修士論文

減災のための Web 上インフラサウンドデータベースの開発

Development of Web-based infrasound database

for disaster mitigation

報 告 者

学籍番号: 1215048

氏名: 平田 大祐

指 導 教 員

山本 真行 教授

平成 31 年 2 月 12 日

高知工科大学 電子・光システム工学コース

1 章 序論

1.1 インフラサウンドについて

インフラサウンドとは周波数 20 [Hz]以下の音である。人間の可聴域は 20 [Hz]から 20 [kHz]程度であるため、インフラサウンドは周波数が可聴域未満の音である。インフラサウンドは、特性周波数が低いため長距離伝搬できるという特徴がある。空気中を伝わる音波は主に 3 つの要素で減衰する[1]。「幾何学的減衰」「非線形減衰」「空気の粘性による減衰」の 3 つである。このうちインフラサウンドは「空気の粘性による減衰」が小さい。「空気の粘性による減衰」は空気との摩擦であるから、振動数が小さいインフラサウンドは摩擦する回数が少ないため結果的に減衰が小さいものとなる[1]。

インフラサウンドが注目される理由の一つに CTBTO(包括的核実験禁止条約機関)の核実験監視網において、インフラサウンド観測が手法の一つとして利用され、国際的にインフラサウンド観測網が整備されているという背景がある[2]。

インフラサウンドは、火山噴火、地震、津波、落雷、土砂崩れ、大規模爆発などの大規模な地球物理学的現象によって発生することが知られており、これらをリモートセンシングすることで、これらの災害の早期探知、規模解析などに用いることができると考えられる[3]。

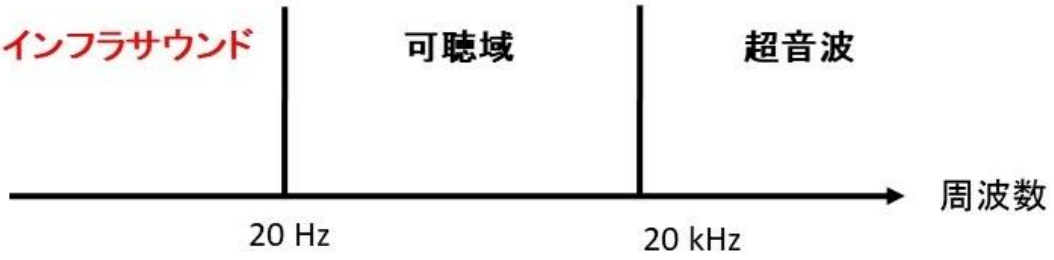


図 1.1 インフラサウンド

1.2 背景

著者の所属する高知工科大学システム工学群宇宙地球探査システム研究室では、インフラサウンドをリモートセンシングするため、2005 年からインフラサウンドに関する研究が開始された。西山(2007)では圧電素子を用いた低価格なセンサーが開発された[4]。山田(2009)では PSD と半導体レーザを用いた非接触式光学センサーが開発されるなど、各種インフラサウンドセンサーの開発が行われてきた[5]。

上述の基礎開発を経て、本研究室では株式会社 SAYA との共同開発により従来の製品より安価で高性能な複合型センサー、SAYA ADX Ⅱ-INF01C/D(図 1.2)の開発に 2015 年 2 月に成功し、センサーの設置地点が増えつつあり、現在では図 1.3、表 1.4 のように高知県の足摺岬、室戸岬を始めとする日本各地 25 か所以上にセンサーが設置されている状況である。

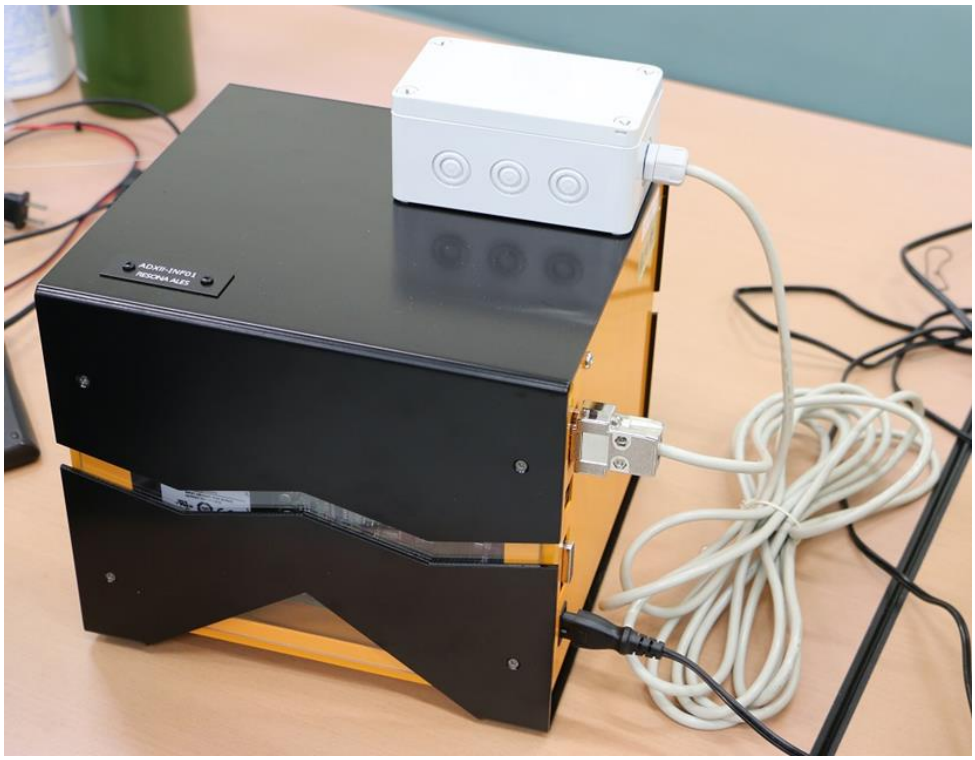


図 1.2　SAYA ADXⅡ-INF01C/D

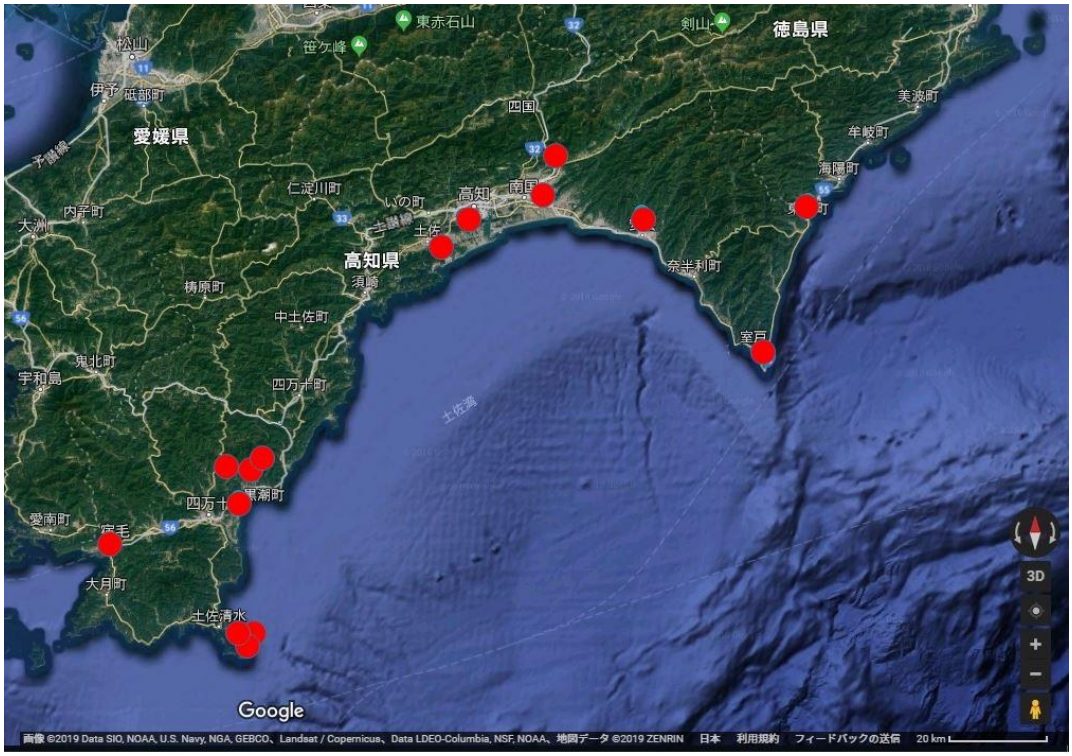


図 1.3 高知県内のセンサー設置状況

表 1.4 高知県内のインフラサウンド設置個所

FTP先フォルダ	市町村名	地点名
csv0	香美市	土佐山田町宮ノ口
csv1	黒潮町	浮鞭
csv2	黒潮町	蜷川
csv3	黒潮町（～2018.11）	上川口
csv4	黒潮町	出口
csv5	黒潮町	馬荷
csv7	宿毛市	小筑紫町田ノ浦
csv8	土佐清水市	足摺岬1
csv9	土佐清水市	足摺岬2
csv10	土佐清水市	足摺岬3
csv11	土佐市	宇佐町宇佐
csv12	高知市	春野町芳原
csv13	南国市	物部
csv14	安芸市	西浜
csv15	東洋町	生見
csv16	室戸市	室戸岬町

現在センサーからのデータは携帯電話回線や光通信網などを利用して、サーバに観測データが自動送信集約されている状況である。しかし、サーバにデータが蓄積されているものの、その解析にあたっては、実際のインフラサウンド発生イベントを別手段で観測後、その時系列付近にデータが存在しているか否かを人の目で確認している状況であり、インフラサウンドから直接的にイベントを自動検出できていない。

また観測データを逐次保存しているものの、一元的な管理については整備途上のため十分には出来ておらず、解析などを行うにあたり手動でデータを検索する必要がある。さらに複数の形式のセンサーが混在するため複数の観測データ規格が混在しているなどの状況もある。

またインフラサウンドに限ったことではないが、このような観測装置を複数配置し、観測データを集めている機関はそれなりに存在するが、一番重要なのは、それらのデータの解析で何が得られるかが重要である。そのため、観測データを公開し、観測機関外の研究者がアクセスできるようにしている機関も多い。実例では IUGONET(Inter-university Upper atmosphere Global Observation NETwork)と呼ばれる、超高層大気地上観測データを検索取得できる研究インフラが存在する[6]。特に IUGONET は、データのデータ、いわゆるメタデータの管理に Git を用いる点が特徴的である[6]。Git は主にソースコードのバージョン管理に用いられる。本研究のソースコードも Git による管理を採用している。IUGONET で

はメタデータを XML 形式で管理し、Git で管理運用を行うことで、観測データを検索取得しやすい環境が整備されている[6]。この例から、データを公開するだけでなく、ユーザフレンドリーな形でアクセス解析できる手段を提供することで、その分野の研究を加速させることが出来ると考えられる。

1.3 目的

本研究では、複数のセンサーを一元的に管理運用し、観測データも含めた管理を実現する総合的な Web アプリケーションの開発を目的とする。具体的には、センサーの監視運用、アーカイブ観測データのデータベース化、また災害時に運用される可能性を考慮した冗長性をもつ Web アプリケーションの開発が目的である。センサーの運用監視は、センサーからデータが継続的に送信されているかという点を監視する。観測データのデータベース化は送信されてくるデータを整理された状態でアーカイブし、ユーザからのリクエストに応じて、出力できる機能をもったデータベースとする。本研究ではサーバの冗長化を行うため、BGP(Border Gateway Protocol)を用いた冗長構成を構築する[7]。

2 章 開発手法

2.1 インフラサウンド観測網について

現在、本研究室では 20 か所以上のインフラサウンドセンサーを設置しており、一か所につき、センサーと PC がセットになって設置されている。センサーからインフラサウンド、加速度、温度、気圧、GPS を含む時刻情報といったデータが PC に送られ一時蓄積されたのち、4096 サンプル分のデータが蓄積されると、それらが CSV ファイルとして整形される。観測は約 5 Hz サンプリングであるので、約 14 分のデータとなる。CSV ファイルは図 2.1 のように FTP で 2 か所の FTP サーバへ同時送信されるようになっており、一定の冗長性が確保されている。現在 FTP サーバの 1 つは、本研究室に設置された物理サーバ上の VM である Ubuntu16.04 と Vsftpd によって構築されている[8]。また送信される CSV データの見本を図 2.2 に示した。

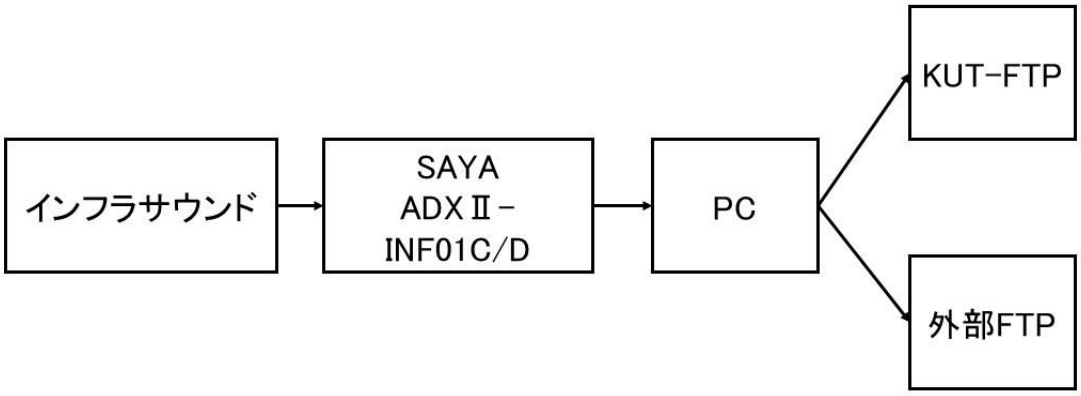


図 2.1 インフラサウンドが送信されるまで

	A	B	C	D	E	F	G	H	
1	PC時間	PC秒	加速度X	Y	Z	騒音	気圧	1PPS	インフ
2	23:56	6.26	17.3134	9.2951	10.4697	9.8177	108.0019	3455.99	
3	23:56	48.635	17.3134	9.1419	10.4697	9.859	108.0004	3839.996	
4	23:56	48.835	17.4155	9.0397	10.4697	10.0048	108.0051	3839.996	
5	23:56	49.59	17.6709	9.1419	10.7251	10.2825	108.0099	3839.996	
6	23:56	49.235	17.8241	9.244	10.8783	10.4558	108.0115	3455.99	
7	23:56	49.424	17.8241	9.193	10.7762	10.3568	108.0067	3455.99	
8	23:56	50.224	17.6709	9.0397	10.4697	10.035	108.0019	3839.996	

I	J	K	L	M	N	O	P	Q
インフラサウンドDC	インフラサウンドAC	ボード温度	GPS年	月	日	時	分	秒
4761.2298	919.8282	38.7809	0	0	2	5	18	43.651
4738.8478	855.2091	38.7779	0	0	2	5	19	21.999
4735.9598	855.9311	38.7772	0	0	2	5	19	22.179
4742.0968	861.7071	38.7756	0	0	2	5	19	22.38
4743.1798	859.9021	38.7786	0	0	2	5	19	22.539
4733.7938	854.1261	38.7792	0	0	2	5	19	22.709
4748.5948	868.2051	38.7796	0	0	2	5	19	23.429

図 2.2 インフラサウンド観測データ例

2.2 運用物理環境について

本研究室で運用している物理サーバ上には Hyper-V 2016 が導入されており、これはいわゆるハイパーバイザであり、VM (Virtual Machine)として複数の OS を同じ物理サーバ上で実行可能である。それらを利用して、今回は Ubuntu18.04LTS などを導入し、Web アプリケーションを運用する。

本研究で用いる物理サーバは SuperMicro 社製のサーバであり、Intel Xeon E5-2620v4 を 2 ソケット、DDR4 Memory を 128 GB、ストレージとして、NVMeSSD 1.2 TB、HDD (RAID6) 32 TB を備える。

Hyper-V Server 2016 は Microsoft 社製の Windows 系 OS であるが、GUI は存在せず、CUI (PowerShell)のみで操作可能な OS でありハイパーバイザ機能しか使用できない。しかしながら無償で使用可能な点が大きな特徴である[9]。本研究室で採用した理由としては、Windows10 などといったクライアント系 OS 上でサーバを管理する UI が提供されており、WindowsPC が多い本研究室には管理運用面でメリットがあると判断した。他にも似たソリューションとしては、KVM (Kernel-based Virtual Machine)や Xen、VMware ESXi などが存在する。

Ubuntu は Debian ベースの LinuxOS であり、”Ubuntu”はズール一語で「他者への思いやり」を意味する。Ubuntu は一般ユーザが DesktopPC に用いるものから、サーバ用途でも幅広く使用されており、近年シェアが高まっている[10]。本研究で用いる LTS (Long Time Support)版は 5 年間のサポートが保証されており、サポート期間は 2023 年 4 月までの予定である。また、Ubuntu は Python が標準搭載されているが、本研究で用いる 18.04 から Python 3 系が標準となった。そのため本研究でも Python3 を利用する。

ネットワーク機器として YAMAHA 社製ルータ RTX1210 を採用している。本研究室にはシステム工学群電子系研究室用の通常用 LAN とグローバル IP アドレスを運用するために、グローバルネットワークの 2 つのネットワークに接続されている。RTX1210 は 3 つの LAN インターフェースを持っているため、研究室 LAN、電子系 LAN、グローバルネットワークと必要なすべてのネットワークに接続可能となっている。また RTX1210 は PBR (policy based routing)と呼ばれる機能が特徴的である[11]。従来は宛先 IP アドレスを見て、パケットルーティングを行う手法が一般的であるが、PBR の場合は、送信元 IP アドレスや、送信元、宛先ポート番号といった要素も、判断材料として利用可能である。これらを利用して本研究室では、高知工科大学の Proxy 宛の通信を電子系 LAN に、Proxy で対処不可能な通信をグローバルネットワークにルーティングすることによって負荷分散と冗長化を行っている。

また本研究室では、前述の SuperMicro 社製メインサーバの他にバックアップ用サーバが存在し、1 日 1 回日本時間午前 2 時ごろにメインサーバからバックアップサーバへインフラサウンド観測データなどを含めた全データを同期し保存している。これは RAID のエラーや人為的ミスによる削除からデータを守るためのバックアップとして利用している。バックアップは Robocopy.exe と呼ばれる Windows に標準搭載されたバックアップ等に用いるソフトを利用した Powershell スクリプトを構築した[12]。これらのバックアップが完了すると、Slack とメールによる管理者への通達が行われる仕組みとなっている。またエラーが発生した際には、管理者に対処を促すエラーメッセージとともに通常より注目度が上がる形で通達される。具体的には Slack 通知がメンション付きで行われる。

図 2.3　SuperMicro 社製メインサーバ

Ubuntu1804-Hirata-01	実行中	0%	8192 MB	12:56:51
Ubuntu1804-Hirata-02	実行中	0%	8192 MB	12:56:48
Ubuntu1804-Hirata-03	実行中	0%	8192 MB	12:50:38
VVVM	実行中	0%	2048 MB	11.06:57:32
VyOS-1.2.0-1	実行中	0%	1024 MB	14:19:49
VyOS-1.2.0-2	実行中	0%	1024 MB	15:48:22
VyOS-1.2.0-3	実行中	0%	1024 MB	15:54:39

図 2.4 Hyper-V 管理画面

図 2.5　YAMAHA RTX1210

2.3 開発仕様

本研究は Django と呼ばれる Python 言語で使用する Web アプリケーションフレームワークを前述の Ubuntu18.04LTS 上で利用する[13]。またバックエンドの DB として MariaDB、Web サーバとして Apache を利用する。クライアントからの HTTP リクエストは一度 Apache で受けとめ、必要に応じて WSGI(Web Server Gateway Interface)と呼ばれるインターフェースを通じて Django に HTTP リクエストを転送し、動的処理を行い、HTTP レスポンスを返す仕組みである[14]。

Django は Python で構築されたオープンソースの Web フレームワークであり、シンプルに開発できることが特徴である。またデータモデルにおいて、SQL 文を自動生成して、開発者側は Python のオブジェクト操作でデータ操作を完結できることも特徴である。

本研究にでは Python3 による Django を採用するが、これらは動的な Web サイトを実現するためのものである。静的サイトは HTML や JPG 画像ファイルといった不変のデータを Apache などの Web サーバがファイルシステムを通じてサーバのフォルダから読み込み、クライアントにファイルを HTTP に乗せて返す。動的サイトは基本的にはクライアントからリクエストがあるたびにプログラムが実行され HTML の形で出力され、それがクライアントに HTTP で返されるような形となる。そのプログラム上でインフラサウンドデータを処理したり、リクエストを受けて、その情報に応じた処理などを行う実装をおこなう。Django 以外にも PHP 言語の CakePHP、JavaScript 言語の Node.js、C#言語の ASP.NET など様々な言語とフレームワークが存在する。

また本研究では、サーバ負荷を低減するために静的なファイル、HTML や JPEG 画像ファイルなどのファイルは図 2.6 のように、Django を通さず Apache が直接応答する設計とした。

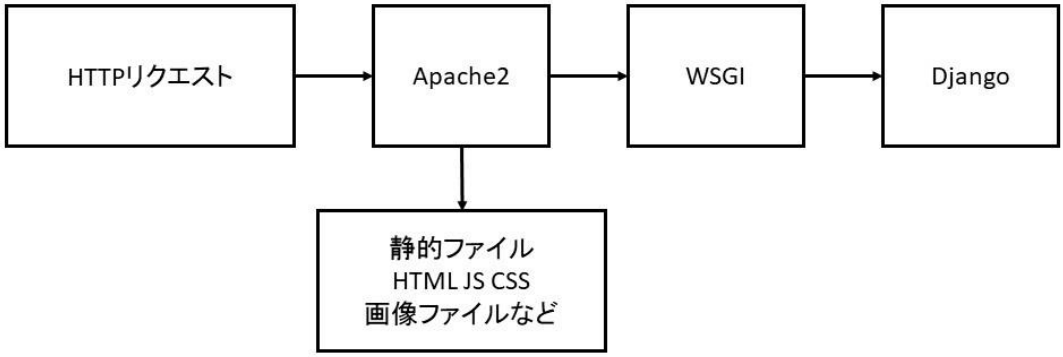


図 2.6　HTTP 処理の流れ

MariaDB は MySQL から分岐して開発されたオープンソースの RDBMS(relational database management system)であり、MySQL と互換性を持ったデータベースである[14]。有償版 MySQL に搭載されているスレッドプール機能が使えるなどのメリットがある。

本研究では災害対応の冗長性を評価するため、Ubuntu18.04LTS を 3 台用意し、またそれらは VyOS とともに設置。Ubuntu と VyOS のセットがそれぞれを 1 拠点と想定する。

VyOS はオープンソースで開発されているネットワーク用 OS であり、Debian 系の Linux がベースとなっている。VyOS は物理環境をはじめ、仮想環境にも構築可能であり、無償で高機能なソフトウェアルータを構築可能としている。これらと BGP を利用し冗長性を担保する。

BGP は Border Gateway Protocol と呼ばれるルーティングプロトコルの一種である。インターネットや LAN 上で使われている IP アドレスは、名前の通り IP(Internet Protocol)における住所を表している。例えば 192.0.2.0/24 であらわされるネットワークは 192.0.2.1 ~ 192.0.2.254 までのホストアドレスを持つ集合体である。このネットワーク内では Ethernet

と呼ばれる MAC アドレス(物理アドレス)をベースとして通信している。このネットワークから別のネットワーク、例えば 198.51.100.0/24 等に通信したいルーティングが必要である。ルーティングを行うにはルータと呼ばれるネットワーク境界に設置される装置が必要となる。必然的にルータはどのネットワークがどこに存在しているかを知っているか、あるいはルータ自身が知らない宛先をすべて委託するデフォルトゲートウェイと呼ばれる宛先が設定されている必要がある。一般家庭やオフィスに設置されているものは基本的に後者である。しかしながらインターネットそのものを構成しているルータ達はお互いの経路情報をすべて共有し、インターネットを構築している。これらは常に動的に変化しており、人力での管理は不可能であるから自動で経路をやり取りするプロトコルが生まれた、それがルーティングプロトコルであり、BGP はその一種で、現在のインターネットの標準となっている。現在 IPv4 の経路数は 70 万を超えており、経路数の肥大化が危惧されている状況である。

このように BGP はインターネットにおける重要な役割を果たしており、それに対応するため柔軟な設定運用が可能である。本研究ではそれらを生かした冗長構成を構築し、Web サーバに対する冗長アクセスを実現した。

3 章 結果

3.1 インフラサウンド網監視

3.1.1 インフラサウンドセンサーリスト

現在インフラサウンドの観測点情報はエクセルファイルなどをベースとした手動の管理体制が続いており、研究進展に伴ってセンサー数が増える状況下で管理が煩雑化している問題がある。そのため、本研究上で各センサーがどこに設置されていて何番の番号が与えられているのかを閲覧できるページを実装した。各センサーのデータは MariaDB に収納されており、管理者が Web 上でデータを書き換えることが可能である。これらのデータの中には各機能で使用する CSV 番号、緯度、経度、IPv4 アドレス、観測データの保存場所などが格納されている。

ユーザがページにアクセスすると、MariaDB から全センサーのデータが読み込まれ、座標や名前が処理されて JavaScript に渡され、HTML ともにユーザへ送信される。ユーザ側のブラウザで JavaScript によって Google マップがレンダリングされ、図 3.1 のような画面が表示される。画面の IPv4Connection はセンサーに対する疎通を表示しており、True で正常、切断時 False になる。

また図 3.2 のように、リスト形式のみページを実装した。これらのページは Google マップをレンダリングせず、HTML の Table 要素を用いて表示を行っている。

また Django の admin ページと連携し、管理者ユーザでログインすることによって MariaDB で管理されている図 3.3 のようなセンサー情報管理画面にアクセスできる。ここでは既存のセンサーの情報変更、センサー移動に伴い、座標データを変更することが出来る。また新たなセンサーを追加したり、既存のセンサーを削除することも可能である。

図 3.1 センサーマップ

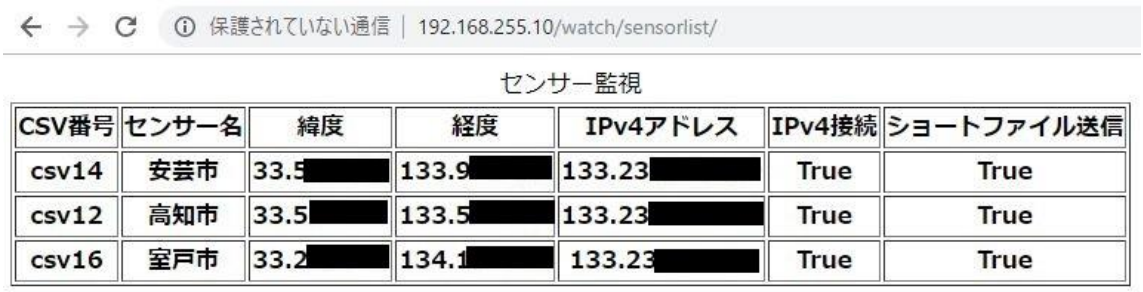


図 3.2 センサーリスト

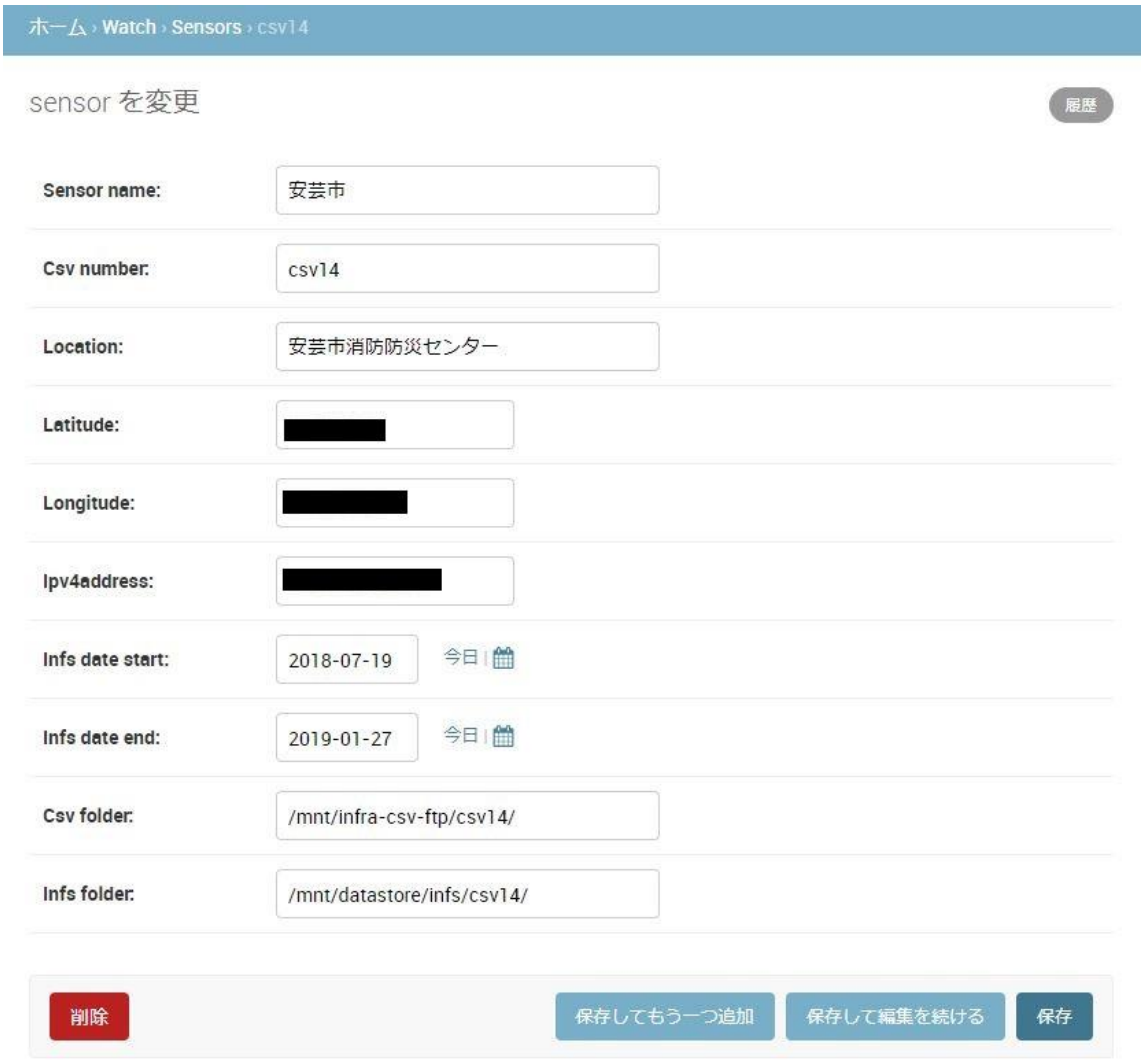


図 3.3 センサー情報管理画面

3.1.2 センサーIPv4 アドレス疎通監視

各センサーには PC が同一拠点に設置され、PC にはグローバル IP アドレスによってリモートデスクトップアクセス可能なシステムが構築されている。そのため、対象の IP アドレスと特定のポート番号に対する疎通を確認することで、少なくとも拠点がインターネットに接続されていて、動作していることの確認が可能となる。これを利用して、本研究アプリケーションからパケットを飛ばし、拠点の生存確認を行うページを実装した。

3.1.3 センサーデータ受信監視

各センサーから FTP で送られてくるデータは 4096 サンプリングの CSV ファイル以外にもショートファイルと呼ばれる、観測データを準リアルタイムで見るための機能が観測ファームウェア上で実装されている。この中身は 4096 サンプルのデータの短縮型であり、常に同名のファイル (short_file.csv)で定期的に上書きされ続けている。つまり、このファイルが生成されたタイムスタンプを追えば、センサーと PC が正常に動作し、最新のデータを送信できているかの確認が可能となる。本研究では、各拠点のショートファイルが監視プログラム実行時の 5 分以内にアップロードされたものであれば、送信されていると判断し、実装を行った。

3.1.4 Infs ファイル形式

現在のインフラサウンドセンサーから出力される CSV ファイルのデータは長期間連続解析などに用いる際には細かな不都合が生じることがあり、それらの CSV ファイルから必要な個所を抽出し、解析及び保存を容易にするための独自バイナリファイル形式を開発した。Infs 形式は header 部と連続した data 部で構成されており、data 部の数は header に記すことによって、その数だけ Data 部を連続させる。これらにより infs 形式は可変長を実現している。実運用上、ファイルサイズ、システム全体でのファイル数などを考慮し、1 センサーの 1 日分データを 1 ファイルにして保存する運用を現在行っている。

Header 部のサイズはファイル先頭から 256 バイト分となっており、図 3.4 のヘッダー例の文字列を、カンマ区切りを文字として結合し、それを ASCII エンコードしたものをヘッダーとしている。

Data 部はインフラサウンドセンサーの GPS 時刻と PC 時刻、GPS フラグ、インフラサウンド振幅 AC 成分を 1 つの data として格納している。GPS 時刻と PC 時刻はエポックミリ秒と呼ばれる 1970 年 1 月 1 日午前 0 時 0 分 0 秒からの経過時間に基づく値として、GPS フラグは GPS が有効だったレコードは受信衛星数が 1 以上の値で格納され、GPS が無効だった場合は 0 が格納される。インフラサウンド AC 成分は CSV データから抽出した数値を 1 万倍して小数点以下をなくし、整数型の状態で格納した。現在は開発段階ということもあり、バイナリエディタ等で可読性が高いビッグエンディアンを採用している。デコード時にはこれらの設定を考慮して読み出す必要がある。またそれぞれの値とバイナリにする際の型の対応を表 3.1 に示した。また実際に生成したバイナリファイルをバイナリエディタで開いたものが図 3.6 である。

Infs ファイル形式を読み込むためにはバイナリ形式を理解しなければならないが、それでは利便性が失われるので、本研究では Python3 による infs デコーダーを実装している。デコーダーはそれ自身が含まれる InfsConvert ライブラリの実装という形をとった。このライブラリを import し、Python3 上で infs ファイルのパスを指定するだけで図 3.5 の様にデコードが可能となっている。

また InfsConvert ライブラリは、CSV ファイルを infs 化する際の関数なども定義されており、infs 形式の中心となるライブラリとなっている。

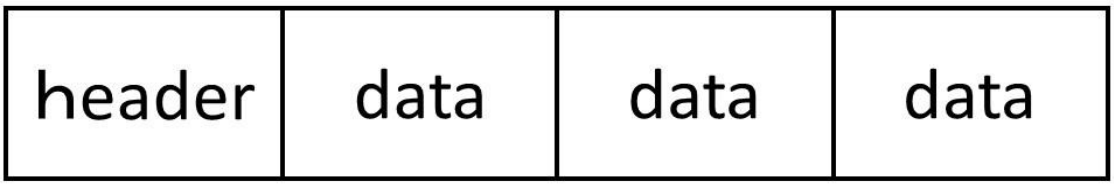
関数としては、「CsvToArray.InfsConvert」、「ArrayToINFS.InfsConvert」、「CsvToINFS1Day.InfsConvert」、「PackINFS.InfsConvert」を実装した。

「CsvToArray.InfsConvert」は引数として CSV ファイルのパスを入力すると、返り値として infs 化手前の Python の配列形式で値が得られる。

「ArrayToINFS.InfsConvert」は上記の配列を実際に infs ファイルとして書き込む関数である。引数として、ヘッダー情報、配列を何時間ごとに分割するか、出力ファイルパスがあり、実行するとファイルが書き込まれる。

「CsvToINFS1Day.InfsConvert」は後述するデータアーカイブに特化した関数で、引数として、CSV 番号、CSV フォルダパス、出力パス、変換する日付情報を入力する。実行すると該当するセンサーの当該日付のデータが infs 化され書き込まれる。

「PackINFS.InfsConvert」は基本的に「ArrayToINFS.InfsConvert」と似ているが、後述のデータ API 向けに特化しており、Django 経由でバイナリデータを返すために、返り値自身が infs 化されたバイナリデータになっている。



ヘッダー例
H_NAME = 'infs'
H_VERSION = '0.1'
H_LOCATION = 'KUT'
H_DATAVERSION = 'Ver1.0'
H_CSV_NUM = csvnum
H_SAMPLES = data部の数
H_INFS_START_TIME = 最初のdata部の時刻データ
H_INFS_END_TIME = 最後のdata部の時刻データ

データ部例
64bit 64bit 8bit 32bit
[ts_csv_gpstime, ts_csv_pctime, GPS_flag, infraAC]
[ts_csv_gpstime, ts_csv_pctime, GPS_flag, infraAC]
[ts_csv_gpstime, ts_csv_pctime, GPS_flag, infraAC]

図 3.4 Infs 形式

表 3.1 値と型の対応

値	型	サイズ
GPS 時刻	unsigned long long	8 バイト
PC 時刻	unsigned long long	8 バイト
GPS_Flag	unsigned char	1 バイト
InfraAC	int	4 バイト

```python
import InfsConvert
import os
import re
import datetime

test = InfsConvert.InfsConvert()
infsfilepath = 'Z:/Data/Infrasound/InfrasoundWEBDB/infs/csv14/csv14-2019_01_24.infs'
data = test.ReadINFS(infsfilepath)
cnt = 0
while cnt <= 10:
    print(data[cnt])
    cnt = cnt + 1

```

PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Daisuke\Documents\GitHub\InfraSoundCSV-Infs\Code> py.exe .\test-class-readinfs.py
[datetime.datetime(2019, 1, 24, 0, 0, 0, 269000), datetime.datetime(2019, 1, 24, 0, 0, 1, 508000), 1, -362.8051]
[datetime.datetime(2019, 1, 24, 0, 0, 1, 332000), datetime.datetime(2019, 1, 24, 0, 0, 2, 689000), 1, -352.6971]
[datetime.datetime(2019, 1, 24, 0, 0, 1, 624000), datetime.datetime(2019, 1, 24, 0, 0, 3, 130000), 1, -347.2821]
[datetime.datetime(2019, 1, 24, 0, 0, 1, 774000), datetime.datetime(2019, 1, 24, 0, 0, 3, 180000), 1, -355.5851]
[datetime.datetime(2019, 1, 24, 0, 0, 1, 961000), datetime.datetime(2019, 1, 24, 0, 0, 3, 388000), 1, -322.3731]
[datetime.datetime(2019, 1, 24, 0, 0, 2, 142000), datetime.datetime(2019, 1, 24, 0, 0, 3, 589000), 1, -321.2901]
[datetime.datetime(2019, 1, 24, 0, 0, 2, 900000), datetime.datetime(2019, 1, 24, 0, 0, 3, 812000), 1, -319.8461]
[datetime.datetime(2019, 1, 24, 0, 0, 3, 49000), datetime.datetime(2019, 1, 24, 0, 0, 3, 978000), 1, -317.3191]
[datetime.datetime(2019, 1, 24, 0, 0, 3, 228000), datetime.datetime(2019, 1, 24, 0, 0, 4, 176000), 1, -309.738]
[datetime.datetime(2019, 1, 24, 0, 0, 3, 419000), datetime.datetime(2019, 1, 24, 0, 0, 4, 389000), 1, -319.4851]
[datetime.datetime(2019, 1, 24, 0, 0, 3, 600000), datetime.datetime(2019, 1, 24, 0, 0, 4, 613000), 1, -318.4021]
PS C:\Users\Daisuke\Documents\GitHub\InfraSoundCSV-Infs\Code>
```

図 3.5　InfsConvert ライブラリとデコーダー

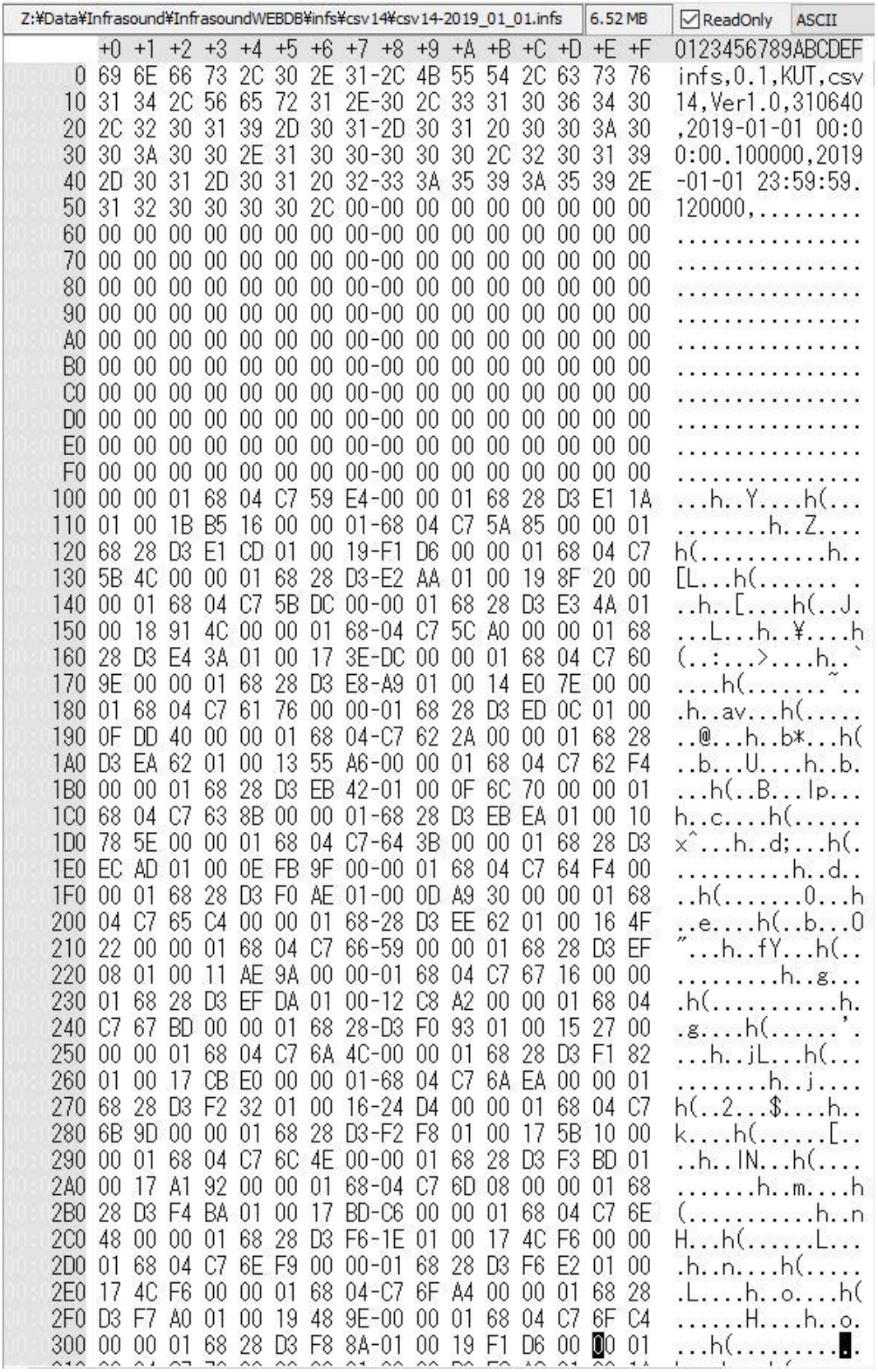


図 3.6　バイナリエディタで開いた infs ファイル

3.2.1　Infs データストア

現在 SAYA ADXⅡ-INF01C/D インフラサウンドセンサーから送られてくるデータはすべて CSV 形式で保存されている。これも本研究で開発した infs 形式に変換し、アーカイブ用生データとして保存する。現在 SAYA ADXⅡ-INF01C/D インフラサウンドセンサー及びその運用には以下のような問題を抱えている。

運用上 CSV データは一日の分のデータを一つのフォルダに保存する(例 フォルダ名 2019_01_01 など)、しかしながら観測は約 5 Hz サンプリングで一定数のサンプリング毎にファイル化されているため、一日のフォルダ内にはどうしても前後する日付のデータが混入してしまう。これを防ぐため、一日の infs ファイルを書き出す際に、前後の日付フォルダ内のデータもメモリ上に取り込み、データの時刻情報に合わせたうえで、改めて 1 日分のデータとして書き出す処理を実装した。

CSV データの中に GPS 時刻情報が存在するが、GPS はその仕組み上、電波受信環境といった外的要因によって、GPS 受信データが途切れたり、受信できないケースがある。この場合のセンサーのデータは西暦 0 年や、データが欠落するなど、時刻データとして矛盾のあるデータが記録されていることがある。それらを検出し、同時に記録されている PC 時刻情報をもとに infs 化を行う処理を実装した。この際に用いる PC 時刻情報は精度が低いことに注意が必要であるほか、PC 時刻は日本標準時で設定しているため UTC に変換を行った上で利用する。

また GPS が正常に受信できている環境においても、GPS の秒の部分のカラムに図 3.7 のような、まるで閏秒のようなデータが出現することが稀にある。おそらくセンサー内部やファームウェア処理に依存したバグだと考えられるが、これも分のカラムの値をインクリメントする事により問題なく処理できるように実装した。この際に Python3 の datetime 型を用いているため、分のカラムが 59 分だったとしても正常に時間のカラムがインクリメントされる実装となっている。

4022	3:27	59.768	1.6343	1.8897	1.7875	30.7293	102.0653	3839.996	-2932.4	195.301	32.7965	2018	10	31	18	27	58.858
4023	3:27	59.957	1.6343	1.9407	1.8386	30.8035	102.0701	3839.996	-2952.98	155.591	32.7971	2018	10	31	18	27	59.028
4024	3:28	0.225	1.6854	1.9918	1.7875	30.7623	102.0669	3839.996	-2991.97	133.931	32.8071	2018	10	31	18	27	59.7
4025	3:28	0.368	1.6854	1.8897	1.6854	30.3223	102.0605	3839.996	-3042.51	81.225	32.8056	2018	10	31	18	27	59.829
4026	3:28	0.558	1.6343	1.7875	1.6343	29.6127	102.0557	3839.996	-2976.81	135.375	32.804	2018	10	31	18	27	60
4027	3:28	0.756	1.6343	1.7875	1.4811	28.7767	102.0526	3839.996	-2970.67	140.79	32.8003	2018	10	31	18	28	0.178

図 3.7　60 秒バグ

本研究で扱うインフラサウンドデータについて言えば、CSV の時刻情報は当然ながら観測時点の未来の日時、観測を開始した 2016 年以前のデータは存在しえない。しかしながら、ごくまれに西暦 1000 年代のような、UNIX タイムスタンプでも表現できない(1970 年 1 月 1 日午前 0 時 0 分 0 秒より前の時刻)データが出現する。恐らく何らかの過程でデータが損傷してしまったと考えられる。このバグの出現はごくまれであるため、バグが含まれる

1 行はデータとして扱わない処理を実装した。これらの実装をまとめると図 3.8 の様になる。

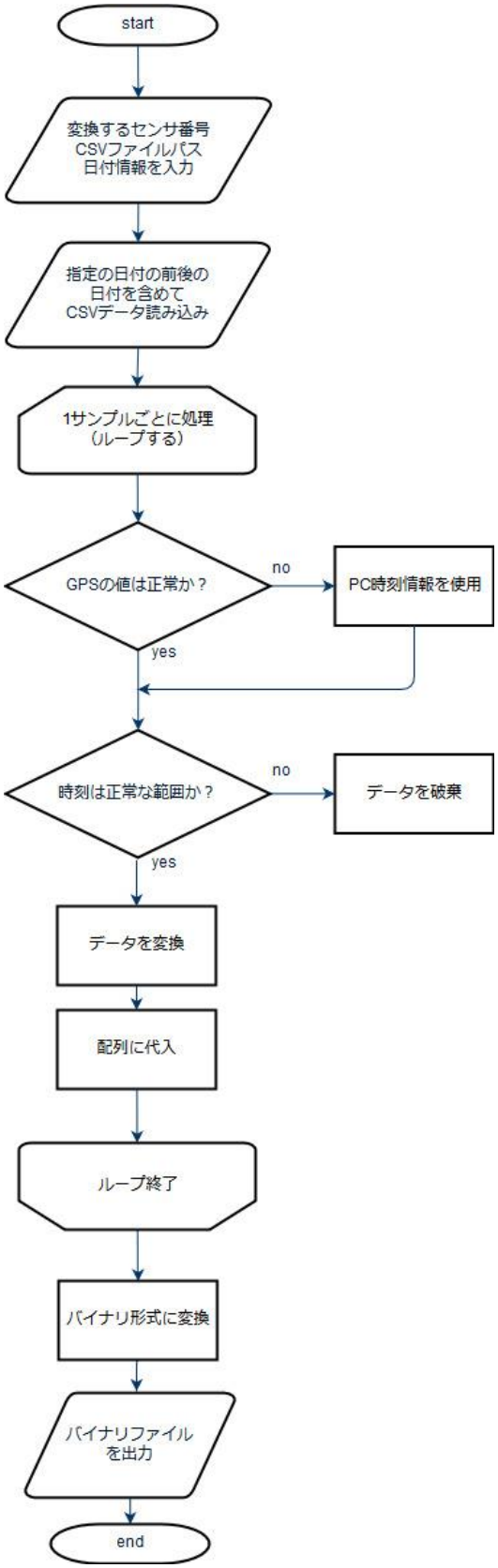


図 3.8 データアーカイブフローチャート

3.2.2　Infs データストア API

実際にユーザが解析を行う際に必要なデータを本研究サーバからダウンロードして使う場合を考えたとき、ユーザがプログラム上でできるだけ簡易に取得できるよう、API を作成した。ユーザは希望する時間、CSV 番号を JSON 形式にまとめ、特定の URL へ POST メソッドで送信することで希望する期間のバイナリデータが出力される仕組みとなっている。

http://192.168.255.10/infs/api/ が API を受け付ける URL になっており、その URL にむけて、以下のような JSON 文字列{"begin"："20190101","end"："20190103", "csvnum"："csv12"}を POST で送信するとサーバで動的処理が行われ、CSV12 番の 2019 年 1 月 1 日 0 時から 2019 年 1 月 3 日 23：59 までの infs 形式データを読み込み、結合処理を行い、単一 infs ファイルを生成、それが HTTP レスポンスで返される実装となっている。ユーザは、そのレスポンスボディをファイルに保存する様にコーディングすることで、infs ファイルを保存できる。

```
1   import json
2   import urllib.request
3
4   headers = {"Content-Type" : "application/json"}
5   url = "http://192.168.255.10/infs/api/"
6   dictdata = {"begin" : "20190101","end" : "20190103", "csvnum" : "csv12"}
7   json_data = json.dumps(dictdata).encode('utf-8')
8
9   request = urllib.request.Request(url, data=json_data, headers=headers, method="POST")
10  response = urllib.request.urlopen(request)
11
12  print("HTTPステータスコード" + str(response.getcode()))
13  data = response.read()
14  with open("DL.infs", mode="wb") as f:
15      f.write(data)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\Users\Daisuke\Documents> py.exe .\Untitled-1.py
HTTPステータスコード200
PS C:\Users\Daisuke\Documents> ls DL.infs


    ディレクトリ: C:\Users\Daisuke\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        2019/02/05      0:05       18819742 DL.infs
```

図 3.9　infs データストア API 利用例

3.3 災害を想定した分散型 Web アプリケーション

3.3.1　BGP を利用した冗長構成の実装

本研究では図 3.10 の様に VyOS と Web Server の構成を 1 拠点とみなし、冗長構成のために 3 地点分を用意した。これらの VyOS と本研究室の RTX1210 との間に BGP セッションを張り、AS65500 と AS65501 との間で eBGP を張っている状態にした。

AS(Autonomous System)とは BGP におけるネットワークの集合の単位で、AS 内では JPNIC(Japan Network Information Center)などから割り当てられた IP アドレスが運用される。例えば高知工科大学が使用している AS は AS55912 である。また 64512～65534 は Private AS 番号と呼ばれ、企業内など、閉じたネットワークや LAN で BGP を使う場合に使用される番号となる。本研究で使用する番号も Private AS である。BGP はルーティングするとき、パケットを次はどの AS におくればよいかを計算し、その AS に送るにはどのインターフェースで送信すればよいかを決定し、ルーティングする。評価にあたっては、RTX1210 を AS65500、VyOS 群を AS65501 として評価を行った。

AS65500 からはデフォルトゲートウェイを広報し、AS65501 からは 192.168.255.0 /24 を広報した。この状態では RTX1210 のルーティングテーブル上に 192.168.255.0 /24 の宛先が出現し、Web Server にアクセスが可能となるが、どの Web Server を優先させるか決めるため、MED (Multi exit discriminator)値を VyOS から広報する。MED 値は本来、同じ AS に複数経路で接続されるときに優先させる経路を決める値で、小さい値を持つ経路が優先される。本来は名前の通り同じネットワークに複数のパスがるときに用いるものである。しかしながら、本研究においては図 3.10 青枠部分の 192.168.255.0 /24 はそれぞれで独立しており、相互に接続されていない。本来の使い方に反するものの、この構成は正常に動作する。なぜならば Web Server は全く同じ構成になっており、クライアントからのアクセスに対して同じ回答を返すため、ユーザから見た場合、どの Web Server に接続されても問題はない。

拠点に障害が発生した、VyOS と RTX1210 間の通信が途絶した、などの理由で BGP セッションが切れると、RTX1210 のルーティングテーブル上から切れた拠点の 192.168.255.0 /24 のエントリが消えるため、自動的に障害拠点にパケットがルーティングされなくなる。そして残りの 2 拠点の 192.168.255.0 /24 エントリが残っているので、クライアントは別拠点のサーバに継続してアクセス行うことができる。

本研究では MED 値を A:100, B:50, C:0 とした。そのため拠点がすべて健在の場合の優先度は高い順に C→B→A となる。

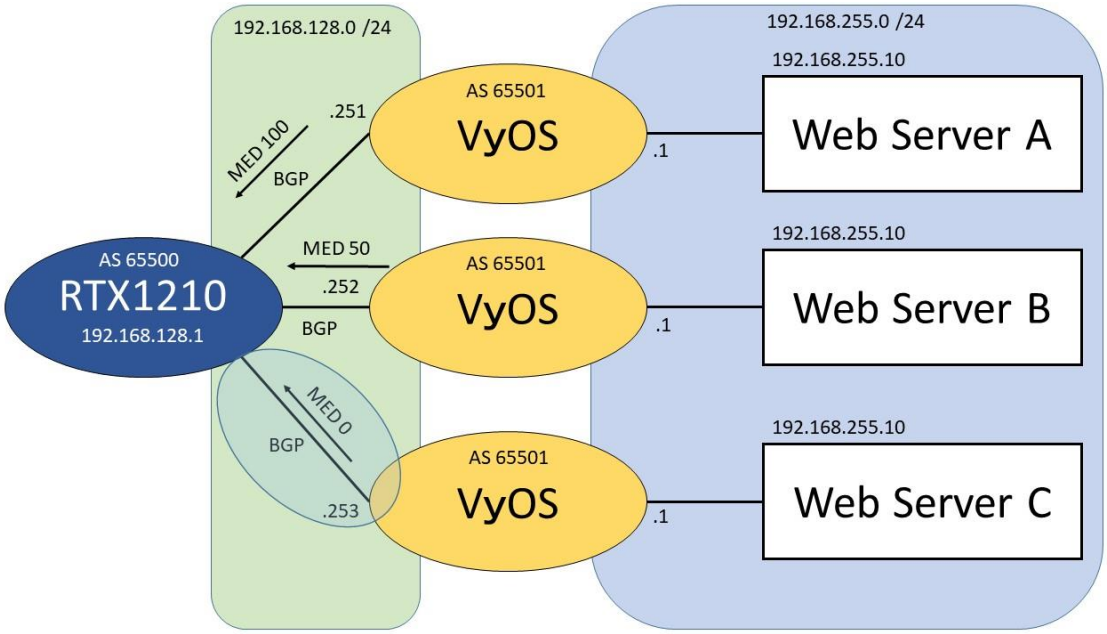


図 3.10 サーバ及びネットワーク構成

4 章 評価

4.1 負荷試験による性能評価

将来的に本研究の一部機能を公開する可能性を考慮し、アクセス負荷に対する耐性を計測する。計測は Apache Bench を用いて、センサー監視ページに対するアクセスを模擬し、評価を行った。

結果は図 4.1 の様になり、接続数に対して比例する形で待ち時間が増加している結果となった。

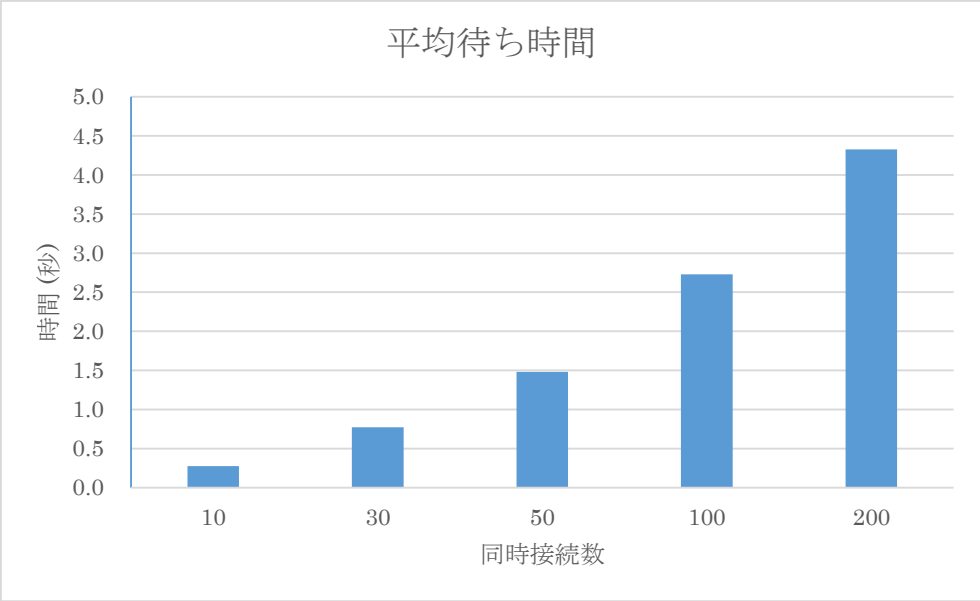


図 4.1 アクセス負荷に対する平均待ち時間

4.2 災害などにおける切断を想定した冗長性評価

本研究の Web サイトは 3 つのサーバで冗長化されているが、それらが正常に切り替わるか否かをテストする。障害は図 4.2 の様に C 拠点がダウンし、RTX1210 と該当拠点間の BGP セッションが切断され、経路が次に優先順位の高い拠点に切り替わるかを評価した。実際には図 4.2 の様に拠点 C の BGP を切断し評価を行った。

ICMP Ping を研究室内の PC から 192.168.255.10 宛に送信し続け、この状態で BGP セッションを 1 回切断する。この時の ICMP の状況を計測した。

また、センサー監視ページに対して、While 文で HTTP リクエストを送信し続ける Python スクリプトを作成し、BGP が切り替わった瞬間の状況を確認した。

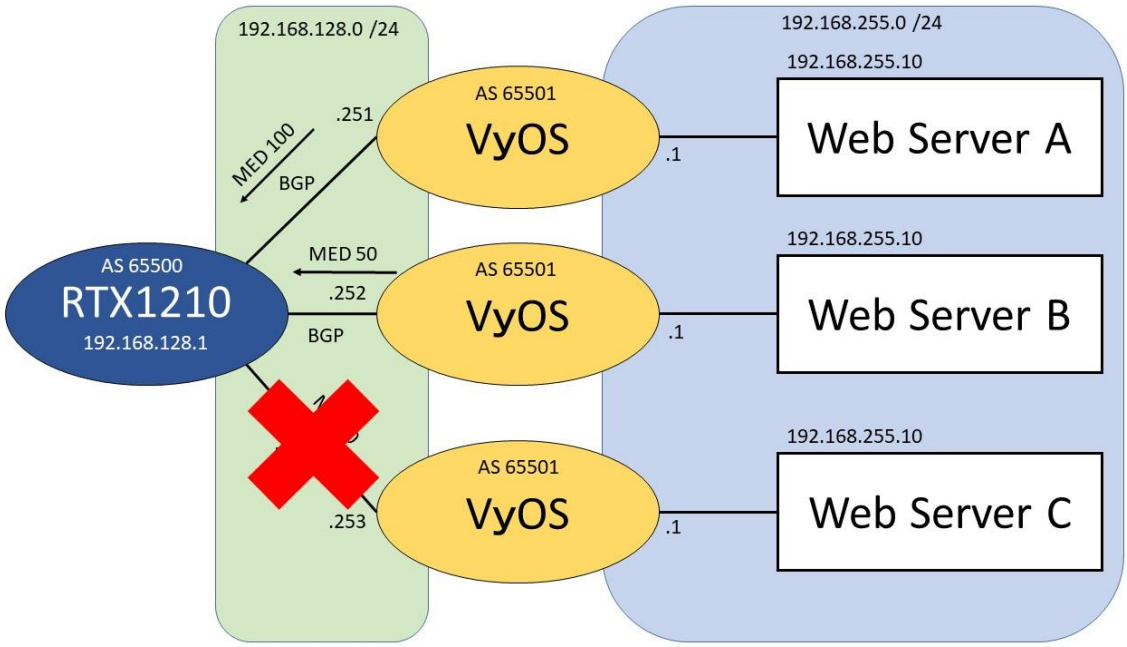


図 4.2 拠点 C の BGP を切断する

結果は以下のようになった。一度もパケットロスすることなく通信が正常に拠点 B に切り替わったことが確認された。また図 4.3、図 4.4 より RTX1210 のルーティングテーブルも正常に B 宛である 192.168.128.252 に切り替わっていることが確認できる。また仮に拠点 A や B で障害が先に発生した場合、外部からのアクセスは拠点 C で処理される。また今回は評価で実際に遠隔拠点に設置していないため、省略しているが各拠点のサーバには管理用としてアクセスできるユニークな IP アドレスが必要である。

```
PS C:¥Users¥Daisuke> ping 192.168.255.10 -t
192.168.255.10 に ping を送信しています 32 バイトのデータ:
192.168.255.10 からの応答: バイト数 =32 時間 <1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 <1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 からの応答: バイト数 =32 時間 =1ms TTL=63
192.168.255.10 の ping 統計:
パケット数: 送信 = 12、受信 = 12、損失 = 0 (0% の損失)
ラウンド トリップの概算時間 (ミリ秒):
最小 = 0ms、最大 = 1ms、平均 = 0ms
```

```
YamamotoLab-RTX1210> show ip route
宛先ネットワーク    ゲートウェイ     インタフェース  種別  付加情報
default             172.22.0.1      LAN3(DHCP)    static  filter:4001
default             222.229.66.1         LAN2     static
172.22.0.0/23       172.22.1.247         LAN3   implicit
192.168.128.0/24    192.168.128.200      LAN1   implicit
192.168.255.0/24    192.168.128.253      LAN1        BGP  path=65501
222.229.66.0/23     222.229.66.220       LAN2   implicit
```

図 4.3 正常時の RTX1210 のルーティングテーブル

図 4.4 拠点 C の BGP 切断時の RTX1210 上のルーティングテーブル

次に HTTP リクエストの結果だが、以下のように、成功すると HTTP ステータスコード 200 と時刻が出力されるテストコードである。BGP が切り替わったタイミングで一回だけ HTTP セッションがタイムアウトすることが確認できた。

PS C:¥Users¥Daisuke¥Documents> py.exe .¥test.py
200 2019-02-04 15:10:53.276179
200 2019-02-04 15:10:53.537227
200 2019-02-04 15:10:53.777453
200 2019-02-04 15:10:54.019959
200 2019-02-04 15:10:54.276318
200 2019-02-04 15:10:54.535443
200 2019-02-04 15:10:54.780852
200 2019-02-04 15:10:55.015166
timeout
200 2019-02-04 15:10:56.257244
200 2019-02-04 15:10:56.498268
200 2019-02-04 15:10:56.756502
200 2019-02-04 15:10:56.997168
200 2019-02-04 15:10:57.215518
200 2019-02-04 15:10:57.483544

5 章 結論

本研究開発では、基本的な Web アプリケーションによるインフラサウンドセンサーの監視、インフラサウンド観測データのアーカイブ化、またそれに伴う独自バイナリファイル形式 infs の開発を試み、基本部分の開発と周辺構築を終えた。

またユーザからのリクエストに応じるデータアーカイブも実装できた。しかしながら、日付単位でしか指定できない点や、結合に少々時間がかかるなど改善すべき点はまだ残っている。

BGP をベースとした冗長化構成は、評価上は問題なく動作に成功したが、本来は物理的に離れた拠点間で VPN などを併用して実行することが想定される。それらにも対応できるかは今後の課題としたい。

謝辞

本研究を進めるにあたり、指導教員として丁寧な指導をしてくださった山本真行教授、 副査を引き受けて下さった星野孝総准教授、小林弘和准教授に感謝いたします。

研究の方向性やインフラサウンドについてディスカッションして下さった研究室の皆様、またインフラサウンド観測網に協力してくださっている皆様に心から感謝申し上げます。

参考文献

[1] 田平誠, インフラサウンドの世界, http://www.senior.aichi-edu.ac.jp/mtahira/IFS/IFS_top.htm, 2018 年 12 月 参照.

[2] CTBTO Preparatory Commission, Infrasound monitoring, https://www.ctbto.org/verification-regime/monitoring-technologies-how-they-work/infrasound-monitoring/, 2019 年 2 月

[3] 山本真行, 国内外のインフラサウンド研究の現状, 月刊地球 '34, 554-559, 2012.

[4] 西山好則, 新方式インフラサウンドセンサの開発, 高知工科大学 平成 18 年度 卒業研究報告, 2007.

[5] 山田龍樹, インフラサウンドによる微小気圧変動の検出方式の検討と空振計の開発, 高知工科大学 平成 20 年度 卒業研究報告, 2009.

[6] IUGONET プロジェクト開発チーム, IUGONET メタデータ作成・登録の手引き, http://www.iugonet.org/data/manual/IUGONET_metadata_manual_v2_20170418.pdf, 2019 年 2 月 参照.

[7] IETF , A Border Gateway Protocol 4 (BGP-4), https://tools.ietf.org/html/rfc4271 , 2019 年 2 月 参照.

[8] Chris Evans, vsftpd , https://security.appspot.com/vsftpd.html , 2019 年 2 月 参照.

[9] Microsoft, Microsoft Hyper-V Server 2016, https://docs.microsoft.com/ja-jp/windows-server/virtualization/hyper-v/hyper-v-server-2016 , 2019 年 2 月 参照.

[10] Canonical Ltd., Ubuntu, https://www.ubuntu.com/, 2019 年 2 月 参照.

[11] YAMAHA, フィルタ型ルーティング, http://www.rtpro.yamaha.co.jp/RT/docs/filter-routing/filter-routing.html, 2019 年 2 月参照.

[12] Microsoft, robocopy, https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy, 2019 年 2 月 参照.

[13] Lawrence Journal-World, Django, https://www.djangoproject.com/, 2019 年 2 月 参照.

[14] MariaDB Corporation Ab, MariaDB, https://mariadb.com/, 2019 年 2 月参照.