SPring-8 次期計画における加速器時系列データ保存基盤の設計

A PLAN OF TIME-SERIES DATA ARCHIVING INFRASTRUCTURE AT THE SPring-8 UPGRADE

岡田謙介#, A), 丸山俊之 B), 福井達 B)
Kensuke Okada #, A), Toshiyuki Maruyama B), Toru Fukui B)
A) JASRI
B) RIKEN

Abstract

At SPring-8/SACLA, time series data such as environment measurements, sensor outputs synchronized with the accelerator cycle, and machine status are stored in a RDB-NoSQL-combined scheme. We plan to keep the same concept into the SPring-8 upgrade. However, the database engine needs to be upgraded, as the current base component (e.g. operating system, Java) and the support for the DB itself will be outdated. A known issue is that Apache Cassandra Version-4 no longer support the Thrift protocol which forms the lowest level of our database library used in both DAQ and API. In this presentation, we report the feature of the native driver (DataStax Driver) and how it can be adapted to the new database library. During the transition period, it might be required to operate both the old (current) and the new method. We think we can get through this likely scenario with some small touch-ups.

1. はじめに

SPring-8/SACLA、NanoTerasu 他において、測定データ、状態値を機器横断的に時系列で保存し、加速器調整に用いている。加速器時系列データ保存基盤では、個々のデータに ID を割り当て RDB で管理し、需要の多い直近の時系列データの保存は Apache Cassandra（=カラムベース NoSQL DB）に役割を与えている[1,2]。

将来の長期運用を考えたとき、老朽化に伴う機器更新で、古い OS や JAVA のバージョンの維持が難しくなることや、データベースのサポート期限切れの問題への対処のため、データベースエンジンを機会毎にアップデートすることが必要である。現行の構成は Cassandra version2 のもとで設計され、2020 年ころから順次 version3 に置き換えた。今後特に問題になるのは Cassandra の方向性として、client library が初期の Thrift protocol のサポートを廃止し、CQL-based native driver に移行することである[3]。現在 GUI やデータ収集などのアプリケーションは Thrift protocol ベースのため、DB アクセスライブラリの改修が必要となる。

本発表では native driver (以下 DataStax driver)[4]を組み込み、古い設計の見直しも含めてライブラリ改修の検討をした結果を報告する。

SPring-8 ストレージリング（SR）アップデート計画として 2027 年夏からのシャットダウンが予定されており、この新しい設計はその前後の投入が当面のターゲットである。一方入射器の SACLA は、XFEL 加速器として SR 建設中もユーザー向けに稼働するため、運転継続性を考慮した段階的な導入計画が必要である。

2. 現行のデータベース設計

加速器時系列データとして取り扱っているデータ型と信号数を Table 1 に示す。過去の発表[2]の第二世代が現行の設計である。

Table 1: Signal Types and #signals as of 2023

Type	Contents	#siganls (XFEL+SR)
pol	Point data (< 5Hz)	82k
sync	Point data with event number	6k
array	1D data (packed)	1
wfm	1D data (meta data on the DB)	1.3k
Img	2D data (meta data on the DB)	36

今回は登録信号数で主な pol, sync について述べる。データベーステーブルはそれぞれ cf_log, cf_data の二種類を用いており、それぞれの役割を Table 2 に示す（テーブル構成の詳細は[1]）。各データ収集ホストの上で稼働する DAQ application の接続は、Thread 毎に Cassandra の 1 ノードに割り当て、cf_data, cf_log の 2 つのテーブルへの書き出しを複数信号分まとめて batch 扱いとし、まとめてデータの書き込みを行っている。

Table 2: Cassandra Tables for ‘pol’ and ‘sync’ Signals

Table	Main Contents	Usage	Unit
cf_log	id, date, value(update)	Newest, Date index	-
cf_data	id+date, time, value	Log data	Month
cf_log_sync	id, date, evno value(update)	Newest, Date index	-
cf_date_sync	id+date, time, evno, value	Log data	Month

k.okada@spring8.or.jp

3. Apache Cassandra バージョン 4 に向けての検討項目

3.1 DataStax driver による Cassandra ノード状態把握機能の活用

Thrift 接続では、基本的に client はノードの一つと結びつけ、接続先ノードの障害時に別の正常なノードへ切り替える処理を DB ライブラリ側で独自に用意している。一方 DataStax driver は全てのノードにコネクションをはる。driver 側でノードの死活を把握し、自動的に障害ノードへのリクエストを回避する。

3.2 batch 書き込みの廃止

複数のノードで構成される Cassandra では、ノードの保持担当範囲はデータの key (正確には partition key と呼ばれる)によって振り分けられる。従って書き込みの際、異なる key を batch で束ねている現状は、ノード分離の効率の面からはアンチパターンとされているものである。cf_log と cf_data の結合を弱めることで、batch を使わない方向を検討する。

3.3 DataStax driver の非同期読み出しによる処理速度向上

Thrift 接続では、client に並列処理が求められる場合は他のノードと紐づけた別 thread を立てることで対応している。DataStax driver では非同期リクエストにより並列処理を実行可能で、client 側が非力な場合でも処理速度向上が見込まれる。

4. 試験

4.1 DataStax driver による Cassandra ノード状態把握機能

DataStax driver の接続は基本的には全ノードにコネクションをはり、各ノードが順番に書き込み読み込みの命令を担当する。ノード障害時は[5]の「Idle disconnect and reconnection」ポリシーで扱われる。

常に運転状態の加速器運転に向けて、Cassandra のノード故障の発生時の影響を懸念した。テスト環境において 9 ノードクラスターに複数の書き込みプロセス、読み込みプロセスが接続されているセットアップ(Fig. 1)で、ネットワークケーブルの物理的取り外しでノード故障を模擬した。

ここで明らかになったのは、運転の継続性に関して Thrift 接続で運用してきた1ノード接続の有効性である。この場合、ノード障害の際にクラスターを構成するノード数に反比例して影響を受けるプロセスが減るのに対し、

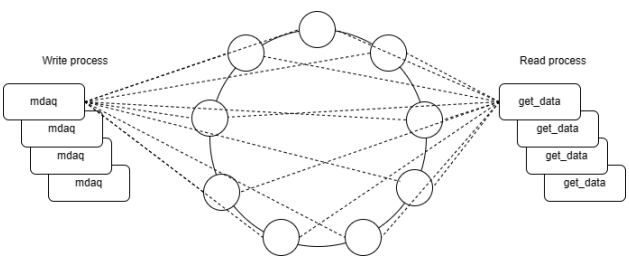

Figure 1: A concept of DB connection in DataStax driver

DataStax driver の接続は全てのプロセスがノード障害の影響を被り、一定時間の断絶が発生する。この問題のため、特に書き込みプロセスでは DataStax driver の有効ノードプールの管理機能の使用は諦め、これまでと同様に 1 ノード接続を強制し DB ライブラリ側において自前で復旧ロジックを組み込むことにした。読み込みに関しては、後述の同時接続による速度向上を見込み、ノード障害の影響を理解した上で限定的に活用する。

4.2 batch 書き込みの廃止

batch 書き込みを使用してきた理由の一つは cf_log と cf_data の 2 つのテーブルの整合性を担保するためであった。cf_log で最新の値を常にアップデートすることで、最新値読み出しの速度向上を図った。ただし、これは Cassandra version2 までのことで、version3 からはデータ保存のフォーマットが変更され[6,7]、cf_data のログデータからの最新値取得の速度改善がみられる。Figure 2 は最新値読み出しの速度測定結果である。single thread で 800 から 2500 信号の読み出し時間を測定し、1 信号あたりの読み出し時間を実行時刻を横軸としてプロットした。cf_data の key は日付を含むため、日をまたいだ時点で新規 key を作成し、日が終わるとき key にひもづくデータが最大になる。まず cf_log からの読み出しは version、実行時刻に関わらず一定である。cf_data からは version2 については実行時刻による違いが現れており、データ量が小さいほど読み出し速度が速い。一方 version3 について(Fig. 2 の★)は実行時刻に関わらず一定で、cf_log からの読み出しと遜色ない速度となっていることがわかる。

これにより、version 3 から変更されたデータ保存フォーマットの下では、あえて最新値を cf_log から読み出す必要はなくなり、batch で cf_log と cf_data の整合性を担保する必然性はないことが分かった。

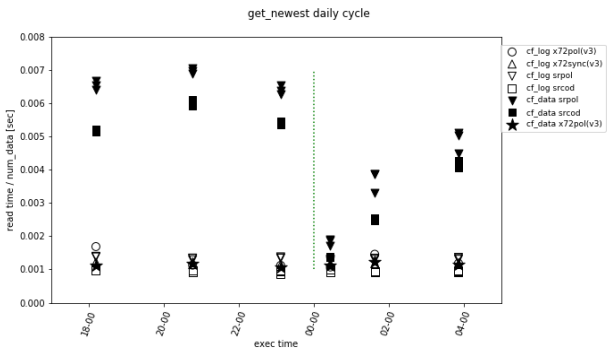

Figure 2: Read speed of the latest value as a function of execution time of day.

ところが、書き込みの通信負荷を調べると、異なる key(信号)を束ねる batch に対して、個別の query を投げる方式は通信負荷が高いことが分かった。Figure 3, 4 は 14 信号を 10sec 周期で様々な方法で書き込んだ場合の通信負荷を示している。ここから、batch 書き込み(Fig. 2 a,d)が最も省データで Thrift 方式と DataStax の違いは見られない。また Fig. 2 b, c の比較で cf_log と cf_data の二つのテーブルで 2 倍の負荷となり、Figure 2 c,e の比較では prepared statement の利用で通信量の節約が見込まれることがわかるが、いずれの通信量も高い。パケッ

トキャプチャで確認すると、個別の query を投げる方式はその度にオーバーヘッドが発生することが分かった。これまで batch 方式での Cassandra 側負荷による不具合はみられてないことから、batch 方式は継続することにした。

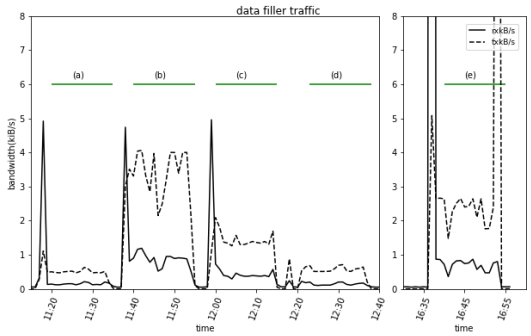

Figure 3: Network load (kiB/s) in various options. The measurement was done with 14 signals of 10 second cycle. 5 periods (a to e) are with the following conditions.

(a) v4, Datastax whitelist node1
batch (async), cf_log & cf_data
(b) v4, Datastax, whitelist node1
every cf_log, cf_data, with prepared statement
(c) v4, Datastax, whitelist node1
cf_data only, with prepared statement
(d) v3 Thrift
batch, cf_log & cf_data
(e) v4, Datastax, whitelist node1
cf_data only without prepared statement

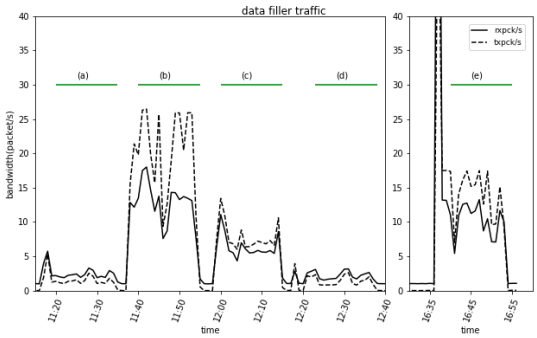

Figure 4: Network load (packet/s) in various options. The conditions are the same as Fig. 3.

4.3 DataStax driver の非同期読み出しによる処理速度向上

ここでは DataStax driver の全 72 ノード接続による、非同期読み出し(Async request)の処理速度測定結果を示す。Figure 5 は 4371 点の並列読み出しの速度(10 回の平均)である。従来の Thrift 接続による読み出しは cf_log, cf_data で違いがみられないのは前述の通り。そして Multi-thread の処理で速度向上がみられる。これに対し DataStax driver の非同期読み出し(Async reques)で全 72 ノードを対象にするとほぼオーダーの速度上昇がみられる。図の左右で縦軸が 10 倍違うのに注意されたい。

コーディングも素直になることもあり、前述の1ノードの障害で一定時間の断絶が起こることを考慮に入れた上で、処理速度向上の点で採用する利点はあると判断した。

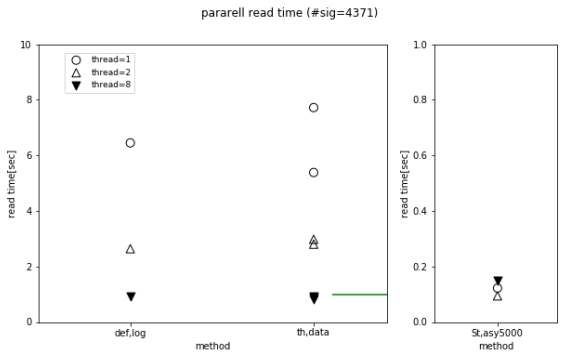

Figure 5: Parallel readout time of 4371 signals.

5. 新設計骨子

これまでの検討を踏まえ以下の方針で DataStax driver 対応の DB アクセスライブラリを作成することにした。

5.1 DB 接続の管理

以下の 2 種類の session タイプを用意する。
A. 1 node 限定(whitelist 方式)
B. full node

A は書き込みと通常の読み込みに使用し、従来の Thrift 版と同様に障害等で接続が切れた場合に正常な別ノードに切り替える。この目的で新たにノード情報管理機能を実装する。B は並列読み出し処理用に使い、初回利用時に接続を確立する。

5.2 Filler(書き込み)

batch の利用は key が混在する点で Cassandra のアンチパターンと言われるが、速度的に問題がない上、逆に batch を使わないと通信量が大幅に上がるデメリットがあるため、従来通り使うこととする。

session はノード障害によるデータ収集への影響を限定する目的で whitelist を使用した 1 node 接続を利用する。

現在 cf_log(インデックス的な役割と最新値保持), cf_data(時系列ログデータ)の両方にデータを書き込んでいるが、後述の読み込み方式の変更により両者を整合させる必要は薄くなる。cf_log の書き込み頻度を調節できるノブを用意し、将来例えば 100 秒に一回程度に減らす。一時的な理由での書き込み失敗時のリトライは cf_data のみで良い。

また cf_data_sync にカラムを追加し、packed data を展開せずに直近の代表値を取得できるように効率化する。

5.3 API(読み込み)

基本読み込みと並列読み込みで読み込みロジックを切り替える。基本読み込みでは書き込みと同様に session タイプ A を用い、ノード障害に対しての影響を局在化する。一方並列読み込みでは、session タイプ B を

用いDataStax非同期呼び出しを用いて処理速度に重点を置く。

以下に各読み込み方式の要点を記述する。

5.3.1 基本読み込み

cf_data からの読み出しを基本とし、2 日以上過去のデータを参照する場合に cf_log を利用する。cf_data_sync は packed data を展開せずに直近の代表値を得る。

直近値の読み出し(get_newest):

まず当日分の key で検索し、時刻ソートにより最新値を取得する。深夜0時の日の切り替えのタイミングでデータなしの場合があるため、一日前の key を試す。それでもデータなしの場合、cf_log を参照する。archive data 読み込みとの連携は不要。

範囲指定読み出し(get_data):

- 時刻範囲指定による取得
  cf_data をもとに読み込みを行う。
- 取得データ数指定による取得
  時刻起点の場合とイベント番号起点の場合が存在する。イベント番号起点の場合はまず時刻起点に直してから読み出す。

5.3.2 並列読み込み

DataStax Driver の非同期呼び出しを利用して、信号毎に並列でデータを取得する。現在 online data の範囲外の場合に自動で archive data 読み出しに接続する仕様となっているが、本仕様では online data のみの参照に限定し、archive data 参照は切り分け別関数とする。

直近値並列(get_newest_multi):

当日 key を用いた cf_data 読み出しを行い、日をまたぐタイミングでのデータ欠損について、前日 key で読み直す。

範囲指定並列(get_data_multi):

- 時刻範囲指定による取得
  cf_data をもとに読み込みを行う。

- 取得データ数指定による取得
  時刻起点の場合とイベント番号起点の場合が存在する。イベント番号起点の場合はまず時刻起点に直してから読み出す。

時系列読み込み:

時系列で分割し、非同期呼び出しを用いる。

範囲指定間引き読み込み(get_data_scan):

長期間の時間範囲のデータを、適当な間引き間隔で取得するための関数である。 指定時間範囲を時間点 $[t_0,t_1,\ldots,t_N]$ に分割し(ここで $t_{n+1}-t_n=\Delta t$, Δt は指定間引き間隔)、それぞれの時間点の近辺のデータ取得を非同期読み出しで行う。その後、追加の条件(sync データのイベント番号に対する条件等)を適用して最終的な戻り値を得る。指定間引き間隔により、データ拾い出しのロジックを切り替える必要がある(例えば二時間の指定なら分単位でそろえれば良いが、10 秒の指定ならば秒単位でそろえるなど)。

時刻列整列読み出し(get_data_multi_align):

リファレンス信号のデータ列の時刻とイベント番号に合わせ、複数信号の間引きデータを取得する。上述の get_data_scan と似たロジックだが、信号単位でも並列化する。

6. 移行のシナリオ

SPring-8 アップグレードで、ストレージリング(SR)のデータ収集は新規 Cassandra Ver.4 以降を採用する良い機会であるが、SACLA データ収集は Cassandra Ver.3 のまま SR シャットダウン前後をまたぐのが現実的である。Cassandra version と library の新旧が混在する状況を想定し、Table 3 に互換性を示す。

収集系について、現行の旧 library に同期収集の新カラム書き出しを追加し、新 library は現行と同じように当面毎回 cf_log へ最新値の書き出しを行うようにすれば、混在環境での運用が可能になると考えられる。

Table 3: New/Current Libraries Compatibility Matrix

Cassandra	Filler library	API library	Conditions
Ver.3.x	Current(Thrift)	Current	NP-
Ver.3.x	Current(Thrift)	New	Adapt to new cf_data_sync
Ver.3.x	New(DataStax)	Current	*1) Need to write all cf_log *2) Adopt to the new format Adapt to new cf_data_sync
Ver.3.x	-	New/Current	NP(host list columns)
Ver.3.x	New(DataStax)	New	NP
Ver.4.x or later	New	New	NP

7 まとめ

SPring-8/SACLA の加速器の状態やセンサーの時系列データの利用のためのデータ保存基盤(リレーショナルデータベースと NoSQL:Cassandra の組み合わせ)の基本方針は、SPring-8 次期計画でも引き継ぐ予定である。そのためにはハードウェアの更新に追従し、データベー

スエンジンの更新も必要である。特に Cassandra の次期バージョンでは、Thrift プロトコルのサポートが廃止となり、データベースアクセスのための基本ライブラリの更新が必要となる。今回 CQL-based native protocol (DataStax driver)の動作を確認し、24 時間利用運転のための高信頼性を保ちつつ、より高速な処理ができるように設計の見直しを行った。

移行期間は現行の旧ライブラリと併用する場合が考えられるが、新旧とも軽微な修正で実現できる見込みである。

参考文献

[1] K. Okada, T. Maruyama, T. Fukui, “SPring-8/SACLA 加速器ログデータベース利用環境の構築”, 第 18 回日本加速器学会年会 (2021) プロシーディングス, TUP039.
[2] K. Okada, R. Fujihara, T. Maruyama, “加速器データログのための NoSQL データベース(Apache Cassandra)安定運用”, 第 17 回日本加速器学会年会(2020)プロシーディングス, THPP23.
[3] 例えば Cassandra 3.x High Availability second edition, PACKT publishing Chapter 6
[4] DataStax C/C++ Driver – Home, https:// docs.datastax.com/en/developer/cpp-driver/2.16/
[5] Reconnection policies::Developing applications with DataStax drivers, https://docs.datastax.com/en/devapp/doc/devapp/driversReconnectionPolicies.html
[6] Apache cassandra – SSTable version, https://docs.datastax.com/en/home/docs/compatibility.html
[7] https://distributeddatastore.blogspot.com/2020/03/cassandra-new-sstable-storage-format.html