

## 目次

第1章 序論.....	2
1.1. 本章の構成.....	1
1.2. 研究背景.....	1
1.3. 研究目的.....	2
1.4. 研究方法.....	2
1.5. プロジェクトマネジメントの関連.....	2
1.6. 本論文の構成.....	3
第2章 GitHub について.....	5
2.1. 本章の構成.....	5
2.2. バージョン管理システムについて.....	5
2.2. 分散型バージョン管理システム.....	6
2.3. Git について.....	7
2.3.1. Git の特徴.....	7
2.4. GitHub とは.....	8
2.4.1. GitHub の基本用語.....	8
2.4.2. GitHub の機能.....	10
第3章 Big Data について.....	13
3.1. 本章の構成.....	13
3.2. Big Data とは.....	13
3.2.1. 4V による Big Data の定義.....	14
3.2.2. 3V でのビッグデータの定義.....	15
3.3. ビッグデータ処理のパターン.....	16
3.3.1. バッチ処理.....	16
3.3.2. インタラクティブクエリー処理.....	17
3.3.3. ストリームデータ処理.....	19
3.4. Google BigQuery.....	20
3.4.1 基本的なシステム構成.....	20
3.4.2 BigQuery を使用する.....	21

# 第 1 章

## 序論

## 1.1. 本章の構成

第1章では、本論文の序論を述べる。研究背景、研究目的、研究方法、プロジェクトマネジメントの関連、本論文について記述する。

## 1.2. 研究背景

ソフトウェア開発プロジェクトのための共有ウェブサービスである、**GitHub** のプロジェクトについて調べれば、オープンソフトウェア開発プロジェクトの実態がつかめるはずである。

実際に **GitHub** を調べて分かったことの例として、怒りの表現を含むコミットメッセージの割合、地域によるオープンソースプロジェクトへの貢献者などの分布図があげられる。これらの結果は、**GitHub Data challenge** というイベントで上位に入賞している分析結果である。

**GitHub** のデータ解析は難しい。なぜなら、データが膨大なため、その収集と処理が難しいからである。データの収集が難しいという問題は、一つのプロジェクトにより簡単になった。大量のデータを集めるために、**GitHub** のプロジェクトのタイムラインを記録し、アーカイブ化させ、簡単にアクセスできるようにするためのプロジェクトで **GitHub Archive** である。

データの処理が難しいという問題は、データ量が多すぎるために膨大な量のデータを処理するソフトウェアが少ない点である。**GitHub Archive** と連動させデータ処理ができるソフトウェアに **Google BigQuery** がある。**BigQuery** は、簡単にビッグデータを処理するためのソフトウェアであり、**SQL** に似たクエリを従来のやり方よりも短時間で簡単に実行できる。このソフトウェアの登場により手軽に大量のデータを処理することができるようになった。

これまでの調査で、オープンソフトウェア開発でどのようなプログラミング言語がよく使われているかを調べることに成功したが、プロジェクトが **Fork** される確率の、プログラミング言語による違いが分かれば、オープンソフトウェア開発プロジェクトについての理解が深まると思われる。**Fork** とは、**GitHub** 上で公開されている成果物に独自の変更を加える際に行う複製のことである。

### 1.3. 研究目的

GitHub 上で公開されているオープンソフトウェア開発プロジェクトを Google BigQuery を利用し調査する。オープンソースソフトウェアの開発プロジェクトにおいて、使用するプログラミング言語が異なると、Fork される確率、つまりプロジェクトに貢献する人が現れる確率が異なるということがわかっている。

しかし、この結果は、Fork された回数が多いものについてのみ調査して得られたものであった。

そこで本研究では、Fork された回数が非常に少ないものも対象にして、プログラミング言語による貢献者の出現確率を調査する。

### 1.4. 研究方法

大量のデータを処理することが予想されるので Google BigQuery を利用する。Google BigQuery を使って、GitHub 上のプロジェクトが採用しているプログラミング言語と Fork されている数を収集・統計処理し、Fork される確率のプログラミング言語による違いを明らかにする。

### 1.5. プロジェクトマネジメントの関連

この研究では、Fork される確率のプログラミング言語による貢献者の出現確立を調査する。

ステークホルダーとは、プロジェクトの意思決定、アクティビティ、成果に影響したり、影響されたり、あるいは自ら影響されると感じる個人、グループ、または組織である。ステークホルダーは、プロジェクトに積極的に関与したり、プロジェクトのパフォーマンスや完了によって自らの利害がプラスまたはマイナスの影響を受けたりする。さまざまなステークホルダーには競合する期待があり、プロジェクト内でコンフリクトを生じることがある。ステークホルダーはまた、戦略的なビジネス目標やその他のニーズを満たす成果を得るため、プロジェクト、プロジェクトの成果物、およびプロジェクト・チームに影響を及ぼすことがある。(PMBOK 見てる) [1]

以上のことから貢献者をステークホルダーと捉えることができるので本研究はPMBOKが提唱する、プロジェクト・ステークホルダー・マネジメントに最も関連があると考えられる。

## 1.6. 本論文の構成

第 1 章では序論，第 2 章ではバージョン管理システムについて，Git を中心として記述し，本研究での調査対象である，GitHub について記述する．第 3 章では Big Data についての説明を記述する．第 4 章ではプロジェクトマネジメントとの関係を記述する．第 5 章では具体的な調査方法，調査経過，データを示し，データ解析を行い，第 6 章で考察，まとめを行う．

參考資料

[1] The GitHub Data Challenge 2012-5-1.

<https://github.com/blog/1118-the-github-data-challenge>

[2]The GitHub Data Challenge II 2013-4-3.

<https://github.com/blog/1450-the-github-data-challenge-ii>

[3]Data Challenge II Results 2013-6-26

<https://github.com/blog/1544-data-challenge-ii-results>

## 第 2 章

### GitHub について

## 2.1. 本章の構成

本章では本研究で調査する対象であるバージョン管理システム, GitHub についての基本知識, GitHub で行われている GitHub Data challenge について記述していく.

## 2.2. バージョン管理システムについて

バージョン管理システムは, ファイルの履歴を管理するシステムであり, 修正や追加などの作業によって生成されたファイルについての複数の履歴を記録し, 後から古い履歴の取り出しや, 差分の参照が可能になるシステムである. これらのファイルの更新履歴をリポジトリと呼び, 自分が作業したファイルの更新をリポジトリに反映させることをコミットすると言う. またバージョン管理システムソフトウェアによっては, ファイルの DELETE や移動の履歴を確認する機能や, 特定の利用者がファイルの管理する権限を獲得するロック機能, 複数の変更を統合するマージ機能がある.

開発プロジェクトにおいて複数人で同一のファイルを編集する必要があるとき, バージョン管理システムの機能が役立つのである. バージョン管理システムを利用すると, 更新者や変更点, 変更日時が確認できるため, 誰がいつこの編集を行ったのかすぐに理解することが出来, 混乱や時間の無駄遣いを避けることが出来, とても効率的に作業が進められるのである.

ソフトウェア開発においては, バージョン管理システムは特に有効的である. ソースコードなど長い文を編集した際など変更点を容易に把握でき管理できるので, 障害がいつ発生したのか, どの時点から問題となっていたのか, いつ修正されたのかななどを容易に調べることが出来, 早期の門ファイ解決が可能になる.

また, バージョン管理システムは, 管理方法の違いにより 3 つに分類される. ファイル単位で個別バージョン管理を行う「個別バージョン管理システム」, リポジトリをサーバーで一元管理し, コミットなどの操作はメンバーが行う「集中型バージョン管理システム」, リポジトリをメンバーで管理でき, そのメンバー間でリポジトリの連携が可能である「分別型バージョン管理システム」である.



## 2.2. 分散型バージョン管理システム

3つの種類のバージョン管理システムが存在すると前述したが、ここでは本論文の調査環境である GitHub に用いられるバージョン管理システムの分散型バージョン管理システムについて述べる。

分散型のバージョン管理システムでは、リポジトリをサーバー上ではなく、クライアントであるメンバーのコンピュータ上にも作成し、コミットの参照や差分を所得する場合に手元にあるリポジトリにアクセスするため、常にネットワークに接続している必要がないのである。またローカルファイルにリポジトリが存在するため、高速に動作することが可能であり、変更点を送受信する仕組みがあるため、複数のリポジトリ間で連携することが出来る。この分散型バージョン管理システムを行えるのが Git である。以下に分散型バージョン管理システムを図で表す。

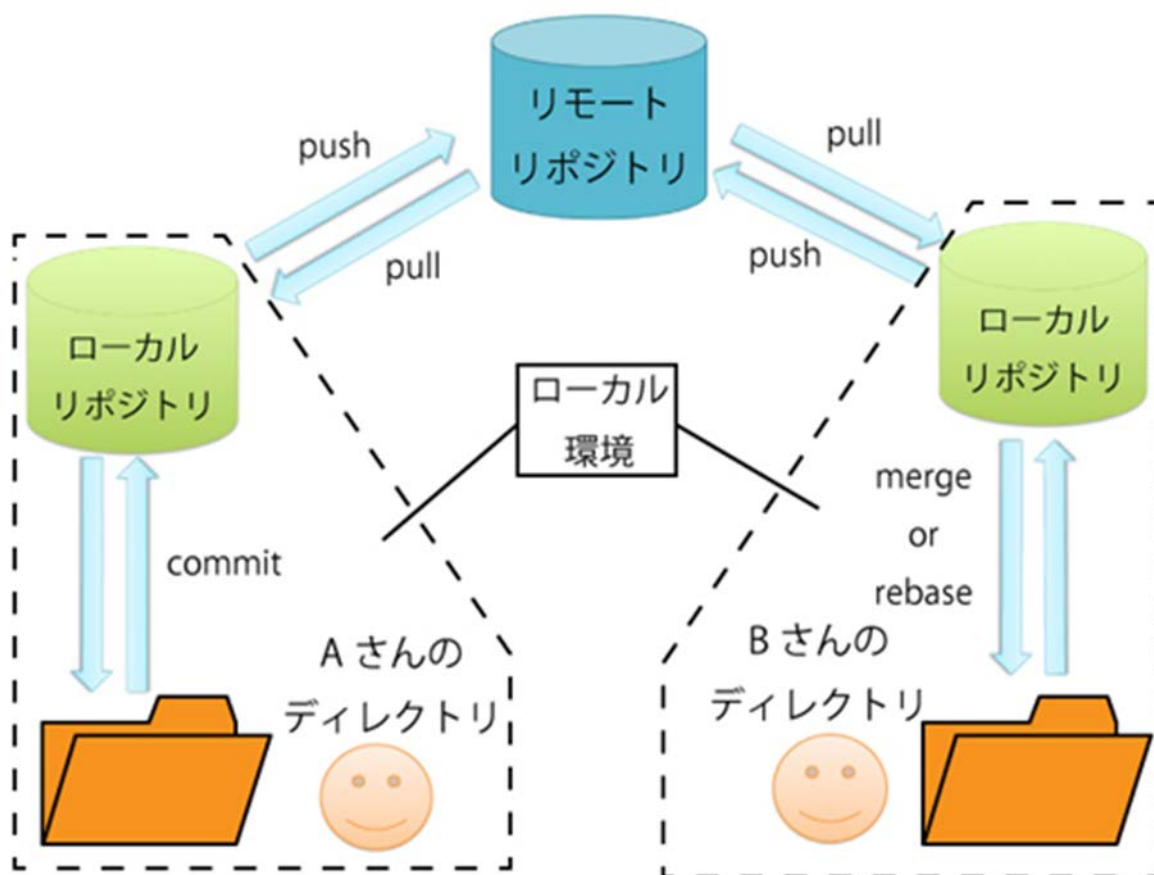


図 3-1 分散型バージョン管理システム

## 2.3. Git について

分散型バージョン管理システムである Git は 2005 年、Linux カーネル開発現場での必要性から開発が始まった。それまで、Linux カーネル開発のソース管理には BitKeeper というバージョン管理システムが用いられていたのである。これは BitMover 社製の商用のバージョン管理システムである。Bitkeeper は、先進の分散型バージョン管理システムで、カーネルプロジェクトが採用した当時、オープンソースの世界にはこれに匹敵する分散型バージョン管理システムの使用が不可欠であった。そのため、Linux は Git を開発したのである。

Git は分散型バージョン管理システムのため、サーバーを必要としない。またユーザーそれぞれのコンピュータ上にリポジトリを持ち、それぞれが互いに連携しあうことができる。さらに、基本的なそれぞれのリポジトリにすべての履歴が保存されるため、差分やログの表示などを高速に行えるのである。

リポジトリ間連携はネットワーク通信やメールを経由して行う。他にもリポジトリを共有リポジトリとして公開する仕組みや、ユーザー管理と組み合わせる方法があり、集中型バージョン管理システムのような利用形態もとれる。

また、他のバージョン管理システムとのデータ交換も可能であり、既存の他のバージョン管理システムのリポジトリを Git リポジトリへ変更することや、中央リポジトリに他のバージョン管理システムのリポジトリを利用し、手元では Git を利用する、といった形態をとることもできるのである。

オープンソースのバージョン管理システムとしては、CVS や Subversion が有名で、今でもこれらの集中型システムはよく使われている。しかし、近年になって、Linux カーネル、X.org, Ruby on Rails, Perl といった有名なプロジェクトが Git に乗りかえて成功裡に使用しているのをみて、Git を使用し始めとする分散型バージョン管理システムを使用するプロジェクトは、飛躍的に増加してきている。

### 2.3.1. Git の特徴

Git は主にファイル自身、ファイルの集合としてのツリー、そしてコミット情報という 3 つの情報を管理している。それぞれの情報はハッシュ値をもとに管理され、このハッシュ値はファイルが同一かどうかの判断にも用いられる。Git はコミット情報を差分管理ではなく、ファイルそのままを保持しているという特徴を持っている。さらに時間的な変遷を管理する仕組みや、コミットをメールで受信する仕組みなどがある。また、ローカルコンピュータ上にリポジトリを持つため、場所や時間、あるいはネットワーク接続の状態に関わらずコミットすることが出来る。

## 2.4. GitHub とは

GitHub とは、GitHub.com により運営されている Git ホスティングサイトであり、Git リポジトリを利用してプロジェクトやソースコードの管理を行うバージョン管理システムを提供しているサービスである。GitHub は Git ホスティングサイトとしては最も多く利用されているサービスで、170 万を超える人が利用している。

また GitHub はソーシャルな機能が特徴で、プログラマー同士がコードの共有を行ったり、コードの公開をし合ったりしている。そして、GitHub ではソースコードだけでなく、画像やドキュメントなど、どんなファイルでもアップロードすることが出来、管理することが出来る。

### 2.4.1. GitHub の基本用語

GitHub の基本用語について以下に記述する。

#### ① Git リポジトリ

GitHub で提供するデータベースのようなものであり、論文に記述してあるリポジトリは Git リポジトリのことである。リポジトリの管理ユーザーが公開するユーザーの範囲を選択することができ、公開の場合は、誰でも閲覧することができる。だが、非公開に選択すると特定のユーザーのみしか閲覧することはできない。

#### ② コミット (commit)

ファイルの変更履歴情報を閲覧したり、ファイルの変更を保存したりすることである。

#### ③ 共有リポジトリ

チームメンバー間で共有できるリポジトリのことであり、ソースコードのメインバージョンが保存されている。

#### ④ ローカルリポジトリ

作業者のコンピュータ上にあるリポジトリである。

#### ⑤ インデックス

ローカルリポジトリへ反映する変更を一時的に保存しておく場所である。インデックスの内容は、commit によりローカルリポジトリへ反映される。

#### ⑥ アドオン (add)

ソフトウェアに追加される拡張機能のことである。

⑦ 作業ツリー

ローカルリポジトリ上にある現在の作業ファイルである。作業ツリーの変更点は `add` によるインデックスに追加される。

⑧ ディレクトリ

フォルダのことであり、ファイルを分類・整理するための保管場所である。

⑨ フォロー

特定のユーザーをフォローすることが出来る。

⑩ フォロワー

ユーザーをフォローしているユーザー。

⑪ スター

Face book, mixi などの SNS で利用されている「いいね!」のようなもの。つけられたスターはカウントされ、スターのカウントが多いほど、他のユーザーから注目されていると認識できる。

⑫ リビジョン

ある期間内までの過去のプロジェクトデータやある程度まとまったプロジェクトデータを記録したものである。

⑬ ウィキ (Wiki)

その場にページが表示され、ドキュメントやコードがかかる場所である。

⑭ Organization

個人的に `GitHut` を使用している人が仕事などで `GitHub` を仕事用に使用したい場合にアカウントをもう 1 つ作成するのではなく、同じアカウントで会社用に使用するアカウントとして使用するものである。

#### 2.4.2. GitHub の機能

GitHub は Git をサポートするために様々な機能を備えている。以下には機能について記述する。

- フォーク (Fork)

GitHub 上で公開されているリポジトリを自分のリポジトリとして複製する動作。既存のリポジトリに対して独自の変更を加えたい場合に利用する。

- ライト (write)

フォークしたときにコピーしたオリジナルのリポジトリのデータに書き込みをすること。

- プルリクエスト (Pull Request)

フォークしたリポジトリに対して変更を加へ、それをフォーク元のリポジトリに取り込んで貰う様にリクエストを送ること。

- マージ (merge)

プルリクエストした人がその人のリポジトリに対して行った変更を自分のリポジトリにも取り入れること。

- イシュー (Issue)

1つのタスクを1つの Issue に割り当てて、データの監視や管理を行えるようにするための機能。1つの機能変更や修正などに対して1つの Issue が割り当てられるため、Issue を見れば、そのタスクの変更や修正に関することがすべてわかるよう管理できるのである。また、Issue にタグやマイルストーンをつけることも可能である。タグ機能は初期の設定の場合では、「バグ」、「重複」、「強化」、「無効」、「質問」が設定されている。タグの種類は増やすことも可能である。また、Issue には、「Open」と「Close」機能があり、Issue を受信した人は受信した Issue を拝見したら Open をクリックし、拝見し終わったら、Close をクリックすることで、Issue を発行したユーザーに拝見し終わったことを通知することができる。

- ウォッチ機能

他の人のデータを見ることが出来る。他の人の進捗状況やプロジェクトの内容を閲覧できる。

- グループ機能

特定のアカウントユーザーで構成し、特定のユーザー同士でリポジトリを共有したりすることができる。グループ機能を公開の状態にすることで、グループ以外のユーザーも閲覧できるようになる。非公開の状態にするとグループで決められたユーザー以外は閲覧やリポジトリを操作することができないように設定することができる。

- 検索機能

検索機能は「検索ウィンドウ」に検索ワード（ユーザー名やプロジェクト名，コードなど）を入力するとそれに関連した情報を表示することができる．

#### 参考文献

- [1] 濱野純. 入門 Git, 秀和システム, 2009-9-25.
- [2] 片岡巖. WEB+DB PRESS Vol.69, 技術評論社, 2012-7-25.
- [3  
] GitHub. GitHub Developer, <http://developer.GitHub.com/v3/issues/>, 2013-10-10.

## 第 3 章

### Big Data について



### 3.1. 本章の構成

本章では本研究で GitHub にあげられているデータを収集するにあたり，利用する Big Data 処理技術についての基本知識,本研究で使用する Google BigQuery について記述していく．

### 3.2. Big Data とは

パソコン，スマートフォンが普及し「ビッグデータ」という言葉が流行することからもわかるように，私たちは膨大な情報を生み出しながら生活している．Google や Yahoo! に寄せられる大量の検索クエリや， Twitter，Facebook などの SNS に投稿される文章や画像，動画，スマートフォンを利用するサービス等で収集される位置情報データ，防犯カメラで記録される人間の表情や動きのデータなどの膨大な量のデータを指す．

ビッグデータとは一般的にペタ（1,000 兆）バイト級（表 1-1）のデータ量と言われている．このような数値的定義もあるが，ペタバイト以下であればビッグデータでは無いという訳でもなく，本質的には，「従来の手段では管理しきれない規模のデータ」を指す．

$10^n$	接頭辞	記号	漢数字表記 (命数法)	十進数表記	分類
$10^{21}$	ゼタ (zetta)	Z	十垓	1 000 000 000 000 000 000 000	ビッグデータ
$10^{18}$	エクサ (exa)	E	百京	1 000 000 000 000 000 000	ビッグデータ
$10^{15}$	ペタ (peta)	P	千兆	1 000 000 000 000 000	ビッグデータ
$10^{12}$	テラ (tera)	T	一兆	1 000 000 000 000	
$10^9$	ギガ (giga)	G	十億	1 000 000 000	
$10^6$	メガ (mega)	M	百万	1 000 000	
$10^3$	キロ (kilo)	K	千	1 000	

表 1-1 単位接頭辞と十進表記

### 3.2.1. 4V による Big Data の定義

IBM 社による Big Data の定義で 4V というものがある。4V とは、容量 (Volume)、種類 (Variety)、頻度・スピード (Velocity)、正確さ (Veracity) から構成されている。

#### ◆容量 (Volume)

ビッグデータの特徴である容量の巨大さを指す。企業内外にはデータが溢れており、数テラバイトから数ペタバイトにもおよぶ。またデータが増大することによる計算量も非常に膨大となる。

#### ◆種類 (Variety)

ビッグデータは企業システムで通常扱っているような顧客情報や販売データ、経理データ、在庫データなどの構造化データであるとは限らない。テキスト、音声、ビデオ、ページ遷移、ログファイルなどのさまざまな種類の非構造化データも存在する。

#### ◆頻度・スピード (Velocity)

今この瞬間にも、ものすごい頻度で RFID などの IC タグやセンサーなどからデータが生成されている。昨今の変化の著しい市場環境では、これらのデータによりリアルタイムに対応したものを求められている。

#### ◆正確さ (Veracity)

データの矛盾、曖昧さによる不確実性、近似値を積み重ねた不正確さなどを排除して、本当に信頼できるデータが意思決定には重要である。

以上が IBM による 4V の定義であるが、容量 (Volume) については最初に記述したように、必ずしもペタバイト以上でなければならないとは考えていない。

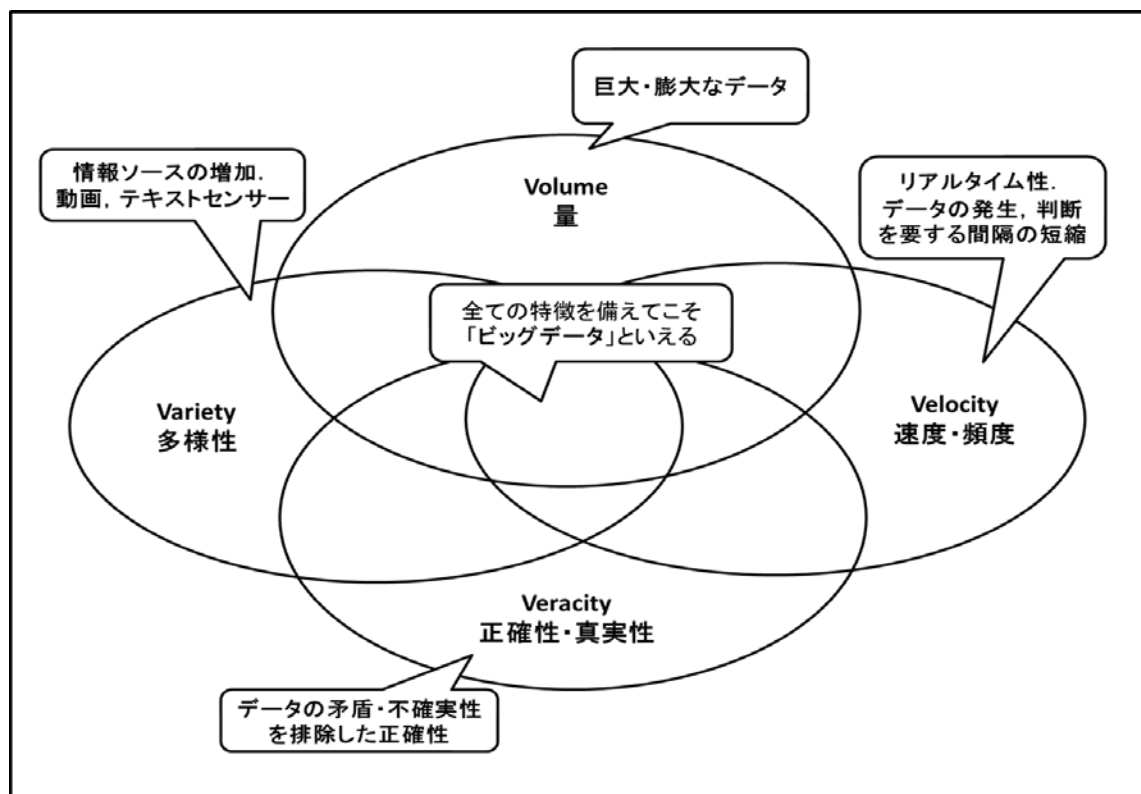


図 1-1 ビッグデータの 4V

### 3.2.2. 3V でのビッグデータの定義

3V で表す場合は、容量 (Volume)、種類 (Variety)、頻度・スピード (Velocity) の 3 つになり、正確さ (Veracity) は含まれない。確かにビッグデータには正確でないデータが混在することもあり、例えば Twitter などの SNS データには、冗談やデマ情報の書き込みなども混じっており、センサーなどでも故障によるノイズが混じることもあります。データ量が少ない場合には、外れ値として手作業で除去することも可能だがいわゆるビッグデータと言われている大量のデータの場合は、手作業によるデマ情報やノイズなどの除去はほとんど不可能である。

しかし、ビッグデータで収集するデータは殆どが生データであるという特徴がある。正確さをどう定義するかにもよるが、センサーからの入力データなどは生データそのものだが、SNS からの入力データなども生データであり、その意味では正確なデータといえる。つまり、編集などで手が加わっていないデータであり、またそこで発せられるメッセージはその人が、その人の環境により制約などを感じるデータでは無いからだ。これは、勤務する企業・組織内で作成する報告書など対比して考えるとわかりやすいだろう。

### 3.3 ビッグデータ処理のパターン

下記の表に、3つの処理パターンの特性を簡潔にまとめたものを記述する。

	バッチ処理	インタラクティブクエリー処理	ストリームデータ処理
実行タイミング	ユーザー指定と定期的実行	ユーザー指定と定期的実行	常時連続実行
処理単位	蓄積データをバッチで一括処理	蓄積データをバッチで一括処理	少数のフローデータ処理
実行時間	分～時間	秒～分	ミリ秒～秒
処理モデル	MapReduce	クエリ・OLTP	ストリーム処理

表 3-1 ビッグデータ 3つの処理パターン

表 3-1 のようにビッグデータの処理パターンには、「バッチ処理」、「インタラクティブクエリー処理」、「ストリームデータ処理」の3種類のパターンがある。

バッチ処理では蓄積データをバッチで一括処理だが、これは Google 検索用に開発された MapReduce 処理を利用した Hadoop が代表的である。しかし、Hadoop はビッグデータ処理用として開発されたものではないので、処理結果作成に時間がかかるという欠点である。

インタラクティブクエリー処理は、蓄積された大容量データをオンラインクエリなど使用して一括解析処理するものである。インタラクティブクエリー処理では蓄積されたビッグデータを数秒から数分で実行する。本研究で使用する BigQuery の処理エンジン Dremel はその処理スピードの速さに特徴がある。

ストリームデータ処理は大量発生する実世界データを逐次に時系列処理する技術である。データ発生時にあらかじめ登録したシナリオに従って集計・分析に必要なデータを抽出し、データ処理を行う。このように逐次時系列でデータ処理できることから、最新の情報、その中での特異な値の発生などに対してリアルタイムに対応するシステムを構築できることが特徴で IoT への応用に最適な処理方法といえる。

#### 3.3.1 バッチ処理

最初に MapReduce で代表される、バッチ処理の特性を見る。

数十年前のメインフレームは、主記憶が数百キロバイト、価格は数千万円程度だった。これを現在の PC（主記憶数ギガバイト、価格は 10 万円程度）と比べた場合、価格性能比では約 100 倍にもなる。また、CPU 処理スピードと、ネットワークの帯域幅についてはそれぞれ、主記憶と類似の性能向上を遂げてきている。

このようなことは、コンピューター関係以外の業種では全く例を見ない群を抜く性能向上であ

り、この急激なプラットフォームの真価がクラウドコンピューティングやそのうえで実行されるビッグデータ処理などを可能にしている。

ただしこれはクラウドなどに限ったことではなく、ITの世界ではこれまでも短いタイムスパンで新しいテクノロジー・ブレイクスルーやビジネスモデルが出現してきており、これはプラットフォームやネットワークの進歩に依存している部分が多くある。言葉を変えれば、これらの新しい発想はその時点でのプラットフォーム性能で初めて成り立つものであり、これをわずかでも前の世代に思いついたとしても、実現不可能である場合が多い。

このようにクラウドなどの先端 IT システムは、現在のそしてこれからも進化を続けるはずのプラットフォームやネットワークに依存したものである。

### 3.3.2. インタラクティブクエリー処理

インタラクティブ処理は、本研究で使う処理方法である。BigQuery による説明を記述す。

Google がリリースするソフトウェアツールには、もともと Google が社内使用の目的で開発していたものも少なくなく、BigQuery もそれに当てはまる。Google も当初は社内使用でも MapReduce を使用してビッグデータ処理を行っていたが、バッチ処理による結果生成の遅延や処理を行うための準備の煩雑さなどから、それに代わるツールとして開発されたのが BigQuery である。

BigQuery はデータの入力または JSON フォーマットのファイルから直接行うことができ、Cloud Storage からのデータロードも可能である。ビッグデータの解析や絞込みは RDB (Relational Database) の SQL に類似したクエリ言語を使用し、UI 画面や PC のコマンドラインから容易にデータ検索を行うことができ、また Excel を使用し検索・表示を行うことが出来る。

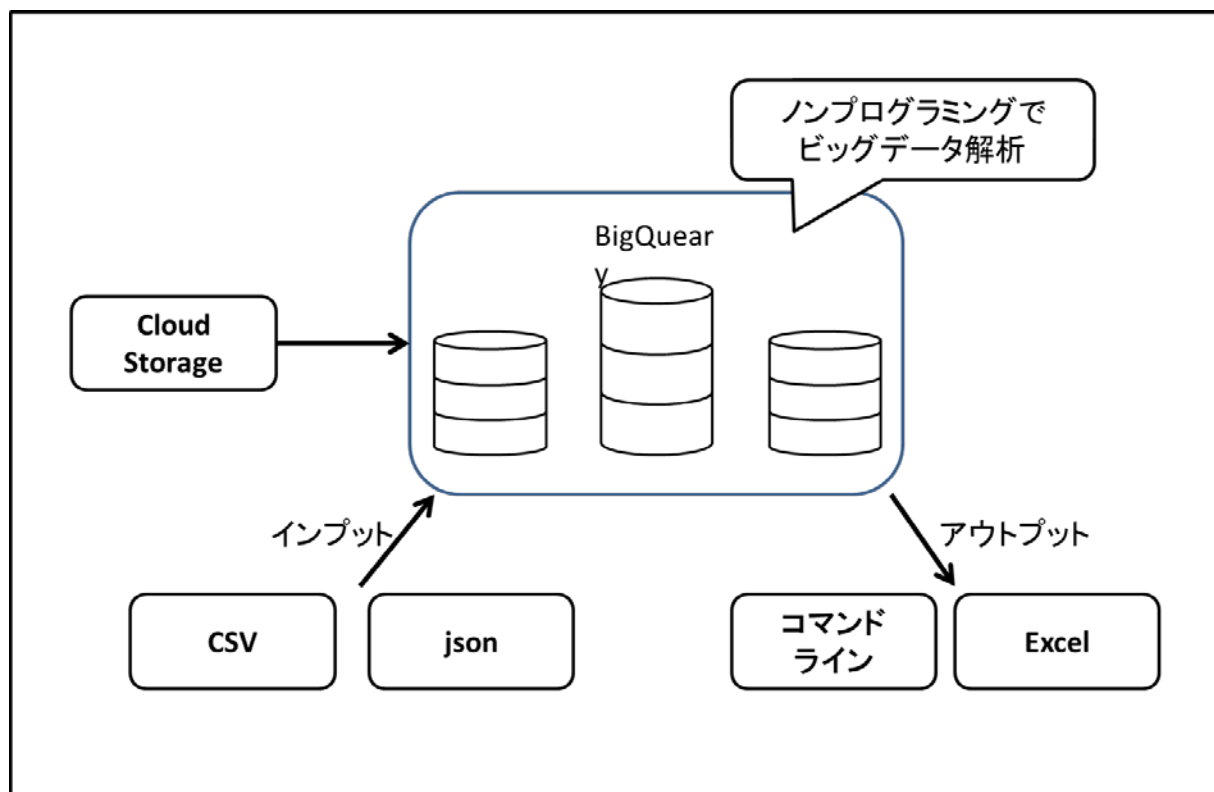


図 3-1 BigQuery の入出力

BigQuery を使用したインタラクティブクエリ処理では、マウス操作と簡単なキー入力によってすべての操作を行うことができ、また結果出力も数秒から数分以内で得ることができる。したがって、何かこのデータを解析したいと思った時、その場で気軽に行えるという点が一番の特徴として挙げられる。

また、BigQuery でのビッグデータ解析は大量のデータを超高速で行えるのも大きな特徴で、例として 15 億行のデータに対する比較的複雑な集計問い合わせが 20 秒から 25 秒で返ってきたというユーザの実行結果もある。BigQuery ではインデスクを作成する必要がなくデータをロードするだけでこのような高速クエリが実行でき、またキャッシュはトグルボタンから有効無効を切り替えられるが、キャッシュを使っていなくてもいなくても同様の結果が得られる。

### 3.3.3 ストリームデータ処理

ストリームデータ処理は、大量発生する時系列のデータ（ストリームデータ）をリアルタイムに逐次処理する技術。

ストリームデータ処理は、データ発生時に、あらかじめ登録したシナリオにしたがって集計・分析に必要なデータを抽出し、データ処理を行う。その際、分析対象データをメモリー上で処理する「インメモリデータ処理技術」により、高速なデータ処理を実現している。これらの技術によって、大量データを高速に、かつリアルタイムに処理でき、たとえば、株価のテクニカル指標やランキング情報から売買をリアルタイムに自動判定する、といったシステムに大変有効だ。ほかにも、リアルタイムの在庫管理や、不正操作の監視を行うシステムなど、多くの利用目的が考えられる。

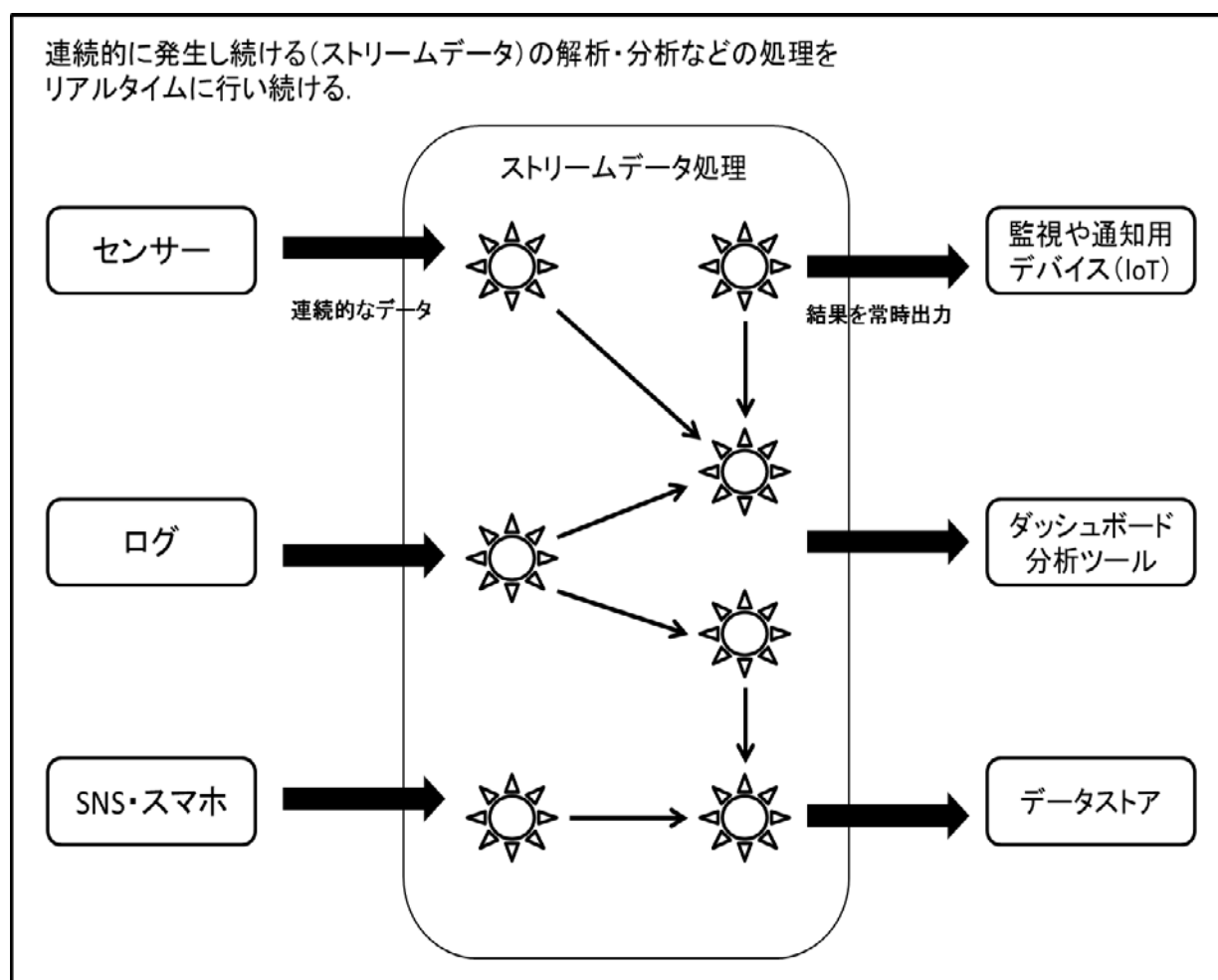


図 1-20 ストリームデータ処理

### 3.4. Google BigQuery

Google BigQuery の基本構成と特徴について記述する。

#### 3.4.1 基本的なシステム構成

BigQuery を使用する場合の基本的なシステム構成はシンプルである。Web UI からの操作以外はすべて Excel と CSV ファイルを入出力に使用する図を以下に記述する。

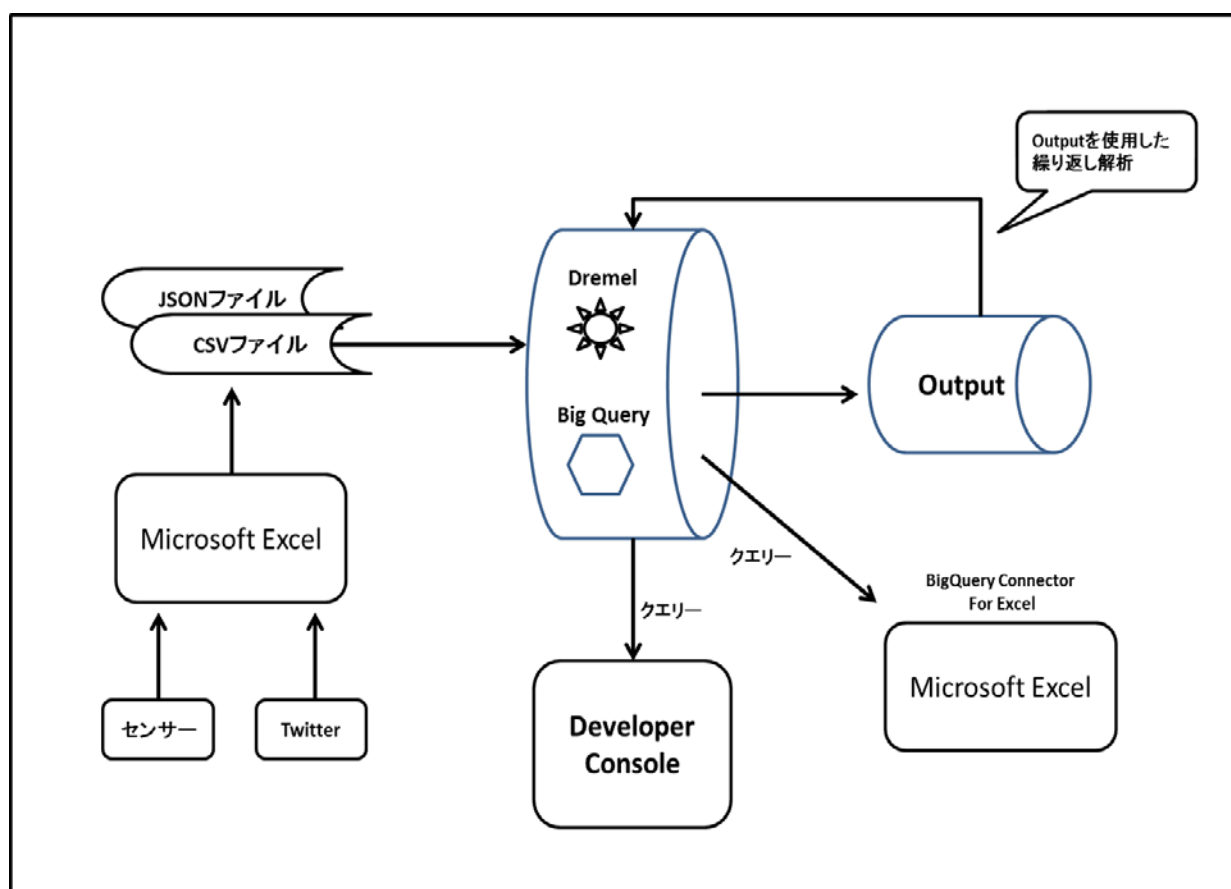


図 3-1 BigQuery 処理パターン



### 3.4.2 BigQuery を使用する

BigQuery にサインアップする。  
最初にサインアップを行う。

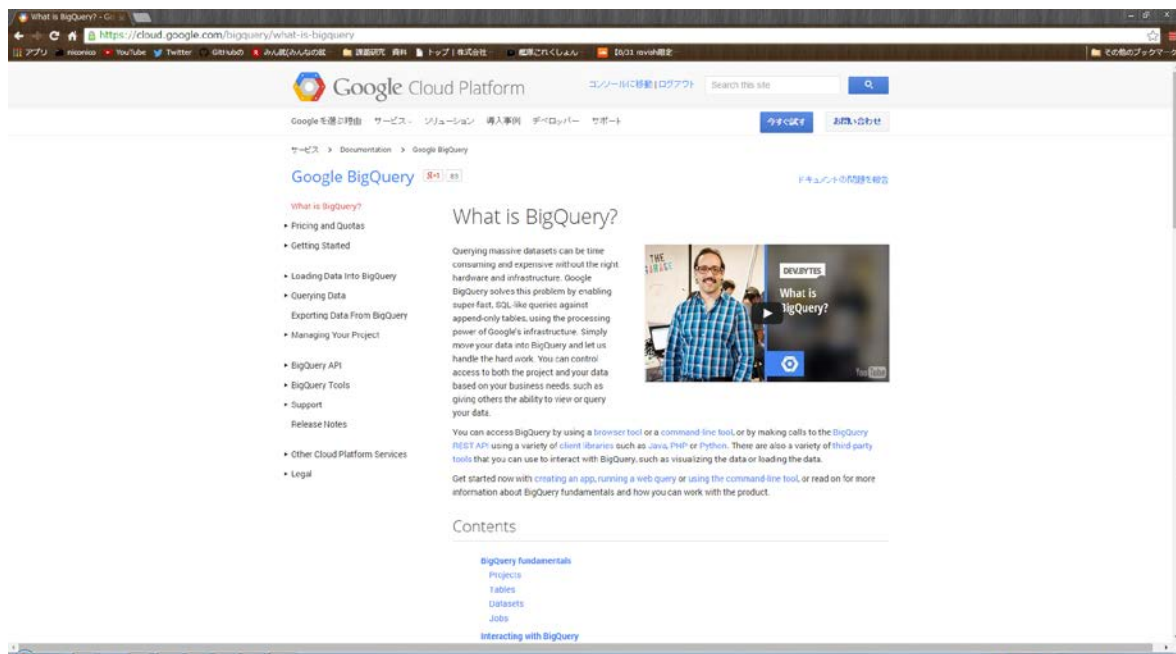


図 3-2 Google BigQuery 公式画面 (https://cloud.google.com/bigquery/what-is-bigquery)

サインアップでは、図 3-2 の BigQuery 公式画面を表示する。画面上部の「今すぐ試す」からサインアップを行う。「今すぐ試す」を押すと下記のような画面が表示される



図 3-3 サインアップの画面

サインアップすると Google Developers Console の「Projects」画面（図 3-4）が表示される。

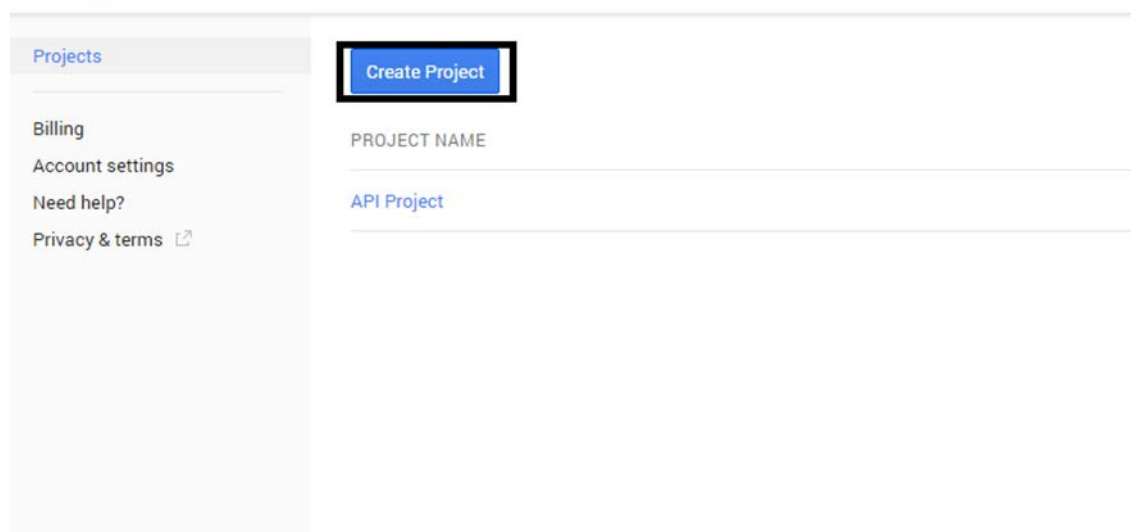
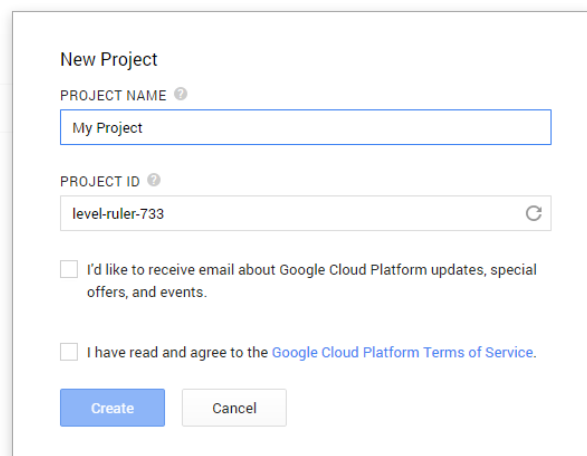


図 3-4 Projects 画面

私の場合,すでにプロジェクトが登録されていますが,初めての場合は「PROJECT NAME」以下に何も表示されていないはずである.画面上部の「Create Project」をクリックして表示される「New Project」サブ画面から,プロジェクト名 (Project name) とプロジェクト ID (Project ID) を入力して,新規のプロジェクトを作成する.プロジェクト名とプロジェクト ID は英字または英数字でなんでも構わないが,プロジェクト名はプロジェクト内容に関連した名前にし,プロジェクト ID はその短縮形にするのが良い.



New Project

PROJECT NAME ?

My Project

PROJECT ID ?

level-ruler-733

☐ I'd like to receive email about Google Cloud Platform updates, special offers, and events.

☐ I have read and agree to the [Google Cloud Platform Terms of Service](#).

Create Cancel

図 3-5 プロジェクト作成サブ画面

次に、作成された「PROJECT NAME」のリンクをクリックして表示される画面の左ペインで APIs&auth 下の APIs をクリックすると図 3-6 の画面表示になるので、BigQuery API の status を画面右端のボタンをクリックして「ON」にする。ただし、この設定は私の研究の場合なのでノン・プログラミングで BigQuery を使用することに徹するのであれば、必ずしもやる必要はない。



図 3-6 プロジェクト画面

GitHub Archive と連動させるために左ペインの Big Data から BigQuery を選択する。

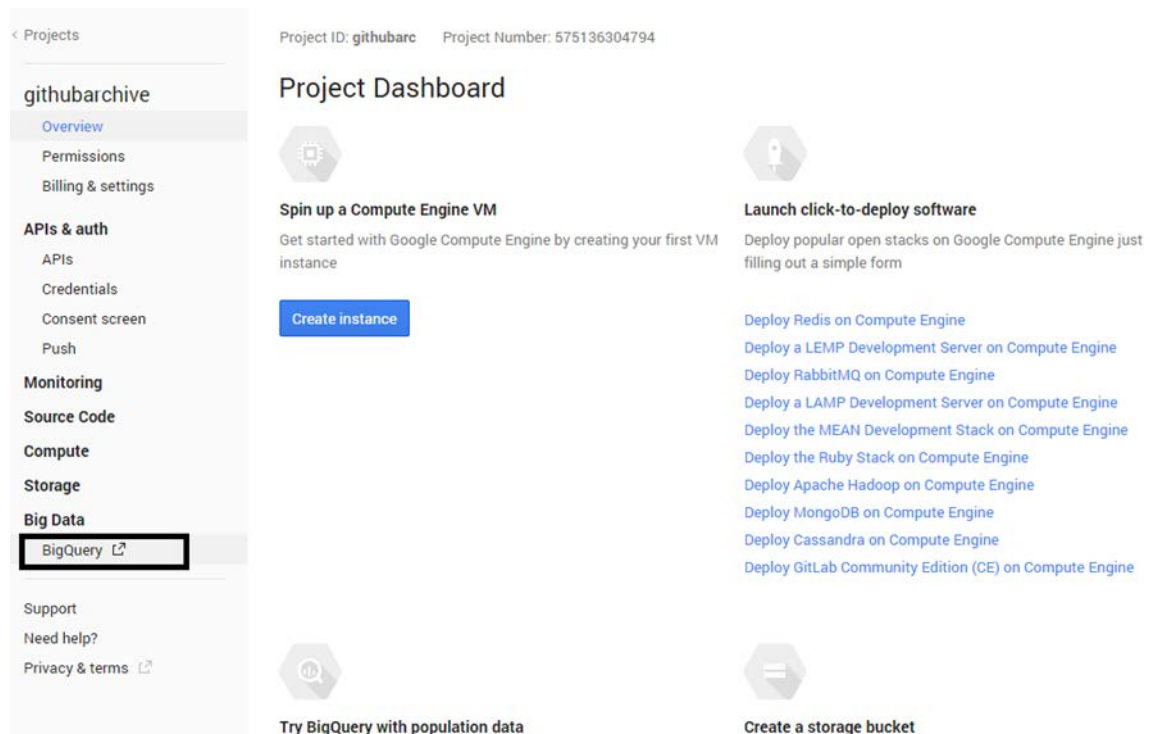


図 3-7 BigQuery 選択

図 3-8 の画面から BigQuery の操作を行っていく。BigQuery では、データの入れ物として「Dataset」を作成し、その中に「Table」を作成する。Table は 1 つの Dataset に複数作成することができる。

図 3-8 の左ペインでプロジェクト名の右にある▽のマークをクリックすると、プルダウンメニューが表示される。プルダウンメニューで「Switch to project」を選択する。

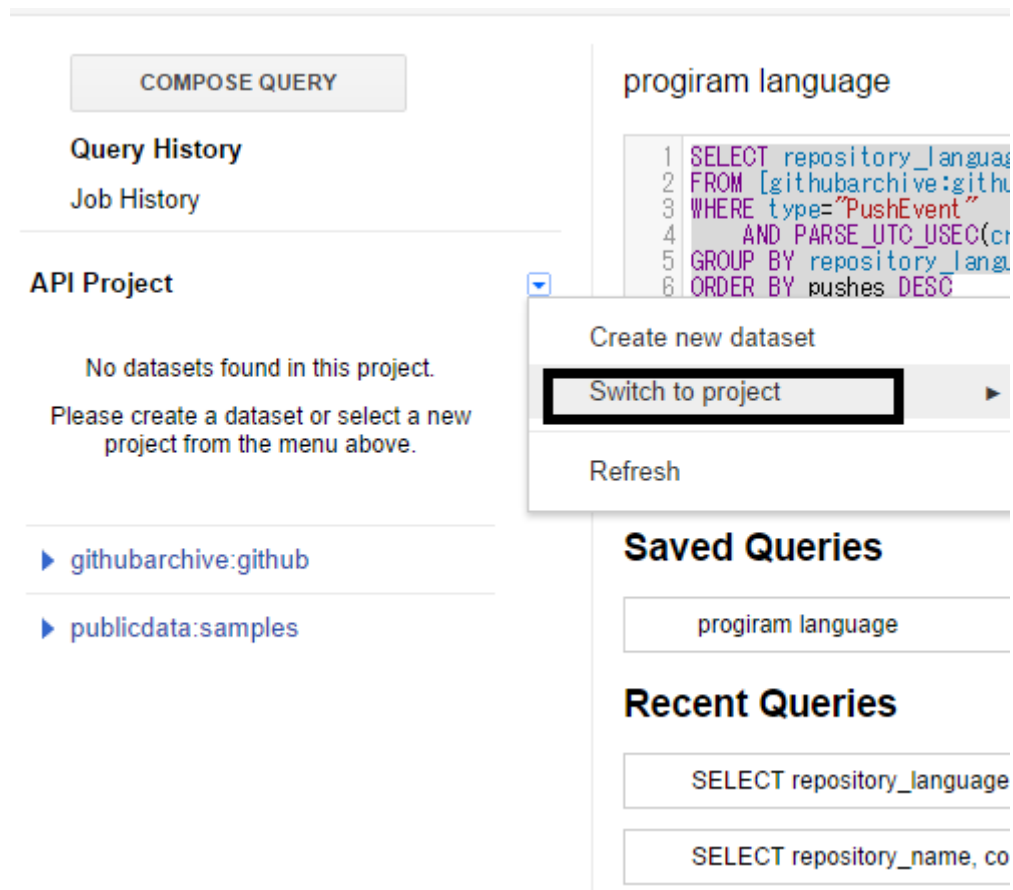


図 3-8 BigQuery 画面から dataset 作成

Switch to project 選択後図 3-9 のような選択肢が表示されるので「Display project」を選択する。

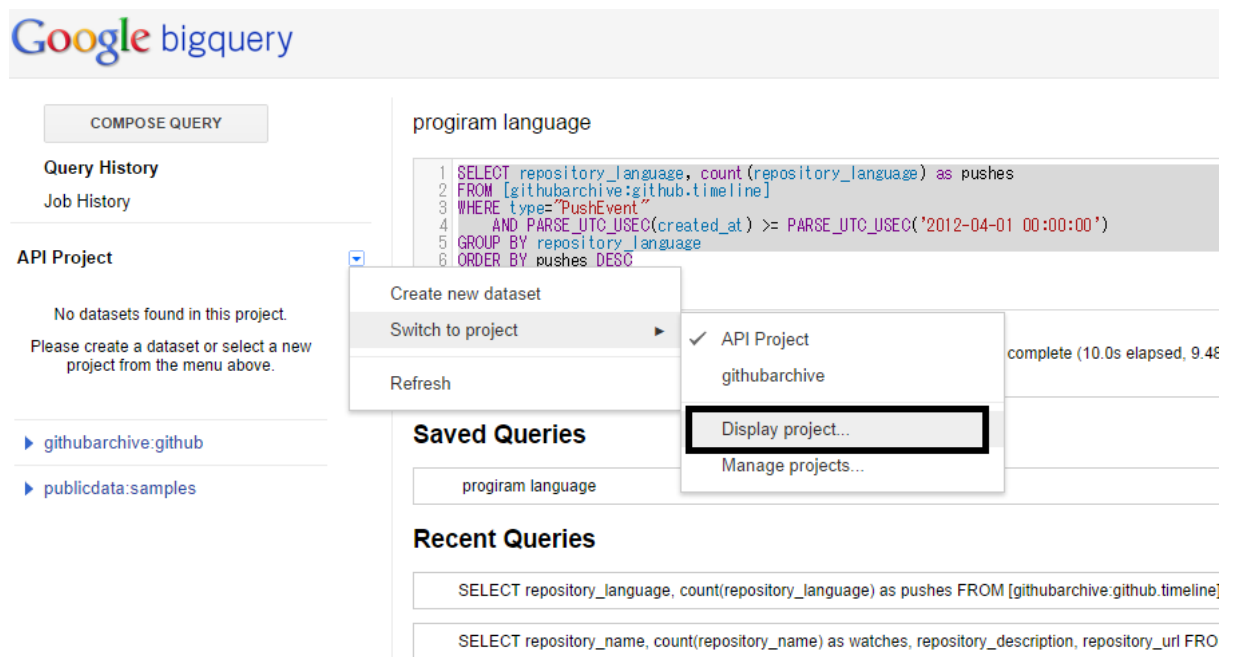


図 3-9 Display project

図 3-10 のようなダイアログが表示されたら Project ID（ここでは githubarchive）を入力する。入力後「OK」ボタンをクリックする。

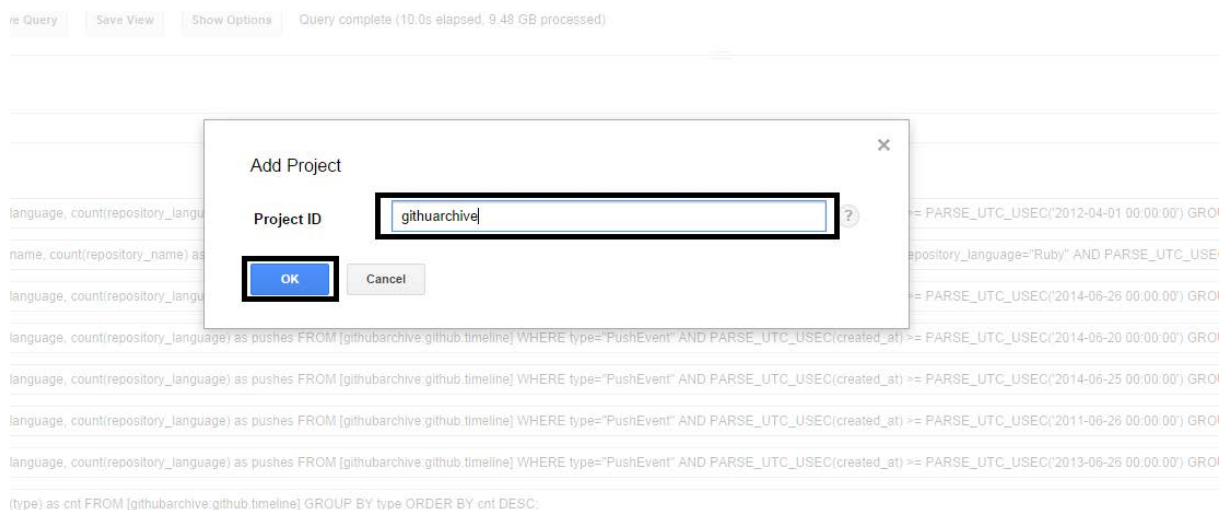


図 3-10 Project ID 指定

完了後、左ペイン上部にある「COMPOSE QUERY」を選択する。

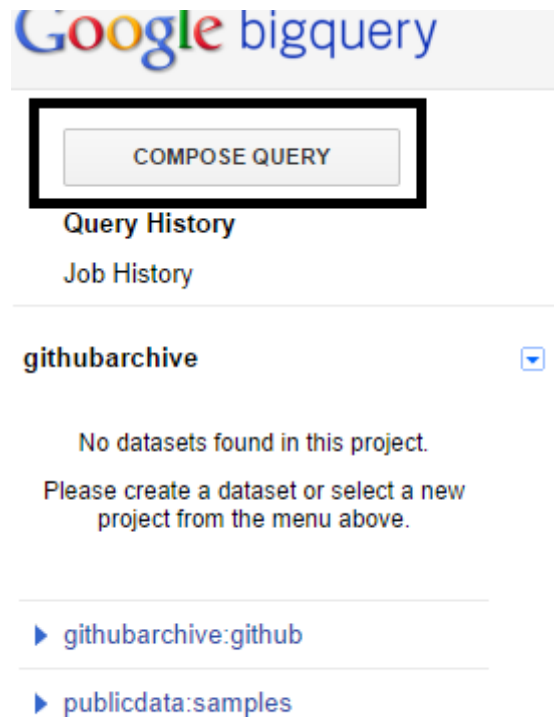


図 3-11 COMPOSE QUERY の選択

選択後図 3-12 のような画面になるので黒枠の部分にコードを入力する。



図 3-12 コードの入力

コードの例として、GitHub で 2012-04-01 から使われているプログラミング言語を解析するコードを以下に記述する。

```
SELECT repository_language, count(repository_language) as pushes
FROM [githubarchive:github.timeline]
WHERE type="PushEvent"
      AND PARSE_UTC_USEC(created_at) >= PARSE_UTC_USEC('2012-04-01 00:00:00')
GROUP BY repository_language
ORDER BY pushes DESC
```

コード入力後、New Query の下にある「RUN QUERY」をクリックする。



図 3-13 コードの実行



### New Query

```

1 SELECT repository_language, count(repository_language) as pushes
2 FROM [githubarchive:github.timeline]
3 WHERE type= 'PushEvent'
4 AND PARSE_UTC_USEC(created_at) >= PARSE_UTC_USEC('2012-04-01 00:00:00')
5 GROUP BY repository_language
6 ORDER BY pushes DESC

```

**RUN QUERY** Save Query Save View Show Options Query complete (5.3s elapsed, cached)

### Query Results

9:35pm, 14 Oct 2014

Row	repository_language	pushes
1	JavaScript	21864210
2	Java	14148511
3	Python	11045096
4	Ruby	10506238
5	PHP	9769443
6	C++	6574131
7	C	5915965
8	CSS	5770702
9	Shell	3653936
10	C#	2774953
11	Objective-C	1978772
12	Perl	1514504
13	VimL	1234261
14	Scala	798215
15	CoffeeScript	774351
16	Go	772313
17	R	612397

参考資料

Google BigQuery ではじめる自前ビッグデータ処理入門 2014-10-10

IMB ビッグデータ <http://www-03.ibm.com/software/products/ja/category/bigdata>

オラクルデータベースインサイダー 2012-10-26

[https://blogs.oracle.com/dbjp/entry/bigdata\\_000244](https://blogs.oracle.com/dbjp/entry/bigdata_000244)

IT ジャーナリスト星暁雄の"情報論"ノート 2012-06-01

<http://hoshi.air-nifty.com/diary/2012/06/githubgoogle-bi.html>

WORKSIGHT 2013-07-22 <http://www.worksight.jp/issues/289.html>

ニッセイ基礎研究所 ビッグデータで何がわかるか 2013-11-08

<http://www.nli-research.co.jp/report/report/2013/11/repo1311-c3.html>

## 第 4 章

### 開発・調査

#### 4.1. 本章の構成

本章では調査対象の記述，GitHub からデータを収集するためのプロジェクトである GitHub Archive の説明，Google BigQuery で使うコードの設計の解説を記述する．

#### 4.2. 調査対象