J-PARC operation data archiving using Hadoop and HBase

Akinobu Yoshii [A)], Nobuhiro Kikuzawa[A)]
[A)] Japan Atomic Energy Agency
2-4 Shirane Shirakata, Tokai-mura, Naka-gun, Ibaraki 319-1195

Abstract

J-PARC (Japan Proton Accelerator Research Complex) is controlled with a lot of equipment, and EPICS record of Linac and RCS extends to about as many as 64000 points. The operation data is archived in the system using PostgreSQL now. But, it is not enough in performance, capacity, and extendibility, if it is taken into consideration that data volume will increase in the future. So the new system architecture will be needed, which can respond to the prospective change.

In order to cope with this problem, the use of Hadoop and HBase has been considered. These products are distributed framework and distributed database, and are excellent in scalability. This time, about 50 TB of Hadoop Distributed File System (HDFS) was built as a prototype system using one MasterNode and nine SlaveNode(s), and HBase was worked on this file system. And the past data was stored to HBase by several data structures, and it verified about performance of data retrieval and registration. When suitable parameter tuning was given with a data structure suitable for HBase, the response time of data retrieval was shortened sharply as compared with the present system, and registration performance was high.

However, while verifying, some problems about the availability and the operability have appeared. This paper describes correspondence of the result obtained from verification, problems, and future.

Hadoop・HBase を利用した J-PARC 運転データアーカイビング

1. はじめに

J-PARC (Japan Proton Accelerator Research Complex) は多数の機器により制御されており、Linac、RCS に関して約 64000 点にも及ぶ EPICS レコードのデータを収集している。現状では、PostgreSQL を利用したシステム[1]にてデータを格納しているが、将来的なデータ量増加を鑑みると、性能や容量、拡張性の面で十分とは言い切れず、新たなシステムアーキテクチャの検討が必要となってきている。

この課題に対応するために、大容量データ対応や拡張性に優れる分散処理フレームワークの Hadoop[2] と分散データベースの HBase[3]の利用について検討を行ってきた[4]。今回はプロトタイプシステムとして、1 台の MasterNode と 9 台の SlaveNode という構成で、約 50TB の Hadoop Distributed File System (HDFS)を構築し、このファイルシステム上で HBase を稼働させた。この HBase に現行システム上のデータを幾つかのデータ構造で移行し、その検索性能やデータ登録性能について検証を行った。その結果、システムに適したデータ構造と適切なパラメータチューニングを施した場合に、現行システムと比較してデータ検索の応答時間が大幅に短縮され、書込みについても性能向上が確認できた。

ただし、検証を行う中で実利用を鑑みた場合、可用性や運用面に関する解決すべき課題が幾つか浮上している。

本稿では、これらの検証から得られた結果や課題並びに今後の対応について記述する。

2. システム構成

今回プロトタイプとして 10 台の IA サーバを用いてシステムを構築した。そのうち 1 台をクラスタ管理やメタデータ管理を行う MasterNode に、9 台をデータブロックの格納や処理を行う SlaveNode に割り当て、これらを Gigabit Ethernet で接続した。システム構成の概略を図 1 に、ハードウェアやソフトウェアの構成品目情報を表 1 に示す。

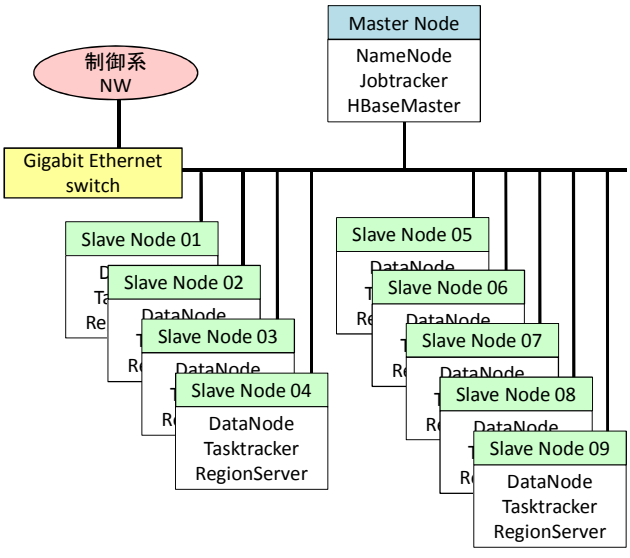

図 1：システム構成

表 1：構成品目情報

役割	主要スペック
Master Node	DELL PowerEdge R610 CPU：Intel Xeon E5620 (4Core 2.4GHz) MEM：12GB (4GB x3,1333MHz, DDR3) HDD：600GB SAS 10krpm x4 (RAID10)
Slave Node	DELL PowerEdge R410 CPU：Intel Xeon E5620 (4Core 2.4GHz) MEM：12GB (4GB x3,1333MHz, DDR3) HDD：2TB SAS 7200rpm x4 (RAID5)
Gigabit Ethernet Switch	PCI FXG-16IRM2 10/100/1000BASE x16
Software	OS：CentOS6.0 (日本語版) java：JDK 10.6.2-1 (1.6.0-29) Hadoop：1.0.3 HBase：0.92.1

3. データ構造の検討

3.1 現行システムのデータ構造

現行のデータアーカイブシステムでは、データ書き込み時のデータベースへの負荷を下げるため、幾つかの EPICS レコードをグループに纏め、それらに対するデータテーブルを日毎に作成する形式となっている。また、取得値のデータを byte 列で一纏まりにして 1 つの column に格納している。そのため、データテーブルとは別に EPICS レコードとデータテーブルの紐付及びデータ格納の順序を表すデータ定義テーブルを保持している。このデータ構造の概要を図 2 に示す。

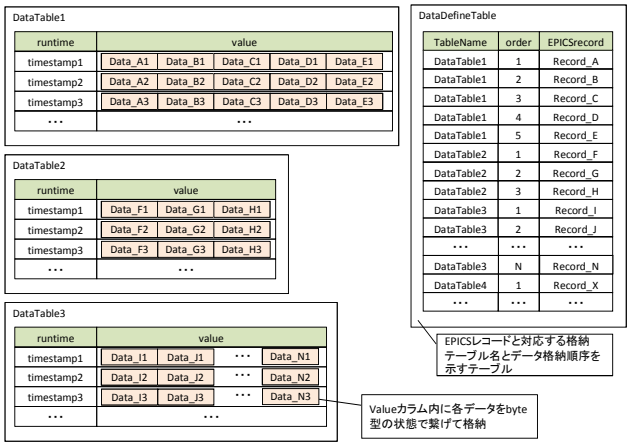

図 2：現行システムデータ構造概要

3.2 HBase 向けのデータ構造

HBase は Key-Value ストア型のデータベースであり、検索で利用する情報を row（テーブルの key に相当）に割り当て、それに紐付く値を column に格納するシンプルな構造が最も効率的である。従って、検索で利用する Timestamp 情報を row に割り当て、取得値を column に格納する構造が HBase に適しているものと考える。なお、テーブル構成については以下の様なパターンが考えられる。

●現行踏襲型
現行システムに近い構造で、幾つかの EPICS レコードの値を 1 テーブルに格納する。ただし、データを 1 つの column に纏めて格納するのでは無く、レコード名を付けた qualifier（一般的なデータベースシステムの属性に該当する）毎の column に格納する。また、別途 EPICS レコードと格納テーブル名の紐付テーブルを持たせる。

●Key-Value 型
EPICS レコード毎にテーブルを作成し、key となる timestamp と紐付く取得値を 1:1 で格納する。

これらデータ構造の概要を図 3 に示す。

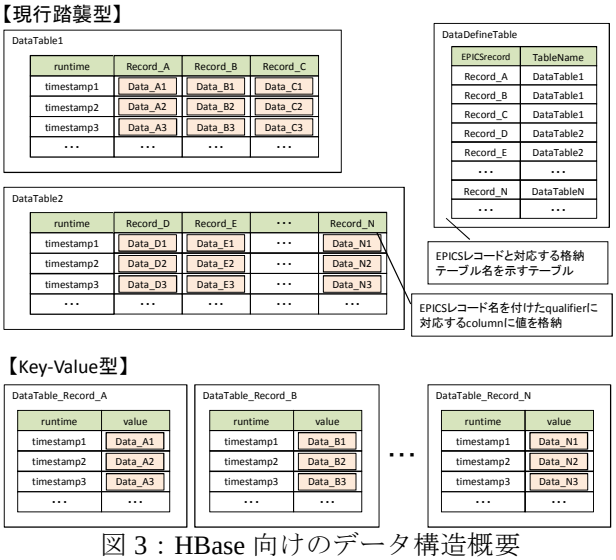

図 3：HBase 向けのデータ構造概要

なお、HBase 上のデータは Region と呼ばれるデータブロックに纏められて各 SlaveNode に分散配置される。登録されるデータは row の昇順でソートされて Region に配置されるため、timestamp の様に単純増加するものを row に割り当てる場合にはデータ追加対象の Region を持つノードが限定されて書込みが集中し、分散化のメリットが得られにくい傾向がある[5]。だが、実際のケースでは登録テーブルは分けられており、自ずとテーブル毎の Region も分かれて書込みが分散化されるため、この問題に対する懸念は回避することが可能と考える。

4. 検証・評価

4.1 データ検索性能

データ検索の性能を評価するため、
- 現行システム
- HBase に現行システム上のデータを「現行踏襲型」のデータ構造で移行した場合
- HBase に現行システム上のデータ「Key-Value」型のデータ構造で移行した場合

の 3 パターンでデータ検索に係る時間を測定した。現行システムサーバの主要スペックを表 2 に示す。

表 2：現行システムの主要スペック

役割	主要スペック
Database Server	DELL PowerEdge R200 CPU：Intel Xeon X3210 (4Core 2.13GHz) MEM：8GB (2GB x4, DDR2） HDD：160GB SATA (データストアは外部のファイルサーバ上の Stornext File System をマウントして利用)

HBase に移行した過去のアーカイブデータは 1 年分で、容量は約 1TB である。また、データは gz 形式の圧縮を行って HBase に格納している。

サンプルとして適当に選んだ EPICS レコードに対する測定値を 3 日分検索するのに掛かった時間を表 3 に示す。検索で取得したデータ数は 1 秒周期のサンプリングで 3 日分、259189 個であり、検索はキャッシュの無い状態で行ったものである。

表 3：データ検索性能比較

	検索時間(秒)
現行システム	36
HBase(現行踏襲型)	20
HBase(Key-Value 型)	6

Key-Value 型では現行システムと比べて 5 倍以上という高い性能が得られた。現行システムと比べて HBase システムのサーバスペックが良いことも影響しているが、それに加えてデータ構造の違いで検索処理効率が良くなり、より大きく性能を上げていると考えられる。

一方、現行踏襲型も現行システムと比べて性能が向上しているものの、Key-Value 型程の性能は出ていない。これは、現行踏襲型は qualifier として設定している EPICS レコードに対しても検索を必要とするが、Key-Value ストア型データベースである HBase は、row を対象とする検索は高速であるのに対して、qualifier や value を対象とした検索は速度が落ちることによるものと考えられる。

なお、HBase システムでは 9 台の処理サーバが存在し、現行システムと比べてサーバ台数が多いが、timestamp の範囲検索においては、検索範囲のデータ Region を持つサーバに処理が集中する傾向にあるため、処理サーバ台数に関する寄与は少ないものと考える。

4.2 データ登録性能

データ登録性能に関する性能評価として、EPICS レコード 100 個分に対して 1 秒周期で 1 日分のデータを取得することを想定し、100×86400 個のデータを HBase に登録する際の時間を計測した。現行踏襲型では全て 1 テーブルに格納し、Key-Value 型では 100 個のテーブルを作成してそれぞれのデータ投入を同時に行った。なお、データは現行データに格納されているものと同じ 8 バイトの Double 型とし、テーブル作成に係る時間は計測に含めない。結果を表 4 に示す。

表 4：データ登録性能比較

	登録時間(秒)
HBase(現行踏襲型)	173
HBase(Key-Value 型)	273

登録性能については、現行踏襲型の方が勝っているが、これはデータのコミット処理の回数が少なくて済むことと、一度にオープンするテーブルの数が少なく、接続に掛かるシステム負荷が低いためと考える。性能の劣る Key-Value 型でも、データ登録性能は約 31600ops であり、現状ピーク時に秒間約 10000 件のデータが発生することに対して十分な処理性能を持っていることが確認された。

一連の検証で確認されたそれぞれのデータ構造における長所・短所を表 5 に示す。

表 5：データ構造による長所・短所

	現行踏襲型	Key-Value 型
長所	・システム負荷が低い ・登録性能が高い	・検索性能が高い
短所	・検索性能が劣る ・EPICS レコードとテーブルの紐付管理が必要	・システム負荷が高くなる

5. 課題と解決策

今回検証を行った中で幾つか浮上した課題とその解決の方向性について以下に記述する。

5.1 ディスク容量効率の改善

HBase は登録するデータ毎にデータバージョン管

理に用いるタイムスタンプ情報を自動的に付加して格納している。更に HDFS でデータ Region のレプリカを保有しており、実際のデータ容量に対して数倍のディスク容量を消費するため、ディスクの容量効率が良くない。そのため、以下の方法にて改善を行った。

●圧縮の活用
HBase では gz や lzo 形式のデータ圧縮が利用可能であるため、この機能を用いて容量を抑えることとした。本来は CPU 負荷が少ない lzo 形式が推奨される[5]が、現在使用しているバージョンでは正常に動作しなかったため今回は gz 形式を用いた。これによりデータ容量を 1/10 程度に圧縮することが可能となった。データ圧縮や伸張に伴う処理負荷のオーバーヘッドが懸念されたが、非圧縮時と比較してもデータ検索や登録の処理性能に大差は見られなかった。

●レプリカの保有数変更と RAID 設定
デフォルトではデータレプリカを 3 個保有する設定になっているが、これを 2 個に減らすことによりディスク使用量を抑えた。ただし、データ保護性が低下してしまうため、代わりに SlaveNode のディスクを RAID5 構成とし、ハードウェア面でのデータ保護性を向上させている。ディスクの RAID 化により利用可能な容量は少なくなるが、それを差し引いても 10%以上利用可能な容量を増やすことが出来ている。また、レプリカ保有数を減らすことでレプリケーションに掛かるシステム負荷が下がり、パフォーマンス向上にも繋がるものと考える。

5.2 システム安定性の向上

検証を行う中で、高負荷を掛けた際に SlaveNode 上で稼働する HBase の RegionServer サービスが停止するという現象が発生した。特にメモリリソースが逼迫した際に発生する傾向が見られた。サービスダウンの防止や影響の低減を行うための対策として以下を検討している。

・サーバのメモリ増設
・Ganglia[6]等の統合リソースモニタツールによるリソース監視
・サービスダウンした際の検知と自動復旧の仕組みの実装

5.3 システム可用性の向上

Hadoop・HBase に於いて、SlaveNode は複数台の冗長構成となっているため、単一ノード障害によるシステム停止は発生しない。一方、MasterNode は単一障害点となるため、ここでの障害はシステム停止に直結する。そのため、MasteNode についてはもう 1 台サーバを追加し、Heartbeat[7]等のクラスタウェアを用いた HA(High Availability)クラスタ構成とし、障害が発生した場合にはもう一方のサーバにサービスを切り替えて継続して利用可能な状態にすることを計画している。

6. まとめ

HBase を利用して現行システム上の運転データを移行して性能評価を行った結果、現行システムの 5 倍以上の高い検索性能を有することが確認された。データ登録処理に於いても十分な性能を有することが確認され、加速器運転データアーカイブの用途で十分利用可能と考える。なお、データ構造については現行踏襲型と Key-Value 型で検索性能や登録性能、システム負荷の面でそれぞれ長所や短所があるため、実利用を鑑みてどのパターンが最適かを改めて検討する必要がある。

また、リソースの有効活用やシステムの安定性と可用性向上に関して改善すべき点が確認されたため、今後は設定の最適化やミドルウェアを補助的に利用するなどの方策で対応する予定である。

参考文献

[1] S.Fukuta, et al., “Development Status of Database for J-PARC RCS Control System (1)”, Proceedings of the 4th Annual Meeting of Particle Accelerator society of Japan, August 2007
[2] http://hadoop.apache.org/
[3] http://hbase.apache.org/
[4] A.Yoshii et al., “Examination of applying Hadoop for J-PARC driving data archive” Proceedings of the 8th Annual Meeting of Particle Accelerator Society of Japan
[5] Apache HBase Reference Guide (http://hbase.apache.org/book.html)
[6] http://ganglia.sourceforge.net/
[7] http://www.linux-ha.org/wiki/Heartbeat