EPICS Portable Archiver の開発

DEVELOPMENT OF EPICS PORTABLE ARCHIVER

路川 徹也#, A), 帯名 崇 B)
Tetsuya Michikawa #, A), Takashi Obina B)
A) e-JAPAN IT Co.,Ltd.
B) High Energy Accelerator Research Organization, KEK

Abstract

A variety of archiving solutions has been developed for EPICS-based control system. Most of them are aimed for large-scale experiments, and its setup and maintenance are not necessarily easy for small experiments. A portable archiver, which utilizes standard Python packages on Linux system, has been developed to realize simple installation and easy startup. In this paper, we report the design and implementation of the new portable archiver, and show the result of a preliminary performance test.

1. 概要

EPICS[1]を使用している制御システムでは、各種のデータを保存・閲覧するためのアーカイブシステムが何種類か存在しており、ユーザーが選択可能である。しかし、その多くは大規模データ向けであるため、設定や維持管理に手間がかかる場合があった。そこで、導入・設定・運用が容易なデータ収集システムとして、Python[2]をベースとした Portable Archiver を新たに開発した。本稿ではその設計から実装、パフォーマンス測定について述べる。

2. はじめに

EPICS を使用した大/中規模な加速器制御システムでは、膨大な測定/設定データを収集・閲覧するためにこれまでいくつかのアーカイブシステムが開発されてきた。各施設で使いやすいように独自開発する場合が多く、例えば KEKB では KBlog が使用されている。現在、比較的多くの加速器施設で使用されているのは CSS Archiver[3]や Channel Archiver[4] であろう。歴史的には独自のバイナリ形式でデータをファイルに保存する Channel Archiver が開発され、そのファイルを管理する多くのユーティリティが開発された。しかし、データ量が増えていくのに従ってファイル管理が煩雑になったり、インデックスファイルが巨大になったり破損したりするなどいくつかの問題があった。また、開発当時は 32bit OS であったためソースコードが 32bit 前提で書かれているという欠点もあった。そこでこれらの問題を解決するために CSS Archiver が開発された。最も大きな違いはデータ保存先としてファイルではなくリレーショナルデータベース（RDB）を採用したことにある。これによって標準的にネットワーク越しにクライアントアクセスが可能なことや各種のユーティリティが使用可能なこと、ファイルの切り替えが不要のため、データ欠損が起きない事など多くのメリットが得られた一方で、最大の欠点はファイルアクセスが遅く、長期間のデータ取り出しに時間を要することであった。そのほかにも、RDB 管理者がいるような大組織では問題にならないものの小規模で人数の少ない組織にとっては導入やパフォーマンスチューニング、その後の維持管理の負担が大きいことが大きな問題となっている。そのため、小さい組織では容易に導入できるとは言い難いのが現状である。

そこで、本研究では以前の Channel Archiver の使いやすさを維持しつつ最新の OS でも簡単にインストールから設定・維持できるようなシステムとして Portable Archiver を開発した。これは EPICS を使用した小規模システムや比較的短い期間で行う実験のデータ収集システムとして最適になることを目指して開発している。そのため、プログラムの設計にあたっては以下の方針を立てた：

(1) 動作環境は Linux を前提とする。ただし他の環境でも稼働できるように出来る限り Python2.7 の標準パッケージで動作するようにする。追加するパッケージは、EPICS 用の PythonCA[5]の他は pip で容易にインストール可能なものとする。
(2) データの保存方法として Channel Archiver や KBlog のような独自形式ではなく、ファイルベースのデータベース(DB)である SQLite3[6]を使用する。
(3) 設定ファイルには CSS Archiver と Channel Archiver で使われている XML 形式を採用し、タグも同じものが使用できるようにする。
(4) プログラムのインストールはソースコードのコピーのみで行えるようにする。
(5) CSS DataBrowser[7]からのデータ参照を可能にするため、クライアントとの通信には既存 Channel Archiver と互換の XML-RPC を使用する。

また、当面の目標性能としては、小規模あるいは短期間の実験をターゲットに、

- 接続レコード数: 5,000 以下
- 更新頻度: 5,000 件/秒
- クライアント数: ~10(最大)

程度を想定している。

hig-mchi@post.kek.jp

3. 実装

3.1 全体構成

プログラムの全体構成を Figure 1 に示す。

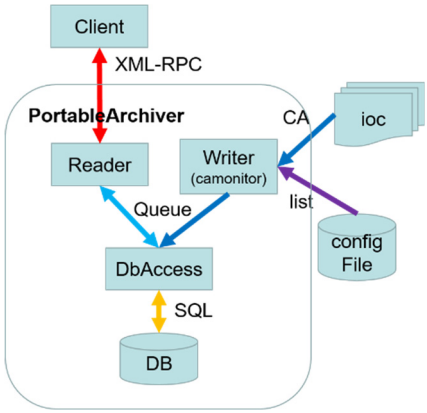

Figure 1: Portable Archiver block diagram.

プログラム内部は、主に3つのブロック(Reader, Writer, DbAccess)で構成されており、それぞれのブロックはスレッドとして動作している。

Reader はクライアントプログラムからのコマンドに対応し DB ファイルからデータを読み出してクライアントにデータを返す。Writer は EPICS IOC からデータを受け取って DB に書き込む処理を行い、DbAccess は Reader/Writer から DB への処理要求を一括して管理する機能を担っている。以下ではそれぞれの機能について詳細に記載する。

3.2 設定ファイル

保存したいデータ(EPICS レコード名)やファイルへの書き込み頻度など、各種の設定には XML ファイルを使用する。利便性と互換性を考慮して、CSS Archiver や Channel Archiver で使用しているタグの一部をそのまま使用する方針としており、これによって以前に使用した Channel Archiver の設定ファイルはそのまま使用可能である。Figure 2 に XML ファイルの例を挙げる。太字の部分が Portable Archiver で使用するタグである。

```
<?xml version="1.0" encoding="utf-8"?>
<engineconfig>
  <write_period>30</write_period>
  <get_threshold>20</get_threshold>
  <file_size>30</file_size>
  <ignored_future>1.0</ignored_future>
  <buffer_reserve>3</buffer_reserve>
  <max_repeat_count>120</max_repeat_count>
  <group>
    <name>NTP</name>
    <channel><name>CERL:OP:TEST</name>
      <period>1</period><monitor/></channel>
  </group>
</engineconfig>
```

Figure 2: XML file example.

3.3 DbAccess スレッド

DB への書き込み・読み込みを行うためのスレッドである。

データ保存に使用している DB である SQLite3 は単一ファイルを使用したファイルベース DB であるため、他の大規模 DB とは異なり、複数のプロセスやスレッドから同時にアクセスするとアクセスエラーが発生してしまう。そこでこのスレッドで DB 処理の逐次実行を行うことによって複数の Reader/Writer スレッドからの書き込み・読み込みを安全に実現している。

3.4 Writer スレッド

EPICS IOC からのデータを受信し、DB に書き込むためのスレッドである。Writer スレッドの内部構成を Figure 3 に示す。

プログラム起動時に1つの EPICS レコードに対して1つの callback を登録し、レコードの値が変更されるとすぐに callback 関数が呼び出される状態になる。callback 処理で受信したデータは書き込み用バッファリストに追加され、リスト内のデータ数が既定の数量(現在は 1000 固定)に達するか、設定ファイルに設定されている秒数(<write_period>)が経過した時に、DB への書き込みを実行する。バッファ用リストはダブルバッファリング構成としており、DbAccess スレッドがデータを読み出すバッファと、モニタースレッドがデータを追記するバッファとを切り替えることで、DB 書き込み中であっても callback 処理でデータ抜けが起きることを抑止している。

また、プログラム終了時には、バッファ内のデータを DB に書き込んでから終了するようになっている。

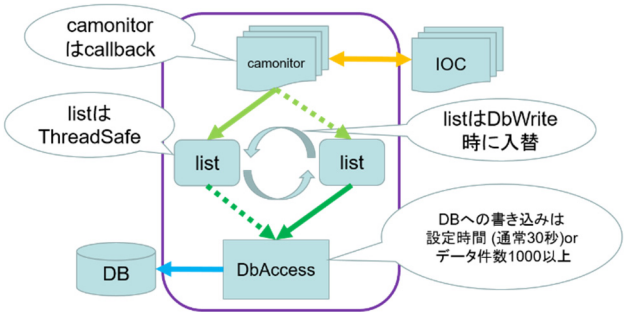

Figure 3: Writer thread block diagram.

3.5 Reader スレッド

クライアントプログラムからのコマンドを受信するためのスレッドである。Reader スレッドの内部構成を Figure 4 に示す。

アーカイブデータの取り出しクライアントには様々なソフトウェアが利用可能であるが、近年では CSS DataBrowser がよく使用されている。これは Java Application として Windows や Macintosh, Linux など多くのプラットフォームで実行可能なクライアントソフトウェアである。CSS ではサーバーとのデータ通信の方法として XML-RPC や DB に直接接続する JDBC 等の標準的な方法が用意されている。このほかに独自のアクセスプロトコル基づいて Data Connection Layer をプラグインとして開発することも可能ではあるが、今回開発した Portable Archiver では標準で実装されている XML-RPC を使用

する方針とした。

Readerスレッドは、クライアントからのデータ要求コマンドに対応して1つのスレッドを作成し DbAccess スレッドに要求を送る。この通信には、Queue を使用したコマンドの送受信を実現している。

Reader スレッド、DbAccess スレッド間の通信は、次の手順で行っている。具体的には、

(1) コマンド受信時にコマンド用オブジェクト(QueueObj)を作成し、受信コマンドをQueueObj に格納する。
(2) 作成した QueueObj を DbAccess スレッド内のコマンド Queue に追加し、Reader スレッドは応答待ち状態になる。
(3) DbAccess スレッドは、コマンド Queue から QueueObj を取り出し、そこに格納されているコマンドを実行する。
(4) コマンド実行後、その結果を QueueObj の Result に格納する。
(5) Result にデータが格納されると内部でイベントが発生し、Reader スレッドが応答待ちから復帰して、結果をクライアントに返すための処理を行う。

この処理で重要なのは、コマンド Queue は Reader スレッドから DbAccess スレッドへ QueueObj を渡すためだけに使っていることである。これによって、複数の Reader スレッドからのコマンドでも、コマンドの混線が起こらないようになっている。

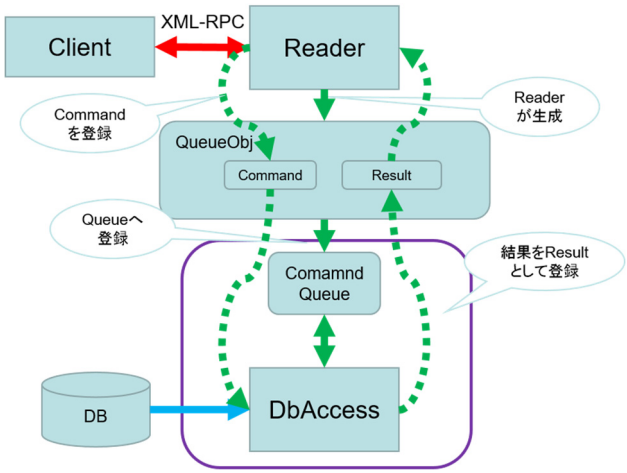

Figure 4: Reader thread block diagram.

3.6 データベース(DB)

DB ファイルはプログラム起動時に引数として指定する。既存のファイルが存在しない場合には新規ファイルを作成し、存在する場合はファイルをオープンして使用する。新規 DB ファイル作成時には必要なテーブルを自動的に作成するため、あらかじめ DB やテーブル作成等を行う必要はない。また、プログラム内の全ての DB データは追記のみを行い、更新/削除を自動では行わない。もちろん、オフライン状態にした後に手動で SQLite3 ファイルを編集することは容易に可能である。

また、DB に保存するデータはレコード値のみとし、現状では EPICS レコードが持っているアラーム情報(Severity)や単位(EGU)などの保存は行っていない。

DB 内では 2 つのテーブル、pv テーブルと archive テーブルを使用し、それぞれの定義は以下の様になっている。

3.6.1 pv テーブル

pv テーブルはレコード名を保存するテーブルである。テーブル定義を Table 1 に示す。id はレコード名に対して一意に設定され、番号はプログラム起動時に自動的に付加される。レコード情報は、レコード定義ファイルに設定されているものを使用する。既存の DB ファイルをオープンした場合、もし pv テーブルに存在しないレコード名が存在した場合にはそれらのレコード名だけ追加する。従って、仮にレコード定義ファイルからレコード名を削除しても、既存 DB ファイル中の pv テーブルからは削除されることはない。

DB ファイルを新規作成し、設定ファイルも変更された場合には、DB ファイル間でのレコード ID とレコード名の関係は保持されない。

Table 1: Pv Table Definition

名前	型	制約	説明
id	integer	Primary key	レコード ID
name	text	Unique	レコード名

3.6.2 archive テーブル

archive テーブルは、測定データを保存するテーブルである。テーブル定義を Table 2 に示す。id は pv テーブルの id とのリンク、time はレコードが process された時間、val はデータ値である。val に保存されるデータは、どのような型のデータでも全て文字列変換されたものを保存する方針とした。レコード値が waveform の場合には、val は”,”区切りの文字列として保存される。

Table 2: Archive Table Definition

名前	型	制約	説明
id	integer	index	レコード ID
time	real	index	Process Time
val	text	Not null	値

4. 性能テスト

今回の性能試験に使用したサーバーのスペックは以下の通りである

- 機種:Dell PowerEdge R630/02C2CP
- CPU:Intel Xeon CPU E5-2640 v4 @ 2.40GHz
- Mem:64GB
- HDD:NAS(10Gbps) or LocalDisk(SAS 15krpm)

4.1 Write 性能

書き込み性能を測定するため、固定周期で値が変化するソフトウェアレコードを作成して一定時間データを取得し、その後で保存したデータに欠損がないか検証した。

レコードの実行周期(SCAN PERIOD)と保存するレコード数を変化させながら以下の条件で計測を実施した。この測定では書き込み性能のみを測定することが目的であるため、並行したデータ読み出しなどの負荷がない状態での測定である。

- 測定条件
  同時接続レコード数:100,200,500,1000,5000
  Scan 時間[秒]:0.1,0.5,1
  データ型:ai

この測定条件では、scan 時間通りに 100%保存できた。性能目標である 5,000 件/秒は十分に達成している。

4.2 Read 性能

測定方法は、XML-RPC でデータを取得する Python スクリプトを作成し、保存データの読み込みに要した時間を測定した。

- 測定条件(1)
  DB ファイルサイズ[MB]:1,10,100,500
  data 数:10,100,200,500,1000,2000,5000
  10 回 Read 時の平均時間[秒]
  読み込みレコード名は毎回ランダムに変更

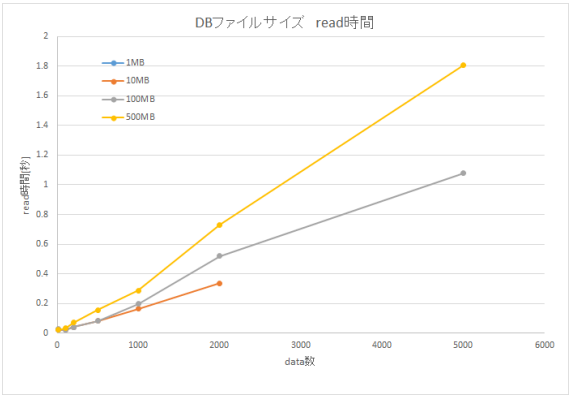

Figure 5: Read performance vs DB file size.

- 測定条件(2)
  同時接続クライアント数:2,5,10,20
  DB ファイルサイズ[MB]:500
  data 数:10,100,200,500,1000,2000,5000
  10 回 Read 時の平均時間[秒]
  読み込みレコード名は毎回ランダムに変更

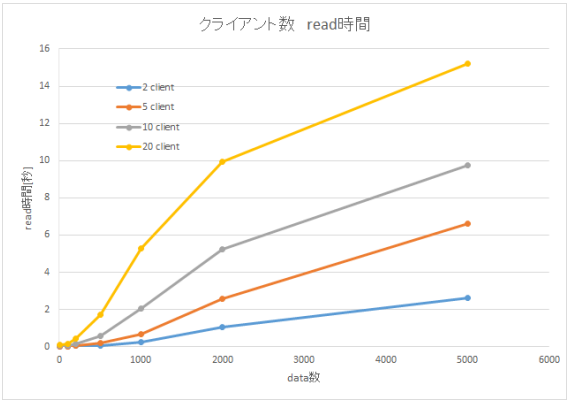

Figure 6: Read performance vs number of clients.

測定条件(1)では、読み込み data 件数と読み込み時間はほぼ比例して増加していることが Figure 5 からわかる。更に DB ファイルサイズが大きくなるほどデータの読み込み速度は遅くなる。

測定条件(2)では、同時接続のクライアント数に比例して、データの読み込み速度が遅くなることを Figure 6 で示している。この結果は、DB からのデータ読み込み処理が DbAccess スレッドで逐次実行していることが原因である。

5. 議論

性能評価の結果から書き込み・読み込みとも当面の使用には過不足無い性能を出せていることが分かった。しかし、Write 性能評価においては 5000record・0.1 秒でテストした際、書き込み処理が追い付かずにバッファに溜まっているデータ数が増加の一途をたどり、いつまでも減らない状態が発生していた。これは、Portable Archiver が EPICS IOC からの受信データの全てを一つのデータバッファに格納する構造になっているのが原因の一つであると考えられる。接続するレコード数が多かったり、レコードの SCAN PERIOD が短くなったりすると、DB への書き込み速度よりバッファに溜まるデータ量が多くなり、最終的には書き込みができなくなる状態が発生する。Channel Archiver や CSS Archiver は、各レコードに最大バッファ数を設定し、バッファからあふれた場合にはデータを破棄して、Buffer Overrun を知らせるという方法をとっている。現状では、Portable Archiver 単体ではこの問題に対処する方法が無いが、複数の Portable Archiver を起動することで、負荷を分散することは可能である。今後の開発ではこの問題をどう解決するかを検討する必要がある。

上記の問題も踏まえて、現在のパフォーマンスを制限している原因調査をより詳細に行うことが必要である。考えられるボトルネックとしてはファイル I/O、ソフトウェア内のキャッシュ、ネットワーク性能など様々な原因があるため、1つ1つのプロファイリングを行い、必要に応じて対策を講じる。

現状でもデータ書き込みと並行して読み出しを行うことは可能であり、体感速度としては全く問題無いことは確認しているが定量的な測定はまだ行っていない。今後は

負荷の有無による書き込み・読み込み性能の劣化について評価したいと考えている。

ファイルサイズがさらに大きくなってギガバイト級になった場合、書き込み・読み込みに要する時間が極端に増える可能性がある。この場合、ファイルを複数 DB ファイルへ分割したり、インデックスを作成したりするなどで対策する可能性もあるが、そのような状況は他の大規模アーカイバーを採用すべき事案であると考えている。

本来ならば既存のアーカイブシステム（Channel Archiver, CSS Archiver, Archiver Appliance)等との速度比較も行うべきであるが、これらのアーカイバーは Portable Archiver とは用途が異なることや、ハードウェア性能、ソフトウェアのチューニング等で性能が大きく変わるため単純な比較は困難である。他のアーカイバーの性能試験は参考文献[8][9]等を参照して頂きたい。

6. まとめと今後の課題

6.1 まとめ

導入・設定・運用が容易なデータ収集システムとして Python2.7 をベースとした Portable Archiver を新たに開発し、性能評価を行った。開発したアーカイバーは例えば PF の冷却水温度モニターに使用している他、マシンスタディのときに一時的に設置した装置や計算レコードなどを保存するなどの目的に使用している。これまでデバッグを兼ねて開発を進めてきており、現在では実用上問題のないアーカイブシステムとして利用可能な状態である。また、Python さえあれば簡単にインストール可能であるため、名刺サイズのシングルボードコンピューターである Raspberry Pi [10] でも動作可能である。この場合はファイル I/O 速度や容量の問題はあるもののレコード数が多くなければ実用上問題無く使用可能である。

6.2 今後の課題

今後の開発の方針はいくつか考えられる。例えば、他のアーカイブシステムでは標準的に搭載されている EPICS のアラーム情報(Severity)や単位(EGU)なども合わせて保存する機能などを追加する可能性はあるが、このような多機能化へ向けた方向は本来の開発方針であった“Portable”と反する場合があるため慎重に検討する必要がある。ユーザーからの要望をみながら今後の開発方針を決定していきたいと考えている。現段階で優先したいと考えている開発項目は以下の通りである。

- Web 管理画面の作成
  現在はコマンドラインのみなので、稼働状況が少しわかりにくい。
- Python3 対応
  現在は Python2.7 のみだが、今後 Python2 系のサポートがなくなった時を考慮すると、Python3 にも対応できるようにしたほうがいいと考えている。
- Reader への read cache 導入
  同時接続するクライアント数が多くなった場合に、キャッシュを用意することで DB へのアクセスを低減し、応答速度の改善が期待できる。
- 他の Channel Access モジュールへの対応
  現在対応している EPICS 用 python モジュールは PythonCA のみであるが、PythonCA は Python3 に対応していない。今後の事を考えると、他の EPICS 用モジュールが使用できるように修正しておく必要がある。候補として、PyEpics[11]を検討している。
- 他の保存方式への対応
  SQLite3 は基本的に ASCII 形式での保存であり、データ量が増えたときにファイル I/O 速度が遅くなる。独自ファイル形式という方向性もあるが、近年大規模時系列データの保存で使用されるようになった HDF5 を採用することで汎用性を確保しつつ高性能化出来る可能性があるため検討している。

最後に、今回開発したソフトウェアは配布可能である。現在は KEK 所内ネットワーク内に設置した gitlab で管理しているが、興味を持った方は著者まで連絡するか EPICS 開発者向けのメーリングリスト（epics-users@ml.post.kek.jp）あるいは 日本語 EPICS 情報サイト wiki（http://cerldev.kek.jp/trac/EpicsUsersJP）などを参照して頂きたい。

参考文献

[1] Experimental Physics and Industrial Control System; http://www.aps.anl.gov/epics/
[2] python 2.7; https://www.python.org/
[3] Control System Studio; http://controlsystemstudio.org/
[4] Channel Archiver; https://ics-web.sns.ornl.gov/kasemir/archiver/
[5] PythonCA; http://www-acc.kek.jp/EPICS_Gr/products.html
[6] SQLite3; https://sqlite.org/index.html
[7] CSS DataBrowser; https://github.com/ControlSystemStudio/cs-studio/wiki/DataBrowser
[8] Takuya Kudou *et al.*,” PRESENT STATUS OF CSS ARCHIVER AT KEK INJECTOR LINAC”, Particle-Accelerator-Society-2015, Tsuruga, Japan, August 5 - 7, 2015; http://www-linac.kek.jp/linac-paper/2015/pasj2015/pasj15-kudoh-wep113.pdf
[9] J. Rowland, M.T. Heron, S.J. Singleton, K. Vijayan, M. Leech, Diamond Light Source, Oxfordshire, UK,” ALGORITHMS AND DATA STRUCTURES FOR THE EPICS CHANNEL ARCHIVER”, Proceedings of ICALEPCS2011, Grenoble, France; https://accelconf.web.cern.ch/accelconf/icalepcs2011/papers/mopkn006.pdf
[10] Raspberry Pi; https://www.raspberrypi.org/
[11] PyEpics: Epics Channel Access for Python; http://cars9.uchicago.edu/software/python/pyepics3/