーケストレータとしてはその分野でデファクトスタンダードであるKubernetes [17] を採用した.

第3章　関連研究

3.1　バースト検知

Eldin ら [18] はクラウドシステムにおいて, スパイクは予測不可能な負荷量を発生させるイベントだと定義し, 時系列データに対してこのバースト性を定量的に測定する手法として, 生体信号解析で用いられる Sample Entropy [12] に対して独自の修正を施した AvgSampEn を提案した. 提案された手法は与えられた時系列データを等間隔に分割した後に, それぞれ Sample Entropy によりエントロピー値を計測し、最後に加重平均を取る方法である. この方法は Sample Entropy の計算量の大きさについて対処した方法ではあるが, 後述するような $N = 50$ のような大きさのデータセットに対して出力値が定義されない問題について考慮されてない.

Mehta ら [19] はクラウドのワークロードの急激な増加を検出する方法として, 適応型信号処理の技術を応用して、スパイクを早期に検出する手法を提案している. この手法では負荷予測モデルに対して, 予測値と実測値の差をバーストの定量的な大きさと捉えバースト検知を行う. しかしこの研究では時系列データに対して前処理が必須であり, また用いる予測モデルのアルゴリズムに強く依存する.

3.2　負荷に応じたオートスケーリング手法

Kubernetes には Horizonal Pod Autoscaler [20] という, 対象のコンテナが測定時に生じている平均リソース消費量と理想的な平均リソース消費量を比較して, 対象のコンテナが理想的な平均リソース消費量となるようコンテナ数を自動で増減させる機能がある. この機能ではコンテナ数を増加させる処理は過去 3 分以内に再スケーリングがなかった場合にのみ実行し, 一方減少させる処理は実行条件が揃っていても最後の再スケーリングから 5 分間待つ実装になっている [21] .

3.3　時系列分析による負荷予測

時系列データに対する負荷推定として, Balaji ら [22] はメモリや CPU 使用量, 帯域幅などのメトリクスデータの 8 分間隔のデータセットに対して Sample Entropy もしくは Hurst Exponent [23] を用いて算出した値から, ARIMA モデルを用いて

将来の負荷量を推定するモデルを提案した．この研究は 10 分以上継続する負荷の急劇な変動に特化しており，かつ推定結果の反映が 8 分後になる．

また Langstom ら [24] は LSTM [25] および SARIMA モデルを用いた時系列分析にてコンピューティングクラスタの CPU 使用率の推定方法を提案しているが，この手法は 20 分毎に集計された時系列データに対する推定であり, 数十秒程度で発生する連続するスパイクには対処できない.

3.4 スライディングウィンドウによる負荷予測

Dalmazo ら [26, 27] はクラウド環境において, 固定または動的なサイズの数分程度の短い時系列データを入力として， ポアソン分布により重み付けする統計モデルに基づいて将来のトラフィックを予測するスライディングウィンドウによる時系列予測手法を提案した． ただしこの手法も 5 分間隔のデータセットでの学習であり，また 1 つのパラメータ（パケット総数）のみの学習となっている．

Yoona ら [28] はポアソン分布による重み付けと線形回帰の組み合わせモデルを用いたスライディングウィンドウによる時系列予測手法を採用した．この手法は 1 年前のデータから傾向を抽出し，それを特徴量の重みづけに用いているが，スライディングウィンドウのサイズは 30 分間であり，短時間のバーストトラフィックへの対応は難しい．

Baig [29] らはスライディングウィンドウ学習を実行するにあたり, 最適なウィンドウサイズを過去のメトリクスデータから深層学習で推定する手法を提案した．この手法では推定精度が過去のメトリクスデータの量に依存し, また推定器がなぜそのウィンドウサイズに設定したのかを説明することが難しい.

Hirayama ら [30] は過去のトラフィックデータの中から負荷を推定するために最も参照するべき時間帯を探索し, その時間帯の説明変数を用いた回帰推定を行う. この手法は時間帯により説明がつくような負荷パターンに特化した推定方法といえる.

3.5 スライディングウィンドウとバースト検知を併用した負荷予測

スライディングウィンドウ形式による時系列予測手法に加え， バースト検知器を併用することにより， バーストトラフィックに対しての対応力を持つ手法が提案されている．

Tahir ら [31] はバースト検知器に Sample Entropy を採用し，一定のエントロピー値を示した場合，過去のクラスタサイズ（コンテナ数）の中で最も大きいクラスタサイズを割り当てる手法を提案した．Abdullah ら [32] はバースト検知器に標準偏差を採用し，過去の負荷に対してウィンドウサイズを変動させながら標準偏差

を計算し，計算された標準偏差が 2 より大きな場合に過去のクラスタサイズの中で最も大きいクラスタサイズを割り当てる手法を提案した.

3.6 本論文の位置付け

本稿では関連研究におけるスライディングウィンドウ学習を踏襲しつつ，バースト検知器およびバースト検知に独自の手法を採用する．我々の手法は，ウィンドウサイズを最低限必要なサイズにすることにより，短い期間のバーストに対応する．本論文の主な貢献は以下の通りである.

- 数分間続く負荷の増減にも対応できる自動リソースプロビジョニングを実現する.
- EEG（Electroencephalogram）の状態判別に活用される Fuzzy Entropy [5] を Web トラフィックのバースト検知に応用する． そのような例は本研究を除いて先行研究にはない.
- 実環境のクラウドサービス上で評価システムを構築し，実際にバースト負荷を発生させて評価を行った.

また先行研究と提案手法の定性比較を表 3.1 にまとめた。

表 3.1: 先行研究と提案手法の比較表

論文名	負荷予測	バースト検知	短時間でのバースト検知	リソース割り当て	最適な学習期間へ動的変更	評価方法
Eldin et al. [18]		✓				シミュレーション
Mehta et al. [19]		✓				シミュレーション
Hirayama et al. [30]	✓				✓	シミュレーション
Bruno Lopes Dalmazo et al. [27]	✓				✓	シミュレーション
Shuja-ur-Rehman Baig et al. [29]	✓				✓	実装実験
Mahesh Balaji et al. [22]	✓	✓		✓		実装実験
Muhammad Abdullah et al. [32]	✓	✓		✓		実装実験
Fatima Tahir et al. [31]	✓	✓		✓		実装実験
Proposal	✓	✓	✓	✓	✓	実装実験

第4章　提案手法

4.1　アルゴリズム

本研究で提案する動的リソースプロビジョニングのアルゴリズムをアルゴリズム 4.1 に示す.

アルゴリズム 4.1 提案手法のアルゴリズム

Input: ウィンドウサイズ（k，初期値 k^0），バッチ間隔（ξ），負荷指標のデータセット（W），負荷推定モデル（δ_W），バースト検知モデル（ϕ_B），トレンド判定モデル（ϕ_T），コンテナ数推定モデル（δ_C）
Output: コンテナ数（$\mathbb{R}$）を更新
$k \leftarrow k^0$;
while true **do**
  $Wait\ for\ \xi\ seconds$
  $t \leftarrow t + 1$
  $W \leftarrow [W_{t-k+1}, W_{t-k+2}, ..., W_t]$ ...(a)
  if $k \neq k^0$ **then**
    $k \leftarrow k + 1$; ...(b)
  end if
  $\widehat{W}_{t+1} \leftarrow \delta_W(W, t+1)$; ...(c)
  $B_t \leftarrow \phi_B(W)$; ...(d)
  $T_t \leftarrow \phi_T(W)$; ...(e)
  if $(T_{t-1} = Down$ and $T_t \neq Down)$ or $(T_t = Up)$ **then**
    if $k = k^0$ and $B_t = Surprise$ **then**
      $k \leftarrow 1$; ...(f)
    end if
  end if
  $\widehat{\mathbb{R}} \leftarrow \delta_C(\widehat{W}_{t+1})$; ...(g)
  $\mathbb{R} \leftarrow \widehat{\mathbb{R}}$ ...(h)
end while

4.1.1 スライディングウィンドウ学習

本研究では細かい粒度で時系列データの傾向および変化にいち早く反応するために，多くの先行研究でも採用されているスライディングウィンドウ学習を採用する．スライディングウィンドウ学習は，時系列データや連続的なデータを扱う際に用いられるデータ分析手法の一つである．この手法では，データセット全体を小さな連続するデータセットに分割し，それらを逐次的に学習する．

4.1.2 学習期間，およびデータ

スライディングウィンドウ学習で使用されるデータについて (アルゴリズム 4.1(a))，本研究ではバッチ間隔 ξ 秒ごとに前回のウィンドウから 1 つ分ずらして学習を行う．$t-k+1$ 個前のデータセットから現時刻のデータセットまでを学習のための時系列データとして利用する．設定値としてバッチ間隔 ξ を 10 秒とし，ウィンドウサイズの初期値 k^0 を 5 に設定した．

負荷指標のデータセット (W) として以下の指標を単位秒毎に取得して使用する．

- リクエスト数
- クラスタ内のコンテナのうち，最も高い CPU 使用率
- クラスタ内のコンテナのうち，最も高いメモリ使用率

4.1.3 負荷の学習及び推定

アルゴリズム 4.1(c) について．負荷推定モデル δ_W は負荷指標のデータセット W を入力として学習を行わせて，近未来の負荷を推定する．本研究では回帰推定モデルである ElasticNet モデルを採用した．

ElasticNet 回帰は L1 正則化（Lasso 回帰）と L2 正則化（Ridge 回帰）を組み合わせて行う線形回帰である． 本研究の線形回帰は負荷指標のデータセット W を入力として現在から ξ 秒後の推定値 $\widehat{W}_{t+1}$ を出力する．

4.1.4 バースト判定

アルゴリズム 4.1(d) について．バースト計測モデル ϕ_B は学習データ W を入力として，予め設定した閾値を下回ったエントロピー値を算出した場合，学習期間中にバースト現象が発生しているかどうかの判定を行う．本研究では Fuzzy Entropy [5] を用いてエントロピー値を算出した．バースト計測モデル ϕ_B の出力値 B_t は 2 種類あり， 学習期間中にバースト現象が発生していると判定された時は $Surprise$, そうでない時は $Expected$ と出力される．

4.1.5 トレンド判定

アルゴリズム 4.1(e) について．トレンド計測モデル ϕ_T は学習データ W を入力としてペアワイズ相関を計算し，予め設定した２つの閾値を基準とする．１つ目の閾値より高くなった場合を上昇トレンド (Up) とし，２つ目の閾値より低くなった場合を下降トレンド ($Down$) とし，どちらでもない場合はトレンド無しの状態 ($Middle$) とする．

4.1.6 学習の履歴データをリセットする処理

アルゴリズム 4.1(f) について．以下の条件の少なくとも一つが成立したときにバーストが発生していると判定する．

- $t-1$ 時点が下降トレンドで，t 時点で下降トレンド以外に変化したとき
- t 時点で上昇トレンドのとき
- 学習の履歴データが初期値 k^0 であるとき
- t 時点で B_t が $Surprise$ されたとき

上記の条件が揃った時，いわゆるバースト事象が生じている．この場合，直近の t のウィンドウより手前ではバースト事象が生じておらず，そのためそれらを含めて学習すると，予測値が理想値に比べて過少に評価される．

そこでバースト事象が判定されたときに一度学習に使用するウィンドウサイズ k を 1 とし，その後ウィンドウをスライドする度にウィンドウサイズに 1 を加算し，ウィンドウサイズ k が初期値 k^0 に戻るまで加算することで，過少に評価され，サービスが停止することを防ぐ．

4.1.7 推定された負荷に対して必要なコンテナ数を決定する処理

アルゴリズム 4.1(g) について．「ξ 秒後に生じるであると推定された負荷量 $\widehat{W}_{t+1}$」を入力として「ξ 秒後に必要となるであろうと推定されたコンテナ数 $\widehat{\mathbb{R}}$」を出力するモデルが決定回帰木 δ_C である．この決定回帰木の学習データは事前に実験環境にて取得し，学習はシステム運用前に行われる．

事前実験において，5 節の比較実験と同様の HTTP リクエストを出力し，リクエスト発生率を変動させながら，10 分間そのリクエスト発生率を続けても 1 秒未満でレスポンスを返すことが可能な必要最低限のコンテナ数を取得する．加えて，特定のコンテナのメモリや CPU の使用率が 90%以上になった場合にコンテナ数を増やす方針を採る．

4.1.8 コンテナ数をスケーリングする処理

アルゴリズムの最後（アルゴリズム 4.1(h)）に，推定されたコンテナ数 $\widehat{\mathbb{R}}$ を適用する．この適用は，アルゴリズム 4.1 の while ループが始まってからちょうど ξ 秒後にサービスを提供できる状態となったコンテナの数が $\widehat{\mathbb{R}}$ 個となるようにする．

コンテナ数を増加させる場合は, コンテナを起動してからサービスが提供するまでの時間を加味して, 処理が開始されてから追加したコンテナがトラフィックを受信できる状態になるまで約 ξ 秒間となるようにプログラム側で調整を行う．

コンテナ数を減少させる場合は, 停止時間を加味しなくても良いため処理が開始されてから約 ξ 秒後に即時停止信号を送る．

4.2 バースト検知器選定

本節にて， 本研究で採用するバースト事象を検知する検知器を選定する過程を説明する．

4.2.1 Entropy の比較実験

本研究においてバースト検知器を用いて検知する事象は「たった今, および直前の時点でバースト事象が生じているかどうか」である．つまりスライディング学習にて現在に近いデータのみを使用して学習することにより，検知精度が上がり，学習データの数も減ることで計算速度も速くなる．

時系列データのランダム性を測定する Approximate Entropy およびその派生系のエントロピー群は，N=50–2000 程度のデータセットでも十分に機能 [1] し，実行速度も 100 ミリ秒未満で終わるものも多く，本研究の目的に沿った動作が期待できる． しかしその中には計算の過程で計算不能な値を取るようなケースを持つものがある．例えば Sample Entropy は入力された値によっては出力値が定義されない場合がある [12].

次節では，候補となるエントロピーに対して以下の観点にて評価を行い，本研究のバースト検知器への採用を判断する．

- N=50 の非常に少ないデータセットに対して適用した場合, 定義された値を常に返却できるかどうか.
- N=50 の非常に少ないデータセットに対して適用した場合, 閾値を設定してバーストを検知できるような出力値であるかどうか.

4.2.2 頑強性比較

本研究では多くの先行研究で採用実績のある Approximate Entropy と Sample Entropy に加えて，参考文献 [1] から特に短いデータセットに対して適した Range Entropy， Distribution Entropy，Fuzzy Entropy に対し，どのエントロピーがバースト検知器に最も適しているか比較実験をおこなった．検証するにあたり，図 4.1 で表されるような N=10000 のホワイトノイズのデータセットを使用し，それぞれのエントロピー関数を用いて N=50 のウィンドウで評価し，ウィンドウを 1 ずつスライドしながら全てのデータセットを評価するようなスライディングウィンドウ学習を行った．

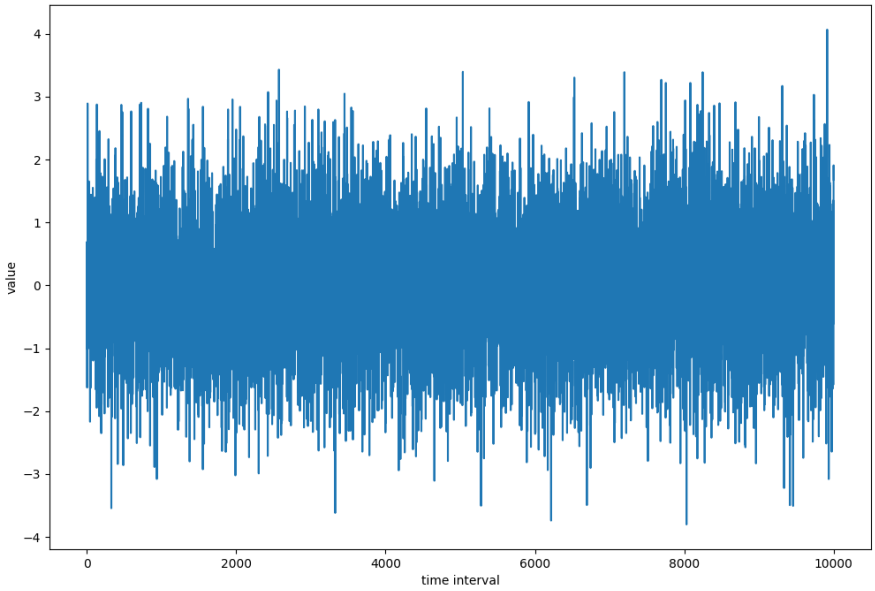

図 4.1: ホワイトノイズ

表 4.1: 各エントロピーとホワイトノイズの対する学習結果の外れ値の占める割合

エントロピー名	外れ値の占める割合 (%)
Approximate Entropy	0.00
Sample Entropy	25.90
Range Entropy A	99.50
Range Entropy B	91.59
Distribution Entropy	0.00
Fuzzy Entropy	0.00

表 4.1 から，ホワイトノイズのようなランダムウォークのデータにおいて N=50 のウィンドウで正常にエントロピー値を算出できるのは Approximate Entropy，Distribution Entropy，および Fuzzy Entropy であることがわかった．

4.2.3 バースト検知性能の比較

前節で Approximate Entropy，Distribution Entropy，および Fuzzy Entropy が特に N=50 のような短いデータセットでも外れ値を出力しないエントロピーであることがわかったが，これらに対してバースト検知が可能かどうかを実験により調べる．

実験データとして NeuroKit2 [33] ライブラリから図 4.2，図 4.3 のようなサンプルデータを出力した．

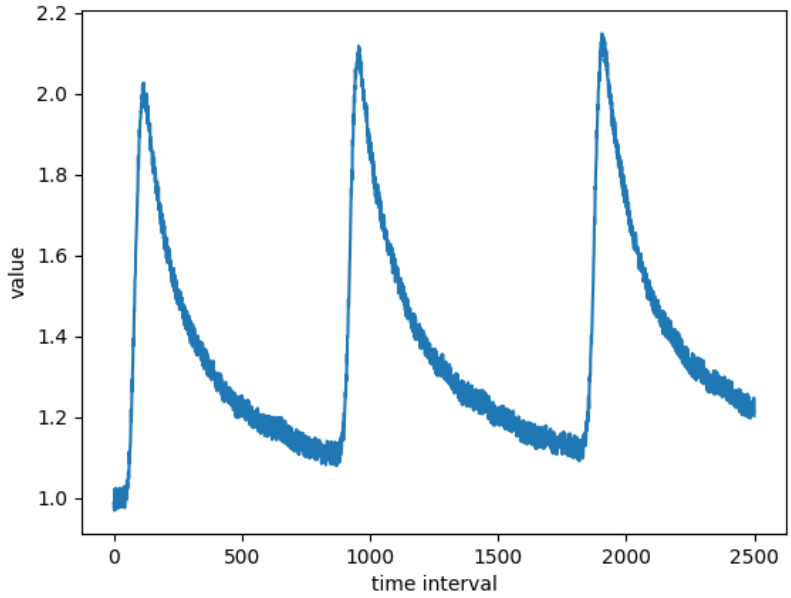

図 4.2: NeuroKit2 で生成したテストデータ#1

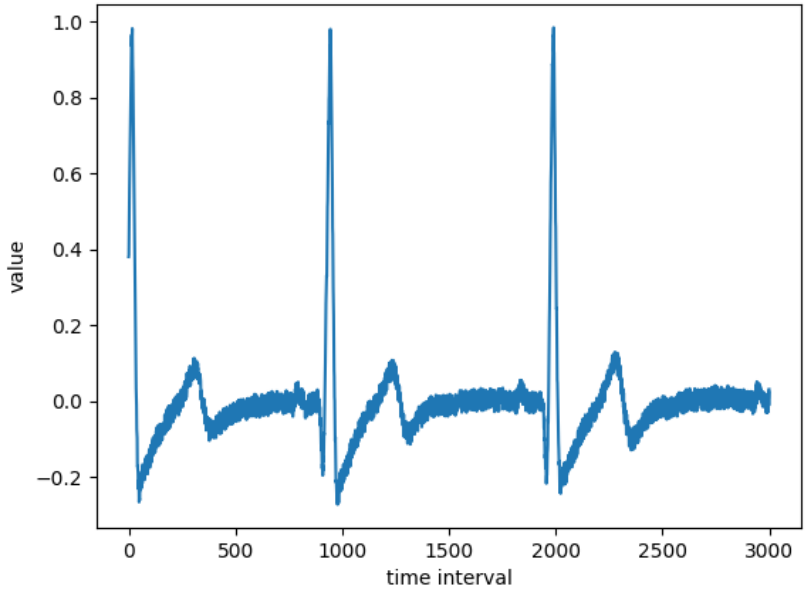

図 4.3: NeuroKit2 で生成したテストデータ#2

これらのデータに対して，N=50 ずつ Approximate Entropy，Distribution En-

tropy，Fuzzy Entropy それぞれスライディング学習を行なった．その結果が図 4.4，図 4.5 である．

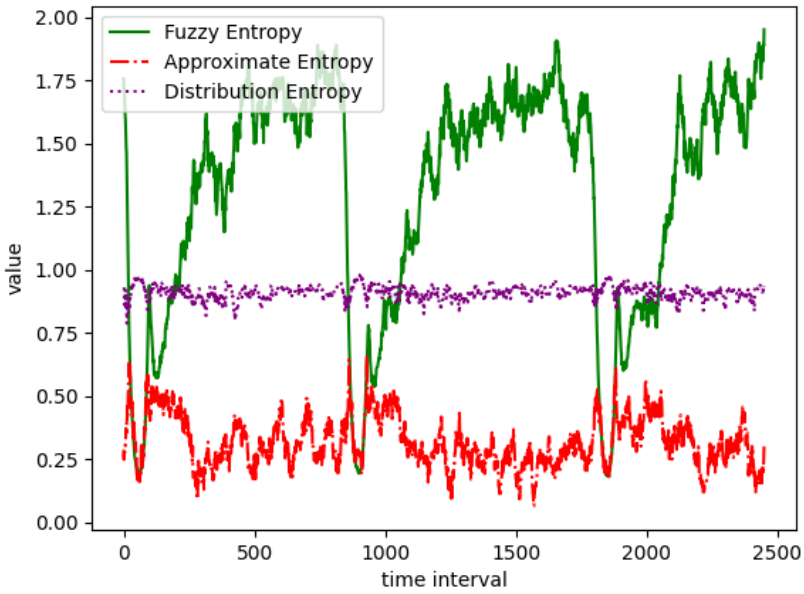

図 4.4: テストデータ#1 に対する各種エントロピー値

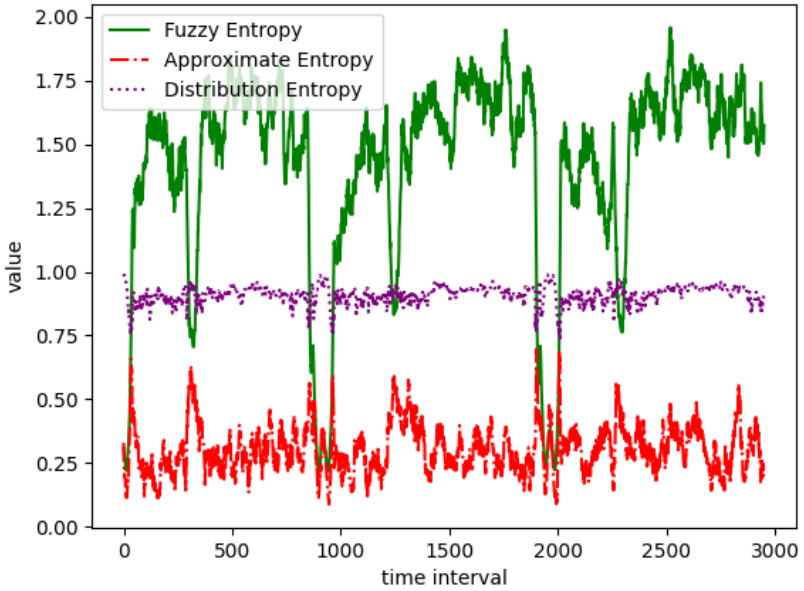

図 4.5: テストデータ#2 に対する各種エントロピー値

図 4.4，図 4.5 から，出力されたエントロピー値に対して，閾値を設けてバース

ト現象が生じているかどうかを判断することを念頭におくと，この実験ケースでは明らかに Fuzzy Entropy の出力したエントロピー値のみが実際のデータ値の急激な増減に対して反応できていることは明白である.

4.2.4 標準偏差と Fuzzy Entropy の比較評価

データの平均値に対する数値の散らばり具合を測定する度合いとして標準偏差があり, 標準偏差を用いてトラフィックのバーストを検知する例もある [32].

本稿では, バースト検出器を決定する上で, 標準偏差と Fuzzy Entropy による平均回帰性に着目し, これらを比較評価した.

4.2.4.1 Hurst 指数

比較評価を行うにあたり, 平均回帰性を定量的に評価する手法として Hurst 指数を用いる. Hurst 指数は 1951 年に Hurst が提案した時系列データのトレンドの持続性を識別するための統計的方法論であり, その時系列データの平均回帰性を定量的な測定値として出力する [23]. Hurst 指数は時系列データの平均回帰性を測定する統計的方法論として幅広く使用されており, $0 < H < 1$ で表される実数で時系列データに対して次の指標を与える.

- $H < 0.5$: 平均回帰する特徴を持つ. 値が 0 に近いほど, 平均回帰の過程が強くなる. つまり平均値より高い, もしくは低い値が出現した後に平均値に戻るような傾向がある.
- $H \cong 0.5$: 0.5 に近いほどブラウン運動のようなランダムウォークに近い動きを行う.
- $H > 0.5$: トレンドのある（持続的な）特徴を持つ. 値が 1 に近いほど, トレンドが強いことを意味する. 平均値より高い値の後により高い値が続き, 平均値より低い値の後により低い値が続く傾向がある.

4.2.4.2 実験データ

用いる時系列データとしては $N = 100$ の以下のような時系列データを用いる.

- 3 つの典型的なブラウンノイズ（図 4.6, 図 4.7, 図 4.8）
- 1 つの平均回帰性の高いホワイトノイズ（図 4.9）
- 1 つの単調増加なノイズ（図 4.10）

ブラウンノイズは $1/f^2$ ノイズとして定義される時系列データで, 連続した値を取りながらランダムに上下を推移する性質を持つ [34]. 本稿では標準偏差の性質を確認するため, 値のスケールが大きく異なる 3 つのブラウンノイズを使用する.

ホワイトノイズは, 全ての周波数帯域で等しい強度（パワースペクトル密度）を持つ時系列データである. [35]. 本稿ではブラウンノイズとの比較のため平均回帰性の高いデータとして採用した.

平均回帰性が低くなるようなデータとして, 底が 1.33 の対数関数にランダムな値を取るノイズを加えた単調増加な時系列データを採用した.

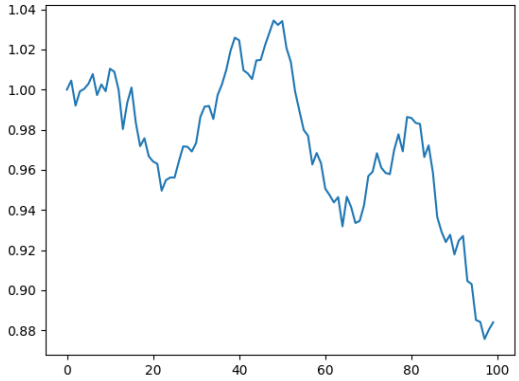

図 4.6: ブラウンノイズ#1

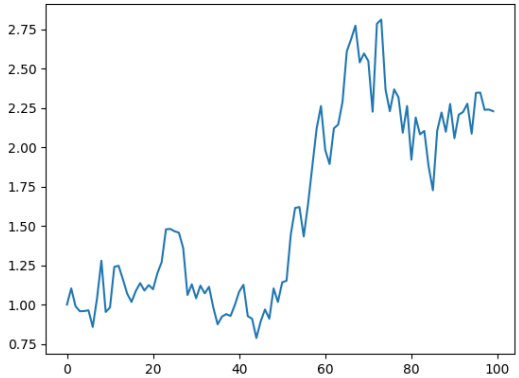

図 4.7: ブラウンノイズ#2

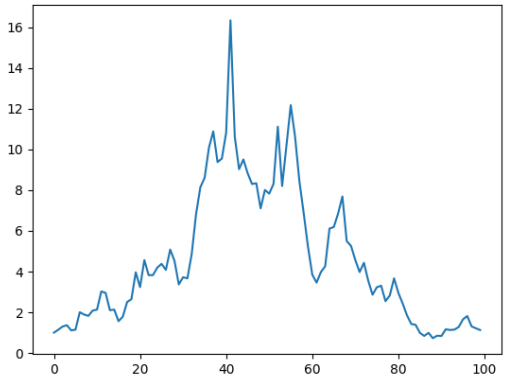

図 4.8: ブラウンノイズ#3

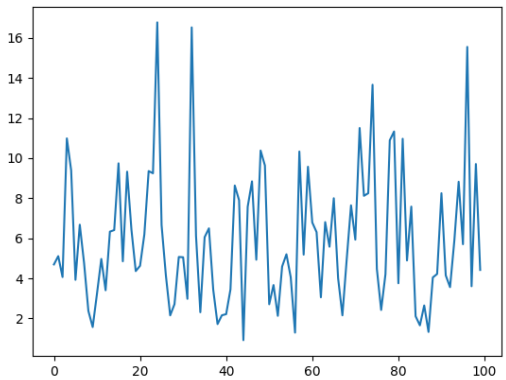

図 4.9: ホワイトノイズ

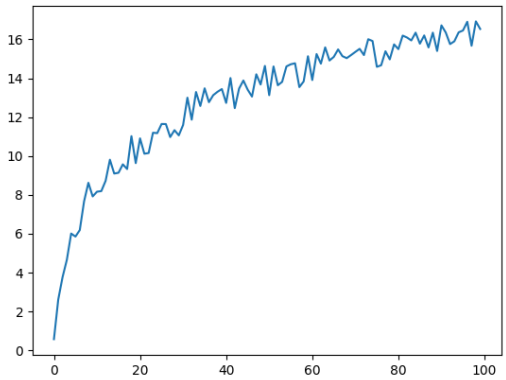

図 4.10: 単調増加ノイズ

4.2.4.3 実験結果

表 4.2: 標準偏差と Fuzzy Entropy の自己相似性の判別性能の比較

信号名	Hurst 指数	最小値	最大値	標準偏差	Fuzzy Entropy
ブラウンノイズ#1	0.50679	0.87563	1.03441	0.03644	0.56797
ブラウンノイズ#2	0.50118	0.78732	2.81288	0.60167	0.56173
ブラウンノイズ#3	0.50697	0.72682	16.34946	3.32832	0.54625
ホワイトノイズ	0.23025	0.91168	16.51550	3.36972	1.52597
単調増加ノイズ	0.87070	0.57787	16.92517	3.39684	0.40842

実験結果は表 4.2 に示す．表内のブラウンノイズ#1-3 から, Fuzzy Entropy は Hurst 指数を持ついずれのデータに対しても殆ど変わらない値を算出したのに対して, 標準偏差では値の分散が大きいほど大きな値を取る．一方ブラウンノイズ#3 とホワイトノイズを比べると両方とも最小値, 最大値, 標準偏差は同様な値を取るが, Hurst 指数が 0.5 より小さく平均回帰性が高い場合, Fuzzy Entropy はブラウンノイズと比べて非常に高い値を出力する．最後にブラウンノイズ#3 と単調増加ノイズを比べると両方とも最小値, 最大値, 標準偏差は同様な値を取るが, Hurst 指数が 0.5 より大きく平均回帰性が低い場合, Fuzzy Entropy はブラウンノイズと比べて低い値を出力する．

以上のことから, 標準偏差と比べて Fuzzy Entropy は測定対象の値の大小だけでは大きく出力値を変えず, また平均回帰性が高い時系列データに対してはランダム性を低く見積もる傾向があり, より負荷のバースト検知に向いていることが分かった．

以上, 4.2 節では比較実験を行い, 情報エントロピー及び標準偏差の中から頑強性や平均回帰性と出力値との相関などを評価した結果, 本研究では最も短期間のトラフィックのバースト検知器として優れている Fuzzy Entropy を採用する．

第5章　評価

5.1　実トラフィックを用いた再現実験

本研究では実トラフィックを用いた再現実験を行うことによって, 提案手法がより有用性が高いことを示す.

5.1.1　実験データ

5.1.1.1　Worldcap

このデータセットは，1998 年 4 月 30 日から 1998 年 7 月 26 日の間に 1998 World Cup Web サイトに対して行われたリクエストから構成される.

本研究ではそのうち 1998 年 6 月 26 日 13:00:00 から 1998 年 6 月 26 日 22:59:54 までの間のトラフィックを再現した（図 5.1）.

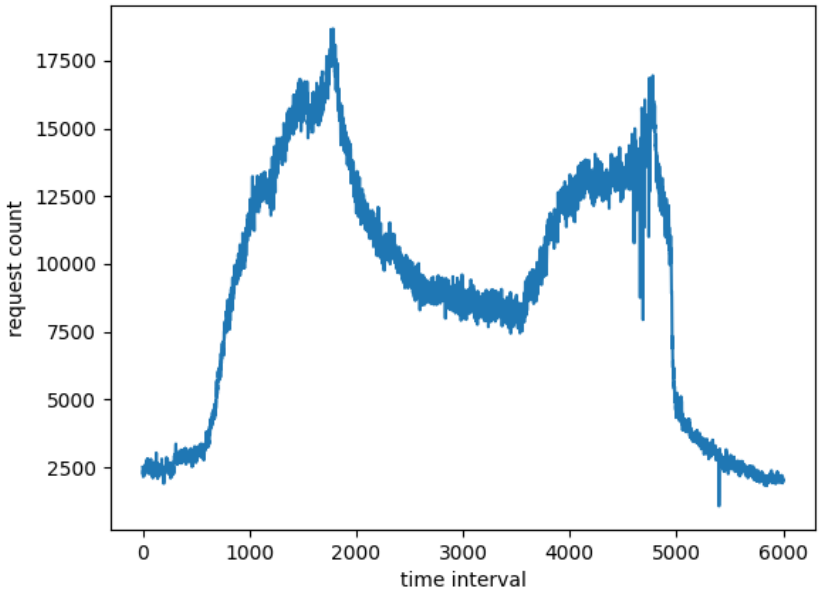

図 5.1: Worldcap'98(https://ita.ee.lbl.gov/html/contrib/WorldCup.html)

5.1.1.2 SWIM

SWIM プロジェクトは MapReduce [36] システムのワークロードのデータを提供している．本研究では 2010 年 10 月から 2010 年 11 月までの Facebook 社の 3000 ノードのクラスタ上の Hadoop トレースにおける CPU 使用率の推移から特徴的な推移を見せている部分を時系列データとして再現した（図 5.2）．

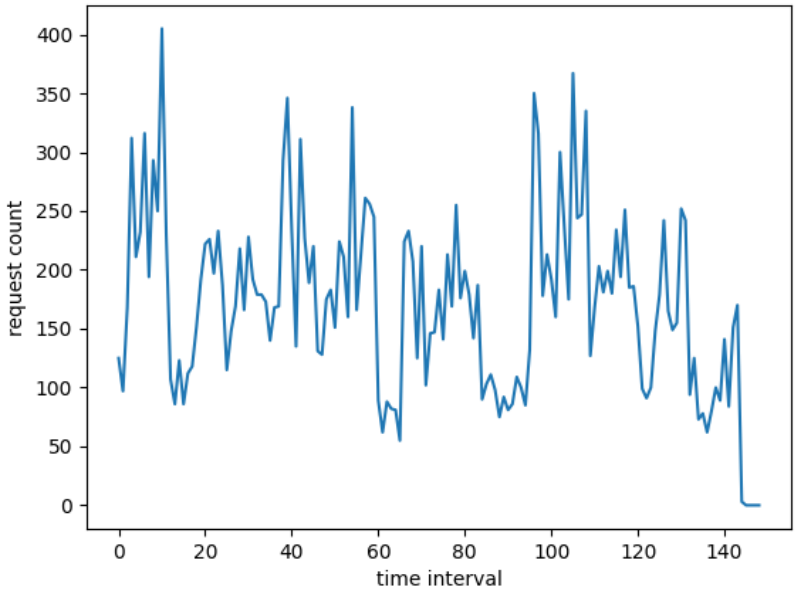

図 5.2: SWIMProject(https://github.com/SWIMProjectUCB/SWIM/tree/master/workloadSuite)

5.1.1.3 Wikipedia

このデータセットは，Wikipedia.org が保有するすべてのページに対する，モバイルアプリのページビューの 1 時間ごとの時系列データである．

本研究ではそのうち 2015 年 09 月 16 日 05:00:00 から 2015 年 09 月 19 日 07:00:00 までの間のトラフィックを再現した（図 5.3）．

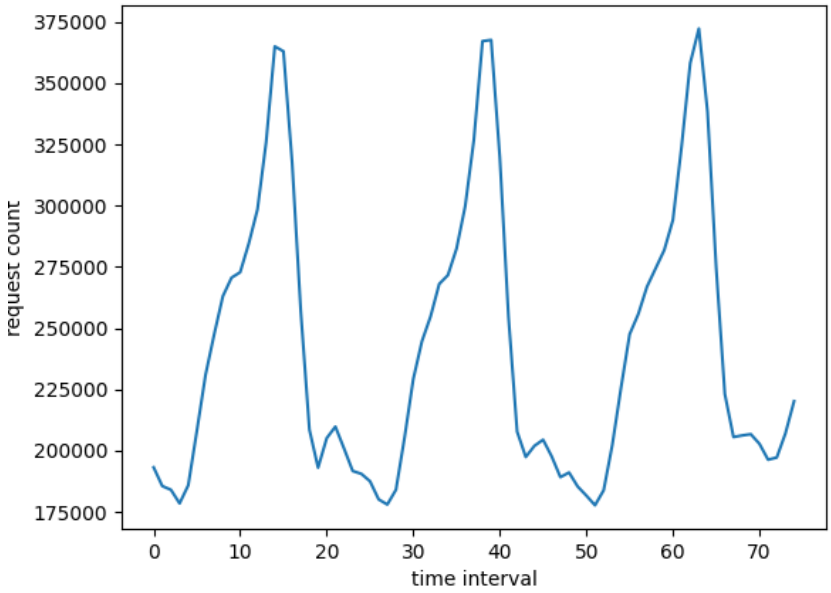

図 5.3: Wikipedia'15(https://wikitech.wikimedia.org/wiki/Analytics/Data_access)

5.1.2 実験環境

本実験での実験環境は図 5.4 に記載した。

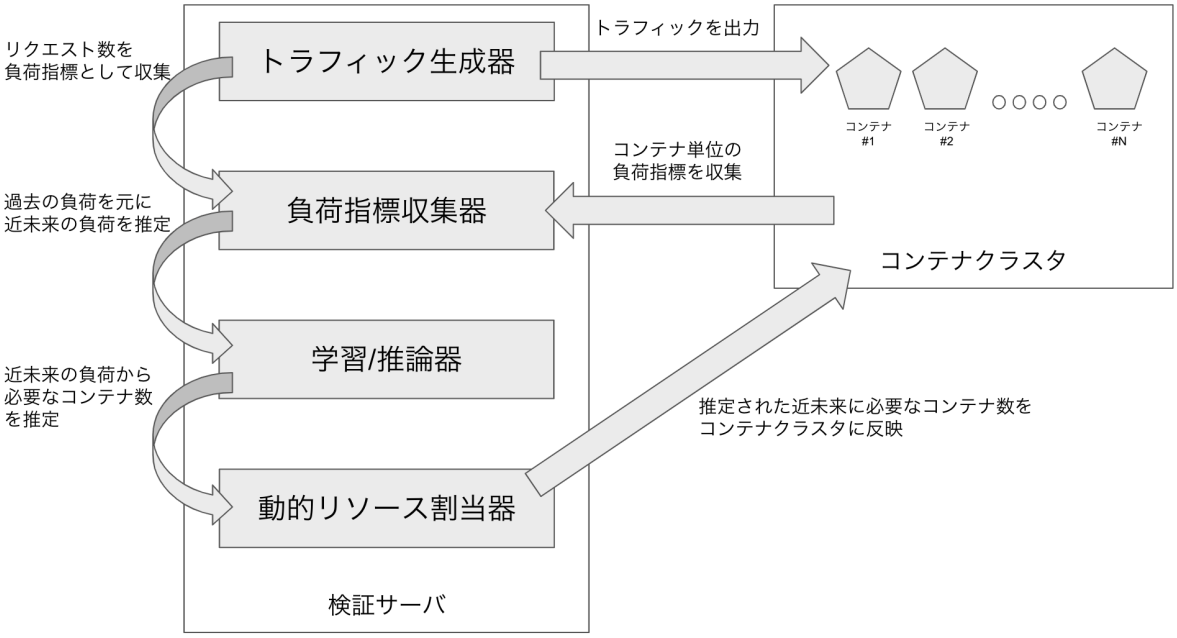

図 5.4: 実験環境

5.1.2.1 コンテナクラスタ

本実験では実験環境として Google Cloud Platform 上で Google Kubernetes Engine と呼ばれるコンテナプラットフォームサービスを利用した．使用した Google Kubernetes Engine はコントローラとノードのバージョンは共に 1.22.8-gke.202 を使用した．また Google Kubernetes Engine では以下のインフラ構成で構築した．

表 5.1: コンテナクラスタに使用した VM の仕様 [2]

OS	VM のタイプ	VM の台数	vCPUs	メモリ
Container-Optimized OS [37]	n1-standard-1	7	1	3.75GB

本実験で使用した Google Kubernetes Engine ではコンテナオートスケーラの Kubernetes を介して，コンテナ 1 個毎に vcpu とメモリに使用制限を設けている．それぞれ vcpu は 0.2 個，メモリは 200MB まで割り当てられるよう設定を行った．Kubernetes はコンテナを起動する際，自身のスケジュールアルゴリズムの設定に則り，コンテナを起動する VM を選択する．今回使用したスケジュールアルゴリズムはデフォルトの設定 [38] を使用した．

5.1.2.2 コンテナ上のアプリケーション

コード 5.1: アプリケーションサーバのソースコード

```
use actix_web::{get, web, App, HttpServer, Responder};
use serde::{Serialize, Deserialize};

#[derive(Debug, Serialize, Deserialize)]
struct FibParams {
    n: usize,
}

#[get("/hc")]
async fn hc() -> impl Responder {
    "OK"
}

fn fib_exec(n: usize) -> usize {
    match n {
        0 => 0,
        1 => 1,
        _ => fib_exec(n - 2) + fib_exec(n - 1)
    }
}

#[get("/fib")]
async fn fib(param: web::Query<FibParams>) -> impl Responder {
    fib_exec(param.n).to_string()
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| App::new()
        .service(hc)
        .service(fib))
        .bind("0.0.0.0:8081")?
        .run()
        .await
}
```

コンテナ上のアプリケーションは起動を速めるために Rust 言語 [39] により記述（ソースコード 5.1）した．外部から HTTP リクエストを受信する度にフィナボッチ数列を計算し，計算が終了次第，HTTP レスポンスを送信する web サーバを構築した．これにより，アプリケーションは CPU バウンドの処理を行うことになる．

5.1.2.3 検証サーバ

本実験では前述したコンテナクラスタとは物理的に独立したサーバとして検証サーバを用意した。

表 5.2: 検証サーバに使用した VM の仕様 [2]

OS	VM のタイプ	VM の台数	vCPUs	メモリ
Debian 11	e2-standard-2	1	2	8.00GB

図 5.4 にあるような, 本実験において様々な機能を搭載している。各機能については以下の通りである.

5.1.2.4 トラフィック生成器

本実験ではトラフィック生成器を用いて 5.1.1 節で述べた web トラフィックを再現した．トラフィック生成器には Gatling [40] を使用した．Gatling が扱う負荷は主に HTTP リクエストである．Gatling は予定する負荷の増減を DSL で記述し，指定したエンドポイントに対して HTTP リクエストを送出し，アクセスログをリアルタイムに出力する．本実験では Gatling を前述した検証サーバ上で実行し，Google Cloud Loadbalancing [41] に対して HTTP トラフィックを送出し，Google Cloud Loadbalancing がコンテナクラスタに対して転送することで負荷を与えた．アクセスログは検証サーバに保存され，負荷指標として利用した.

5.1.2.5 負荷指標の収集器

負荷指標として，1) HTTP リクエスト数，2) コンテナ 1 台あたりの CPU 使用量，3) コンテナ 1 台あたりのメモリ使用量を収集した．1) については前節で説明した Gatling が出力したアクセスログを，2) および 3) はコンテナクラスタ上の Kubernetes Metrics Server [42] に集約された負荷指標を検証サーバ上の kubectl [43] を介して収集した．収集された負荷指標は同じく検証サーバ上にある時系列 DB である InfluxDB [44] に保存した.

5.1.2.6 学習/推論器

前節の負荷指標の収集器にて永続化された負荷指標を参照し，過去の負荷の学習および未来の負荷の推定を行った．4.1 節で説明した提案手法のアルゴリズムを用いて，次のバッチ間隔で必要なコンテナの数を出力する.

5.1.2.7　動的リソース割当器

前節で得られたコンテナの数をコンテナクラスタに適用する．コンテナクラスタに適用する際，kubectl [43] を介しコンテナクラスタ上の kube-apiserver [20] に対して適用するべきコンテナの数を HTTP リクエストとして送信し適用した．

5.1.3　比較対象

Tahir らの研究 [31]（本稿では SampEn モデルと呼ぶ）と Abdullah らの研究 [32]（本研究では Std モデルと呼ぶ）, Kubernetes の Horizontal Pod Autoscaler [21](本研究では HPA モデルと呼ぶ）を提案手法の比較対象とする．

5.1.3.1　SampEn モデル

Tahir らの研究では負荷推定モデルとして ElasticNet 回帰を使用し，推定した負荷に対応するコンテナ数は決定回帰木により求める．バースト検知は Sample Entropy を用いて行っており，事前に設定した閾値を超えるかどうかでバースト現象の発生を判定している．バースト検知された場合は直近 10 回分の推定履歴から最も大きなコンテナ数を適用し，バースト検知されなかった場合は ElasticNet 回帰により推定された負荷に対応するコンテナ数を採用する．本稿の実験ではスライディング学習時のウィンドウサイズについては Sample Entropy が定義されていない値を頻出しない値である 100 を採用する．

5.1.3.2　Std モデル

Abdullah らの研究は SampEn モデルから派生したものであるが，直近 10 回分の負荷推定履歴を使用して標準偏差を計算し，得られた標準偏差の値が 2 より大きいかどうかによりバースト現象の発生を判定する．バースト検知された場合は履歴の中で標準偏差が最大になったときのコンテナ数を適用し，バースト検知されなかった場合は ElasticNet 回帰により推定された負荷に対応するコンテナ数を採用する．また本稿の実験では提案手法と同じウィンドウサイズ（50）を使用する．

5.1.3.3　HPA モデル

HPA モデルは現在コンテナプラットフォームとしてデファクトスタンダードである Kubernetes に搭載されたコンテナ数をオートスケールするコントローラである．以下のような設定（ソースコード 5.2) を Kubernetes に適用することで対象のコンテナをオートスケールする．

コード 5.2: HorizontalPodAutoscaler の設定コード

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: query-api-hpa
  namespace: query-api
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: query-api-app
  minReplicas: 5
  maxReplicas: 40
  targetCPUUtilizationPercentage: 90
```

本実験で用いた設定（ソースコード 5.2) ではスケールするコンテナ数の下限（$minReplicas$）を 5, 上限（$maxReplicas$）を 40, コンテナ数をスケールさせる閾値（$targetCPUUtilizationPercentage$）を 90 とした.

この設定値を元に HPA モデルは式 5.1 を用いて、必要なコンテナ数を算出する.

$$desiredReplicas = ceil\left(currentReplicas * \frac{currentMetricValue}{desiredMetricValue}\right) \tag{5.1}$$

ここで, $desiredReplicas$ は必要なコンテナ数, $currentReplicas$ は現在のコンテナ数, $currentMetricValue$ は現在の負荷の値, $desiredMetricValue$ は理想の負荷の値 $(= targetCPUUtilizationPercentage)$ である.

本実験における HPA モデルは, 全コンテナの CPU 負荷の平均値 $currentMetricValue$ が $desiredMetricValue$ より低い場合、オートスケーラは配置内のコンテナの数を $minReplicas$ を下限として減らす. 逆に $desiredMetricValue$ を超えると、オートスケーラは配置内のコンテナの数を $maxReplicas$ を上限として増やす, といった動きをする.

5.1.4 実験内容

5.1.2 節で述べた実験環境において，4.1 節で提案した手法と 5.1.3 節で示した比較対象の 3 つのモデルについて実験を行った．負荷生成ツールを用いて，5.1.1.1 節（以下 WorldCap），5.1.1.2 節 (以下 SWIM#1，SWIM#2)，5.1.1.3 節（以下 Wikipedia）のデータからトラフィックの特徴を抽出し，それぞれ再現した．再現したトラフィックは HTTP 通信として Kubernetes 上のアプリケーションに送信され，アプリケーションは処理を行い，レスポンスを生成して返答する．

5.1.4.1 評価指標

本実験では以下の評価指標を用いた.

5.1.4.2 リクエスト失敗数

負荷で用いた HTTP 通信は以下の条件を満たした場合に成功したとみなす.

- HTTP レスポンスステータスが 200 だった場合
- 負荷生成ツールが HTTP リクエストを送信してから 1000msec 以内に HTTP レスポンスが返ってきた場合

失敗した数をサービス品質を測る 1 つの評価指標として採用した.

5.1.4.3 レスポンス時間の 99 パーセンタイル (msec)

負荷生成ツールから HTTP リクエストを送信してから，アプリケーションから HTTP レスポンスが返ってくるまでの時間（レスポンス時間）に関して，99 パーセンタイルの値をサービス品質を測る 1 つの評価指標として採用した.

5.1.4.4 平均二乗誤差（MSE）

実際に送出したトラフィック量に対して予め学習させた決定回帰木を用いて出力したコンテナ数を理想とし，推定に基づき実際に変化させたコンテナ数との MSE（Mean Squared Error）をサーバ資源の効率性を測る 1 つの評価指標として採用した.

5.1.5 実験結果

表 5.3: 実験結果

再現負荷	モデル	リクエスト失敗数/総数	返答時間の 99%タイル (msec)	コンテナ数推移の MSE
WorldCup	Proposal	65/548874	85.9939	3.8207
	Std	2537/548874	463.9645	26.5503
	SampEn	2464/548874	664.5799	20.1853
	HPA	0/548874	70.1982	427.3342
SWIM#1	Proposal	0/644567	76.5469	3.5199
	Std	45/644567	228.4452	43.9405
	SampEn	50/644567	110.8375	28.7488
	HPA	0/644567	70.2215	403.0624
SWIM#2	Proposal	0/249952	119.6593	4.0468
	Std	1/249952	367.5863	49.0919
	SampEn	7475/249952	998.7673	66.6986
	HPA	0/249952	120.0219	150.9566
Wikipedia	Proposal	0/263322	106.6347	5.2798
	Std	149/263322	176.5385	27.6993
	SampEn	178/263322	682.2893	28.1205
	HPA	27/263322	102.9789	579.6785

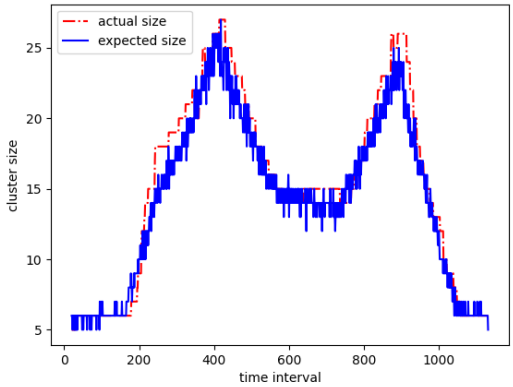

図 5.5: コンテナ数推移
Worldcap Proposal

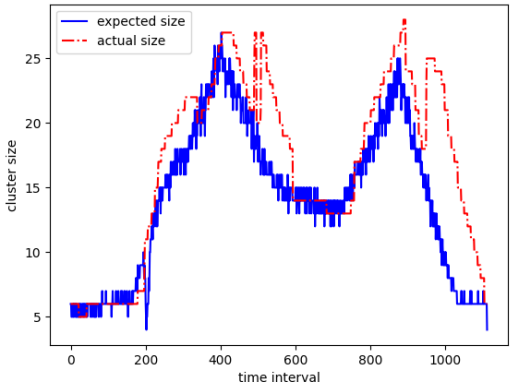

図 5.6: コンテナ数推移
Worldcap Sampen

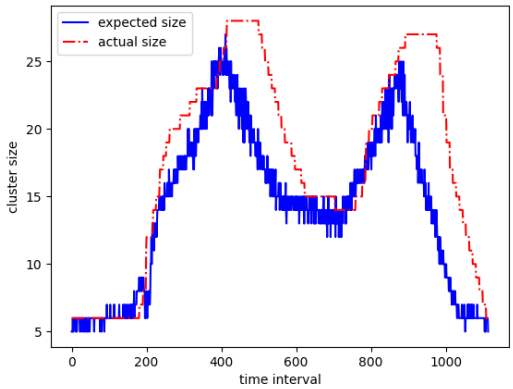

図 5.7: コンテナ数推移
Worldcap Std

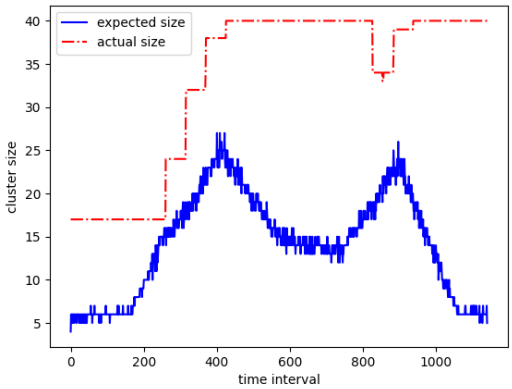

図 5.8: コンテナ数推移
Worldcap HPA

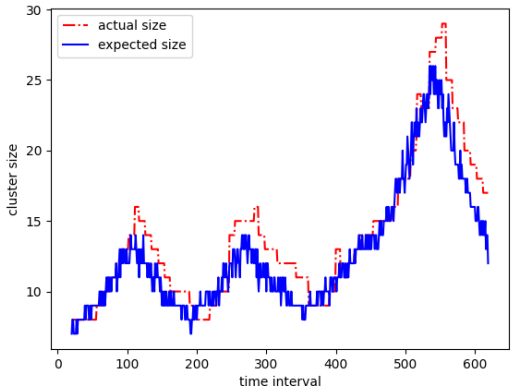

図 5.9: コンテナ数推移
SWIM#1 Proposal

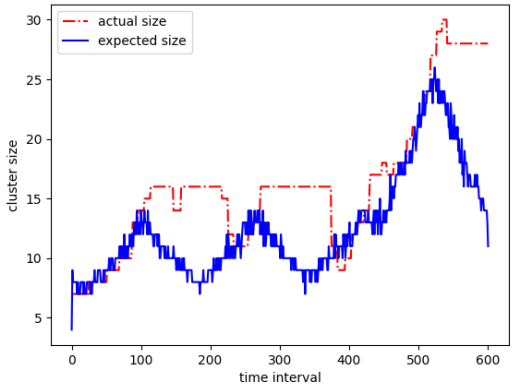

図 5.10: コンテナ数推移
SWIM#1 SampEn

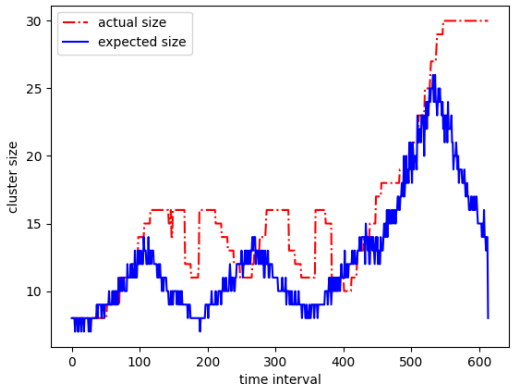

図 5.11: コンテナ数推移
SWIM#1 Std

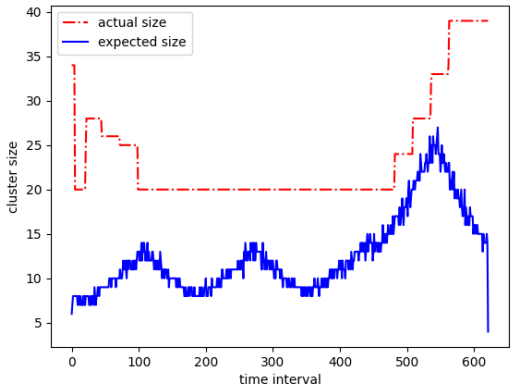

図 5.12: コンテナ数推移
SWIM#1 HPA

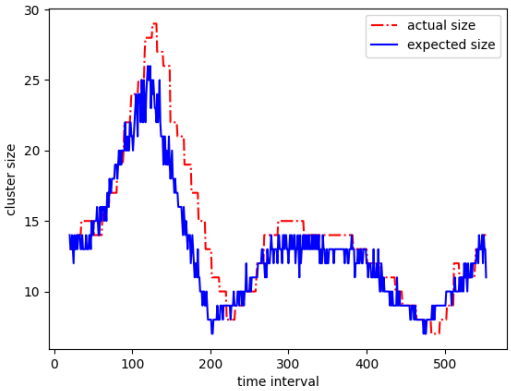

図 5.13: コンテナ数推移
SWIM#2 Proposal

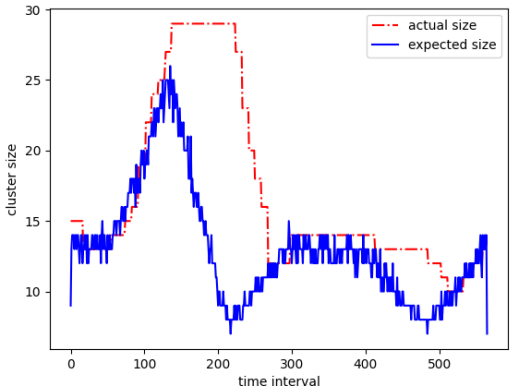

図 5.14: コンテナ数推移
SWIM#2 SampEn

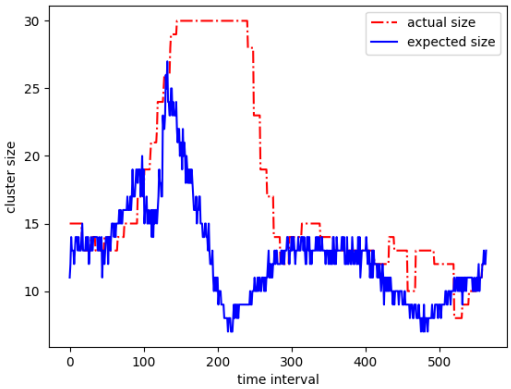

図 5.15: コンテナ数推移
SWIM#2 Std

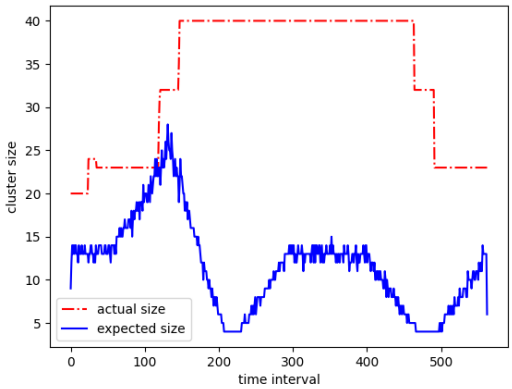

図 5.16: コンテナ数推移
SWIM#2 HPA

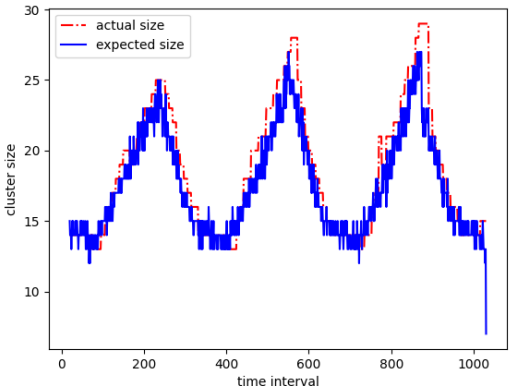

図 5.17: コンテナ数推移
Wiki Proposal

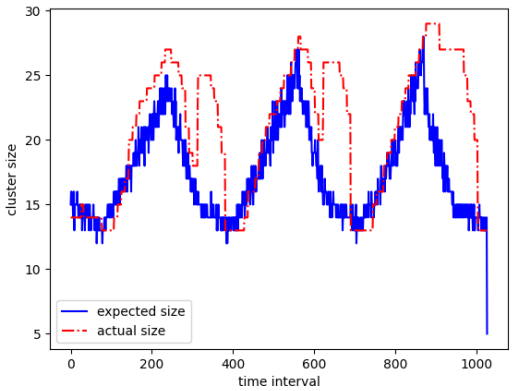

図 5.18: コンテナ数推移
Wiki SampEn

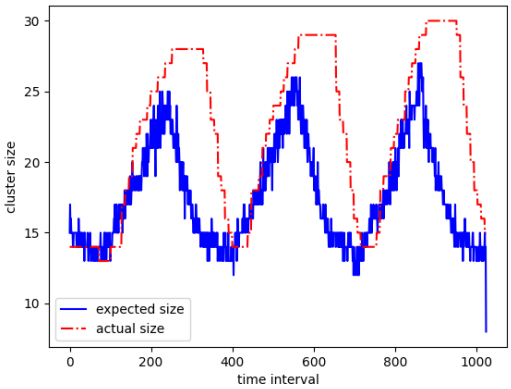

図 5.19: コンテナ数推移
Wiki Std

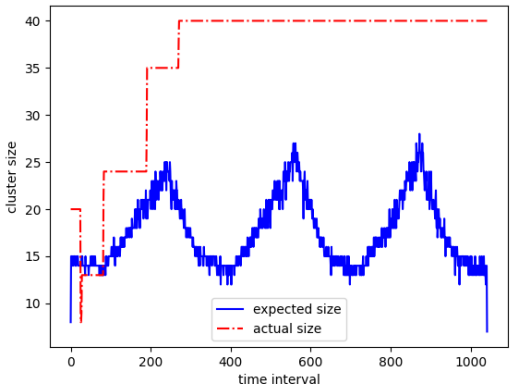

図 5.20: コンテナ数推移
Wiki HPA

5.1.5.1 レスポンスタイム推移

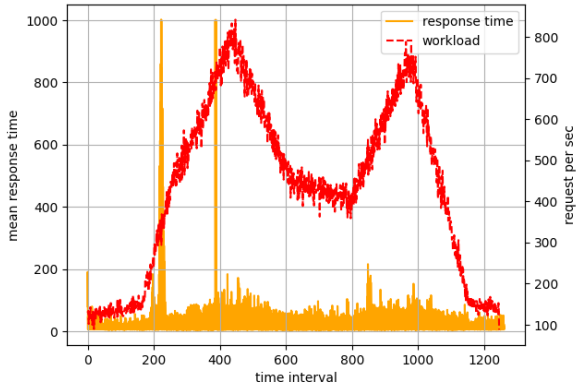

図 5.21: レスポンス時間推移
Worldcap Proposal

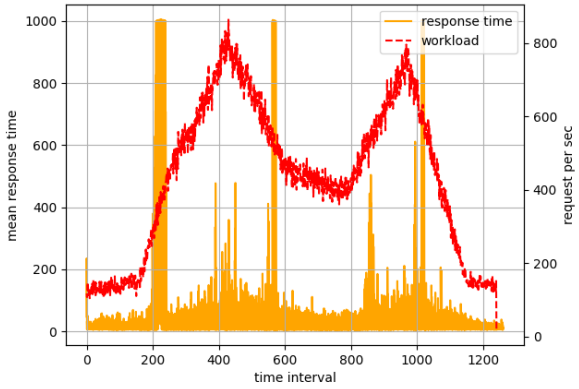

図 5.22: レスポンス時間推移
Worldcap SampEn

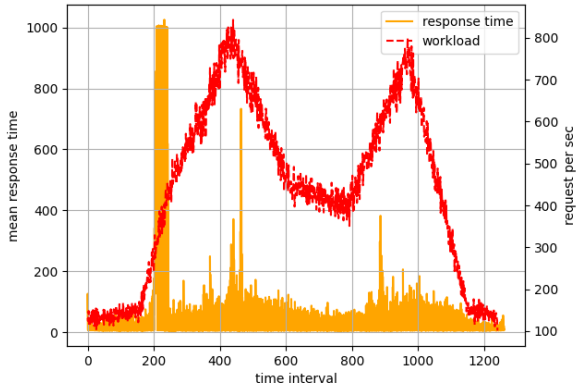

図 5.23: レスポンス時間推移
Worldcap Std

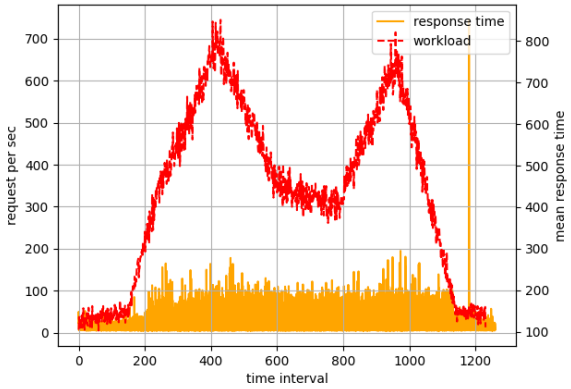

図 5.24: レスポンス時間推移
Worldcap HPA

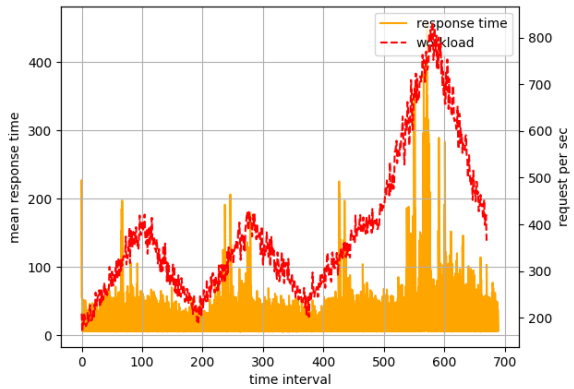

図 5.25: レスポンス時間推移
SWIM#1 Proposal

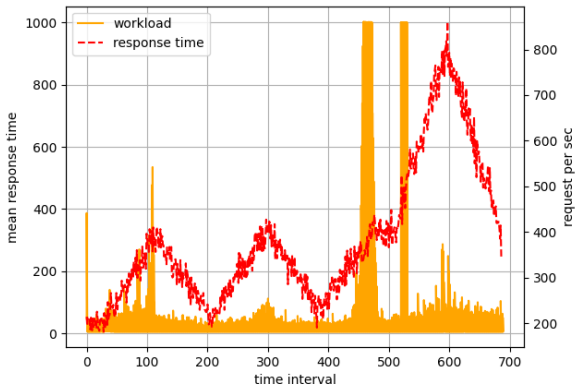

図 5.26: レスポンス時間推移
SWIM#1 SampEn

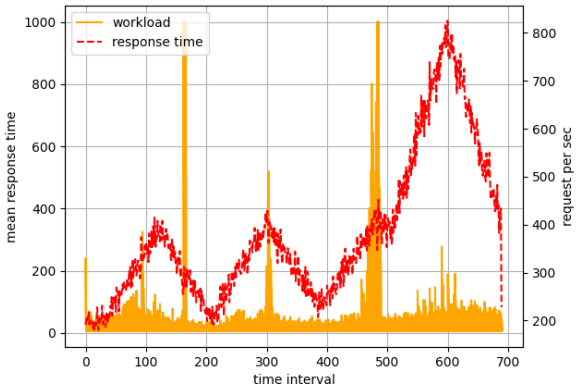

図 5.27: レスポンス時間推移
SWIM#1 Std

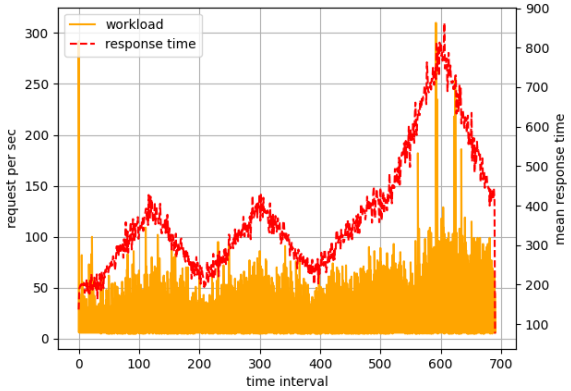

図 5.28: レスポンス時間推移
SWIM#1 HPA

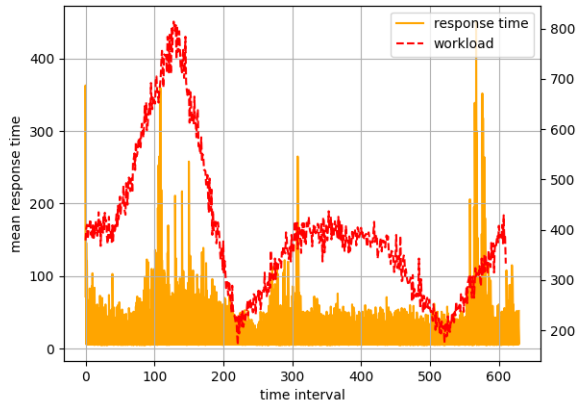

図 5.29: レスポンス時間推移
SWIM#2 Proposal

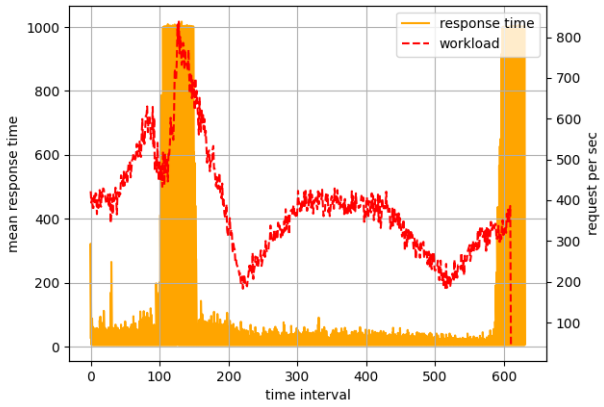

図 5.30: レスポンス時間推移
SWIM#2 SampEn

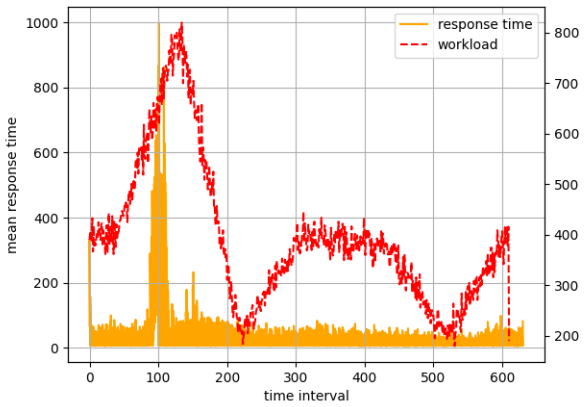

図 5.31: レスポンス時間推移
SWIM#2 Std

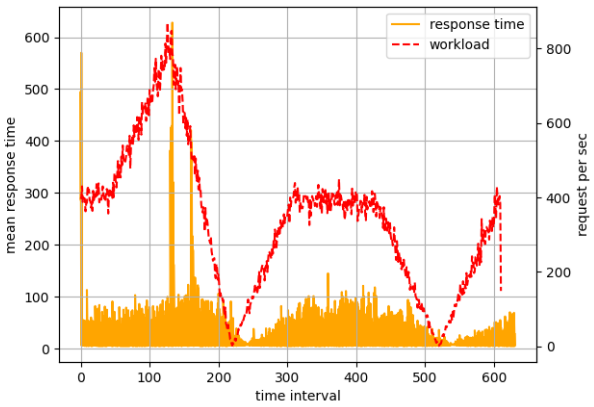

図 5.32: レスポンス時間推移
SWIM#2 HPA

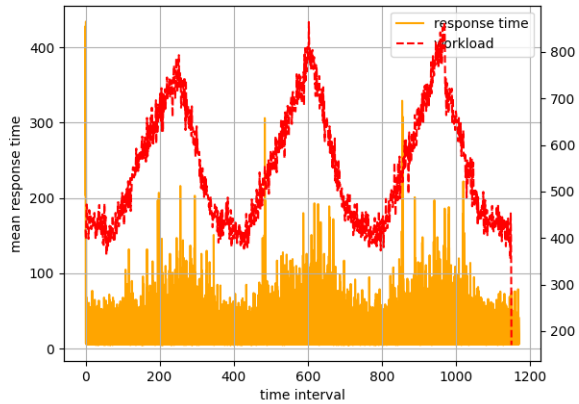

図 5.33: レスポンス時間推移
Wiki Proposal

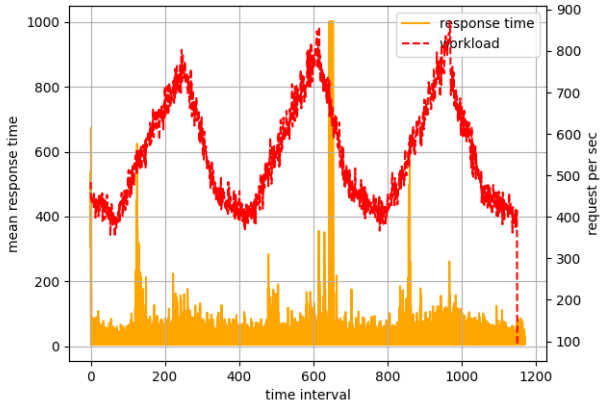

図 5.34: レスポンス時間推移
Wiki SampEn

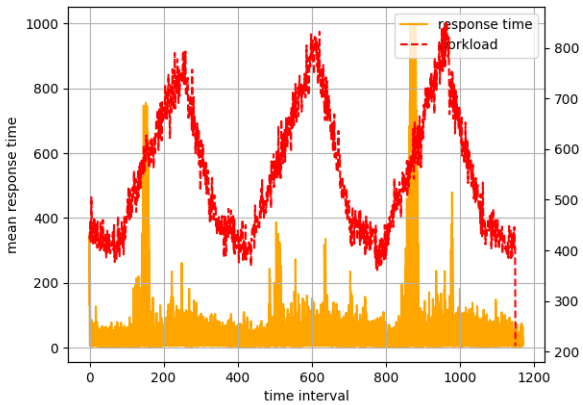

図 5.35: レスポンス時間推移
Wiki Std

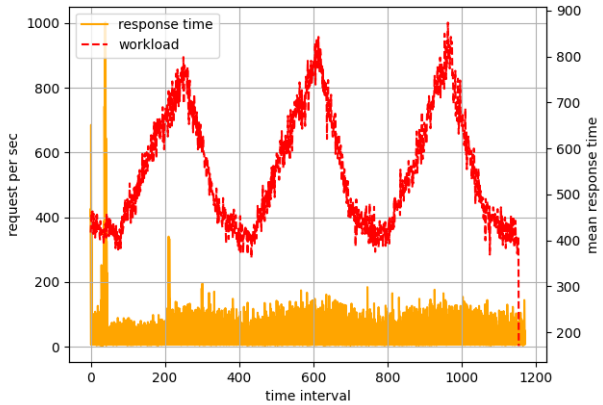

図 5.36: レスポンス時間推移
Wiki HPA

5.1.5.2 WorldCap

WorldCap の実験では，リクエスト失敗数に関して提案手法は SampEn モデル, Std モデルと比べて大きく改善されている．特に提案手法のレスポンス時間（図 5.21）は SampEn モデル, Std モデル（図 5.22，図 5.23）と比較し，200〜230 秒付近のバーストトラフィックに対してレスポンス時間が抑えられている．またコンテナ数の推移も提案手法（図 5.5）が他の 3 つのモデル（図 5.6，図 5.7, 図 5.8）と比べて理想的である．

5.1.5.3 SWIM#1

SWIM#1 の実験では，提案手法はリクエスト失敗数がゼロであり，SampEn モデル, Std モデルと比べて改善されている．特に提案手法のレスポンス時間（図 5.25）は SampEn モデル, Std モデル（図 5.26，図 5.27）と比較し，440〜480 秒付近のバーストトラフィックに対してレスポンス時間が抑えられている．またコンテナ数の推移も提案手法（図 5.9）が他の 3 つのモデル（図 5.10，図 5.11, 5.12）と比べて理想的である．

5.1.5.4 SWIM#2

SWIM#2 の実験では，提案手法はリクエスト失敗数がゼロであり，SampEn モデル, Std モデルと比べて改善されている．一方で他の 3 つのモデル（図 5.30，図 5.31）と比べて、提案手法（図 5.29）のレスポンス時間は 240〜310，560〜590 秒付近で長くなっている．コンテナ数の推移に関しては，提案手法（図 5.13）が他の 3 つのモデル（図 5.14，図 5.15，図 5.16）と比べて理想的である．

5.1.5.5 Wikipedia

Wikipedia の実験では，提案手法のリクエスト失敗数はゼロであり，SampEn モデル, Std モデルと比べて改善されている．一方で 2 つのモデル（図 5.34，図 5.35）と比べて、提案手法（図 5.33）のレスポンス時間は平均的に長くなっているが，リクエスト失敗数がゼロであることからも，急激なレスポンス時間の増大が回避できている．コンテナ数の推移は提案手法（図 5.17）が他の 3 つのモデル（図 5.18，図 5.19，図 5.20）と比べて理想的である．

5.1.6 考察

本実験では他の 3 つのモデルと比較し，総合して提案手法が優れた結果を示した．特に MSE については，提案手法は他の 3 つのモデルと比べて大きく改善した．これは今回比較した Std モデルおよび SampEn モデルがバースト事象を検知したときに，過去の 10 回の推定結果から最大値を選ぶことに起因する．この挙動は上昇傾向のトラフィックに対しては有効であるが，下降傾向のトラフィックに対しては過大評価となる．また，SampEn モデルで使用した Sample Entropy はバースト発生時に値が定義されない場合があり，その場合はバースト検知に失敗する．

レスポンス時間について，提案手法は 99 パーセンタイルにおいて SampEn モデル, Std モデルに比べて優れた結果を示した．SampEn モデル, Std モデルのレスポンス時間が大きく鈍化してタイムアウトする原因は，バースト発生時にサーバ資源の提供が遅れることであるが，これに対して提案手法は十分に機能している

といえる．一方で提案手法は，SWIM#2 や Wiki では大きなレスポンス時間の鈍化は見受けられなかったものの、急速にトラフィック量が低下した後に再び急増した際に，他の 2 つのモデルに比べてレスポンス時間が長くなる場合も観測された．これは他の 2 つのモデルは過去履歴のデータを用いて過大に評価することが，この現象に上手く適応できているためである．

HPA モデルはレスポンス時間に対しては優秀な結果 (図 5.24, 図 5.28, 図 5.32, 図 5.36) ではあったが, 他の 3 つのモデルや理想的なコンテナ数の推移と比べて, 過大なリソースを消費していることが要因である．今回 HPA モデルは CPU 使用率が 90%以上になったタイミングで増加させるようなロジックを取り, 30 秒ごとにメトリクスを取得しコンテナ数の増減を図るが, 3 節で述べた通り, コンテナ数の増加は過去 3 分以内に再スケーリングがなかった場合にのみ実行する．一方減少は実行条件が揃っていても最後の再スケーリングから 5 分間待つ実装になっているため, どちらも実行タイミングが間に合わないケースも見られた．さらにコンテナ数の増加量も一度に既存コンテナ数の倍に近い数に増やしている．総じて今回の 4 パターンの負荷の増減に対して効率の良いコンテナ数に調整出来ているとは言い難い.

第6章　おわりに

本研究では，クラウド上で予測困難なバーストを発生しうるサービスに対して，効率的にサーバ資源を提供するオートスケーリング手法を実際のクラウド環境上で構築した．本稿では特に数分程度でバースト現象を生じるようなケースに着目し, 既存研究である Tahir らの研究と Abdullah らの研究, コンテナ管理ツールのデファクトスタンダードである Kubernetes の Horizonal Pod Autoscaler 機能と提案手法との比較を行った.

特にバースト検知については Tahir らの研究で採用された Sample Entropy, Abdullah らの研究で採用された標準偏差に対しては,4.2 節にて比較評価し, Fuzzy Entropy が優れた性能を示した. それに加えて 5 節にて, 実際のクラウド環境上に実験環境を構築し, 四つの負荷パターンからいずれも提案手法が既存研究と比べて優れた性能を持つことを示した.

今後の課題や研究の展望として， 以下に述べる.

一つ目は本研究は負荷に対してコンテナ数を変更させたが, コンテナを実行するサーバの数は固定して行った．今後の展望としてサーバの数を変動させる場合はサーバの起動時間やコンテナの起動時間の影響が非常に大きいため, 本研究で 4.1 節で述べたアルゴリズムや Kubernetes 自体のコンテナ配置アルゴリズムに対して拡張を施す必要がある.

二つ目は本稿で用いたアルゴリズムでは短時間の学習データでバースト事象が生じるサービスに対して最適化されたシステムであった. このアルゴリズムを拡張しより長時間の学習データに対して動作するアルゴリズムを検討した際には, 現状のアルゴリズムでウィンドウ間隔を延ばすだけでは, 他の十分に学習, 推論時間を費やせる RNN や LSTM と比べて推論精度は劣るため, 違ったアルゴリズムを検討する必要がある.

三つ目は, 本研究ではスライディングウィンドウ学習にて, 直前の時系列データのデータからのみ学習データとしたが, 過去の特徴的なデータを学習し, フィードバックする事でより高い推論精度が出る可能性が考えられる.

最後にバースト検知器について. 本研究ではバースト検知器として Fuzzy Entropy を採用したが, より優れたバースト検知器について検討する際に, 以下のような改善案が考えられる.

- Fuzzy Entropy にて近似度を測定するメンバーシップ関数を指数関数より, より適切な関数の適用.

- より頑強性, バースト検知性能に優れた新たなエントロピー関数の考案.
- Fuzzy Entropy によるウィンドウあたりの自己比較によるエントロピー計算だけでなく他の適切なウィンドウとの交差エントロピー計算の導入

などが挙げられる.

謝 辞

本研究を進めるにあたり，田中教授には熱心にご指導いただき，多大な助言を賜りましたこと，心より感謝致します．また，日ごろの議論において的確な助言をして頂いた田中研究室の皆様にも御礼申し上げます.

私の修士論文がこの形で完成することができたのは、多くの方々の厚い支援と助けがあってこそです。その人々への深い感謝と敬意をここに表します。

まず初めに, 田中教授には一貫して私の研究に対する助言と指導をいただき, 心から感謝申し上げます．当初のテーマから大きく方向性が変わるという難題に直面したときでも, 教授は私と粘り強く議論を重ね最終的に一つの研究成果として成立するまで導いてくださいました. 本当にありがとうございました.

研究室の先輩方や同期の皆様、そして後輩たちにも心からの感謝の意を表します。皆様と活発に議論させて頂き, より良い研究ができました.

副テーマ指導の井口教授にも心から感謝申し上げます．研究計画書の立案の際にはご助言をいただき, 副テーマの際にも最終的には対外発表を経験させていただきました. 本当にありがとうございました.

この修士論文は、私の一人の業績ではなく、これまで私を支えてくれた全ての人々の成果と言えます．これからも私の研究が皆様のご期待に応えられるよう, さらに努力し続ける所存です.

参考文献

[1] David Mayor, Deepak Panday, Hari Kala Kandel, Tony Steffert, Duncan Banks, "Ceps: An open access matlab graphical user interface (gui) for the analysis of complexity and entropy in physiological signals," *Entropy 2021*, vol. 23, no. 3, p. 321, 2021.

[2] Google Cloud inc, "Compute Engine general-purpose machine family," 2023. https://cloud.google.com/compute/docs/general-purpose-machines.

[3] G. E. P. Box, David A Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," vol. 65, no. 332, pp. 1509–1526, 1970.

[4] G. E. P. Box, G. M. Jenkins, "Time series analysis: Forecasting and control, operational research quarterly," vol. 22, no. 2, pp. 199–201, 1970.

[5] Weiting Chen, Zhizhong Wang, Hongbo Xie, and Wangxin Yu, "Characterization of surface emg signal based on fuzzy entropy," *IEEE Transactions on Biomedical Engineering*, vol. 15, no. 2, pp. 266–272, 2007.

[6] C. E. Shannons, *A Mathematical Theory of Communication*. No. 379–423, Nokia Bell Labs, 1948.

[7] Alfonso Delgado-Bonal, Alexander Marshak, "Approximate entropy and sample entropy: A comprehensive tutorial," *MDPI JOURNAL ENTROPY*, pp. 4–5, 2019.

[8] Pincus, S, "Approximate entropy (apen) as a complexity measure," *Chaos Int. J. Nonlinear Sci*, no. 110-117, p. 5, 1995.

[9] Ruqiang Yan and Robert X. Gao, "Approximate entropy as a diagnostic tool for machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 21, no. 2, pp. 824–839, 2007.

[10] Srinivasan, Vairavan and Eswaran, Chikkannan and Sriraam, Natarajan, "Approximate entropy-based epileptic eeg detection using artificial neural

networks," *IEEE Transactions on Information Technology in Biomedicine*, vol. 11, no. 3, pp. 288–295, 2007.

[11] Fleisher, LA and Pincus, SM and Rosenbaum, SH, "Approximate entropy of heart rate as a correlate of postoperative ventricular dysfunction," *Anesthesiology*, vol. 78, p. 683—692, April 1993.

[12] Richman, J.S., Moorman, J.R., "Physiological time-series analysis using approximate entropy and sample entropy," *Am. J. Phys. Heart Circul. Physiol*, vol. 278, no. 6, pp. H2039–H2049, 2000.

[13] Amir Omidvarnia, Mostefa Mesbah, Mangor Pedersen, Graeme Jackson, "Range entropy: A bridge between signal complexity and self-similarity," *Entropy*, vol. 20, no. 12, p. 962, 2018.

[14] Peng Li, Chengyu Liu, Ke Li, Dingchang Zheng, Changchun Liu, Yinglong Hou, "Assessing the complexity of short-term heartbeat interval series by distribution entropy," *Medical & Biological Engineering & Computing*, 2014.

[15] L. A. Zadeh, "Fuzzy sets, " information control," vol. 8, p. 338–353, 1965.

[16] Docker inc, "Docker: Accelerated, containerized application development," 2023. https://www.docker.com.

[17] Kubernetes io, "Production-grade container orchestration," 2022. https://kubernetes.io.

[18] Ahmed Ali-Eldin, Oleg Seleznjev, Sara Sjostedt-de Luna, Johan Tordsson, Erik Elmroth, "Measuring cloud workload burstiness," *IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014.

[19] Mehta, Amardeep and Dürango, Jonas and Tordsson, Johan and Elmroth, Erik, "Online spike detection in cloud workloads," pp. 446–451, 2015.

[20] Kubernetes io, "Horizontal pod autoscaling," 2023. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/.

[21] Kubernetes io, "kube-apiserver," 2023. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/.

[22] Mahesh Balaji, Ch. Aswani Kumar, G. Subrahmanya V. R. K. Rao, "Nonlinear analysis of bursty workloads using dual metrics for better cloud resource management," *IEEE Transactions on Biomedical Engineering*, no. 1966-1972, p. 55, 2008.

[23] Hurst H., "Long-term storage capacity of reservoirs," *Transactions of the American Society of Civil Engineers*, p. 770, 1951.

[24] Langstom Nashold, Rayan Krishnan, *Using LSTM and SARIMA Models to Forecast Cluster CPU Usage*. PhD thesis, Stanford University, 2020.

[25] Alex Graves, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, 1997.

[26] Bruno Lopes Dalmazo, Joao P. Vilela, Marilia Curado, "Predicting traffic in the cloud: A statistical approach," *IEEE Third International Conference on Cloud and Green Computing*, p. 1, 2013.

[27] B. L. Dalmazo, J. P. Vilela, and M. Curado, "Online traffic prediction in the cloud: a dynamic window approach," *IEEE Cloud and Green Computing*, pp. 9–14, 2014.

[28] Min Sang Yoona , Ahmed E. Kamalb , and Zhengyuan Zhuc, "Requests prediction in cloud with a cyclic window learning algorithm," *IEEE Globecom Workshops*, pp. 4–5, 2016.

[29] Shuja-ur-Rehman Baig, Waheed Iqba, Josep Lluis Berral, David Carrera, "Adaptive sliding windows for improved estimation of data center resource utilization," *Future Generation Computer Systems*, 2020.

[30] T. Hirayama, T. Miyazawa, M. Jibiki, and V. P. Kafle, , "Sparse regression model-based relearning architecture for shortening learning time in traffic prediction," *IEICE Trans. on Info. & Systems*, vol. E104-D, no. 5, pp. 606–616, 2021.

[31] Fatima Tahir, Muhammad Abdullah, Faisal Bukhari, Khaled Mohamad Almustafa, "Online workload burst detection for efficient predictive autoscaling of applications," *IEEE ACCESS*, pp. 4–5, 2020.

[32] Muhammad Abdullah , Waheed Iqbal , Josep Lluis Berral , Jorda Polo , and David Carrera, "Burst-aware predictive autoscaling for containerized microservices," *IEEE TRANSACTIONS ON SERVICES COMPUTING*, pp. 4–5, 2022.

[33] Dominique Makowski, "The python toolbox for neurophysiological signal processing," 2023. https://github.com/neuropsychology/NeuroKit.

[34] Neil SulakheLorin J. EliasLisa Lejbak, "Hemispheric asymmetries for gap detection depend on noise type," *SCIENCE DIRECT*, 2003.

[35] Carter, Mancini, Bruce, Ron, "Op amps for everyone," *Texas Instruments*, pp. 10—-11, 2009.

[36] Apache Software Foundation, "Apache hadoop," 2023. https://hadoop.apache.org.

[37] Google Cloud inc, "Container-optimized os documentation," 2023.

[38] Kubernetes io, "Scheduler configuration," 2022. https://kubernetes.io/docs/reference/scheduling/config.

[39] "Rust programming language," 2023. https://www.rust-lang.org.

[40] Gatling Corp, "Gatling," 2023. https://gatling.io/open-source/.

[41] Google Cloud inc, "Cloud load balancing," 2023. https://cloud.google.com/load-balancing.

[42] Kubernetes io, "Kubernetes metrics server," 2023. https://github.com/kubernetes-sigs/metrics-server.

[43] Kubernetes io, "Kubectl," 2023. https://github.com/kubernetes/kubectl.

[44] InfluxData Inc, "It's about time. build on influxdb.," 2023. https://www.influxdata.com.

本研究に関する発表論文

[1] 横山 尚弥, 田中 清史,“ 突発的な Web トラフィックの増減に適応する fuzzy entropy を用いたオートスケーリングシステム ”，マルチメディア，分散，協調とモバイル DICOMO2023 シンポジウム， 2023 (優秀プレゼンテーション賞)