Vol.2017-OS-139 No.4
2017/3/1

# 自動運転を支えるアプリケーション安定実行制御基盤

小比賀亮仁[†1]　金子紘也[†1]　伊藤暢彦[†1]　岩井孝法[†1]

**要**：本稿は自律モビリティシステムを支えるアプリケーション安定実行制御基盤について述べる。 自律運転技術は、これまでの車両内のセンサ情報による自動走行実現が中心課題であったが、現在は、道路に設置されたカメラや無線機器によって車両の状況を外から監視する、「路側システム」が重要であると考えられており、路側システムを実現する具体的な方法として、携帯電話網とエッジコンピューティングを利用する方法が最も現実的であると考えられている。しかしながら、エッジサーバでは、近傍のセンサ情報の一時集約機能や、不審者の監視など、多種多様なアプリケーションが動作しており、このような環境においてはアプリケーションの静的な優先度付が難しく、リクエストに対する応答時間が著しく変動する。応答時間が変動すると車両に安定して情報提供をすることができない。本稿では、静的な優先度付けが難しいような環境において、制御工学のフィードバック制御に基づく、応答時間の測定・分析・制御をミリ秒単位で実行するアプリケーション安定実行制御基盤を提案する。アプリケーション安定実行制御基盤は、測定対象となるアプリケーションの応答時間を測定し、当該測定値が応答時間の目標値を超過するような場合は、優先度を上げるなど適切なリソース制御とそのパラメータを計算・実行することによって、静的な優先度付けが難しいようなエッジコンピューティング環境においても、アプリケーションの応答時間を均一に保つことが可能となる。

**キーワード**：負荷分散、スケジューリングアルゴリズム、エッジコンピューティング

# Application response time regulation system for auto mobility system

AKIHITO KOHIGA[†1]　HIROYA KANEKO[†1]
NOBUHIKO ITOH[†1]　TAKANORI IWAI[†1]

***Abstract***: Nowadays, research and development activity on auto mobility technologies are focusing on “vehicle to infrastructure communication system (V2I)”, instead of “vehicle to vehicle communication system (V2V)” to compensate for difficult situations where V2V can’t see the other side of a blind corner which a big building makes and an express ramp which the other vehicles run on the upper road. Practically, telecommunication (LTE) and mobile edge computing system will be used for embodiment of such V2I system. However it is difficult to keep the response time between vehicle and infrastructure stable. Because there are many applications running on edge computing environment and execution of them makes fluctuation of the response time. This kind of response time fluctuation is critical for the V2I system. We suggest an application execution environment based on a feedback mechanism. This system continuously monitor the response time of a target application and if it degrades, the system controls the amount of resource (CPU time slice etc) to be given, keeping the response time stable. We also made a simulation system and evaluated it. This evaluation shows that the system has good effect for this objective.

***keywords***: Load balancing, scheduling algorithm, edge computing

## 1. 研究背景

近年、自動運転技術の研究開発が盛んに行われている。自動運転技術は、これまで車両内にセンサやカメラを搭載して、車両自身が道路や障害物を認識するための技術開発が中心に行われてきた[1]。しかしながら、見通しの悪い交差点での衝突防止や、高速ランプ下からの合流など、単体車両の自動化では解決が難しい状況が多く存在することが明らかになりつつある。このように、単体車両の自動化では解決が難しいような状況を解決するために、車両の外側、つまり道路側にカメラや無線機器を備えて、車の状況を外から監視する「路側システム」に注目が集まりつつある。**我々の研究目的は、自動運転を支える路側システムの安定動作である。**

### 1.1 自動運転を支える路側システム

**図 1** に単純化した路側システムの構成図を示す。図に示す通り、路側システムは、車両、基地局(eNodeB)、キャリア設備、クラウドから構成されており、キャリア設備内には、電話交換機網(EPC)とエッジサーバが備えられている。エッジサーバとは、キャリア設備に併設されている計算機を指している。エッジサーバの設置は 2 つのメリットがある。

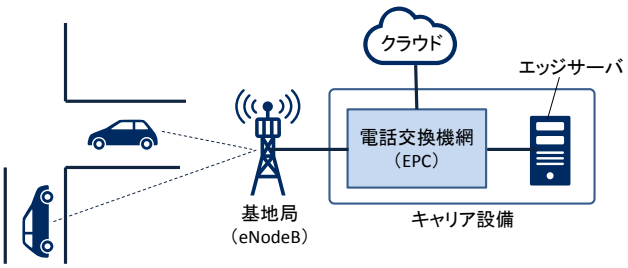


**図 1 路側システム**

1 つ目は「通信遅延の減少」である。エッジサーバは、携帯端末からインターネットへのアクセスする前段に配置されているため、エッジサーバ上で実行される処理は、インターネット上に配置されている計算機で実行される処理

†1 日本電気株式会社
NEC Ltd.

ⓒ 2017 Information Processing Society of Japan

よりも、通信遅延が極めて短い。よって、車両の衝突判定、画像認識（監視カメラ）など、特にリアルタイム性を必要とするようなサービスが実行される。例えば、図に示す路側システムを使って衝突判定を実施する場合は、車両が自身の位置情報をキャリア設備に併設されているエッジサーバに送信し、エッジサーバは、各車両から贈られてくる位置情報を使って衝突判定処理を実施し、仮に衝突の危険性がある場合は、該当する車両に危険信号を通知する。我々の研究開発において、位置情報は、「100 ミリ秒程度の短い周期で収集・分析される」と仮定している[2]。

２つ目は、「通信帯域の削減」である。センサから集めたデータをクラウドに送信する前に、エッジサーバ上で一時集約して、その結果をクラウドへ送信することによって、インターネットに流れるデータ量を削減できる。例えば、センサ情報から得られたデータをエッジサーバで一時的に保存しておいて、ある時刻になったら、当該データ群の統計上の特徴（データの平均や分散など）のみをクラウドへ送信する場合などが、通信帯域の削減例として挙げられる。

### 1.2 ダイナミックマップ構想

路側システムは、自動車制御だけではなく、自動車に関連するあらゆる情報を処理する基盤として利用される。路側システムの考え方をさらに拡大したのがダイナミックマップ[3]である。**図 2** にダイナミックマップの構造を示す。

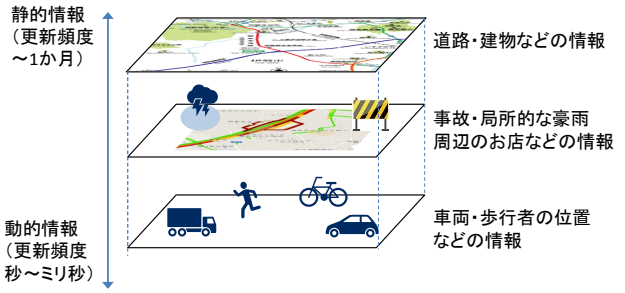


**図 2 ダイナミックマップ**

ダイナミックマップは、情報の更新頻度が異なる情報群を階層構造として表現し、状況に応じて、それらの情報を組み合わせて出力する自動車関連情報の集合体である。ダイナミックマップは、これまで VICS 等で取得していた事故・渋滞・交通規制などの情報の他に、天気予報、周辺地域情報、車両・歩行者などの位置情報を一元化して、より幅広いユーザが利用できるような情報基盤の提供を目指している。ダイナミックマップは、個々の車両の状況に応じて情報の収集・分析・展開を実施するため、多種類・多数のサービスプログラムが並行して動作することになる。例えば、位置情報と道路情報をダイナミックマップから取得して、それらの情報をもとに衝突判定を実施したり、事故や局所的な天気予報などから、渋滞の発生個所などを予測したりする。

1.1 節で述べた路側システムにおけるエッジサーバの活用と同様に、ダイナミックマップにおいても、通信遅延やデータ量を削減したいタスクは、エッジサーバ上で実行されることとなる。

以上のように、エッジサーバでは、通信遅延削減やデータの集約など、多種多様なアプリケーションが実行される。このような環境においてアプリケーションの応答時間を一定に保つことは極めて難しくなる。応答時間を均一に保つことができなければ、自動運転を実現するうえで必要不可欠な要素（衝突回避・危険予測など）が実現できなくなる。

## 2. サービスの応答時間に対する分析

応答時間を均一に保つことがなぜ難しいのかということを説明する前に、まずサービスの応答時間について議論する。

### 2.1 サービスの応答時間定義

まず、「車両等の位置情報に関しては 100 ミリ秒程度の短い周期で収集・分析する」ということについての定義を与える。**図 3** は衝突回避をエッジサーバと各車との情報のやり取りで実現する例を示している。図中、A 車と B 車に衝突の危険性がある。A 車と B 車はエッジサーバに対して位置情報を送信し、エッジサーバは、互いの距離や速度から、衝突の危険性があるかどうかを判断して、各車両にブレーキ操作命令を送信するものとする。図では、説明を簡単化するために一方向の情報の流れのみを記載している。この中で、衝突回避に必要な時間は、各車が位置情報を送信し始めてから、各車がブレーキ操作を実行するまでの時間となる。各車が位置情報を送信し始めてから、各車がブレーキ操作を実行するまでの時間は、位置情報送信にかかる時間と、無線通信による時間の変動幅、情報収集機能実行時間、情報分析機能実行時間、ブレーキ命令送信時間、ブレーキ実行時間から構成される。この中で、位置情報送信と無線通信による変動幅は、アプリケーション安定実行制御基盤技術の研究対象外となる。つまり、情報収集・分析に必要な時間とは、情報収集機能実行時間と情報分析機能実行時間と、ブレーキ命令送信時間を合わせたものであり、これを一回当たり最大で 100 ミリ秒以内に収める必要がある。

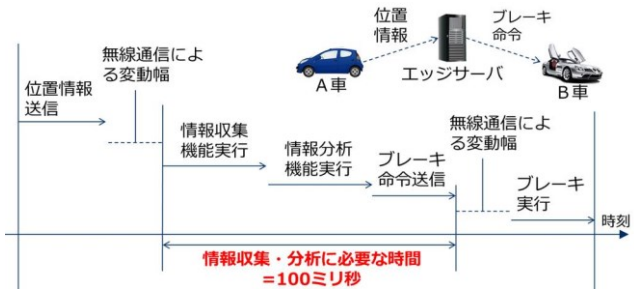


図 3 衝突回避の例

### 2.2 計算リソース量と応答時間の関係

アプリケーションの処理時間は、データ量（走行車両の

台数）と情報収集・分析アプリケーションに割り当てられるリソース量によって変化する。走行車両の台数は秒単位で急激に変化しないので、データ量はほぼ一定であると仮定して、ここでは計算リソース量と応答時間の関係について説明する。

**図 4** は、ある CPU 上で実行されるアプリケーションの実行時間を図示したものである。図中上部が情報収集・分析アプリケーションに対する理想的な CPU リソースの割り当てを表しており、図中下部が実際の CPU リソース割り当てを表している。理想的な CPU リソース割り当てでは、情報収集・分析アプリケーションに常に CPU 時間が割り当てられている。このような場合、アプリケーションの実行時間は、データ量に対してのみ増減することになる。一方、実際のリソース割り当てでは、情報収集・分析アプリケーション以外にも、他のアプリケーションや OS 機能、割り込み処理などが CPU を共有することになる。他のアプリケーションなどが実行されている間、情報収集・分析アプリケーションは実行できないので、結果として当該アプリケーションの実行時間が長くなる。情報収集・分析アプリケーションとその他の処理が実行される CPU を完全に分離すれば、情報収集・分析アプリケーションの実行時間がデータ量以外の原因で増減することはないが、エッジサーバは、その性質上 CPU リソースに制約があるので、このようなアプリケーション同士の CPU 共有は避けられないと考えている。

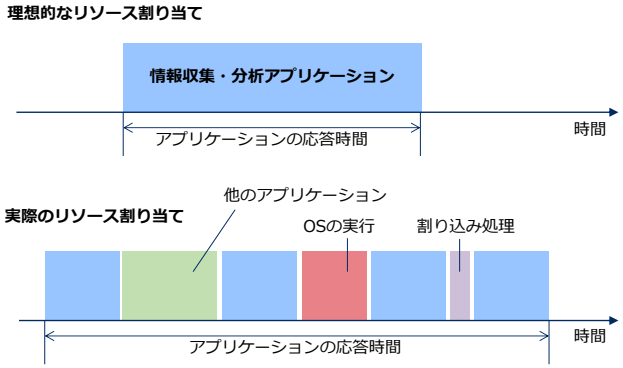


図 4 リソース制御の実際

### 2.3 サービスの応答時間変動を抑えるには

**図 5** は、「データ量を増減させる」か「計算リソース量を増減させる」かのどちらかの手法を適用した場合に、応答時間がどう変化するかをグラフで表したものである。アプリケーションの応答時間は、データ量に対して線形に増加するものとする。今、情報収集・分析アプリケーションの応答時間が目標となる値を上回っているとする（図中、測定されたデータ量が指す点）。データ量を減らす手法とは、具体的には、データをリソースに比較的余裕のある他のエッジサーバに転送したり、データの送信元に送信データ量の削減を依頼したりするといった方法が挙げられる。アプリケーションの計算リソース量を増減させる手法とは、つまり、情報収集・分析アプリケーションに対して割り当てられる CPU リソースを**図 4** で示した理想的な状態に近づける制御手法を指している。それぞれの手法に対する具体的な手段をまとめると以下のようになる。

- アプリケーション毎のデータ量を減らす手法
  - リクエスト負荷分散
    - 他のエッジサーバへのデータ転送
    - コンテナなどの仮想化技術を使ったエッジサーバ内リクエスト分散処理
  - 位置情報収集アプリケーションへのデータ削減通知（アプリケーション連携）
- アプリケーションの実行時間を減らす手法
  - タスクマイグレーションによるタスク再配置
  - CPU スケジューリング、タスク優先度設定などの CPU リソース制御

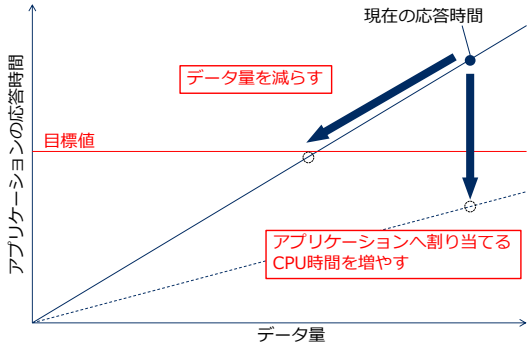


図 5 それぞれの応答時間削減方法に対する効果

## 3. 問題の定義

前節において、アプリケーションの応答時間を改善するには、「データ量を減らす」か「リソース割り当てを変更する」かのどちらかの方法があることを説明した。車や歩行者の量など、外部環境に依存するようなデータ量を減らすことはできないし、走行車両の台数は秒単位で急激に変化しないので、本論文では「リソース割り当てを変更する」ことに注目する。

本節では、サービスの応答時間変動を抑える方法として「計算リソースの割り当てを変更する」ことによって、応答時間を均一に保つことが、多種多様なサービスを実行するエッジサーバにおいて、なぜ難しい課題なのかを述べる。

### 3.1 計算リソース量と応答時間の関係が不明確である

エッジサーバを利用するユーザ（以下、サービス提供者）が求める SLA は、計算機リソースの使い方ではなく、あくまでもサービスを利用するユーザ（以下、サービス利用者）の体感品質が指標になっている。つまり、サービス提供者がエッジサーバの運用会社（以下、サービス運用者）に SLA を指定する場合は、CPU 使用率やメモリ使用量といったよ

ⓒ 2017 Information Processing Society of Japan

うな計算機リソースの絶対量を指定することはなく､「応答時間」といったようなサービス利用者の体感品質を希望することが多い。言い換えると、エッジサーバが配置されている環境がどのような構成になっているかはサービス提供者から見えないので、CPU 使用率やメモリ使用量といったような計算機リソースの絶対量を指定することが難しい。

一方で、タスクの優先度設定やリクエスト分散処理機能などの既存のリソース制御手法は、CPU 使用率やメモリ使用量などの計算機リソースの絶対量を調整するための仕組みである。

よって、計算機リソース量と応答時間の関係についての定量的な関数を定義できなければ、応答時間の改善は難しい。計算機リソース量と応答時間の関係についての定量的な関数は本来であればサービス提供者がサービス運用者に対して提供すべき情報であるが、サービス提供者は、他のアプリケーションの実行状況やエッジサーバの構成に関する情報をもっていないため、上記関数を正しく定義することが難しい。よってアプリケーションの応答時間が安定するよう安全側に倒して、必要以上の計算リソースを要求するが、通常のデータセンタとは異なり、エッジサーバが実行される環境は、計算機リソースに限りがあるので、サービス運用者は、必要以上の計算リソースを容認することはできない。

### 3.2 複数のアプリケーションが瞬間的に同じタイミングで動作することによって突発的な負荷上昇が発生

高い優先度を設定すべきアプリケーションが一つに決まっている場合は、当該アプリケーションを最優先で実行するのが最も単純な応答時間の安定化方法である。アプリケーションの優先度を高くするということは、割り当てられる CPU 時間が他のアプリケーションより多くなるということである。

多種多様のアプリケーションが動作するということは、一意に優先度を設定することが困難であることを意味している。このような場合、応答時間が劣化する場合には、場当たり的に当該アプリケーションの優先度を上げることで、応答時間の劣化を防ぐというような方法が妥当であると考えている。

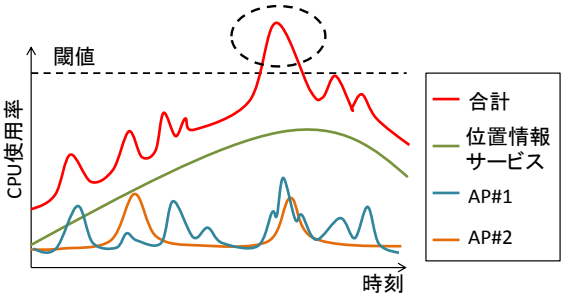


図 6 負荷の重ね合わせ

図 6 は各アプリケーションの負荷の重ね合わせが原因で閾値を超過する例を表している。基本的には、観測されたデータ量がアプリケーション実行時間の目標値を上回る場合は、以上に挙げた制御手法を用いて、アプリケーションの実行時間を減らすとよい。

しかしながら、我々は、車両の台数やセンサからのデータ量が時々刻々と変動し、また、複数のアプリケーションとリソースを共有する環境の中で、情報収集・分析アプリケーションの実行を安定化させるための決定的な制御手段は存在しないと考えている。これまで説明した制御手法は、測定されたデータ量が想定される範囲内に収まる場合にのみ有効である。測定されたデータ量に対して、上記制御手段の中から一つを選択し、当該制御手段実行中にデータ量が変化した場合は、変化したデータ量の差分に対して、さらに何らかの制御手段を選択するという段階的な制御手段の適用によってのみ、想定したデータ量を超えた場合にもシステムが破綻することなく制御を続けることが可能となる。よって、ある手段を実行したときのアプリケーション実行時間の削減量を適宜確認しながら、次の手段を選択していくというフィードバック制御が今回のアプリケーション安定実行制御にふさわしいと考えている。

## 4. アプリケーション安定実行制御基盤

多種多様なアプリケーションが混在する中においても、アプリケーションを安定的に動作させ、応答時間を一定に保つための基盤技術として、我々はアプリケーション安定実行制御基盤を提案する。

### 4.1 アプリケーション安定実行制御基盤概要

アプリケーション安定実行制御基盤は、エッジサーバ上で動作する情報処理サービスの応答時間を計測し、当該応答時間が目標値を上回る（閾値を超えて応答時間が劣化する）と予想される場合に、目標値を上回るまでの時間や程度に応じて、CPU の優先度変更、affinity 設定、CPU の占有割り当てなどのリソース制御手法を選択･実行する。図 7 にアプリケーション安定実行制御基盤の概要を示す。図 7 に示すとおり、アプリケーション安定実行制御基盤は、測定・分析・制御の３つのフェーズを繰り返し実行する。

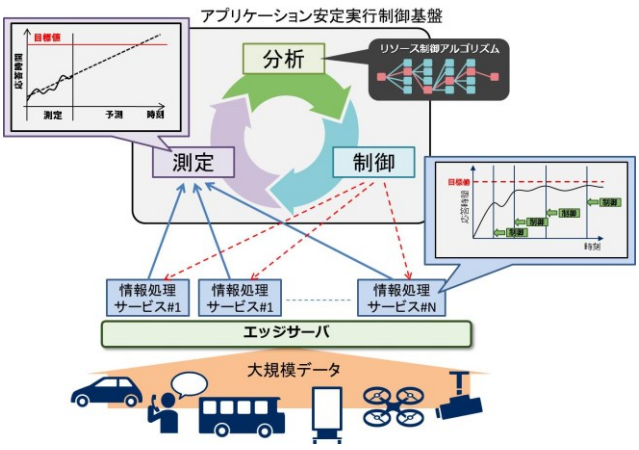


図 7 アプリケーション安定実行制御基盤概要

ⓒ 2017 Information Processing Society of Japan

測定は、エッジサーバ上で動作する各アプリケーションの応答時間と CPU 使用率を測定する。測定対象は、応答時間の測定対象となるアプリケーションと、CPU 使用率の測定対象となるアプリケーションの 2 種類が存在し、応答時間の測定対象となるアプリケーションは、「応答時間の定義」が可能であり、かつ、応答時間に関する SLA（Service Level Agreement）が設定されているものである。ここでの SLA とは、具体的な応答時間となる（例えば 100ms など）。応答時間の定義が可能であるというのは、「リクエストに対する返信」が一意に識別可能な状態を指す。言い換えると、リクエストを投げるだけで、返信がないようなアプリケーションは応答時間を定義することができない。例えば、センサから土壌の水分量を定期的にエッジサーバに送信し、エッジサーバでは一日の水分量の平均を計算してクラウド（もしくはユーザ）に送信するといったアプリケーションの場合は、リクエスト毎の返信に相当する手続きが存在しないので、このような場合は、応答時間を定義できない。応答時間の定義できないアプリケーションは、CPU 使用率を測定することになる。

分析は、測定フェーズで測定した応答時間と CPU 使用率から、応答時間の測定対象となっているアプリケーションの応答時間が、いつ閾値を超えて劣化するのかを予測し、閾値を超えないように適切なリソース制御手法とそのパラメータを計算する。

分析フェーズの予測は、「応答時間の劣化予測」と「CPU 使用率の負荷パターン分析」の 2 種類が存在する。

「応答時間の劣化分析」は、これまで測定された応答時間の数値を元に、将来的な応答時間の傾向を予測する。予測は、統計学のどのような手法を用いても良いが、本論文では、「線形単回帰分析」を用いている。線形単回帰分析とは、「線形」でかつ「単回帰」な分析方法を指す。「線形」とは、対象となる数値（ここではアプリケーションの応答時間）が Y=aX+b の線形グラフに従って推移すると仮定して、今まで取得した数値から当該線形グラフの傾きを計算することを表し、「単回帰」とは、予測する数値（Y=aX+b の Y に相当、目的変数と呼ぶ）が一つの測定数値（Y=aX+b の X に相当、説明変数と呼ぶ）によってのみ計算されるということを指している。本論文では、応答時間の計測対象となるアプリケーションは車両の位置情報の送受信なので、事故渋滞を除き、ある区域に存在する車両台数が、数秒で急激に増えるということがないと仮定し、応答時間の劣化分析に線形単回帰を選んだ。その他のアプリケーションで応答時間を測定し、応答時間の劣化を防ぎたい場合は、他の分析手法（非線形単回帰や、重回帰分析など）を利用する必要がある。

「CPU 使用率の負荷パターン分析」は、図 6 のような負荷の重ね合わせによって、突発的に応答時間が目標値を上回ることを避けるために、エッジサーバ上で動作するアプリケーションのすべての CPU 使用率を測定して、各アプリケーションの負荷パターンとその周期性を発見する。

負荷パターンとは、具体的には、あるアプリケーションが使用する時刻毎の CPU 使用率を関数に近似することを指している。「応答時間の劣化予測」でも説明した線形単回帰分析も一種の関数近似であるが、アプリケーションが使用する CPU 使用率が示す負荷パターンは多種多様であり、ほとんどの場合において、単純な線形単回帰により算出できない。よって、CPU 使用率が示す負荷パターンを識別する汎用的な機能を実現するには、統計学上の複雑な数式を扱うことになる。

しかしながら、アプリケーションの種類を限定することによって、負荷パターンの表現として、近似関数ではなくて、より単純な表現方法を用いることができると我々は考えている。1.1 節でエッジサーバの特徴について述べたように、エッジサーバで動作するアプリケーションは、「通信遅延を減少させたい」アプリケーションか、「通信帯域を減らしたい」アプリケーションの 2 種類に分類される。このうち「通信帯域を減少させたい」アプリケーションは、センサ情報の一時集約などのように「バッチ処理」のように動作するようなアプリケーションが想定できる。つまり、このようなアプリケーションが使用する CPU 使用率が示すパターンは、より単純で、ある瞬間に CPU を 100%（厳密には、その時に使用できる可能な限りの CPU 時間）使用して処理を実行し、処理が終了したら再び 0%に戻るというような、電子回路の矩形グラフのようなパターンを持つことが多い。このようなアプリケーションが示す負荷パターンは、「定常状態」と「実行状態」と「実行状態が継続している時間」の 3 つの特徴がわかれば、複雑な近似関数を計算しなくても、負荷パターンとしての特徴を十分に表すことができる。

このように、アプリケーション毎の負荷パターンと周期性を発見し、周期的な負荷パターンの重なりによって、突発的な負荷上昇がある場合は、その負荷上昇を避けるように、アプリケーションの実行タイミングをずらすよう、CPU スケジューリングを実施する。

応答時間の劣化予測の結果、対象となるアプリケーションの応答時間が目標値を超える場合、適切なリソース制御手法が選択される。リソース制御手法とは、単体計算機内で実行可能な CPU の割り当て制御手法のことを指しており、具体的には、プロセスの優先度変更や CPU アフィニティ制御、CPU の排他割り当てなどが挙げられる。ここでの「適切」とは、「目標値を超えるまでの時間で実施可能」であり、かつ「他のアプリケーションに可能な限り影響を及ぼさない」リソース制御手法とそのパラメータのセットを指している。単純に、あるアプリケーションの応答時間を改善するためには、当該アプリケーションに排他的に CPU を割り当てることが考えられる。そうすると、図 4 の理想

ⓒ 2017 Information Processing Society of Japan

的なリソース割り当てと同じように、すべての CPU 時間を一つのアプリケーションに割り当てることができ、応答時間が改善する。ただし、このような極端な方法を取ると、それまで当該 CPU を使用していた他のアプリケーションの応答時間が劣化してしまうため、可能な限り CPU を占有するというような方法は避けるべきであると我々は考えている。そこで、我々は、まず、CPU 時間をどれだけ使うと、どれだけ応答時間が改善されるのかという関係を明らかにして、それを元にアプリケーションの優先度を少しずつ変更することによって、他のアプリケーションの応答時間になるべく影響を及ぼさないような、リソース制御手法を取ることにした。

制御は、上記分析フェーズで実施した分析の結果算出されたリソース制御手法とそのパラメータを使ってリソース制御を実施する。

アプリケーション安定実行制御基盤は、これら測定・分析・制御を数十ミリ秒から数秒単位で何度も実施することによって、多種多様なアプリケーションが実行され、不確定要素が多いような計算機環境においても、対象となるアプリケーションの応答時間の劣化を未然に防ぐことが可能となる。

### 4.2 システムアーキテクチャ

ここでは、アプリケーション安定実行制御基盤の具体的なシステムアーキテクチャを説明する。図 8 に、システムアーキテクチャを示す。

今、応答時間の測定対象となっているのは、位置情報サービスであり、当該サービスの測定機能として、応答時間測定機能がある。また、測定した応答時間から未来の負荷を予測するために負荷予測機能があり、さらに、応答時間と CPU 使用率を変換するために応答時間 CPU 負荷変換機能が存在する。また、エッジサーバで実行される他のサービスは CPU 使用率測定機能によって各アプリケーションの CPU 使用率の推移が測定され、当該測定結果を元に、負荷パターン検出機能によって負荷パターンが検出される。

ここで注意すべきは、負荷パターンが検出されなかった場合である。負荷パターンが検出されない場合、現状は当該アプリケーションの負荷を無視せざるを得ないと考えている。つまり、負荷パターンをもって将来の突発的な負荷上昇を予測するという手法を実施することができないため、予想できないような CPU 負荷の上昇がみられる場合は、「場当たり的」に対処せざるを得ないと考えている。負荷パターンの検出は、周期性と規則性を元にしている。したがって、周期性や規則性のない CPU の使い方をするようなアプリケーションについては負荷パターンを検出することが難しい。アプリケーション負荷の周期性と規則性については 4.3.4 にて議論する。

測定対象となるアプリケーションの応答時間予測と、他のアプリケーションの CPU 使用率の負荷パターンを用いて、閾値超過判定機能は、ユーザが求める SLA を満たしているかどうか判定する。

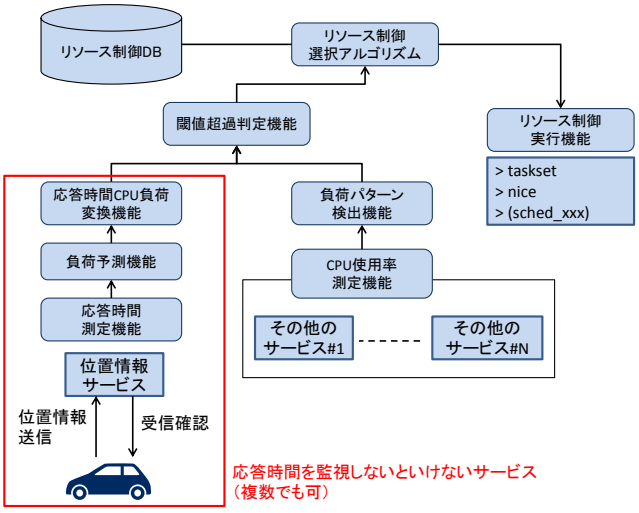


図 8 システムアーキテクチャ

ユーザの求める SLA を満たしていない場合は、リソース制御選択アルゴリズムが、SLA を満たすようなリソース制御とそのパラメータのセットを計算する。具体的には、SLA が応答時間である場合は、ある応答時間の閾値（ex.100ms）を超えそうな場合は、上述の優先度変更・CPU アフィニティ設定・CPU の排他割り当てなどから適切な方法を選択するとともに、当該リソース制御を実施する際のパラメータセットを計算する。優先度変更のパラメータセットなら、具体的な優先度を表す数値であり、CPU アフィニティであれば、アフィニティ設定を実施する CPU 番号である。特に、優先度設定において、優先度がいくつであれば、応答時間がどれくらい短くなるかといったような情報は、リソース制御 DB に事前に蓄えられているものとする。

### 4.3 各要素機能説明

本節では、アプリケーション安定実行制御基盤のシステムアーキテクチャを構成する各要素機能について詳細に説明する。

#### 4.3.1 応答時間測定機能

応答時間測定機能は、測定対象となるアプリケーションの応答時間を測定する。

#### 4.3.2 負荷予測機能

負荷予測機能は、上記応答時間測定機能によって測定された応答時間に基づいて、測定対象となるアプリケーションの将来の負荷を予測する。具体的には、測定した応答時間を説明変数とする線形単回帰分析により、測定値を近似する線形式を計算する。ここで簡単に線形単回帰分析を説明する。線形単回帰分析とは、一つの目的変数（ここでは応答時間）と一つの説明変数（ここでは、モニタリングによって得られた各データ系列）の関係を線形式で表現することを指す。具体的には、計測によって得られた目的変数と説明変数から線形式（Y=aX+b）の傾き(a)と切片(b)を導出する。導出した線形式は、目的変数の推定に利用される。

ⓒ 2017 Information Processing Society of Japan

すなわち、当該線形式の変数 X に説明変数の値を入力して、目的変数の推定値である Y を得る。このような回帰分析は、目的変数の値を直接計測できない場合に、説明変数の値から目的変数の値を推定したい場合に利用される。線形式の係数 a および b は、最小二乗法によって求める。最小二乗法は、求める近似式と実際の数値の差の 2 乗和を最小にするような近似式の係数決定に用いられる一般的な手法である。求める近似式と実際の数値の差の 2 乗和を最小にするには、近似式の各未知数(上記 a および b) について偏微分した式のそれぞれを 0 と置いて、それら偏微分した式の連立方程式を解くことによって求めることができる。最小二乗法は、線形式だけではなく、非線形式（曲線）や、説明変数が複数存在する場合(重回帰)の場合にも用いられる。

#### 4.3.3 応答時間 CPU 負荷変換機能

応答時間 CPU 負荷変換機能は、現在の応答時間とその際の CPU 使用率の変換表から、応答時間の測定対象となっているアプリケーションの応答時間と CPU 使用率を変換する応答時間とその際の CPU 使用率の変換表はリソース制御 DB に格納されており、変換表は、本システム実行前に事前に計測しておく必要がある。図 9 は、位置情報システムへのリクエスト数を増加させた時の応答時間と CPU 使用率の関係を表す散布図である。あるリクエストの処理に使う CPU 時間は、その CPU の性能に依存する。よって、このような CPU 使用率と応答時間の変換データを作るには、計算機ごとに応答時間と CPU 使用率に対する上記事前評価が必要となる。図 9 のグラフは、車両を模擬する位置情報クライアントプログラムから、図 8 に現れる位置情報サービス（以下、位置情報サーバプログラムと呼称）へリクエストを送信して、単位時間当たりのリクエストを 0.1 秒ごとに 30 ずつ増加させた際の、位置情報サーバプログラムの応答時間と CPU 使用率を計測して、同時刻の応答時間と CPU 使用率の組み合わせをプロットしている。

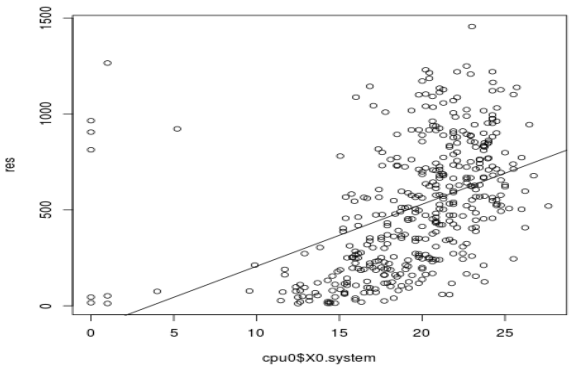


図 9 CPU 使用率と応答時間の関係

#### 4.3.4 負荷パターン検出機能

負荷パターン検出機能は、エッジサーバ上で動作する各アプリケーションの負荷パターンを検出する機能である。負荷パターンとは、アプリケーション（厳密にはプロセス）が利用する CPU 時間の時間的な推移に関する規則性と周期性を指す。規則性とは、一定の CPU 使用率を、一定の時間継続して利用している状態が、時刻経過の中に何度も現れる状態を指す。周期性とは、上記規則性が一定間隔で現れることを指す。図 10 は負荷パターンの例を表すグラフである。4.1 節でも少し述べたが、エッジサーバ上で動作するアプリケーションは、応答時間の計測対象となるアプリケーション以外には、センサ情報の一時集約処理など、いわゆる「バッチ処理」が定期的に動作しているものと想定している。このようなバッチ処理は、集めたデータを定期的に処理するため、CPU 使用率が図 10 のような矩形グラフのような形になると考えている。

CPU 使用率の時間的推移が矩形グラフのような形になる場合、「定常状態」と「実行状態」の 2 種類が区別できれば、当該アプリケーションの負荷パターンの規則性と周期性を表現することができると考えている。「定常状態」とは、データを収集しているだけで、それらのデータに対して特に処理を実施していない状態を指している。「実行状態」とは、集まったデータに対して何らかの処理を実行している状態である。

我々は図 10 の負荷パターンを検出するために、k 平均法を用いた。k 平均法は、統計データを分類（クラスタリングと呼ぶ）するための一般的な手法である。まず k 平均法を用いて、CPU 使用率の統計データを 2 種類のクラスタに分類して、それぞれのクラスタについて平均値を得る。これによって、定常状態と負荷発生状態を分類することができる。クラスタリングされた各データには「クラスタタグ」が割り当てられる。クラスタタグとは、各データがどのクラスタに属するかを識別するための識別子である。各データにクラスタタグが割り当てられているので、定常状態に割り当てられているクラスタタグが継続して出現しているデータ区間が図 10 の負荷発生間隔となり、負荷発生状態に割り当てられているクラスタタグが継続して出現しているデータ区間が負荷継続時間となる。

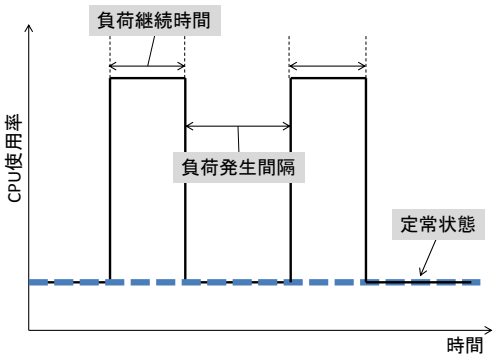


図 10 負荷パターン

#### 4.3.5 閾値超過判定機能

閾値超過判定機能は、応答時間を監視しなければならないアプリケーションの応答時間がユーザの所望する SLA を満たしているかどうかを判定する機能である。

ⓒ 2017 Information Processing Society of Japan

本稿における SLA は、ユーザの所望する「応答時間の上限値」として設定される。ここでのユーザとは本システムを利用してサービスを実施する実施者（サービス提供者）を指している。応答時間は長くなるほど、エンドユーザ（サービス利用者）に対するサービスの応答を返す時間が長くなる。よって、リアルタイム性を必要とするアプリケーションは、ある一定以上に応答時間が長くなることを防ぐ必要がある。本システムにおいては、ユーザの利便性を高めるため、ユーザから指定できる SLA の指標を応答時間の上限値とした。

閾値超過判定機能は、最初に応答時間 CPU 負荷変換機能を使って「負荷予測」と「SLA」を CPU 使用率に変換する。4.3.2 で説明した負荷予測の単位は「応答時間」である。一方で、4.3.4 の負荷パターン検出機能によって得られる負荷パターンの単位は「CPU 使用率」である。したがって、これらの単位を合わせないと、負荷予測と負荷パターンを加算して将来予測を実施することができない。閾値超過判定機能は、「負荷予測」の単位を「CPU 使用率」に変換することによって、負荷パターンとの CPU 使用率の加算を実施する。また、SLA の単位も応答時間であるので、こちらも CPU 使用率に変換する。

閾値超過判定機能は、CPU 使用率に変換した「負荷予測」と「SLA」に加えて、「負荷パターン」を重ね合わせた結果、現在時刻から将来的に負荷が閾値を超過するかどうか判定することができる。どの時点までを「将来」と呼ぶのかについては、例えば、現在時刻から 10 分後というような形で予め設定されているものとする。

#### 4.3.6 リソース制御選択アルゴリズム

リソース制御選択アルゴリズムは、上記 CPU 使用率の重ね合わせが閾値を超えないように、CPU 使用率の「調整」を実施するためのリソース制御手段とそのパラメータを決定するためのアルゴリズムである。

```
for resource_control in resource_controls:
    if (exec_time(resource_control) < limit_time:
        candidates.append (resource_control)
return sorted(candidates, key = lambda x: impact(x))[0]
```

リソース制御のために用いられる手段は複数存在する。具体的には 2.3 節で提示したような手法がある。これらのリソース制御手段に関して、実行時間と影響度を考慮して以下のアルゴリズムを実行する。影響度とは、あるアプリケーションの応答時間確保のために当該リソース制御を実行すると、他のアプリケーションの実行にどれだけの影響を及ぼすかということを数値化したものである。例えば、リソース制御手法の一つであるタスクの優先度変更を影響度 1 とした場合、タスクへの CPU の排他割り当ては、割り当てようとしている CPU 上で実行していたアプリケーションを他の CPU で実行しなければならないため、影響度は大きくなる。以下に、リソース制御選択アルゴリズムの疑似コードを示す。

疑似コード中 exec_time 関数によるリソース制御命令の実行時間は、予め取得されているものとする。また、疑似コード中 impact 関数による影響度についても、今のところ技術者の経験則に基づいてタスクの優先度変更を基準とした相対値が各リソース制御手段に割りてられている。影響度の計算方法について、尤もらしい関数の定義方法は、これからの研究課題となる。

利用されるリソース制御手段が決まったら当該リソース制御に与える入力パラメータを計算する。リソース制御手段に与えるパラメータは、各リソース制御手段に対して、入力パラメータを計算する関数を用意した。

## 5. 評価

本節では、我々の提案するアプリケーション安定実行制御基盤技術の評価について述べる。

### 5.1 評価環境概要

**我々は、自動運転を支える路側システムの安定動作を目標としている。**したがって、評価のためには、「車両が走行している状態」と「当該車両から位置情報がエッジサーバに送信されている状態」を実現し、さらに、位置情報を受信するサーバ上では「多種多様なアプリケーションが実行されている状態」を再現する必要がある。そして、その再現された状態において、位置情報を送受信する際の応答時間が、我々の提案するシステムによって安定するかどうかを確認する必要がある。我々は「車両が走行している状態」を模擬するために交通シミュレータを用い、シミュレーション結果となる時刻毎の車両の位置情報を使って「当該車両から位置情報がエッジサーバに送信されている状態」を模擬する位置情報発信プログラムを作成した。また、「多種多様なアプリケーションが実行されている状態」を模擬するために、負荷パターンの異なる 2 つの疑似アプリケーションを作成した。

図 11 に評価環境のシステム構成を示す。評価環境はアプリケーション安定制御実行基盤用計算機とエッジノード用計算機と疑似車載機用計算機から構成される。

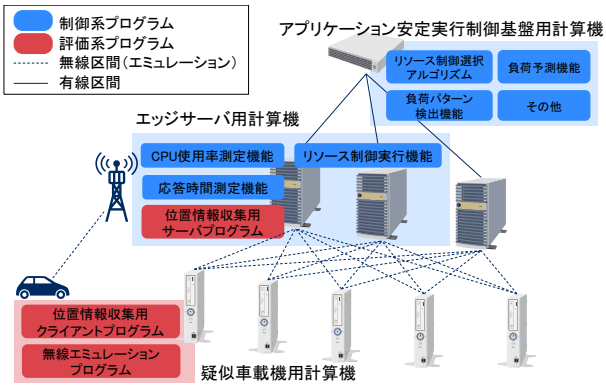


図 11 評価環境のシステム構成

疑似車載機用計算機とは、車両を模擬するプログラムが実行される計算機であり、当該計算機上では、位置情報収集用クライアントプログラムと無線エミュレーションプログラムが実行される。位置情報収集用クライアントプログラムは車両ごとに生成されるプロセスであり、自身の緯度・経度情報を図中のエッジノード用計算機上で動作する位置情報収集用サーバプログラムへ送信し、当該サーバプログラムから受信確認を受け取る。個々の車両の位置情報は前述のとおり交通シミュレータから取得する。図中、点線で書かれた無線区間は、評価環境では有線で接続されており、無線環境を模擬するために無線エミュレーションプログラムを使って通信遅延を図の点線で示す区間に付与している。今回、通信遅延の大きさは、単純にエッジサーバの位置と車両の位置を元に、距離が離れれば通信遅延が大きくなるように設計されている。本来なら、無線の電波状況は、車両と基地局間に存在する遮蔽物の種類や大きさ、他の電波によって大きく変化する。このような現実に近い無線状況を再現するためには、文献[20]や文献[21]などの無線エミュレーションプログラムを使用しなければより正確なエミュレーションは実現できない。これは今後の課題となる。

エッジノード用計算機とは、エッジサーバを表す計算機であり、ここでは、位置情報収集用サーバプログラムとアプリケーション安定実行制御基盤の応答時間測定機能とCPU 使用率測定機能とリソース制御実行機能が動作する。エッジノード用計算機では、その他に「多種多様なアプリケーション」を表現するために、異なる負荷パターンを持ついくつかのアプリケーションを実行している。応答時間測定機能は、位置情報収集用サーバプログラムの位置情報送受信処理について、その応答時間を測定して、アプリケーション安定実行制御基盤に送信する。また、CPU 使用率測定機能は、位置情報収集用サーバプログラム以外にエッジサーバで動作するアプリケーションの CPU 使用率を計測してアプリケーション安定実行制御基盤に送信する。リソース制御実行機能は、アプリケーション安定実行制御基盤で生成されたリソース制御命令を受信して、エッジサーバ用計算機で実行する。

アプリケーション安定実行制御基盤用計算機は、アプリケーション安定実行制御基盤を実行するための計算機であり、エッジサーバ用計算機で計測された CPU 使用率や応答時間を受け取って、ユーザの要求する SLA を満たすようアプリケーションが動作しているかどうかを監視する。また、SLA を満たしていないようであれば、リソース制御命令を生成して、エッジサーバ用計算機上で動作するリソース制御実行機能に送信する。

### 5.2 交通シミュレータ

位置情報収集用クライアントプログラムが自身の位置情報を送信するには、元になる車両の位置情報が必要になる。各車両の位置情報を生成するために、我々は、SUMO 交通シミュレータ[19] を使って、車両の時刻毎の位置情報を生成した。図 12 に交通シミュレータを使った車両位置情報の生成手順を示す。SUMO への入力は大まかに、1．道路情報、2．車両情報の２つから構成されている。道路情報は、自分で作るのではなく、フリーで公開されている道路情報である OpenStreetMap[18] から地理上のある区画の道路情報を SUMO 交通シミュレータにインポートした。車両情報は、SUMO に付属している車両の走行情報生成機能を使って車両の出現位置、走行ルートなどを生成している。登場させた車両台数は 200 台である。

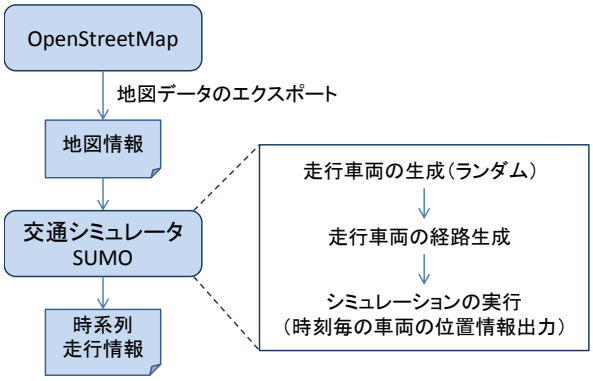


図 12 交通データの作成

図 13 に実際に生成した時系列走行情報の一部を示す。時系列情報は、時刻（time）、車両 ID（veh_id）、緯度情報（vertical）、経度情報（horizontal）から構成されている。緯度と経度情報は、実際のものとは異なり、シミュレータで使われている座標系となっている。よって、緯度と経度はそれぞれシミュレーション上の Y 座標と X 座標を表している。SUMO から出力される時系列走行情報（SUMO では dump ファイルと呼ばれている）は、XML 形式で出力される。図 13 に示す時系列情報は、位置情報送受信システムで利用しやすいように、SUMO の出力を CSV 形式に整形している。

```
time, veh_id, vertical, horizontal
0.0, 0, 2292.85003801, 2518.65957292
0.1, 0, 2292.85446794, 2518.6400697
0.2, 0, 2292.86554278, 2518.59131164
...
```

図 13 時系列走行情報

SUMO から得られた時系列走行情報を使って、車両に相当するプロセスを生成して、当該プロセスから位置情報収集サーバに対して、位置情報を送信することによって、実際には車両を使わないと再現できない評価環境をシミュレーション上で実現することができる。

### 5.3 交通シミュレーションの結果を使った位置情報送受信シミュレーションシステムの実現

図 14 に評価環境となるソフトウェアの構成図を示す。

ⓒ 2017 Information Processing Society of Japan

図には、評価環境として用いられるソフトウェアコンポーネントと、それらの関係、および実行手順が示されている。図に示す通り、評価環境となるソフトウェアは、評価環境の管理ソフトウェアとして、位置情報送受信プログラム管理サーバと疑似位置情報提供プログラムを用意した。これらのプログラムでは、5.2 節で作成した時系列走行情報を入力する。

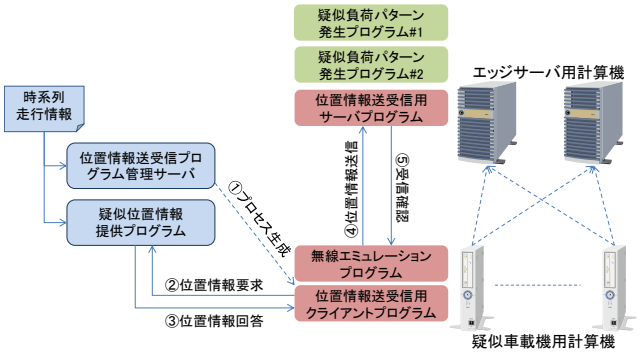


図 14 位置情報送受信シミュレータ

評価を始める前に、位置情報送受信用サーバプログラムと疑似負荷パターン発生プログラムがエッジサーバ用計算機で起動されているものとする。位置情報送受信用サーバプログラムはクライアントからの位置情報受信待ち状態であり、疑似負荷パターン発生プログラムは、評価を実施する管理者から疑似負荷パターンの発生開始命令の受信を待っている状態である。

次に、位置情報送受信プログラム管理サーバは、車両の時系列走行情報を元に、車両に相当する位置情報送受信用クライアントプログラムを各疑似車載機用計算機上で起動する（図中の①）。位置情報送受信用クライアントプログラムは、自身の位置情報を疑似位置情報提供プログラムに問い合わせて（図中の②）、回答を得る（図中の③）。問い合わせには「仮想時間」が用いられる。仮想時間とは、位置情報送受信シミュレーションの中で用いられる仮想的な時間であり、この仮想時間に同期して車両の位置や、負荷パターンが変化する。仮想時間は、位置情報送受信プログラム管理サーバに時系列走行情報が入力された時点が開始時刻となり、時系列走行情報の最後の行が終了時刻となる。仮想時間の単位は、任意で設定できるが、この評価環境では 1 秒としている（つまり、図 13 の 0.1 が実時間の 1 秒に相当する）。位置情報送受信用クライアントプログラムの生成（図中の①）では、この仮想時刻を引数として、クライアントプログラムが起動され、クライアントプログラムは、自身の位置情報を問い合わせる（図中の②）際に、この仮想時刻を疑似位置情報提供プログラムへ送信し、現在時刻（シミュレーション中の仮想時刻）の自身の位置情報を疑似位置情報提供プログラムから取得する。上記問い合わせの結果得た位置情報を位置情報送受信用サーバプログラムに送信し（図中の④）、受信確認をサーバプログラムから得る（図中の⑤）。

車両は、自分の位置から最も近いエッジサーバへ自身の位置情報を送信するものとする。これにより、車両の走行状況によって、エッジサーバへかかる位置情報の処理負荷が増減する。つまり、車両が少ない地域を担当するエッジサーバは、負荷が少なくなり、車両が多い地域のエッジサーバは負荷が高くなる。

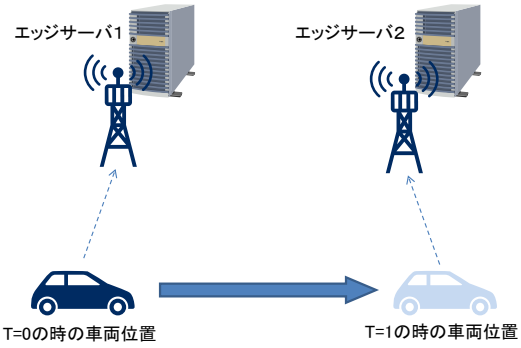


図 15 車両の位置と位置情報の送信先

図 15 は、車両の移動に伴って、車両の送信する位置情報を受け取るエッジサーバが変化することを表している。車両を表す位置情報送受信用クライアントプログラムは、予めすべてのエッジサーバの位置情報（図 13 の説明で例示したシミュレーション上の座標系 X 座標と Y 座標）を持っており、位置情報をサーバプログラムへ送信する前に、自身の位置と各エッジサーバの相対距離を計算して、最も近いエッジサーバへ自身の位置情報を送信する。

### 5.4 疑似アプリケーション群

疑似アプリケーションは、エッジサーバ用計算機で動作し、ユーザから指定された負荷パターンで、CPU 時間を消費するプログラムである。疑似アプリケーションは、負荷の大きさ、長さ、周期性を指定することによって、矩形グラフに似た CPU 使用率の負荷パターンを生成する。負荷の大きさとしては、CPU 使用率を指定し、長さは、0.1 秒単位で指定する。そして、周期性については、1 秒単位で負荷上昇のインターバル時間を指定する。疑似アプリケーションは、linux の stress コマンドによって CPU 負荷を生成して、CPU 負荷の上限値を cpulimit コマンドで設定することによって実現した。

### 5.5 評価結果

我々が提案するシステムの目的は、多種多様なアプリケーションが実行されるエッジサーバにおいて、特定のアプリケーションの応答時間を安定化させることである。したがって、評価としては、我々の提案するアプリケーションの安定実行制御技術がある場合とそうでない場合の比較が最低限必要となる。以下では、我々の提案する制御システムがある場合とない場合についての比較結果を述べる。

図 16 は、我々の提案する制御システムが無い場合の CPU 使用率の推移を表している。縦軸は CPU 使用率、横軸が時間経過を示している。グラフは、位置情報サービス

ⓒ 2017 Information Processing Society of Japan

の CPU 使用率、アプリケーション１（以下 AP#1）の CPU 使用率、アプリケーション２（以下 AP#2）の CPU 使用率の時間推移を表しており、さらにこれらの合計 CPU 使用率が記されている。図中、CPU 使用率 50%が閾値となっており、これを超えると応答時間がユーザの指定した SLA を満たせないものとする。負荷がある時刻に集中すると、CPU 使用率が閾値を超過していることがグラフからわかる。このような状態になると、位置情報サービスに十分な CPU 時間を割り当てることができなくなり、結果として応答時間がユーザの指定した SLA に違反して劣化することになる。

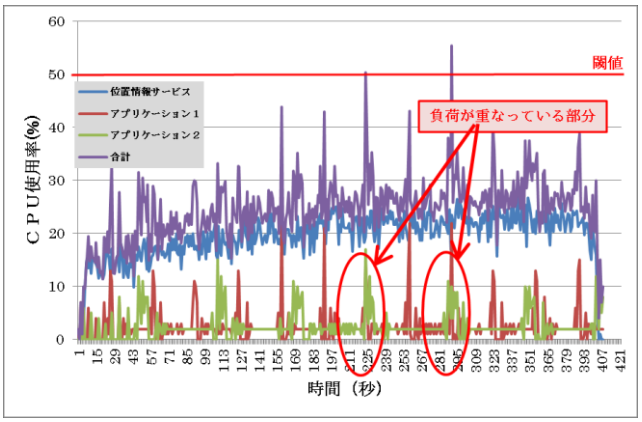


図 16 評価結果（安定制御なし）

図 17 は、我々の提案する制御システムを有効にした場合の CPU 使用率の推移を表している。グラフからもわかるとおり、AP#1 と AP#2 の負荷の集中を避けるように CPU 時間が割り当てられる。実際の制御に用いられたリソース制御は、cpulimit コマンドであり、AP#2 が実行されるタイミングで cpulimit コマンドを AP#1 に適用することによって、負荷が重なるような状態を避けるようにして各アプリケーションに CPU 時間を割り当てている。これによって、システムの CPU 使用率が閾値を超えることがなくなり、結果として、応答時間がユーザの指定した SLA に違反して劣化することがなくなった。

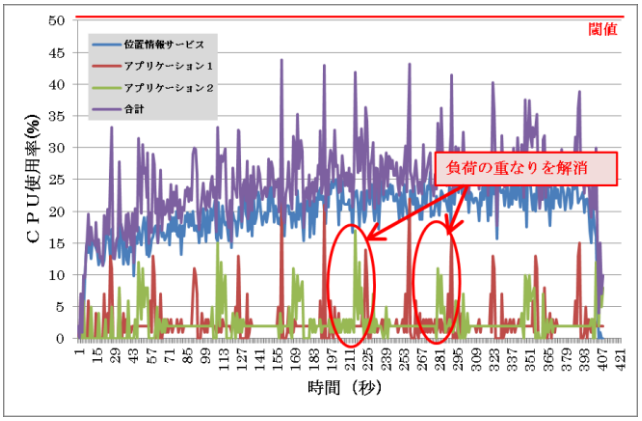


図 17 評価結果（安定制御あり）

以上の評価結果によって、我々の提案するアプリケーション安定実行制御基盤の有効性を確認することができた。

## 6. 考察と関連研究

### 6.1 今回の結果からなにがわかったのか？

今回の結果から、我々は、多種多様なアプリケーションが動作し、静的な優先度付けが困難な状況においては、負荷の測定と分析に基づいた動的なタスクの優先度設定が有効であると考えている。

### 6.2 関連研究

#### 6.2.1 負荷パターンの検出と応答時間の変動予測について

文献[5]から文献[10]は、CPU 使用率やネットワーク帯域使用量に関する将来予測に統計処理上のテクニックを応用している例である。

統計学には、データのパターンを認識するための様々な手法が存在する[1]が、我々は、任意の負荷パターンを検出する統計学上のテクニックは実質的に存在しないと考えている。今回、負荷パターンの検出に用いた k 平均法は、予めクラスタ数がわかっていないとデータを分類することができない。今回は単純な矩形グラフの検出であったため、2 種類のクラスタが存在すると断定したが、実際の負荷パターン検出では、数種類の矩形パターンが出現する場合や、そもそも矩形グラフでない場合も考えうる。矩形グラフのようにデータをクラスタに分類できる場合は、k 平均法のようなクラスタリング技術が有効であるが、任意の負荷パターンを検出するのであれば、クラスタリングがうまくいかなくなり、単純な回帰分析（データ全体に対して、ある関数を当てはめる分析手法）から、基底関数展開法（データを一定区間に区切って、各区間に近似関数を当てはめて、それらを線形結合する）まで様々な手法を、アプリケーション１つ１つに対して繰り返し試さなくてはならなくなり、負荷パターンの検出が実時間上不可能になってしまう。

一方、本研究において「応答時間の変動分析」と「負荷パターンの検出」という 2 つのカテゴリに分けて分析を実行した理由は、2 つのカテゴリで利用する統計学上の手法をわかりやすく分けることができたからである。「応答時間の変動分析」は、ある携帯電話基地局がカバーするエリアに侵入した車両とエッジサーバ間でやり取りされる位置情報の応答時間について、その変動を分析する。ある携帯電話基地局エリアに出入りする車両の数は、数秒や数分単位で見た場合、急激に変動することはない。したがって、数秒や数分単位の応答時間の変動を推定するのであれば、基本的には応答時間が線形式で近似できると考えた。また、「負荷パターンの検出」では、バッチ処理など、ある時刻になったら一定時間処理を実行するようなアプリケーションを仮定した。これによって、負荷が上昇している時間帯とそうでない時間帯がはっきりと分かれると仮定できたので、単純な k 平均法が有効であると考えた。

したがって、応答時間の変動予測や負荷パターンの検出には、技術者が経験則に基づいてアプリケーションの特性

ⓒ 2017 Information Processing Society of Japan

に応じた統計処理手法を最初に選択（仮定）する必要があると我々は考えている。

**6.2.2 リソース制御選択アルゴリズムについて（フィードバック制御に基づいた計算機リソース制御）**

文献[11]から文献[16]は、CPU 使用率やネットワーク帯域の制御に制御工学分野のフィードバック制御を応用している例である。

制御工学上のフィードバック制御は、現在と過去の数値情報を元にして制御パラメータを算出している。ここに上げた文献も現在の数値情報と過去の数値情報を元にしているものがほとんどであり、6.2.1 で上げたような統計処理を基礎とした負荷の将来予測を現在・過去に次ぐ第 3 の入力としているものはない。

これらの手法が負荷の将来予測を使わない理由は、フィードバック制御を基本とする方式が、極めて短時間（ミリ秒）でのリソース制御を実施しているためである。短時間で制御を繰り返すことによって、将来予測をすれば検出できたであろう過去の制御の誤りを速やかに修正することが可能となる。今回の負荷パターンである矩形グラフを負荷の例として挙げると、矩形の始まり（負荷の発生開始時刻）を捉えることができなかったとしても、文献[11]から文献[16]の方式を使えば、数十～数百ミリ秒後には、負荷に応じた適切な制御パラメータを算出可能である。

しかしながら、時間のかかるようなリソース制御を実施するのであれば、将来予測は欠かせなくなる。文献に上げたフィードバックシステムは、リソースの制御手法として、タスクの優先度変更など、ミリ秒単位で実施できる制御手法のみ用いている。我々の提案手法は、将来的にはタスクの移動（VM マイグレーション）や動的な負荷分散システムの構築など、実行に長い時間のかかるリソース制御の実施を検討しているため、数分から数十分単位での長時間の将来予測が必要不可欠となる。

**6.2.3 現実性の高い交通環境のシミュレーション**

文献[22]から文献[26]は、交通環境シミュレーションとネットワークシミュレーションに関する文献である。これらの文献を含み、一般的に、交通環境シミュレーションとネットワークシミュレーションの融合は、主に車両を用いたアドホックネットワークの形成についてシミュレーションを目的している。一方、我々が使用した評価環境は、携帯電話網への接続を模擬することを目的としているので、シミュレーションの構成要素が異なる。車両を用いたアドホックネットワークは実現性に乏しく、実際には、我々の用いたような評価環境が必要であるが、今のところ既存研究はない。

## 7. まとめ

本稿では、静的な優先度付けが難しいような環境において、制御工学のフィードバック制御に基づく、応答時間の測定・分析・制御をミリ秒単位で実行するアプリケーション安定実行制御基盤を提案した。アプリケーション安定実行制御基盤は、測定対象となるアプリケーションの応答時間を測定し、当該測定値が応答時間の目標値を超過するような場合は、優先度を上げるなど適切なリソース制御とそのパラメータを計算・実行することによって、静的な優先度付けが難しいようなエッジコンピューティング環境においても、アプリケーションの応答時間を均一に保つことが可能となる。交通シミュレーションによってエッジサーバにかかる負荷を再現し、本提案手法を実施した結果、従来と比較して、CPU 使用率の極端な変動を抑え、結果として応答時間が安定することを確認した。

**謝辞** 本研究は、総務省の委託研究「自律型モビリティシステム（自動走行技術、自動制御技術等）の開発・実証（Ⅱ.自律型モビリティシステムの高精度化に係る技術の確立）」の成果である。

## 8. 参考文献

[1] Suda, Y., & Aoki, K. (2015). Current activities and some issues on the development of automated driving. Journal of Information Processing and Management, 57(11), 809–817. https://doi.org/10.1241/johokanri.57.809

[2] 総務省. (2015). 自律型モビリティシステム （ 自動走行技術 、自動制御技術等 ） の開発・実証 基本計画書, 1-13.

[3] 名古屋大学大学院情報科学研究科附属組込みシステム研究センター. (2016). ダイナミックマップ 2.0・コンソーシアム. Retrieved from https://www.nces.is.nagoya-u.ac.jp/

[4] C.M.ビショップら; パターン認識と機械学習（上巻）（下巻）, 丸善出版, 2012

[5] Dinda, P. A. P., O'Hallaron, D., & Hallaron, D. R. O. (2000). Host load prediction using linear models. Cluster Computing, 3, 265–280. https://doi.org/10.1023/A:1019048724544

[6] Yang, L., Foster, I., & Schopf, J. M. (2003). Homeostatic and tendency-based CPU load predictions. Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003, 0(C). https://doi.org/10.1109/IPDPS.2003.1213129

[7] Akioka, S., & Muraoka, Y. (2004). Extended forecast of CPU and network load on computational grid. Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid, 765–772. https://doi.org/10.1109/CCGrid.2004.1336711

[8] Zhang, Y., Sun, W., & Inoguchi, Y. (2007). CPU load predictions on the computational grid. IEICE Transactions on Information and Systems, E90–D(1), 40–47. https://doi.org/10.1093/ietisy/e90-1.1.40

[9] Di, S., Kondo, D., & Cirne, W. (2012). Hostload prediction in a Google compute cloud with a Bayesian model. Retrieved from http://research.google.com/pubs/pub42553.html

[10] Andreolini, M., & Casolari, S. (2006). Load prediction models in web-based systems. 1st International Conference on Performance Evaluation Methodologies and Tools. https://doi.org/10.1145/1190095.1190129

[11] Stankovic, J. A., He, T., Abdelzaher, T., Marley, M., Tao, G., Son, S., & Lu, C. (2001). Feedback control scheduling in distributed real-time systems. Proceedings 22nd IEEE RealTime Systems Symposium RTSS 2001 Cat No01PR1420, 59–70. https://doi.org/10.1109/REAL.2001.990596

[12] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., &

ⓒ 2017 Information Processing Society of Japan

Doyle, R. P. (2001). Managing Energy and Server Resources in Hosting Centers.
[13] Stankovic, J. A., Abdelzaher, T. F., Lu, C., Sha, L., & Hou, J. C. (2003). Real-time communication and coordination in embedded sensor networks. Proceedings of the IEEE, 91(7), 1002–1022. https://doi.org/10.1109/JPROC.2003.814620
[14] Abdelzaher, T. F., & Stankovic, J. a. (2003). Feedback performance control in software services - Using a control-theoretic approach to achieve quality of service guarantees. IEEE Control Systems Magazine, 23(3), 74–90. https://doi.org/10.1109/MCS.2003.1200252
[15] Lu, C. L. C., Wang, X. W. X., & Koutsoukos, X. (2005). End-to-end utilization control in distributed real-time systems. IEEE Transactions on Parallel and Distributed Systems, 16(6), 550–561. https://doi.org/10.1109/TPDS.2005.73
[16] Wang, X., Jia, D., Lu, C., & Koutsoukos, X. (2007). DEUCON: Decentralized end-to-end utilization control for distributed real-time systems. IEEE Transactions on Parallel and Distributed Systems, 18(7), 996–1009. https://doi.org/10.1109/TPDS.2007.1051
[17] Jared P. Lander, et. al; みんなの R, マイナビ出版, 2015
[18] OpenStreetMap, https://www.openstreetmap.org
[19] SUMO 交通シミュレータ, http://www.dir.de
[20] NetSim Network Simulator & Emulator, http://tetcos.com
[21] Razvan Beuran, et. al. (2008). A Multi-purpose Wireless Network Emulator: QOMET, 22nd International Conference on Advanced Information Networking and Applications
[22] Khairnar, V. D., & Pradhan, S. N. (2011). Mobility models for Vehicular Ad-hoc Network simulation. In *2011 IEEE Symposium on Computers & Informatics* (pp. 460–465). IEEE. https://doi.org/10.1109/ISCI.2011.5958959
[23] Krajzewicz, D., Erdmann, J., Behrisch, M., & Bieker, L. (2012). Recent Development and Applications of {SUMO - Simulation of Urban MObility}. *International Journal On Advances in Systems and Measurements*, *5*(3), 128–138. Retrieved from http://www.iariajournals.org/systems_and_measurements/
[24] Piórkowski, M., Raya, M., Lugo, A. L., Papadimitratos, P., Grossglauser, M., & Hubaux, J.-P. (2008). TraNS. *ACM SIGMOBILE Mobile Computing and Communications Review*, *12*(1), 31. https://doi.org/10.1145/1374512.1374522
[25] Sommer, C., German, R., & Dressler, F. (2011). Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*, *10*(1), 3–15. https://doi.org/10.1109/TMC.2010.133
[26] Bellomo, N., & Dogbe, C. (2011). On the Modeling of Traffic and Crowds: A Survey of Models, Speculations, and Perspectives. SIAM Review, 53(3), 409–463. https://doi.org/10.1137/090746677

ⓒ 2017 Information Processing Society of Japan