Micro Focus Visual COBOL 2.3J for x64/x86 Linux

Hadoop/Spark 連携アプリケーション 動作検証 検証結果報告書

企業や社会インフラが扱うデータが爆発的に増大し続ける現代においてデータ処理の高速化は切要な課題です。このようなビッグデータの処理に対して有効なソリューションとして近年注目を集めるのが Apache Hadoop や Apache Spark といった分散並列処理を実現するフレームワークです。これらはそれぞれ Java、Scala で開発されているため、COBOL アプリケーションへの適用は困難と考えられがちです。しかし、COBOL for JVM の登場により COBOL も JVM 言語の 1 つに加わり、国際規格に基づく手続き型で書かれたプログラムであっても JVM アプリケーションとして運用することが可能となりました。つまり、これらのフレームワークで扱える形式へビルドが可能です。長時間に渡る処理を要するバッチアプリケーションで特に COBOL は活用されることが多く、事務処理専用の言語として策定された COBOL で書かれたロジックを他言語に置き換えることは非常にリスクの高い行為です。COBOL for JVM はこの COBOL 資産を Hadoop や Spark のフレームワーク上で理論上、並列分散処理可能とします。本書では ANSI 85 規格に準拠した手続き型の COBOL プログラムが Hadoop 及び Spark のフレームワーク上で動作することを検証しました。また、シングルノード及びレガシー環境で計測した処理時間との比較も行い並列分散処理化の効果も検証しています。

MICRO FOCUS

目次

1. 検証目的及び概要 1

1) 検証目的 1

2) 検証概要 1

2. 検証環境詳細....3

3. 検証アプリケーション詳細....4

4. 検証方法....7

5. 検証結果....8

6. 考察 9

1. 検証目的及び概要

1) 検証目的

企業や社会インフラが扱うデータは年々増加の一途を辿っています。この情報爆発時代、動画データ、画像データ、音声データ、ログファイル等といった非構造化データの増大に目がいきがちですが、COBOL アプリケーションによりアクセスされる RDB やデータファイルで管理される企業が扱う構造化データに関しても年率で 15 ～ 20％ 増加するという調査結果もあるようです[1]。つまり現時点で想定時間内に終了するバッチアプリケーションが数年後データ量が増えることによって時間内に終了できなくなるということも起こり得ます。また過去データが年々累積することでより多くのデータに対して集計等を行いたいという要件も求められるようになるかもしれません。

従来、COBOL バッチアプリケーションの処理時間短縮に対する取り組みとしてはスケールアップによる処理高速化が中心でした。一方、非構造化データも扱うビッグデータの分野では Apache Hadoop や Apache Spark のような分散並列処理を実現するフレームワークを活用し大きくパフォーマンス改善を実現した事例が近年多く確認されます。これらはそれぞれ Java、Scala で開発され JVM アプリケーションを対象としているため、COBOL とは無縁と思われがちです。しかし、Micro Focus は COBOL を JVM 言語の１つとして扱うことを可能とする COBOL for JVM という技術を提供しています。本機能を利用することで国際規格に準拠して書かれた手続き型の COBOL プログラムを JVM クラスとして運用し、これらのフレームワークより扱うことを可能とします。

本書では、COBOL for JVM により典型的な手続き型の COBOL プログラムから生成された JVM クラスが Hadoop や Spark 上で正しく動作することを検証します。また、シングルノード及びレガシー環境で計測した処理時間との比較も行い並列分散処理化の効果も検証します。

2) 検証概要

本検証ではアプリケーションの実行プラットフォームとして Oracle 社が提供する Oracle Big Data Appliance を利用しました。Oracle Big Data Appliance は Hadoop 処理に必要なソフトウェア等が予めエンジニアードされた複数のノードから構成されるプラットフォームです。本プラットフォームを利用することで Hadoop や Spark のセットアップが大幅に簡略化されます。Oracle Big Data Appliance の

[1] 【参考】Oracle Corporation 「Boost Your Database Performance 10x with Oracle SecureFiles」
IDC, Gartner の調査を基とした統計データ
http://www.oracle.com/technetwork/database/performance/boost-your-database-performance-10x-130376.pdf

各ノードには Micro Focus Visual COBOL 2.3J for x64/x86 Linux がサポートする Oracle Linux が OS として利用されており、COBOL アプリケーションの開発・運用が可能です。本検証では各ノードに Visual COBOL をインストールし、JVM クラス化された COBOL アプリケーションを並列分散稼動させました。

Hadoop で並列分散処理をさせるには MapReduce にロジックを落とし込む必要があります。本検証では架空の損害保険会社が年間の売上額集計に利用するバッチアプリケーションを Hadoop で処理できるよう加工しました。このうちこの企業がこの企業のビジネス要件に基づきカスタマイズ作成した保険料算出ロジックについては COBOL のまま維持しました。一方、これ以外のデータ IO の部分のような汎用的なロジック並びにメインロジックについては MapReduce で扱えるよう変更しました。この MapReduce は Java 等の JVM 言語で組み上げることが可能ですが、非常に複雑なプログラミングを要します。そこで本検証ではノーチラス・テクノロジーズ社が提供する Asakusa Framework を利用しこの部分のロジックを平易に組み上げました。

並列分散処理による効果を確認すべく、オープン環境で従来より利用されるネイティブの COBOL アプリケーション及びレガシー環境で稼動するレガシー COBOL アプリケーションというシナリオで更に２パターンの検証も行っています。各アプリケーションが利用するデータは全環境共通となります。各パターンの概要は下表の通りとなります。

パターン	環境環境概要	アプリケーション概要
並列分散処理	Oracle Big Data Appliance 上の６ノードを利用	[ビジネスロジック] - COBOL for JVM を利用し COBOL ソースを JVM クラスへビルド - 利用した COBOL ソースは他パターンと共通 [メイン、IO ロジック] Asakusa Framework が提供する DSL を主に利用し MapReduce のロジックを組み上げ
オープン環境におけるネイティブバッチアプリケーション	Oracle Big Data Appliance 上のノードを１つ利用	全ロジックを COBOL で組み上げ Oracle Linux のネイティブアプリケーションとしてビルド
レガシーバッチアプリケーション	Micro Focus が保有する IBM メインフレームを利用	- 全ロジックを COBOL で組み上げ - ソースは基本的に上のパターンと共通

2. 検証環境詳細

本検証に利用した環境に関する情報を以下に記します。

＞ 並列分散処理環境

ハードウェア	ソフトウェア
Oracle Big Data Appliance X5-2[2] (Hadoop Server 6台で構成) [各 Hadoop Server のスペック] CPU Intel® Xeon® CPU E5-2699 v3 @ 2.30 GHz 18 core × 2 メモリー 128 G bytes OS Oracle Linux 6 Release 7	＞ Cloudera's Distribution including Apache Hadoop 5.7.1 - Apache Hadoop 2.6.0 - Apache Spark 1.6.0 ＞ Asakusa Frame Work 0.8.1 ＞ Micro Focus Visual COBOL 2.3J Update 1 for x64/x86 Linux

＞ オープン環境

ハードウェア	ソフトウェア
Oracle Big Data Appliance X5-2 における Hadoop Server を 1 台利用 スペックは並列分散処理の検証に利用した各ノードのスペックと同じ	Micro Focus Visual COBOL 2.3 Update 1 for x64/x86 Linux

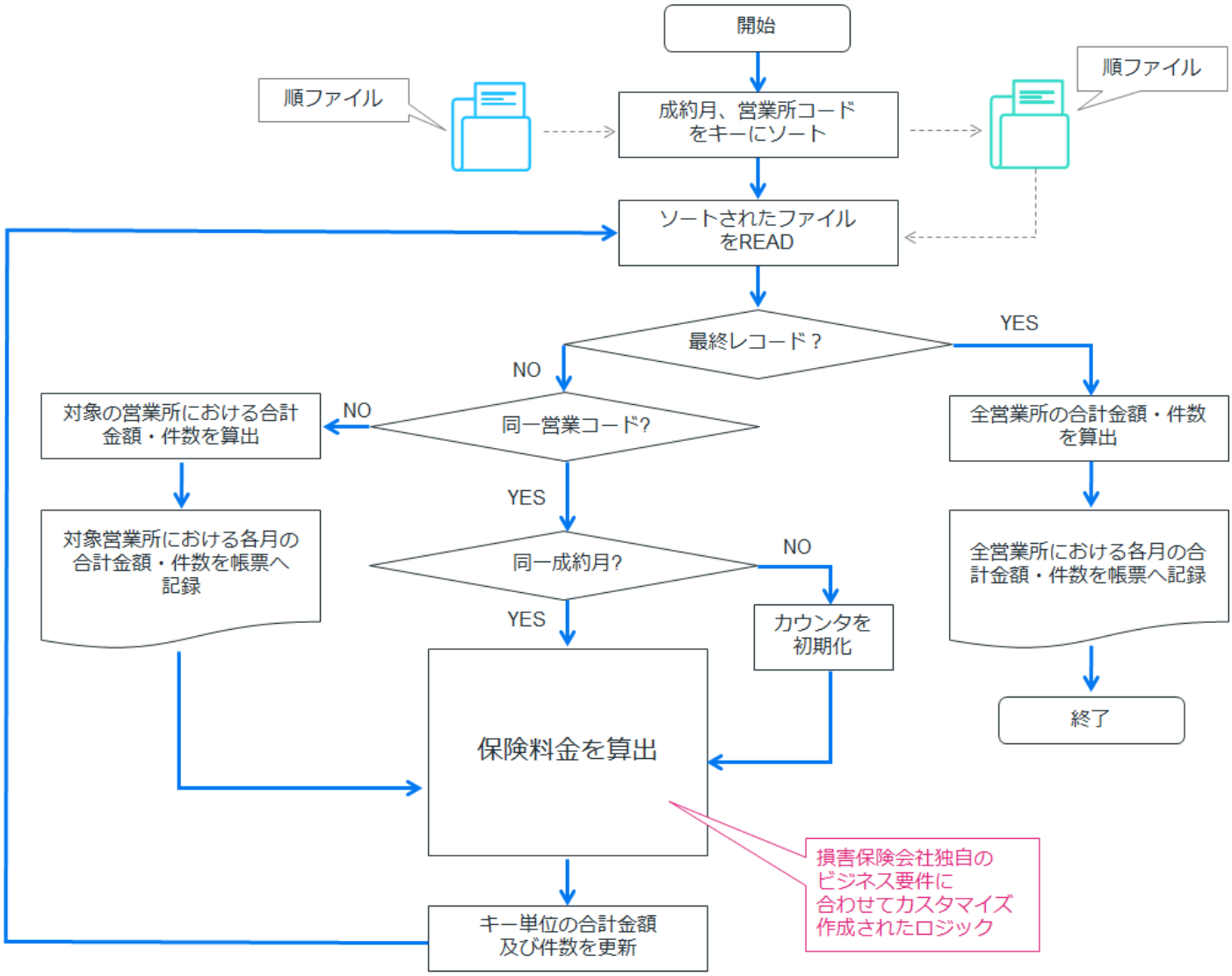
＞ レガシー環境

ハードウェア	ソフトウェア
IBM z13 2965-N10 capacity model E03 利用 CPU 2965(3 CPUs) メモリー 32 M bytes OS Z/OS 02.02.00 MIPS 約 353	＞ IBM COBOL FOR OS/390 & VM 2.1.1 ＞ Z/OS DFSORT V2R2 ＞ JES2 Z/OS 2.2

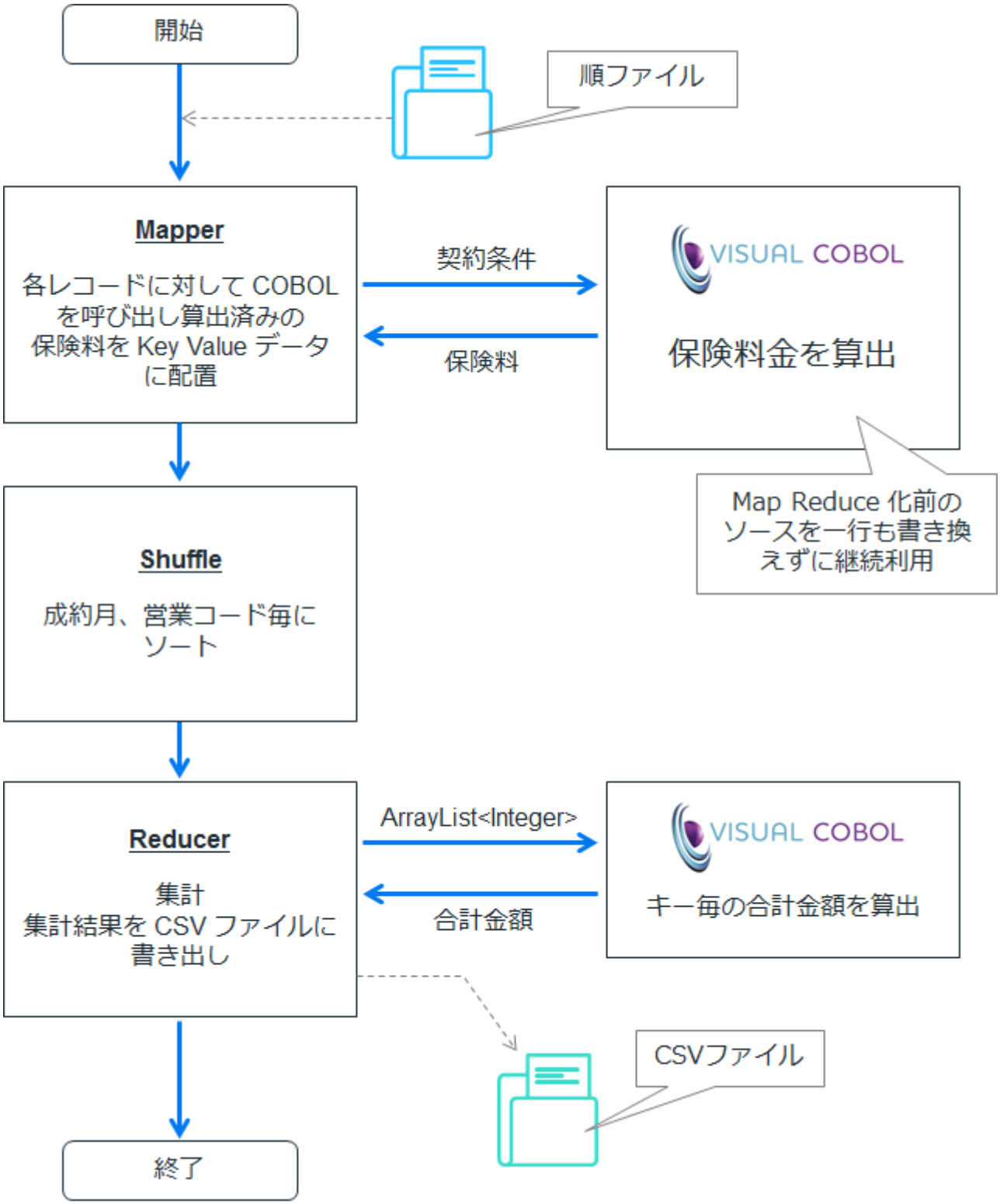
[2] 本書執筆時点(2017 年 2 月)現在の最新版は Oracle Big Data Appliance X6-2 となります。

3. 検証アプリケーション詳細

本検証では架空の損害保険会社が利用する過去１年間の自動車保険の売り上げを集計するバッチアプリケーションを例にとり検証しました。このアプリケーションはオンラインアプリケーションにより日々溜め込まれた成約情報が記録された順ファイルを入力ファイルとして使用します。本ファイルには成約時に指定された保険条件が記録されており、この条件を基にアプリケーション実行時点における保険料算出ロジックを用いて保険料を算出します。アプリケーションは算出した保険料並びに契約数を月別で集計します。以下にこのアプリケーションが並列分散環境向けにカスタマイズされる前の処理フローを記します。オープン環境並びにレガシー環境の想定で用意した環境ではこちらのアプリケーションが動作します。

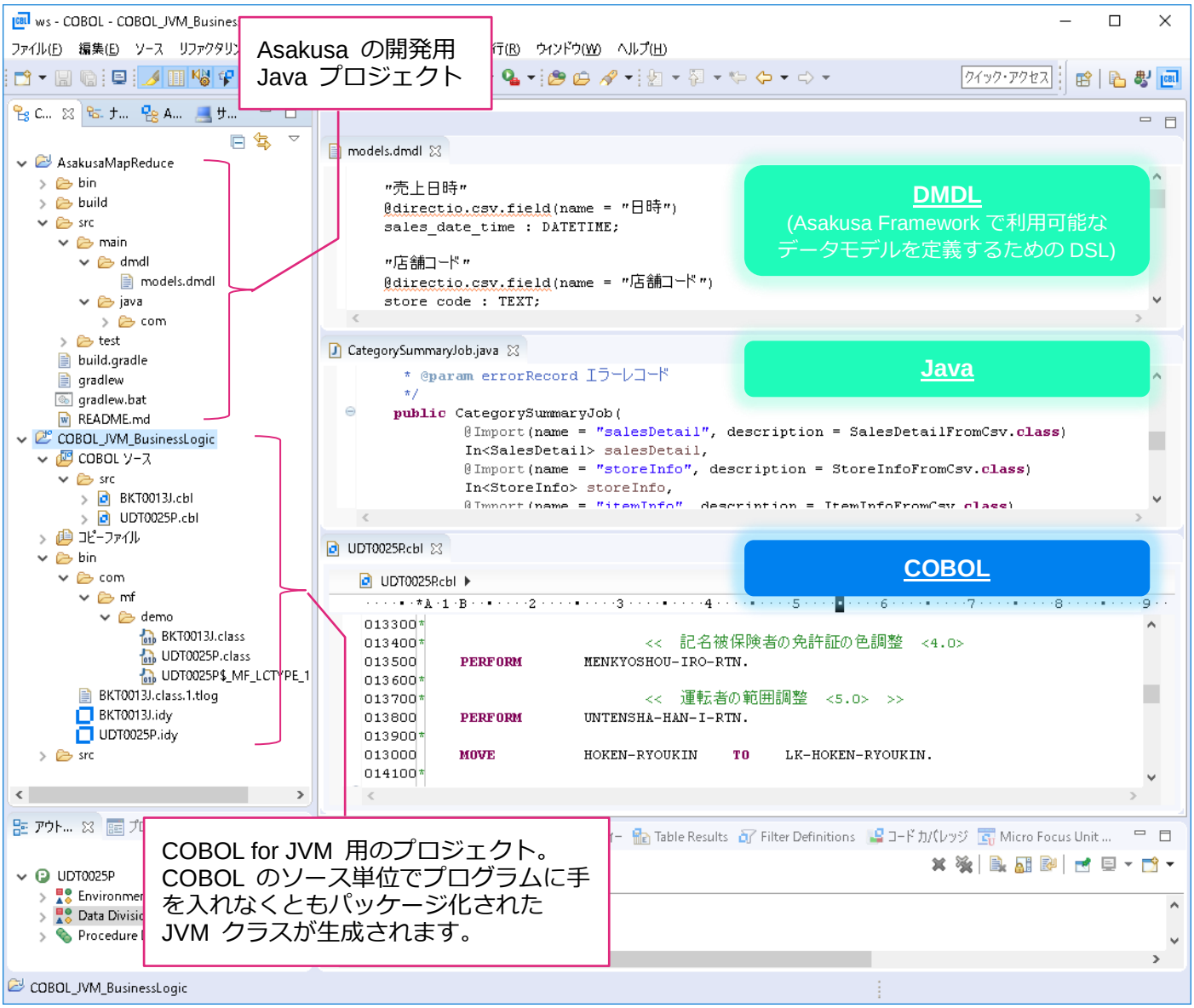


下図は先のアプリケーションを並列分散処理可能なかたちにカスタマイズした後の処理フローイメージとなります：



先のアプリケーションは Hadoop で動作させるにあたり MapReduce のロジックに落とし込みます。ファイル IO 部分並びにメインロジックについては Asakusa Framework を使って MapReduce の処理を実現できるよう書き換えました。この書き換えをしたロジックについてはこの架空の企業独自のロジックではないため特に問題はありませんでしたが、この企業のビジネス要件に沿って組み上げられた保険料算出ロジックについては代用が効きません。そこでこのロジック部分（図中の「保険料算出」の部分）については COBOL for JVM を利用し JVM クラスへリビルドし JVM 上で動作する MapReduce のロジックより利用できるようにしました。リビルドの際 COBOL のソースは一行も書き換えをしていません。また、Reducer でキー毎に合計金額を算出する箇所で、COBOL の強みである演算処理を利用すべく集計演算で利用する COBOL ロジックも用意しました。変更前のアプリケーションでは集計結果はテキストベースの帳票に出力していましたが、様々な BI ツール等からの利用も想定し、これらから柔軟に digest 可能な CSV 形式で出力するようにしました。

COBOL は Java の開発環境として汎用的に利用される Eclipse IDE 上で開発することが可能です。同一の Eclipse ワークスペースにて Java と COBOL for JVM のプロジェクトが共存できます。次ページに示す図は Asakusa Framework によるアプリケーション開発を支援する Jinrikisha を用いて作成した Eclipse プロジェクトと COBOL for JVM のプロジェクトが同一ワークスペースに同居するようすを例示したイメージです。COBOL のソースフォルダを Java デバッグ構成のソースルックアップパスに追加すれば Java から開始されたデバッグであっても COBOL が呼ばれた時点で COBOL ソースへデバッガが移ります。COBOL ソース上では COBOL の変数に格納された値の参照や条件付きのブレークポイント等 Micro Focus が COBOL 向けに作りこんだデバッグ機能を利用できます。



4. 検証方法

200 MB、2 GB、20 GB とバリエーションをもたせた入力ファイルを用意し各シナリオパターンの環境で取り込ませ処理に要した時間を計測しました。入力ファイル中のデータは各シナリオパターンで共通です。それぞれの入力ファイル、シナリオパターンの組み合わせで 5 回ずつ流した平均値を取得し、その値をベースに比較を行いました。参考値として並列分散処理環境に限り 200 GB の入力ファイルも用いて検証を行いました。並列分散処理環境では Hadoop、Spark ともに正常に動作することを確認しました。

5. 検証結果

本検証における計測結果の概要を下表に記します：

	入力ファイルのサイズ			
	200 MB	2GB	20 GB	200 GB
並列分散処理	00:00:41.065*1	00:00:49.367 *1	00:01:16.767*1	00:06:05.588*2
オープン環境	00:00:19.897	00:03:21.683	00:34:31.370	n/a
レガシー環境	00:02:24.600	00:19:36.600	03:21:58:982	n/a

(単位: hh:mm:ss.millisec)
*1: Spark を利用
*2: Hadoop を利用

200 MB、2 GB、20 GB のファイルに対してはインメモリで処理できるサイズだったこともあり Spark の方が Hadoop よりも高速に処理していました。一方、200 GB となりますと Hadoop の方が高速に処理していました。一般的に Hadoop と Spark は優劣比較されるものではなく対象のデータサイズ等の条件に応じて適切に選択されるもののようですので、本検証においてもよい速い数値を出した方の技術が実際に選択されるという想定の下、上の表のように記録しています。
各パターンとの対比についても下表にまとめます。

	200 MB	2GB	200 GB
		並列分散処理との対比	
オープン環境	48.45%	408.54%	2698.27%
レガシー環境	352.13%	2383.39%	15786.81%
		オープン環境との対比	
レガシー環境	726.75%	583.39%	585.07%

6. 考察

前章の結果を見ますと入力ファイルが 200 MB の時点では従来 COBOL バッチアプリケーションが運用されてきたパターンの方が高速に処理しています。Hadoop や Spark は処理開始時に対象のデータを各ノードへ振り分ける等の処理を必要とします。この起動に伴う処理をしている間にオープン環境で稼動するネイティブの COBOL アプリケーションは処理を完了させたため、このような関係になったと推測します。一方、対象のデータを拡幅させますとこの関係は逆転します。オープン環境のネイティブアプリケーションはデータサイズと処理時間が線形回帰のような関係で上昇しますが、並列分散で処理した場合はデータサイズの増加に対して処理時間の伸びが緩やかです。そのため、データサイズを 2 GB、20 GB と増やしますと並列分散処理の方がそれぞれ 4 倍、27 倍の速度で処理できていることがわかりました。この数値だけに着目しますと従来のネイティブ COBOL アプリケーションが要した時間が際立ちそうですが、メインフレーム上で処理した時間と比較すると一貫して 6、7 倍の速度で処理できています。つまり、この結果は巨大なデータを扱う場合、並列分散で処理することでシングルノードでもともと高速な処理を実現するネイティブアプリケーションを凌ぐスピードで処理できる可能性を示唆しています。

例えば、20 GB のデータ処理に関してはメインフレーム環境と Hadoop を利用した場合とでは 150 倍以上差があります。これまでメインフレーム上で 3 時間以上かかっていた処理が Hadoop 化することで 1 分強で完了します。これ程までの高速化が実現できるのであればこれまでのように夜間の営業時間外に処理しなくとも営業時間中オンバッチのように利用してコスト削減を図ることも可能かもしれません。もしくは、処理対象のデータを更に追加して処理結果の価値を高めるといった取り組みもこれまでの処理時間内で実現できます。

COBOL for JVM は COBOL を JVM 言語としても扱うことを可能としました。これにより様々な JVM 向けの技術から COBOL 資産を利用できるようになりました。ビッグデータを並列分散で高速処理すべく実績のある Hadoop を利用したいというケースであっても、本検証で確認したように既存の COBOL 資産を利用することが可能です。別の目的で別の JVM 向け技術を利用したいというケースでも COBOL for JVM により COBOL 資産の活用が可能です。その一方で COBOL は引き続き CPU ネイティブにコンパイルすることも可能です。本検証結果のようにデータの特性によっては、並列分散処理化したした方が効果が望めず、アプリケーションに手を入れた分コストが嵩むこともあります。このように様々な選択肢ができ可能性が拡がった分、JVM 化することは効果があるか、JVM 化するのであれば継続利用すべき COBOL 資産は、といったかたちで要件に応じて適切な技術を見極めることが成功の鍵となると考えます。

マイクロフォーカス株式会社
〒106-0032 東京都港区六本木 7-18-18 住友不動産六本木通ビル 9 階
Tel: 03-5413-4800 http://www.microfocus.co.jp/