ビッグデータシステムにおける
バグの実証的研究

中村 友海

平成 30 年 2 月

電気情報工学科

概 要

現在、ビッグデータシステムがデータマイニング、情報検索、インターネット上での広告、および購買行動の解析など多岐にわたって利用されている。このようにしてビッグデータシステムの用途が広がって行くにつれて、システムそのものの信頼性も重要になって来ている。そこで、ビッグデータシステムの信頼性を担保するためには、システムで発生したバグを理解することが重要である。現在、様々なソフトウェアシステムに対してバグの理解や修正に関する研究が行われているが、ビッグデータシステムにおけるバグそのものの分類や特性分析に関する研究はほとんど行われていない。

本研究では Apache が公開しているビッグデータを扱う分散処理システム Hadoop と Spark のバグそのものへの取り組みとして、バグレポートを集計し、精製されるバグがどのような特性をもつか調べることによって、バグが発生した際に分類を同定する足がかりとなる役に立つのではないかと考える。これによって、今後のビッグデータシステムの発展に対して信頼性を確保していくための導となると考えられる。

目 次

第1章

はじめに

本章では、今回の研究の背景、目的、そして本論文の構成について述べる。

1.1　研究の背景

今現在、さまざまな、データマイニング、情報検索、サジェスト機能、購買行動等、多くのビッグデータシステムが使用されている。ビッグデータシステムの用途はインターネット上での広告、検索エンジン、購買行動の解析など多岐にわたって利用されている [1] [2]。このようにしてビッグデータシステムの用途が広がって行くにつれて、システムそのものの信頼性も重要になって来ている。そこで、システムの信頼性を担保するためには、システムで発生したバグを理解することが重要である。現在、様々なソフトウェアシステムに対してバグの修正や研究が行われている一方、ビッグデータシステムについてはバグそのものの特性や分類の同定といったことへの取り組みが先行研究としてほとんど行われていない。なぜならビッグデータを扱う分散処理システムは、アルゴリズムを集中的に取り扱う性質があること、大規模なデータに対して適用を行うことといった性質があるためだ。

そこで、本研究ではビッグデータを扱う分散処理システムのバグそのものへの取り組みとして、バグレポートを集計し、精製されるバグがどのような特性をもつか調べる。こうした背景によって、今後、バグが発生した際に分類を同定する足がかりとなる役に立つのではないかと考える。

1.2　研究の目的

ビッグデータシステムのバグを統計的分類で表し、どのようにバグがあらわれる傾向にあるのかを分析する。

1.3　研究の方針

本研究では、細心の注意を払いつつビッグデータシステムのバグに関する実証的研究を行う。Apache Hadoop [3] , Apache Spark [4] について分析する。具体的には Apache が公開しているバグレポートを調べ、バグの様々な特性について分析し考察する。

1.4　本論文の構成

1 章では研究の背景、目的、方針、構成について述べる。2 章については本論文での研究対象である Apache Hadoop, Apache Spark と研究方法について述べる。3 章では前章にて記した方法に基づいた研究結果を述べる。4 章では前章での結果にも続いた考察を述べる。5 章では本研究のまとめ及び今後の課題について述べる。

第2章

研究対象と研究方法

2.1　研究対象

本研究で取り扱う対象である、Apache Hadoop, Apache Spark について概要を述べる。

2.1.1　Apache Hadoop とは

Hadoop は、Doug Cutting によって開発され 2004 年に Doug Cutting と Mike cafarella によって最初のバージョンが実装された。Doug Cutting は、広く使われている検索ライブラリである Apache Luche の開発者である。Hadoop の起源は、Luche プロジェクトの一部であるオープンソースの Web 検索エンジン、Apache Nutch である。

また、 Hadoop はオープンソースで公開されているビッグデータの分散処理技術のことである。Hadoop の特徴として、複数のハードディスクを用いて構成したクラスタ上で、並列に処理することで高速でデータ処理を行う Hadoop MapReduce というフレームワークがある。同様のシステムも存在しているが、データ処理の性能は Hadoop が抜きん出ている。なお、このようなシステムには 2 つの問題がある。1 つ目の問題はハー

ドディスク障害への対処であり、2 つ目の問題は多くの場合、データ処理には何らかの形でデータを結合できなければならないというものである。ここで 2 つの問題に対する解決方法として、1 つ目には、Map-Reduce フレームワークを用いた処理系がタスクに障害が起きたことを感知し、代わりのタスクを健全なマシン上に再度スケジュールすることで解決している。2 つ目には、ハードディスクの読み書きの問題を抽象化し、キーと値の集合に対する演算処理に変換するプログラミングモデルを提供することによって解決している [5] [6]。

次に Apache Hadoop の公式サイトの URL を記載する。

http://hadoop.apache.org

2.1.2　Apache Spark とは

Spark はもともとカリフォルニア大学バークレー校 AMPLab の研究プロジェクトとして始まったものである。2010 年初頭にオープンソースとなり、2013 年に Apache Incubator Project に採択されたことで、Apache Spark となった [7]。Spark は、Hadoop データ用の高速かつ一般的な計算エンジンであり、ETL、機械学習、ストリーム処理、グラフ計算など、幅広いアプリケーションをサポートするシンプルで表現力豊かなプログラミングモデルを提供するものである [3]。

Spark の特徴として大きなものを 3 つ挙げる。1 つめは、Spark はもともとデータ処理に対して、連続処理の中で無駄なディスクやネットワークの I/O を起こさないようにして繰り返し起きる処理を置き換えていくことで高速化を実現していることである。2 つめは、Spark は、並列分散環境を意識することなく処理できるようにすることを指針にしているため、RDD とよばれるデータセットを用い、API で変換を記述することで並列分散処理が実現することである。3 つ目は、SQL ライクなクエリを処理できる Spark

SQL や機械学習ライブラリ MLlib を用いて異なるタイプのアプリケーションを一つの環境で扱えることである [7]。

次に Apache Spark の公式サイトの URL を記載する。

https://spark.apache.org

2.2　研究方法

本章では、2 つのビッグデータシステムのバグについてデータの収集、そのバグが報告されているバグレポートの分類、分類したバグレポートから得た情報についての統計計算、それぞれについて述べる。

2.2.1　データ収集

前述した Spark と Hadoop の報告されたバグを分析するために次の場所にある JIRA のバグレポートを蓄積する場所を分析する。

https://issues.apache.org/jira/

本来、ソフトウェアシステムのバグリポジトリとソース管理リポジトリをダウンロードし、報告されたバグ、コミット、およびそれらのリンクに基づいて解析を行うものである。しかし、Apache Software Foundation の多くのプログラムは JIRA リポジトリに報告されたバグとコードコミットしソース管理リポジトリの位置がわかるリンクを一緒に記録している。多数の無作為にサンプリングされたリンクとバグレポートに関する最初の分析では、ほとんどのリンクが正しいと判断し、ほとんどのバグレポートはそれらを修正したすべてのコミットに適切にリンクしている。 したがって、報告されたバグ、コミット、およびそれらのリンクに基づいて分析を行う。なお、本研究では、1 つのバグレポートに対して 1 つのバグが記載されているものとして研究を行う。

2018 年 1 月中旬現在の最新のバージョンは Hadoop が 3.0.0 [8]、Spark が 2.2.1 [4] である。2.2.1 で詳細を記す分類では、Hadoop , Spark の JIRA リポジトリに含まれているさまざまな種類のバグ（改善、新機能、タスク、テスト、要望、ブレインストーミングなど）を Hadoop は 6057、Spark は 8969 のバグレポートの中から先頭の約 1000 個強 (2017 年 1 月 15 日現在) の中から特別な意図なくピックアップした 100 ずつのものを用いる。以下に表 2.1 を記す。

表 2.1: ビッグデータシステムにおける閉じられたバグレポートの数

Application	サイズ	バグ数	期間
Hadoop	2,812,309	8,980	5.26 years
Spark	679,457	6,058	11.87 years

2.2.2　バグレポートの分類

マニュアルバグレポートの分類では、2.2.1 で記した 200 個のバグレポートのバグを手動で表 2.2 の項目のどれかに該当するように各バグレポートの記述を調べながら分類する。分類に際して、先行研究として Ferdian Thung 諸氏が Seaman 諸氏が提案した分類 [9] を参考にして作成した種類 [10] を使用する。表 2.2 に分類の種類名とその意味を示す。

表 2.2: バグの種類

アルゴリズム/メソッド	アルゴリズム/メソッドの実装が、予想される動作に従わない場合。
割り当て/初期化	変数値の割り当てによってエラーが生じた場合。
チェック	エラーまたは間違ったエラーメッセージにつながりうることを表すのに必要な確認がなかった場合。
データ	データ構造が誤っていた場合。
外部インターフェース	他のシステムやユーザとのインタフェースでエラーが発生した場合。具体的には、他のシステムから廃止予定のメソッドを使用するなど、使い易いように外部インタフェースを所有する必要があるようなときなど。
内部インターフェース	継承契約違反、他のクラスからの操作の誤った使用など、同じシステムの別のコンポーネントとのインタフェースでエラーが発生した場合。
論理	条件文の不適切な表現（if、while など）の場合。
機能不全	不適切な変数やメソッド名、メソッドの実装に対する誤った文書化など、非機能要件が原因の場合。
タイミング/最適化	デッドロック、高メモリ使用率などの並行性やパフォーマンスの問題を引き起こすエラーの場合。
設定	機能に影響する非コード（設定ファイルなど）のエラーの場合。
その他	上記の種類に該当しないその他のバグ。

また、分類分けに際し使用したバグレポートの例を図 2.1、その参考リンクとしてしばしばあげられる github についても図 2.2 に記載する。

今回は手動でするにあたって、200 個全てのバグレポートについて 3 回分類し、異なる結果が出るたびになぜ違うのかを照会することで極力バグレポートの分類が正しくなるように試みた。

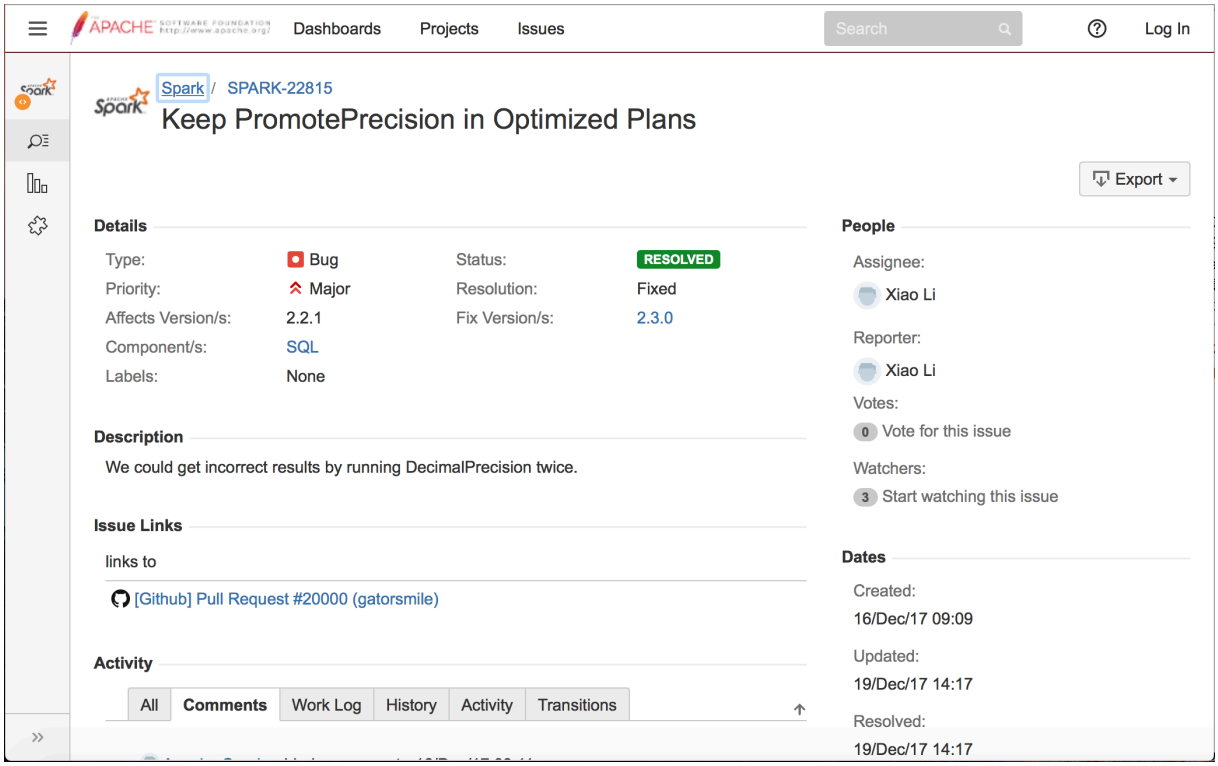

図 2.1: Apache の提供するバグレポートの例

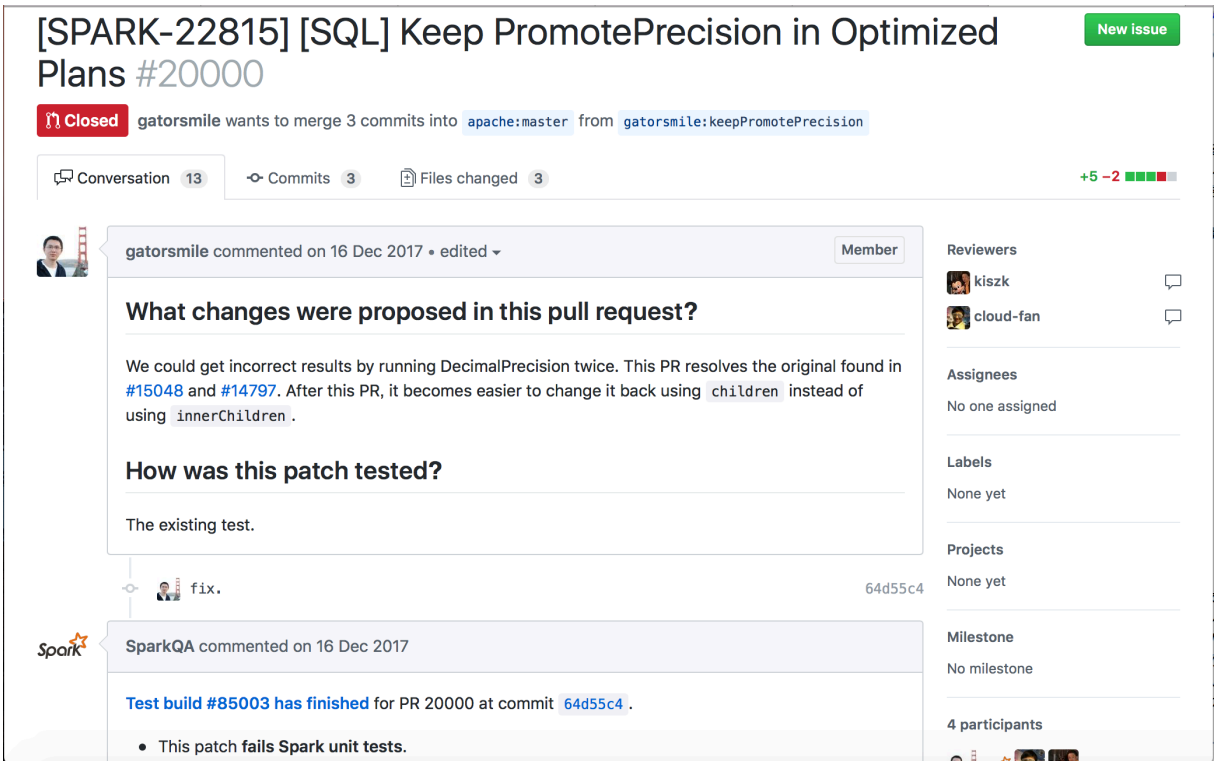

図 2.2: プルリクエストに用いる github の例

2.2.3　統計計算

前述した種類別に分類したバグについて様々な統計計算をする。これらの統計によって、多くのビッグデータシステムでのバグの出現、バグの様々な種類分け、バグの種類と重大度の間にある関係性、修復への労力と時間の関係、様々なシステム全体のバグの重要度、といった様々な研究の疑問に回答することに役立つ。

第3章

研究結果と考察

本章では、研究結果と、その考察を記す。3.1 項では 2 つのビッグデータシステムにおけるバグの密度を調べる。3.2 項では各種類のバグの発生割合を分析する。3.3 項でバグが各種類毎に各重大度がどのような割合で発生しているかを分析する。

3.1　バグの頻度

本項では、2 つのビッグデータシステムである、Hadoop と Spark について、2 章で記したバグ数に基づいて、システム内の過去にクローズされたバグの数をコード数で割り、それに 1000 をかけた、1000 行あたりにバグがいくつ現れるかをまとめたものを表 3.1 に表す。また、表 3 の最後の列は、システムごとに年間発生したバグの平均数を示している。 これらのことから、Apache 開発者は開発環境として土台となる Hadoop より、それを利用して分散処理を行う Spark の方をバグの修正をする必要があると考えられる。

表 3.1: バグの密度

Applications	Bug Count Per kLOC	Bug Count Per Year
Hadoop	2.15	1704.32
Spark	13.21	510.03

3.2　バグの種類

バグレポートの記述にはあいまいなものがある。これらのケースでは、バグレポートにおけるテキストの記述のみならず、コミッターに対するコメントや、バグを修正するコードの変更内容も見る。こうすることでバグの性質についてより多くの情報を見つけ、バグレポートの種類の決定を行うことがより深くできる。

ここで、一見するとあいまいなバグレポートについてどのように分類の判断を下したかの例を図 3.1 と図 3.2 を用いて 1 つ、図 3.3 を用いてもう 1 つ記す。

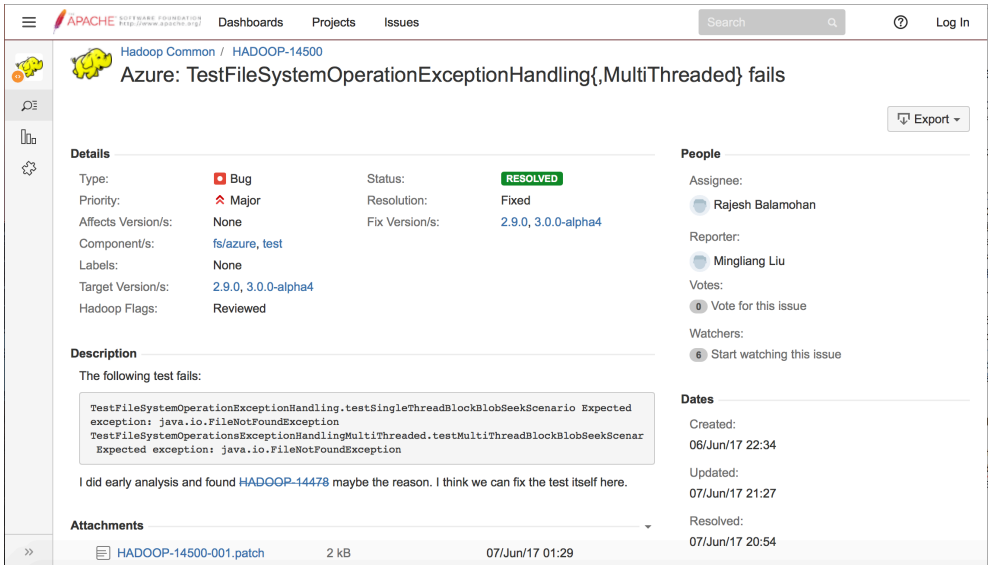

図 3.1: アルゴリズム/メソッドの例 HADOOP-14500

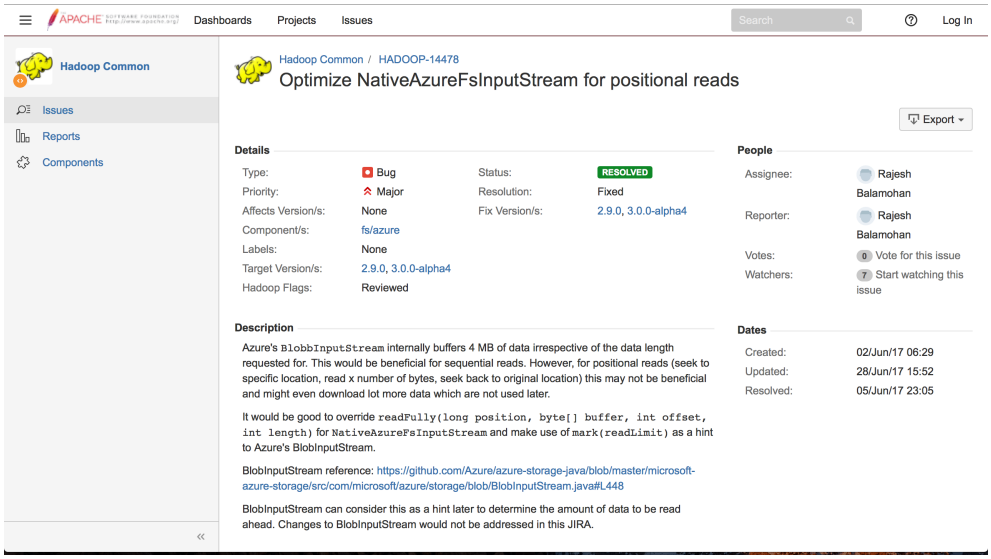

図 3.2: アルゴリズム/メソッドの例 HADOOP-14478

https://issues.apache.org/jira/browse/HADOOP-14500

1つ目の例について述べる。図3.1はHADOOP-14500というバグレポートのものである。タイトルと説明からはテストが失敗していることしかわからないが、説明の最後に添えられている別のバグレポート HADOOP-14478 を開くと、タイトルから NativeAzureFsInputStream を最適化する手法が間違っていたのではないかということがわかる。バグレポート HADOOP-14478 は図 3.2 また、最適化に際しては説明文から、位置読み取り（特定の場所を探す、x バイト数を読む、元の場所に戻す）の場合、有益ではないかもしれないし、後で使用されない多くのデータをダウンロードする場合があるかもしれないことを最適化する対象としたのではないかと推測できる。

なお、FsInputStream とは、RAF 形式のシーク能力を備えた汎用性の古い InputStream [11] であり、今回のバグレポートでは”It would be good to override readFully(long position, byte[] buffer, int offset, int length) for NativeAzureFsInputStream and make use of mark(readLimit) as a hint to Azure’s BlobInputStream.”とあるから、バッファ

数である 4MB を最大値とし、指定された値を読み込むメソッドを用いていることがわかる。

以上のことを踏まえて、元のバグレポートである、HADOOP-14500 のコメント欄を見ると Rajesh Balamohan によるコメントに”When earlier patch results were submitted “fs.contract.test.fs.wasb” was not added IIRC. So some of the tests were getting ignored as mentioedn in.”とある。以上のことから、バグの種類はアルゴリズム/メソッドに分類されるものであることがわかる。

2 つ目の例について述べる。図 3.3 は HADOOP-14500 というバグレポートである。タイトルと説明から、CRAN チェックが同じマシンを通過するため、旧型の部分的なダウンロードや Spark の部分インストールが残っていれば、テストは失敗することがわかる。Issues Links である [Github] Pull Request #20060 (shivaram) を図 3.4 に示す。ここでも、元のバグレポート同様の説明がなされていることから、やはり元のバグレポートの解釈は間違っていなかったことがわかる。以上からデータ構造に分類できるとわかる。

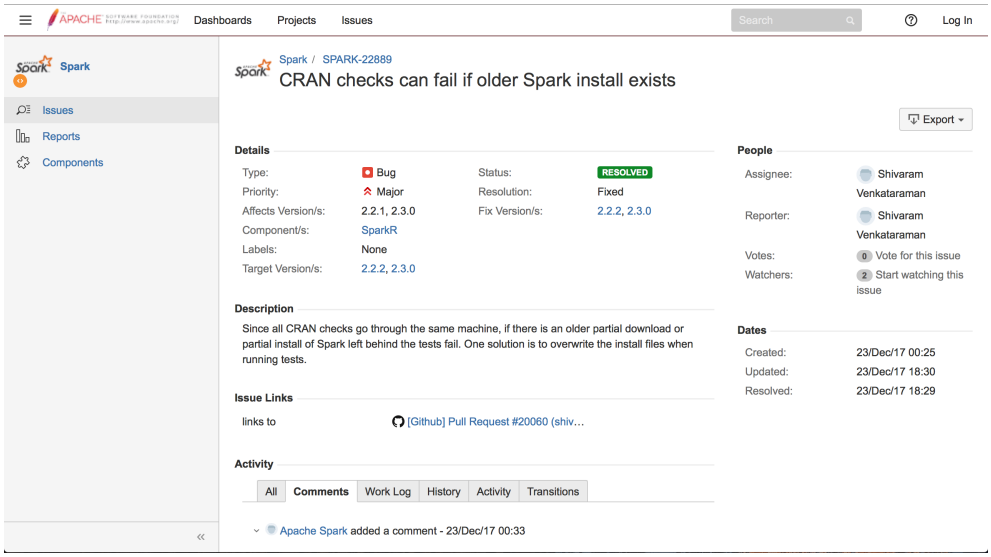

図 3.3: データの例 HADOOP-22889

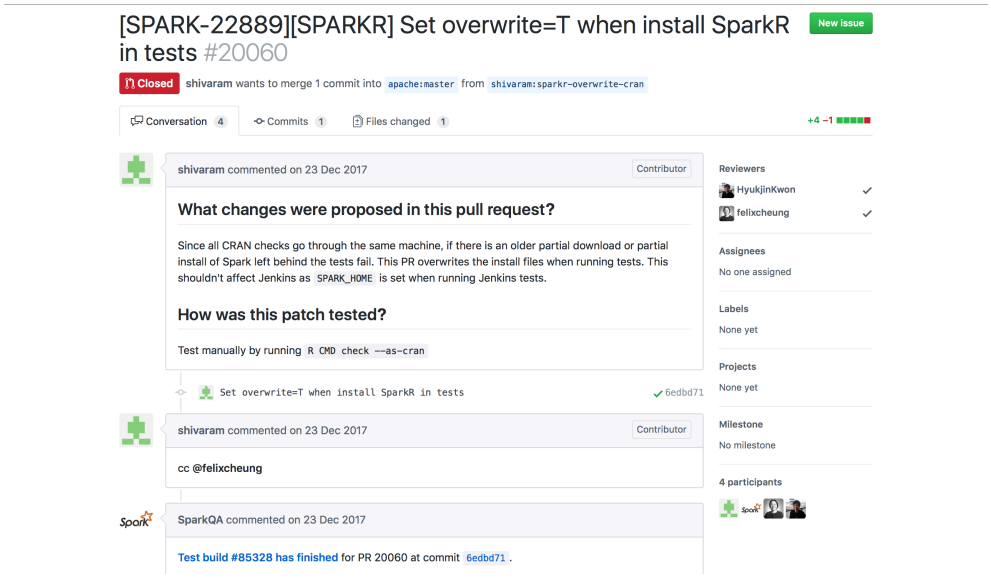

図 3.4: データの例 [Github] Pull Request #20060 (shivaram)

このようにして、2 つのビッグデータシステムから 200 個の手動でランダムに抽出されたバグを手動で種類分けする。11 の種類に基づくバグの分布を表 3.2 に示す。アルゴ

リズム/メソッド、チェック、内部インターフェースで分類されている。この 3 つの占める割合は合計で 46.5%もあるため、バグの種類分けをする際はまず、その他を含めた 11 の種類の内、この 3 つの種類のどれかに該当するのではないだろうかと推測する指針になると考えられる。

表 3.2: 種類ごとのバグの数とその割合

種類	数	割合 (%)
アルゴリズム/メソッド	38	19
割り当て/初期化	19	8.5
チェック	25	12.5
データ	11	5.5
外部インターフェース	14	7
内部インターフェース	30	15
論理	4	2
機能不全	17	8.5
タイミング/最適化	13	6.5
設定	20	10
その他	9	4.5

3.3　バグの重大度

次に、さまざまなバグの種類と報告された重大度との関係を調べる。バグ報告者が割り当てることができる重大度レベルは、Blocker, Critical, Major, Minor, Trivial の 5

つである。Blocker は最も重大度の高いものであり、Trivial は最も重大度の低いものである。表 3.3 は、バグの重大度レベルについてバグレポートの数と割合の分布を示している。

Blocker について、データにおける Blocker の割合は母数が 11 しかない故に 1 つのみで 9.09%という突出した値になっているため、偶然ずば抜けた値が検出された可能性はある。しかし、他の項目を見ても Blocker がしめる割合は 3%超えている。このことから、Blocker についてはアルゴリズム/メソッド、割り当て/初期化、チェック、データ、内部インターフェースはバグの総数 200 に占める全体の 60.5%をしめているので Blocker は偏在する特徴があるように考えられる。

また、種類内の総数が 4 と極めて少ない論理を除いて Major が 65%を超えているため、まずまず分類先は Major であるだろうと目星をつけて良いであろうと考える。

表 3.3: バグの重大度に応じたバグの個数と割合

種類	重大度	数	割合 (%)	種類	重大度	数	割合 (%)
アルゴリズム/メソッド	Major	29	76.31	論理	Major	2	50
	Minor	5	13.15		Minor	2	50
	Critical	0	0		Critical	0	0
	Blocker	2	5.26		Blocker	0	0
	Trivial	2	5.26		Trivial	0	0
割り当て/初期化	Major	14	73.68	機能不全	Major	12	69.23
	Minor	3	15.78		Minor	1	23.07
	Critical	1	5.26		Critical	3	7.69
	Blocker	1	5.26		Blocker	0	0
	Trivial	0	0		Trivial	1	0
チェック	Major	18	72	タイミング/最適化	Major	9	69.23
	Minor	4	16		Minor	3	23.07
	Critical	0	0		Critical	1	7.69
	Blocker	1	4		Blocker	0	0
	Trivial	2	8		Trivial	0	0
データ	Major	8	72.72	設定	Major	13	65
	Minor	2	18.18		Minor	4	20
	Critical	0	0		Critical	0	0
	Blocker	1	9.09		Blocker	1	5
	Trivial	0	0		Trivial	2	10
外部インターフェース	Major	12	85.71	その他	Major	6	66.66
	Minor	0	0		Minor	3	33.33
	Critical	2	14.28		Critical	0	0
	Blocker	0	0		Blocker	0	0
	Trivial	0	0		Trivial	2	10
内部インターフェース	Major	21	70				
	Minor	6	20				
	Critical	2	6.66				
	Blocker	1	3.33				
	Trivial	0	0				

3.4　バグの修正時間

バグの種類とバグ修正時間との関係を調べる。バグ修正の時間は、バグレポートが作成されてから、解決するまでのの経過日数を測定する。表3.4 は、さまざまな種類のバグが修正されるまでの経過日数の最小値、最大値、中央値、および平均値を示している。

バグレポートが提出されてからバグが修正されるまでの最小限の時間はわずか数秒から数十分のものである。このようなバグのほとんどは、開発者自身によってバグレポートが報告され、修正されている。バグの解決策を見つけて、ソース管理リポジトリに必要なコード変更を行う直前にバグをリポジトリに報告している。

本論文では、他のバグレポートとこのような開発者自身によってバグレポートが報告され、修正されている同一の扱いをしたが、修正に当たってバグレポートを提出する前に報告者が解決するための取り組みの時間について考慮できていないため、例外扱いするべきなのかもしれない。

一方で、最大バグ修正時間は、数ヶ月から数年かかることがある。最大バグ修正時間が 1400 日をこえるほど長い種類は順に、設定、アルゴリズム/メソッド、外部インターフェース、データ、チェック、内部インターフェースである。

これらの種類は長年にわたってわたってバグの修正に対して取り組まなくてはならない場合があることを念頭に入れなくてはならないであろうと考えられる。

また、割り当て/初期化、機能不全、その他に分類されるバグは最大でも 2 年前後で解決している。一方、論理に分類されるバグは最大でも 4 日弱と、解決まで早期なことがわかる。

バグの問題解決にかかる時間の目処が立たない場合、Blocker のような最優先で対処しなければならないケースを考慮しなければ、割り当て/初期化、論理、機能不全、そ

の他のバグを優先的に扱うことでより多くのバグの対処が見込まれると考えられる。

表3.5は、さまざまなバグの種類の期間を1ヶ月未満、1年未満、および1年以上に分割したものです。ほとんどのバグは1ヶ月以内に修正されている。

表 3.4: バグの種類に応じて修正にかかる時間

種類	最小値	最大値	中央値	平均値
アルゴリズム/メソッド	0.027	1545.94	3.195	102.61
割り当て/初期化	0.072	948.41	2.03	65.90
チェック	0.0090	1510.32	1.95	212.18
データ	0.066	1538.09	87.87	426.31
外部インターフェース	0.25	1520.48	3.43	151.96
内部インターフェース	0.18	1649.99	5.23	170.15
論理	0.066	3.77	1.76	1.84
機能不全	0.29	901.50	3.01	105.35
タイミング/最適化	4.19	840.90	35.69	140.07
設定	0.054	2465.47	9.37	222.88
その他	0.045	382.44	1.31	45.28

表 3.5: バグの種類毎の修正時間とその割合

種類	期間	数	割合 (%)	種類	期間	数	割合 (%)
アルゴリズム/メソッド	1 月未満	32	84.21	論理	1 月未満	6	100
	1 年未満	2	5.26		1 年未満	0	0
	1 年以上	4	10.52		1 年以上	0	0
割り当て/初期化	1 月未満	17	85	機能不全	1 月未満	14	77.77
	1 年未満	2	10		1 年未満	2	11.11
	1 年以上	1	5		1 年以上	2	11.11
チェック	1 月未満	14	73.68	タイミング/最適化	1 月未満	4	36.36
	1 年未満	1	5.26		1 年未満	5	45.45
	1 年以上	4	21.05		1 年以上	2	18.18
データ	1 月未満	22	73.33	設定	1 月未満	14	60.86
	1 年未満	5	16.66		1 年未満	7	30.43
	1 年以上	3	0.1		1 年以上	2	8.69
外部インターフェース	1 月未満	4	36.36	その他	1 月未満	8	88.88
	1 年未満	3	27.27		1 年未満	0	0
	1 年以上	4	36.36		1 年以上	1	11.11
内部インターフェース	1 月未満	11	73.33				
	1 年未満	2	13.33				
	1 年以上	2	13.33				

第4章

終わりに

4.1　まとめ

本研究では、ビッグデータシステムのバグレポートの統計的分析の導となることを目的とし、ビッグデータシステムのバグを実証的に研究した。このことにより、バグの重大度について Blocker の占める割合が高いことが判明し、ビッグデータシステムについて、対処すべき優先順位が極めて高い Blocker が多いことなどの固有の特徴が判明したことで今後のバグの同定に役立ち、Hadoop , Spark をはじめとしたビッグデータシステムのバグの分類の先駆けになるだけでなく、今後他のビッグデータシステムに対してもバグを研究する際の導になるであろう。

4.2　今後の課題

筆者のみでの手動による分析等が原因でバグレポートの種類分けに誤りが混在していることが予想されるため、引き続き研究を行う際には誤りの発生を抑えることが1つ目の課題となる。この課題をこなすために、新たな参加者を交えた共同研究とすることで

多角的な方向から取り組むことを挙げる。2 つ目の課題は、バグレポートは今回の対象に対し集計した数が 200 であるが、バグレポートそのものは現時点で集計数の 200 よりはるかに多いため、研究対象の数としては不十分であることから、標本数を増やすことを今後の課題の一つとして挙げる。

謝辞

本研究を進めるにあたってご指導をして下さった九州大学大学院 システム情報科学研究院の趙建軍教授に深く感謝申し上げます。また、週毎に進捗の確認と共に助言をくださったフォン ヤオカイ助教にも感謝を申し上げます。そして、研究の最中、さまざまな協力や助言、励ましをくださった超研究室の皆様にも感謝致します。最後に、私生活の面に加え研究の最中も支えてくださった方々に感謝致します。

参考文献

[1] NTT DATA Global IT Innovator. (http://www.nttdata.com/jp/ja/services/sp/hadoop/)

[2] 城田真琴. ビッグデータの衝撃 巨大なデータが戦略を決める. 2012,pp.31-45.

[3] Welcome to Apache Hadoop!. (http://hadoop.apache.org)

[4] Apache Spark - Lightning-Fast Cluster Computing. (https://spark.apache.org)

[5] 城田真琴. ビッグデータの衝撃 巨大なデータが戦略を決める. 2012,pp.49-52.

[6] Tom White. Hadoop. 玉川竜司 金田聖司 訳. 日経印刷株式会社,2010,pp.1-12.

[7] 猿田浩輔, 土橋晶, 吉田耕陽, 都築正宜.Apache Spark 入門.2015,pp.3-12.

[8] Apache Hadoop Releases. (https://hadoop.apache.org/releases.html)

[9] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, and S. Godfrey, “Defect categorization: making use of a decade of widely varying historical data,” in ESEM, 2008.

[10] Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang, ”An Empirical Study of Bugs in Machine Learning Systems” School of Information Systems Singapore Management University

[11] FSInputStream (Hadoop 1.0.4 API). (https://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/fs/FSInputStream.html)