学位論文

学内の Elasticsearch システムのデータ移行と
冗長化

提出年月日　令和 4 年 X 月 X 日
改定日　　令和 年 月 日

指導教員　　都築　伸二　教授

入学年度　　令和 6 年
学科名　　電子情報工学専攻
論文提出者　　祖父江　匠真

内容梗概

本論文は，筆者が愛媛大学大学院理工学研究科電子情報工学専攻電気電子工学コースに在学中に行った、学内の Elasticsearch のデータ移行と冗長化についてまとめたものであり、以下の 5 章から構成されている。

第 1 章 緒論

本研究を行うに至った経緯及び、本研究の目的について述べている。

第 2 章 学内ゾーンにおける Elasticsearch クラスタへのデータ移行

ここでは学内ゾーンにおける Elasticsearch クラスタへのデータ移行について述べる。

第 3 章 サーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証

ここでは、サーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証について述べている。

第 4 章 サーバーゾーンでのクラスタ構築

ここでは、サーバーゾーンでのクラスタ構築について述べている。

第 5 章 結論

本研究によって明らかになった事項や今後の研究課題について簡単にまとめている。

目 次

第1章

緒論

急速に進化するデータ管理において、Elasticsearch は極めて重要な技術として登場し、データの保存、検索、管理方法に革命をもたらした. 当初、大量のデータを効率的に処理するために設計された Elasticsearch は、シンプルなシングルノードシステムから複雑なクラスタ構成へと移行し、データハンドリング技術において大きな進歩を遂げた. この変遷は、様々な分野で堅牢でスケーラブル、かつ冗長性のあるデータ管理システムに対する要求が高まっていることを裏付けている.

データシステムの冗長性、特に Elasticsearch クラスタにおける冗長性は、データの信頼性と可用性を確保する上で重要な要素となっている. 冗長性とは、システムの重要なコンポーネントや機能を二重化することで、信頼性を高め、一点障害のリスクを低減することを指す.

Elasticsearch のノード管理における現在のトレンドは、クラスタ化されたシステムを好む傾向が強まっている. しかし、Elasticsearch の技術的な側面については多くの研究があるが、同じ環境内でシングルノードシステムからクラスタ化されたシステムへデータを移行し、さらに別のネットワークゾーンに新しくクラスタ化されたシステムを構築する際の実際的な課題や戦略について掘り下げた研究はほとんどない.

本研究の目的は、シングルノードの Elasticsearch システムから学内ゾーン内のクラスタ化システムへデータを移行するプロセスを分析することである. 同時に本研究では、この 2 つのゾーン間のデータ移行を行わずに、サーバゾー

ンに新しい冗長化されたクラスタ化システムを構築する.

第 2 章では学内ゾーンにおける Elasticsearch ノードから新クラスタへのデータ移行について述べる. 第 3 章では仮想環境を使用してサーバーゾーンの Elasticsearch ノードをシミュレートし、マルチコンテナ Docker アプリケーションのツールである docker-compose を用いてクラスタ構築の実現可能性を検証したことについて述べる. 第 4 章ではサーバーゾーンで既存の Elasticsearch ノードを用いたクラスタ構築について述べる. 第 5 章では結論と課題を述べる.

第2章

2章のタイトル

2.1節　緒言

本章では学内ゾーンにおける Elasticsearch クラスタへのデータ移行について述べる.

2.2節　概要

今回は, ElasticSearch サーバー間でのデータ移行と, その際に行われた重複データの削除方法, kibana による可視化結果について報告する.

2.3節　データ移行手順について

co2 のデータ移行を行う上で, タイムスタンプと部屋番号の組み合わせが重複しているデータが一部存在しており, この重複データを取り除いた上でデータ移行を行う必要があったので, 一度, 移行元の ElasticSearch サーバーのデータをローカルマシンにエクスポートして, 重複データを取り除いた上で, 移行先の ElasticSearch サーバーにデータをアップロードした.

2.3.1 データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには, elasticdump ライブラリを使用して, JSON 形式でエクスポートした. その際, co2 という文字列を含むインデックスのデータのみをエクスポートした.

2.3.2 データの重複削除

重複データの削除は SQLite データベースを用いて行った.

SQLite データベースはリレーショナルデータベースの一種であり, 複合主キーを使って複数のテーブルカラムの組み合わせを一意の識別子として扱うことができる. これにより, 同じ組み合わせのデータを重複して挿入しようとした場合, データベースエンジンがコンフリクトエラーを発生させ, 重複データの挿入を阻止する. そのため, 今回の重複データ削除には適していると判断した.

今回使用した SQLite データベースでは, 部屋番号（number）とタイムスタンプ（JPtime）を一意のキーとして設定した. 以下のリスト 2.1, リスト 2.2 に示すように, 移行元の ElasticSearch サーバーに保存されている co2 インデックスのドキュメントは, フィールドのメンバーが統一されておらず, 一部センサー情報が存在しない場合がある. そのため, データの挿入時にコンフリクトエラーが発生した場合は, 既存のレコードと挿入しようとしたレコードを比較し, 既存レコードの値が NULL であるカラムにおいて, 挿入しようとしているレコードの値が非 NULL である場合には, 既存レコードのカラムの値を更新するようにした. これにより, 重複データ削除時に一部センサー情報などが欠けてしまう問題を解決した.

Listing 2.1　_source フィールドのメンバー数が少ないドキュメント

```
{
  "_index": "co2_e411",
  "_type": "_doc",
  "_id": "nEi2nnoB2-iFXnrMOobM",
```

```
  "_score": 1,
  "_source": {
      "utctime": "2020-10-09T05:09:06+00:00",
      "number": "E411",
      "PPM": "481",
      "data": "Thingspeak"
  }
}
```


Listing 2.2　_sourceフィールドのメンバー数が多いドキュメント


```
{
  "_index": "co2_e411",
  "_type": "_doc",
  "_id": "YKBqU4QBugDzeydA2gyi",
  "_score": 1,
  "_source": {
      "RH": 26.98,
      "PPM": 423,
      "JPtime": "2022-11-06T22:45:30.080925",
      "ip": "172.23.68.19/16",
      "utctime": "2022-11-06T13:45:30.080895",
      "TEMP": 24.47,
      "index_name": "co2_e411",
      "ms": "",
      "number": "E411"
  }
}
```

2.3.3　データのインポート

重複データ削除後のデータが保存された SQLite テーブルからすべてのレコードを読み出して, ターゲットの ElasticSearch サーバーに移行した.

その際, python の elasticsearch ライブラリを使用し, タイムスタンプが 2023 年より以前のデータは 2022_co2 という名前のインデックスに保存し, 2023 年のものは 2023_co2 という名前のインデックスに保存した.

2.4 節　kibana によるデータの可視化

移行後のデータを kibana を用いて可視化した.

2022_co2 インデックスと, 2023_co2 インデックスについて, 横軸をタイムスタンプとし, 縦軸を PPM, RH, TEMP としてそれぞれプロットしたものを図 2.1 〜 図 2.6 に示す.

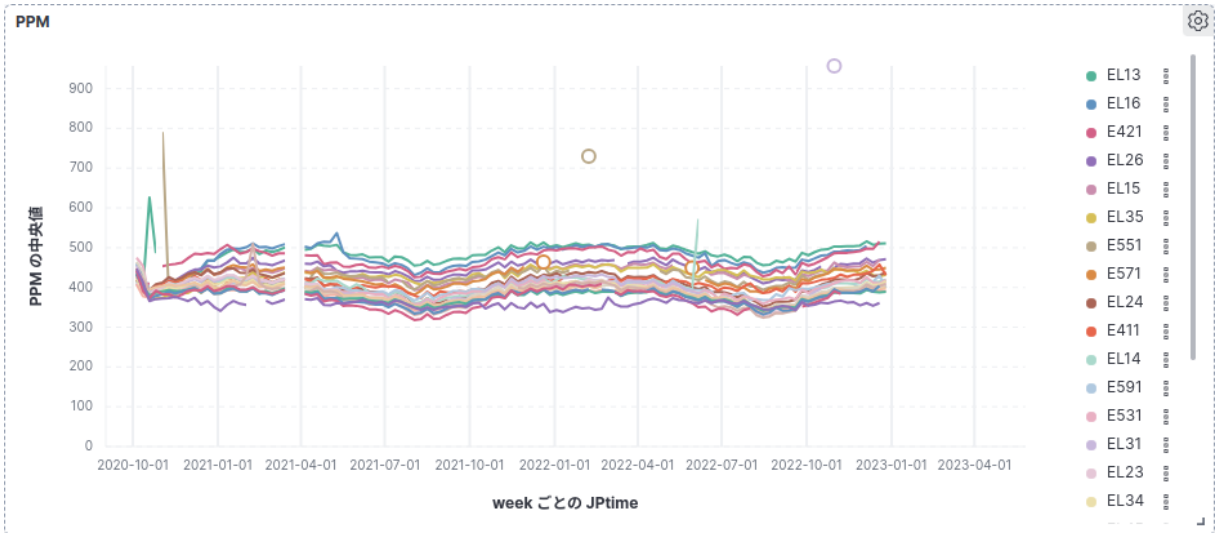


図 2.1　2022_co2 の PPM

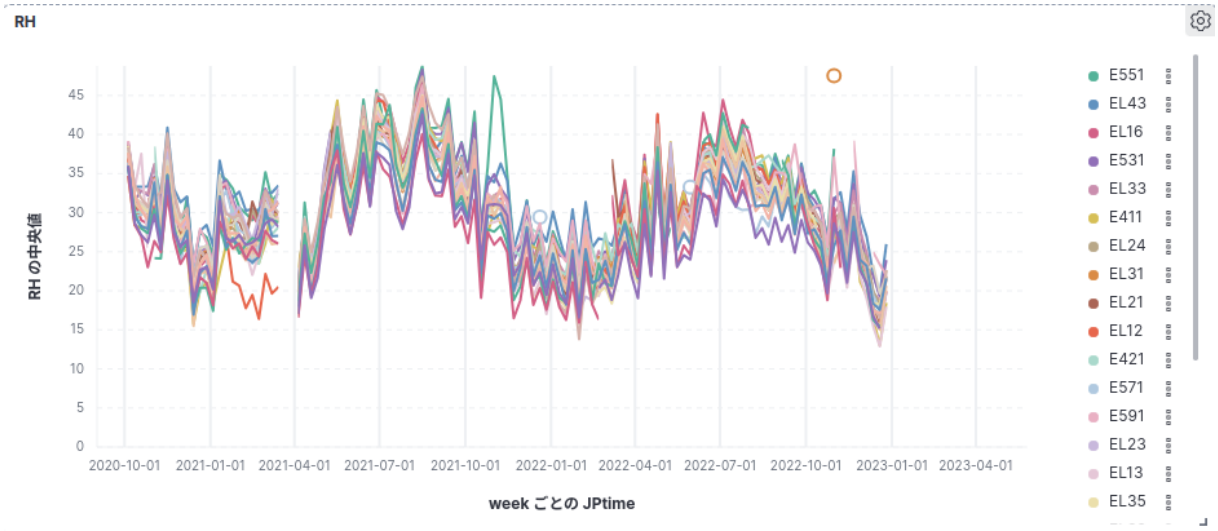


図 2.2　2022_co2 の RH

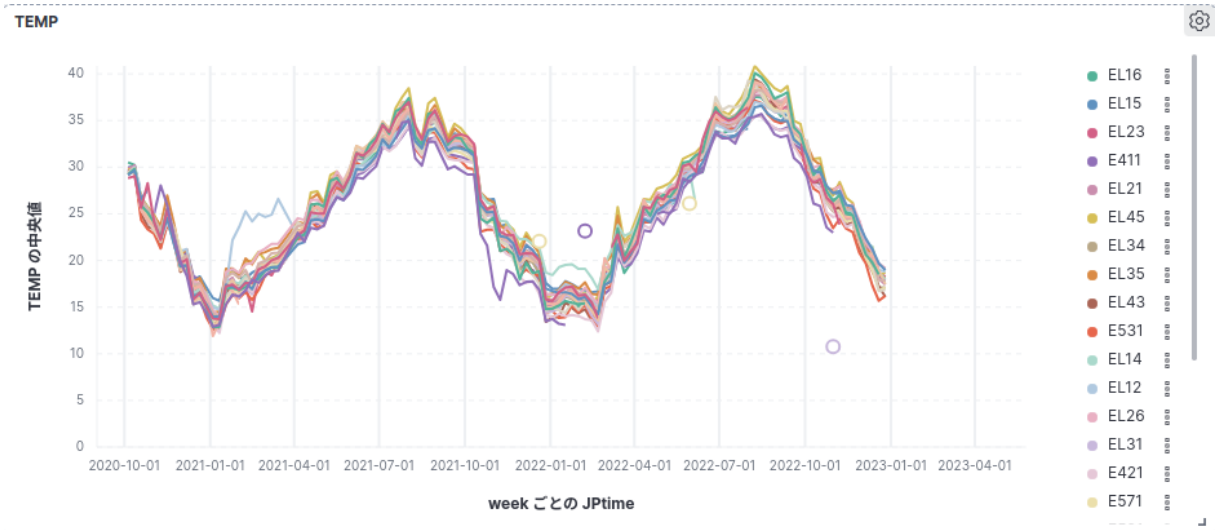


図 2.3　2022_co2 の TEMP

2.5 節　概要

今回は, 前回に引き続き 133.71.106.168 から 133.71.106.141 への ElasticSearch サーバー間のデータ移行と kibana を用いた可視化結果について報告する．ま

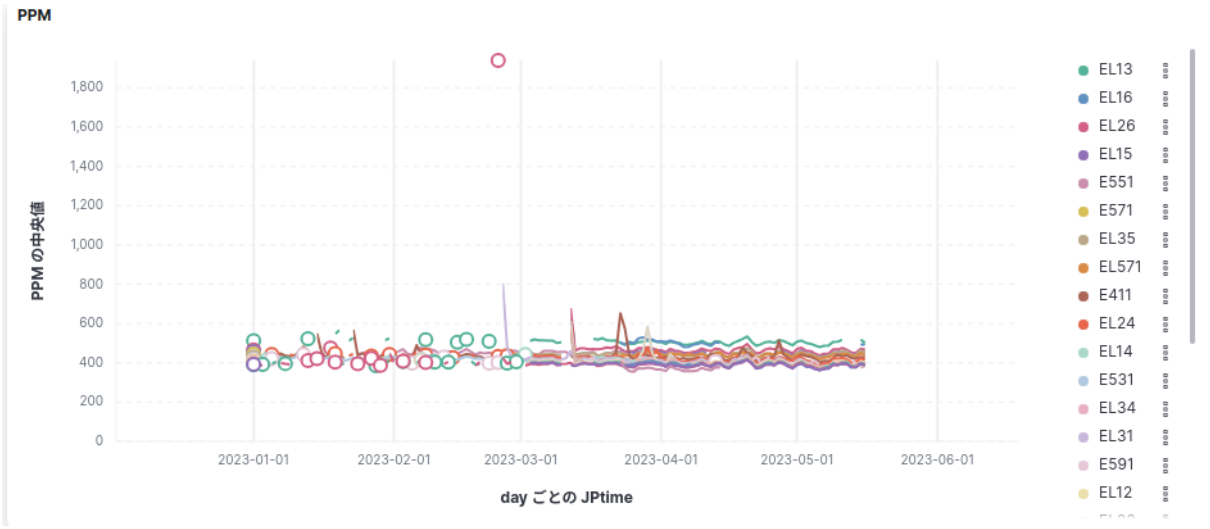


図 2.4　2023_co2 の PPM

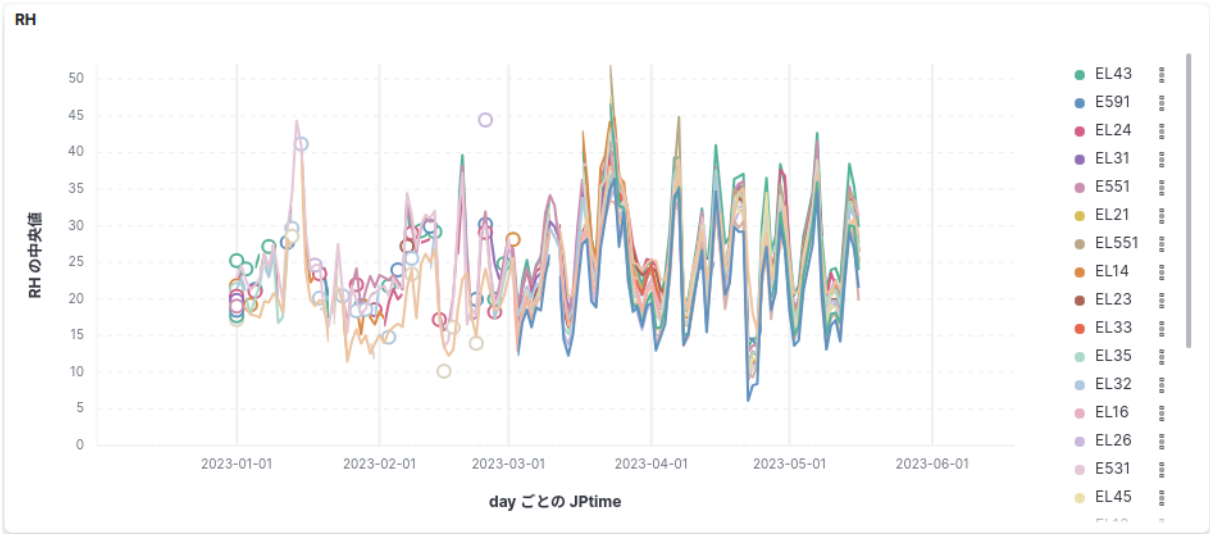


図 2.5　2023_co2 の RH

た, 移行元 ElasticSearch サーバーにあった CO_2 データ以外のデータの移行についても報告する.

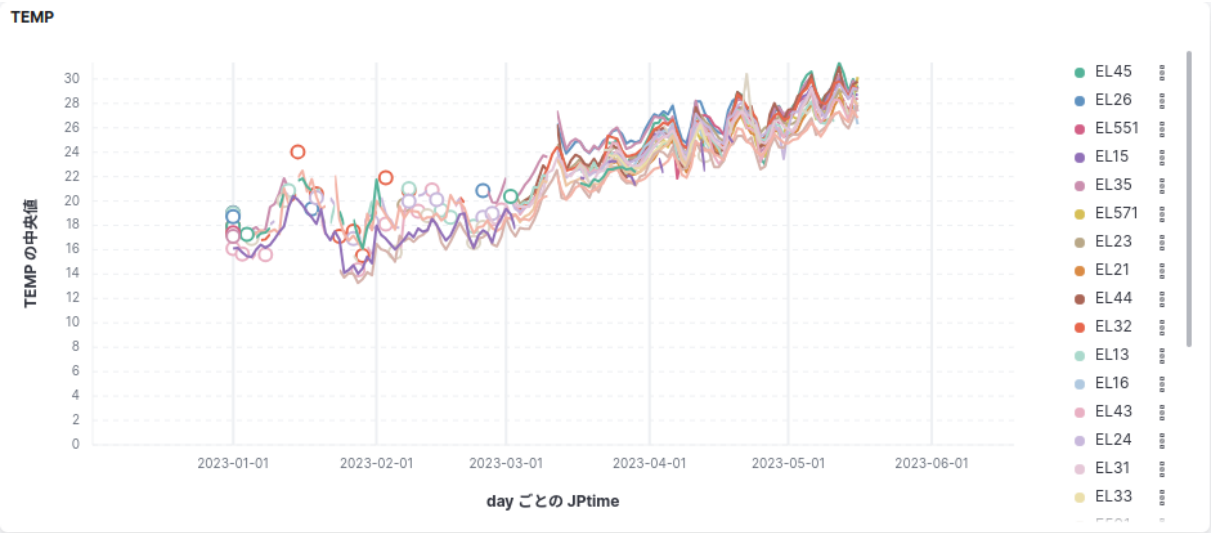


図 2.6　2023_co2 の TEMP

2.6 節　CO_2 データの移行作業

前回の報告書では, 移行元 ElasticSearch サーバーの CO_2 データについて, JPtime が 2023 年より以前のドキュメントを 2022_co2 という名前のインデックスに保存し, JPtime が 2023 年のドキュメントを 2023_co2 という名前のインデックスに保存した. しかし, インデックスを分けることで 2023 年以前と 2023 年のデータを kibana で同じグラフにプロットして確認することが難しい可能性があることと, ElasticSearch サーバー間のデータ移行が正しく完了したか kibana で可視化することによって確認することが出来ないという問題があったため, 今回は JPtime の値に関わらず co2 という名前のインデックスに保存した. 保存先のインデックスが co2 であることを除くと, データの移行手順は前回のものと同様の手順で行った.

2.7 節　kibana によるデータの可視化

移行後の co2 インデックスに保存されたデータを kibana を用いて可視化した.

横軸をタイムスタンプとし, 縦軸を PPM, RH, TEMP としてそれぞれプロットしたものを図 2.7 〜 図 2.9 に示す.

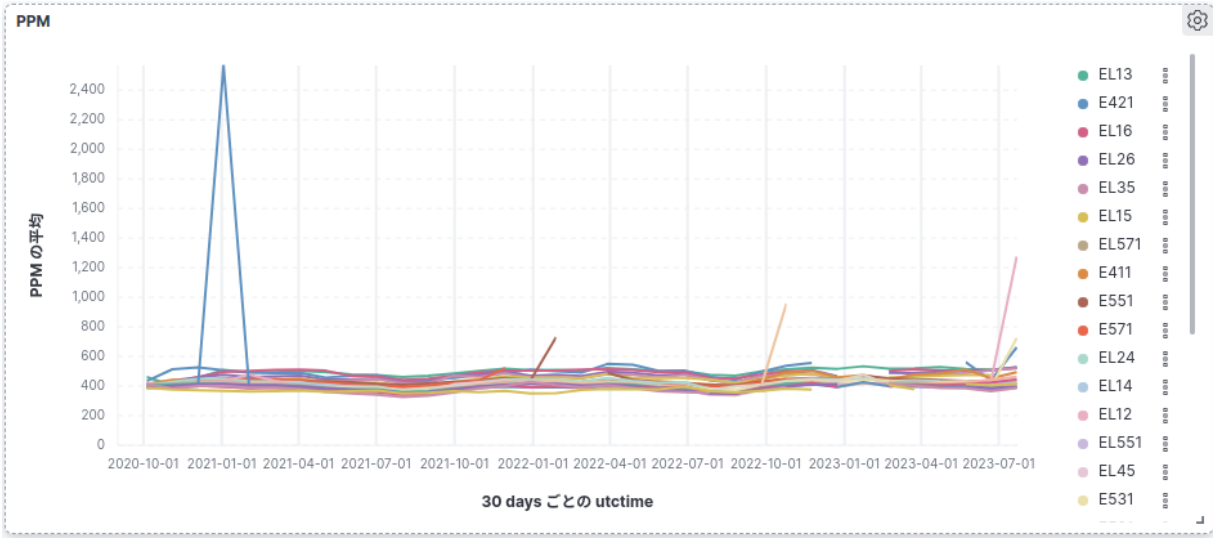


図 2.7　co2 の PPM

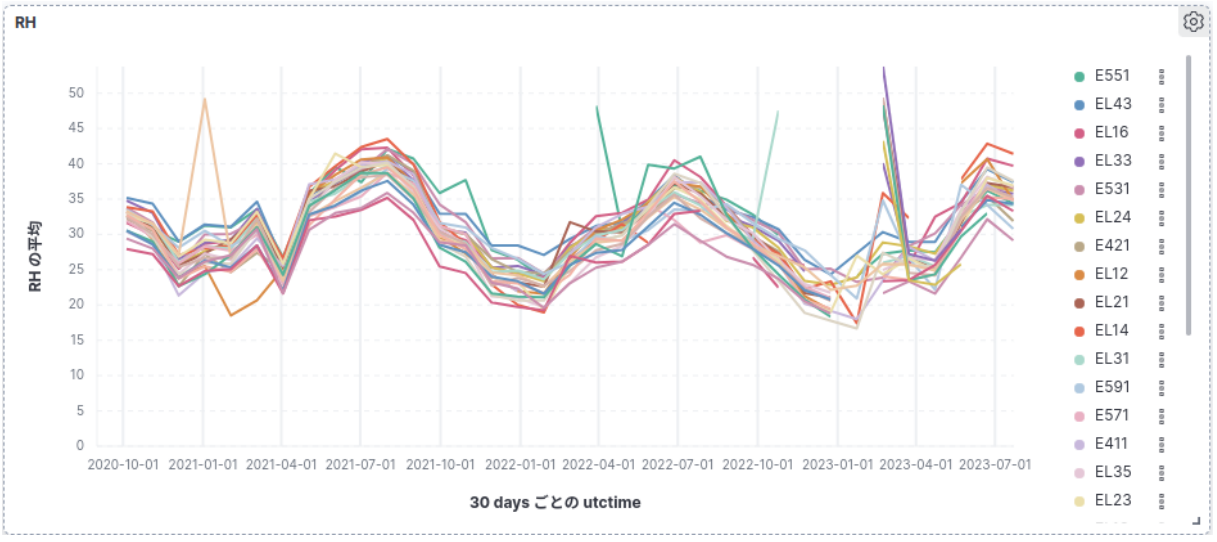


図 2.8　co2 の RH

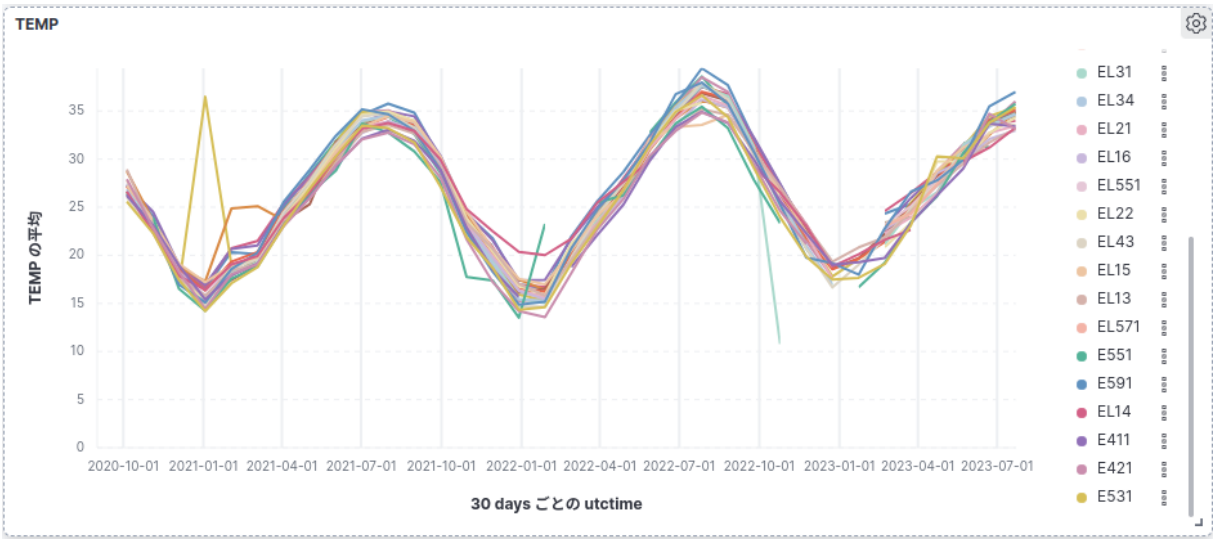


図 2.9　co2 の TEMP

図 2.7 ～ 図 2.9 について, 2022 年と 2023 年の境目でデータが連続的に変化していることが確認出来るので, データ移行作業は正常に行うことが出来たと判断できる.

2.8 節　CO_2 データ以外のデータの移行について

移行元の ElasticSearch サーバーにはインデックス名に co2 という文字列を含むインデックス以外に以下のインデックスがあった.

- movement_diary
- movement_diary01
- temp2
- temp3
- test

これらのインデックスのデータ移行は, 同名のインデックスを移行先の ElasticSearch サーバーに作成して, 作成したインデックスにデータを挿入することで行った.

ただし, test インデックスは移行先の ElasticSearch サーバーに既に同名のインデックスが作成されていたので, 133.71.106.168_test というインデックス名にした.

次に, 上記のインデックスに保存されているデータについて説明する.

temp2, temp3 は TIME, TEMP, HUMI フィールドを持ったドキュメントが格納されており, temp2 のドキュメント数は約 150 件, temp3 は約 60 件であった. test は JPtime, PPM, RH, TEMP, ip, number, utctime フィールドを持ったドキュメントを格納しているインデックスであり, ドキュメント数は 95 件であった. これらの情報から, temp2, temp3, test インデックスは CO_2 データの収集の研究の中で動作確認目的に作成されたインデックスではないかと考えられる.

以下に movement_diary と movement_diary01 のドキュメントの違いを列挙する.

1. driver フィールド:
   - movement_diary のドキュメントでは, driver フィールドは文字列である.
   - movement_diary01 のドキュメントでは, driver フィールドは配列で, その中に文字列と 2 つの null 値が含まれている.
2. “destination” フィールド:
   - movement_diary のドキュメントでは, “destination” フィールドは単一の文字列である.
   - movement_diary01 のドキュメントでは, “destination” フィールドは配列で, その中に 2 つの文字列が含まれている.
3. “charge_place” フィールド:
   - movement_diary のドキュメントには, “charge_place” フィールドは存在しない.
   - movement_diary01 のドキュメントでは, “charge_place” フィールドが追加されているが, その値は空文字列である.
4. “battery_rate” フィールド:

- movement_diary のドキュメントには, “battery_rate” フィールドは存在しない.
- movement_diary01 のドキュメントでは, “battery_rate” フィールドが追加されており, その値は数値である.

5. “battery_rate_distance” フィールド:

- movement_diary のドキュメントには, “battery_rate_distance” フィールドは存在しない.
- movement_diary01 のドキュメントでは, “battery_rate_distance” フィールドが追加されており, その値は数値である.

2.9 節　概要

今回は, 133.71.201.197 から 133.71.106.141 への ElasticSearch サーバー間のリサイクル館の太陽光パネルの測定データ移行と kibana を用いた可視化結果について報告する. また, 133.71.106.168 の ElasticSearch サーバーにあった movement_diary インデックスと movement_diary01 インデックスのドキュメントについて調査した結果も報告する.

2.10 節　データ移行手順について

データ移行を行う上で, ローカルマシンに JSON 形式でダンプしていたデータと, 133.71.201.197 の ElasticSearch サーバー上に存在するデータとの間で重複しているデータが一部存在しており, この重複データを取り除いた上でデータ移行を行う必要があった.

そこで一度, 移行元の ElasticSearch サーバーのデータをローカルマシンにエクスポートして, 重複データを取り除いた上で, 移行先の ElasticSearch サーバーにデータをアップロードした.

2.10.1　データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには, elasticdump ライブラリを使用して JSON 形式でエクスポートした. その際, pcs_recyclekan という名前のインデックスのデータをエクスポートした.

2.10.2　データの重複削除

重複データの削除は, 予めローカルマシンに JSON 形式でダンプしていたデータと, elasticdump ライブラリを使用して JSON 形式でエクスポートしたデータを, utctime フィールドの値がユニークになるようにフィルタリングすることで行った.

2.10.3　データのインポート

重複データ削除後のデータを保存した JSON ファイルを読み出して, 移行先の ElasticSearch サーバーにアップロードした.

その際, python の elasticsearch ライブラリを使用し,133.71.201.197 の ElasticSearch サーバーと同名の pcs_recyclekan という名前のインデックスに保存した.

2.11 節　kibana によるデータの可視化

移行後の pcs_recyclekan インデックスに保存されたデータを kibana を用いて可視化した.

次に, 移行元である 133.71.201.197 の ElasticSearch サーバーの pcs_recyclekan インデックスに保存されたデータを図 2.2 に示す.

図 2.10, 図 2.11 について, 2022 年 8 月以降のグラフの概形が一致していることが確認出来るので, データ移行作業は正常に行うことが出来たと判断できる.

図 2.10　133.71.106.141 の pcs_recyclekan

図 2.11　133.71.201.197 の pcs_recyclekan

2.12 節　movement_diary のデータについて

movement_diary と movement_diary01 はデータ型が異なる一部のフィールドを除いて全て同じデータを保有しており, それぞれのインデックスのドキュ

メント数の比較と, タイムスタンプ情報を格納するフィールドのインデックス間での比較を行うことで, movement_diary が不要なインデックスであるかを調査した.

まず, movement_diary と movement_diary01 のドキュメント数を調べたところ, 同じ 142 件であった.

次に, movement_diary と movement_diary01 でタイムスタンプ情報を持つフィールドである dt_S フィールドの値同士を比較した.

ただし, dt_S フィールドが null のドキュメントが一部存在するので, その場合はタイムスタンプ情報を持つ inspection フィールドの値同士を比較した.

比較した結果, dt_S フィールドと inspection フィールドの両方が null である 3 件のドキュメントを除いて他全てのドキュメントは dt_S フィールドもしくは inspection フィールドの値が movement_diary と movement_diary01 の間で一致した.

dt_S フィールドと inspection フィールドの両方が null だった 3 件のドキュメントについても, 全てのフィールドにおいて, movement_diary01 のドキュメントが movement_diary のドキュメントの持つ情報を持っていたので, これらの調査結果から movement_diary インデックスは movement_diary01 インデックスで代替でき, 削除して良いインデックスであると判断した.

2.13 節　概要

今回は, 133.71.201.197 から 133.71.106.141 への ElasticSearch サーバー間の CO_2 データの移行後に発生したラズベリーパイからデータのインサートが出来ない問題への対応と, 移行元 ElasticSearch サーバーから移行されていない CO_2 データの移行について報告する.

2.14 節　ラズベリーパイからデータのインサートが出来ない問題について

私が実装したデータ移行プログラムを使用して作成した ElasticSearch のインデックスに対してラズベリーパイからデータのインサートが出来ない問題が発生した.

そこで, インデックスの作成を私のデータ移行プログラム上からではなく, 高木君側で行ってもらい, ラズベリーパイから正常にデータのインサートが出来ていることを確認した上で, 私が実装したデータ移行プログラムを使用して CO_2 データの移行を行うことで問題を解決した.

2.15 節　移行元 ElasticSearch サーバーから移行されていない CO_2 データの移行について

私が実装したデータ移行プログラムを使用して 133.71.201.197 から 133.71.106.141 の ElasticSearch サーバーへ CO_2 データを移行したのが 2023 年 5 月中旬頃であり, 高木君が, 移行先である 133.71.106.141 の ElasticSearch サーバーに対してラズベリーパイから CO_2 データのインサートを行うよう対応したのが 2023 年 7 月中旬であったため, 2023 年 5 月中旬から 2023 年 7 月中旬までの間の約 2ヶ月間の CO_2 データが移行先の ElasticSearch サーバーに移行出来ていなかった. そこで, 追加の移行作業を行った.

移行方法は以下のとおりである.

1. まず, 2023 年 5 月中旬に移行した際の全移行データの中で最も最新の utctime フィールドの値を検索する.
    - 検索した結果, 2023 年 5 月中旬に移行した際の全移行データの中で最も最新の utctime は「2023-05-16T05:48:30.081305」であった.
2. 次に, 移行先 ElasticSearch サーバーに対してラズベリーパイからインサートされた全データの中で最も古い utctime フィールドの値を検索する.

- 検索した結果, ラズベリーパイからインサートされた全データの中で最も古い utctime は「2023-07-20T07:15:39.314008」であった.

3. 前回の CO_2 データの移行は 2023 年 5 月中旬頃に行ったため, 2023 年 5 月 1 日 0 時 0 分 0 秒以降の utctime を持つドキュメントを, 移行元 ElasticSearch サーバーのインデックス名に co2 という文字列を含むインデックスから elasticdump [?] ライブラリを使用してローカルマシンにエクスポートする.
4. 部屋番号（number）とタイムスタンプ（JPtime）の組み合わせがユニークになるようにエクスポートしたデータをフィルタリングする.
5. 更に, 1 と 2 で得られた utctime の範囲に含まれる utctime を持つドキュメントのみになるようフィルタリングする.
6. フィルタリング後のデータを移行先 ElasticSearch サーバーにバルクインサートする.

2.16 節　kibana によるデータの可視化

2023 年 5 月中旬から 2023 年 7 月中旬までの間の約 2ヶ月間の CO_2 データを移行した後の co2_modbus インデックスについて, 横軸をタイムスタンプ (utctime) とし, 縦軸を PPM, RH, TEMP としてそれぞれプロットしたものを図 2.12 〜 図 2.14 に示す.

今回, 追加で CO_2 データを移行した 2023 年 5 月中旬から 2023 年 7 月中旬までの期間とその前後の期間において, 図 2.12 〜 図 2.14 より, 連続的にデータが変化していることが目視で確認できるので, データ移行は正常に出来たと判断できる.

2.17 節　概要

今回は, 133.71.201.197 の ElasticSearch サーバーにある pcs_recyclekan という名前のインデックス以外のインデックスについて調査を行い, 133.71.106.141 の ElasticSearch サーバーにデータ移行を行ったことについて報告する.

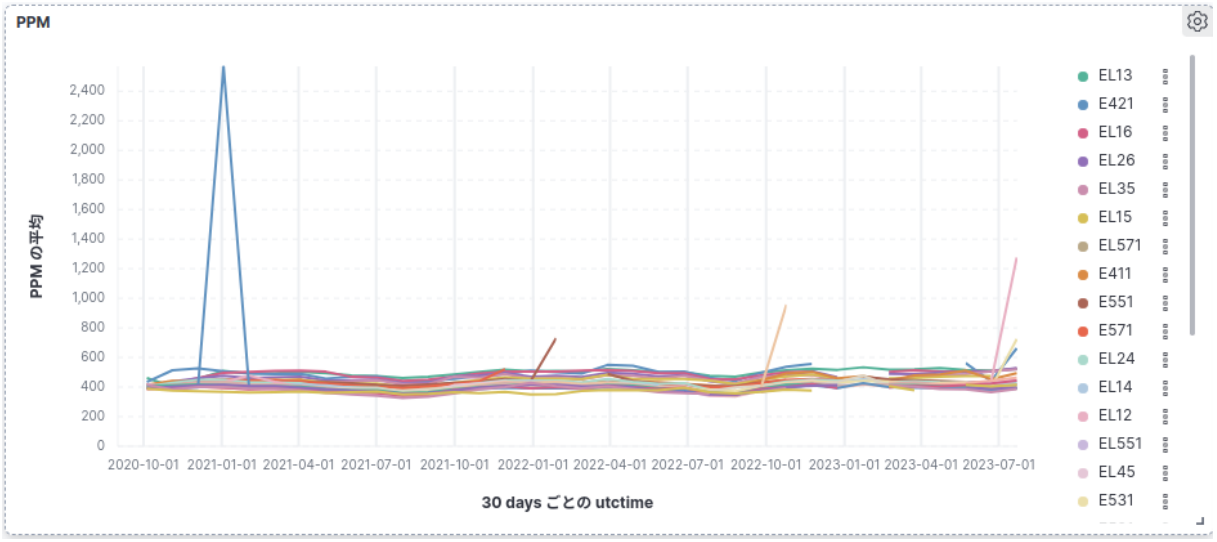


図 2.12　co2_modbus の PPM

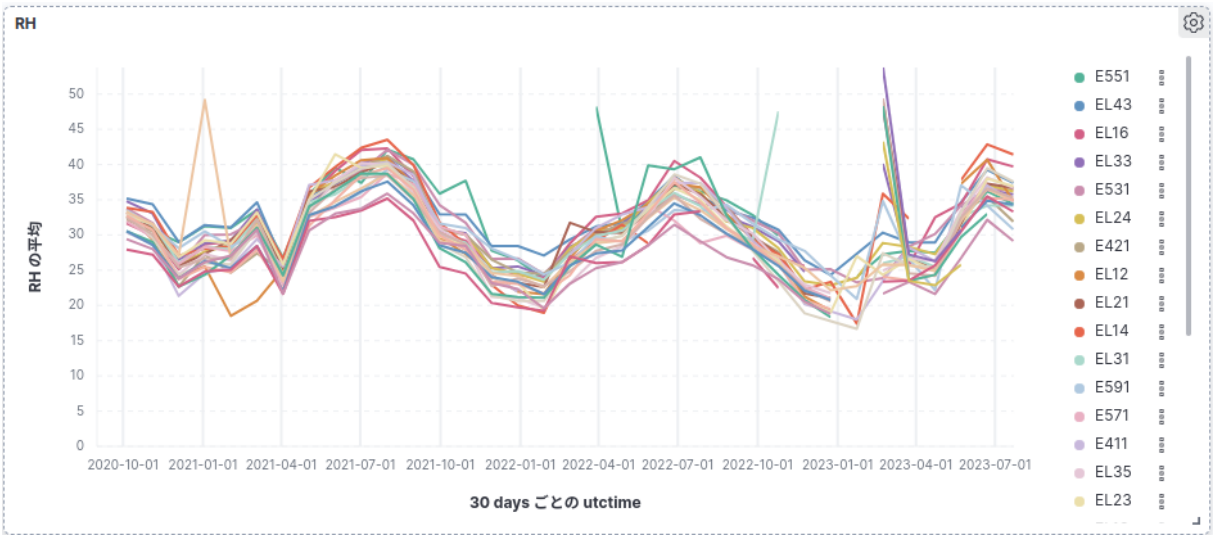


図 2.13　co2_modbus の RH

2.18 節　133.71.201.197 の ElasticSearch サーバーにあるインデックスについて

図 2.15 に 133.71.201.197 の ElasticSearch サーバーにあるインデックスの一覧を示す.

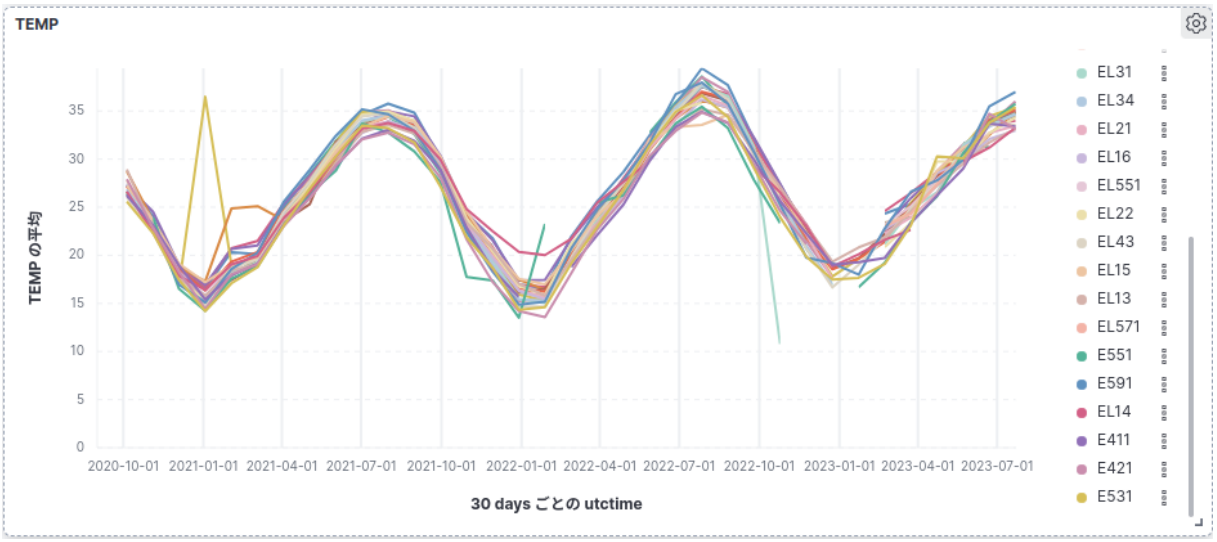


図 2.14　co2_modbus の TEMP

これらのインデックスが保存しているデータについて説明する.

- pcs_test
  - 恵村君がプログラムの検証目的で使用しているインデックス
- pcs_recyclekan
  - リサイクル館の太陽光発電に関するデータを保存しているインデックス
- leaf_load
  - leaf のデータが保存されているインデックス
- pcs_log
  - リサイクル館の太陽光発電に関するデータを ElasticSearch にインサートする Python プログラムのログ情報を保存しているインデックス
- pcs_test4
  - 恵村君がプログラムの検証目的で使用しているインデックス

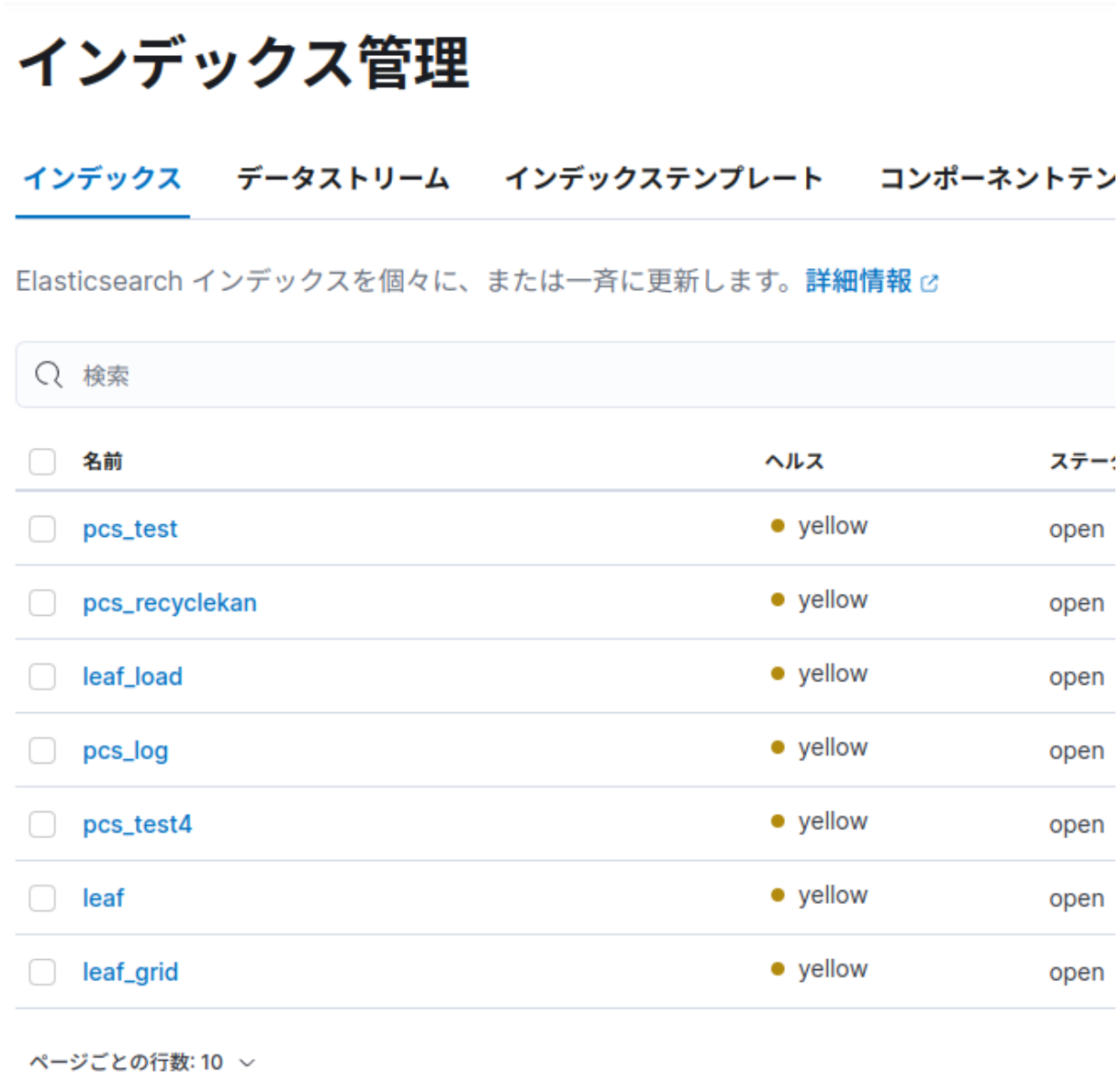


図 2.15　133.71.201.197 の ElasticSearch サーバーにあるインデックスの一覧

- leaf
  - leaf のデータが保存されているインデックス

- leaf_grid
  - leaf のデータが保存されているインデックス

なお, leaf, leaf_grid, leaf_load インデックスについて, Kibana で各インデックスのマッピング情報を確認したところ, 3 つともすべて同じマッピング情報を保持しており, 同じフィールドを持つドキュメントをそれぞれのインデックスで保存していることが分かった.

pcs_test インデックスと pcs_test4 インデックスに関しては, 恵村君が検証用途で使用しているものであるため, 今回の移行対象からは除外し, pcs_log, leaf, leaf_load, leaf_grid インデックスのみを移行対象とした.

2.19 節　データ移行手順について

データ移行手順について, まず移行元の ElasticSearch サーバーのデータをローカルマシンに JSON 形式でエクスポートして, 作成した Python プログラムを実行して移行先の ElasticSearch サーバーにデータをインサートした.

2.19.1　データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには, elasticdump [?] ライブラリを使用して JSON 形式でエクスポートした. その際, pcs_log, leaf, leaf_load, leaf_grid という名前のインデックスのデータをエクスポートした.

2.19.2　データのインポート

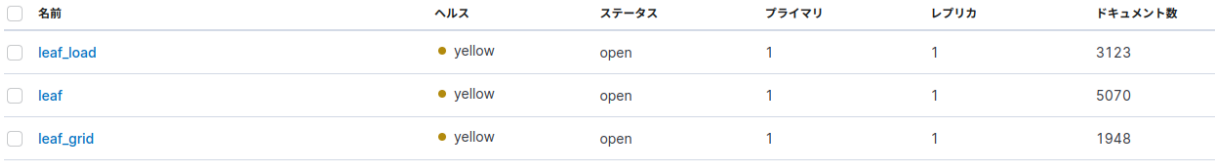
エクスポートした JSON ファイルを, 作成した Python プログラムから読み込んで, Python の elasticsearch ライブラリを用いて移行先の ElasticSearch サーバーにインサートした.

移行先の ElasticSearch サーバーにおけるインデックス名については, 133.71.201.197 の ElasticSearch サーバーと同名のインデックスに保存した.

2.20 節　データ移行が正常に行えたか確認

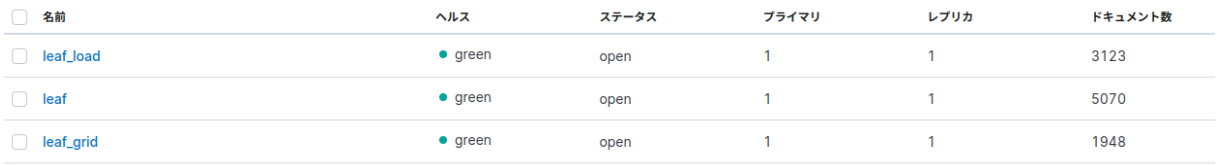
図 2.16 に移行元の ElasticSearch サーバーの leaf という文字列を含むインデックスのドキュメント数をカウントしたものを, 図 2.3 に移行先の ElasticSearch サーバーの leaf という文字列を含むインデックスのドキュメント数をカウントしたものを示す.

図 2.16 と図 2.17 より, ドキュメント数が一致していることからデータ移行が正常に行えたと判断できる.



名前	ヘルス	ステータス	プライマリ	レプリカ	ドキュメント数
leaf_load	yellow	open	1	1	3123
leaf	yellow	open	1	1	5070
leaf_grid	yellow	open	1	1	1948

図 2.16　133.71.201.197 の ElasticSearch サーバーの leaf という文字列を含むインデックスのドキュメントのカウント結果



名前	ヘルス	ステータス	プライマリ	レプリカ	ドキュメント数
leaf_load	green	open	1	1	3123
leaf	green	open	1	1	5070
leaf_grid	green	open	1	1	1948

図 2.17　133.71.106.141 の ElasticSearch サーバーの leaf という文字列を含むインデックスのドキュメントのカウント結果

次に, 図 2.18 に移行元の ElasticSearch サーバーの pcs_log インデックスのドキュメント数をカウントしたものを, 図 2.5 に移行先の ElasticSearch サーバーの pcs_log インデックスのドキュメント数をカウントしたものを示す.

図 2.18 と図 2.19 より, ドキュメント数が一致していることからデータ移行が正常に行えたと判断できる.

名前	ヘルス	ステータス	プライマリ	レプリカ	ドキュメント数
pcs_log	yellow	open	1	1	1535

図 2.18　133.71.201.197 の ElasticSearch サーバーの pcs_log インデックスのドキュメントのカウント結果

名前	ヘルス	ステータス	プライマリ	レプリカ	ドキュメント数
pcs_log	green	open	1	1	1535

図 2.19　133.71.106.141 の ElasticSearch サーバーの pcs_log インデックスのドキュメントのカウント結果

2.21 節　概要

今回は, CO_2 データの収集を行っているラズベリーパイの一部が, データ移行作業の移行先である 133.71.106.141 の ElasticSearch サーバーの co2 インデックスに対してインサートを行っていたことにより, 正しいデータ移行先である co2_modbus インデックス以外のインデックスに一部の CO_2 データが保存されている問題を解消したことについて報告する.

2.22 節　データ移行手順について

ラズベリーパイから co2 インデックスに対してインサートした全てのドキュメントをローカルマシンに JSON 形式でエクスポートした後, 作成した Python プログラムを実行して co2_modbus インデックスにインサートした.

2.22.1　データのエクスポート

移行元の ElasticSearch サーバーのデータのローカルマシンへのエクスポートには, elasticdump [?] ライブラリを使用して JSON 形式でエクスポートした.

co2 インデックスに対してラズベリーパイからインサートしたデータには JPtime フィールドが存在しないため, JPtime フィールドが存在しないドキュ

メントのみを対象としてエクスポートした.

また, エクスポートした JSON データを解析したところ, 最も古い utctime フィールドの日付は 2023 年 7 月 5 日だった.

2.22.2　データのインポート

エクスポートした JSON ファイルを, 作成した Python プログラムから読み込み, Python の elasticsearch ライブラリを用いて co2_modbus インデックスにインサートした.

2.23 節　データ移行が正常に行えたか確認

図 2.20 に co2 インデックスの 2023 年 7 月 1 日以降の PPM 値の推移をグラフにしたものを, 図 2.3 に co2_modbus インデックスの 2023 年 7 月 1 日以降の PPM 値の推移をグラフにしたものを示す.

図 2.20 と図 2.21 より, co2 インデックスのデータが正常に co2_modbus インデックスに移行できていることが分かる.

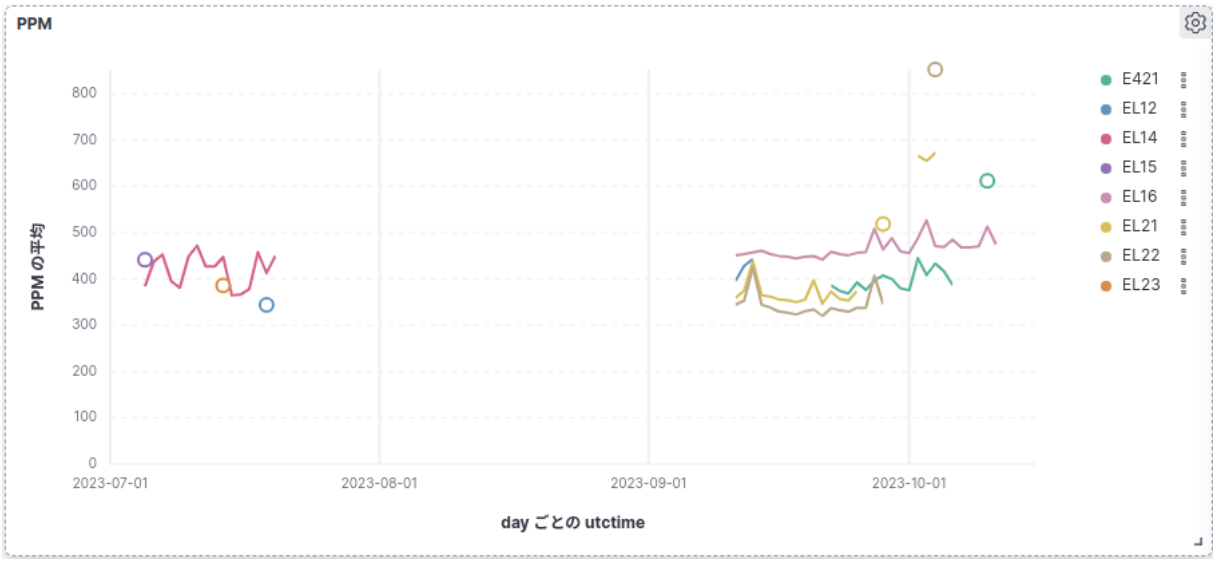


図 2.20　co2 インデックスの PPM

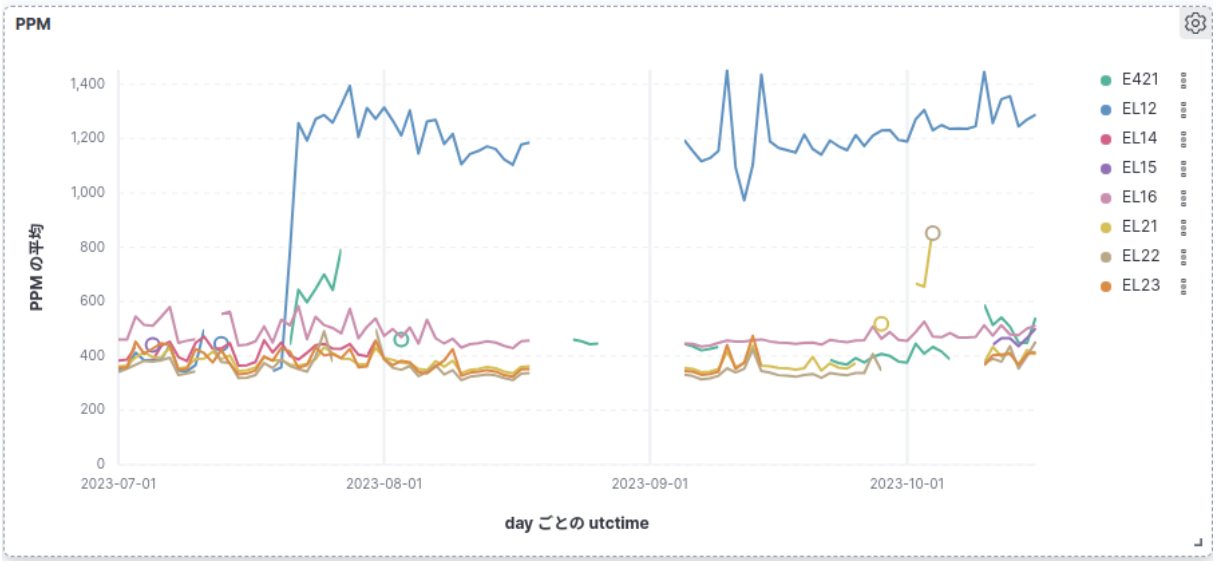


図 2.21　co2_modbus インデックスの PPM

次に, 図 2.22 に co2 インデックスの 2023 年 7 月 1 日以降の RH 値の推移をグラフにしたものを, 図 2.5 に co2_modbus インデックスの 2023 年 7 月 1 日以降の RH 値の推移をグラフにしたものを示す.

図 2.22 と図 2.23 より, co2 インデックスのデータが正常に co2_modbus インデックスに移行できていることが分かる.

次に, 図 2.24 に co2 インデックスの 2023 年 7 月 1 日以降の TEMP 値の推移をグラフにしたものを, 図 2.25 に co2_modbus インデックスの 2023 年 7 月 1 日以降の TEMP 値の推移をグラフにしたものを示す.

図 2.24 と図 2.25 より, co2 インデックスのデータが正常に co2_modbus インデックスに移行できていることが分かる.

2.24 節　結言

本章学内ゾーンにおける Elasticsearch クラスタへのデータ移行について述べた。

次章ではサーバーゾーンでのクラスタ構築における仮想環境を使用した事前検証について述べる。

図 2.22　co2 インデックスの RH

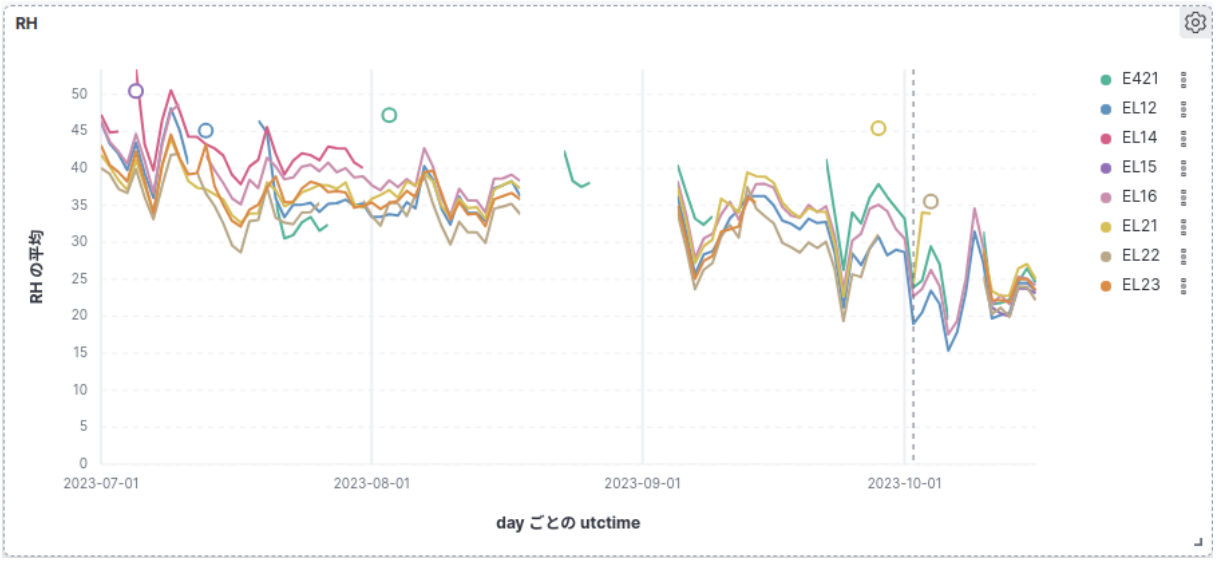


図 2.23　co2_modbus インデックスの RH

図 2.24　co2 インデックスの TEMP

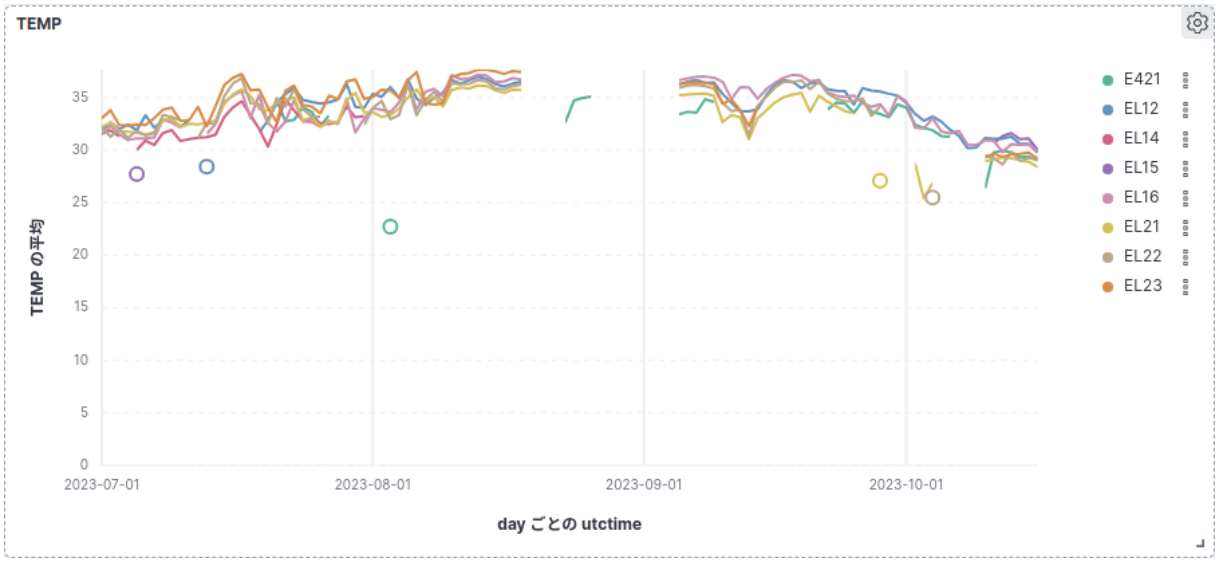


図 2.25　co2_modbus インデックスの TEMP

第3章

3章のタイトル

3.1節　緒言

本章では、… ついて述べる。

3.2節　結言

本章では、… について述べた.

第4章

4章のタイトル

4.1節　緒言

本章では…について述べる.

4.2節　結言

本章では…について述べた.

第5章

結論と今後の課題

……

謝 辞

本研究を行うにあたり、終始、懇切丁寧な御指導と適切な御助言を賜りました本学工学部電気電子工学科通信システム工学研究室の都築伸二教授に深甚なる感謝の意を表します。

最後に，有益な御助言を賜りました本大学大学院の〇〇に心より御礼申し上げます。

参考文献

[3]　著者, タイトル, 引用日など.

付録 A

付録

A.1 水源監視システム (送信用 Python スクリプト)

LoRa_obs_transmit.py のソースコードを A.1 に示す.

Listing A.1 LoRa_obs_transmit.py

```
## ++++++coding:utf-8++++++

import time
import os
from datetime import datetime
import serial

""" sleep()をいれて，少し待たないとエラー落ちする """
time.sleep(60)

class Main() :

    def __init__(self):
        """ 初期値および対象ディレクトリの設定 """
        self.s_num = 0

        self.copy_dir = "C:/Users/taikimizukan/Dropbox/sumitomo
          /obscsv/"
        self.target_dir = "C:/Users/taikimizukan/Desktop/
          obs_data/"
```

```
19          self.temporary_log = "./temporary_log.txt"
20
21          """ 最終データを取得 """
22          with open(self.temporary_log,"r") as f :
23              self.old_line = f.readline()
24              print("前回のデータ:"+str(self.old_line))
25
26          """ 起動時 [$RFINF,ON コマンド送信***] """
27          INF = "$RFINF,ON***"
28          with serial.Serial("COM7",115200,timeout=2) as ser :
29              time.sleep(2)
30              while True :
31                  for i in INF :
32                      ser.write(i.encode("utf-8"))
33
34                  result = str(ser.readline())
35                  if result.find("RESULT,RFINF,ON,OK") > 0 :
36                      break
37                  else :
38                      time.sleep(2)
39
40          """ ループ関数実行 """
41          self.Loop()
42
43
44      """ 作成日時が最新ファイルのフルパスを取得し返す関数 """
45      def get_file_path(self, target_dir) :
46          """ 対象ディレクトリ下の.ファイルのパスを取得し
              dat, [target_files に納める] """
47          target_files = []
48          for root, dir, files in os.walk(target_dir) :
49              target_file = [os.path.join(root,f) for f in files
                  if f.endswith(".dat")]# .txt -> .へ
                  dat
50              target_files.extend(target_file)
51          """ 取得した.ファイルのフルパスに作成時間を足してリストに納める
              dat """
52          file_ctime = []
53          for f in target_files :
54              file_ctime.append((f,os.path.getctime(f)))
55          """ 取得時間でソートし最新の.ファイルのパスのみ返すdat"""
56          sorted_file_ctime = sorted(file_ctime,key=lambda x :x
```

```
            [1])
57
58          return sorted_file_ctime[len(sorted_file_ctime)-1][0]
59
60      """ 最終行を取得，シークエンス番号を加えてコピー  """
61      def check_copy(self):
62  ##          name = self.target_file.replace(self.target_dir,"")
63          with open(self.target_file,"r") as f :
64              """ ファイルデータを全て読み込，最終行だけを取得  """
65              lines = f.readlines()
66              if len(lines) > 0 :
67                  line = lines[len(lines)-1]
68              else :
69                  line = self.old_line
70
71          """ 最終行が前回のものと異なるか？  """
72          if line != self.old_line :
73
74              """ シークエンス番号を追加  """
75              self.s_num += 1
76
77              file_name = line.split(",")
78              file_name = "obs_" +"".join(file_name[0:3])
79              self.ymd = "".join(file_name[0:3])
80
81              self.old_line = line
82
83              """ にコピーDropbox  """
84              with open(self.copy_dir+file_name+".csv","a") as cf
                    :
85                  data = line.strip() + "," + str(self.s_num)+"\n
                      "
86                  cf.write(str(data))
87
88              """ 最終行を保存  """
89              with open(self.temporary_log,"w") as f :
90                  f.write(line)
91
92              self.arduino_serial(data)
93              time.sleep(5)
94              self.TxMSG()
```

```
95              self.Ping()
96
97          else :
98              print("Not␣updated")
99              pass
100
101     """ を経由してにデータを送信する関数  arduinoLoRa"""
102     def arduino_serial(self,d) :
103         print("---"*5 + "arduino_serial" + "---"*5)
104         buf = 0
105         with serial.Serial("COM7",115200,timeout=1) as ser :
106             """ ポートを開いて少し待機が必要 """
107             time.sleep(2)
108             """ごみの吸出し"""
109             buf = ser.readlines()
110             d = d.strip()
111             """ 送信コマンドの形に """
112             d = "$RFSND,0004,"+d+"***"
113             print("To␣arduino␣Data␣-->␣" +d)
114
115             """ Python(PC) -> arduino -> LoRa だと文字ずつ送らないと
                  いけない？1 """
116             for i in d :
117                 ser.write(i.encode("utf-8"))
118
119         """ 0009 : 第二中継機にダミーをとばすMSG, 戻り値を保存 """
120     def TxMSG(self) :
121         target_add = "0009"
122         self.now = datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
123         self.today = datetime.today().strftime("%Y%m%d")
124
125         msg = "$RFSND,{0},{1},{2},{2}***".format(target_add,
              self.now,self.counter)
126
127         with serial.Serial("COM7",115200,timeout=15) as ser :
128                 time.sleep(2)
129                 for i in msg :
130                         ser.write(i.encode("utf-8"))
131                         time.sleep(0.05)
132                 print(ser.readline().decode("utf-8"))
133                 res = ser.readline().decode("utf-8")
```

```
134
135         if len(res) > 10 :
136                 res = res.replace("␣",",").replace("*",",").
                      replace(":",",")
137                 with open("C:/Users/taikimizukan/Dropbox/
                      sumitomo/RSSI_CHECK_TX/rssi_tx_obs_{}.csv".
                      format(str(self.today)),"a") as f :
138                     f.write(res+"\n")
139         else :
140                 pass
141
142     """発電所のにを送って生存確認LoRaping"""
143     def Ping(self) :
144         PING = "$RPING,0004***"
145         with serial.Serial("COM7",115200,timeout=10) as ser :
146             time.sleep(2)
147             for i in PING :
148                 ser.write(i.encode("utf-8"))
149                 time.sleep(0.05)
150
151             print(ser.readline().decode("utf-8"))
152             res_ping = ser.readline().decode("utf-8")
153
154         if len(res_ping) > 10 :
155             print(res_ping)
156             now = datetime.now().strftime("%Y,%m,%d,%H,%M,%S")
157             with open("C:/Users/taikimizukan/Dropbox/sumitomo/
                      PING/ping_{}.csv".format(str(self.today)),"a") as
                       f :
158                     f.write(str(now)+","+str(self.s_num)+"," +
                          str(res_ping))
159         else :
160             pass
161
162     """ 繰り返し """
163     def Loop(self):
164         while True:
165             try :
166                 time.sleep(20)
167                 self.target_file = self.get_file_path(self.
                      target_dir)
```

```
168
169                     self.check_copy()
170
171                 except Exception as E :
172                     with open("./Error_Log.txt","a") as ef :
173                         ef.write(str(E))
174
175
176 if __name__ == "__main__" :
177     Main()
```

A.2 水源監視システム (受信用 Python スクリプト)

LoRa_obs_receive.py のソースコードを A.2 に示す.

Listing A.2 LoRa_obs_raceive.py

```
1  ## coding:utf-8
2
3  import paho.mqtt.client as mqtt
4  import serial
5  from datetime import datetime
6  import time
7
8  class Main() :
9      def __init__(self) :
10         self.INF_Input()
11         self.Loop()
12
13     def INF_Input(self) :
14     """ 起動時 [$RFINF,ONコマンド送信***] """
15
16         INF = "$RFINF,ON***"
17         with serial.Serial("COM3",115200,timeout=2) as ser :
18             time.sleep(2)
19             while True :
20                 for i in INF :
21                     ser.write(i.encode("utf-8"))
22
23                 result = str(ser.readline())
24                 if result.find("RESULT,RFINF,ON,OK") > 0 :
25                     print("RFINF,OK")
26                     break
27                 else :
28                     time.sleep(2)
29
30     def LoRa_Receive(self) :
31         try :
32         """ からデータを読み込むLoRa """
33             while True :
34                 with serial.Serial("COM3",115200,timeout=120)
                    as ser :
```

```
35                      res = ser.readline().decode("utf-8")
36                      if len(res) > 15 and res.find("RFRX") > 0
                          and res.find("RECEIVED") < 0 :
37                          print("==="*25)
38                          print("RXData␣␣==>␣"+res)
39                          break
40
41                      else :
42                          print("==="*25)
43                          print("else_data=>"+res)
44
45              """ 必要なデータを取り出す """
46              res_list = res.split("*")[0].split(",")[1:]
47              data = ",".join(res_list)
48              add = res_list[0]
49
50              """ データの日付を確認(ymd) """
51              ymd = "".join(res_list[1:4])
52
53
54              return = res, data, add, ymd
55
56          except Exception as E :
57              now = datetime.now().strftime("%y/%m/%d␣%H:%M:%S")
58              with open("LoRa_Receive_Error_LOG.txt","a") as ef :
59                      ef.write(now +"␣:␣"+ str(E)+"\n")
60              self.Loop()
61
62
63
64      def MQTT_Publish(self, res):
65      """ 情報MQTT(publish) """
66          host = "133.71.***.***"
67          port = 1883
68          topic = "***********"
69          """ MQTT-Publish """
70          try :
71              print("publish␣==>␣" +str(res))
72              client = mqtt.Client(protocol=mqtt.MQTTv311)
73              client.connect(host,port=port,keepalive=10)
74              client.publish(topic,res)
```

```
75
76          except Exception as E :
77              now = datetime.now().strftime("%y/%m/%d␣%H:%M:%S")
78              with open("Publish_Error_LOG.txt","a") as ef :
79                      ef.write(now +"␣:␣"+ str(E)+"\n")
80
81      def Storage(self,res,data,ymd):
82          try :
83              """ メタデータを保存 """
84              with open("./meta_data/meta_data_{}.txt".format(ymd
                  ),"a") as f :
85                  f.write(res+"\n")
86
87              """ データを保存 """
88              with open("./data/data_{}.txt".format(ymd),"a") as
                  f :
89                  f.write(data+"\n")
90
91              """ に保存Dropbox """
92              with open("C:/Users/sumitomo02/Dropbox/test_folder/
                  RX/DATA/RX_{}.txt".format(ymd),"a") as f :
93                  f.write(data+"\n")
94
95              with open("C:/Users/sumitomo02/Dropbox/test_folder/
                  RX/META_DATA/RX_{}.txt".format(ymd),"a") as f :
96                  f.write(res+"\n")
97
98          except Exception as E :
99              now = datetime.now().strftime("%y/%m/%d␣%H:%M:%S")
100             with open("Storage_Error_LOG.txt","a") as ef :
101                 ef.write(now +"␣:␣"+ str(E)+"\n")
102             print(E)
103
104     def Loop(self) :
105         while True :
106             res, data, add, ymd = self.LoRa_Receive()
107             self.Storage(res, data, ymd)
108             self.MQTT_Publish(res)
109             self.Ping(add)
110
111
```

```
112 main = Main()
```