IoT機器における熱情報とCPU状態を用いた閾値とプロセスの動的制御

多川 悠太[1,a)]　串田 高幸[1]

概要：近年，Alexa や Amazon Echo，AppleWatch を始めとする Internet of Things(以下 IoT と表記する) 関連機器が注目を集めてきている．一昔前まではなかった外出先から起動出来るエアコン，blutooth のイヤホン．人々の暮らしを豊かにする IoT．その IoT では，長時間利用における機器全体の発熱，CPU 本体の発熱や熱によるクロックダウン (アンダークロック), システムダウンに関する問題があり，物理的な観点では空冷を行う方法が存在する．今回の論文では IoT 機器である RaspberryPi に着目し，RaspberryPi を使用している利用者に向けて RaspberryPi の温度と動作 (クロック) 周波数の関係を利用した閾値を利用したプロセス制御を行うことで温度を気にすることなく長時間利用が可能となる提案を行う．また，今回は温度を変動させる事でクロック周波数がどの様に変化するのかを検証した．結果として，70 ℃を超えた時にクロックダウンが起こり 75 ℃を超えた時にはシステムダウンが起こり処理効率に関しても温度が低い方が優れていることが分かった．

1. はじめに

1.1 背景

今日，人々の生活を豊かにしている IoT 機器．IoT とはモノのインターネットのことで，田んぼの水位の測定，観測を自動化することやトイレの空き状態をスマートフォンで確認することが可能になっている．Krintz らは，プロセッサ (以下，単に CPU と呼ぶ) と周囲の大気温度との関係性を調べ，温度測定器として農業現場で利用することを目的としている [1]．温度測定器として RaspberryPi を利用し，リアルタイムの温度を測定し一週間のデータを取ったり、大気温度と RaspberryPi の CPU 温度を比較することで，局所的な水分の蒸発散量を予測することを可能にしている．

そんな中，学生にとって身近な勉強しやすい IoT 機器はシングルボードコンピュータでもある RaspberryPi である．RaspberryPi とは温度センサーや体感センサー，ビーコンセンサーを使って測定，観測を行い，データを分析する IoT 機器の一種である．だが RaspberryPi は測定や観測を行うという性質上，一週間や一ヶ月，長時間稼働し続ける必要があることが多い．当然稼働している間は CPU の熱が発生し，埃や水から RaspberryPi を守るために保護ケースに入っている場合が多いため，長時間の稼働によりその熱が籠ってしまう．そこで導入されている機能としてクロックダウンという機能が RaspberryPi には存在している．クロックダウンとは，CPU の動作 (クロック) 周波数を強制的に下げ，プロセスの処理を行っているかどうかに関わらず CPU を所有している機器が自らの判断で処理性能を落とすことで動作により発生する電力の消費や発熱を低減することが出来る機能である．また，RaspberryPi にはクロックダウンの機能があるバージョンとクロックダウンの機能が無いバージョンがある．そのため，機能が無いバージョンを利用している利用者の中には急に RaspberryPi の電源が落ちてしまった，強制終了になった後に起動できなくなってしまったという報告が存在している．また，クロックダウンが機能として存在している RaspberryPi はどの様な基準でクロックダウンが行われているのか，またクロックダウンが解除されるのはいつなのかという問題も存在している．クロックダウンの他にも，サーマルスロットリングと呼ばれる CPU の温度が上昇しすぎた場合にクロック周波数を落として温度を下げる機能が存在しているが，プロセスの処理が渋滞している時には殆ど機能していな事が後の検証で判明している．更に，この様なことが度々起こるようでは RaspberryPi は状態を確認しに付きっきりで利用者が監視する必要がある．今回のレポートでは，その様なことがないように熱に関する項目を活用し監視者がその場にいなくとも RaspberryPi を使用することが出来るような制御システムを提案する．

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[a)] C0117191

従来の IoT 機器を使用したサービスが図 1 である．この図では畑や田んぼといった水田やビニールハウスで利用されているシステムで，RaspberryPi を要所要所に設置し 24 時間稼働する事で，その箇所の温度や水面の高さ，室温が自動的に測定されサービスを提供している企業の利用しているクラウドにデータが送信され保存される．ユーザーはスマートフォンや PC でそのデータを閲覧することで農作物が良い状態になるように最適化を行える．これにより台風や水路の水不足の予測や対策が立てられるようになるといったサービスである．尚，矢印に含まれている番号は実行手順である．

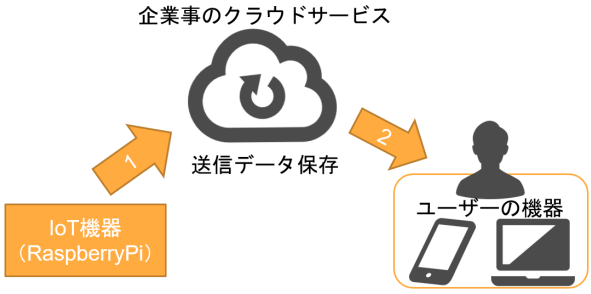

図 1　従来のサービス

今回の提案が従来の方法と異なる点として図 2 を挙げる．図では温度情報を取得するという点とそのデータをクラウドに送信可能になっている点は変わらず，RaspberryPi 自身が 24 時間稼働しても熱によるシステムダウンが起きないようにプロセスを制御する．前述したとおり RaspberryPi がシステムダウンしてしまうと原因以外のプロセスの活動も止まってしまい，作業全体がストップしてしまうことになってしまうため今回の提案を行う．

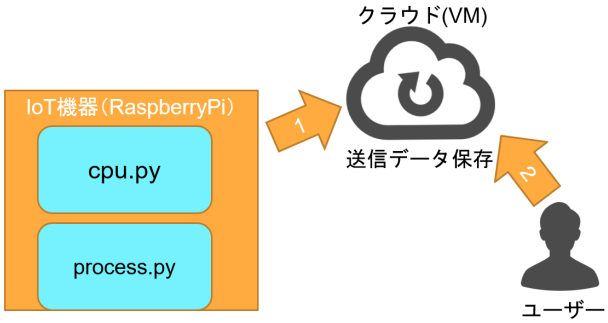

図 2　今回の提案

1.2　課題

今回のレポートでは，IoT 機器を長時間使用している際，CPU から発せられる熱によるシステム全体のダウンで作業全体が中断されてしまうことを課題としている．作業全体が中断されてしまうと，放熱され，システムが復旧できるレベルになるまでダウンする前までにやっていた作業が出来なくなってしまう．そのため，このレポートでは熱に関したシステムダウンに関する提案を行う．

1.3　各章の概要

本論文では，2 章で IoT 機器についての既存技術について紹介を行い，3 章以降で今回の熱情報と CPU 状態を用いた閾値とそれを利用したプロセスの動的制御を行うシステムについて報告する．

2.　関連研究

課題に関連する研究を記述する．RaspberryPi の温度についての対策や，負荷によって引き起こされるプロセスに対する研究が存在している．

Brian らは RaspberryPi の CPU チップが 85 %の過熱状態に達成した時にクロックダウンしてしまうがオーバークロックを使用しない通常の設定であれば発熱を心配する必要がほぼない可能性を挙げている [2]．実験では，オーバークロック状況下で RaspberryPi を使用するという前提条件で RaspberryPi のケースがある場合と無い場合の比較を二分に一回温度センサーで計測することにより行っている．ケースなしでの過熱試験では，アイドル状態での平均温度は 45.5 度で稼働後 55 度に上がったことを挙げている．ケースありの実験もアイドル状態から調べていて，56 度であった．ケースを用いた状態での過熱試験では，最高温度が 68 度，その他は 66 68 度の間で推移していた．この様にケース付きで稼働させている RaspberryPi の温度上昇値はベンチマークのみの状態でも高く，この論文の実験気温である 23 度を上回る環境下であれば非常に高くなってしまうことが懸念される．

Zhengguo Sheng，Chinmaya Mahapatra らは産業用の無線センサーネットワーク，IoT について効率的な管理について述べている [3]．小型のワイヤレスセンサデバイスはバッテリー電力，処理能力と貯蔵能力，無線接続範囲と信頼性が制限されるという特徴があり，パフォーマンスの監視やセンサーノードへのコマンドの送信を人の手を介さずに時間の遅れがないようにリアルタイムでリモート管理するための通信のプロトコルを設計しなければならないことを挙げている．

また文中では IoT を使用した環境を指すスマートシティ，スマートホームについて述べられている．スマートホームは家電製品を始めとし，家の中に存在するデバイスが接続し合うことで，利便性の高い生活を提供する環境のことで，スマートシティは IoT を活用して，基礎インフラと生活インフラ，サービスを効率的に管理して経済発展目指す都市である．

IoT システムに存在している幅広いインテリジェント (データ処理を持つ機器) で小型のセンシングデバイスが存在しているが，すべて共通のアーキテクチャとネットワー

ク要素で共有しており，一般的な特徴と課題として 6 つに要約している．

(1) リモートデバイスと通信監理を介して信頼出来る通信を確保する必要，つまりセキュリティ面での問題がある．
(2) 現在の互いに異なるプラットフォームを使うのではなく，単一のプラットフォームとオープンプラットフォームのアプリケーションプログラミングインターフェース (以後 API と記す) でサービス機能を統合することでカスタマイズ性を上げることが出来るため重要度が高い．
(3) IoT ネットワークへの大規模な接続ためにアドレスリソース不足やアクセスの輻輳などの問題に対処する必要がある．
(4) 通信速度で遅延や再送が発生してしまい余計な負荷がかからないよう，QoS 要件を保証するために，異種アクセスの命名とアドレス指定に対処する必要がある．
(5) IoT サービスを実現できるようにするために，異種アクセスのネーミングやアドレッシングに対応する必要がある．
(6)　大規模な情報の蓄積，共有，マイニングに対応しなければならない．

そのため，今回は (6) に対しての対策としてサーバを利用することを提案で後述する．

Chandra，Krintz らは，この論文では IoT を農業に活かしている IoT を農業に活用するメリットとして，日照り対策，防霜，凍結対策を挙げ，例えばアメリカでは，霜害による損失は他のどの気象関連の現象よりも大きいとしている [1]．また大規模な防霜の対策には水の散布，風力発電機の使用，ヒーターの組み合わせから選ばれるか，その全てが必要になってくる．

従来のこの方法であるとかなりの手間がかかり，霜害が発生するタイミングを誤ってしまうと対策分費用が増え生産者にとってはコストパフォーマンスが悪くなってしまう．その上， どの作業においてもリアルタイムで正確に温度を測定し，予測することが求められ農業の土地環境の差異の要素である，地形の違い，周囲の構造物，地表面の覆い，植物の成熟度，近隣の水域が挙げられる。そして，膨大な数のパターンが存在する微気候を踏まえて的確に温度を測定することは，高コストで手間のかかることである．そこで IoT ベンダーが提供しているサービスを利用することで，データ抽出や高度な分析が可能となる．しかし，これらのサービスは高価であり，データをクラウドに送信する必要と定期的な分析のための定期料金を課しており，その結果として IoT の農業への導入は進んでいない．そこで，この論文では，オープンソースのクラウドウェアを使用しエッジクラウドを設計することで自己管理を可能としている．また微気候の温度を正確にリアルタイムに推定するために比較的安価である RaspberryPi を使用し予測を均一ではなく，局所的な温度の違いをリアルタイムで正確に低コストで予測することが出来ると主張している．

Tetsuya Oda らは，ノードの CPU 動作周波数と CPU 温度に関する実験をしている [4]．実験では，CPU の動作周波数 100MHz，700MHz の二つについて比較していて，平均アイドル時間が 58.120 %，88.820 %で，CPU 温度の平均値は 49.107 度，49.487 度という結果になっている．平均スループットに関してもベンチマークを使用することで検証していて，CPU 動作周波数 100MHz，700MHz の条件下で 505.8Kbps，511.2Kbps とどちらの実験，検証でも動作周波数 (またはクロック周波数) が高い方が CPU 温度上がりやすくなり，良い処理結果が得られることが分かっている．ただし，Raspberrypi の場合，動作周波数が高くなり過ぎると比例して CPU 温度も上昇し強制的にクロックダウンしてしまう．そのため，処理が重くなっているプロセスを突き止め，プロセスの停止と再会の制御を行う必要がある．

Raid Ayoub らは，CPU ソケットとプロセッサ (CPU) が引き起こすホットスポットに焦点を当て，温度が上昇することで性能が低下するということを挙げている [5]．そのための冷却コストを大幅に削除するためのアプローチを提案していて、ランタイムワークロード，即ち実行時の CPU 使用率の特性評価を取り入れ、効率的な熱管理スケジューリングを実現している．この方法ではファンの速度を管理することで、最新の技術と比較して平均 80 %の冷却エネルギーの節約が可能であるとしている．

Young Geun Kim らは電力消費による IoT の熱による問題を取り挙げた [6]．この熱問題を解決するために熱を考慮した DVFS 方式 (プロセッサの動作周波数・電圧を実行時の負荷状 況などをもとにして動的に制御し、省電力化を行う技術の総称) を利用し，オンチップ温度と CPU 使用率を監視，オンチップ温度が所定の温度閾値以下になると，CPU 使用率に基づいて CPU の動作周波数を決定する．使用率が 80 %を超えると最大動作周波数まで周波数をスケーリングする．ただし，DVFS 方式が比較的効果的なのは，プロセッサの使用率がある程度低い場合である．

Atis Elsts らは TSCH ネットワークにおける時刻同期の精度には温度が干渉してくることを挙げ，干渉によっては同期の誤差が出てしまうことを述べ，軽量な IoT のための適応的な温度耐性を持つ時刻同期方式を提案している [7].

Odiowei らはプロセスモニタリングを提案した．提案した SSICA のモニタリング性能を評価し、本研究で検討した CVA や DICA 技術と比較し，提案手法と CVA や DICA 手法との比較のために信頼性と遅延のパフォーマンスを評価するベンチマークを使用している [8].

Michael らは大規模なコンピューティング・クラストを構築する際の消費電力は重要な問題である．その消費電力

を削減，低減する方法は低消費電力の組み込みプロセッサを使用 [9]．使用するクラスタは 25 台の RaspberryPi Model 2B システムを 100MBEthernet で接続したものであり，RaspberryPiZero や 2B を始め，多数の RaspberryPi の、速度、コスト、プロセッサの種類を比較した．1Hz と 100Hz のサンプリング周波数をベンチマークで比較し電力に関わるそれぞれの要素についての検証を行っていて，RaspberryPi の種類による電力消費や処理速度の差異を注目した．

M. Sridharan らは二重管熱交換器のための IoT ベースの監視・制御システムを提案した [10]．熱媒流体の入口から出口までの温度変化を測定するために温度センサを使用し，二つの質量流量センサを用いて、高温流体と低温流体の入口から出口までの流量変化を測定している．温度観測から結果のクラウド送信に至るまで RaspberryPi を介していて，温度測定とデータ保存・閲覧にクラウドを利用する手段は．私が今回提案する手法と似ているため，参考とした．

3. 提案

このレポートでは機器が壊れてしまう原因である熱に関して注目する．

題では IoT 機器としているが今回は RaspberryPi として考える．理由として RaspberryPi には CPU が熱くなり過ぎると機器を守るために CPU の性能・稼働率を下げるクロックダウンが行われる．また，クロックダウンが行われなかった場合，システムダウンに至ってしまう．加えて，クロックダウンやシステムダウンしてしまうと稼働中であるシステム，プロセス全体が通常以下の処理速度になるか，処理自体が止まってしまう．そのため，今回の提案では RaspberryPi に熱がこもりすぎてしまった場合に原因となっているだろうシステム，プロセスを特定し，そのプロセスの優先度を一時的に下げることで閾値を越えた温度を下げる．温度が閾値を下回ったならば，プロセスの優先度を元に戻す．このサイクルを行い，事前に温度を調節し下げることで，利用者が予測出来ないタイミングで急なクロックダウン・システムダウンが行われて作業が中断されることがなくクロック周波数が高い水準で保たれたまま作業が継続出来るようになる．また，温度がある程度高くなってきてから起動するクロックダウンは，長期的な処理性能を引き起こしてしまい，行っている作業や環境によっては排熱が上手くいかずシステムダウンが起きてしまう可能性がある．この提案での閾値とは，RaspberryPi の上限設定近くのクロック周波数を継続的に出せている時の温度の値を指している．RaspberryPi が CPU から取得した時間毎のクロック周波数と温度情報は，サーバに向けてデータを送信することで本体の microSD に保存するよりも膨大な量，月単位での保存・閲覧が可能になる．

提案の全体の図として挙げられるのが図 1 である．図 1 では大きく三つに分けてサーバ，ユーザー，IoT 機器が存在する．まずサーバであるが，このサーバでは RaspberryPi で取得した CPU の温度，クロック周波数を保存する．次にユーザーだが，ユーザーはサーバにアクセスすることで現在から過去までの情報を確認する事が出来る．最後の IoT 機器には，cpu.py と prosess.py といった二つのプログラムが搭載されている．この二つのプログラムは RaspberryPi の温度が上昇し過ぎてしまった時にプロセスを制御する事で温度を下げていくものである．

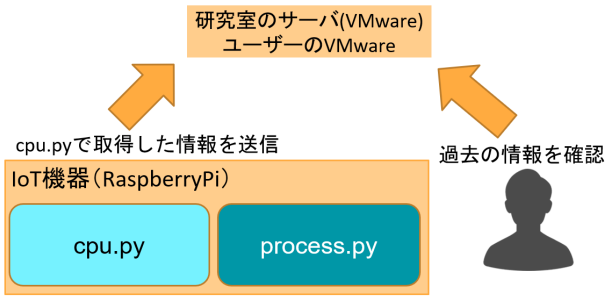

図 3　提案するソフトウェアの構成図

4. 実装と実験環境

4.1 実装

今回の論文で提案を行った RaspberryPi の中に存在する cpu.py，process.py について挙げていく．

図 2，図 3 は cpu.py，process.py それぞれの構成図である．図 2 では，今回作成した python プログラムである cpu.py が CPU から，CPU の温度と CPU のクロック周波数を取得している．cpu.py は cron で利用者が指定した自由な時間から実行され三分刻みで取得を行っている．また，取得した温度と周波数情報に加え，日付，日時を csv ファイルとして作成するものが cpu.csv である．cpu.py が 23:58〜24:01 の様に日付を跨いで実行された場合は 24:01 時点，即ち終了時点での日付で csv ファイルが作成される．

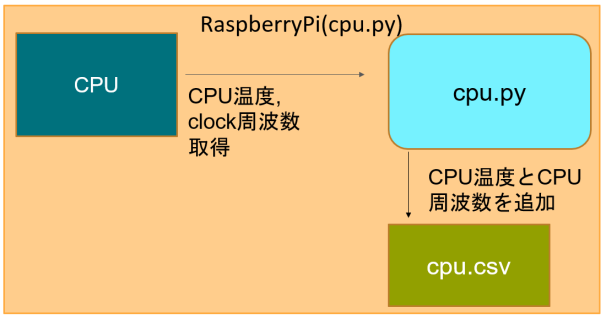

図 4　RaspberryPi から本体の温度とクロック周波数の取得する cpu.py の仕組み

図 3 は，process.py という python のプログラムの概要である．process.py は CPU からプロセスの情報と現在の CPU の温度を取得し，閾値を超えた時にプロセスの優先度を下げる機能を持つ．この閾値は先に挙げた cpu.csv から導いている．

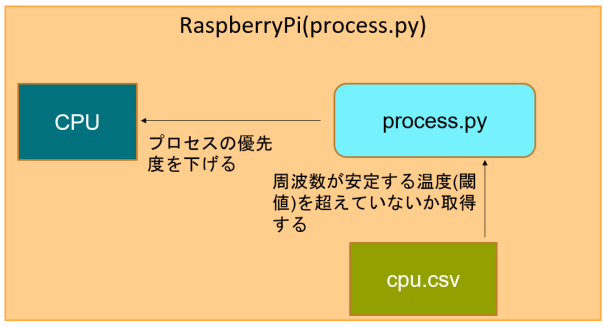

図 5　プロセスの優先度を変化させる process.py の仕組み

4.2　実験環境

今回の実装環境は，以下の通りである．

- RaspberryPi3B+:
  Rasbian GNU/Linux 10 (buster)
- VMware(ubuntu)

温度を測定している関係から外部気温と RaspberryPi を囲っている保護ケースの変化で結果が多少変わる可能性があるが，室温は 20.5 度〜23 度 (冬)，23 度〜28.5 度 (夏) の条件の元 (室温計測定) で行い，ケースは図 6 の様に金属製のもので実験を行った．また，電圧は変化させていない．なお，RaspberryPi はネットワークから現在時刻を取得して使用しているため，cpu.py が停止しても正しい時間を得られる．

図 6　RaspberryPi のケース

5.　評価と分析

ここでは実際に cpu.py を動作させた結果として得られたデータについての分析を行う．条件として UnixBench や yes コマンドを使用し CPU 使用率を 100 %にした上で cpu.py を用いて計測した．また，図 7 に実験に使用した RaspberryPi の最低動作 (クロック) 周波数と最高動作周波数を挙げ，最低周波数の設定が 0.7GHz，最高動作周波数の設定が 1.4GHz である事が分かっている．

```
pi@raspberrypi:~ $ cpufreq-info -o
          minimum CPU frequency  -  maximum CPU frequency  -  governor
CPU  0       700000 kHz ( 50 %)  -    1400000 kHz (100 %)  -  ondemand
CPU  1       700000 kHz ( 50 %)  -    1400000 kHz (100 %)  -  ondemand
CPU  2       700000 kHz ( 50 %)  -    1400000 kHz (100 %)  -  ondemand
CPU  3       700000 kHz ( 50 %)  -    1400000 kHz (100 %)  -  ondemand
```

図 7　RaspberryPi の最低動作周波数と最高動作周波数

図 10〜図 16 は連日で測定を行っている．また，図 17 は図 10〜図 16 までの全ての測定を全体図として纏めたものであり，測定を行った全ての図は三分間毎に動作周波数と温度を取得しているが都合上一部省略されてしまっている．

UnixBench や yes コマンドを使用し CPU 使用率を 100 %にして負荷をかけた状態で三分間測定を行ったが，UnixBench は二つ動かし，yes コマンドは 60 回使用している. これらの図の全体的な傾向として，温度が上昇し続けるとクロック周波数が使用している RaspberryPi の上限である 1.4GHz から下がり，1.2GHz になっている．図 17 は図 10〜図 16 までの計測方法とは異なり，CPU 使用率を増加させていた UnixBench や yes コマンドを停止し，cpy.py を 10 個起動し，三分経過後同じタイミングで 10 回測定したデータをグラフ化したものである．図 18 は言わば特に仕事をさせていない時の状態である．そのため，本来であれば大きな割合を占めるケース内部の熱環境に影響されず，最低 CPU 温度値である 41.3 ℃から最高 CPU 温度値である 59.6 ℃の間に収まり，RaspberryPi の上限である 1.4GHz も値として観測できるようになっているため，図 10〜図 16 は温度が上昇しているために CPU のクロック周波数を落としている (クロックダウンしている) ということが言える．図 10 と図 11 である 11 月 1 日と 11 月 2 日には，UnixBench や yes コマンドに加え RaspberryPi 上で Youtube で生配信を開き，1,2,3 と段階を踏みながら同時に同じ配信をブラウザを開いた．その結果，三つ目からは RaspberryPi の画面が固まり始め，一時間程度画面がフリーズし何も操作を受け付けない状態に陥った．その時の図が図 8, 図 9 である．このフリーズの際にはプロセスを含め全てが完全停止していたと考えられ cpu.py も三分間の継続観測が出来ていない時間が存在している．図 8，図 9 の様に温度が 77 ℃以上の状態を継続するとフリーズすることが見て取れ，いずれもクロック周波数は 1.2GHz で上限である 1.4GHz は出ておらず，1.2GHz よりも下回らな

かった．これらのことから温度が上昇するにつれてクロック周波数を上限から下げ，温度の上昇を抑えようとし，抑えきれずに 77 ℃以上の状態が連続するとフリーズ (システムダウン) に至るということが言える．

また CPU 使用率 100 %，クロック周波数の最高点が 1.2GHz という条件で UnixBench のスコアを比較した際には，計測時の温度が高い方のスコアは処理効率の部分を筆頭にスコアが落ちていたため，処理効率という点に置いても温度が低い方が良い状態だと分かる．

20:13	77.9	1.2	0:20	停止点
20:19	77.4	1.2	0:33	
20:39	77.9	1.2	0:26	
21:12	77.9	1.2	1:04	
21:38	78.4	1.2	0:03	

図 8　11 月 1 日の停止時間

12:10	79.0	1.20	0:03	停止点
12:13	79.0	1.20	0:04	
12:17	78.4	1.20	0:07	
12:24	78.4	1.20	0:14	
12:38	78.4	1.20	0:08	
12:46	75.2	1.20	0:21	
13:07	78.4	1.20	0:03	

図 9　11 月 2 日の停止時間

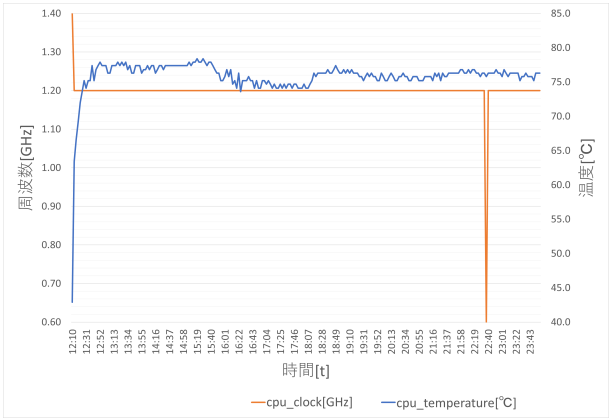

図 10　10 月 30 日の計測結果

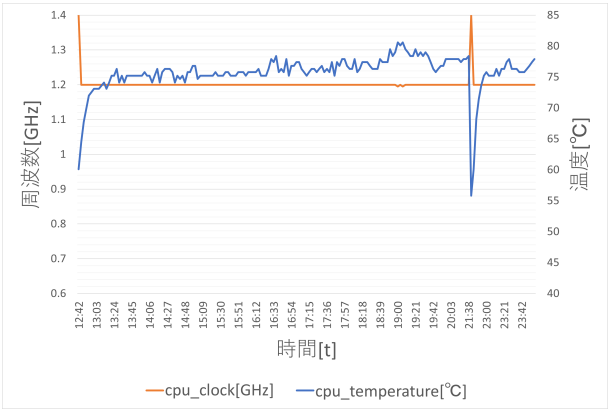

図 11　11 月 1 日の計測結果

図 12　11 月 2 日の計測結果

図 13　11 月 3 日の計測結果

6.　議論

RaspberryPi が CPU の熱によってクロックダウン・システムダウンしてしまう事を課題として挙げ，RaspberryPi 本体の CPU のクロック周波数と温度の関係を検証し，安定したクロック周波数を出し続けるための閾値とする事でプロセスの温度での制御を行えるようにした．

クロック周波数と温度の関係を特定するために検証で行った cpu.py だが，取得したグラフから CPU 使用率を

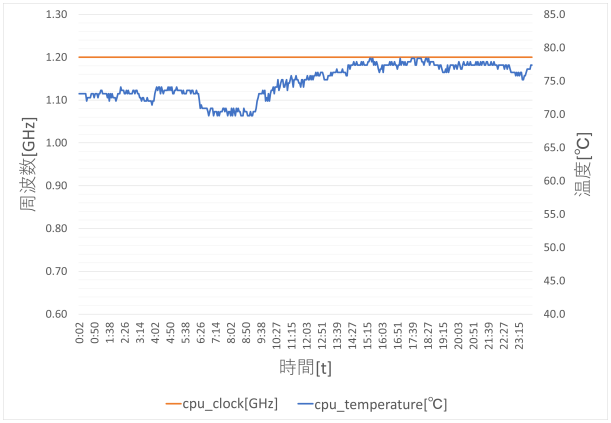

図 14　11 月 4 日の計測結果

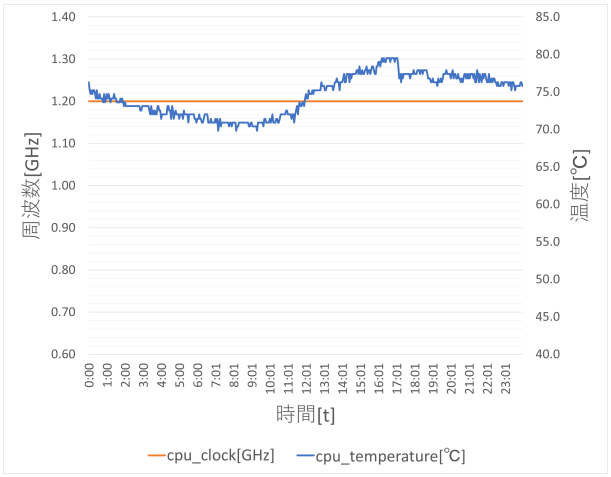

図 15　11 月 5 日の計測結果

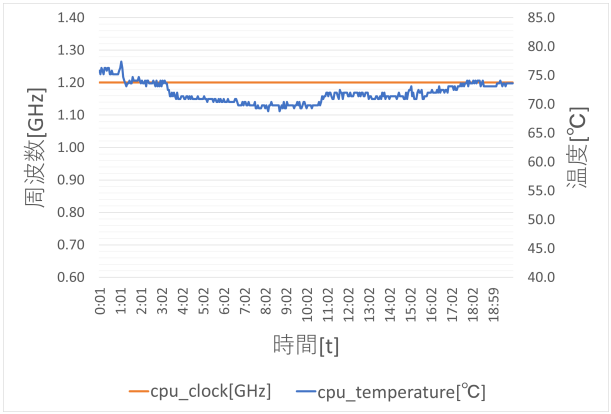

図 16　11 月 6 日の計測結果

100 %にした状態である稼働中には，70 ℃を超えた辺りでクロック周波数が 1.2GHz に制限され，75 ℃を超えるとプロセスの実行が不安定になるということが分かった．図 10〜図 17 の様にクロックダウンが 70 ℃付近で起こり，77 ℃を超えたまま温度が下がらなかった場合にはシステムダウンすることが，その時間の計測結果である図 8,9 の cpu.py を始め全てのプログラム，プロセスが停止している事から事実として証明された．

クロックダウン自体は，RaspberryPi の CPU 温度が 70

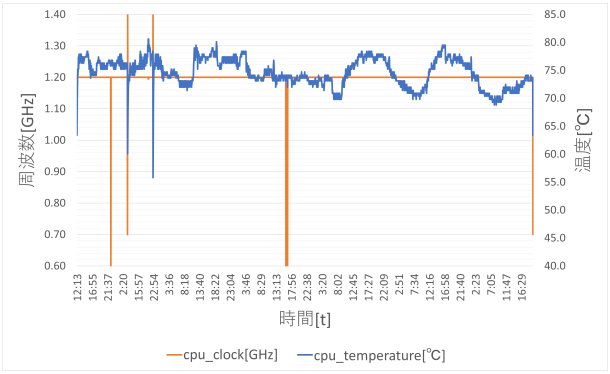

図 17　図 10〜図 16 の計測結果を纏めたもの

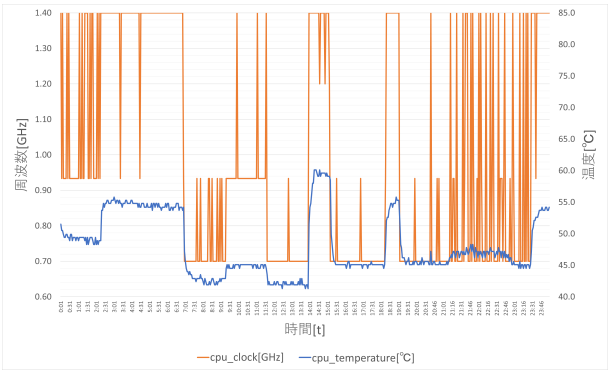

図 18　cpu.py を 10 個起動しているのみの状態での温度と周波数

℃の時に起き，以降 75 ℃でも続いていた．また，温度が 70 ℃を下回ればクロック周波数は 1.4GHz，上回れば 1.2GHz となるわけだが，cpu.py を実行しただけで 65 ℃を超えてしまうため，閾値を 65 ℃から 70 ℃の間とするのは実行できるプロセスの数や重さ (大きさ) を制限してしまう事になってしまうため理想的とは言えない．そのため，RaspberryPi 上でプロセスが動作している状況下であっても，75 ℃を超えないように閾値を設定し，process.py で実行中のプロセスに対して働きかければ温度は 75 ℃以下となりクロック周波数は安定して 1.2GHz を出せる．これにより，プロセスの制御が出来，安定した処理効率でプロセスが実行出来るといえる．

提案では原因となっているだろうプロセスの優先度を一時的に下げることで閾値を越えた温度を下げ，閾値を下回ったならば，プロセスの優先度を元に戻すとしている process.py だが，現状は nice コマンドの一種である renice コマンドを使用する考えである．この renice コマンドは実行前のプロセスの優先度を変更する nice コマンドと異なり，実行中のプロセスの優先度を変更することが出来る．

今回は cpu.py を使いクロック周波数を安定させることが出来る閾値を決めた．次回は process.py を使いプロセスを制御することでどのくらい温度の上昇を抑えることが出来るのか，また今回閾値とした温度が最もクロック周波数が安定するのかについて検証評価していく．

7. おわりに

今回は様々な計測で使用されることの多い RaspberryPi について取り挙げ，RaspberryPi が熱によってクロックダウン・システムダウンをしてしまい計測や作業が中断されてしまうことを課題とした．

提案では，RaspberryPi の CPU の温度とクロック周波数といった CPU 情報を利用してプロセスを制御することで，RaspberryPi が熱によってクロックダウン・システムダウンの対策が出来ることを挙げた．また，検証と評価では，RaspberryPi の温度の上昇・下降に着目し，上限近くクロック周波数が出ている温度について突き止めた．

この論文では，RaspberryPi が CPU から発せられる熱によってクロックダウン・システムダウンしてしまい作業全体が停止または，進行が遅れてしまうことを課題として挙げ，CPU 情報を利用したプロセス制御を行うことで RaspberryPi を始め，CPU を搭載している熱を発する IoT 機器に対する熱問題を解決する事に貢献した．

参考文献

[1] Krintz, C., Wolski, R., Golubovic, N. and Bakir, F.: Estimating Outdoor Temperature from CPU Temperature for IoT Applications in Agriculture (2018).
[2] Dye, B.: Distributed computing with the Raspberry Pi, *K-State Electronic Theses* (2014).
[3] Zhengguo Sheng, C. M.: Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT, *IEEE* (2015).
[4] Tetsuya Oda, L. B.: Experimental Results of a Raspberry Pi Based WMN Testbed Considering CPU Frequency, *IEEE* (2016).
[5] Raid Ayoub, Krishnam Indukuri, T. S. R.: Temperature Aware Dynamic Workload Scheduling in Multisocket CPU Servers, *IEEE* (2011).
[6] Jae Min Kim, Young Geun Kim, S. W. C.: Stabilizing CPU Frequency and Voltage for Temperature-Aware DVFS in Mobile Devices, *IEEE* (2013).
[7] Elsts, A., Fafoutis, X., Duquennoy, S., Oikonomou, G. and Goto, N.: Temperature-Resilient Time Synchronization for the Internet of Things, *IEEE* (2018).
[8] Odiowei, C.: State-space independent component analysis for nonlinear dynamic process monitoring, *Chemometrics and Intelligent Laboratory Systems* (2010).
[9] Michael, Chad Paradis, V. M. W.: A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement, *MDPI* (2016).
[10] Sridharan, M., Devi, Dharshini and Bhavadarani: IoT based performance monitoring and control in counter flow double pipe heat exchanger, *Internet of Things* (2019).