一般論文

データ解析システム開発の効率化に向けた生産環境の構築と処理性能改善

神戸事業所　開発部　開発第1課
河端 友理

1．まえがき

近年、プラント事業では、設備故障に備えて機器のデータを解析するシステムの導入を進めている。その目的は、設備状態に応じた保全対応を可能とし、信頼性確保と保全コストの削除や適正化を図ることである。

三菱電機(株)では2017年に、プラントから収集した時系列データの解析を行い、異常の兆候を検出するデータ解析システムの開発を行った。当社は、本システム開発で、三菱電機(株)のデータ解析ライブラリを利用した解析機能・解析実行機能の開発と、解析機能の実行結果を表示する結果表示機能の開発を行った。

本システムでは、機械学習向けのライブラリやフレームワークが豊富に存在するPython(注1)を採用した。本論文では、データ解析システムの開発で取り組んだ生産環境の構築と、開発時に発生した処理性能の課題・対策について述べる。

2．システム概要

本システムは、プラントから収集した温度や圧力などの実測値データを解析し、通常と異なる変化を異常の予兆として検知する。実測値が制限範囲内でもデータの変化の崩れを判断して異常を検知できることが特徴で、従来の制限値チェックによる検知と比べ、より確度の高い異常予兆が可能である。異常の検出例を図1に示す。

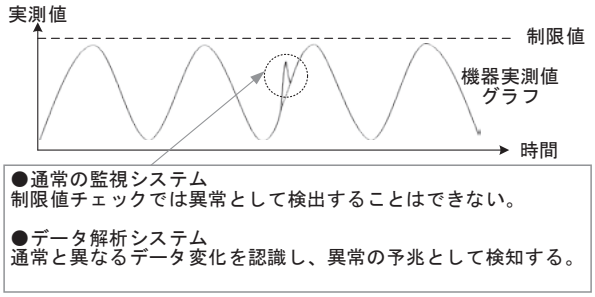


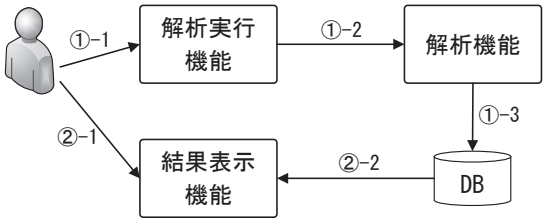
図1．異常検出例

データ解析システムは、主に表1の機能を提供する。

表1．データ解析システムの主な機能

機能名	説明
解析機能	時系列データを解析し、異常の兆候を検出する機能。 実行結果は、データベースに登録する。
解析実行機能	解析機能の実行と、実行状況の確認を行う機能。
結果表示機能	データベースに登録された解析機能の実行結果を表示する機能。

機能間の関連を図2に示す。



①「解析実行機能」を利用し、「解析機能」を実行する。
「解析機能」は、実行結果をデータベースに登録する。

②「結果表示機能」を利用し、データベースから取得した「解析機能」の実行結果を確認する。

図2．機能間の関連図

3．生産環境の構築

3.1　ソースコードフォーマッタ、静的解析ツールの導入

本開発では、Pythonを採用した開発を行ったが、使用実績が少なかったため、あらかじめコーディング規約を定めた。また、開発コードのコーディング規約への準拠率を高めるために、ソースコードフォーマッタと静的解析ツールを導入した。

導入した2つのツールの特徴と導入効果を述べる。

(1)ソースコードフォーマッタ(注2) AutoPEP8

(a)特徴

(i)Pythonの標準ライブラリに含まれているため容易に導入できる

(ii)Python標準のコーディング規約であるPEP8に準拠している

(注1) 汎用のスクリプト言語。数値計算などで使用される。
(注2) ソースコードをコーディング規約に準拠するように自動的に整形するツール。

(b)導入効果

AutoPEP8を導入することで、コーディング規約に準拠した可読性が高いソースコードが作成できた。またツールを適用していない評価用のソースコードと比較すると、ソースコードレビュー指摘率を66%、障害発生率を50%削減することができた。

(2)静的解析ツール(注3) PyLint

(a)特徴

(i)解析項目をファイル単位で一括して指定ができる

(ii)意味のない文、変数名の規則違反などの検出に対応している

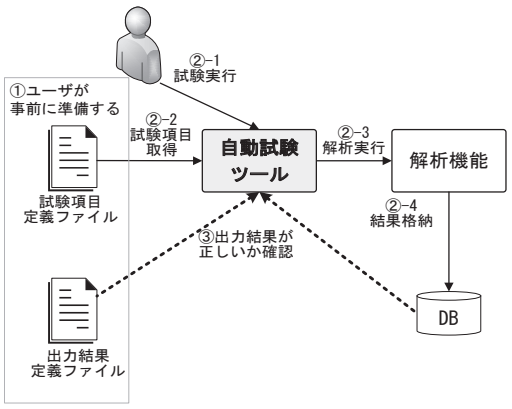
(b)導入効果

PyLintを導入し、早い段階で静的解析を実施して文法的もしくは構文的な不具合を修正することにより、習熟している他言語と同等の品質を確保できた。また、コーディング段階で修正が完結するため、ツールを適用していない評価用のソースコードと比較すると、DRでの文法的な指摘率を84%削減することができた。

3.2 自動試験ツールの整備

解析機能は、時系列データの投入状況により実行パターンが多い。そのため、機能追加や改修を行った場合の影響確認に30時間程度の時間を要する。作業時間削減のため、回帰試験を自動化するためのツール(以降、本ツール)を独自開発により整備した。

ツールの処理概要を図3に示す。



① ユーザは、事前に「試験項目定義ファイル」と「出力結果確認ファイル」を作成しておく。

② ユーザは、「自動試験ツール」を実行する。「自動試験ツール」は、「試験項目定義ファイル」から、試験内容を取得し、解析機能を実行する。「解析機能」はデータベースに実行結果を格納する。

③ 自動試験ツールは、データベースに格納された結果が、「出力結果定義ファイル」の内容とあっているかを確認する。

図３．自動試験ツールの処理概要

(注3) ソフトウェアの解析方法の一種であり、ソースコードを実行せずに行う検証のこと。ツールを利用して実施する。

本ツールは、試験項目ごとに試験項目定義ファイルと出力結果ファイルの定義を行うことで、様々な条件の試験を実施できる。

(1)試験項目定義ファイル

試験項目単位で、解析対象期間・対象信号を定義したCSVファイル。定義例を図4に示す。

```
試験No.1
対象期間,2015/8/1 0:00,2015/8/10 0:00
対象信号,SIGNAL1, SIGNAL2, SIGNAL3
試験No.2
対象期間,2018/1/1 0:00,2018/12/31 0:00
対象信号,SIGNAL4
```

図４．試験項目定義ファイルの例

(2)出力結果定義ファイル

確認したい値をデータベースから取得するためのSQL文を定義したファイルと、そのSQL文を実行した場合に得られる正しいデータを出力したテキストファイル。確認したいテーブルや値を試験ごとに変更することが可能である。定義例を図5に示す。

```
試験No.1
select * from tableA where …
select * from tableB where …
試験No.2
select * from tableC where …
select * from tableD where …
```

図５．出力結果定義ファイル（SQL文定義）の例

試験結果は、試験項目ごとに表形式でファイルに出力され、容易に確認を行うことが可能である。出力例を図6に示す。

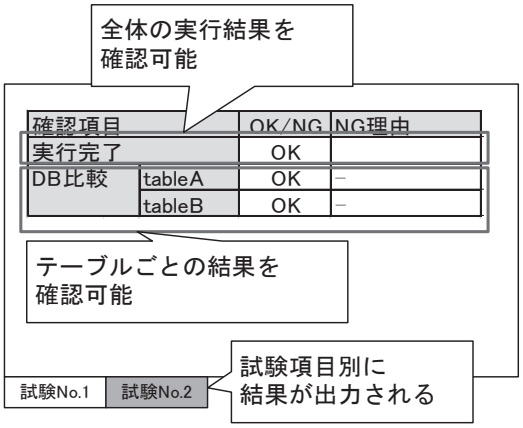


図６．試験結果出力例

本ツールの導入により、影響確認にかかる時間が、2つの定義ファイルの作成と結果確認の3時間のみとなり、ツール導入前と比較して作業時間を27時間削減することができた。

4．処理性能改善

4.1　課題

解析機能の性能要件を表2に示す。

表２．性能要件

項目	内容
計測対象PC	物理コア8コア ハイパースレッディング
信号数	200信号
処理時間	60秒以内

解析機能は新規性が高く、設計時に机上で処理時間を求めることが難しかった。そのため、性能評価用のプログラムを作成し、評価を行ったところ、表2の性能を満たさないことが判明した。

処理性能を分析した結果、性能要件を満たすためには3つの課題が存在した。

(1)処理フローの課題

解析機能の処理フローを図7に示す。

図7に示すとおり、信号ごとに逐次的に処理を行っているため、対象信号数の増加に比例して、処理時間も増加する。

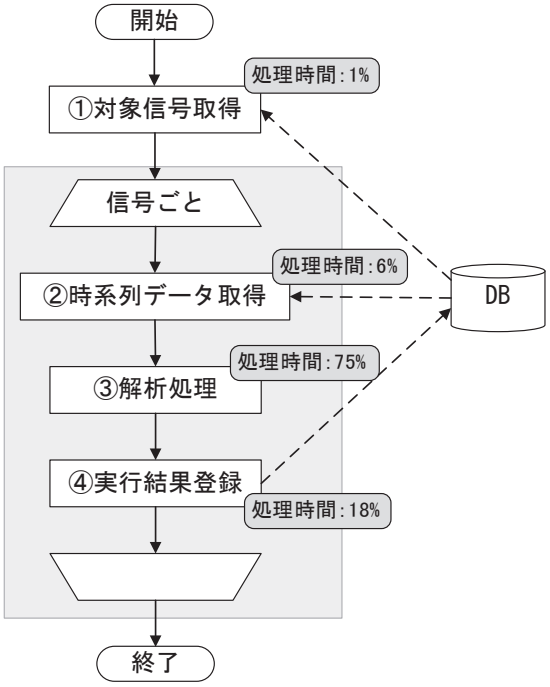


図７．処理フロー（改善前）

(2)データベースアクセス処理の課題

図7に示す処理時間は、全体の処理時間と比較した項目ごとの処理時間の割合である。データ解析ライブラリに依存しないデータベースアクセス(②時系列データ取得・④実行結果登録)に、全体の24%の処理時間を要している。

(3)Pythonの処理の課題

Pythonは、逐次呼び出しにより処理を行うスクリプト言語である。そのため、コンパイル時に最適化するコンパイラ言語と比較して、処理速度が遅い。

4.2　対策

4.1節で述べた課題を解決するため、3つの対策を実施した。

(1)並列処理の実現

Pythonは、処理をマルチコア化するためのライブラリを提供しており、並列処理を容易に実現できる。解析処理は、信号ごとに処理が独立しているため、ライブラリを利用して、並列処理を行う。

(2)データベースアクセス処理の改善

データベースへのアクセス処理を改善するために、複数信号一括して時系列データの取得と登録を行った場合に、どれくらいの処理時間が改善するか調査を行った。

表3に、200信号の10分データを処理した場合の削減率を示す。

表３．データベースのアクセス時間の削減率

一括信号数	処理時間の削減率(%)	
	取得	登録
10信号	87	89
50信号	96	97
200信号	98	99

表3より、多くの信号を一括して処理した方が、処理時間が削減できる。しかし、多くの信号を一括して取得すると保持するデータ数が多くなり、メモリエラーが発生する可能性が高くなる。保持可能なデータ数は、PCのスペックによって異なるため、一括で処理する信号数は定義ファイル化し、チューニング可能とした。

(3)Pythonの記載方法の見直し

Pythonには、高速化のための記載方法がある。代表的な2項目を紹介する。

(a)ループ処理の回避

Pythonに限らずスクリプト言語はforループ内部の処理を最適化できないため、高速に処理を行う必要がある

場合は避けた方がよい。

ループ処理を回避する方法として、高階関数(注4)mapを使用できる場合がある。mapは、配列の全ての要素に対してアクセスを行うため、forループの処理が不要となる。以下はどちらも配列に対して整数に丸めるround関数を適用している。

改善前:

```
# 要素に対して round 関数 を適用する
for idx in range(len(data)):
    result[idx] = round(data[idx])
```

改善後:

```
# 要素に対して round 関数 を適用する
result = list(map(round, data))
```

改善前と改善後のソースコードで、1億個のデータに対して処理時間を計測した結果、処理時間が24%削減した。

(b)内包表記を使用する

リスト生成を内包表記とすることで高速化できる。内包表記を利用することで、バイトコードに変更したときの命令数が少なくなり、高速な処理が可能となる。以下は、1～30の値のうち、3で割り切れる数値をリストに代入する例である。

改善前:

```
# 通常の書き方
for i in range(1, 30):
   if i % 3 == 0:
      result.append(i)
```

改善後:

```
# リスト内包表記
result
 = [i for i in range(1, 30) if i % 3 == 0]
```

改善前と改善後のソースコードで、1億個のデータに対して処理時間を計測した結果、処理時間が27%削減した。

(1)～(3)の対策を踏まえ、事前に一括処理信号数で信号をグループ化し、グループ単位でデータの取得、信号ごとに並列して解析処理、実行結果登録を行うように見直した。一度に並列可能な信号数は、PCのスペックによって異なるため定義ファイル化する。また、解析処理に対してPythonの記載方法の見直しを行った。

見直し後の処理フローを図8に示す。図中のNoは、図7に記述した番号と対応する。

(注4) 引数や、戻り値が関数である関数のこと。同じパターンの処理を抽象化できる。

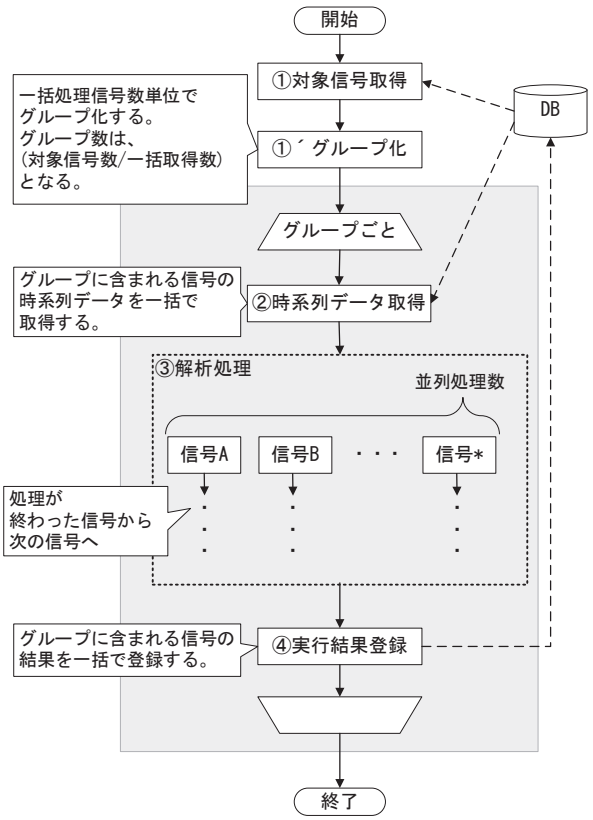


図８．処理フロー（改善後）

4.3 成果

対策の成果を確認するため、改善前後の処理時間の削減率を算出した結果を表４に示す。表中のNoは、図7、図8に記述した番号と対応する。なお、改善後の計測では、一括取得数は200信号、並列処理数は16にそれぞれ設定して計測した。

表４．計測結果

No	項目	処理時間の削減率(%)
①	対象信号取得	—(改善なし)
①´	グループ化	—(改善なし)
②	時系列データ取得	99
③	解析処理	98
④	実行結果登録	99
—	全体	98

各項目の評価を述べる。

①対象信号取得は、改善前後で差異無し。

①´グループ化の処理は、改善後に追加された処理であり、改善前より処理時間が増加した。

②200信号の時系列データをまとめて取得するため、処理時間が99%削減した。

③物理コア8コア・ハイパースレッディングのPCを使用する

ことにより、16並列で解析処理を動作させることができる。また、高速化に向けたPythonの記載方法の見直しを行ったため、処理時間が98%削減した。

④200信号の実行結果をまとめて登録するため、処理時間が99%削減した。

①´の処理で、改善前より処理時間が増加しているが、②～④の処理で処理時間を98%削減することができ、表2に記述した性能要件を満たすことができた。

5．むすび

本稿で紹介した取り組みで、Pythonを用いた開発の生産環境を構築することにより、効率的な開発を行うことができた。また、マルチプロセッシング技術を活用した処理性能改善を実現することができた。

Pythonは、日々進化を続けているため、さらなる情報収集と技術力習得に努めるとともに、生産性を改善する新たなツールを積極的に導入する所存である。

最後に、本開発に当たり、技術的な助言をはじめとして、様々な面でサポートいただいた関係者の方々に深く感謝申し上げる。

執筆者紹介

河端　友理　　カワバタ　ユリ
2009年入社。主にデータ解析システムのソフトウェア開発に従事。現在、神戸事業所開発部開発第1課。