学位論文

太陽光発電データの時刻補正手法と
Elasticsearch ノードのクラスタ化手法の提案

提出年月日　令和 6 年 2 月 13 日
改訂日　　令和 6年 3月 1日

指導教員　　都築　伸二　教授

入学年度　　令和 4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, 太陽光発電データの時刻補正手法と Elasticsearch ノードのクラスタ化手法の提案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 太陽光発電データの時刻補正手法

ここでは, 太陽光発電データの計測時刻の補正手法を提案する. また, pvlib と呼ばれる太陽光発電シュミレーターライブラリを導入して, 計測データを前処理することによる補正誤差の改善手法についても提案する.

第 3 章 単一 Elasticsearch ノードをクラスタ化する前に行うデータ移行

ここでは, 単一 Elasticsearch ノードをクラスタ化する前に行うデータ移行について述べており, CO_2 データの移行と LEAF の運行日誌のデータ移行手法を提案する.

第 4 章 仮想環境を使用したクラスタリング動作の検証

ここでは, Docker による仮想環境を使用したクラスタリング動作の検証について述べており, バージョンの異なる Elasticsearch ノードを用いたクラスタ構築の可否や, 既存 Elasticsearch ノードの異なるクラスタへの参加の可否の検証結果について述べる.

第 5 章 結論と今後の課題

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まとめている.

目 次

第1章

緒論

太陽光発電は, 再生可能エネルギー源として世界中で注目されており, 効率的な運用と管理には正確な計測データが不可欠である. しかしながら, 松山市で管理している太陽光発電データは, データを計測している PC の内部時計が標準時刻とずれているため, 他地点で計測しているデータと時刻同期ができない問題があった.

本論文の 2 章では, この問題に対処するため, 太陽光発電データの計測時刻の補正手法を提案する. pvlib と呼ばれる太陽光発電シュミレーターライブラリを導入して, 計測データを前処理することにより補正する手法である.

次に, 発電データ等を蓄積している Elasticsearch サーバの故障耐性を向上することを目的としてクラスタ化する際に実施した作業を 2 つ述べる.

第 3 章では, 単一で動いているサーバのデータをエクスポートする方法. エクスポートしたデータのうち重複するデータを削除する方法. 最後にクラスタにインポートする方法について述べる.

第 4 章では, バージョンの異なる Elasticsearch ノードを用いたクラスタ構築の可否や, 既存 Elasticsearch ノードの異なるクラスタへの参加の可否の検証結果を述べる.

第 5 章では, 本研究の成果と課題を明らかにする.

第2章

太陽光発電データの時刻補正手法

2.1節　緒言

本章では, 太陽光発電データの計測時刻の補正手法を提案する.

2.2節　太陽光発電の計測データの問題点について

本研究で管理している太陽光発電データは, データを計測しているPCの内部時計が標準時刻とずれているため, 他地点で計測しているデータと時刻同期ができない問題があった.

そこで, 時刻がずれていない実測値と, 計算式により求まる大気外日射量との間の時間的遅延の秒数を相互相関を用いて求めることで, 実測値を標準時に補正する.

2.3節　大気外日射量の計算式

任意の緯度経度, 時刻における大気外日射量Qは, 任意の緯度ϕ, 経度λの地点における任意の時刻, 太陽高度αから求めることができる.

まず, 次式より元旦からの通し日数dnに基いて定めたθを用いて, 当該日の

太陽赤緯 δ, 地心太陽距離 $\frac{r}{r^*}$, 均時差 E_q をそれぞれ以下の式により求める.

$$\theta = \frac{2\pi(dn-1)}{365} \tag{2.1}$$

$$\begin{aligned}\delta = 0.006918 - 0.399912\cos\theta + 0.070257\sin\theta - 0.006758\cos 2\theta \\ +0.000907\sin 2\theta - 0.002697\cos 3\theta + 0.001480\sin 3\theta\end{aligned} \tag{2.2}$$

$$\begin{aligned}&\frac{r}{r^*} \\ &= \frac{1}{\sqrt{1.000110 + 0.034221\cos\theta + 0.001280\sin\theta + 0.000719\cos 2\theta + 0.000077\sin 2\theta}}\end{aligned} \tag{2.3}$$

$$\begin{aligned}E_q = 0.000075 + 0.001868\cos\theta - 0.032077\sin\theta \\ -0.014615\cos 2\theta - 0.040849\sin 2\theta\end{aligned} \tag{2.4}$$

日本標準時間から, 太陽の時角 h を求める.

$$h = \frac{(\text{日本標準時間} - 12)\pi}{12} + \text{標準子午線からの経度差} + E_q \tag{2.5}$$

δ, ϕ, h の値が既知となったので α は

$$\alpha = \arcsin(\sin\phi\sin\delta + \cos\phi\cos\delta\cos h) \tag{2.6}$$

により求まる.

最後に, Q が

$$Q = 1367(\frac{r^*}{r})^2\sin\alpha \tag{2.7}$$

により求まる. 1367W/m^2 は太陽定数である.

式 (2.1)〜式 (2.7) を用いることで, 任意の緯度経度, 時刻における大気外日射量が求まる.

2.3.1　実測値と大気外日射量との比較

Elasticsearch サーバーから取得した 2022 年 6 月 2 日のリサイクル館の実測値と, 計算式から求めた大気外日射量を図 2.1 に示す. 実測値は地表に設置された太陽光パネルが計測したものであるのに対して, 大気外日射量は地球大気の上端（約 8km 上空）で受け取る日射量を計算したものであるため, 一日を通して最大となる日射量の大きさに差がある.

なお, 図2.1 は`python3 calc_corr.py -dt 2022/06/02 -surface_tilt 22 -surface_azimuth 179 -show_graph` コマンドを実行してプロットしている.

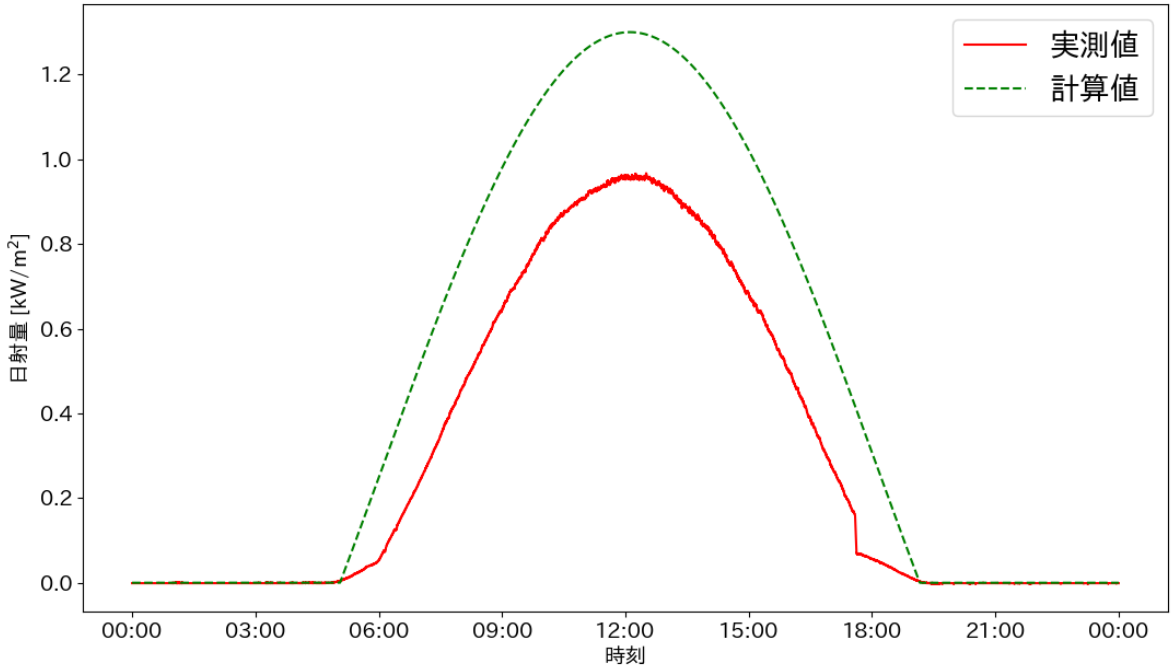

図 2.1　2022 年 6 月 2 日の実測値と大気外日射量（式 (2.7)）

2.3.2　相互相関による時刻補正法

相互相関の計算に使用するデータの計測期間を, 快晴だった 2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分までとした. なお, 使用したデータは

リサイクル館で計測し, インターネット時刻に同期したタイムスタンプを計測時刻としている.

同じ期間の大気外日射量を式 (2.7) で求め, 実測値との相互相関を計算した結果, 実測値を 124 秒遅らせると, 相関が最大となった. また, 同様に快晴であった 2022 年 5 月 3 日と 2022 年 5 月 18 日ではそれぞれ 39 秒, 71 秒遅らせた際に相関が最大となり, 3 日間の平均値は 78 秒であった.

使用した実測値はインターネット時刻に同期しているので, 遅れ時間は 0 秒となるのが正しいにも関わらず平均 78 秒となったのは, 地表日射量ではなく大気外日射量を使用したためと考えられる.

2.4 節　地表日射量の予測

地表日射量の予測を行うため, 式 (2.1)〜式 (2.7) を使った方法ではなく, pvlib ライブラリ [2] を使用して地表日射量を求め, 相互相関を計算した.

2.4.1　pvlib の概要

pvlib は, 太陽光発電システムの性能シミュレーションや関連するタスクを実行するための関数とクラスのセットを提供するライブラリである.

以下は, pvlib の主な特徴である.

- 太陽位置計算: pvlib は, 地球上の任意の場所における太陽の位置を計算する機能を提供する. これは, 太陽の方位角や高度角を求めるのに使用される.
- 大気透過モデル: 大気を通過する太陽放射の量や質を推定するモデルが含まれている.
- 太陽光発電システムの性能モデリング: 太陽光発電モジュールやインバーターの性能モデルが含まれており, これらの条件下での太陽光発電システムの出力をシミュレートできる.

2.4.2　実測値と pvlib により求まる地表日射量の比較

Elasticsearch サーバーから取得したリサイクル館の実測値と, pvlib を用いて求めた地表日射量との比較を図 2.2 に示す.

なお, 図2.2は`python3 calc_corr.py -dt 2022/06/02 -surface_tilt 22 -surface_azimuth 179 -show_graph` コマンドを実行してプロットしている.

図 2.2 と図 2.1 と比較すると, 大気外日射量より地表日射量の方が実測値により近い概形となっていることが分かる.

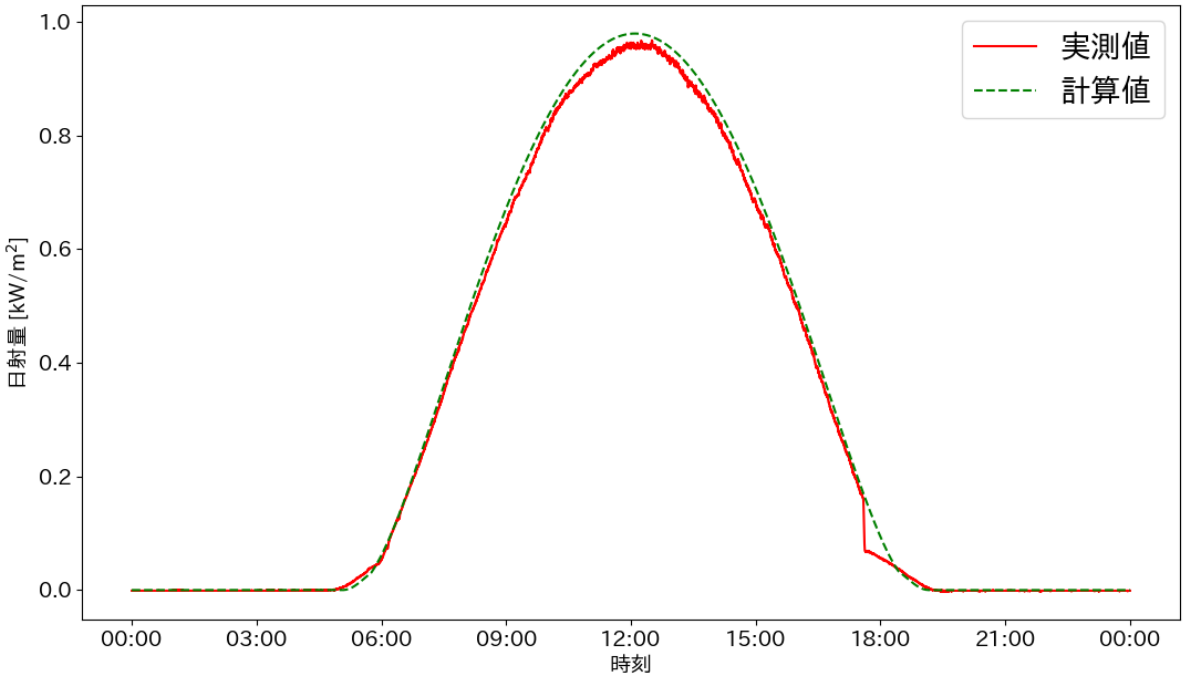

図 2.2　2022 年 6 月 2 日の実測値と地表日射量（pvlib 値）

2.4.3　地表日射量との相互相関による時刻補正法

相互相関の計算に使用した期間は図 2.1 と同じ 2022 年 6 月 2 日 0 時 0 分から 2022 年 6 月 2 日 23 時 59 分とした.

この期間の地表日射量を求め, 相互相関を計算した結果, 実測値を 121 秒遅

らせた際に, 相関が最大となることが分かった. また, 2022 年 5 月 3 日と 2022 年 5 月 18 日ではそれぞれ 26 秒, 59 秒遅らせた際に相関が最大となり, 3 日間の平均値は 68 秒であった.

式 (2.1)〜式 (2.7) より求めた大気外日射量を用いて相互相関を計算した時と比較して, 78 秒から 68 秒へと 10 秒改善した.

2.5 節　前処理の追加による時刻補正法とその精度

図 2.2 では, 日の入時刻の辺りにおいて, 実測値と地表日射量の概形が大きく異なっている.

太陽光パネルの周囲にある建造物の影による実測値のひずみを事前に取り除いた上で相互相関を計算することで, 相互相関の計算結果が改善するかを検証するために, 実測値に対して前処理を追加した.

前処理を含めた相互相関の計算方法は以下の手順で行う.

1. 実測値が歪んでいる時間の日射量を 0 $\mathrm{kW/m^2}$ と見なすためのしきい値の指定: 本論文では, 0.2 $\mathrm{kW/m^2}$ をしきい値として設定した.
2. しきい値に該当する計測時刻の特定: 続いて, 実測値の日射量データから 0.2 $\mathrm{kW/m^2}$ を減算して絶対値を取った際に最も 0 に近い値を取る計測時刻を午前と午後でそれぞれ一点ずつ特定した.
3. 特定した計測時刻を使った実測値のフィルタリング: 前のステップで得た計測時刻を使用して, 2 点の計測時刻の外側にある実測値の日射量を 0 $\mathrm{kW/m^2}$ とした.
4. 地表日射量との相互相関の計算: 実測値の歪み部分を 0 $\mathrm{kW/m^2}$ とした後, 地表日射量との相互相関を計算した.

図 2.3 に, この前処理を行ったリサイクル館の実測値と地表日射量を比較する.

なお, 図 2.3 は `python3 selective_corr.py -dt 2022/06/02 -slide_seconds 0 -surface_tilt 22 -surface_azimuth 179 -threshold_q 0.2 -show_preprocessing -show_threshold_q` コマンドを実行してプロットしている.

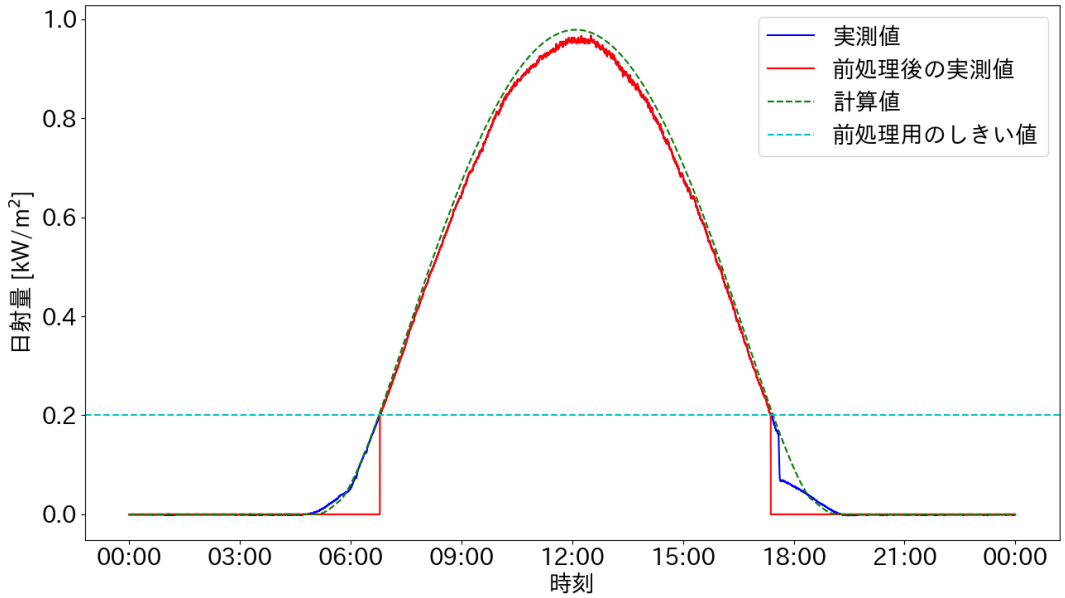

図 2.3　前処理を行った 2022 年 6 月 2 日の実測値と地表日射量（計算値）

前処理を行った実測値と, 地表日射量から相互相関を計算した結果, 実測値を 70 秒遅らせた際に, 相関が最大となった. また, 2022 年 5 月 3 日は 21 秒進めた際に相関が最大となり, 2022 年 5 月 18 日は 12 秒遅らせた際に相関が最大となった. 3 日間の平均値は 34 秒であった.

前処理を追加したことで相互相関の結果が, 68 秒から 34 秒へと 34 秒改善した.

2.6 節　結言

本章では太陽光発電データの計測時刻の補正手法を提案した. その結果, 前処理を追加したことで相互相関の結果が 68 秒から 34 秒に改善した.

次章では本章で述べた 2 年分のデータを蓄積している単一 Elasticsearch ノードをクラスタ化する前に行うデータ移行手法について述べる.

第3章

単一Elasticsearchノードをクラスタ化する前に行うデータ移行

3.1節　緒言

本章では単一 Elasticsearch ノードにあるデータのエクスポート, 重複データの削除, そしてクラスタにインポートするデータ移行に関する作業ついて述べる.

3.2節　Elasticsearchの概要

Elasticsearch は, 分散処理に対応した全文検索エンジンである. 主な特徴は, 以下の通りである.

- 高速な検索性能: ビッグデータなどの巨大で複雑なデータの集合にも対応可能
- 部分一致検索が可能: 検索キーワードの一部に一致するドキュメントも検索可能
- ほぼリアルタイムの検索: ドキュメントにインデックスを付けてから検索可能になるまで約 1 秒程度

- スケーラビリティ: サーバー数を増やすことで, 検索性能と処理能力を拡張可能

これらの特徴から, Elasticsearch は, 以下のような用途に適している.

- ログ分析: Web サイトやアプリケーションのログから, アクセス状況やエラー情報を分析する
- セキュリティインテリジェンス: ネットワークやシステムから, セキュリティ脅威を検知する
- ビジネス分析: 顧客データや販売データから, トレンドや傾向を分析する

3.3節 Kibanaの概要

Kibana は, Elasticsearch に保存されたデータを可視化するためのツールである. 主な特徴は, 以下の通りである.

- 直感的な操作性: ドラッグ＆ドロップで簡単に可視化を作成できる
- 豊富な可視化機能: グラフ, 表, 地図など, さまざまな可視化機能を提供
- 高度なフィルタリング機能: 条件を指定して, データを詳細に絞り込むことができる

これらの特徴から, Kibana は, 以下のような用途に適している.

- ログ分析: Web サイトやアプリケーションのログから, アクセス状況やエラー情報を可視化する
- セキュリティインテリジェンス: ネットワークやシステムから, セキュリティ脅威を可視化する
- ビジネス分析: 顧客データや販売データから, トレンドや傾向を可視化する

3.4節 データ移行対象のElasticsearchインデックスについて

133.71.106.168で稼働している単一ノードのElasticsearchに保存されたCO_2データとLEAFの運行日誌に関するデータを, Elasticsearchクラスタへ移行する.

3.4.1 CO_2データ

CO_2データが保存されたインデックスは, インデックス名にco2という文字列が含まれているため, co2という文字列を含む全てのインデックスを移行対象とした.

3.4.2 LEAFの運行日誌に関するデータ

LEAFの運行日誌に関するデータが保存されたインデックスは以下の2つである.

- movement_diary
- movement_diary01

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.
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.

上述したドキュメントの違いより, movement_diary01 は movement_diary のもつ全ての情報を保持しており, 更に movement_diary にはないフィールドを持っている. 更に, movement_diary と movement_diary01 のドキュメント数は等しく, すべてのドキュメントのタイムスタンプが一致しているため, movement_diary01 インデックスのみ移行した.

3.5 節 CO_2 データの移行手順ついて

Elasticsearch に保存された CO_2 データには, 計測時刻, 部屋番号, 部屋の気温, CO_2 濃度などの情報が含まれている.

CO_2 デーのタ移行を行うに当たって, 計測時刻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いるため, 重複データを削除した上でデータ

を移行する必要があった. 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インサートした.

3.5.1 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.

3.5.2 データの重複削除

重複データの削除はインストールが容易だった SQLite データベースを用いて行った.

SQLite は, 軽量で自己完結型のデータベースエンジンである. SQLite は以下の特徴を持っている.

- 軽量: SQLite は非常に小さく, リソースの少ない環境でも動作する.
- 自己完結型: データベースが単一のファイルとして存在し, 外部の依存関係がない.
- トランザクション: SQLite は ACID トランザクションをサポートしており, データの整合性を保つ.
- フリーかつオープンソース: SQLite はパブリックドメインに属し, 誰でも自由に使用, 変更, 配布可能である.

SQLite では, 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 では, 部屋番号 (number) と計測時刻 (utctime) を複合主キーとして設定した. 以下のリスト 3.1, リスト 3.2 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, ドキュメントの持つフィールドが統一されておらず, 一部のセンサー情報が存在しないドキュメントが存在する. そのため, SQLite へのデータ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のセンサー情報などが欠けてしまう問題を解決した.

リスト 3.1 _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"
  }
}
```


リスト 3.2 _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"
    }
}
```

3.5.3 データのインポート

重複データ削除を行った後のデータが保存されているSQLiteからすべてのレコードを読み出して, 移行先のElasticsearchサーバーにインサートした.

インサートする際は, pythonのelasticsearchライブラリを使用し, co2_modbusという名前のインデックスに保存した.

3.6節 一度目のデータ移行で移行できなかったCO_2データの移行について

2023年5月中旬頃に, 実装したデータ移行プログラムを使用してElasticsearchクラスタへCO_2データの移行を行った. しかし, CO_2濃度監視システムの開発と運用を担当している高木君が, 計測データのインサート先を, Elasticsearchクラスタに変更したのが2023年7月中旬頃であった. このため, 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Elasticsearchクラスタ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ての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.
2. 次に, Elasticsearchクラスタに対して, CO_2濃度監視システムからインサートし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.
3. co2という文字列を含むインデックスに保存された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, elasticdumpライブラリを使用して移行元Elasticsearchサーバーからローカルマシンにエクスポートする.

4. 部屋番号 (number) と計測時刻 (utctime) の組み合わせがユニークになるよう SQLite を用いて, エクスポートしたデータの重複削除を行う.
5. ステップ 1, 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重複削除後のデータを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 クラスタにインサートする.

上記に手順に従い, 追加のデータ移行を行った. その結果, 338GB あった CO_2 データが削除後は 9.7GB に圧縮できた.

3.7節 KibanaによるCO2データの可視化

計 2 回の CO_2 データを移行した後の co2_modbus インデックスについて, 横軸を計測時刻 (utctime) とし, 縦軸を PPM, RH, 気温としてそれぞれプロットしたものを図 3.1 〜 図 3.3 に示す.

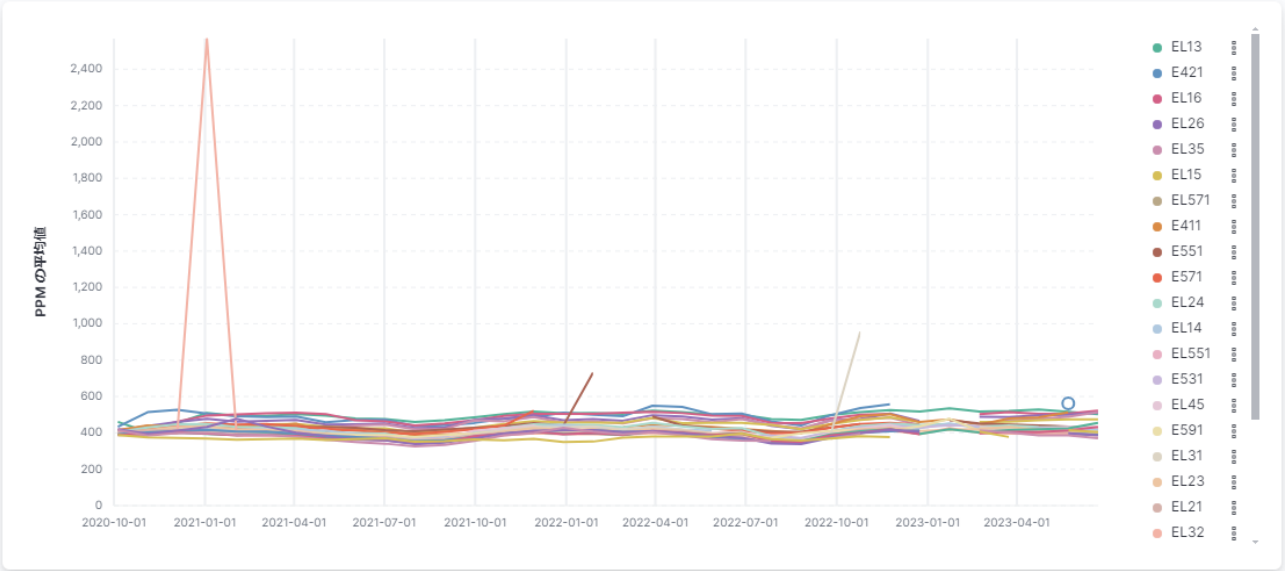

図 3.1 co2_modbus の PPM（30 日間の PPM の平均値）

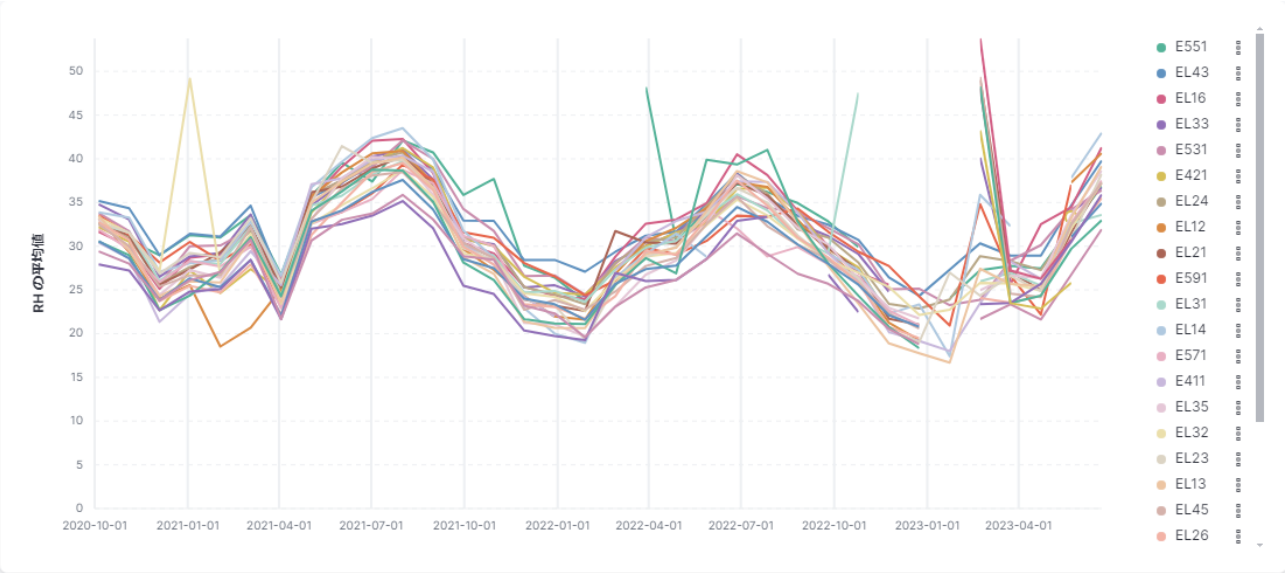

図 3.2　co2_modbus の RH（30 日間の RH の平均値）

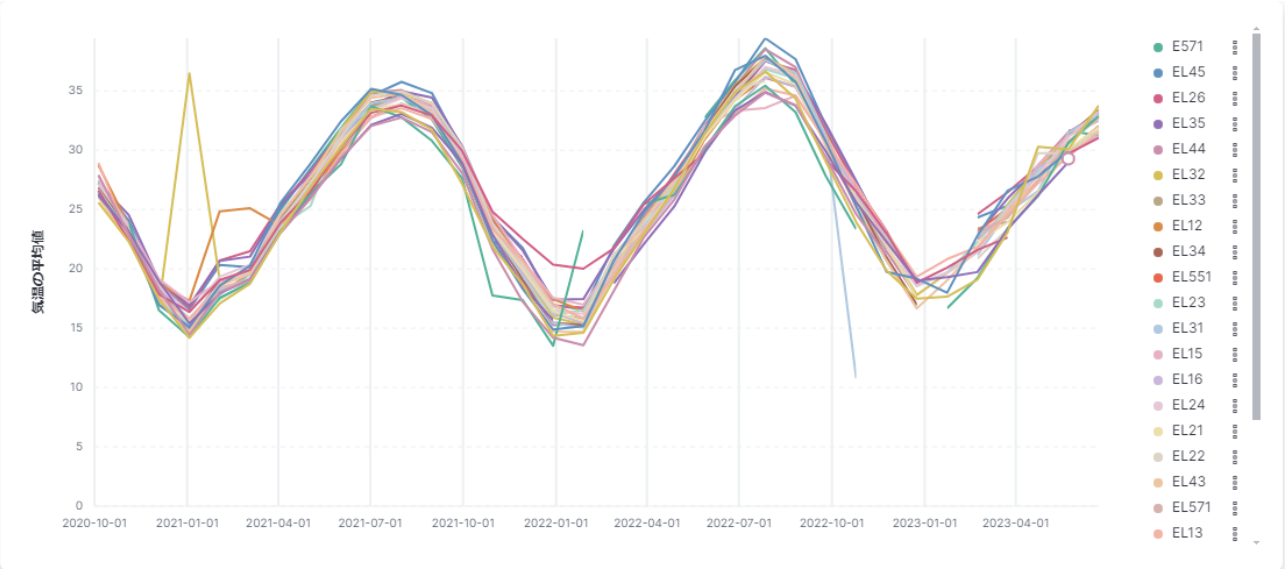

図 3.3　co2_modbus の気温（30 日間の気温の平均値）

図 3.1 ～ 図 3.3 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, データを欠落することなく重複データを削除出来たと判断した.

図 3.1 ～3.2 について, 他の教室より高い値を取り不連続点となっている箇

所が何点かあるが, これは計測システムで使用しているセンサーの不良によるものであり, 連続的に変化しなくて良いデータである.

図 3.3 についても, EL31 教室にて気温が連続的に変化していないがこれはデータ計測システムが正常に動作せず計測出来ていなかったことが原因である. また, 2021 年 1 月 1 日に E531 教室の気温が他の教室より 20 ℃程度高く計測されているが, これはセンサーの故障による誤った計測値であり, 連続的に変化しなくて良いデータである.

3.8 節 LEAF の運行日誌に関するデータの移行手順について

movement_diary01 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 クラスタに作成して, 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.

3.8.1 データのエクスポート

移行元の Elasticsearch サーバ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, movement_diary01 インデックスの全ドキュメントを JSON 形式でエクスポートした.

3.8.2 データのインポート

python の elasticsearch ライブラリを使用し, 移行先の Elasticsearch クラスタに movement_diary01 という名前のインデックスを作成して, エクスポートしたデータを全てインサートした.

3.9 節 Kibana による LEAF の運行日誌に関するデータの可視化

LEAF の運行日誌に関するデータを移行した後の movement_diary01 インデックスについて, 横軸を計測時刻 (dt_S), 縦軸を battery_S としてプロットし

たものを図 3.4 に示す.

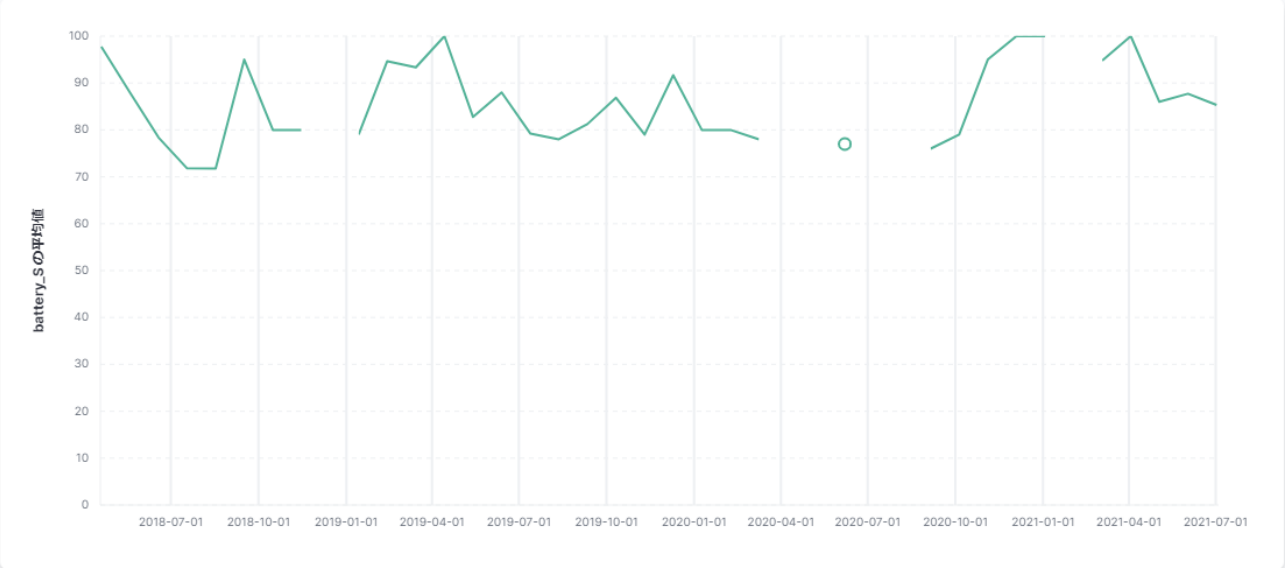

図 3.4　movement_diary01 の battery_S（30 日間の battery_S の平均値）

図 3.4 より, 一部を除いて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, データを移行出来たと判断した. なお, 連続的に変化していない箇所は, LEAF を稼働させていないことによるものであるため, 連続的に変化しなくて良いデータである.

3.10節　結言

本章では単一 Elasticsearch ノードから Elasticsearch クラスタへのデータ移行手順について述べた.

次章では, 移行先のクラスタを構成するノードを増設する方法を Docker による仮想環境を使用して検討する.

第4章

仮想環境を使用したクラスタリング動作の検証

4.1節　緒言

移行先のクラスタを構成するノードを増設する方法が不明であったので, 本章で検討する. 4.4 節では 2 ノードで動作しているクラスタを 3 ノードにする. 4.5 節では 3 ノードで動作しているクラスタに 1 ノード追加する. 本章では効率よく検討するために Docker による仮想環境を使用した.

4.2節　Dockerとは

Docker は, 軽量で独立したコンテナ型仮想環境用のプラットフォームである [4].

従来の仮想化では, VMWare [5] などの仮想化ソフトウェアを用いて, ホスト OS 上にゲスト OS を構築していた. しかし, Docker はホスト OS 上にゲスト OS なしで独立したコンテナ型の仮想環境として構築される. Docker コンテナは, Docker Engine 上で動作する. Docker Engine はコンテナの立ち上げ, 停止, 削除といった操作を行う.

本論文では, Docker イメージの形式で供給されている Elasticsearch を使用したため, 以降で述べる動作する/しないの判定が素早くできた.

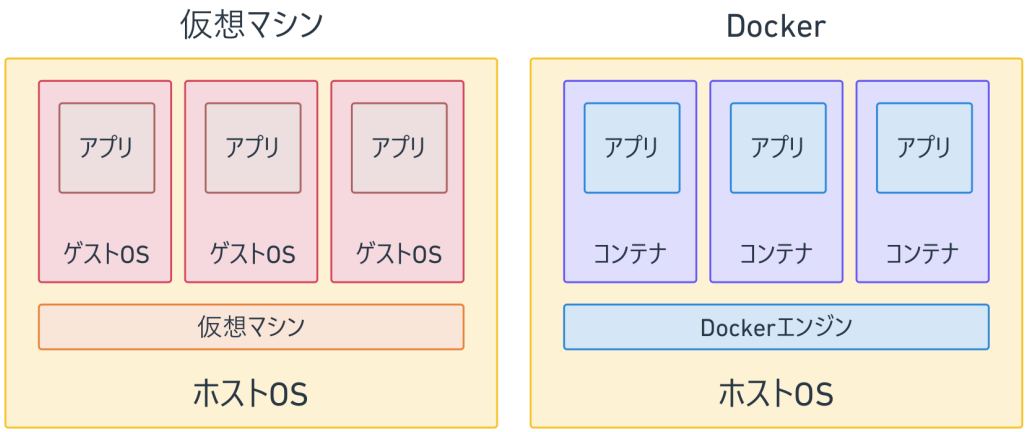

図 4.1　仮想マシンと Docker の違い

4.2.1　コンテナとは

コンテナは, アプリケーションとそのすべての依存関係（ライブラリ, 実行環境など）をカプセル化した軽量な実行単位である. Docker の場合, コンテナの作成には Docker イメージが必要となる.

4.2.2　Docker イメージとは

Docker イメージとは, Docker コンテナを作成するためのテンプレートであり, Docker イメージの中には, Docker コンテナの実行に必要な Linux ファイルシステムとメタ情報を含む.

Linux ファイルシステムというのは, / ディレクトリ以下の /etc /bin /sbin /usr などのディレクトリ階層およびファイルである.

Docker では, コンテナとして動かしたいアプリケーションが必要とする, 最小限のファイルを Docker イメージの中に入れる.

さらに, そのアプリケーションを動かすために必要なデフォルトのコマンドや引数の指定, 外に公開するポート番号の情報などの情報がある. これらをメタ情報として, 同じく Docker イメージの中に入れられる

Docker イメージは Docker Hub やその他のレジストリで共有されており, こ

れらのサービスから取得することが可能である.

今回は Elasticsearch の開発元である Elastic 社が提供している Elasticsearch の Docker イメージを使って検証を行う.

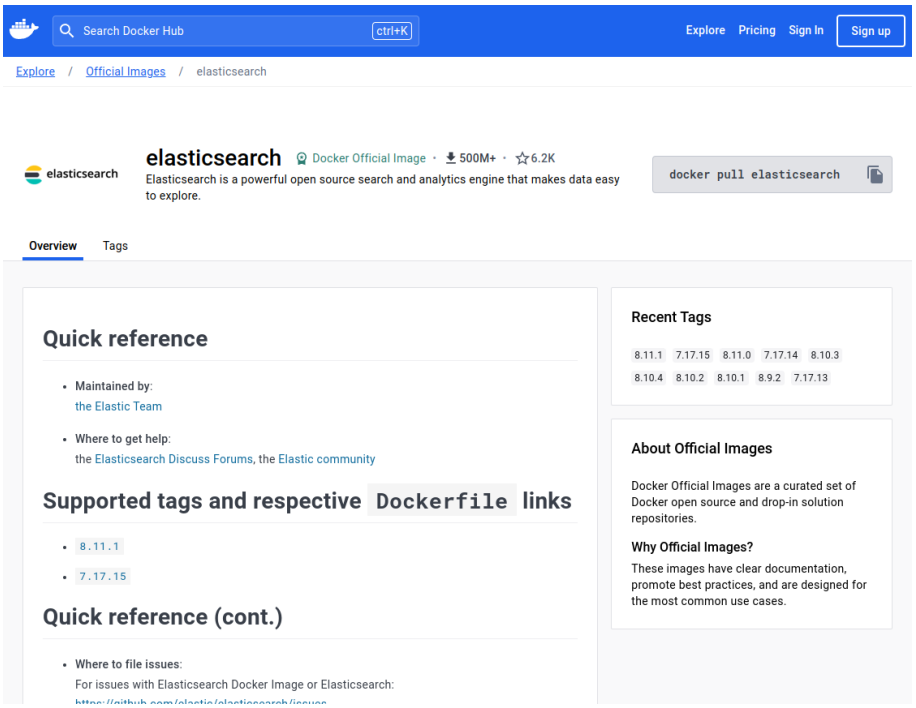

図 4.2　Elasticsearch の Docker イメージ

4.3 節　Docker Compose とは

Docker Compose は, 複数のコンテナを定義し, 実行するためのツールである. これは YAML ファイルを使用して設定され, 複数のコンテナで協調して動作するアプリケーションの開発を単純化する.

4.4 節　バージョンの異なる Elasticsearch ノードを用いたクラスタ構築の可否の検証

4.4.1　全て同じバージョンの Elasticsearch を使用したクラスタ構成 (全ノード バージョン 7.17.9)

図 4.3 に, 7.17.9 バージョンの Elasticsearch のみを使用してクラスタを構築した時の docker-compose.yml を図で表現したものを示す.

図 4.3　docker-compose.yml を図で表現したもの

クラスタの起動には, docker compose up -d コマンドを使用する.

docker compose up -d コマンドを実行した後, curl コマンドを使用してクラスタに参加しているノードを一覧表示した結果をリスト 4.1 に示す.

リスト 4.1　クラスタに参加しているノードを一覧表示した結果

```
$ curl -X GET "localhost:9200/_cat/nodes?v=true&pretty"
ip heap.percent ram.percent cpu load_1m load_5m load_15m
   node.role master name
172.21.0.2 17 92 1 2.25 1.22 0.81 cdfhilmrstw * es02
172.21.0.4 11 92 1 2.25 1.22 0.81 cdfhilmrstw - es03
172.21.0.3 39 92 2 2.25 1.22 0.81 cdfhilmrstw - es01
```

リスト 4.1 より, 3 つのノード（es01, es02, es03）すべてが正常にクラスタに参加できていることが確認できる.

4.4.2　異なるバージョンの Elasticsearch を使用したクラスタ構成 (2 ノード バージョン 7.17.9, 1 ノード バージョン 7.17.6)

図 4.3 で示された docker-compose.yml において, es03 のコンテナで使用される Docker イメージを変更して, Elasticsearch のバージョンを 7.17.9 から 7.17.6 にダウングレードする.

図 4.4 に変更後の docker-compose.yml を図で表現したものを示す.

図 4.4　変更後の docker-compose.yml を図で表現したもの

変更後, docker compose up -d コマンドを実行してクラスタを起動する.

クラスタの起動後, curl コマンドを使用してクラスタに参加しているノードを一覧表示した結果をリスト 4.2 に示す.

リスト 4.2　クラスタに参加しているノードを一覧表示した結果

```
$ curl -X GET "localhost:9200/_cat/nodes?v=true&pretty"
ip heap.percent ram.percent cpu load_1m load_5m load_15m
    node.role master name
172.22.0.2 22 77 29 2.10 1.28 0.87 cdfhilmrstw * es01
172.22.0.3 24 77 27 2.10 1.28 0.87 cdfhilmrstw - es02
```

リスト 4.2 より, Elasticsearch のバージョンが 7.17.9 である 2 つのノード（es01, es02）のみがクラスタに参加できていることが確認できる.

es03 のコンテナで Elasticsearch の起動を試みた時に出力されたログを確認したところ, リスト 4.3 に示すエラーログを出力して起動に失敗していた.

リスト 4.3　es03 のログ

```
es03 | java.lang.IllegalStateException: cannot downgrade a
node from version [7.17.9] to version [7.17.6]
```

4.5 節　既存 Elasticsearch ノードの異なるクラスタへの参加の可否の検証

クラスタ構築において, 既に稼働している Elasticsearch ノードを新たなノードとして異なるクラスタへ参加できるか, Docker による仮想環境を用いて検証した.

4.5.1　単一ノードで稼働するクラスタ A の構築

まず, docker-compose を用いて単一ノード (コンテナ名は es04) でクラスタを構築する. 以後このクラスタをクラスタ A と呼ぶ.

図 4.5 にクラスタ A を構築する際に使用した docker-compose.yml を図で表現したものを示す.

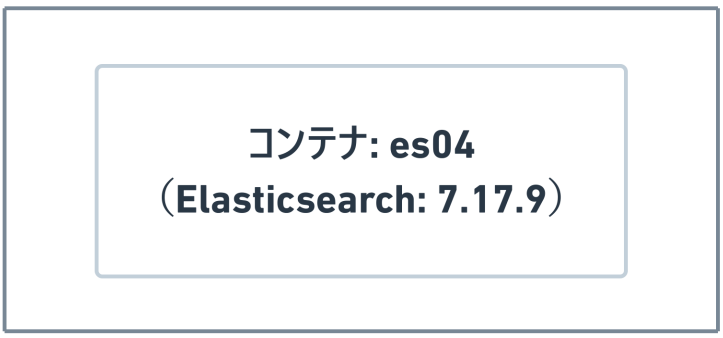

図 4.5　クラスタ A を構築する際に使用した docker-compose.yml を図で表現したもの

docker-compose を用いてクラスタ A を起動した後, クラスタの情報について問い合わせた結果をリスト 4.4 に示す.

リスト 4.4　クラスタ A の情報について問い合わせた結果

```
$ curl -XGET http://localhost:9200/
{
  "name" : "es04",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "NCg_UaW0Q6KTakJG5HJrHg",
  "version" : {
    "number" : "7.17.9",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef48222227ee6b9e70e502f0f0daa52435ee634d",
    "build_date" : "2023-01-31T05:34:43.305517834Z",
    "build_snapshot" : false,
    "lucene_version" : "8.11.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

クラスタの情報について問い合わせた後, Docker コンテナを停止してノードをシャットダウンした.

4.5.2　3 ノードで稼働するクラスタ B の構築

次に, クラスタ A の構築に使用したノードとは別の 3 ノード (コンテナ名はそれぞれ es01, es02, es03) でクラスタを構築する. 以後このクラスタをクラスタ B と呼ぶ.

図 4.6 にクラスタ B の構築の際に使用した docker-compose.yml を図で表現したものを示す.

図 4.6　クラスタ B を構築する際に使用した docker-compose.yml を図で表現したもの

docker-compose を用いてクラスタ B を起動した後, クラスタの情報につい

て問い合わせた結果をリスト 4.5 に示す.

リスト 4.5　クラスタ B の情報について問い合わせた結果

```
$ curl -XGET http://localhost:9200/
{
  "name" : "es01",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "2A69V1hxSFCisgmIo1LB_A",
  "version" : {
    "number" : "7.17.9",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef48222227ee6b9e70e502f0f0daa52435ee634d",
    "build_date" : "2023-01-31T05:34:43.305517834Z",
    "build_snapshot" : false,
    "lucene_version" : "8.11.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

リスト 4.4 とリスト 4.5 を比較した結果, クラスタ A とクラスタ B はそれぞれ異なるクラスタ ID を付与されたことが分かった.

クラスタ B の起動後, クラスタに参加しているノードの一覧を取得した結果をリスト 4.6 に示す.

リスト 4.6　クラスタ B に参加しているノード一覧

```
$ curl -X GET 'localhost:9200/_cat/nodes?v&pretty'
ip heap.percent ram.percent cpu load_1m load_5m load_15m
   node.role master name
172.30.0.4 22 88 28 5.99 2.99 1.72 cdfhilmrstw - es03
172.30.0.3 17 88 29 5.99 2.99 1.72 cdfhilmrstw * es02
172.30.0.2 36 88 30 5.99 2.99 1.72 cdfhilmrstw - es01
```

リスト 4.6 より, es01, es02, es03 ノードがクラスタ B に参加できていることを確認した.

クラスタ B に参加しているノードの一覧を取得した後, 全ての Docker コンテナを停止してノードを全てシャットダウンした.

4.5.3　クラスタ A に参加しているノードのクラスタ B への参加試行

次に, 図 4.6 の docker-compose.yml に対して, クラスタ A のノード (es04 コンテナ) を追加し, 合計 4 ノードでのクラスタ B の起動を試みる.

図 4.7 に, es04 コンテナを含めた 4 ノードでクラスタ B の起動を試みた際に使用した docker-compose.yml を図で表現したものを示す.

図 4.7　es04 コンテナを含めた 4 ノードでクラスタ B の起動を試みた際に使用した docker-compose.yml を図で表現したもの

クラスタ B の起動後, クラスタに参加しているノードの一覧を取得した結果をリスト 4.7 に示す.

リスト 4.7　クラスタ B に参加しているノード一覧

```
$ curl -X GET 'localhost:9200/_cat/nodes?v&pretty'
ip heap.percent ram.percent cpu load_1m load_5m load_15m
   node.role master name
172.30.0.4 22 88 28 5.99 2.99 1.72 cdfhilmrstw - es03
172.30.0.3 17 88 29 5.99 2.99 1.72 cdfhilmrstw * es02
172.30.0.2 36 88 30 5.99 2.99 1.72 cdfhilmrstw - es01
```

リスト 4.7 より, クラスタ A のノードがクラスタ B に参加できていないことを確認した.

es04 コンテナ（クラスタ A に参加しているノード）で出力されたログの一部をリスト 4.8 に示す.

リスト 4.8　es04 コンテナのログの一部

```
This node previously joined a cluster with UUID
[2A69V1hxSFCisgmIo1LB_A] and is now trying to join a different
cluster with UUID [NCg_UaW0Q6KTakJG5HJrHg].
This is forbidden and usually indicates an incorrect discovery
or cluster bootstrapping configuration. Note that the cluster
UUID persists across restarts and can only be changed by
deleting the contents of the node's data paths [] which will
also remove any data held by this node.
```

リスト 4.8 では, 異なるクラスタ ID を持つクラスタにノードが参加することは禁止されており, 参加するためにはインデックスやドキュメント情報などが格納されているデータパス配下のフォルダ, ファイルを削除する必要があると書かれている.

以上の検証結果から, 既に稼働しているノードを別のクラスタに新しいノードとして参加させることは出来ないことが分かった.

4.6 節　結言

本章では, 異なるバージョンの Elasticsearch ノードを用いたクラスタリングの動作の検証と, 異なるクラスタへの Elasticsearch ノードの参加の可否の検証を行った.

はじめに, 異なるバージョンの Elasticsearch ノードのクラスタリングを検

証した. まず同じバージョンの Elasticsearch ノードを用いてクラスタを構築した後, 異なるバージョンの Elasticsearch ノードを新たにクラスタに含めてクラスタ起動を試みたが, バージョンの不一致によりクラスタリングは失敗した. これにより, 異なるバージョンの Elasticsearch ノードを用いたクラスタ構築は出来ないことを確認した.

次に, 既に稼働している Elasticsearch ノードを異なるクラスタに参加させる検証を行った. まずクラスタ A とクラスタ B をそれぞれ構築した後, クラスタ A の Elasticsearch ノードをクラスタ B の Elasticsearch ノードとして参加させて, クラスタ B の起動を試みた. しかし, クラスタ A の Elasticsearch ノードはクラスタ B に参加できなかった. これにより, 既に稼働している Elasticsearch ノードを異なるクラスタに参加させることは出来ないことを確認した.

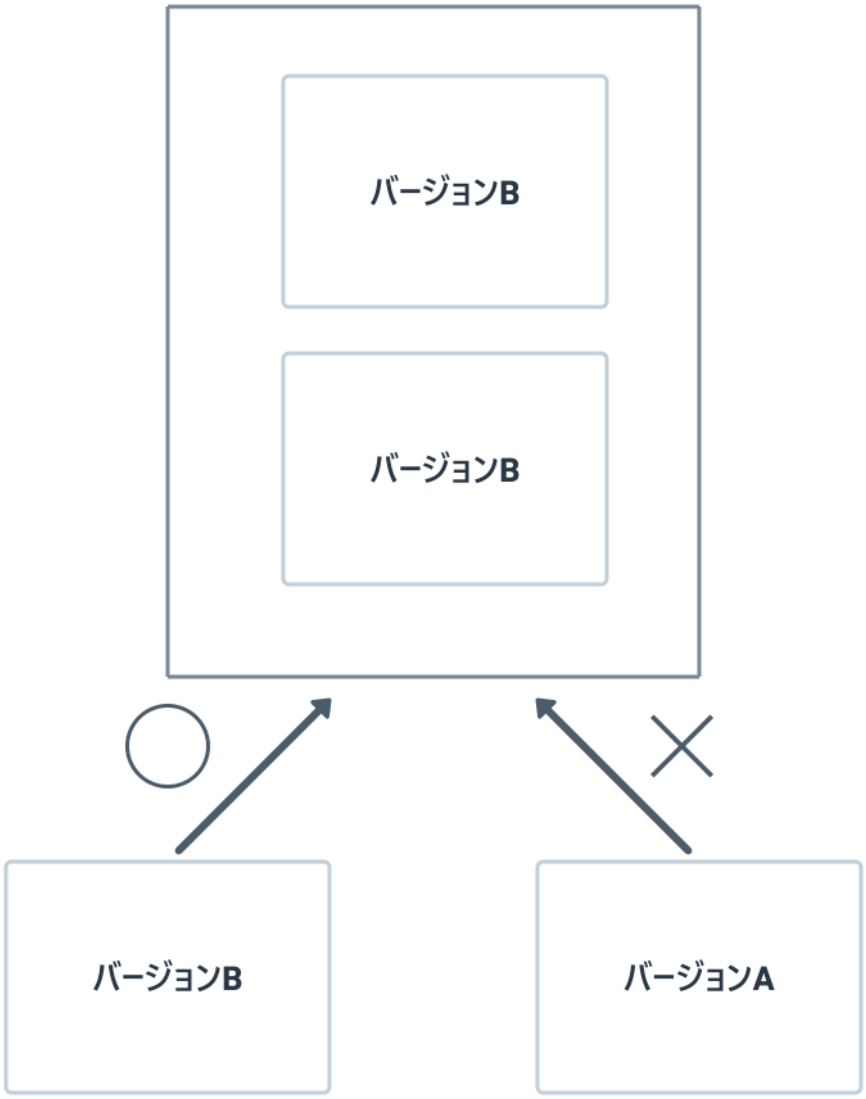

図 4.8　Elasticsearch ノードのバージョンが異なる場合のクラスタリングの可否

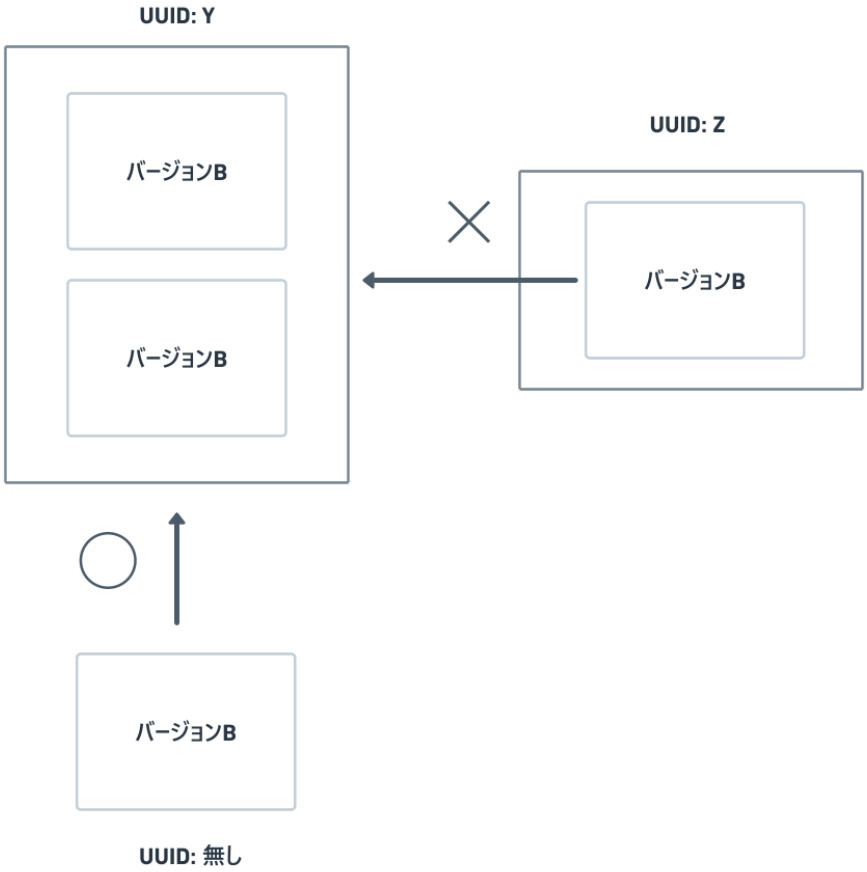

図 4.9　UUID が異なるクラスタに Elasticsearch ノードを参加させた場合のクラスタリング動作の可否

次章では結論と今後の課題について述べる.

第5章

結論と今後の課題

5.1節　結論

本研究では, 太陽光発電データの同期を目的として3種類の時刻補正手法を提案した. また, Elasticsearch ノードのクラスタ化手法を提案した.

太陽光発電データの計測時刻を大気外日射量との相互相関によって補正する手法を用いる場合に比べて, pvlib により求まる地表日射量を用いれば補正誤差を78秒から68秒に改善できた. さらに実測値のうち, 日の出と日の入時刻付近のデータを削除する前処理により, 補正誤差は68秒から34秒に改善したことを述べた.

CO_2 データを蓄積しているサーバには重複があるデータが多数存在していた. 4章で述べる Elasticsearch クラスタにデータ移行するに先立ち行った SQLite による重複削除機能が有効だったことを述べた. 本論文の場合では 338GB あったデータが削除後は 9.7GB に圧縮でき, 圧縮後のデータは必要以上に削除していないことを確認した.

4章では, Docker による仮想環境を使用することで, クラスタリング動作の可否の検証が効率よく行えたことを述べた. またその検証の結果, Elasticsearch ソフトウェアのバージョンや UUID の管理方法について得た知見を述べた.

5.2 節　今後の課題

今後の課題としては, pvlib に設定するパラメータをチューニングして地表日射量の計算値を実測値により近づけてから再度提案法を検証する必要がある. また, 得られた知見に基づきクラスタを実際に構築し, それを運用し続けるためのノウハウを蓄積していく必要がある.

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.

参考文献

[1] 中川清隆, “ 太陽方位、高度、大気外日射量の計算 ”, http://www.es.ris.ac.jp/~nakagawa/met_cal/solar.html, 参照 Feb 13, 2024.

[2] Sandia National Laboratories and pvlib python Development Team, “ pvlib python ”, https://pvlib-python.readthedocs.io/en/stable/, 参照 Feb 13, 2024.

[3] Elasticsearch B.V., “ Install Elasticsearch with Docker — Elasticsearch Guide [7.17] — Elastic ”, https://www.elastic.co/guide/en/elasticsearch/reference/7.17/docker.html, 参照 Feb 13, 2024.

[4] Docker, Inc., “ コンテナーとは？ — Docker ”, https://www.docker.com/ja-jp/resources/what-container/, 参照 Feb 13, 2024.

[5] Broadcom Inc., “ VMware Japan：クラウド、モビリティ、ネットワークとセキュリティ — JP ”, https://www.vmware.com/jp.html, 参照 Feb 13, 2024.