HBase/Hadoop を利用した J-PARC 運転データアーカイビング用ツールの開発

DEVELOPMENT OF TOOLS FOR THE J-PARC OPERATION DATA ARCHIVING USING HBASE/HADOOP

菊澤信宏[#, A)], 池田浩[†, A)], 吉位明伸 [B)], 加藤裕子 [A)], 大内伸夫 [A)]
Nobuhiro Kikuzawa[#, A)], Hiroshi Ikeda[†, A)], Akinobu Yoshii [B)], Yuko Kato [A)], Nobuo Ouchi [A)]
[A)] Japan Atomic Energy Agency
[B)] NS Solutions Corporation

Abstract

J-PARC requires us to refer enormous operation data in order to control a large amount of its equipment. The Linac and the RCS in J-PARC have about 64,000 EPICS records producing such data, and we have been accumulating the data into a PostgreSQL database system.

We are planning to replace the storage with HBase/Hadoop. HBase is so-call NoSQL, which has scalability to data size at the cost of the high broad utility of SQL. HBase is constructed on a distributed file system provided by Hadoop, a cluster with advantages including automatically covering its composing nodes' breakdowns and easily adding new nodes to extend its capacity. While HBase and Hadoop are practical products as Facebook or LINE indeed utilizes them, HBase/Hadoop developers have been still strenuously enhancing functions and fixing bugs.

We built a test system with HBase 0.94.5/Hadoop 1.0.4, and we developed a tool to bulk-load data from PostgreSQL to HBase, a tool to retrieve and accumulate data from the Linac and the RCS to HBase, and a tool to refer data in HBase, but we failed to complete transferring all the data in PostgreSQL due to an existing bug in HBase. Having eventually known that it was difficult to work around the bug, and the bug was already reported and fixed in a new version of HBase, we took this occasion to update them to HBase 0.96.1.1/Hadoop 2.2.0, which the new version of Hadoop was officially released at just the right time, conquering a weak point of a so-called single point of failure, although we didn't expect that the architecture of HBase and Hadoop was progressively changed and there were version compatibility problems especially for HBase. We have been required to fix or change our tools.

This report shows issues, considerations and solutions about developing the tools and fixing them into the new versions of HBase/Hadoop.

1. はじめに

大強度陽子加速器施設 J-PARC では、それを構成する多数の機器の制御のために大量の運転データが必要となる。Linac 及び RCS では約 64000 点ものEPICS レコードによりデータがもたらされ、これらは現在 PostgreSQL に格納されている[1]。

我々はこの格納先として HBase[2]を採用する計画を進めている。HBase はいわゆる NoSQL と呼ばれるデータストアで、SQL の高度な機能性を不要とすることで大量のデータをスケーラブルに扱うことが可能である。HBase は Hadoop[3]の分散ファイルシステム上で構築され、複数のマシンで構成するクラスタを使用し、障害時の自動復旧や容量増設の容易性が利点として挙げられる。HBase および Hadoop は Facebook や LINE などで実際に利用されており実用レベルの製品だが、現在も機能拡張やバグフィックスが盛んに行われている。

現在までに、HBase 0.94.5/Hadoop 1.0.4 にて構築したクラスタに PostgreSQL からデータを移行するツール、PostgreSQL に代わって Linac や RCS からデータを取得し HBase に格納するツール、および、格納したデータを閲覧するツールを開発してきたが、HBase の不具合が顕在しその回避が難しいことが分かったため、HBase のバージョンを上げることとした。この際、Hadoop の弱点である単一障害点を克服したバージョンが正式リリースされたことから、HBase 0.96.1.1/Hadoop 2.2.0 を採用した。アーキテクチャが大きく変わり、特に HBase に関してはバージョン互換性について問題があり、これまで開発したツールも大きな対応・修正を行う必要があることが分かった。

本件は、これらのツールの作成上の問題点、および、新しい HBase/Hadoop のバージョンへの対応に関する問題点と、その検討・解決方法について報告するものである。

2. システム構成

2.1 クラスタ

HBase/Hadoop のアーキテクチャは基本的にいわゆるマスタ型であり、コンポーネントの配置にはさまざまな選択肢が考えられるものの、最小限の構成を考えた場合は、クラスタを構成するノードは全体を統合するマスタノードと実作業を行うスレーブノードの 2 種類に分けて管理するのが簡単である。

kikuzawa.nobuhiro@jaea.go.jp
† ikedahi@post.j-parc.jp

Hadoop の新バージョンでは、単一障害点であった分散ファイルシステムのメタ情報を管理する Name Node に冗長性が導入されたが、この機能を有効にするにはマスタノードが 3 台必要となる。現在、テスト環境用にスレーブノードは 9 台用用意している。Table 1 にクラスタの構成を示す。なお、HBase/Hadoop の旧バージョンにて構築していたクラスタでは、Hadoop の単一障害点に対し、これとは他にさらに 3 つのソフトウェア Heartbeat, Pacemaker, DRBD を使用し、Master Node 1 を Master Node 2 でコールドスタンバイする構成を採用していた[4]。

マスタノードについてはスペックの違いから、コンポーネントを Table 2 のように配置している。なお、HBase 及び Hadoop は ZooKeeper を利用しており、ZooKeeper は別途、3 台以上奇数台のノードで構成されたクラスタを組む必要がある（いわゆるクオラムを構成する）。これは、3 台のマスタノードを流用する。

運用段階に移行する際には、これらの幾つかは再構成する必要があるだろう。Master Node 2 については配備されたコンポーネントに対してメモリが不足しており、また、Name Node を担うには HDD が RAID 化されていない。スレーブノードに関しては、HDD×4 台の RAID5 構成は、HDD を独立に稼働させる場合と比べて、特に書込み処理が並列化できないために、相対的にスループットが 1/4 まで低下する可能性が考えられる。現在、これらの HDD のファイルシステムは ext4 を使用しており、適切なマウントオプションについて検討する必要がある。

Table 1: Hardware Specification

Master Node 1	DELL PowerEdge R610 CPU: Intel Xeon E5620 (4core, 2.4GHz) MEM: 24GB HDD: 600GB x4 (RAID10)
Master Node 2	DELL PowerEdge R200 CPU: Intel Xeon E3210 (4core, 2.13GHz) MEM: 8GB HDD: 160GB x1
Master Node 3	DELL PowerEdge 860 CPU: Intel Xeon X3210 (2core, 2.4GHz) MEM: 4GB HDD: 250GB x1
Slave Node	DELL PowerEdge R410 CPU: Intel Xeon E5620 (4Core, 2.4GHz) MEM: 24GB HDD: 2TB x4 (RAID5)

Table 2: Components Deployment (Master Nodes)

Master Node	1	2	3
Name Node	o	o	
Journal Node	o	o	
ZKFC	o	o	
Resource Manager	o		
History Server	o		
ZooKeeper	o	o	o
HBase Master	o	o	o

2.2 テーブル

現在、EPICS のチャネルアクセスから取得した運転データは PostgreSQL に格納しているが、PostgreSQL では 1 テーブルに格納するデータ量に上限があるため、チャネルのグループを表す文字列（以下、チャネルセット）と測定年月日の組み合わせで名前付けされたテーブルを動的に生成して格納している。これは、データ量のために RDB の特徴・能力の幾つかを妥協する使い方であり、NoSQL が求められてきている理由そのものとも言える。

HBase ではテーブルに対するデータ量の制限はなく、複雑な構造を導入する必要性がない。チャネル名のバイナリ表現と測定日時の 1970 年 UTC エポックの経過ミリ秒を 64 ビット整数で表現したバイナリ表現を組み合わせたものをキーとし、測定データを関連付ければよい。なお、測定データは倍精度実数の配列で与えられ、IEEE754 によるバイナリ表現の並びで関連付けられる。多くのデータは配列のサイズが 1 つ、すなわち、単なる倍精度実数が使われる。

この構成では、キーのサイズが測定データに対して大きく、また、キーに同じ（あるいは同じような）バイト列が繰り返し発生するが、HBase ではデータを圧縮して格納するのが通常の使い方であり、一般的にこのような繰り返しが現れるデータは高い圧縮率が期待できる。

ここで、キーを表すバイナリ表現のオーダは検索方法に直接的に影響するため、データを具体的にどのような順番でどのようにバイナリ表現にエンコードするかを、良く考えて決める必要がある。我々は、指定したチャネルに対し指定した期間に含まれる測定日時のデータを取得するという使い方を想定し、キーの構成内容としてチャネル名を測定日時よりも前に配置することを選択した。また、HBase では、キーに測定日時を含めるときはそのオーダを逆順にして最新のデータを最初に検索できるようにするのが一般的である。内部的には日時は符号付き整数で表されていることもあり、ここでは、符号付き 64 ビット整数の最大値から減算することにした。

なお、バイナリのオーダを乱さないようにすることもあり、バイナリ表現は一貫してビッグエンディアンを用いる。

3. データ移行ツールの開発

当初、PostgreSQL から取得したデータを逐次 HBase へ格納するツールを作成したが、実行速度を計測してみると、PostgreSQL に蓄積した数年間のデータに対し、移行にかかる時間の見積もりがおよそ数か月単位となった。HBase/Hadoop に対して本格的に運用を開始しても、しばらくの間は PostgreSQL と併用し、HBase に予期せぬ障害が発生したときに PostgreSQL から復旧することを考えている。このため、この移行速度はこの目的では実用的でない。

このため、HBase に用意されているバルクロードを利用することを検討した。これは、Hadoop の MapReduce を利用して HBase におけるデータの内部

表現である HFile と呼ばれる形式のファイルを複数生成し、これを HBase に用意されている命令を呼び出して取り込むという手順になる。MapReduce では、およそ Map タスクによってバルクロード元からデータを取得して振り分け、Reduce タスクによって HFile を生成する。

HBase に予め用意されている MapReduce タスクの実装をそのまま使用すると、以下に述べるような問題が発生した。このため、この実装のロジックを参考にしつつ、独自の MapReduce を利用したツールを作成した。以下にこれらの問題とその解決方法について記述する。

3.1 PostgreSQL へのアクセス

個々の Map タスクで PostgreSQL に接続してデータを取得することについては、以下のような問題が発生する。

- PostgreSQL サーバに対しては同時接続数を制限しないと無駄に負荷が掛かるため、Map タスクの分散処理数は制限しておきたい。一方、大量のデータを捌くためには、クラスタを最大限に活用すべきであり、両者は相反している。また、基本的に MapReduce のタスクは使い捨てであるため、一般的に負荷が高いとされるデータベースへの接続処理を Map タスク毎に行うことになる。
- PostgreSQL に格納されるデータはテーブルによって保持している量が大きく異なるため、均等に Map タスクに負荷を分散させるのが難しい。

このため、PostgreSQL サーバへの接続及びデータの取得は MapReduce とは別に行うものとし、また、PostgreSQL から得られたデータは Hadoop の分散ファイルシステム（以下、HDFS）上にアップロードし、これを Map タスクに分散処理させるものとした。MapReduce による HFile の生成中に、次のデータ測定日に対する PostgreSQL から HDFS へのアップロードを先んじて行うことで処理の並列が可能であり、並列処理のリソースが十分であれば、処理に要する時間は短くなることはあっても長くなることはないと考えられる。

3.2 リージョンとパーティション

HBase はリージョンと呼ばれる連続したデータの範囲を基本単位として、データをノードに分散させる。おおよそ HFile はリージョンを跨がないように作る必要がある。Map タスクからの出力はパーティションと呼ばれる条件で仕分けされて Reduce タスクに渡されるが、HBase でバルクロード用に用意されている実装では、リージョンでパーティションを切るものとなっている。この場合、各リージョンで高々一つの HFile を生成するだけで済み、すなわち、HFile の生成は必要最小限の個数で済むことが期待される。

これは、一般的な状況であれば十分かもしれないが、我々の用途では、以下の問題が発生する。

- 必ずしもバルクロードするデータがリージョンに均等に分散されているとは限らず、Reduce タスクの負荷が均等にならない。
- リージョンの分割があまり行われていないテーブル（特に新規テーブル）に対しては、少数の Reduce タスクに負荷が集中する。

実のところ、HFile の生成は HDFS 上であり、その Reduce タスクを実行したノードやリージョンサーバの位置とは直接的な関連はない。さらに言えば、リージョン毎に個別の Reduce タスクを用意する必要はなく、すなわち、リージョンの境界でパーティションを切るのは必要条件ではない。必要なのは、HFile を作成するためにデータがソートされていることと、１つの HFile が可能な限り１つのリージョン内に納まっていることである。

均等にパーティションを割り当てるには、どのチャネルがどの程度データを持っているかを調べる必要がある。PostgreSQL サーバに格納されているままでは、テーブルによってレコードに含まれるデータ量が異なるという先に挙げた問題が障害になる。これには、PostgreSQL サーバから HDFS へデータをアップロードする過程を利用できる。

この場合、各 Reduce タスクは、バルクロード先のテーブルのリージョンの境界に気を付けて HFile を作ることになる。

3.3 コンパクションとスプリット

HBase のリージョンサーバは、概念的に、データをストアと称されるコンテナに格納する。ストアには、HFile の形式でファイルとして保存されるストアファイルや、メモリ上に保存されているメモリストアが複数個格納されており、コンパクションと称する操作でストアファイルを結合したり、スプリットと称する操作でリージョンを分割したりする過程において、管理するストアファイルの個数を一定数以下に調整し、また、これらの処理が低負荷になるように、ストアファイルのサイズの関係が一定の好条件を保つように調整している。

HFile のバルクロードは、ストアに格納されるストアファイルの個数を増やし、また、ストアファイルのサイズに関する好条件を崩すことに注意する必要がある。バルクロード自身はコンパクションやスプリットをトリガーしないため、定期的にトリガーされるコンパクションを待つことになる。

単発のバルクロードならこれで問題が発生することは考えにくい。しかし、バルクロードを繰り返した場合、ストアに大量のストアファイルが格納され、コンパクションやスプリットに要する時間が現実的でなくなる可能性がある。特に、ストアファイルが一定数以上存在する場合は、メモリストアのフラッシュが一定時間ブロックされるため、HBase へのデータの書き込みに支障がでることが考えられる。

バルクロードを繰り返し行う場合は、ストアの状態を調べ、取り返しがつかなくなる前に明示的に小まめにコンパクションやスプリットをトリガーするような仕組みを作る必要があるだろう。これには、

HBase の詳細にアクセスする必要があり、これが後に HBase/Hadoop のバージョンを上げた際の障害となった。

4. データ収集ツールの開発

PostgreSQL に代わって Linac や RCS からデータを取得し HBase に格納するツールを作成した。データ取得・格納は、EPICS のチャネルアクセスを利用しており、具体的には JCA[5]を使用している。

HBase の書き込みはブロックされることがある。例えば、大きなリージョンのコンパクションに巻き込まれ、ストアファイルのフラッシュがブロックされた場合、既定で最大 90 秒の書き込みのブロックが発生する可能性がある。また、HBase では深刻でないエラーが発生してリトライする場合、1 秒間のインターバルを挟むコーディングをしているところもある。これは、例えばポーリング型の 1 秒周期の書き込みには間に合わない。このため、アプリケーション側でデータをバッファリングする仕組みを作成した。

5. データ閲覧ツールの開発

HBase に格納したデータを閲覧するツールを Control System Studio[6]（以下、CSS）を用いて作成した。CSS は、加速器等の大規模システムのモニタやオペレーションを行うツールを集めたもので、Eclipse[7]のプラグインとして開発されている。HBase/Hadoop が Java 6 にてサポートされているため、Java 6 対応の SNS 3.1.7 のタグが付けられているものをターゲットとし、CSS のプラグイン org.csstudio.archive.reader の拡張ポイント ArchiveReader を拡張して HBase へのアクセスを可能とするプラグインを作成した。これにより、CSS の Data Browser（Figure 1）から HBase のデータにアクセスすることができるようになった。

この拡張ポイントでは、データを提供するインタフェース ArchiveReader（拡張ポイントと同名）の実装を与えることなる。CSS には実装クラスが幾つか含まれているが、その一部は ArchiveReader に記述されている仕様に従っていない。また、ArchiveReader を使用する側のロジックを確認したところ、幾つか重要な仕様が記述されていないことが分かった。

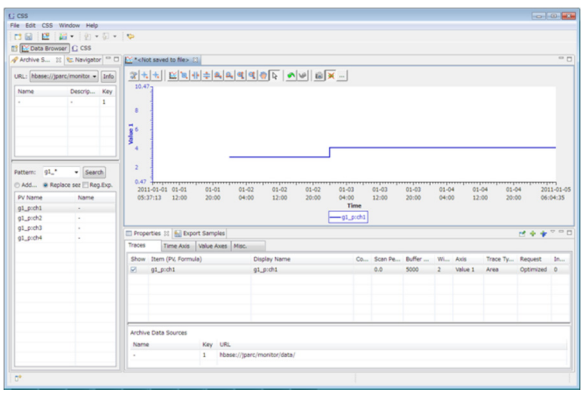

Figure 1: Data browser.

詳細は省略するが、ArchiveReader のインスタンスでリソースを管理するとリークする恐れがあり、その外部でリソース管理をするのが望ましい。拡張ポイント ArchiveReader に実際に登録するクラスは、インタフェース ArchiveReader の実装クラスではなく、そのインスタンスを生成するインタフェース ArchiveReaderFactory の実装クラスであり、これがリソース管理の候補となろう。しかし、この拡張ポイントに登録した実装クラスは、使う・使わないにかかわらずそのインスタンスが生成されるようになっているため、リソースの生成はこれとは別に遅延させる必要がある。さらに、このインスタンス化の要求は必然的に、この拡張ポイントを使用したプラグインを活性化することを意味し、さらに芋づる式に複数のプラグインの活性化が行われる可能性があることに注意する。これは Eclipse のプラグインの思想に反するものである。

ところで、インタフェース ArchiveReader は、生データと「最適化」されたデータの 2 種類のデータを与える仕様になっているが、CSS 側のロジックでこれを混在させて扱う処理が不適切であり、Data Browser にてユーザにミスリーディングさせる可能性に注意する。なお、「最適化」が何を意味するかは ArchiveReader の実装依存とされている。

もう一つ注意が必要な点を挙げておきたい。Eclipse では時間が掛かる可能性がある処理を行わせる場合は、インタフェース IProgressMonitor のインスタンスを同時に渡す。これにより処理側では、中断の通知を確認したり、進行状況の報告を通知したりすることを、処理の実行に支障のない頻度やタイミングで行うことを可能にしている。ところが、CSS では渡された IProgressMonitor のインスタンスをそのまま使うことをよしとせず、中断通知だけを別スレッドから通知するような仕組みをコーディングしている。これは、スレッドセーフ性の問題を導入し、さらに中断対象の処理の指定が曖昧であるため誤動作を引き起こす可能性すらあるため、注意して扱う必要がある。

6. HBase/Hadoop のアップデート

PostgreSQL から HBase 0.94.5/Hadoop 1.0.4 へデータ移行ツールを長期間のデータに対して稼働させたとき、HBase の不具合が発生してツールが異常終了した。既にこの不具合は報告されており、新しい HBase のバージョンでは修正されていること、また、ツール側で回避策が難しいことから、HBase のバージョンを上げることとした。

このころ、Hadoop の新しいバージョンである Hadoop 2.2.0 がリリースされた。Hadoop 2.x の新機能として、以下のものが挙げられる。

- Name Node HA
- HDFS Federation
- YARN

各機能を以下に簡単に述べる。Name Node HA は、

HDFS の単一障害点である Name Node をホットスタンバイによって冗長化し高可用性（High Availability）を実現する。従来の Hadoop 1.x でも Heartbeat 等の外部のソフトウェアを使えば冗長化ができたが、これはコールドスタンバイであり、クラスタの規模によっては Name Node が立ち上がるまでに 30 分から 1 時間必要となる（Data Node から通知されるブロックレポートと呼ばれる情報は HDD には記録されないため、起動時にかき集める必要がある）。HDFS Federation は、HDFS に名前空間を導入し、あたかも複数の HDFS を同一のクラスタで運用する機能である。YARN は MapReduce のリソース管理に関するロジックに代わるもので、スレーブノードからの申告をもとに、分散処理時に、より丁寧なリソースの割り当てを行うものである。従来の MapReduce は YARN を基盤にして再構築されている。なお YARN の API は未だ洗練されておらず、これを直接使うのは避けたい。

HBase を使うにあたっては、Name Node HA の効果が大きい。このため、Hadoop を 2.2.0 にアップデートすることとなった。このとき、HBase については、互換性を考えれば 0.94.x 系の最新バージョンを使うのが望ましいが、Hadoop 2.2.0 と HBase 0.94.x の組み合わせが有効かどうかについてはドキュメントによって説明が異なり、このため、Hadoop 2.x を始めからターゲットとした 0.96.x 系である HBase 0.96.1.1 を採用した。

Hadoop は後方互換性について配慮しているので、コードの上での互換性はおよそ問題ないが、Hadoop の設定パラメータについては YARN のアーキテクチャを理解しておく必要がある。また、HBase については互換性を重要視しておらず、Hadoop よりも HBase への対応が大きな課題となる。

アップデートに伴い HBase、Hadoop とも複数のモジュールで構成されるようになった。開発を分担するためと思われるが、元々、パッケージ間の依存性について注意を払ってきたわけではなく、不自然な分離が行われ、またモジュール間の依存関係が明らかでない。また、数多くの 3rd パーティライブラリに依存するようになり、一部、バージョンの異なる同じライブラリが含まれるなどの混乱が生じている。

7. ツールのアップデート

現在、データ移行ツールに対して、新しいバージョンの HBase/Hadoop への対応を行ってきている。ただし、新しく構築したクラスタの設定が不十分であることが分かり、パラメータの再検討・修正を行う必要があるため、ツールの動作確認にまでは至っていない。

データ移行ツールでは前述のように、HBase に予め用意されている MapReduce タスクの実装のロジックを参考にしつつ、独自に MapReduce を利用するロジックを作成してきた。具体的には、HBase に含まれるクラス HFileOutputFormat に記述されるロジックを参考にしているが、新しいバージョンの HBase では、HFileOutputFormat そのものの使用を強要するように API を変更・削減してきている。特に、HBase が内部的にどのように HDFS 上のストアファイルを作成・管理するかの詳細については、積極的に隠ぺいされている。

そもそも、詳細を隠ぺいして公開 API だけにアクセスさせるのは、後方互換性を保つためであり、これは本末転倒である。実用性から見ても、我々の目的にかなう実行速度を得るためには、HBase の内部詳細へのアクセスが必要である。このため、公開 API かどうかにこだわらず、必要に応じて HFileOutputFormat と同じロジックを用いることを優先した。

また、単に可視性を制限されるだけにとどまらない変更については、ロジックを丁寧に追ったり、HBase のソースコードのレポジトリから変更が行われた過程を追ったりすることで、修正方法を検討した。

なお、データ収集ツール及びデータ閲覧ツールについては、データ移行ツールのように HBase の内部詳細にアクセスする必要がないため、アップデートはそれほど困難でないと予想される。

8. 今後の対応

今後の作業予定及び対応について、以下に箇条書きにて挙げる。

- クラスタの設定を正しく行い、データ移行ツールの動作確認を行う。また、他のツールのアップデート及び動作確認を行う。
- 運用を見据えたクラスタの再構成を行う。
- ビックデータを扱っているというにはまだ容量が小さい。クラスタの増強にしたがって定期的な評価が継続的に必要だろう。
- MapReduce を利用した運転データの解析により何か有意義なものが得られないかを検討する。
- HBase/Hadoop はバージョン依存性が強く、特定のバージョンにのみ対応するクライアントアプリケーションを作成することになる。ただし最近のアップデートでは通信プロトコルに Protocol Buffers[8]を使うように変更されており、近い将来、互換性のあるクライアント専用のライブラリが作成されることが期待される。

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://hbase.apache.org/
[3] http://hadoop.apache.org/
[4] N.Kikuzawa, et al., “STATUS OF J-PARC OPERATION DATA ARCHIVING USING Hadoop AND HBase”, Proceedings of the 10th Annual Meeting of Particle Accelerator Society of Japan, August 2013.
[5] http://epics-jca.sourceforge.net/jca/
[6] http://controlsystemstudio.org/
[7] http://www.eclipse.org/
[8] https://code.google.com/p/protobuf/