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

A STABLE OPERATION OF NOSQL DATABASE （APACHE CASSANDRA) APPLIED TO THE ACCERELATOR DATA LOGGING

岡田謙介#, A), 藤原綾潜 A), 丸山俊之 B)
Kensuke Okada #, A), Ryosen Fujihara A), Toshiyuki Maruyama B)
A) JASRI
B) RIKEN

Abstract

At the SPring-8/SACLA complex, several NoSQL database (Apache Cassandra) clusters have been utilized for storing log data. For various types of signals, it is essential to couple the NoSQL database with a relational database to build a complete data stream. Since the first generation of the system implemented in 2014, an overhaul was already made in 2018. In this paper, we describe the essence in the database design and troubles we got through in the 24-hour operation for a future occasion implementing a similar system.

1. はじめに

大型放射光施設 SPring-8 では、初期の設計段階から機器を統一的に管理し、ログデータをデータベースに集めて運転に利用する方針で制御系が設計された[1]。管理データ量の増大とデータベース技術の進化により、2014 年からオンラインデータベースとしてカラム型 NoSQL（Apache Cassandra[2])の利用を始めた[3]。そして、主に SACLA の同期信号 60Hz に対応するために、さらにデータベースの性能を活かすデータ収集系を 2018 年から導入している[4]。現在 Apache Cassandra Database (Cassandra DB)をログデータの管理に用いて第一世代（MADOCA2DB）、第二世代（M&M）の両方を運用中である。

第一世代を利用し始めてから 6 年が経ち、当時新規導入したハードウェアの故障が目立つようになってきている。また登録信号数の増加により、第二世代のデータベース増強を計画している。加速器ログデータ収集の目的で運用していく上で主だったトラブルは一通り経験してきたと思われるので、この機会に Cassandra DB 運用についての経験をまとめ今後の似たシステム導入の際の参考としたい。

2. 取り扱いデータについて

取り扱っているデータについて、Cassandra DB 利用の視点からまとめる。

2.1 データの種類、総数

現在運用中の Cassandra DB は4つの clusters（第一世代：1、第二世代：3）で、加速器によって分けている。データ量の大部分を占めるものは時刻に紐づけたポイントデータである。規模の把握のため Table 1, 2 に信号点数と書き込みレートの合計を示した。pol (polling)信号は、定期的（信号によって 0.2 秒間隔から 1 分くらい）に書き込むタイプの収集データである。sync(synchronized)信号では、トリガー信号の配信でデータの同時性が担保される。SACLA のトリガー信号は加速器の繰り返し周波数の 60Hz である。この場合 0.5 秒間のデータはまとめて 1 カラムのデータとして取り扱い、2Hz で書き込みをしている。

Table 1: Number of Signals and DAQ Rate of the First Generation System

Cluster name	#signals (pol)	total rate
SY/LI/BL	11215	3.8kHz

Table 2 Number of Signals and DAQ Rate of the Second Generation System

Cluster name	#signals (pol)	total rate
SR	21087	15kHz
SACLA	55034	19kHz
SCSS+	6929	2.5kHz
	#signals (sync)	
SR	2481	2.5kHz
SACLA	3001	6kHz
SCSS+	293	0.6kHz

2.2 読み出しの傾向

加速器運転中は最新値を取得することが多い。例えばアラーム監視で定期的に最新値を参照している。また異常があった場合などに特定の信号を過去にさかのぼってデータを取得しドリフト傾向を確認し、他の信号との相関を解析する。何か月も前のデータに遡る需要はあまり発生しない。

2.3 その他のデータ型について

波形データやイメージデータの多次元データへの対

k.okada@spring8.or.jp

応も需要がある。例えば cod beam position data はかつて第一世代のシステムで同期したポイントデータの集合体として収集していた。bunch current 測定値はどちらの世代でも波形データとして取り扱っている。また第二世代は波形、イメージのメタデータのみを Cassandra DB に保存、実体はマウントされたファイルサーバーに保存するという方式での運用を開始している[5]が、本稿では説明を割愛する。

3. システム構成

Cassandra DB は信号に付随する機器グループの属性やデータ収集方式などの信号情報を管理するリレーショナルデータベース（RDB）と組み合わせてシステムを構成している。

機器側、フロントエンドでは収集信号のリストやその収集タイプを元にデータ収集プロセスを起動して Cassandra DB へデータを書き込み、読み出し側のアプリケーションでは信号の属する機器グループの集計を行うなど多面的なアプローチが必要となり、柔軟にテーブル間の連携が取れる RDB は必須と考えている。

第一世代、第二世代のデータの流れを Fig. 1, 2 に示す。図の中央の Cassandra DB 枠内の cf*はデータベース内のテーブルを示し詳しくは 5 章で述べる。第一世代(Fig.1)では relay プロセスの下に 2 種類の writer プロセスが存在し、一つは in-memory DB、一つは Cassandra DB への書き込みを担当する。in-memory DB は速度の要求される最新値の読み込みに利用する。第二世代(Fig. 2)では DAQ プロセスが直接 Cassandra DB のみに書き込む。ここでは最新値の読み込みの役割を Cassandra DB が兼ねている。

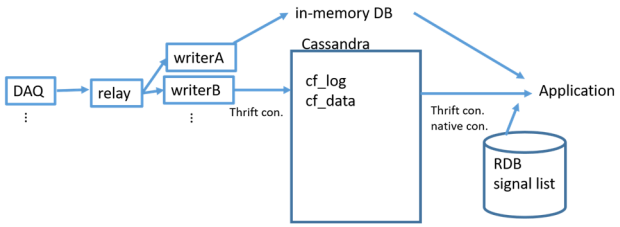

Figure 1: The data flow of the first generation system.

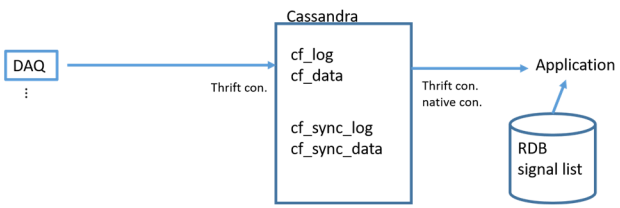

Figure 2: The data flow of the second generation system.

4. ノードスペック

Table 3 に加速器の運転に利用中の 4 台の Cassandra DB クラスターのスペックについてまとめた。表の中 1 台目の「LI/SY/BL」のみ第一世代で残りの 3 台のクラスターは第二世代である。第一世代は1U サーバー機を 1 ノード毎に割り当てたが、第二世代からは Figure 3 のような高集積サーバーを使用している。

第一世代は導入当初は SR のデータ収集も担当しており過去データをそのままアーカイブとする運用方針をとったので、大容量の HDD を積んでいる。第二世代は過去データを定期的にアーカイブに移動し、その際はデータの間引きを行っているため、当面のオンラインデータを保持する分だけの SSD を搭載した。3 replication の設定で、SACLA クラスターが月当たり約 300GB と最も書き込み量が大きい。

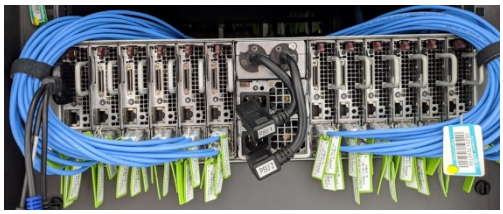

Figure 3: A photo of the SR cluster's front panel. A 3U chassis contains 24 servers.

5. Cassandra DB テーブル設計

一信号について、時系列分割の異なる 2 つのテーブルの組で運用している。最終的に一日 1 エントリーとなる cf_log(, cf_log_sync)とログデータ本体のための cf_data(, cf_data_sync)である。信号は新たに追加されたり、廃止になったり、またデータ収集が中断されたりするが、後者のログデータのみからデータ有効期間を洗い出すのは時間がかかる。その時前者のテーブルを目次として役立てる。

第二世代では第一世代の経験をもとに修正を加えた。

5.1 第一世代

Table 4 に cf_log, cf_data のテーブル構造について示した。

特徴の一つとして Cassandra の partition key として signal_name（信号名）を用いた点がある。RDB を用いずに Cassandra DB のみで運用する狙いがあったが、結果的には RDB の信号情報なしにデータを引用することは

Table 3 Cassandra Server Specifications

System	Cassandra version	#nodes	storage/node	memory/node	replication/#racks
LI/SY/BL(2014-)	2.0	12	9TB(HDD*3)	16GB	3/2
SR(2018-)	2.2	20	2TB(SSD)	32GB	3/1
SACLA(2018-)	2.2	44	2TB(SSD)	32GB	3/1
SCSS+(2017-)	2.2	18	2TB(SSD)	32GB	3/1

無理があり、信号名を変更した場合の過去データを引き継ぎに支障がでた。

cf_data のデータ部分は MessagePack[6] で固め blob の形式で収めている。ポイントデータだけでなく波形データ、画像データの保存の狙いもあった。しかし、実際には 1 partition key(この場合は一日)のデータサイズが大きくなると Cassandra DB の挙動が不安定になるので、さほどの自由度は得られなかった。この点については後の 8 章でトラブルの一例を紹介する。

Table 4: Table Definition (The First Gen.)

table / column	comment
cf_log	
signal text, date varint, PRIMARY KEY ((signal), date))	`signal` is signal_name: `date` is CCYYMMDD
cf_data_######	CCYYMM
signal text, time varint, value blob, PRIMARY KEY ((signal), time))	`signal` is signal_name+CCYYMMDD `time` in [ns] `value` holds a MessagePacked data.

5.2 第二世代

第一世代の経験を踏まえて設計した。Table 5, 6 に紹介する。

まず partition key は RDB との連携を前提に signal_id (通し番号)とした。

これまで主だったせいぜい1Hz 程度のポーリングデータに加え、SACLA の運転周波数 60Hz に同期した収集(sync)を取り込むため、cf_log_sync, cf_data_sync の組を導入した。60Hz を随時書き込むのは速度的な問題と容量の問題があり、DAQ 側でバッファして 2Hz の書き込み頻度に抑えている。30 ティック分(=60/2)のデータは差分時間、差分 event number の自作の形式でパックして Table 6 に示す`t_val`に収めている。

Table 5: Table Definition (The Second Gen.)

table / column	comment
cf_log	
key text, date int, err int, evno bigint, f_val float, i_val int, stat float, time bigint, PRIMARY KEY (key, date)	`key` is signal_id `date' is YYMMDD `time` in [μs]
cf_data_####	YYMM
key text, time bigint, err int, vno bigint, f_val float, i_val int, stat float, t_val text, PRIMARY KEY (key, time)	`key` is signal_id+YYMMDD `time` in [μs]

大きな変更は cf_log にポイントデータを書き込み、第一世代で利用していた in-memory DB の役割を持たせた点である。これまでのところ並列処理によって読み出し速度は許容範囲内である。利用の一例を 7 章で紹介する。上書きされたデータは Cassandra DB の compaction プロセスのタイミングで消去され最終的に1日 1 エントリーとなる。

Table 6: Table Definition (The Second gen.)

table / column	Comment
cf_log_sync	
key text, date int, err int, evno bigint, f_val float, i_val int, stat float,time bigint, PRIMARY KEY (key, date) INDEX ON (evno);	`key` is signal_id" `date' is YYMMDD `time` in [μs]
cf_data_sync_####	YYMM
key text, ime bigint, err int, evno bigint, f_val float, i_val int, num int, stat float, t_val text, PRIMARY KEY (key, time) INDEX ON (evno);	`key` is signal_id+YYMMDD `time` in [μs] `t_val` is packed data (private format)

6. アクセスのためのアプリケーション

GUI などの制御用アプリケーションからのデータベースへの接続は Thrift protocol[7] を利用した C 言語のライブラリを作成した。接続ノードが不調の場合の接続切り替えの機能を備えている。また python の CGI で WEB ブラウザでの単純な時系列グラフ表示のための基本的なデータ閲覧環境を提供している。

書き込み側はタイムスタンプから partition key を作成し、cql 命令を発行する。データベースからの応答が遅れる場合に備えてバッファ機能がある。

読み込み側については利用者に DB のテーブル格納構造を意識させる必要はないので、ライブラリで日毎の partition key、月毎のテーブルへのアクセスへの変換を行う。パックされた格納データは解凍しなければならないが、このため信号情報を納めた RDB との連携を行っている。

7. フィードバックへの利用

本来 Cassanda DB は結果整合性のコンセプトで設計されており、機器のフィードバックの取り合いへの利用には適していないが、1, 2 秒程度の遅れを許容しての利用は可能である。ここでは、信号数が多いために最も時間的に要求の厳しい利用例を紹介する。

SR の全周 290 個のビームポジションモニター(BPM)の 1 秒周期のデータを用いてステアリングマグネットを制御し軌道補正をかけている。Figure 4 に DB 側のタイム

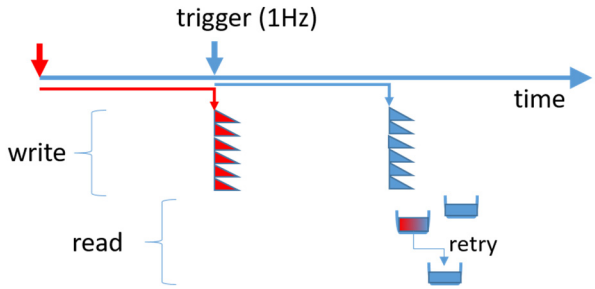

Figure 4: A timing chart of the cod beam position monitor data. The writing process starts 1 second later of the measurement. It takes about 80 [ms] to write a group of signals. An asynchronous reading may mix up two different data sets by chance. In this case, a retry process is launched.

チャートを示す。

トリガー信号のタイミングでバッファに取り込んだデータを複数処理単位に分散して DB に書き込んでいる。これには1Hz の次のサイクルから 80[ms]程の処理である。

読み出しは信号数分のファクターがかかるので、複数の DB 接続を使って並列高速化している。現在は 8 本の接続を張って 100[ms]程度で収めている。2 つのトリガータイミングのデータが重なる場合は再読み込みを専用に作成したアプリケーションの側で行っている。読み出し速度はクエリの条件、含まれる partition key の数の影響を受けるので、将来的な安定運用にはさらに並列度を上げる必要があると考えている。実際、別件のバグ対応のために取得クエリに日付の追加条件を付け加えたところ、信号当りの影響は微々たるものだったが、信号数のファクターのかかったこの用途での遅延が無視できず問題になった。

8. モニタリング

現在第一世代のみ、定期実行スクリプトを作成し、いくつかの定期監視を行っている。Cassandra DB クラスター状況の監視として 1 分毎に status をとり通信できないノードがあれば管理者にメール通知する。また代表的な信号の読み込み時間を 10 分毎に測定して一日一回のレポートに集計している。さらに Cassandra の system log の中で ERROR と顕わに分類されていなくても普段見られない処理が走ったり、何度も繰り返しているようなことがあると不具合の兆候だったりパフォーマンスの低下の要因となるため、system log を解析してキーワードで分類、頻度の 1 日のサマリーをメールで送るスクリプトを動かしている。キーワードは手作業で随時更新している。

9. トラブル

9.1 partition key 当たりのデータサイズによるトラブル

第一世代が SR のデータを取り扱っていた初期のころ、ストレージリング全周に渡る同期の取れた BPM 信号約 2000 点を MessagePack でまとめるということを行っていた。これは Cassandra DB の partition key は 100MB 程度までに収めることという指針[8]に反しており、この場合は 10Hz で取得して 1 日単位の partition key が 20GB 超となる状況で、その結果 compaction task 等の他の負荷と重なった際に FullGC が発生し、15 分程度の接続不能、その後も当データの replication を担当する 3 ノードがフリーズするという事態が数か月にわたって数度発生した。

フリーズしたノードと日付 partition key 毎のノード割り振りの関連を手がかりにして上記原因に至り、応急対処として、収集周期 10Hz を 1Hz に落とし、compaction task を上記信号の割り振りノードと重ならないよう手動で発動することとした。

その間に partition key を小さくするために時間分割するか信号分割するかの検討を行い、時間分割する方針を取って、この信号に関しては分単位で partition key を作成するようにアプリケーションの修正を行った。一日あたり 1440 分なので、2000 点のパックされた信号については partition key のデータサイズがほぼ相殺され他の多くの 1 点 1 信号と同サイズになる。この結果不安定な状態は解消し、フリーズのトラブルは以降起こっていない。

9.2 ノードのハードウェア故障のトラブル

Cassandra DB ではハードウェア故障は織り込み済みで、故障時はハードウェアの交換とホスト ID の継承を行ってからデータの修復を行う手順で問題はない。しかし、全てのノードが同じ稼働率になるため、同時期に稼働開始したハードウェア、特にハードディスクは同じ時期に寿命が訪れがちである。データのリペアの最中に別のノードに問題があると、replication=3 の場合は冗長性が失われる。Figure 5 に稼働時間の長い第一世代のハードウェア故障の発生履歴を示す。初期のころはメモリー故障などもあったが、図に示したものは全て HDD の故障である。今年（2020 年）に入って故障頻度が顕著に上昇し、実際にデータの保持ノードが1にまで陥った。SSD を採用した第二世代でも、導入時期が近く同じロットの可能性が高くノード稼働率が平均化されるため、同時期に寿命がくるということは十分考えられる。

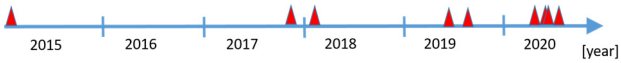

Figure 5: History of hardware failures of 12 nodes in Generation 1 system. The failure rate is rising recently.

第一世代のノード 12 台は 2020 年夏のメンテナンス期間に全て新規1U サーバーに置き換えた。

10. 今後の計画

10.1 Cassandra DB クラスター増強

近年 SACLA から SR への入射計画[9] 中のオンデマンド振り分けへの対応で、運転周波数 60Hz に同期した信号収集の新規登録が大幅に増え、間引きを行わずにデータを保持できる期間が少なくなった。2020 年夏に SACLA のクラスターについて約 2 倍程度の保持容量増加を行った。この実現にあたって Cassandra DB の売りの一つである、運用しながらノード追加する手法は以下の理由で断念した。ノード数を一度に増やした場合、token range の組み換えが発生し、それに対応したデータリペアが読み出し可能になるまでに必要であるが、その時間が不確定なことと、今回 version2 から 3 へのアップグレー

ドで DB データフォーマットを更新する時間も必要であることである。現実問題として運用中にノード追加という手法の選択は取りにくいと感じている。

そこで今回は新規で立ちあげることにし、耐障害性を上げるため、これまで第 2 世代では導入していなかった複数 rack 構成を設定することにした。こちらも運用継続中の変更は難しい。

10.2 Cassandra version up への追従

何年かの内にハードウェアの世代交代に伴い、Cassandra DB の version も古いままではセットアップが困難な状況になると予想される。新しい version 導入での問題の一つは、Cassandra DB の次期バージョンの 4 から Thrift protocol での接続がサポート外になることである。読み出しのライブラリを含め周辺のアプリケーションに影響する。

11. まとめ

Apache Cassandra DB を加速器のログデータ用に導入し 6 年運用してきた。

信号の数や種類が多い場合に RDB と連携したシステムを組むことが必要である。第二世代ではその前提で Cassandra DB のキーは信号 ID を利用して RDB の信号情報と紐づけた。

第一世代ではカラムタイプを blob とし自在なデータを MesseagePack で固めて収納したが、partition key のサイズが大きくなるとクラスターの動作が不安定になるので、データ収集の設計の自由度はそれほど上げられたわけでもない。そこで第二世代ではいくつかの型に限定する方針をとった。

ハードウェアは導入時期が同じものについて同時期に故障が相次ぎ、冗長性が保てない状態にまで陥った。2020 年夏に稼働時間の長い第一世代の全てのノードのハードウェアを交換した。

Cassandra DB 導入当初の、データ形式は自在に決められ、性能が不足したら運用中でもノード追加で対応でき、しかも比較的性能の低いハードウェアで十分というシステム計画は幻想だったが、同じスキームでスケールアップでき、故障には予備ノードを準備してさえおけば良いことは事実で運用継続の上でのトラブルは比較的少なかったとはいえる。その中で、構築上の注意点が見えてきた。今後のハードウェアの置き換え準備、新規構築案件に生かしていく。

参考文献

[1] R.Tanaka *et al*., “The first operation of control system at the SPring-8 storage ring”, Proceedings of ICALEPCS 1997, Beijing, China, (1997).
[2] Apache Cassandra; https://cassandra.apache.org
[3] A. Yamashita and M. Kago, “MADOCA II Data Logging System Using NoSQL Database for SPRING-8”, in Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, Oct. 2015, pp. 648-651.
[4] T. Fukui *et al*., “Status of the Control System for the SACLA/SPring-8 Accelerator Complex”, in Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17), Barcelona, Spain, Oct. 2017, pp. 1995-1999.
[5] A.Kiyomichi *et al*., “SACLA/SPring-8 ビーム輸送系における新スクリーンモニタ制御システムへの GigE カメラの適用“第 17 回日本加速器学会年会（2020）プロシーディングス FRPP24.
[6] MessagePack; https://msgpack.org
[7] Apache Thrift; https://thrift.apache.org
[8] https://docs.datastax.com/en/dse/5.1/cql/cql/ddl/dataModelingAnalysis.html
[9] H. Maesaka *et al*., “On-Demand Beam Route and RF Parameter Switching System for Time-Sharing of a Linac for X-ray Free-Electron Laser as an Injector to a 4th-Generation Synchrotron Radiation Source”, in Proc. 10th Int. Particle Accelerator Conf. (IPAC'19), Melbourne, Australia, May 2019, pp. 3427-3430.