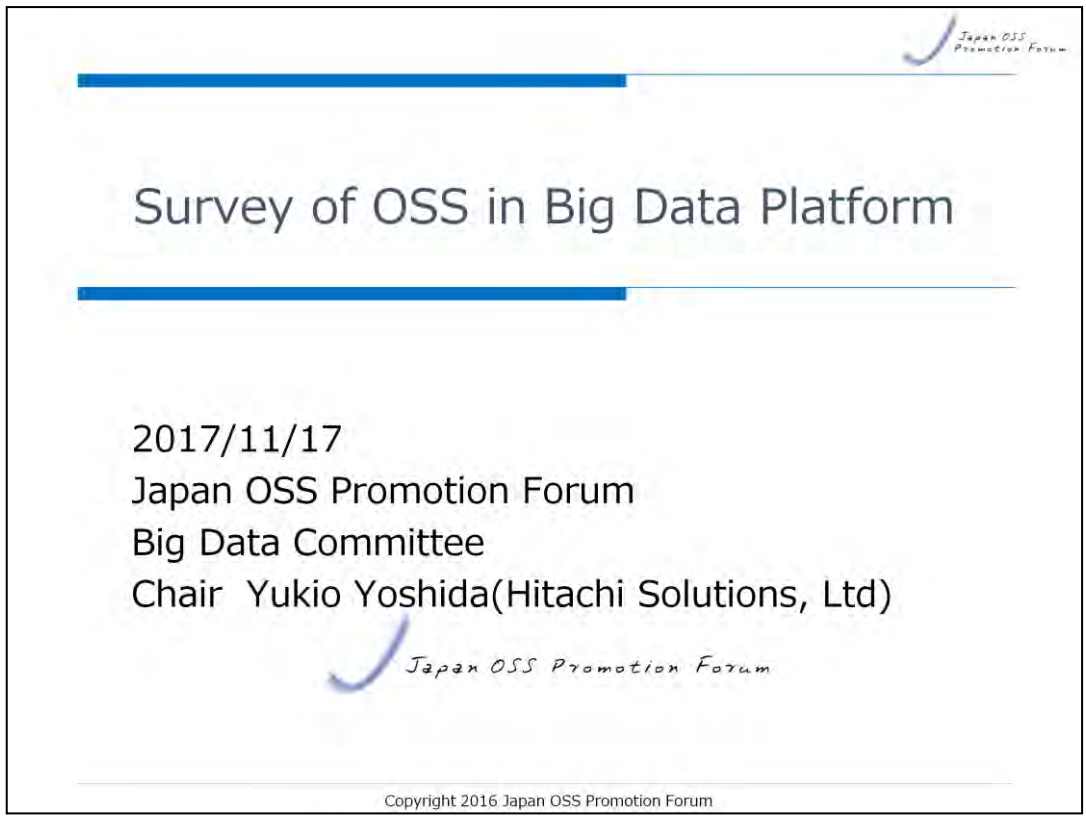
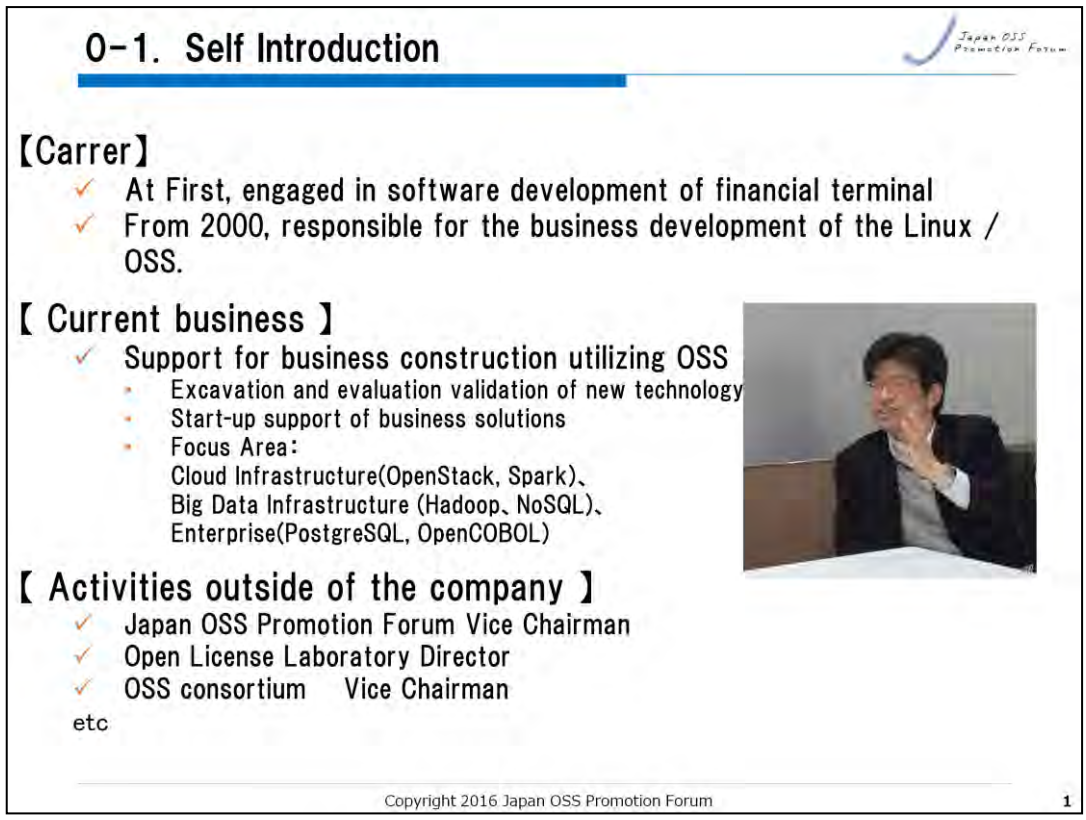




経歴

・入社当時は、OSSに関係なく、金融端末のソフトウェア開発をしていました。
・2000年頃からLinuxやOSSのビジネス企画を担当し、現在に至っています。

現在の業務

・現在は、OSSを活用したビジネスの構築の支援をしています。
特に、新しい技術やOSSの発掘や技術検証をしています。
また、ビジネスソリューションの立ち上げの支援もしています。
現在、フォーカスしている領域は、
・OpenStackに代表されるクラウドを構成する基盤のOSS
・Hadoopに代表されるビッグデータを構成する基盤のOSS
・企業システムで活用されるデータベースやCOBOLの環境です。

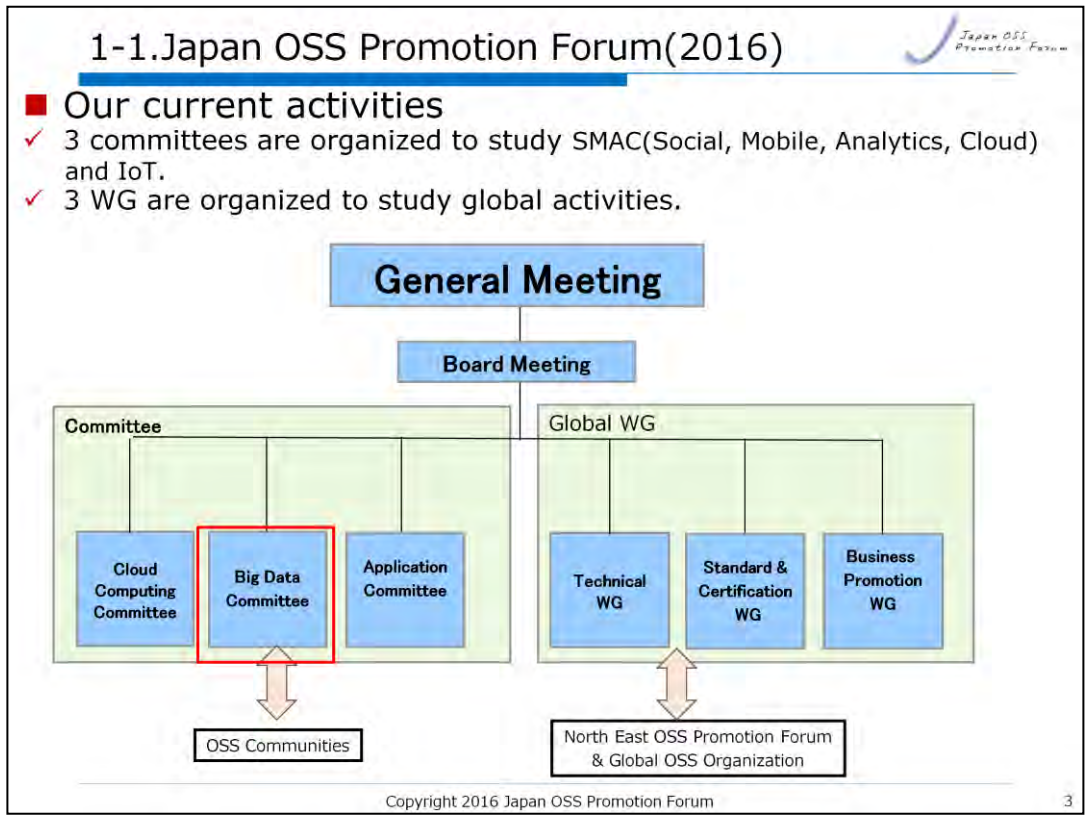
社外での活動

社内だけではなく、社外でもさまざまな活動をしています。

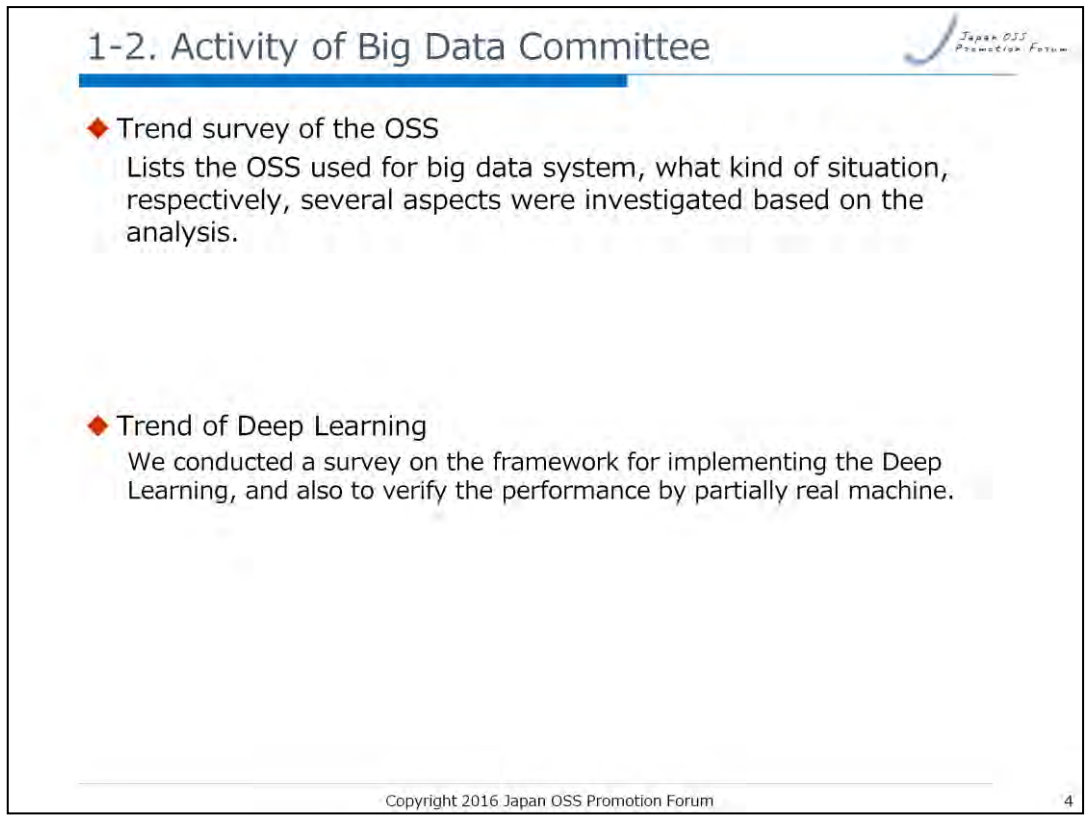
Contents

Introduction JOPF BigData Committee
Result of Data Analysis
Trend of Deep Lerning

Copyright 2016 Japan OSS Promotion Forum



日本OSS推進フォーラムは、3つの部会とWGで構成されています。
部会は、クラウド、ビッグデータ、アプリケーションです。
特に注目しているのは、Social、モバイル、分析、クラウドとIoTです。
また、WGは、北東アジアOSS推進フォーラムと連携しています。



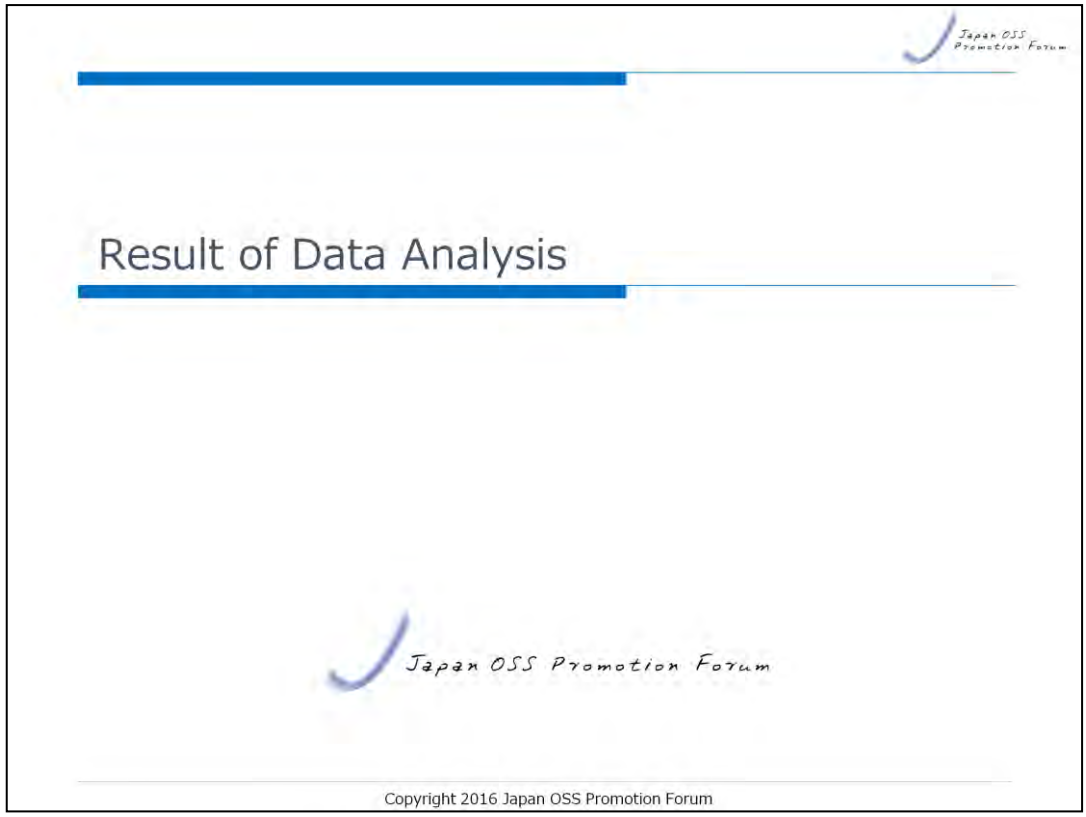
ビッグデータ部会では、下記の活動を行っています。

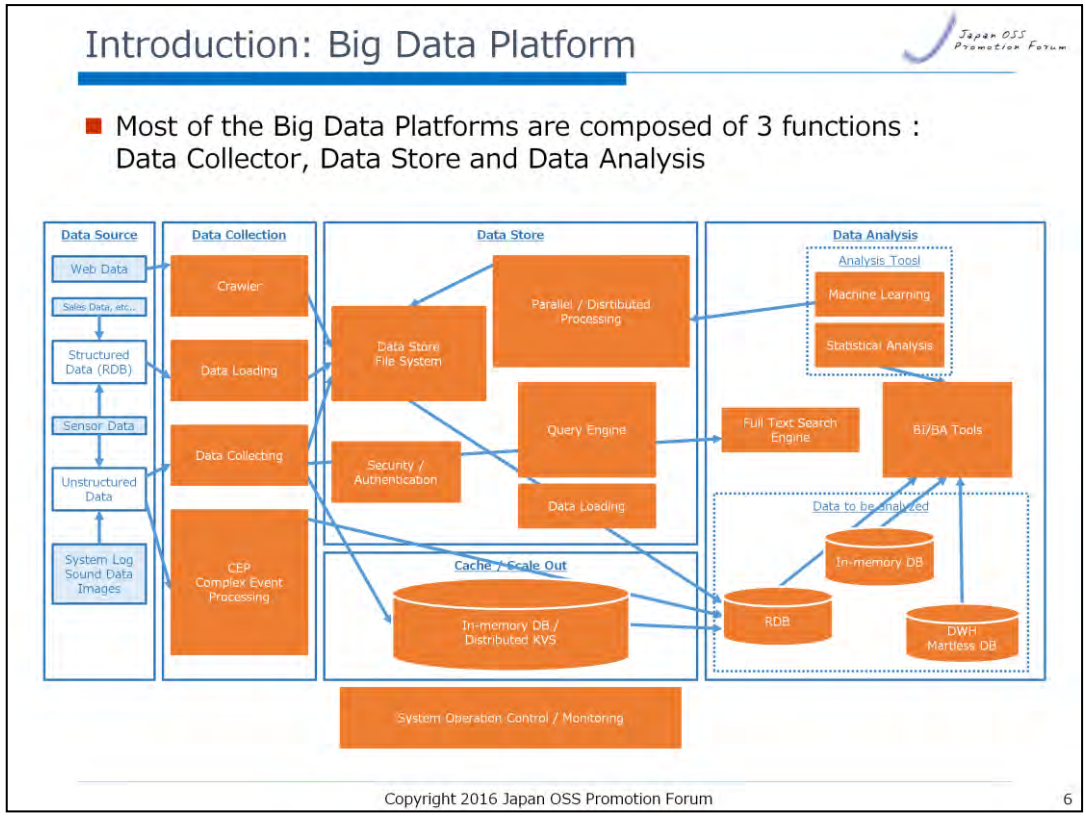
ビッグデータ関連OSS動向調査

ビッグデータシステムに使われるOSSを洗い出し、それぞれがどのような状況であるか、いくつかの観点をもとに調査し、「今使えるビッグデータOSSは何か？」を分析しました。

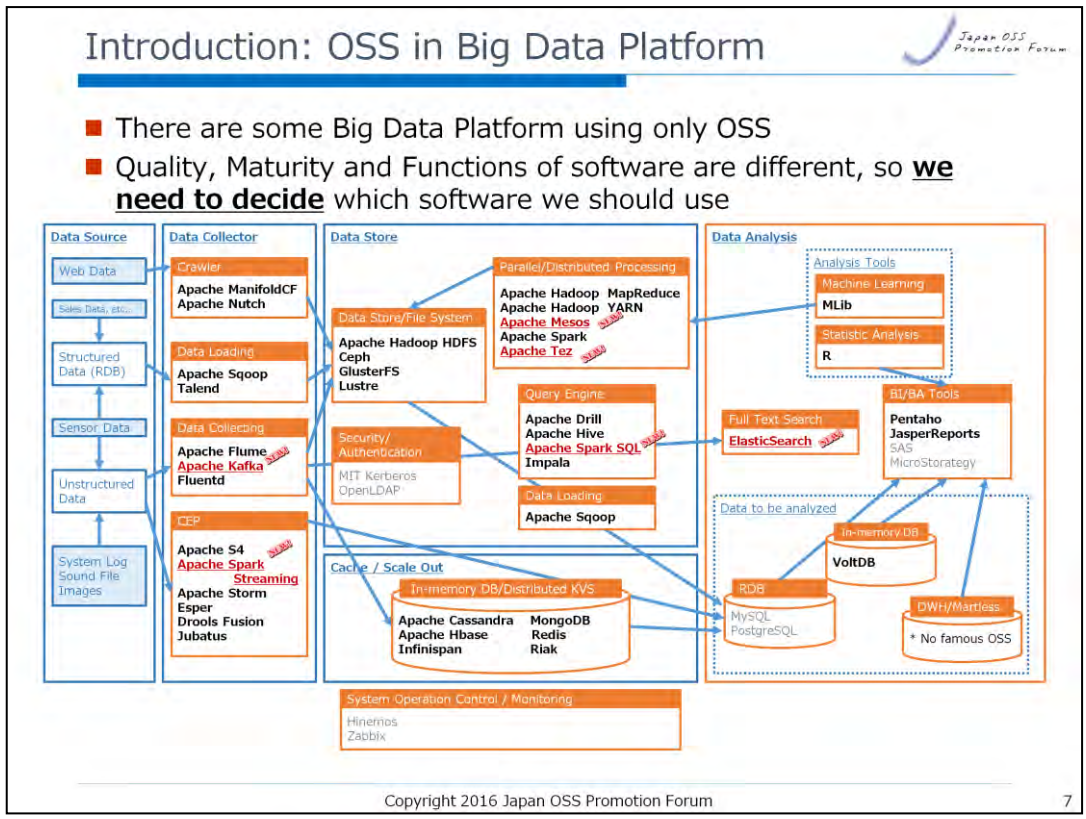
ディープラーニングの動向

今話題のディープラーニングを実現するためのフレームワークに関する調査を行い、また部分的に実機にて性能を検証しました。

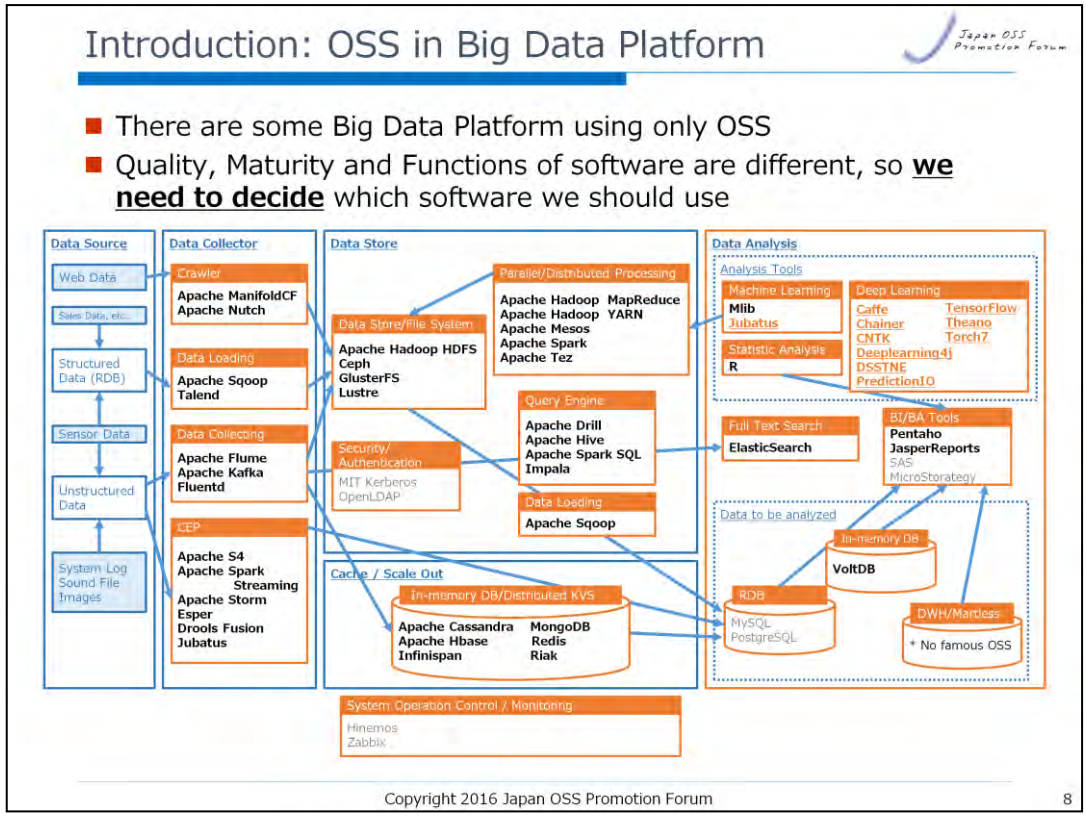

Japan OSS Promotion Forum
Result of Data Analysis
Japan OSS Promotion Forum
Copyright 2016 Japan OSS Promotion Forum



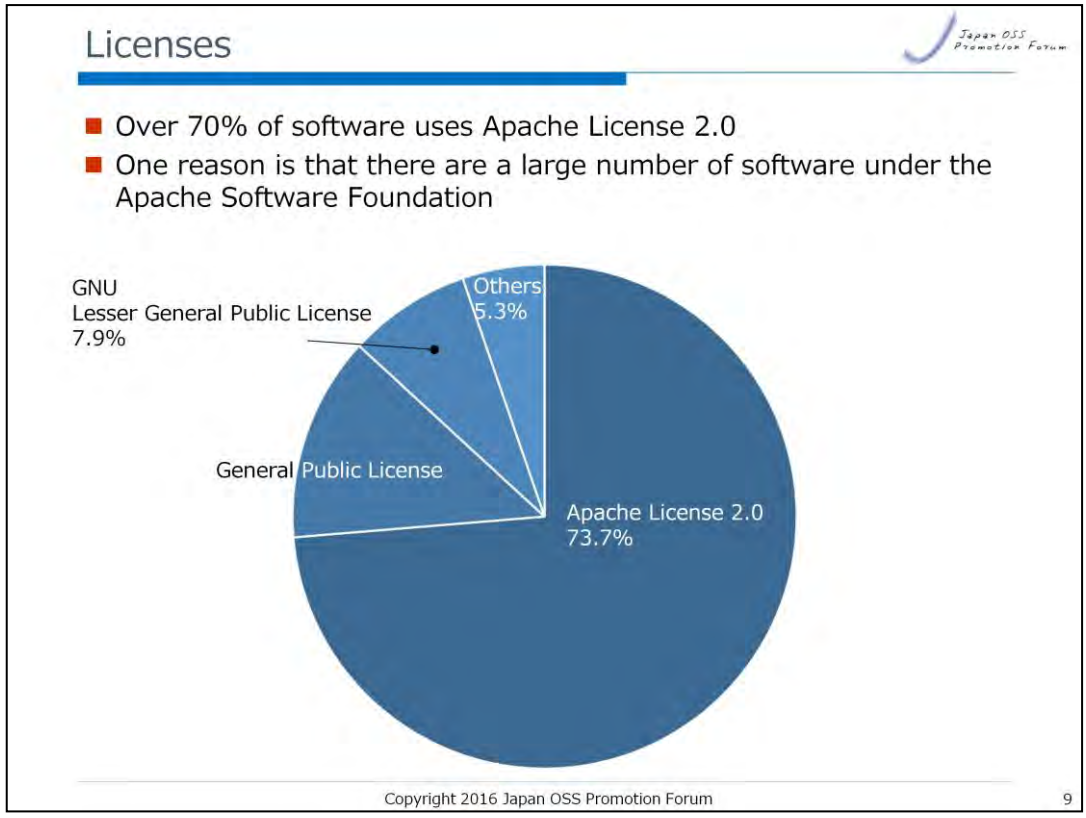
ビッグデータを処理する基盤は、主に、様々なデータを「収集・検知」する機能、
収集・検知したデータを「蓄積・貯蔵」する機能、蓄積・貯蔵したデータを
「分析」する機能から構成されています。



ビッグデータ基盤をオープンソースをベースとして構成する事は十分に可能です。
ただし、個々のソフトウェアにおいて品質や成熟度が異なる為、見極めが必要

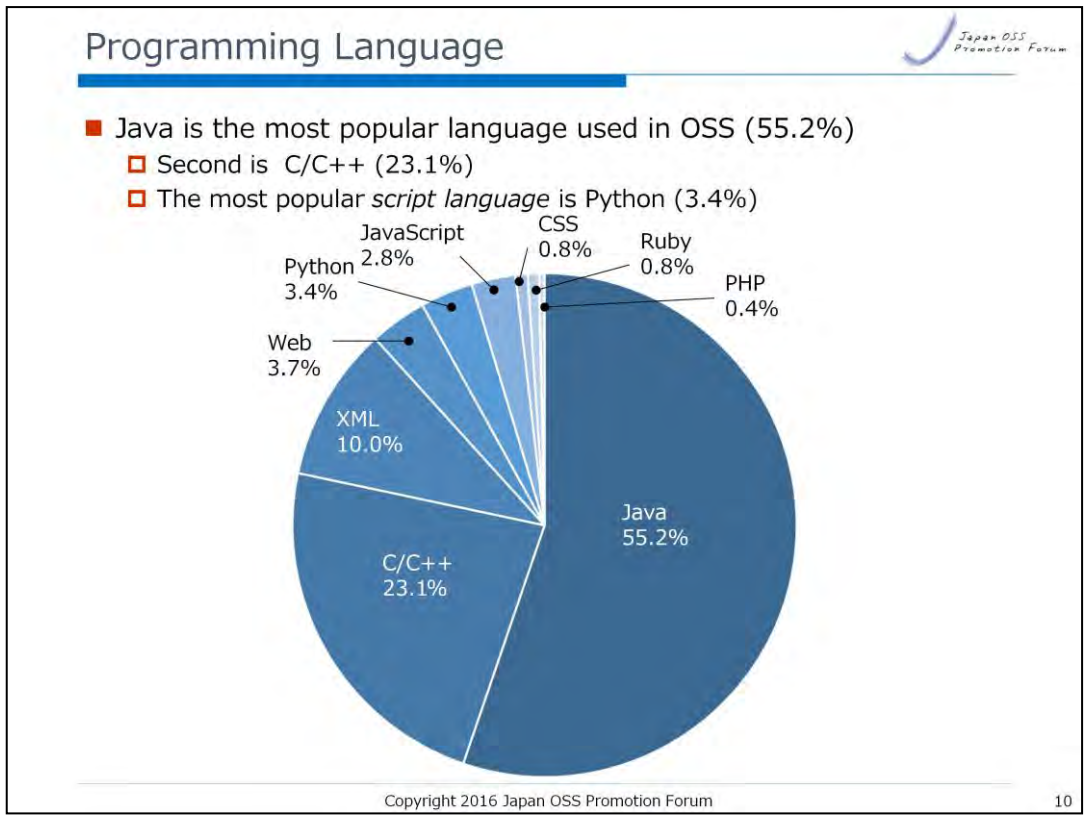


これは、今年度情報を更新した図です。



ライセンス

OSSの70%以上がApache2.0のライセンスを採用しています。
その理由は、多くのソフトウェアがApacheソフトウェア財団の下で、
開発しているからです。

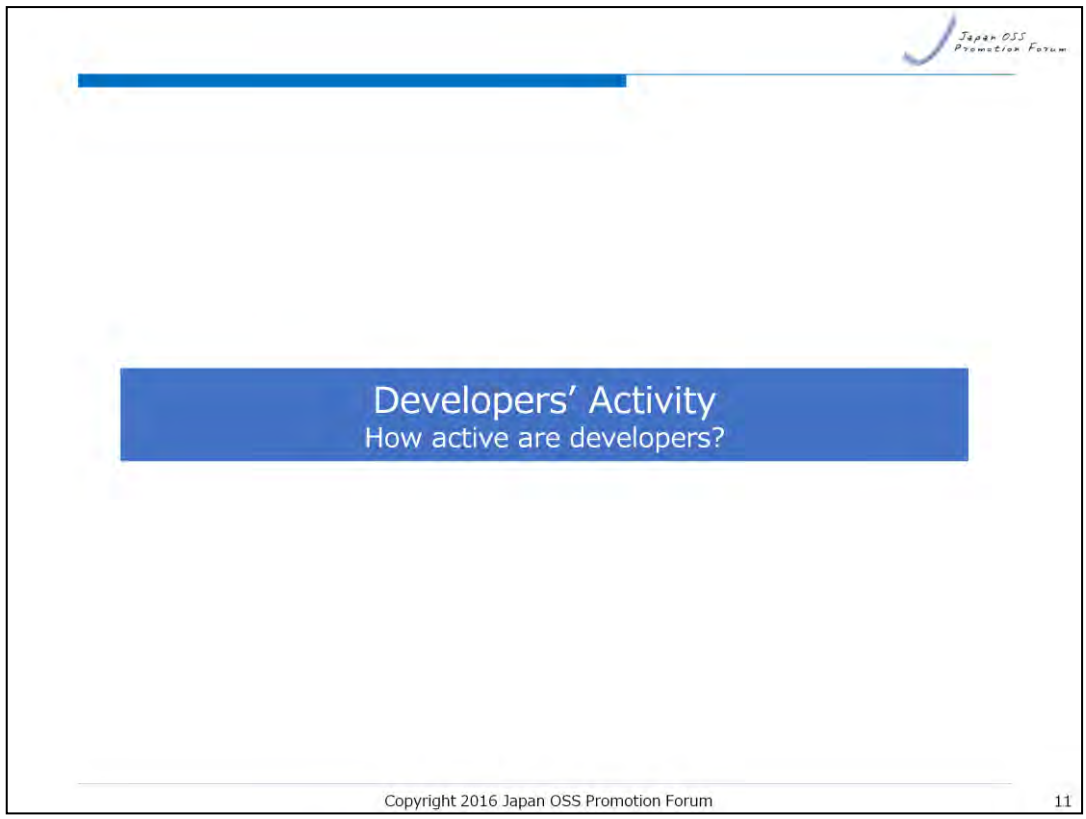


次に開発言語です。

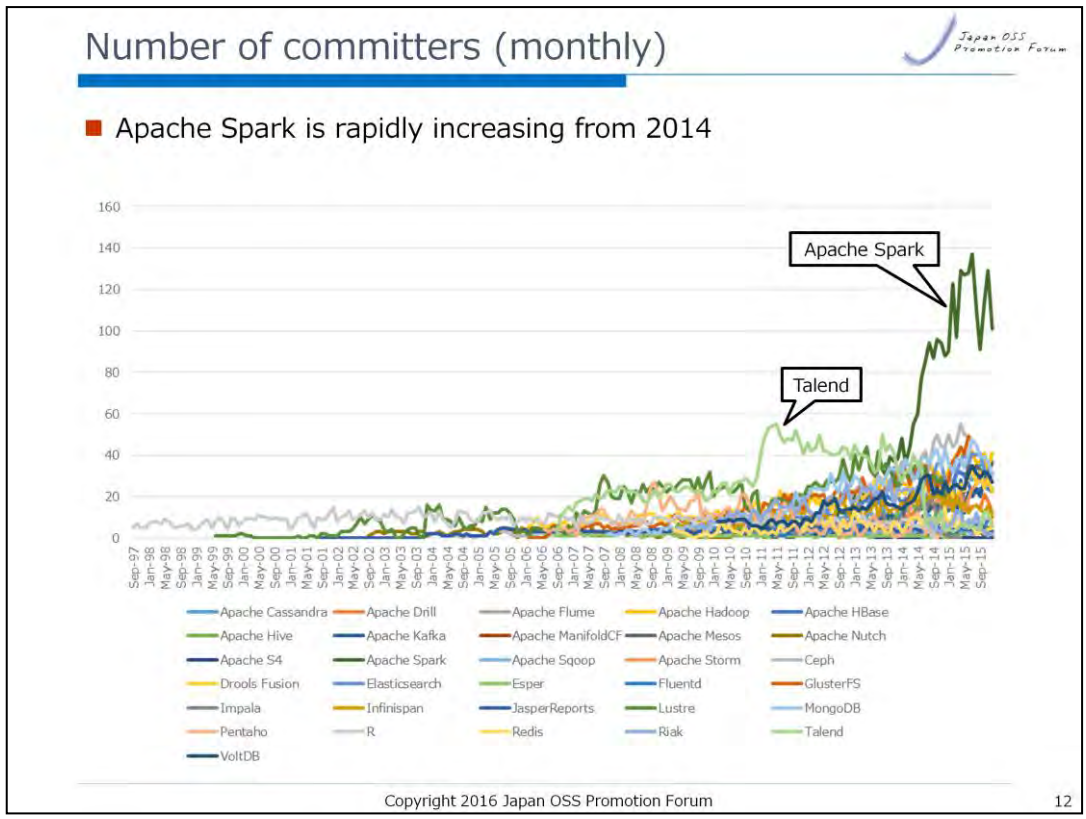
・約半数のOSSがJavaで開発されています。

・2番目は、C及びC++言語です。

・最も使用されているスクリプト言語は、Pythonです。
最近では、OpenStackもPythonで記述されるなど、
全世界的に非常に使用されています。

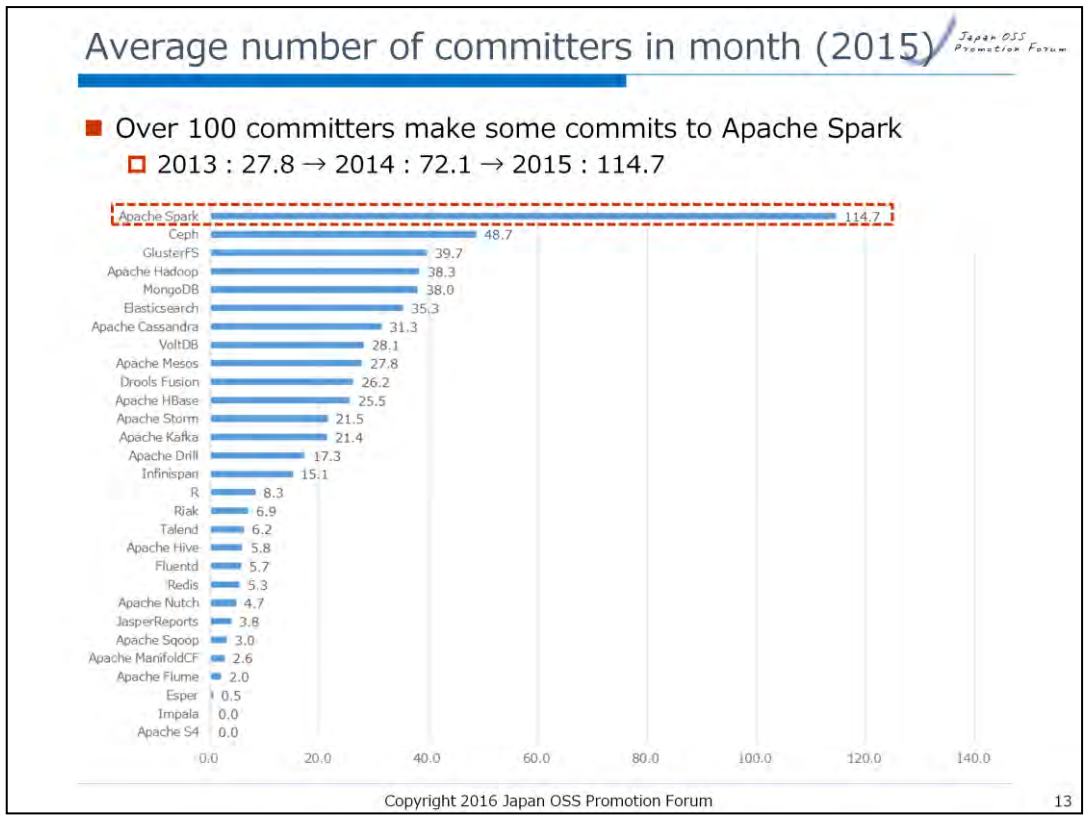


開発の活性度について、説明します。

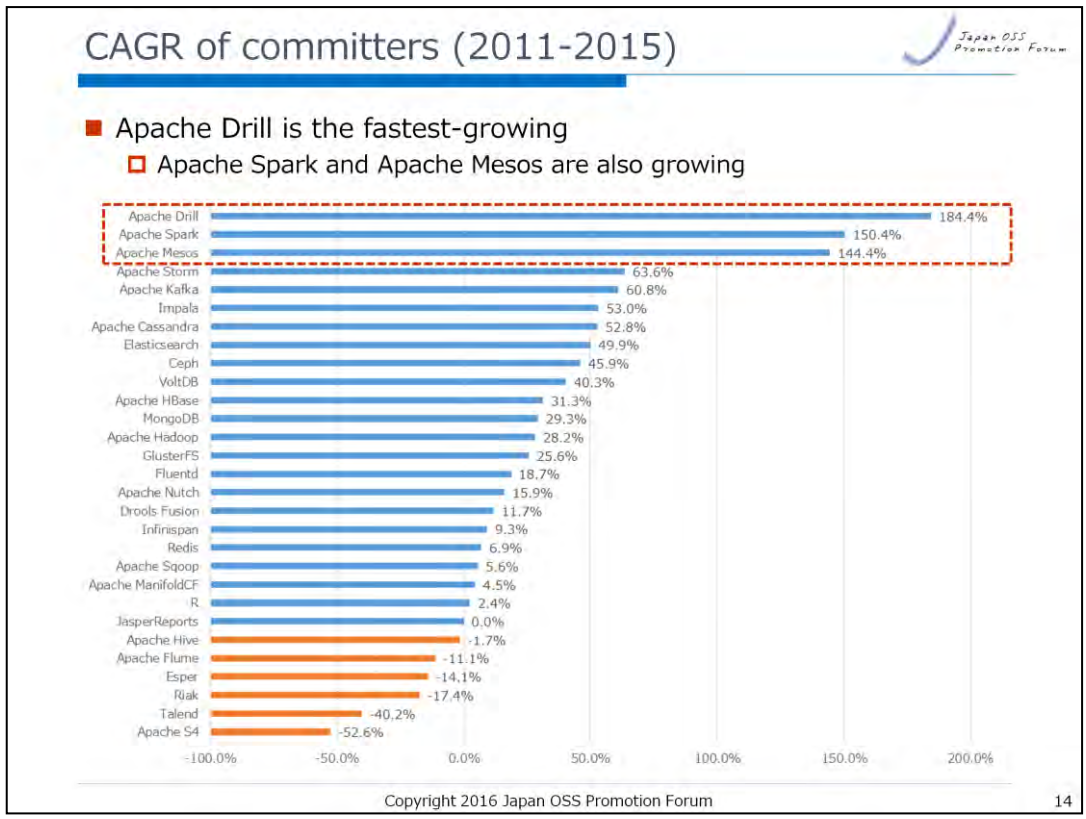


まず、コミッター数の推移です。
ソースコードを更新できるコミッターの増加を図ることで、
開発の活性度を測ることができます。

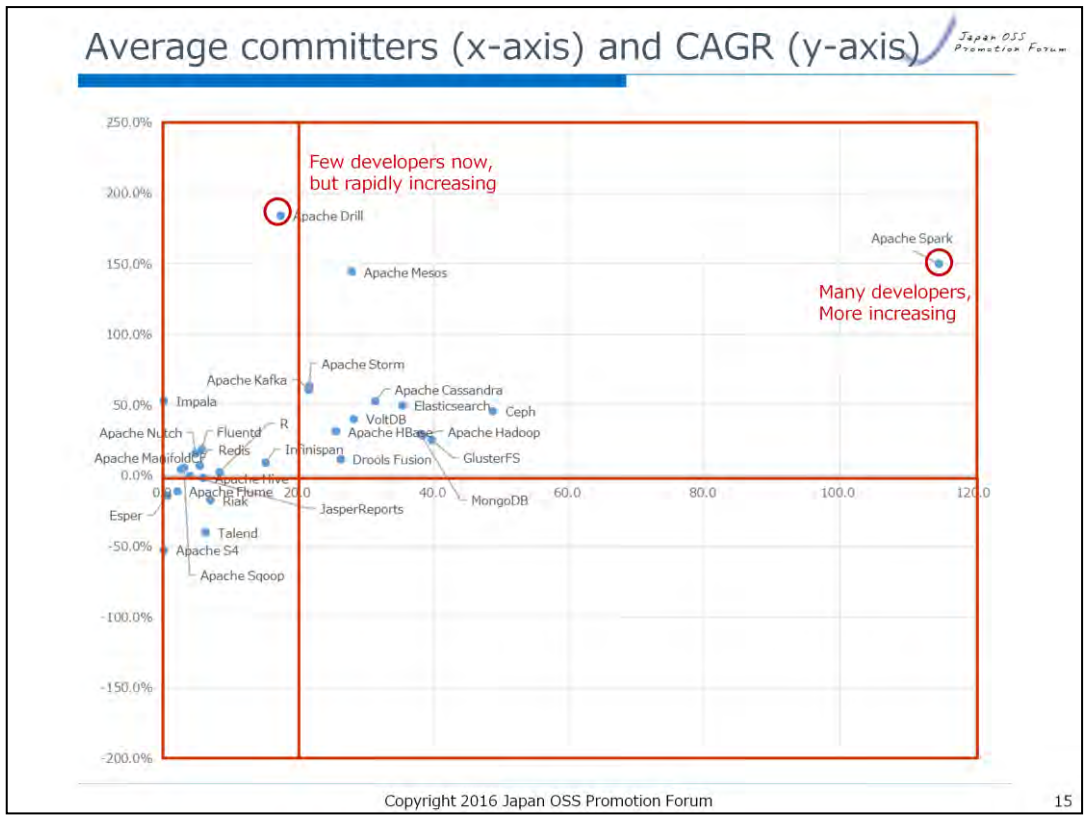
ここでは、Sparkが2014年ころから急速に成長していることが
わかります。



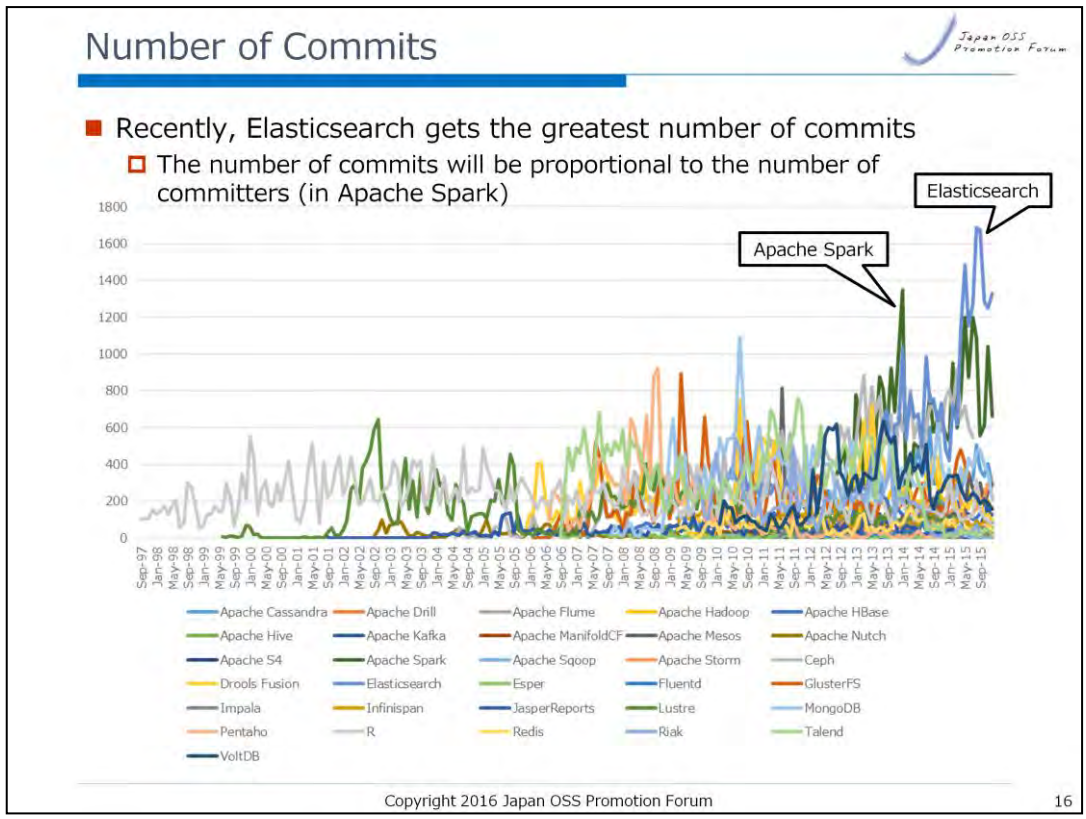
次に月当たりの平均のコミッタ数です。
Apache Sparkのコミッター数は、2013年から
飛躍的に増加しています。



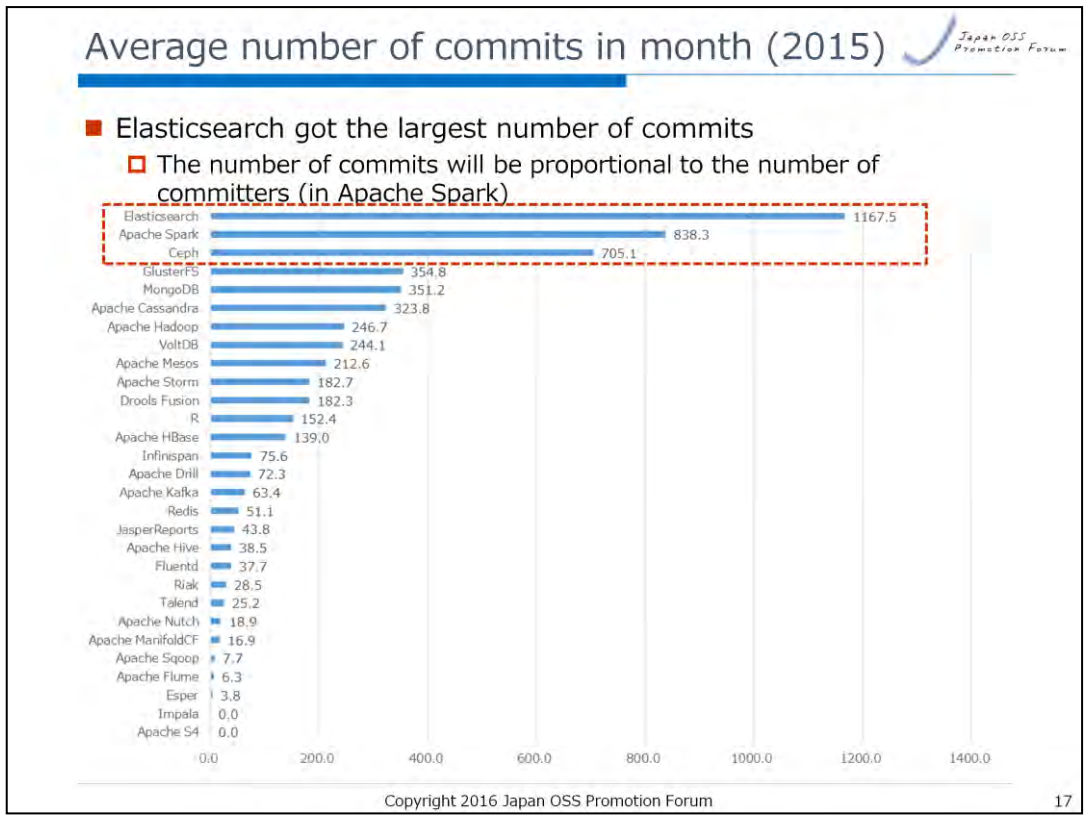
2011年から2015年までのコミッタ数の平均がこのグラフになります。
ここでは、Sparkだけではなく、Mesosも増加していることが
わかります。



月平均コミッタ数を年平均成長率で相関関係をグラフにしました。
縦軸が成長率、横軸がコミッタ数になります。
左上のDrillは、開発者の数はまだ少ないですが、急増中ということがわかります。
また、左上のSparkは、開発者も多く、また増加中であることがわかります。



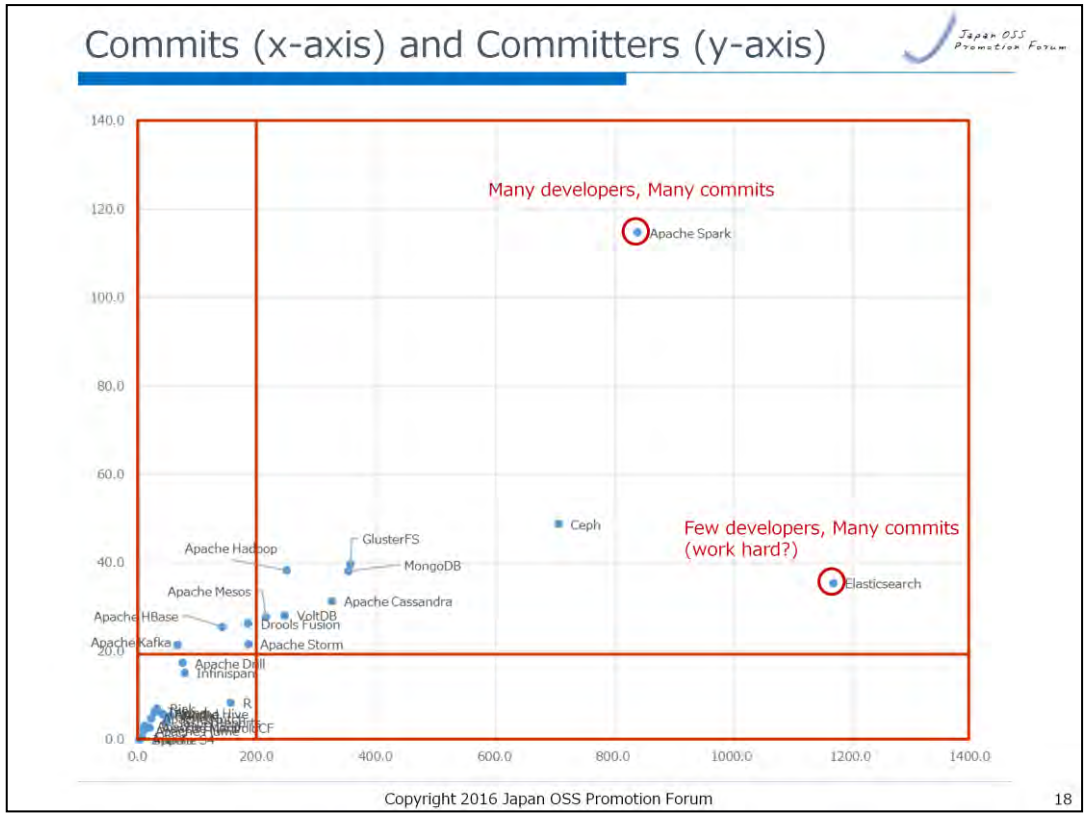
次にコミット数の推移です。
2013年ころは、Sparkが活発でしたが、
現在、最も活発なのは、ElasticSearchという
検索エンジンです。
ほぼ、検索エンジンではデファクトの地位を占めていると
いってよいと思います。



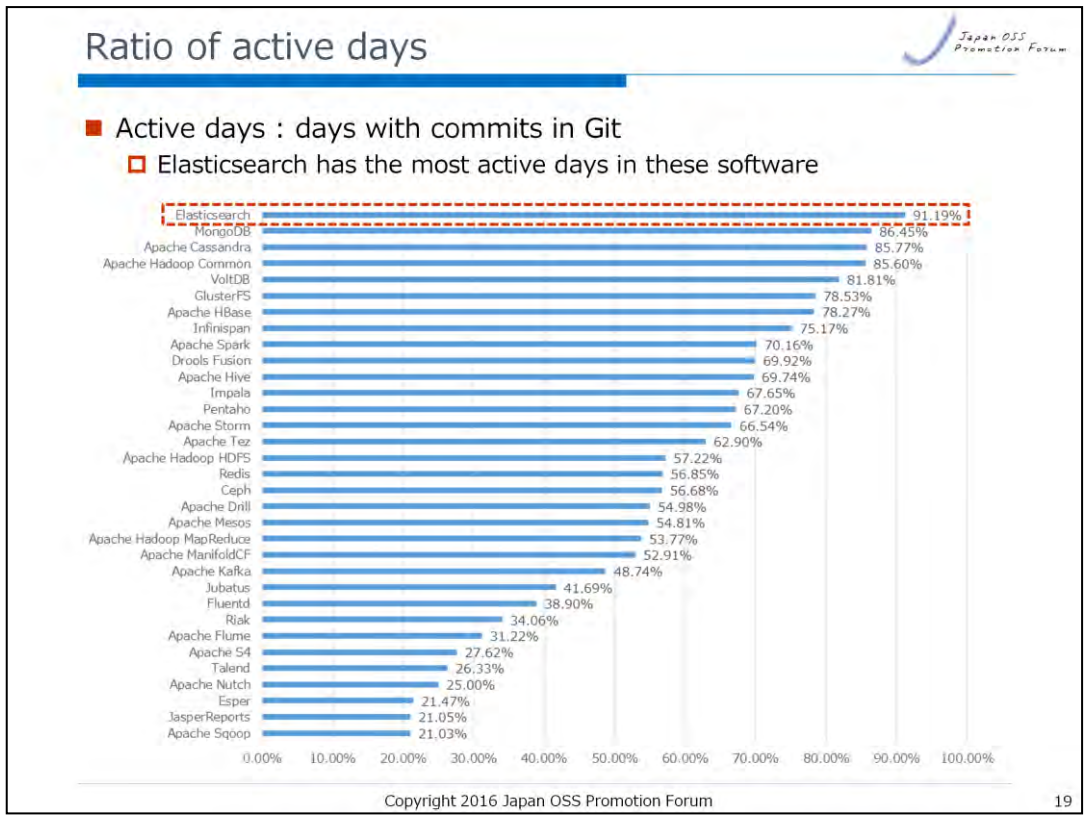
コミット数の月平均のグラフです。

ここでも、ElasticSearchが一番です。

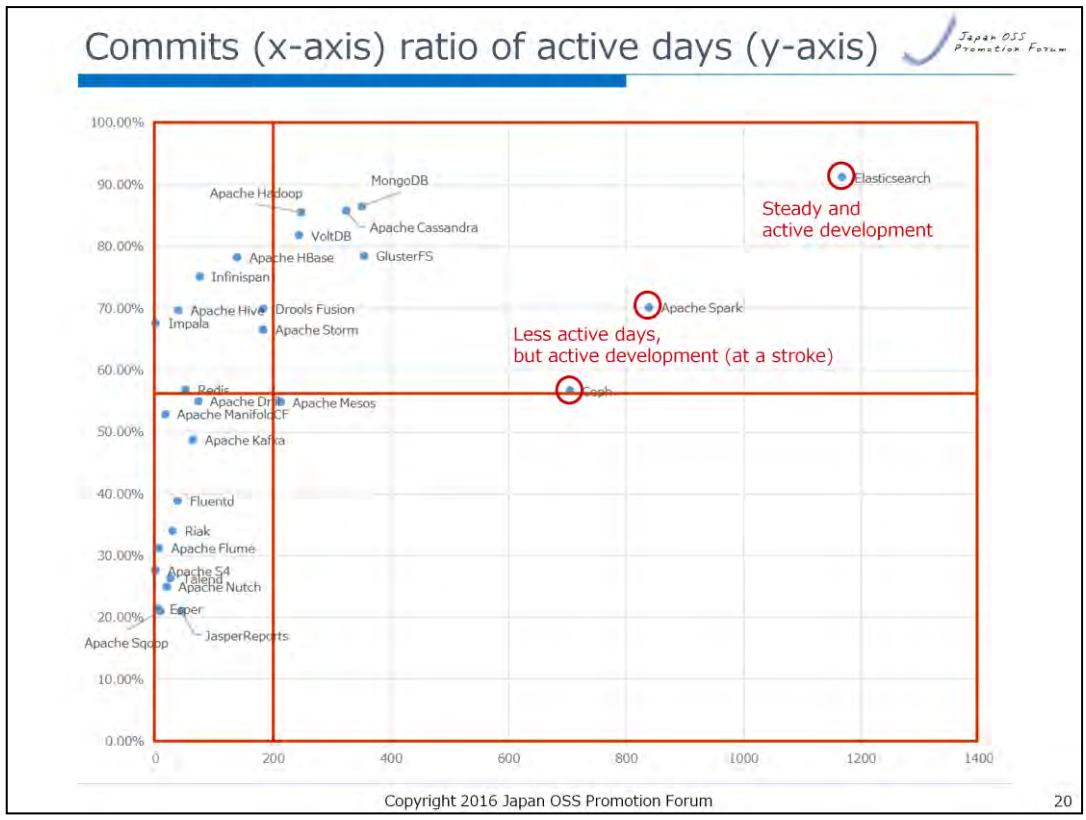
もちろn、Sparkも多くなっています。



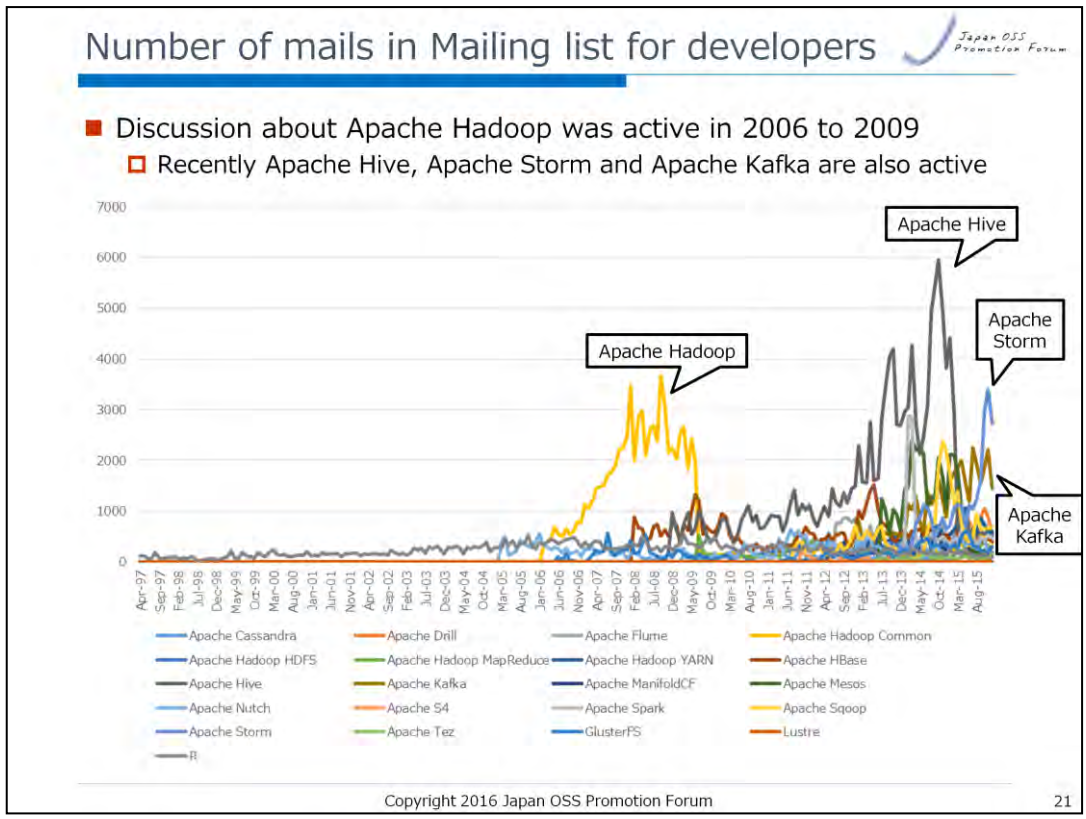
これは、コミッタ数をコミット数の相関関係を表したグラフになります。
左上のSparkは、開発者の数も多く、開発も活発であることが良く分かります。
一方、右下のElasticSearchの場合は、開発者の数はそれほど多くはありませんが、
開発が非常に活発であることがわかります。



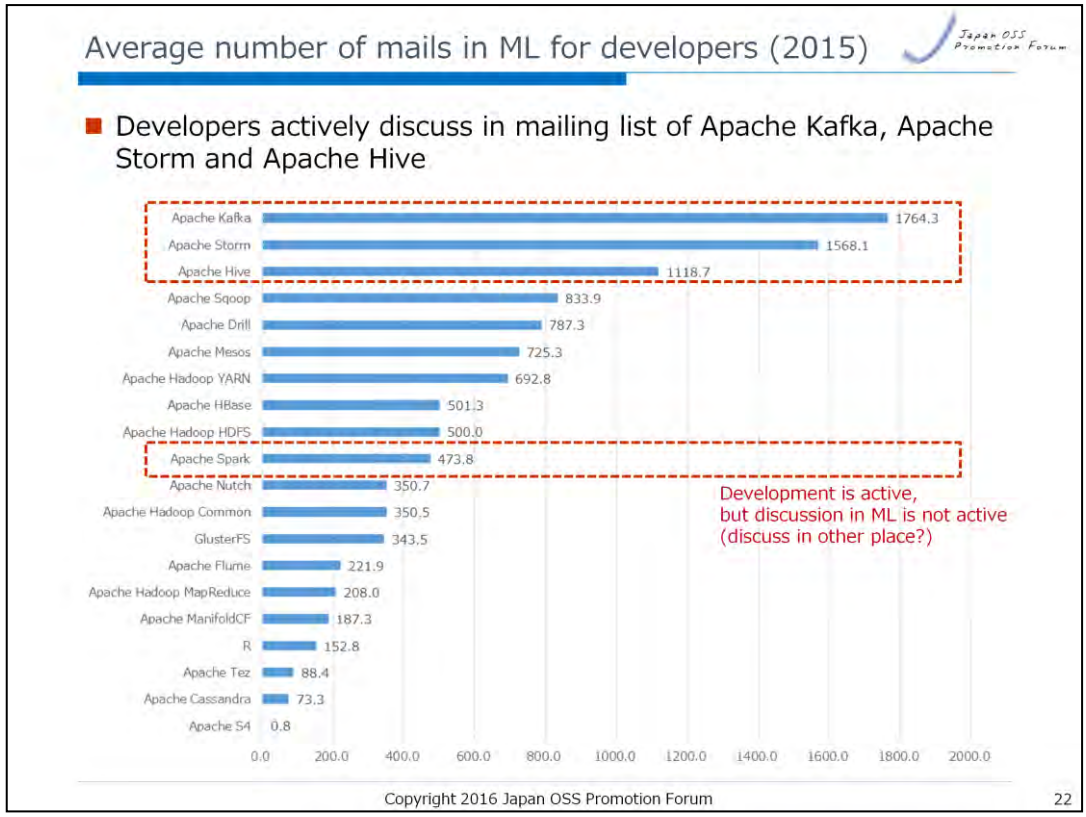
次は、稼働率のグラフになります。
一日のうち、どれくらいの時間を開発に使用しているかがわかるグラフです。
これを見ると、ElasticSearchが一番活発であることがわかります。



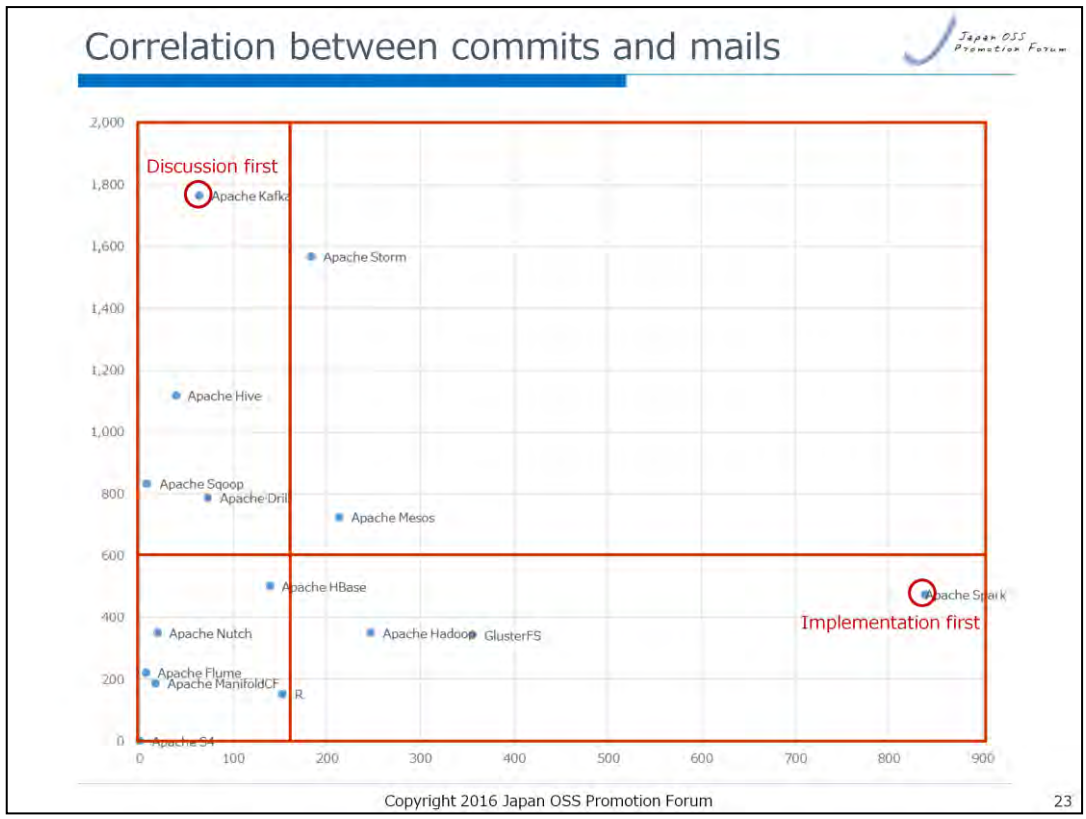
月平均のコミット数を横軸に、稼働率を縦軸にとった相関関係のグラフです。
これを見ると、右上にあるElasticSearchは、稼働率も高く、開発も活発にするタイプで、いわゆるコツコツとやるタイプであることがわかります。
また、真ん中上にあるSparkは、稼働率があまり高くありませんが、開発が活発なので、どちらかというと開発を一気にやるタイプであるといえると思います。



これは、開発者むけのメーリングリストの流量の推移を表したグラフです。
2006年から2009年にかけては、Hadoopの議論が活発でした。
最近はApache HiveやApache Storm、Apache Kafkaの議論が活発



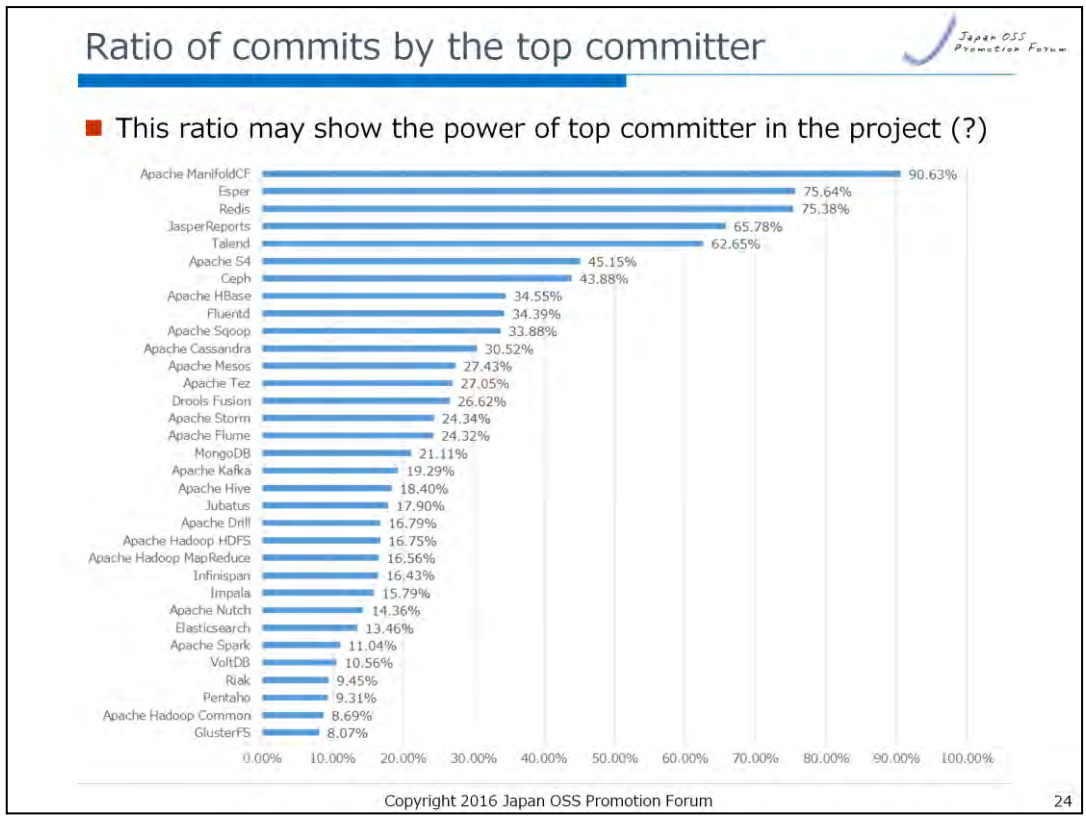
2015年の開発者向けメーリングリストの月平均流量です。
Apache KafkaやApache Stormの議論が活性化していることがわかります。
また、Apache Hiveは2014年をピークとして2015年は落ち着いている模様であることもわかります。
一方、Sparkは、開発は活発ですが、メーリングリストでの議論は、あまり多くありません。



コミット数とメールの議論の相関関係です。

右下のSparkは、議論よりも開発に集中していることがわかります。

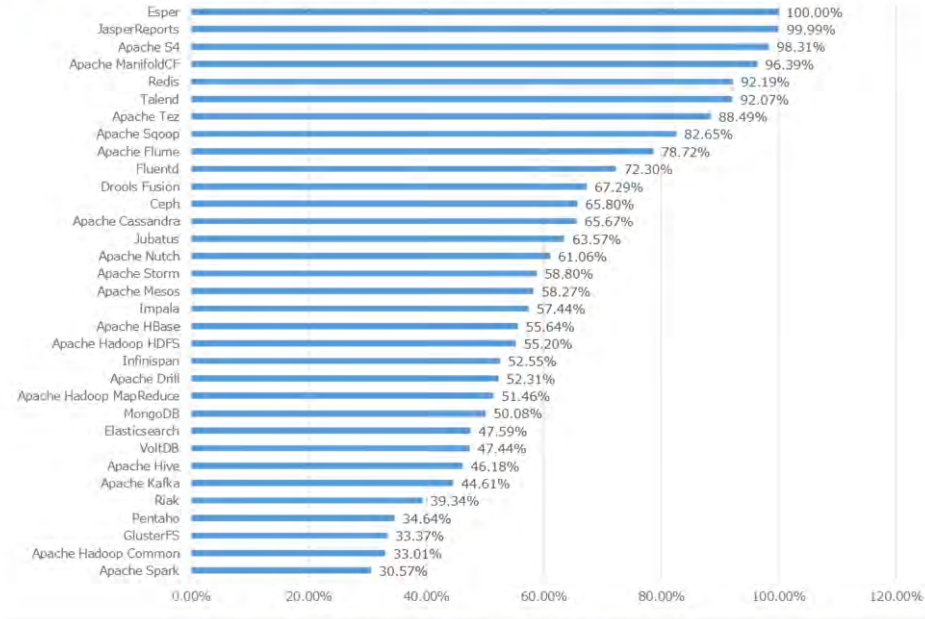
一方、左上のKafkaは、まだまだ議論が中心でであることがわかります。



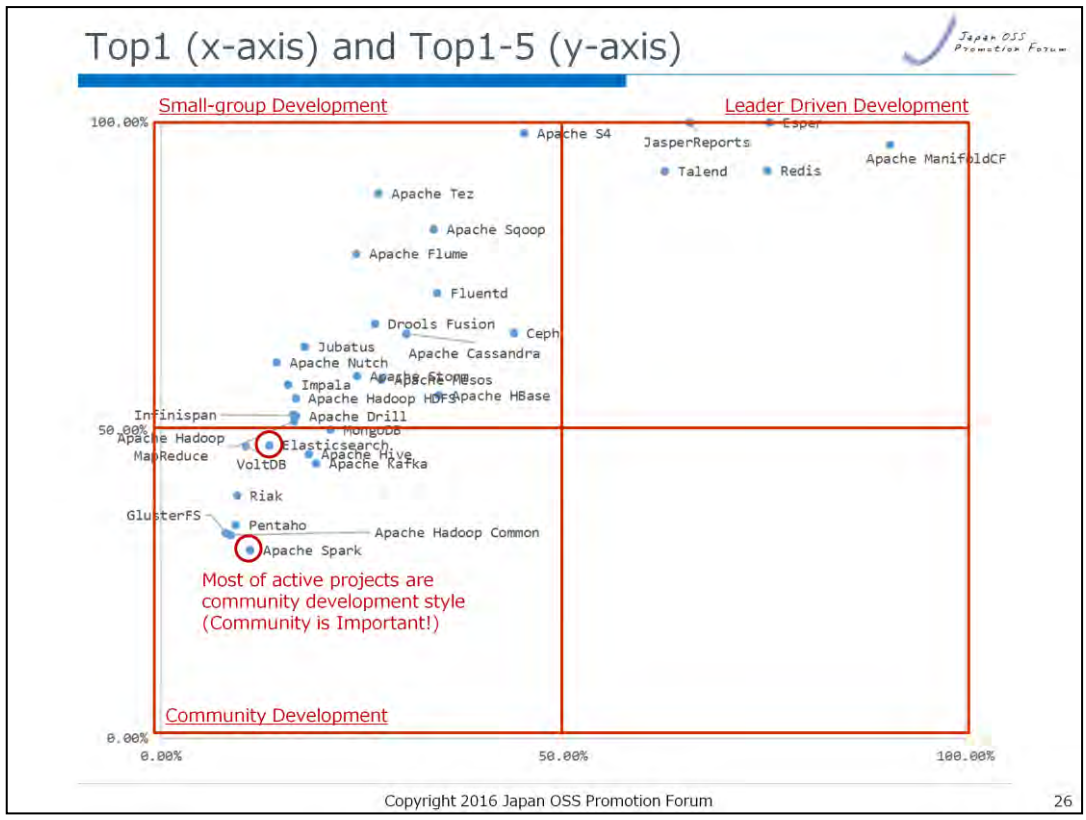
トップコミッターのコミット数の比率です。

Ratio of commits by the top 1-5 committers

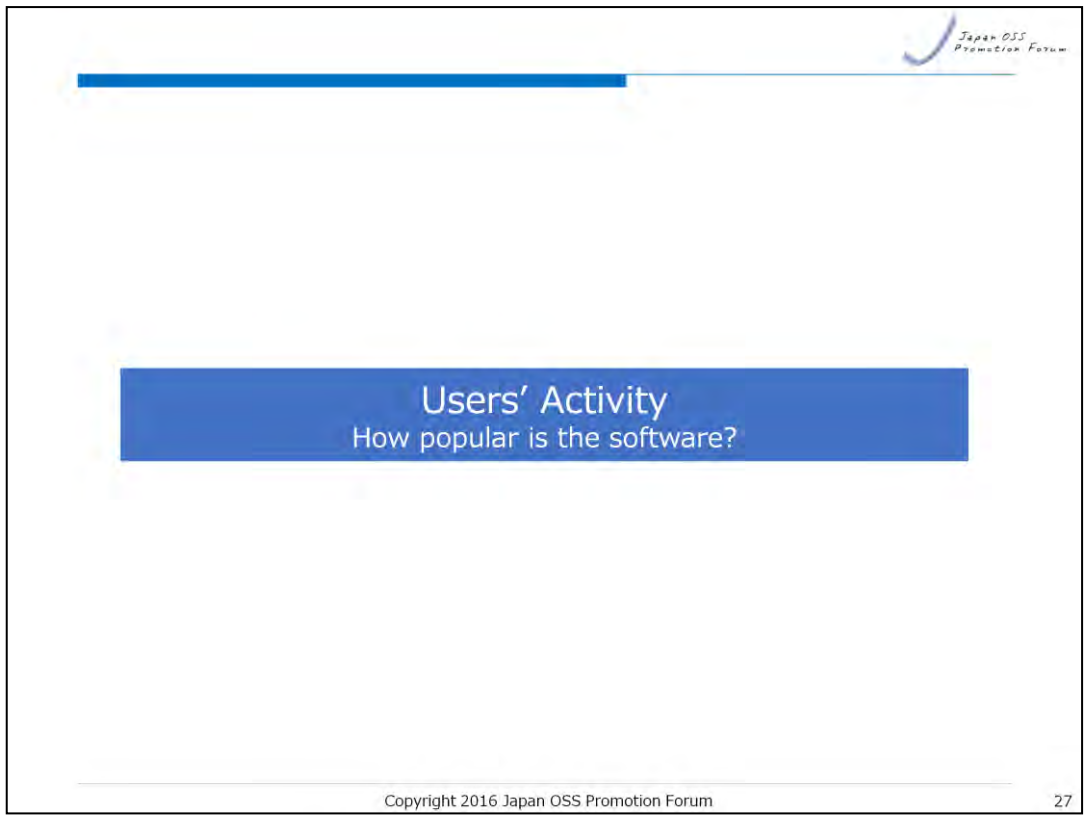
- Esper and JasperReports may be committed by only 5 committers



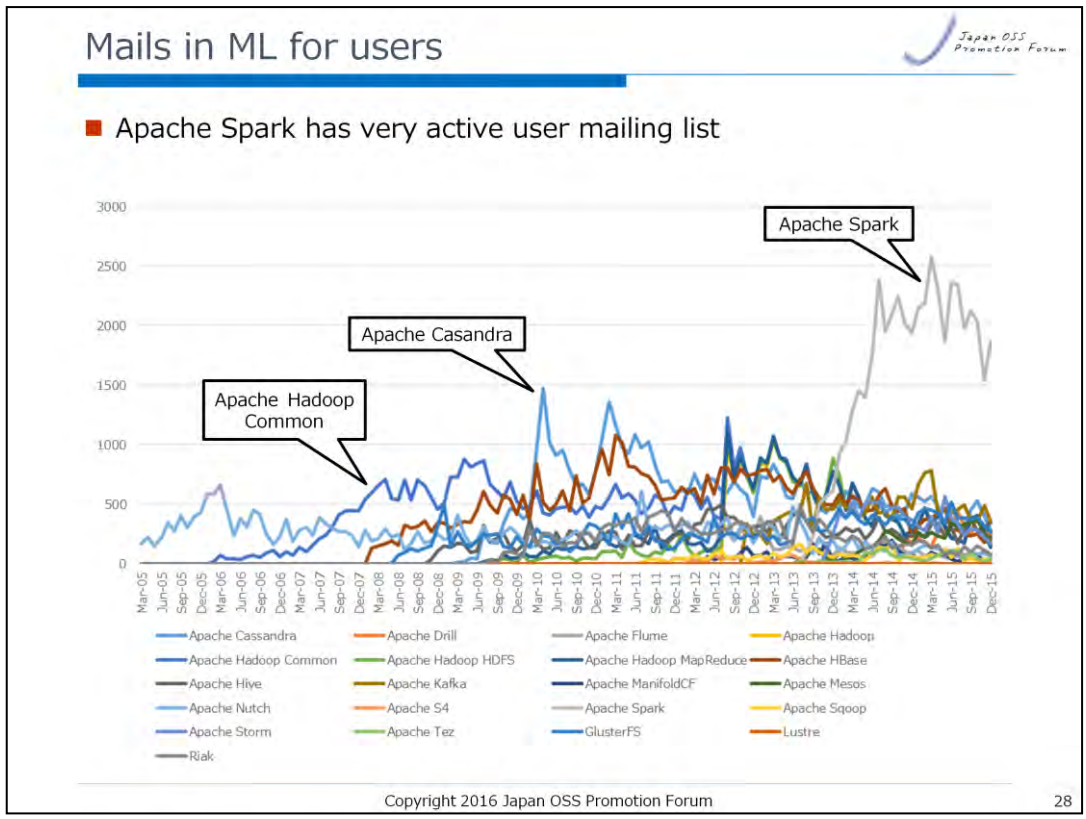
Copyright 2016 Japan OSS Promotion Forum



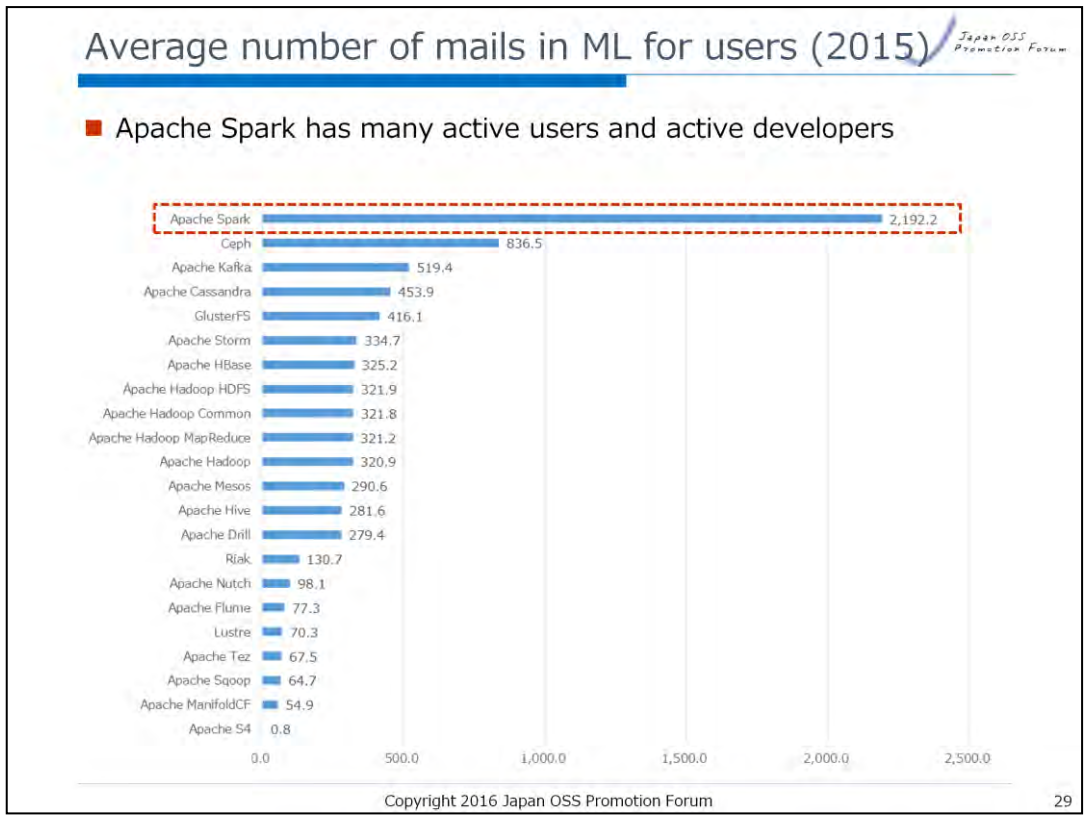
コミュニティの開発形態がよくわかるグラフです。
開発が活性化しているプロジェクトは、コミュニティ型開発が多いことがわかります。
当初は、リーダのみの開発から少人数への開発、やがてコミュニティ型へと
開発のスタイルが変わっていることで、開発も進みますし、
みなさんに使ってもらえるようにもなるということが良く分かります。



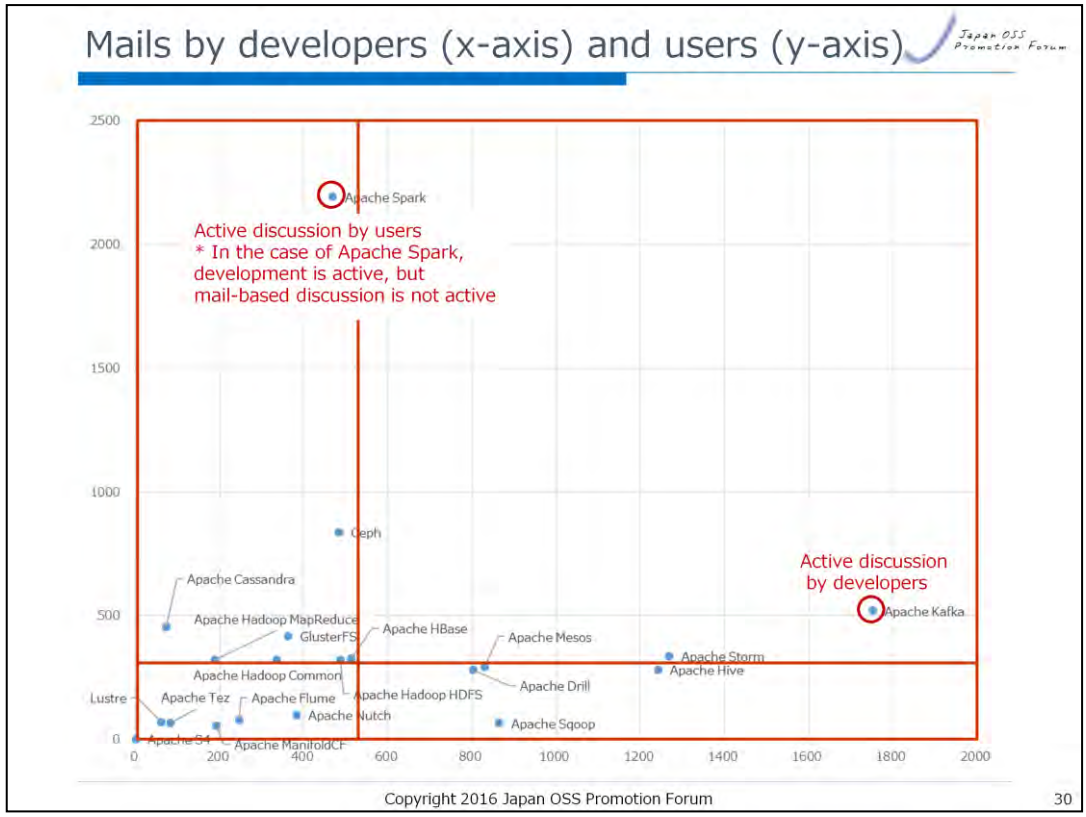
次に利用の活性度について説明します。
つまり、そのソフトウェアがどれだけ利用されているかということです。



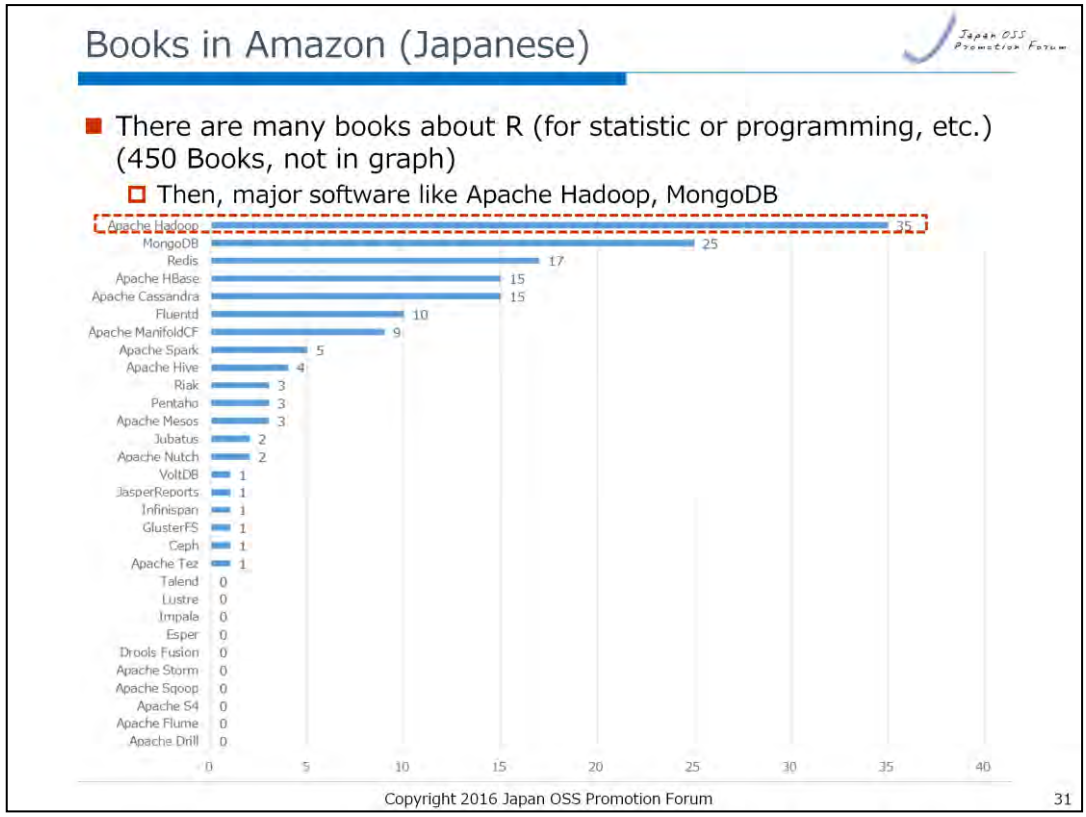
2008年頃は、Hadoopの登場したころなので、質問が集中しています。
その次は、Cassandraについての議論が増えています。
2014年以降は、こちらもでSparkについての議論が圧倒的に活発になっています。



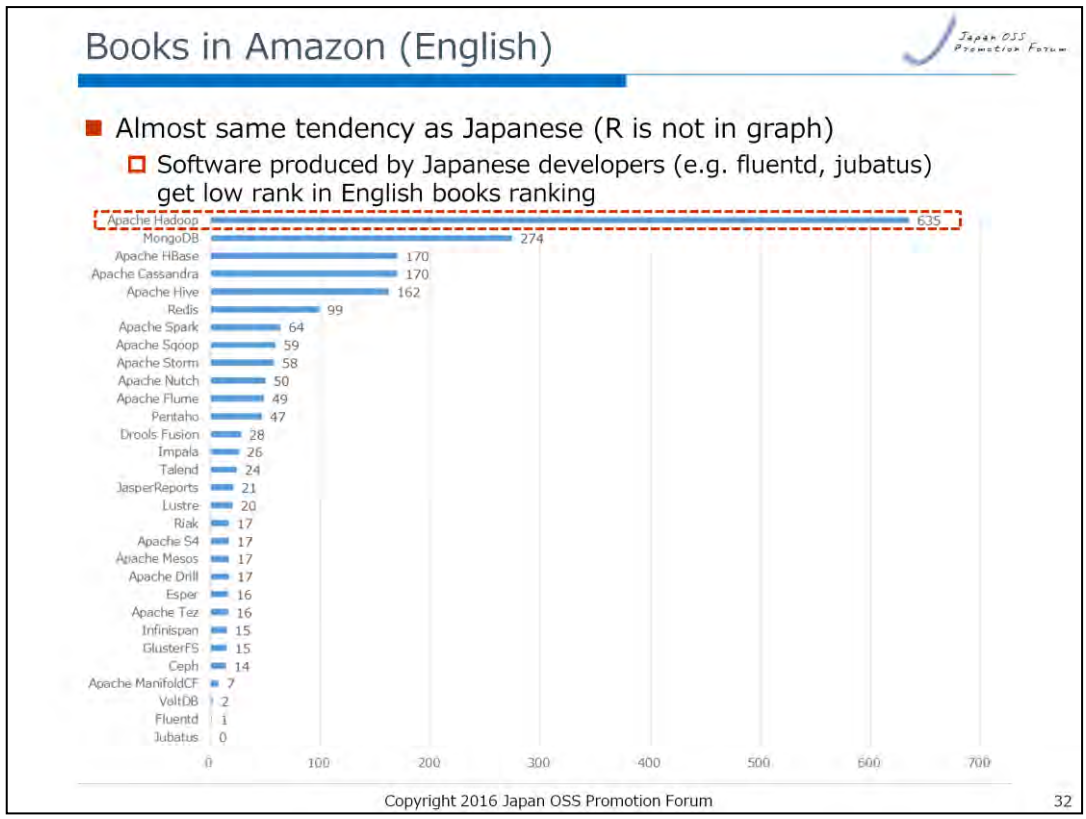
2015年だけを抜き出してグラフにしてみると圧倒的にSparkが活発です。
その次に活発なのは、SDS(Software Defined Storage)のCephです。



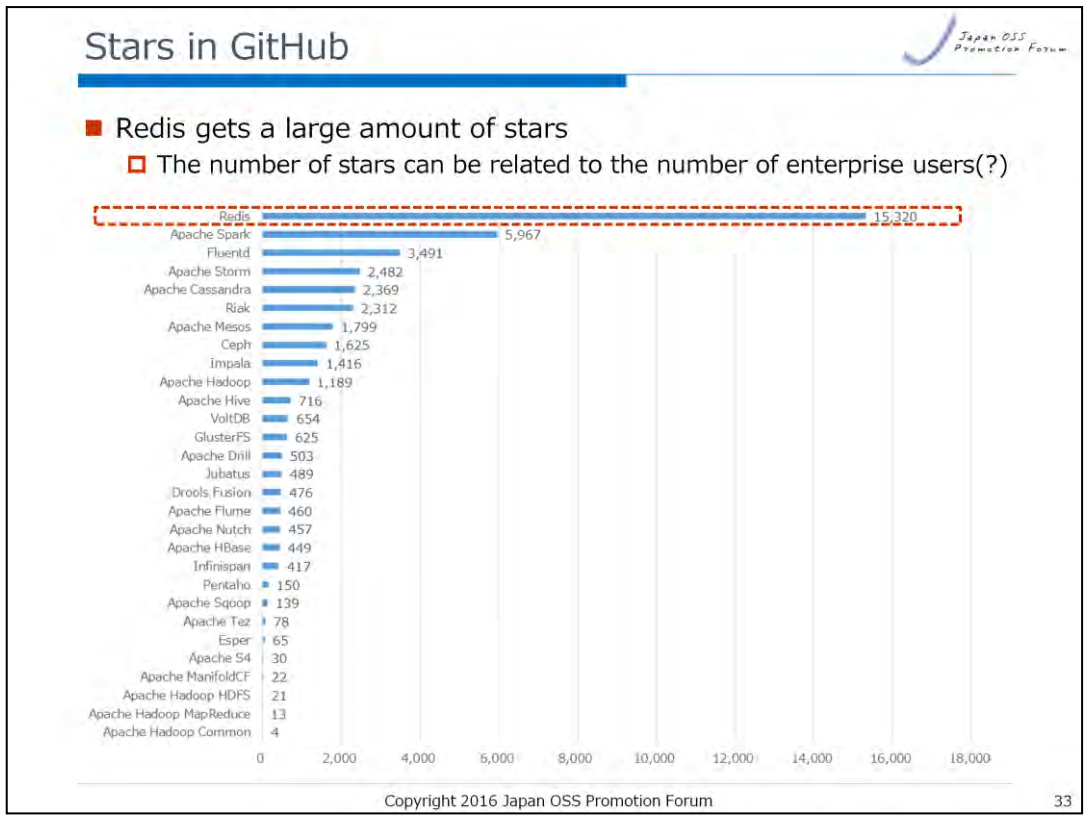
このグラフは、横軸に開発者、縦軸に利用者の相関関係を表したグラフです。
左上のSparkは、利用者の議論が活発であることがわかります。
Sparkに関しては、先ほど説明したように開発は活発ですが、
開発者のメールベースでの議論は、少なめです。
かたや、右下のKafkaは開発者の議論が活発であることがわかります。
しかしながら、まだそれほど利用されていないことが利用者のメールの
少なさでわかります。



次にアマゾンでの書籍数を比較しています。
もちろん、日本で調べているので、日本語の書籍の数のみです。
しかしながら、R言語は言語や統計の分野で使われることが多いので、
このグラフからは、削除しています。
このグラフでは、Hadoop、MongoDBなどの書籍が多いことがわかります。

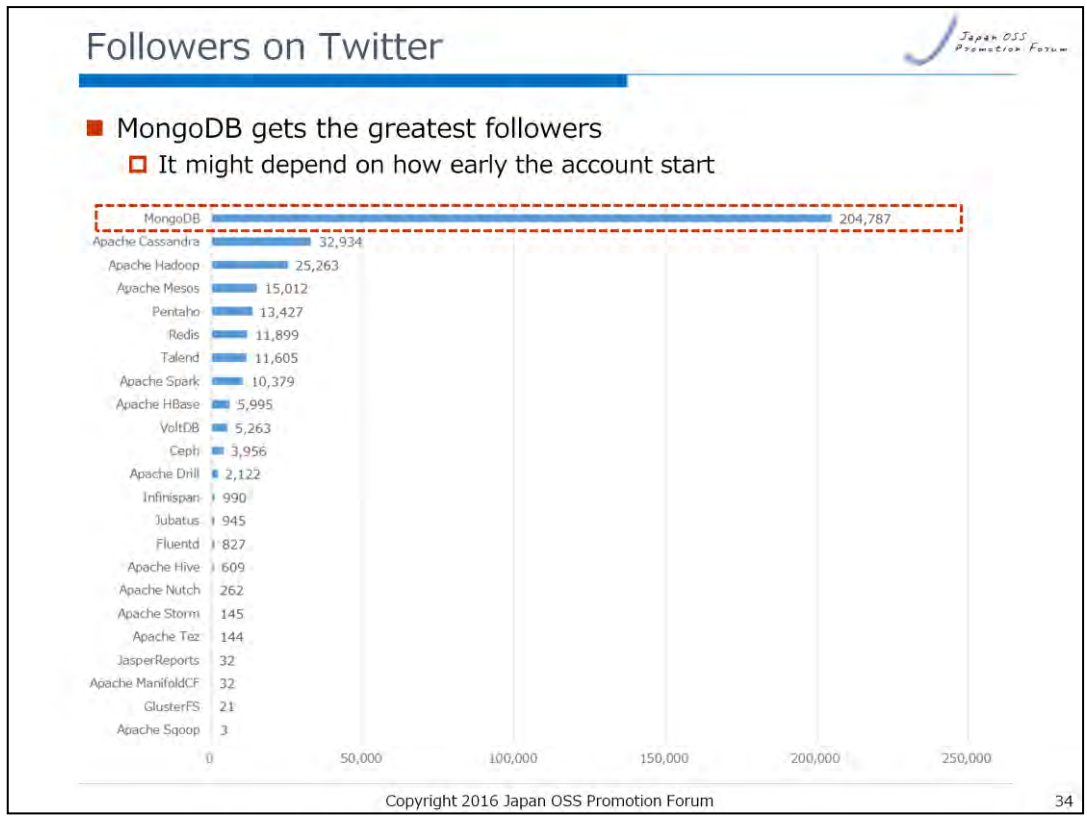


次は、英語の書籍のデータです。
ほとんど日本語の傾向と同じように見えます。
このなかで、FluentdやJubatusは日本人が中心になって
開発しているソフトウェアなので、日本語の書籍に比べると
少なくなっています。

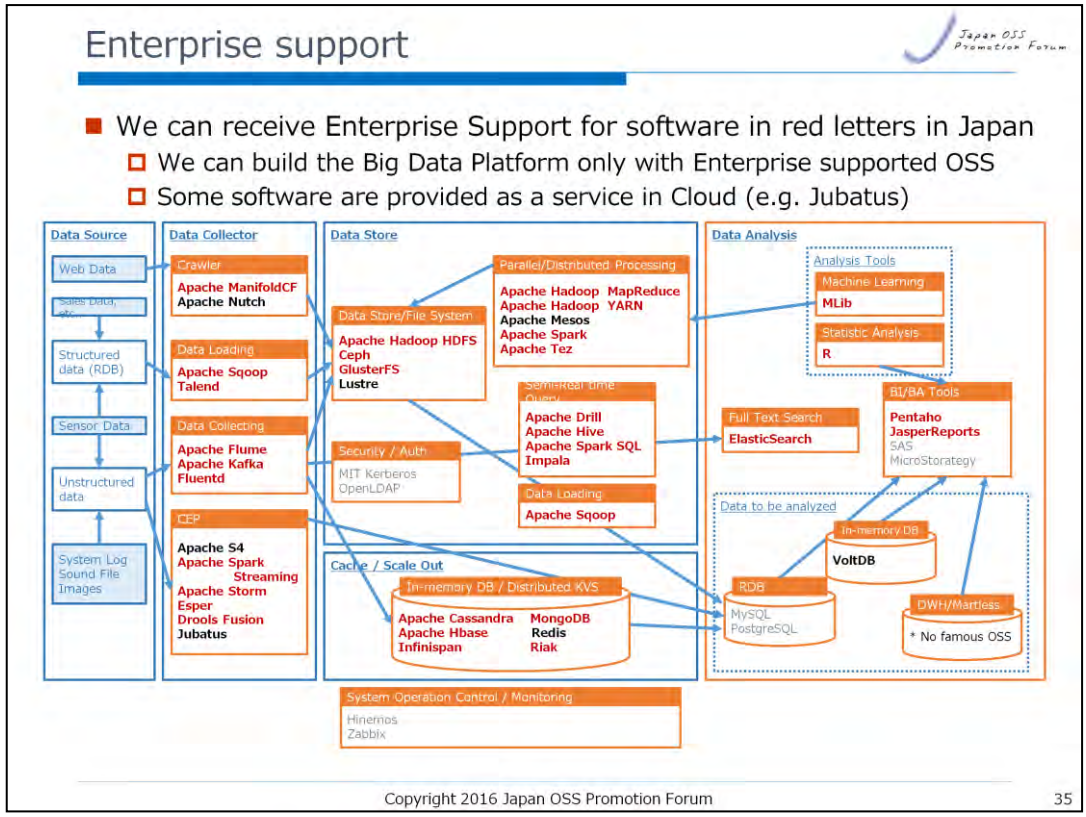


GitHubのスター数です。
これは、スター数が多いほど、人気の高いプロジェクトであることを表しています。
ここでは、Redisが圧倒的によい評価となっています。

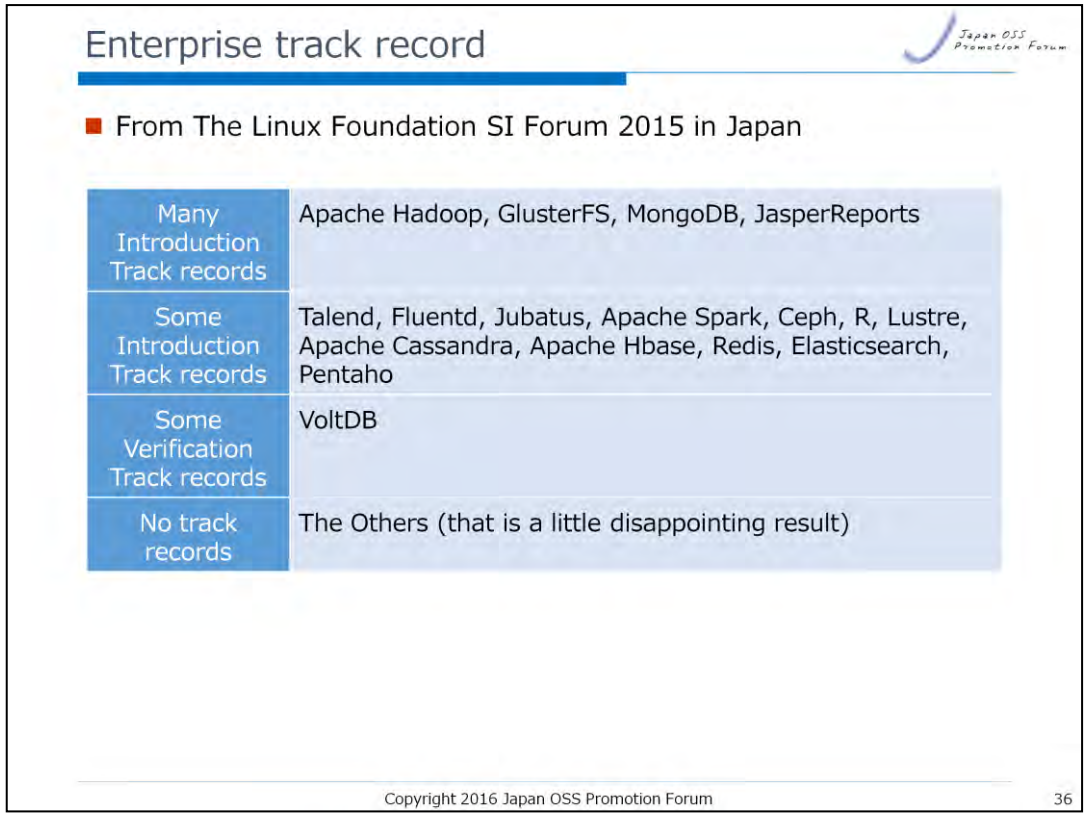
商用で展開されているソフトウェアに関しては、組織票があるかも知れませんね。



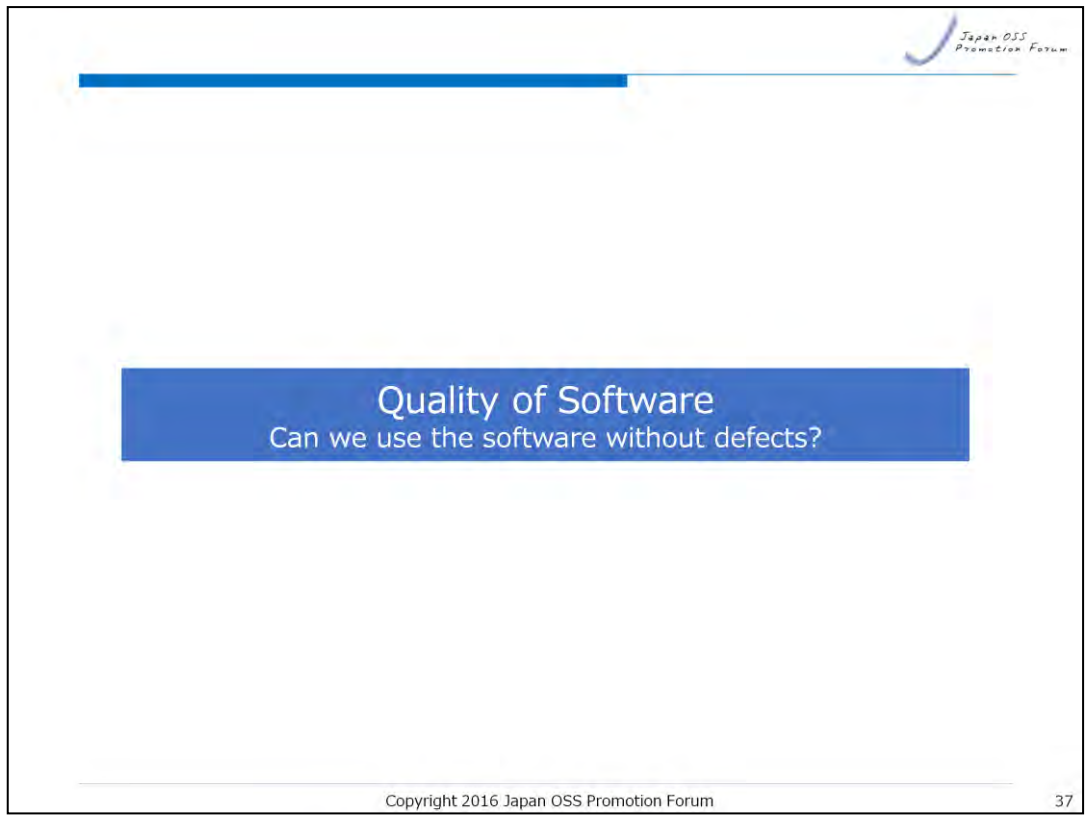
次は、Twitterのフォロアー数です。
アカウントの開設時期にもよりますが、MongoDBが圧倒的に多数に
なっています。



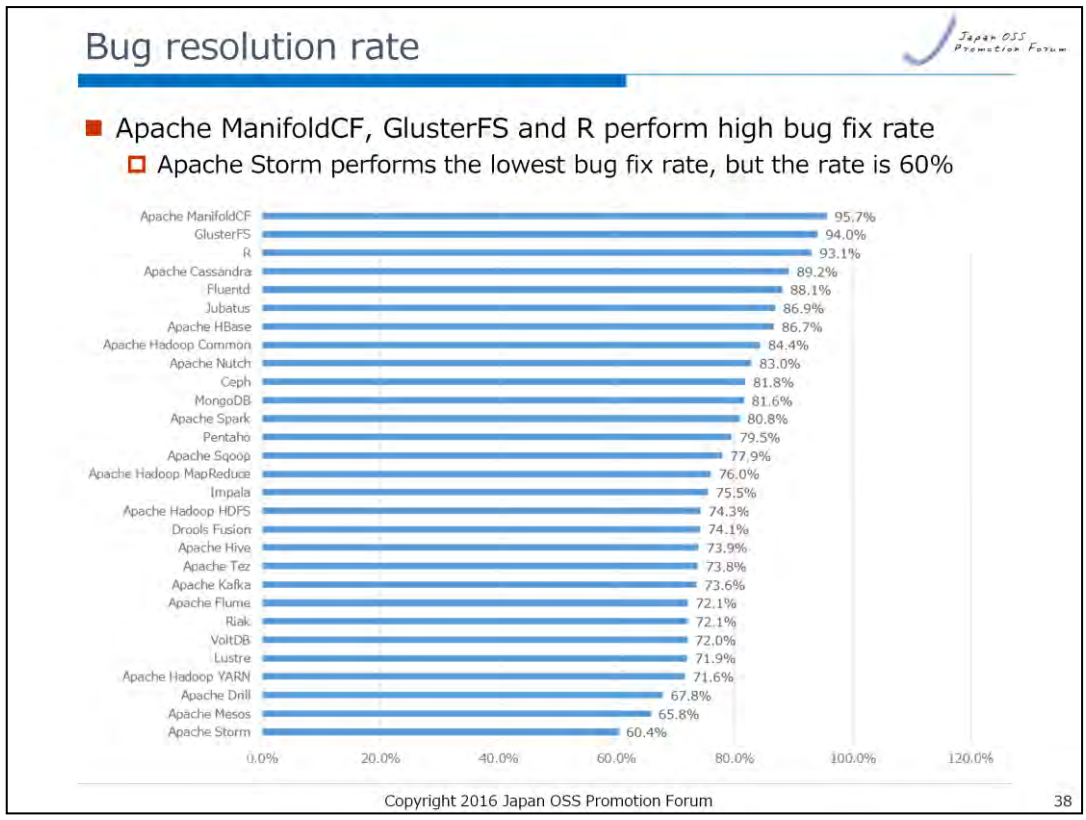
この図は、有償サポートが提供されているものは、赤字で表示してあります。
このようにビッグデータ基盤を構成するOSSは、有償サポートされているものが
多いので、これらOSSだけで基盤を構築することは十分可能です。
また、Jubatusなどはクラウドでサービス提供されているOSSもあります。



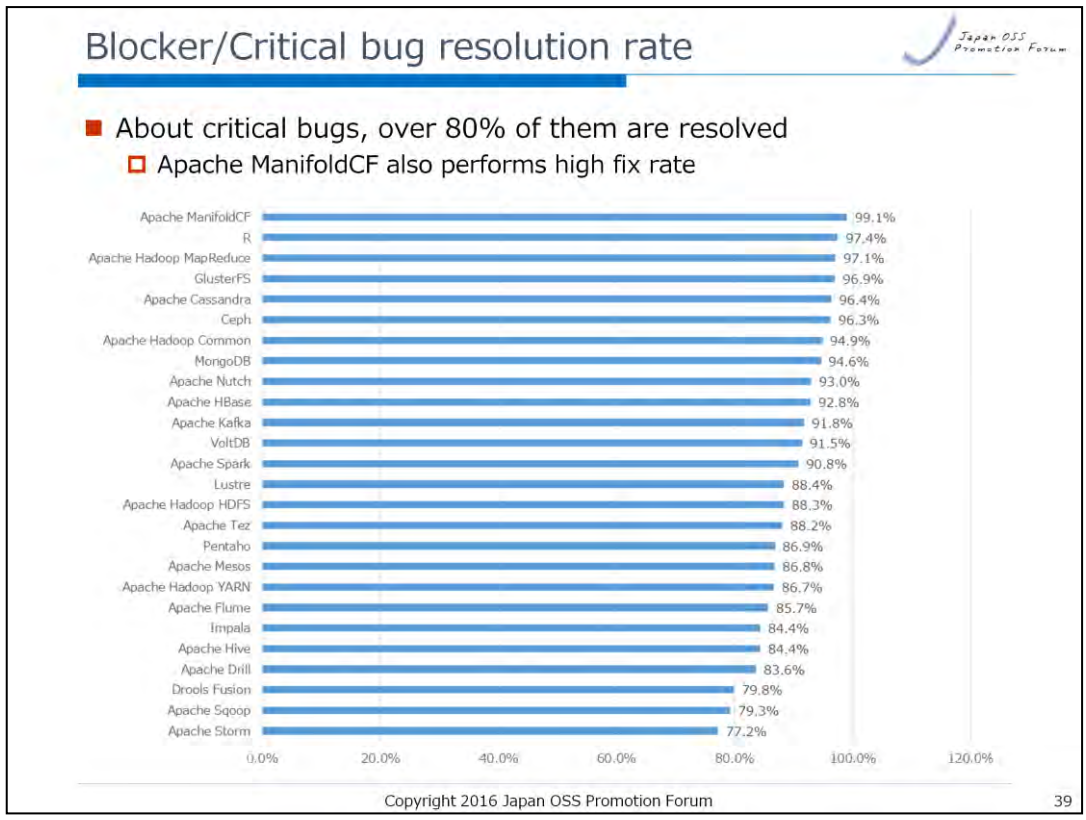
これは、LinuxFoundationの日本でのWGである「SI　Forum」が
毎年実施している2015年の活用動向調査の結果です。
全体的に拡大傾向ではありますが、本格的な導入はこれから
という印象です。



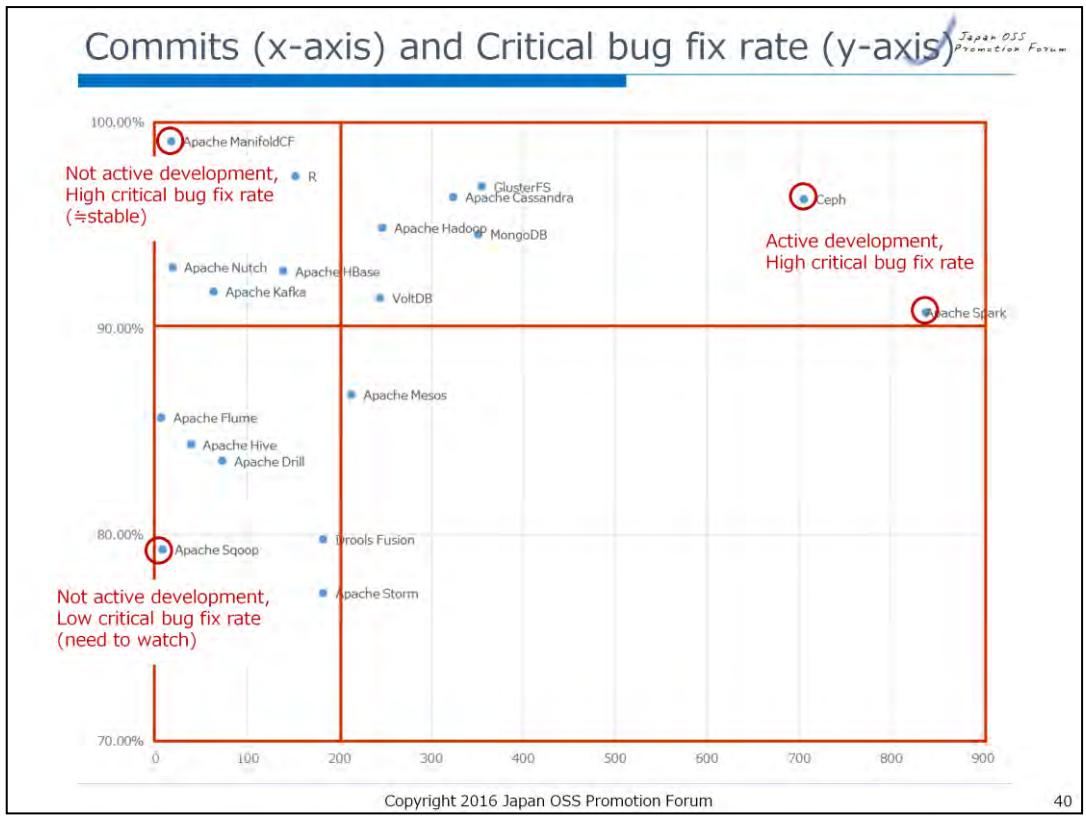
次にソフトウェアの品質について、調査しましたので、ご紹介します。



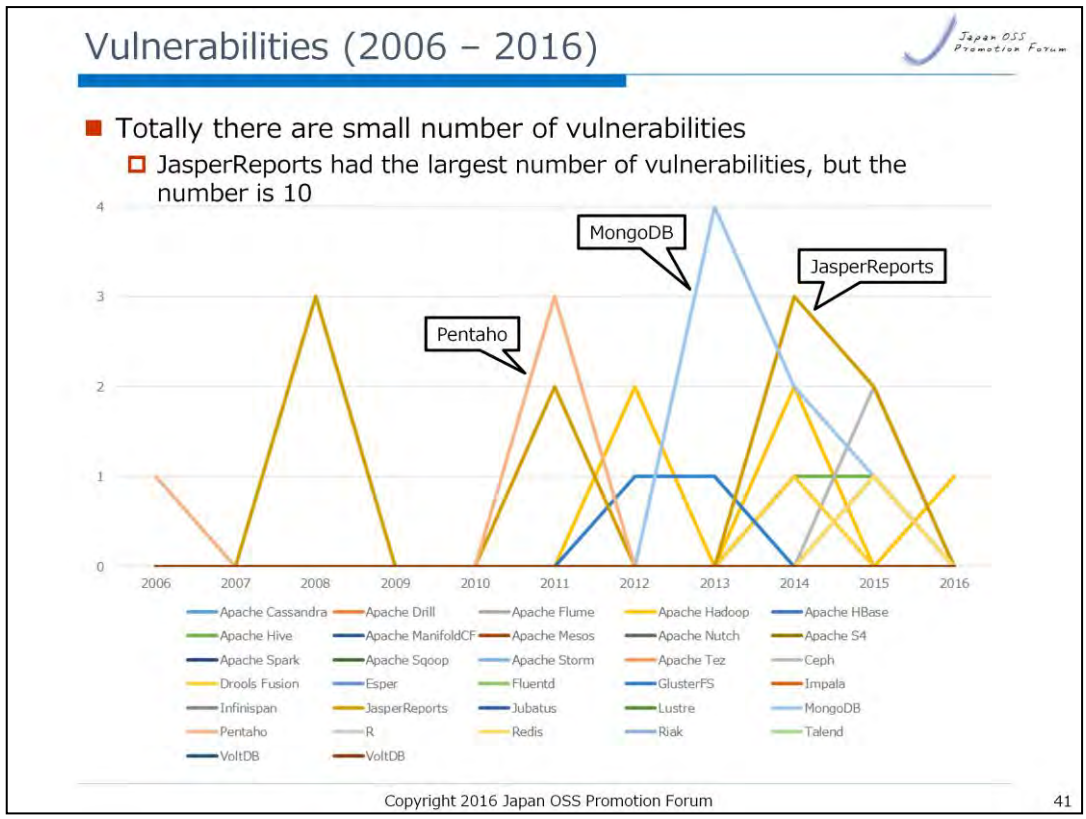
このグラフは、バグの解決率のグラフです。
Apache ManifoldCF, GlusterFS や R などで解決率が高いという結果がでました。
また、最低のStormでも、60%ということで、全体的に解決率が高いと
いえることができます。



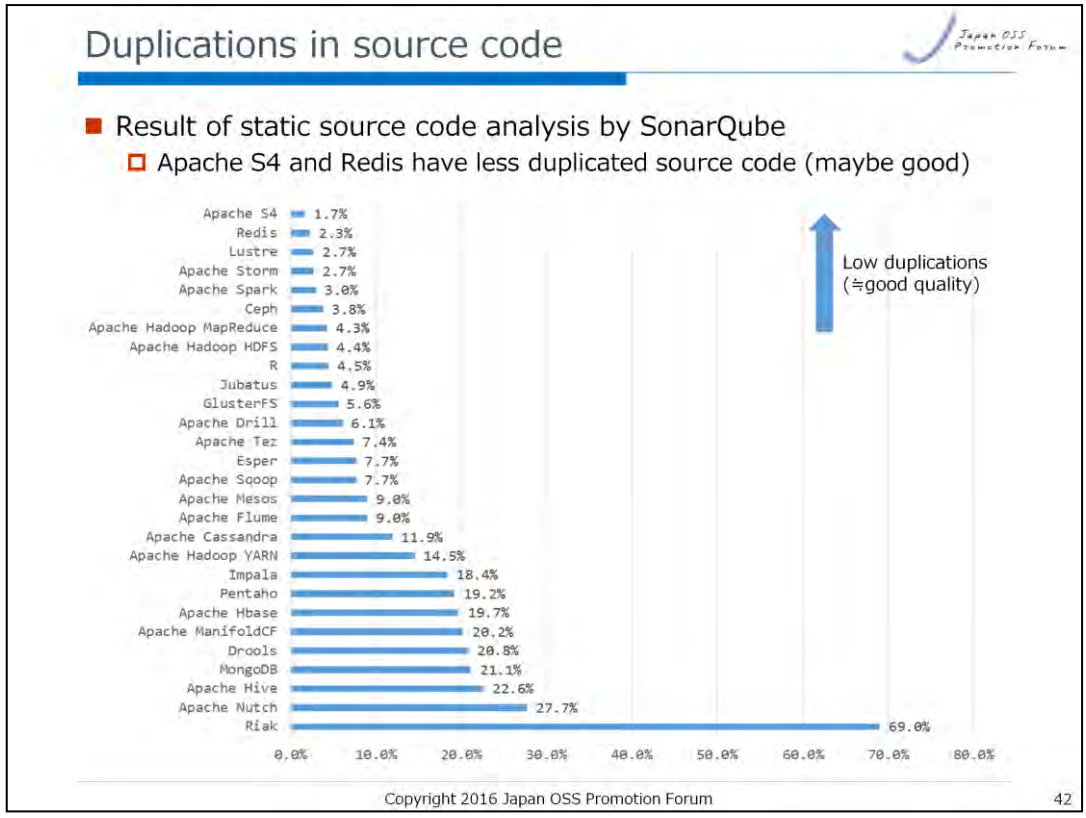
次は、重要なバグの解決率についてです。
重要なバグの解決率に限定すると解決率は80%を超えています。
特に、Apache ManifoldCF の解決率が非常に高くなっています。



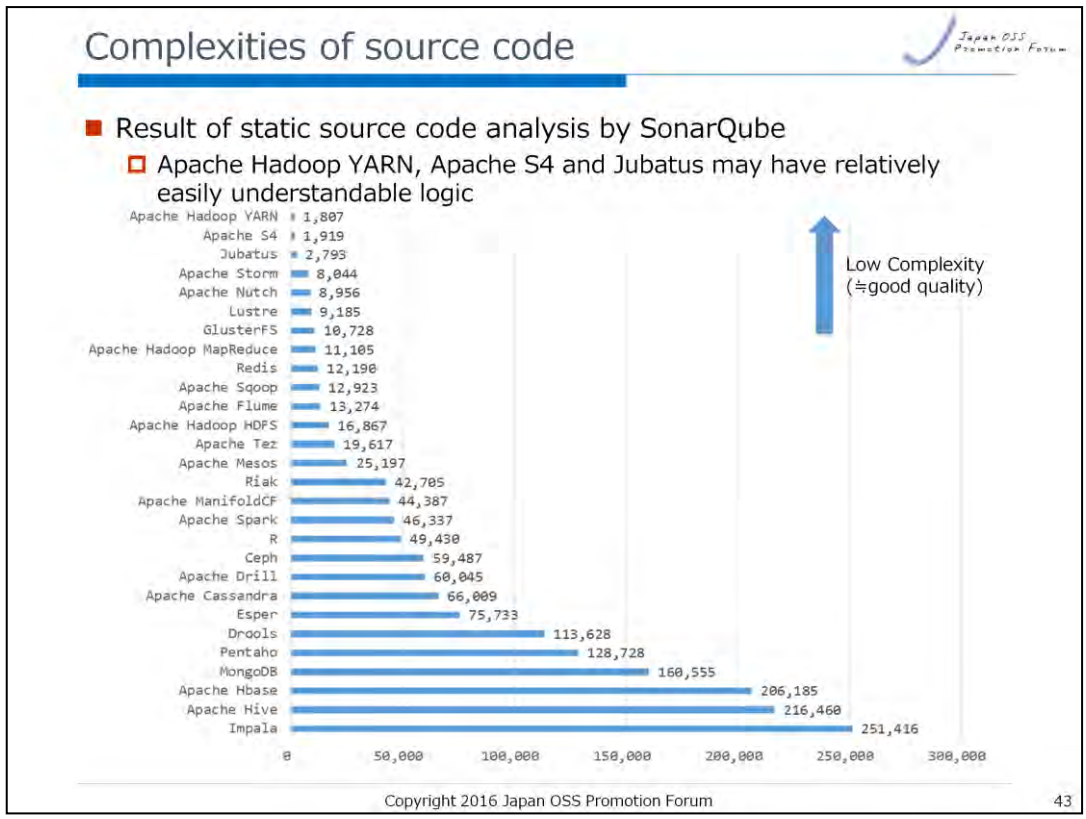
コミット数(横軸)と重要バグ解決率(縦軸)の相関関係です。
左上のApache ManifoldCF は、開発は活発ではありませんが、
重要バグの解決率は高めで、安定しているように見えます。
左下のSqoopは、重要バグの解決率も低く、開発もあまり活発では
ないようなので、利用する場合は、注意が必要かも知れません。
また、右上のCephやSparkは、開発も活発で、重要バグの解決率も
高いので、利用する場合はそれほど心配しなくても良いかも知れません。



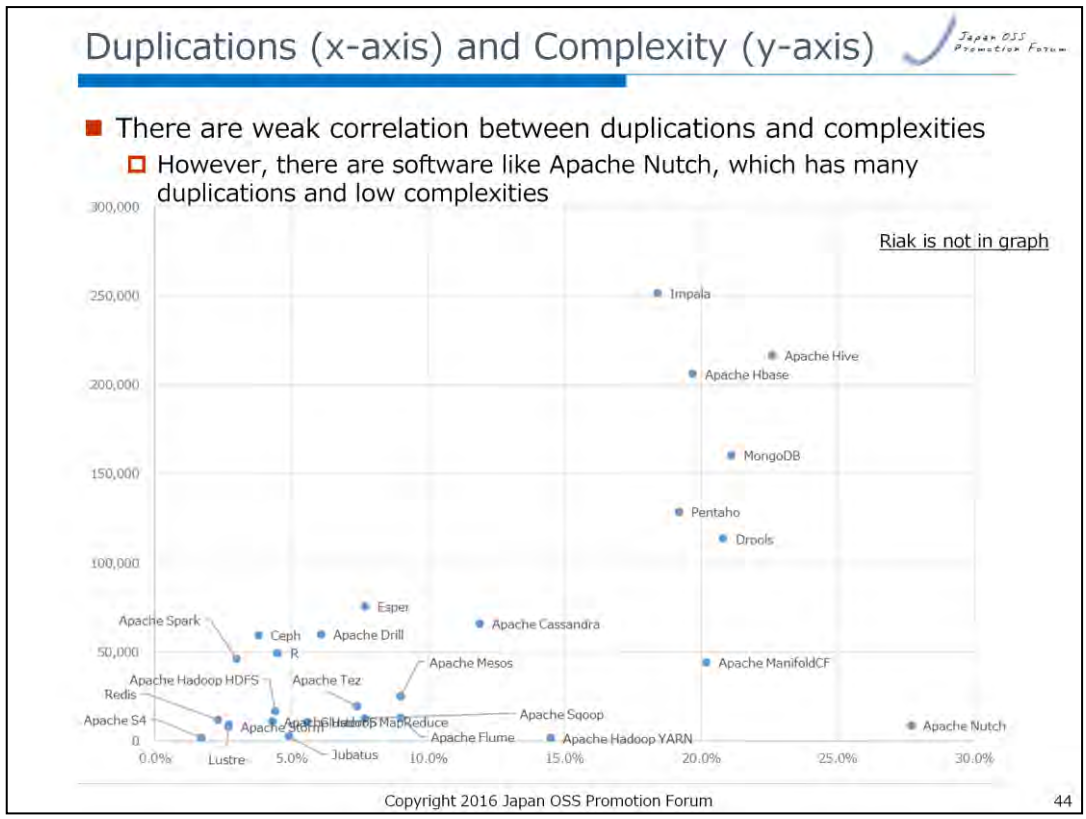
これは、脆弱性の発生状況を表しています。
全体的に脆弱性の検出は少なく、セキュリティでの問題は
あまり心配しなくても良いようです。
一番多いJasperReportでも全体で10件くらいです。



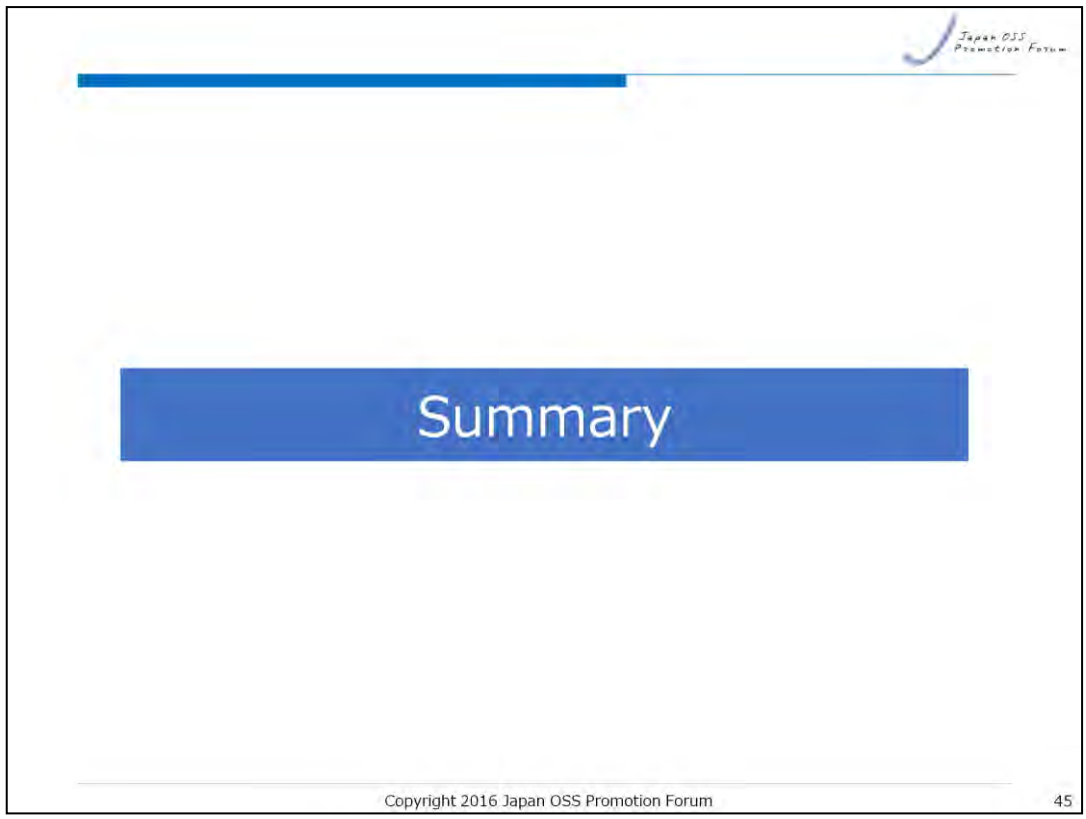
これは、SonarQubeというソフトウェアのソースコード解析機能を
使って抽出したソースコードの重複率です。
ソースコードの重複率が少ないということは、品質が良いということなので、
このグラフでは、上にあるソフトウェアほど、品質が良いということになります。



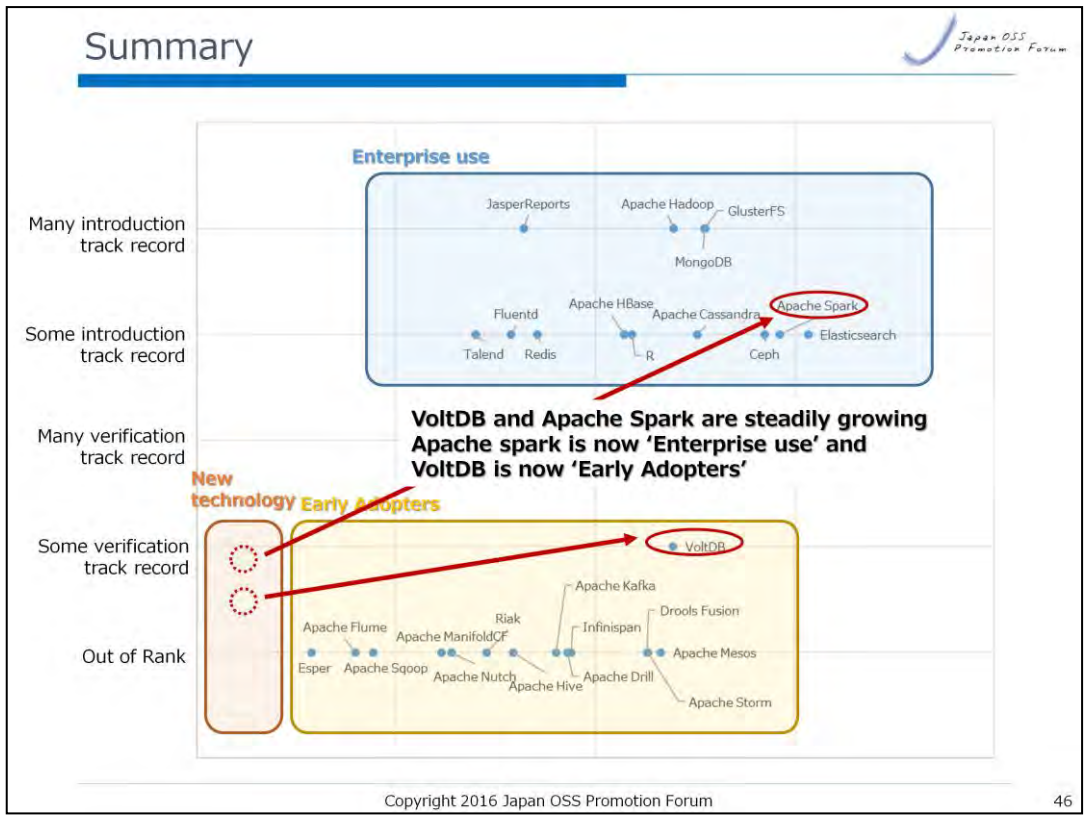
同じくこれは、ソースコード全体の複雑度を表したグラフです。
ソースコードの複雑度が低いほど、高品質であるといえます。



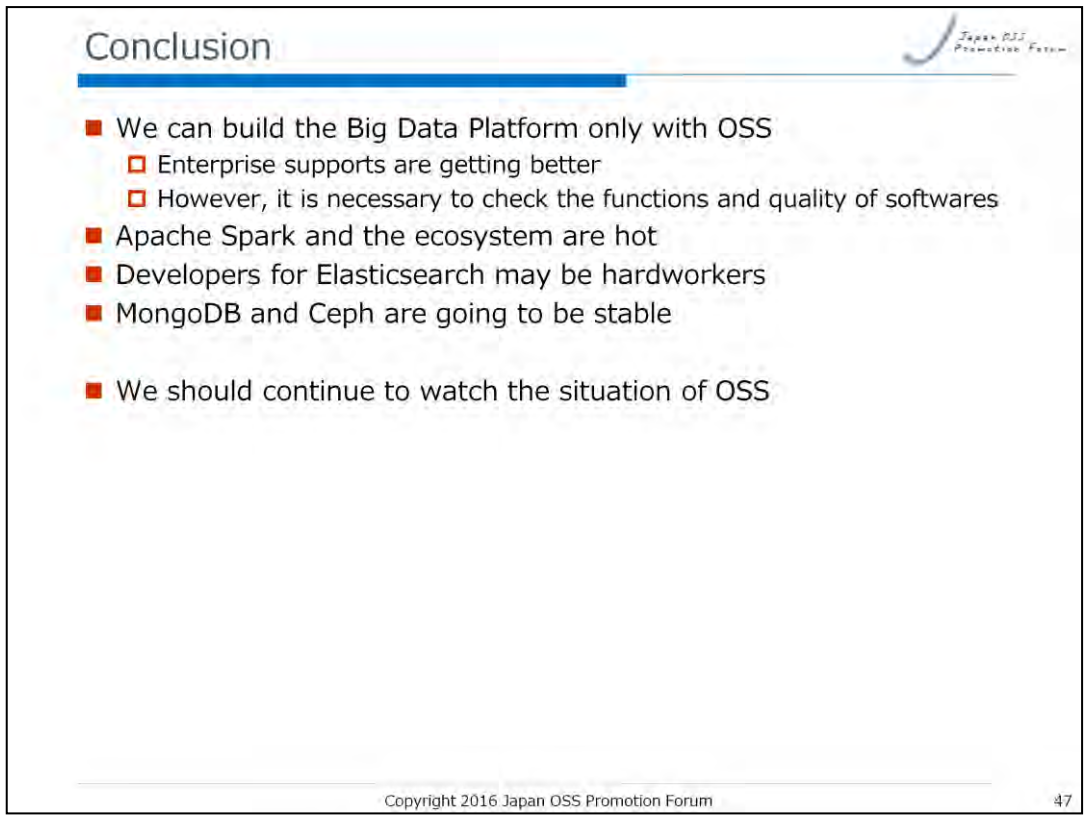
このグラフは、重複率(横軸)と複雑度(縦軸)の相関関係を表したものです。
何らかの関係があるように見えますが、それほど強い関係があるとは
いえない感じです。



全体とまとめます。



昨年から今年への変化した部分で見るとVoltDBやSparkが
一気に伸びてきています。
特にSparkは、エンタープライズでもすでに事例が出ています。



ビッグデータ基盤をオープンソースだけで構築することが充分に可能となってきています。

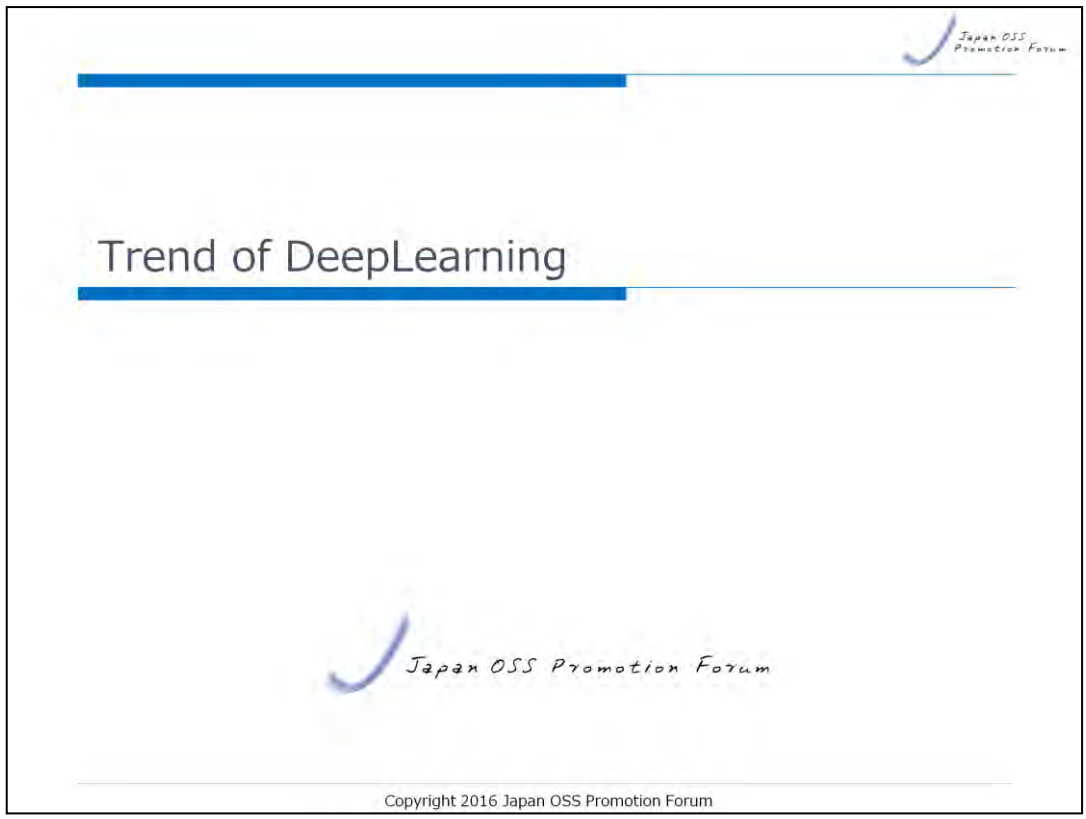
また、有償サービスも揃いつつあります。ただし、機能や品質の見極めは当然必要です。

・Apache Sparkとそのエコシステムは開発、利用ともに活発度が高い

・ElasticsearchはGitコミット日の割合が91%程度でとても開発が活発

・MongoDBやCephはこの領域では比較的安定期に入りつつある

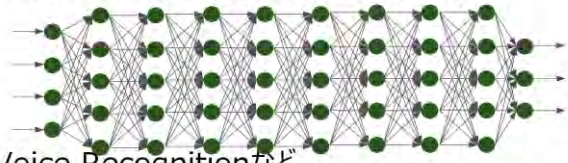
→ただし、状況は変化しており今後も継続的なウォッチが必要です。



次にDeepLearningに動向についてご紹介します。

What is DeepLearning?

- **Deep Learning** is a type of Machine Learning technique.
  - Deep layer, width is to use a broad neural network



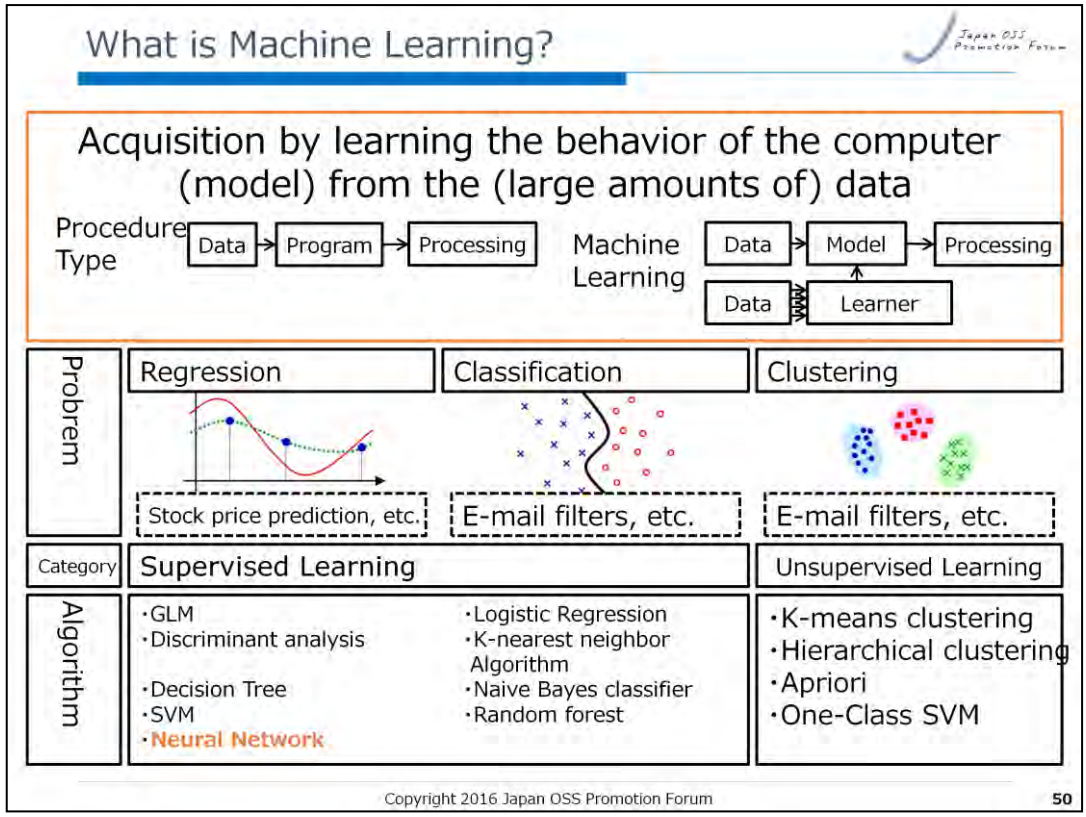
- **What Can do?**
  - Image Recognition , Voice Recognitionなど
    - There are various types of neural network, the area of application is different
    - Higher accuracy than the system using the conventional Machine Learning

Name		Application Areas
Convolutional Neural Network	CNN	Image Recognition
Recurrent Neural Network	RNN	Voice Recognition, Video Recognition, NLP
Long Short-Term Memory	LSTM	Voice Recognition, Video Recognition, NLP
Deep Belief Network	DBN	Attribute Classification
Deep Boltzmann Machine	DBM	Attribute Classification
Deep Auto Encoder	DAE	Feature Extraction
Restricted Boltzmann Machine	RBM	Feature Extraction,Attribute Classification,NLP

(*)NLP : Natural Language Processing

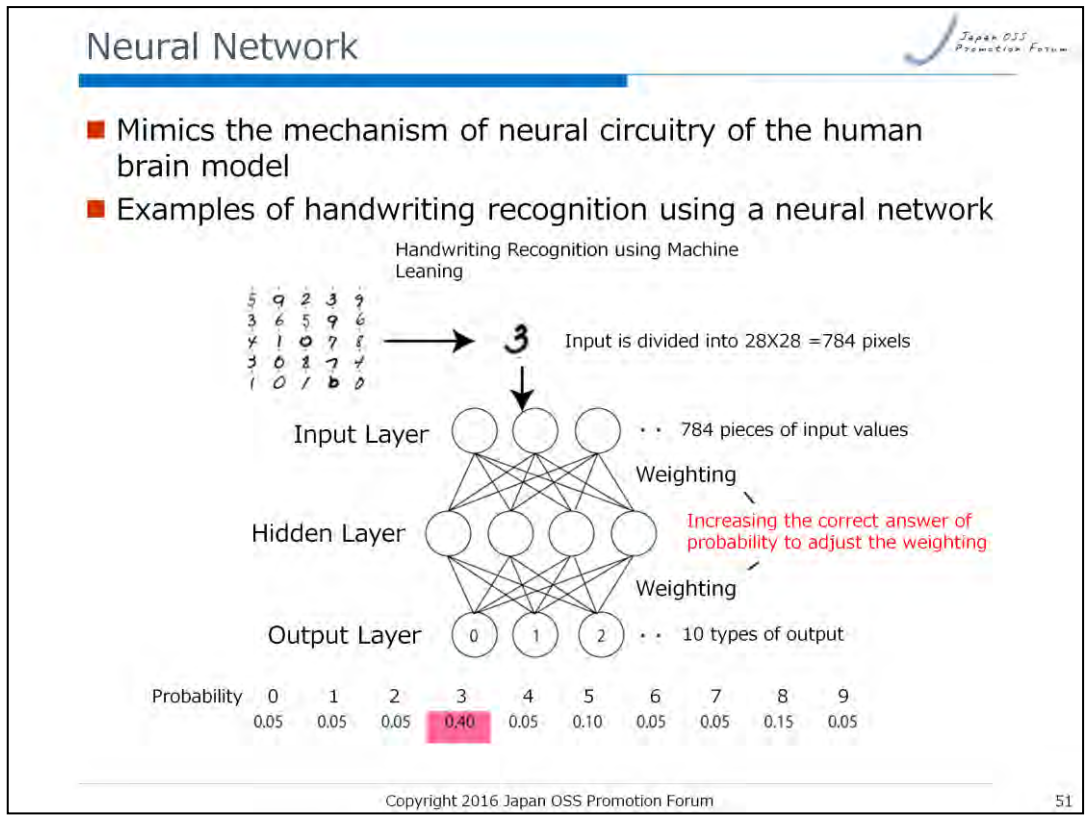
まず、DeepLearningとは何かいうことを説明します。
DeepLeaningは、機械学習の手法の一種で、
　多層に重ねたニューラルネットワークを利用しています。

何ができるかというと、現在一番活用されている部分は、
画像認識や音声認識になります。
とはいえ、さまざまな種類のニューラルネットが存在するので、適用領域が異なります。
また、従来の機械学習より、精度が高くなっています。



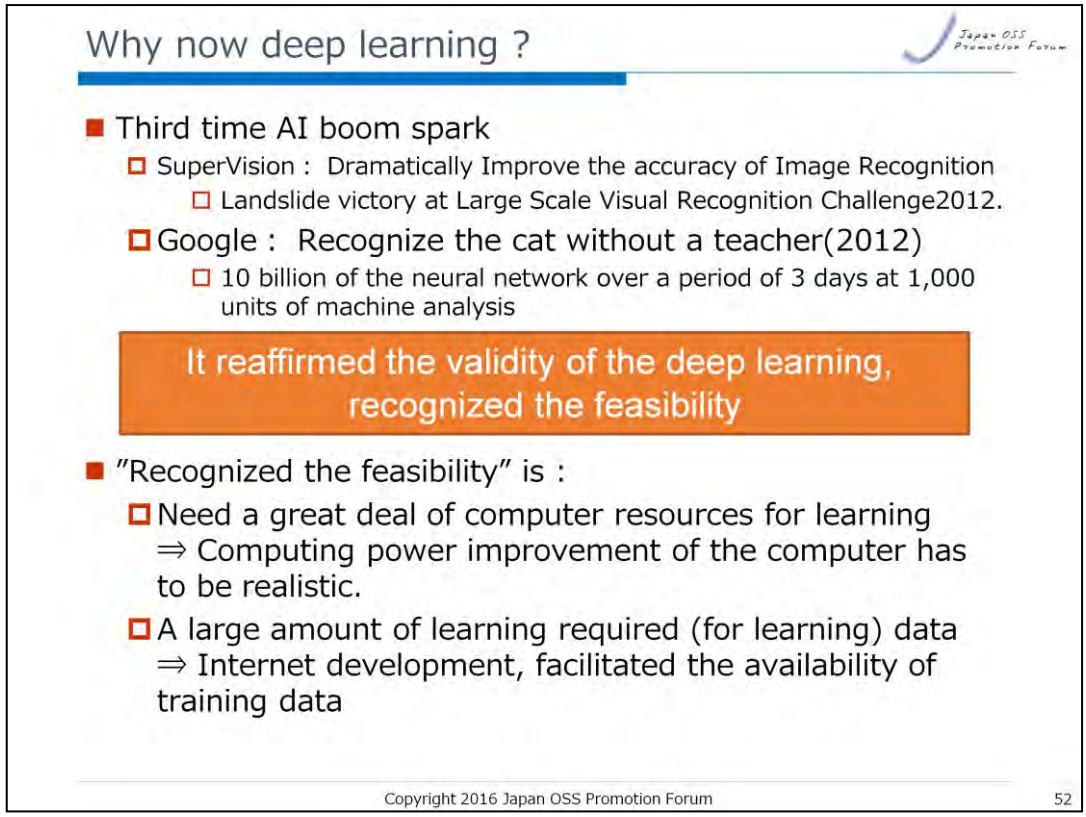
次に機械学習について、説明します。
機械学習とは、コンピュータの振舞い方を大量のデータからコンピュータ自身に学習させることです。
解決できる問題は、株価予測や迷惑フィルタなどになります。

GLM：線形モデル



次にニューラルネットワークです。
これは、人間の脳の神経回路を模したモデルです。

手書き文字の認識の例を示します。
このように784ピクセルに分割してデータを入力し、それぞれのデータに
重み付けをし、その重み付けを調整することで、正解にたどり着くようにします。
この例の場合では、「3」の確率が一番高くなっています。



なぜ、ディープラーニングかというお話をします。

現在、第三次人工知能ブームといわれています。

最初は、1956年のダートマス会議で初めて「人工知能」という

用語が使われたころです。このころは、数学の定理を証明するのが、

関の山でした。

次は、1980年代で、そのころは専門家の思考過程をシミュレーション

するような「エキスパートシステム」が流行りました。

今回は、以下の二つの事例が大きく影響しています。

一つ目は、2012年に画像認識コンテストで、従来の方法に比べて2倍の精度で認識できたことです。

二つ目は、Googleが1000台のコンピュータを三日間かけて学習した結果、

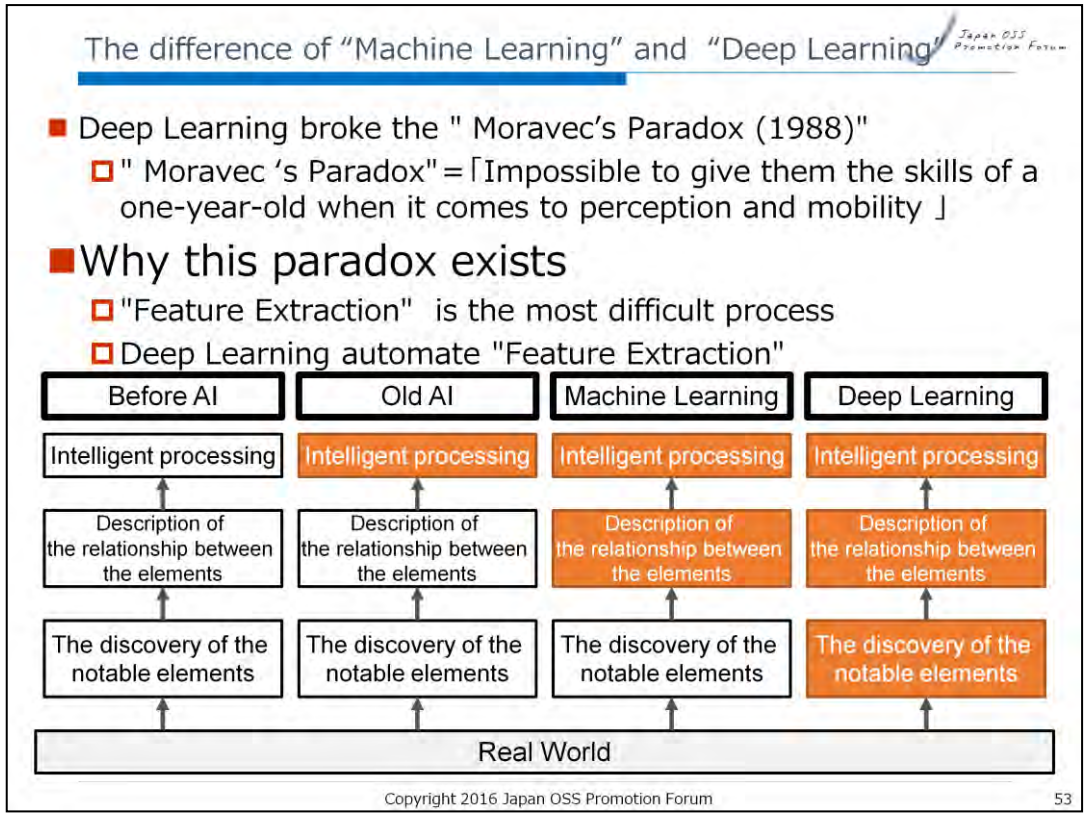
猫の画像を識別できるようになったことです。

このような事象が大きく影響し、ディープラーニングが有効であることが再認識されました。

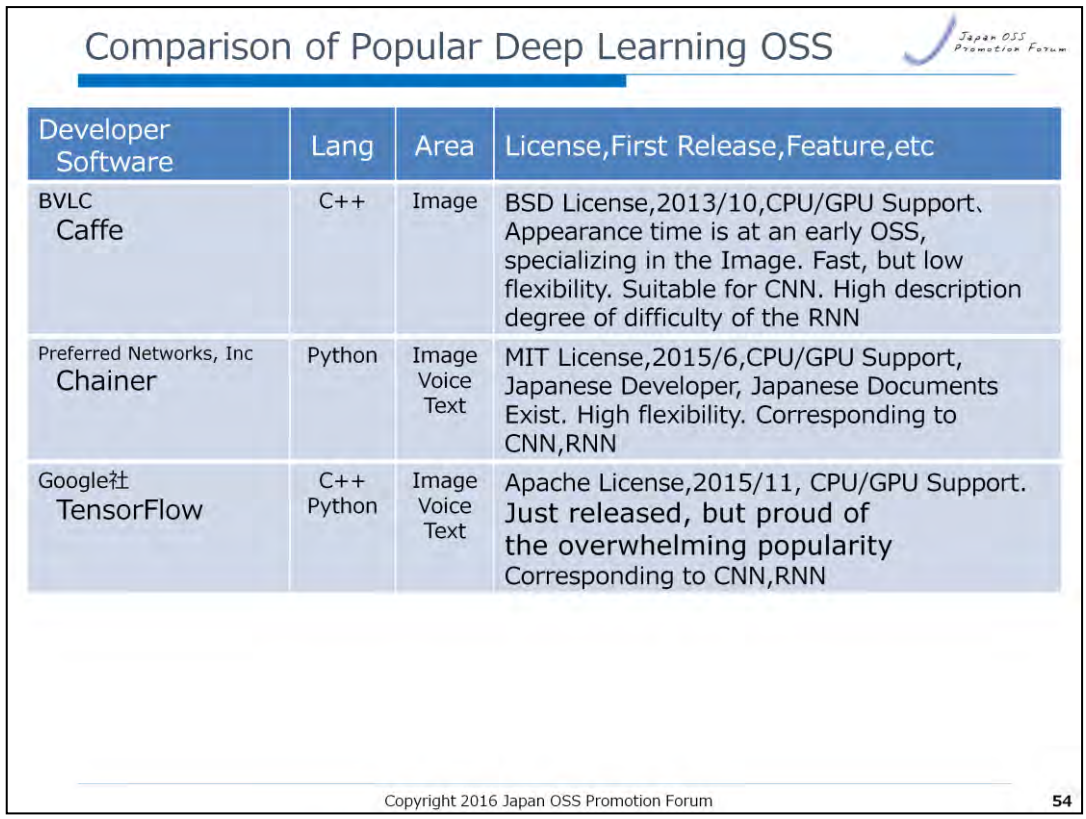
しかしながら、コンピュータの計算能力が大幅に向上したことと

インターネットの発展によって、大量の学習データが入手できるようになったことも

大きな要因となっています。



機械学習とディープラーニングの違いについて、説明します。
ディープラーニングは、子供のできることの方が難しいというモラベックのパラドックスを崩したと言われています。
人工知能には、「大人の人工知能」と「子供の人工知能」があると言われています。要素間の関係の記述については、データを分析することで認知できますが、注目すべき要素の発見は、データだけでは、獲得することができません。言葉をどのように獲得するかというような言葉で説明できない要素の発見を自らできるようになってきています。また、その実現スピードも速くなってきています。



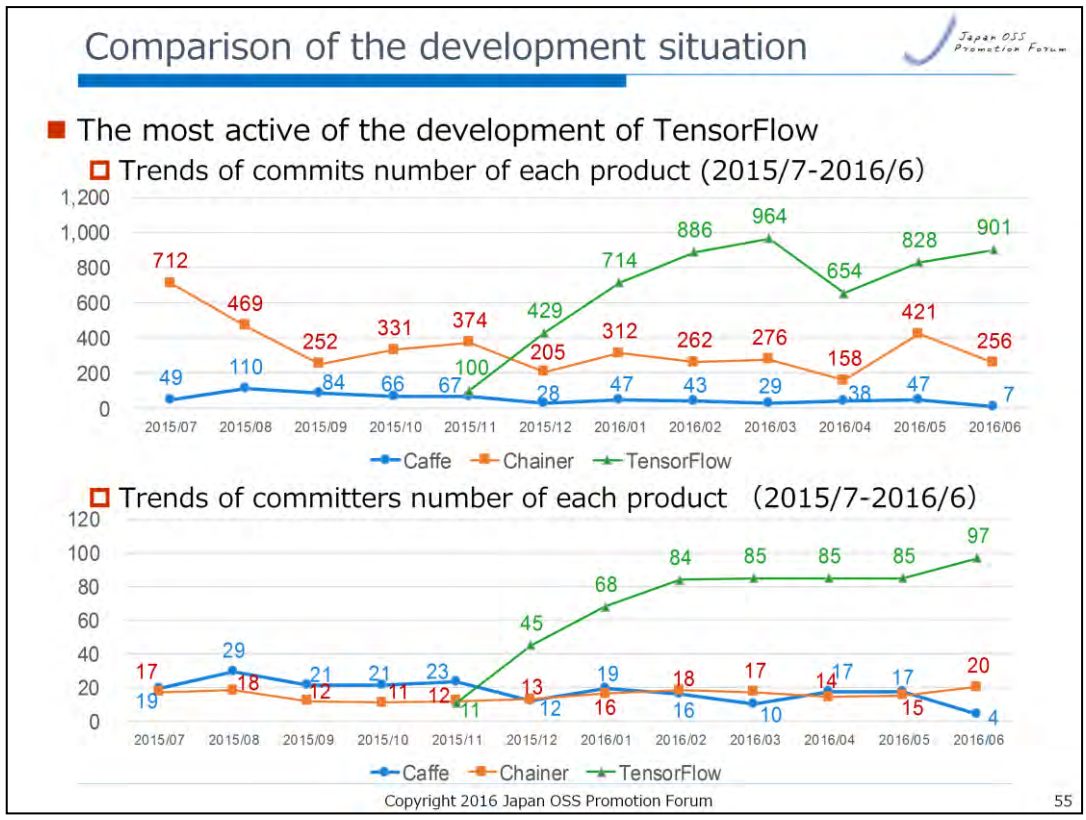
次にディープラーニングを実現する代表的なオープンソースのフレームワークを説明します。

一つ目は、「Caffe」です。これは、米国のバークレー大学の研究所で開発されたもので、2013年の10月という早い時期に登場したものです。画像に特化しm高速ですが、柔軟性がひくいという特徴があります。

二つ目は、「Chainer」です。これは、日本のプリファードネットワークという会社が開発したものです。日本で開発されたこともあって、日本語のドキュメントも多く日本人としては、とっつきやすいものになっています。

三つ目は、「TensorFlow」です。これは、Gogglが開発したもので、2015/11にOSS化されたということで、一番新しいのですが、圧倒的な人気を誇っています。

注目すべきは、これら三つともBSD系のライセンスを採用していることです。すなわち、自社製品の中に取り込めるという特徴があります。これらフレームワークの普及のカギは、そのあたりにあるかもしれません。

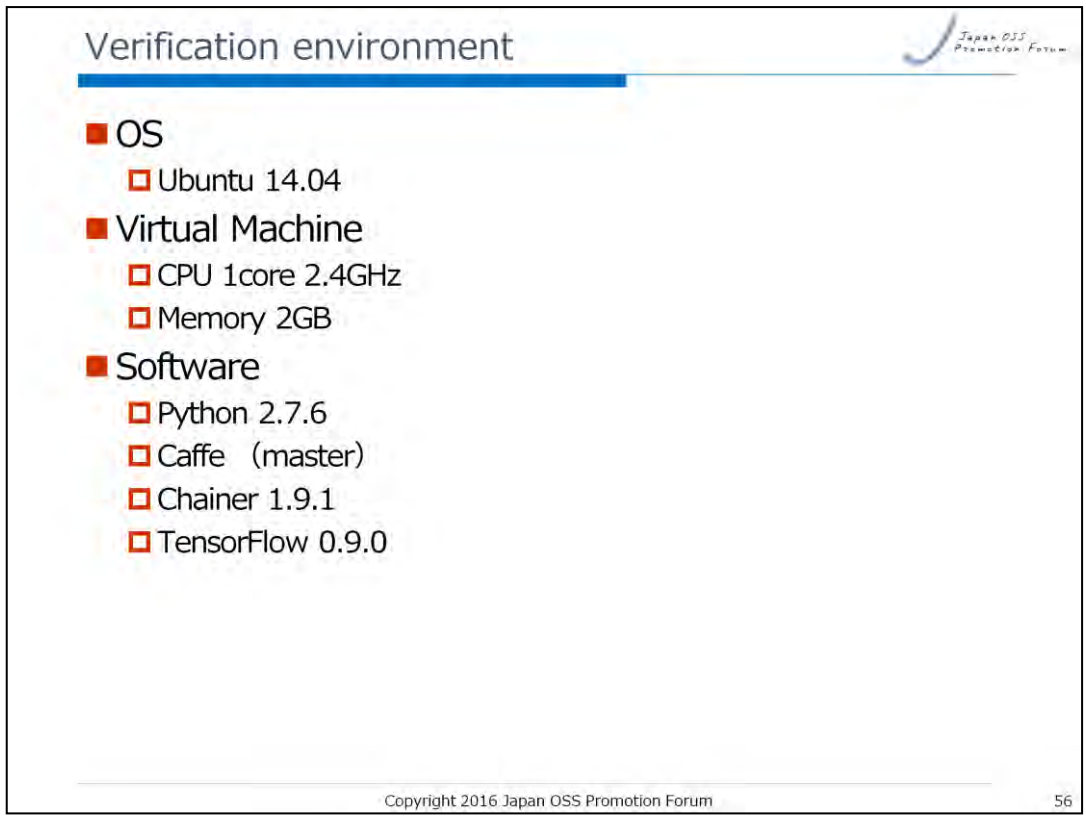


開発状況の比較をしてみましょう。

各製品のコミット数の推移を見てみると、TensorFlowのコミット数が非常に
伸びていることがわかります。

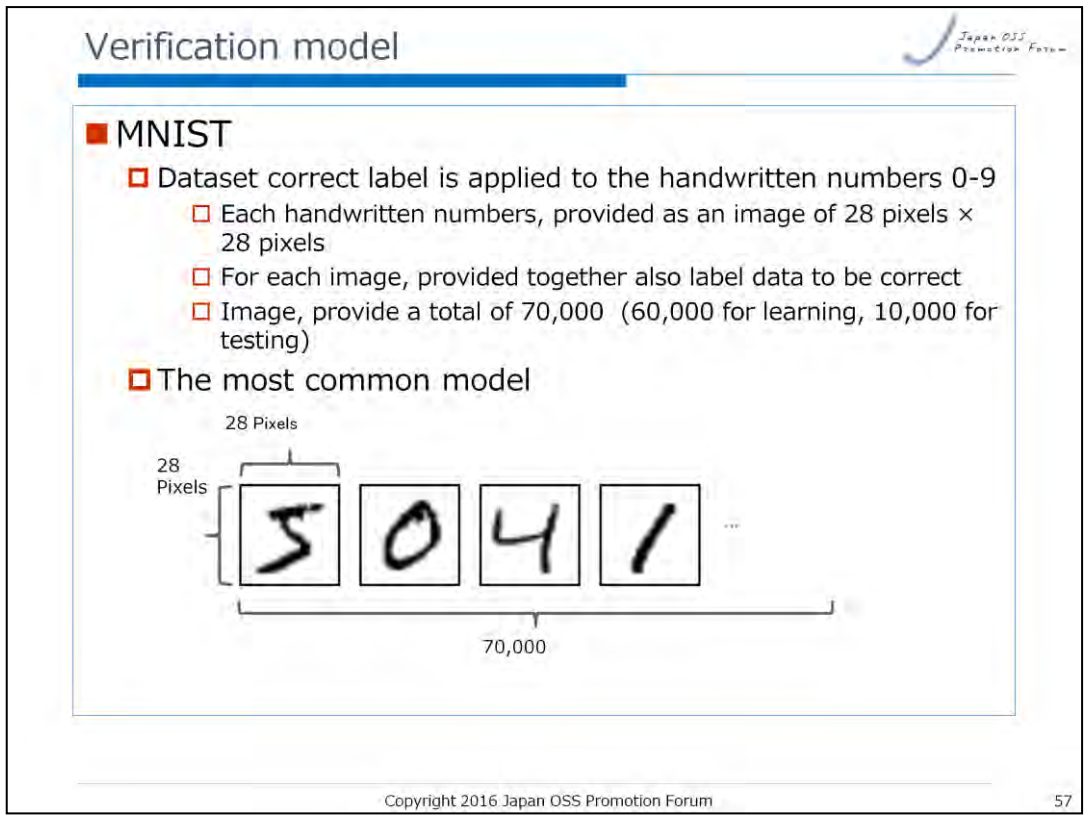
また、コミッタ数の推移をみてもTensorFlowが圧倒的に伸びています。

このような状況を見るとTensorFlow開発が最も活性化していると
言ってよいを思います。



次に検証環境をご紹介します。

皆さんがお持ちのPCのような環境です。

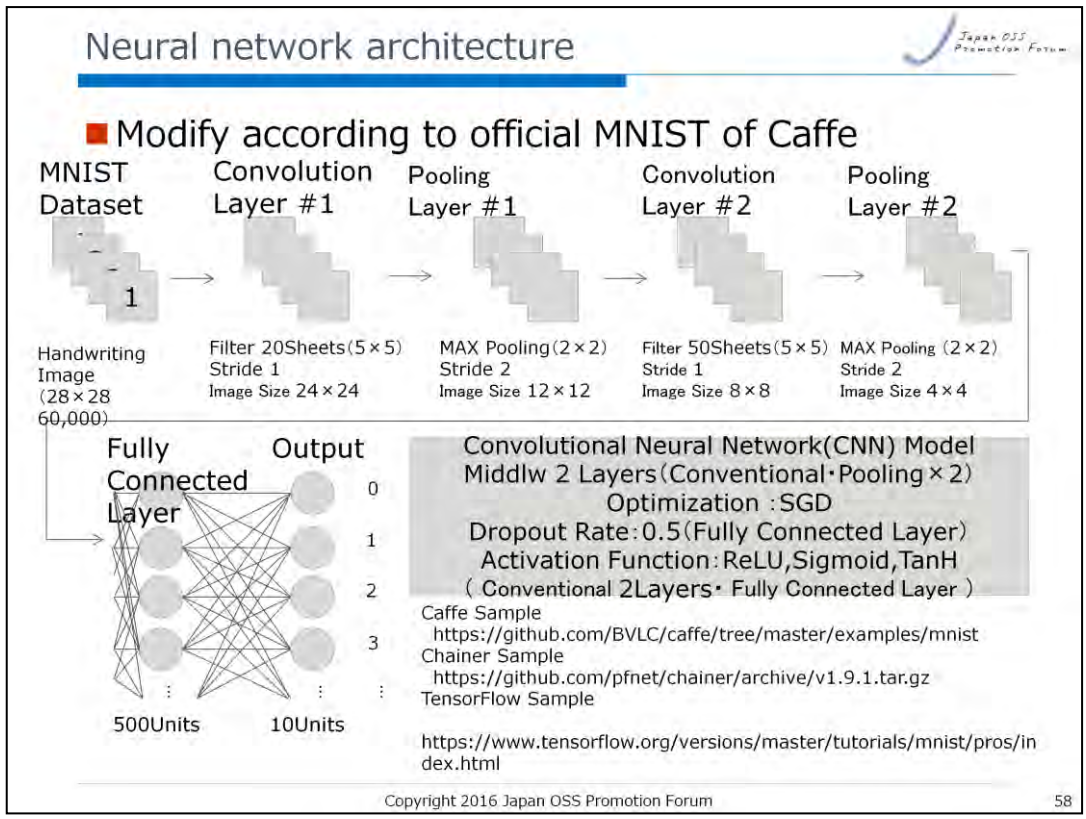


検証モデルについて、説明します。

これは、MNISTから提供されているデータを採用します。
手書きの数字0から9に正解ラベルが与えられているデータで、
28×28ピクセルの画像で提供されています。
各画像には、正解となるデータも提供されています。

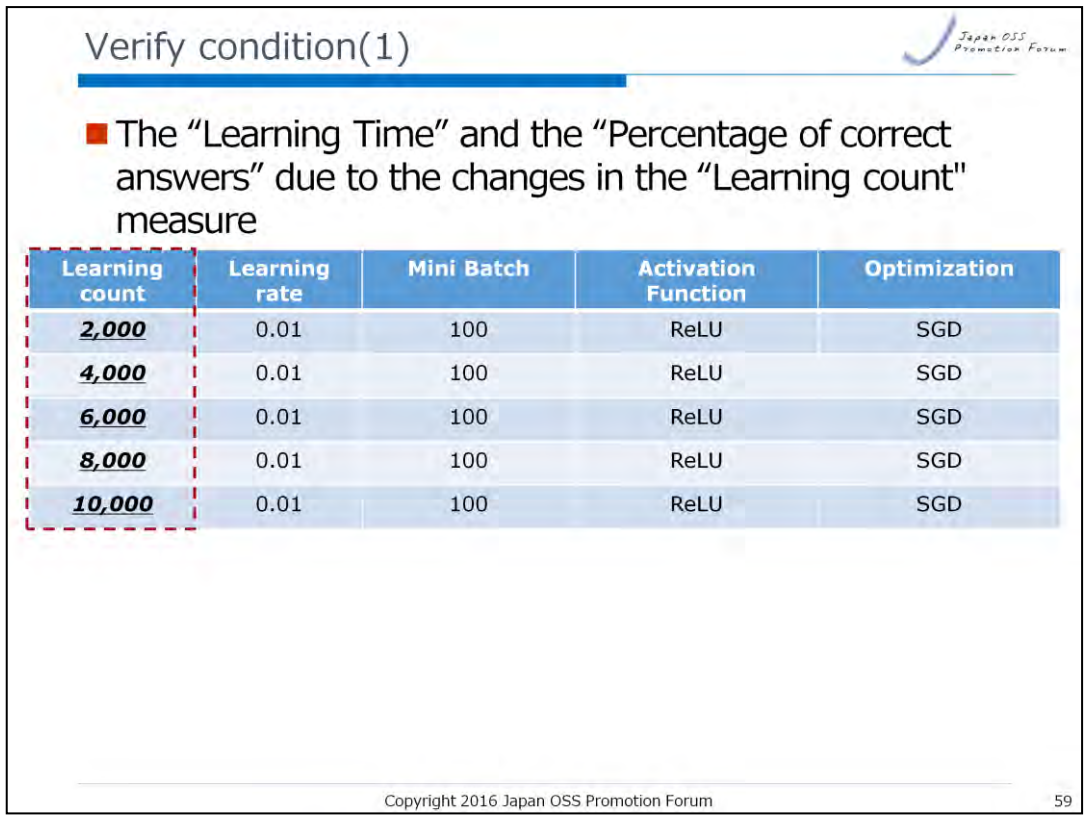
画像は、学習用60,000枚、テスト用10,000枚の70,000枚を提供されています。

このモデルは、機械学習やディープラーニングの世界で
最も一般的なモデルで、いわゆる「Hello World」のようなものです。

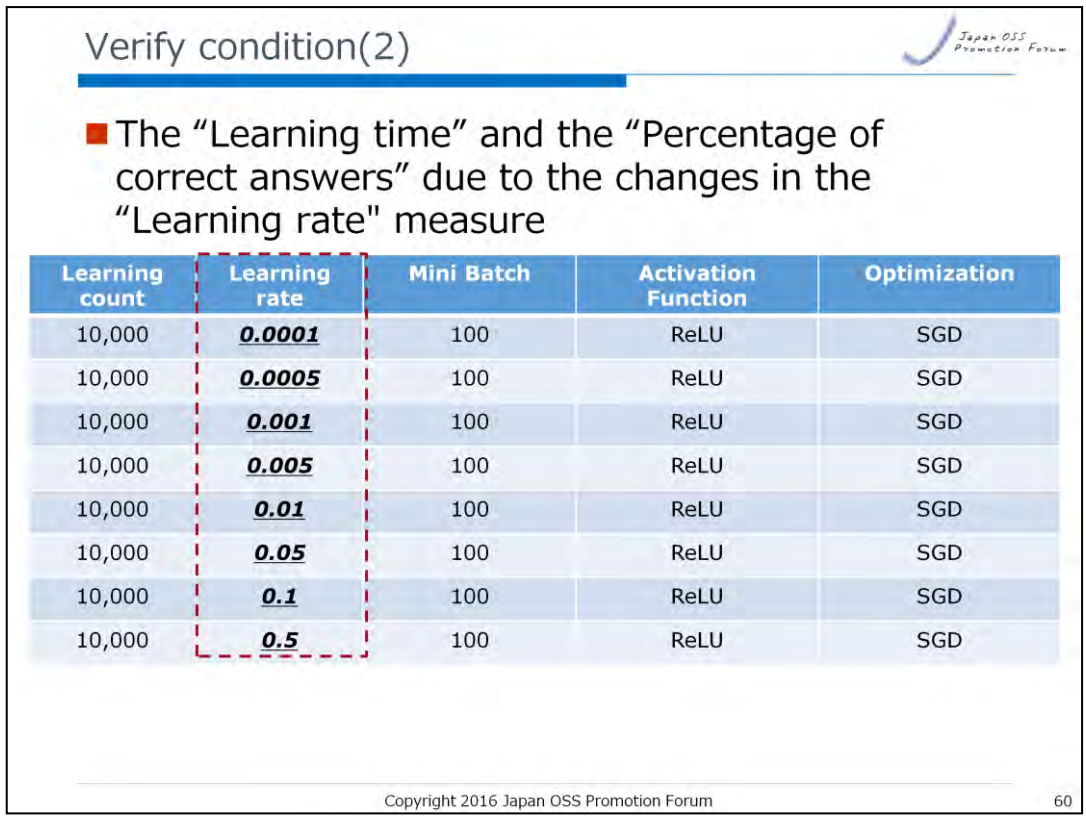


CaffeのMNISTを使用したサンプルにほかのフレームワークも合わせた形で、
検証のパターンを作成しました。

CNNと呼ばれる畳みこみニューラルネットワークモデルを採用しています。
中間は2層で、畳み込みとプーリングをそれぞれ2層）
最適化手法は、SGD関数
ドロップアウト率は、0.5（全結合層）
活性化関数は、ReLU,Sigmoid,TanH（畳み込み2層・全結合層
を使用しています、



検証条件の一つ目は、
学習回数の変化に伴う学習時間及び正答率の変化です。



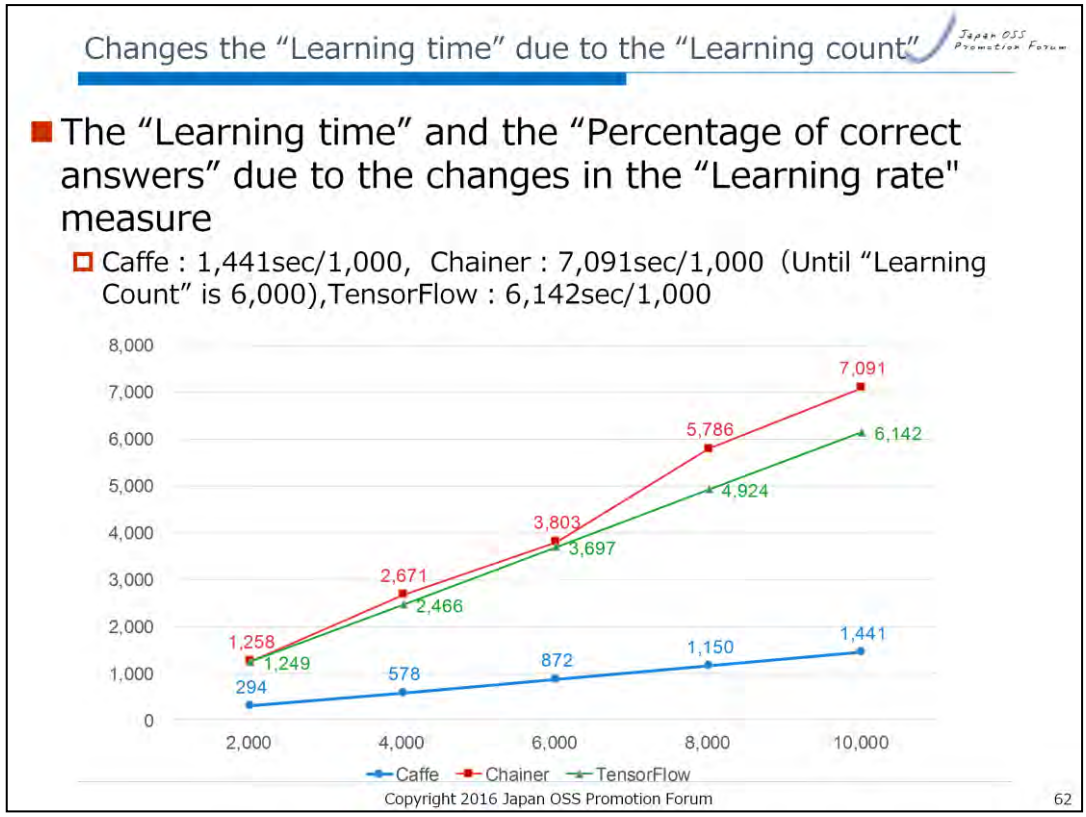
二つ目は、
学習率の変化に伴う、学習時間及び正答率の変化です。

Verify condition(3)

- The “Learning time” and the “Percentage of correct answers” due to the changes in the “Learning rate" measure

Learning count	Learning rate	Mini Batch	Activation Function	Optimization
2,000	0.01	100	***Sigmoid***	SGD
4,000	0.01	100	***Sigmoid***	SGD
6,000	0.01	100	***Sigmoid***	SGD
8,000	0.01	100	***Sigmoid***	SGD
10,000	0.01	100	***Sigmoid***	SGD
2,000	0.01	100	***TanH***	SGD
4,000	0.01	100	***TanH***	SGD
6,000	0.01	100	***TanH***	SGD
8,000	0.01	100	***TanH***	SGD
10,000	0.01	100	***TanH***	SGD

三つ目は、
活性化関数の種類に伴う、学習時間及び正答率の変化を測定します。



＜事実のみ＞

・Caffeは、各学習回数において、他フレームワークよりも約4.0～4.7倍高速に学習

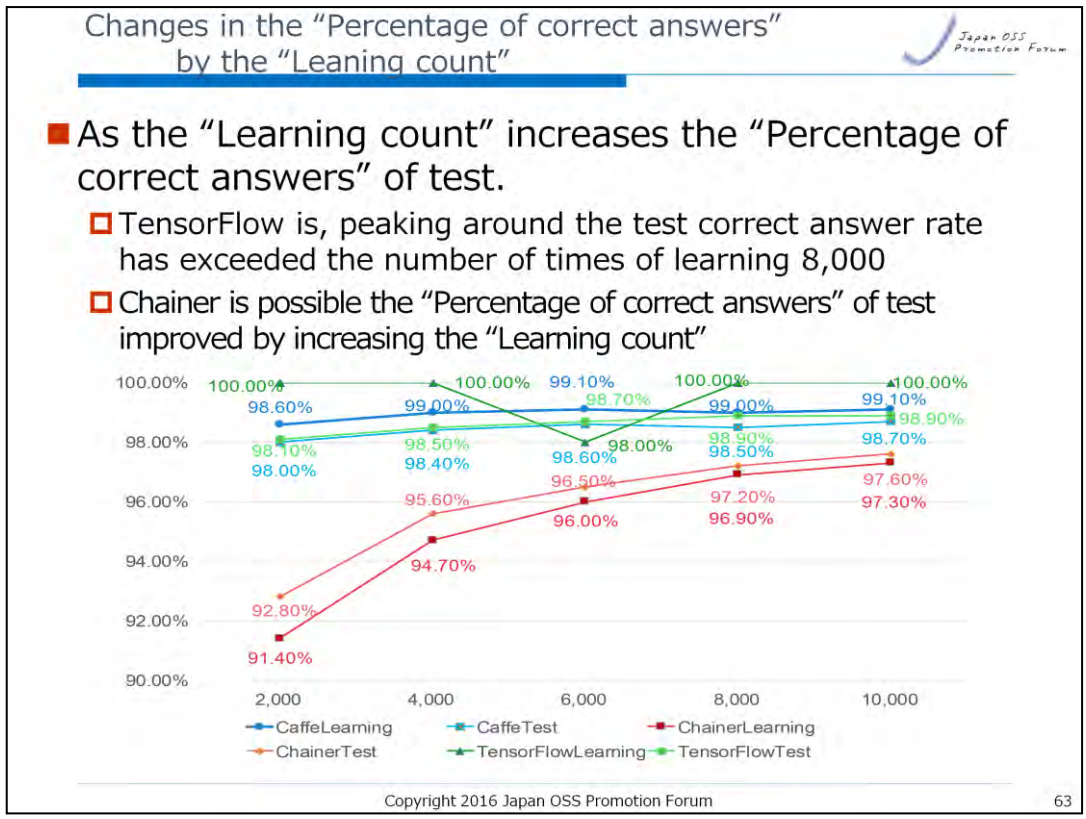
・Caffe、TensorFlowともに学習回数に単純比例、Chainerもほぼ学習回数に単純比例

・Chainer、TensorFlowの学習時間は、学習回数6,000までであればほぼ同じ

・Chainerは、学習回数が6,000より大きくなると、何らかの原因で学習時間が増加

※一般的に学習回数が少なすぎると正答率が低く、多くなると学習時間が大きくなる

※一般的に中間層の数や中間層の要素数が多くなると正答率は大きくなるが、時間がかかる。いずれ頭打ちになる。



＜事実のみ＞

・Caffeは、学習回数が大きくなっても正答率がほぼ横ばい

・Caffeは、各学習回数において、学習時とテスト時の正答率の差が常に小さい

・Chainerは、学習回数が大きくなるにつれ、正答率が順調に増加

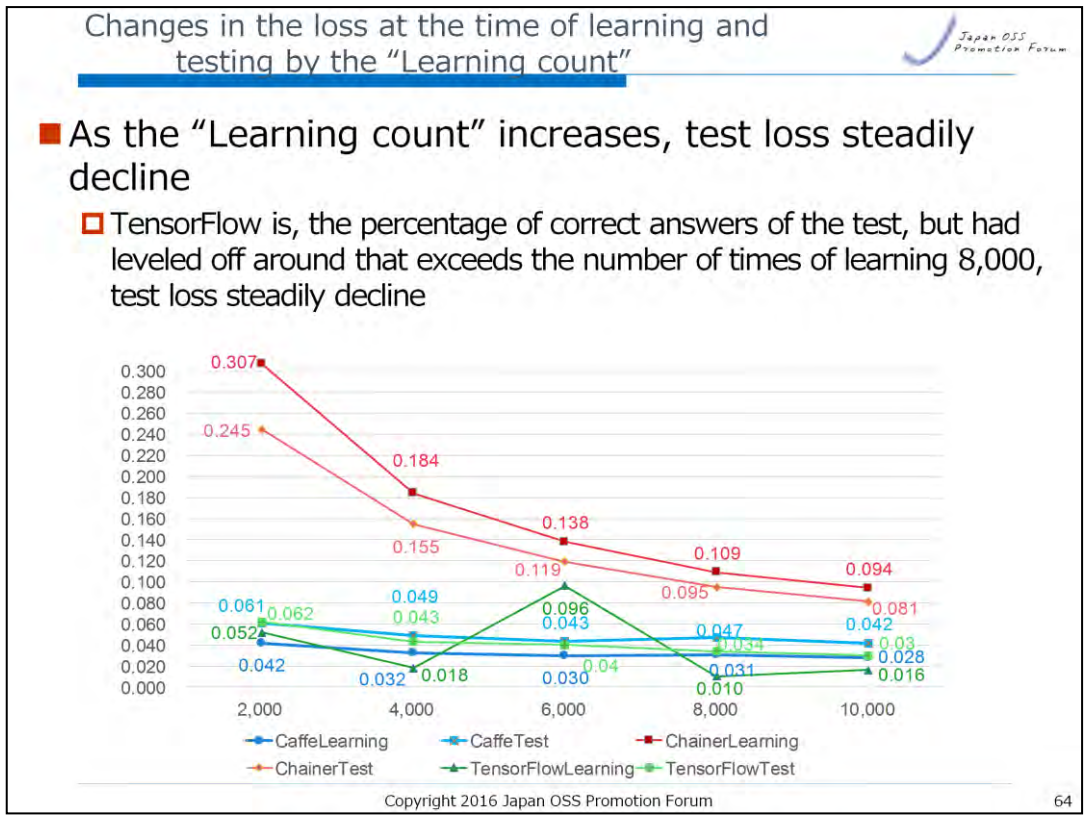
・Chainerは、各学習回数において、他フレームワークよりも正答率が低い

・TensorFlowは、学習回数が大きくなっても正答率がほぼ横ばい

・TensorFlowは、学習回数2,000回で、学習時の正答率100%

・TensorFlowは、全ての学習回数において、他フレームワークより学習時とテスト時の正答率の差が大きい

※一般的に学習回数が少なすぎると正答率が低く、多くなると学習時間が大きくなる



＜事実のみ＞

・Caffeは、学習回数が大きくなるにつれ、損失がゆらぎながら低下

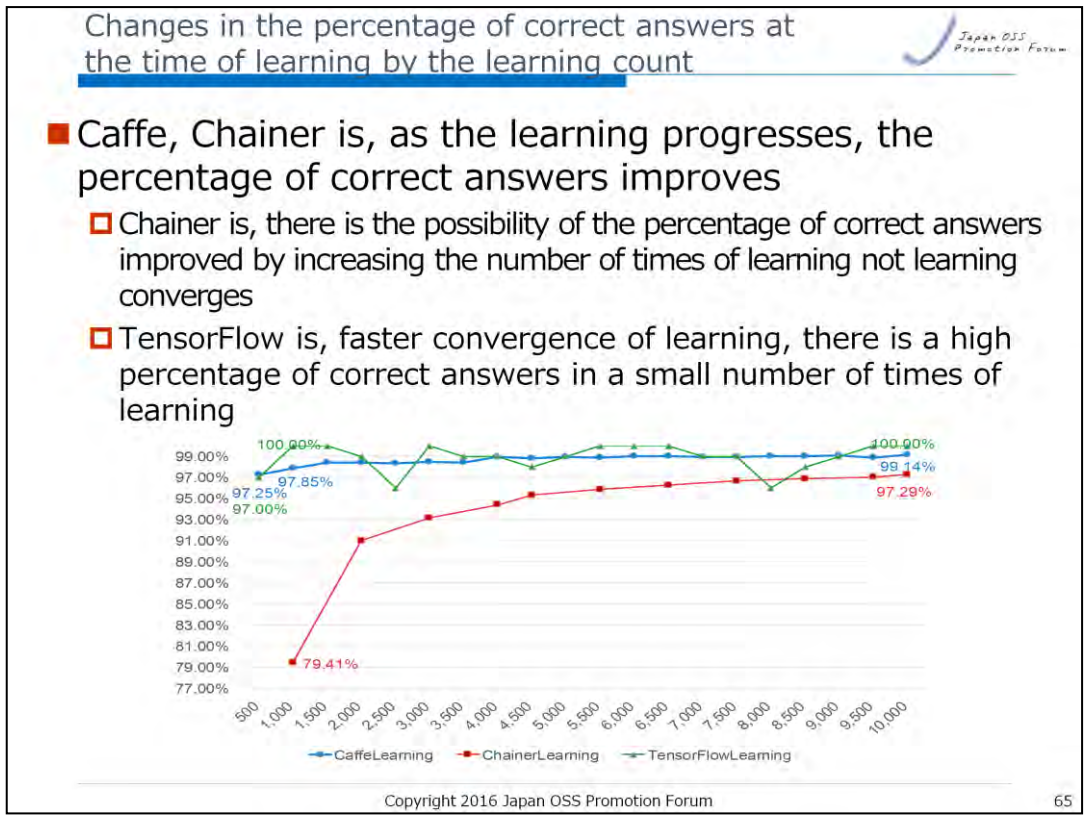
・Chainerは、学習回数が大きくなるにつれ、損失が順調に低下

・Chainerは、学習時・テスト時ともに、損失が他フレームワークに比べ常に大きい

・TensorFlowは、学習時の損失がゆらぎながら低下

・TensorFlowは、テスト時の損失は順調に低下

・TensorFlowは、学習回数4,000、8,000、10,000の場合、学習時の損失が他フレームワークに比べ最も低い

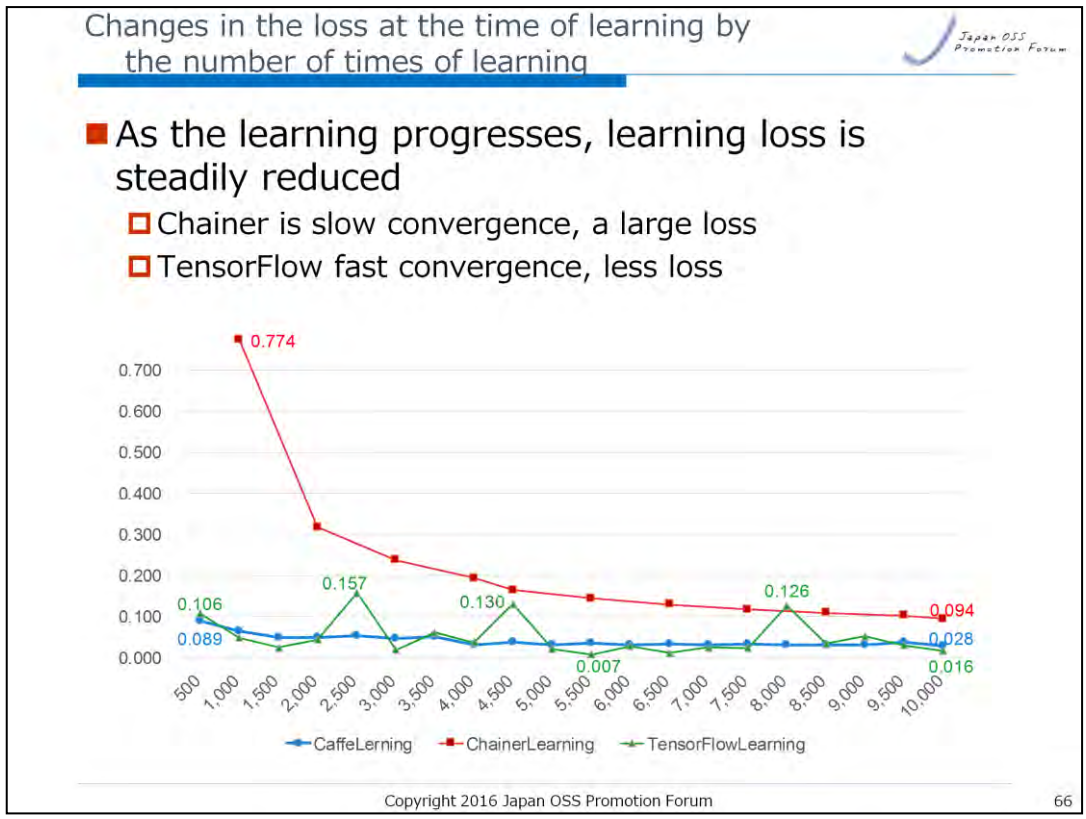


＜事実のみ＞

・Caffeは、他フレームワークに比べ学習回数の増加に伴う正答率が安定

・Chainerは、正答率の伸び率は最も大きいが、正答率は他フレームワークより低い

・TensorFlowは、正答率に揺らぎがあり、安定していないが、他フレームワークに比べ高い正答率（学習回数1,000の時点で既に正答率100%）

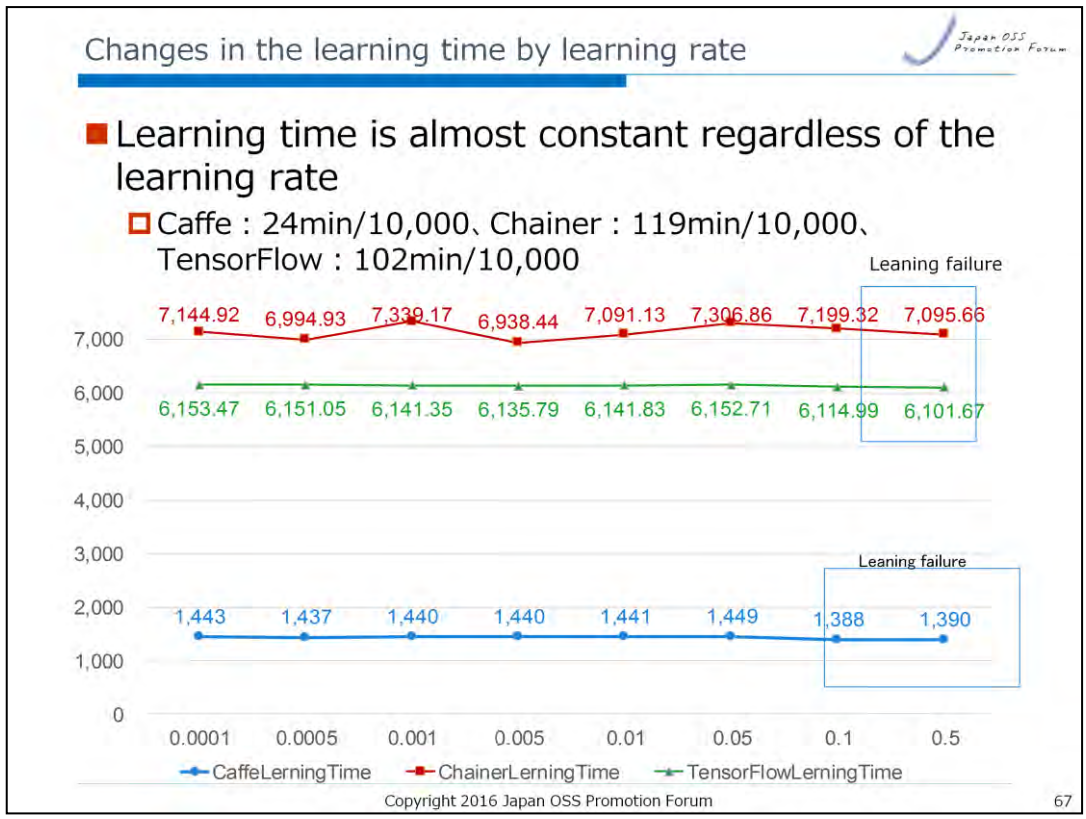


＜事実のみ＞

・Caffeは、多少の揺らぎはあるが、順調に低下

・Chainerは、下げ幅は大きいが、他フレームワークに比べ損失が大きい

・TensorFlowは、揺らぎがあり、安定しないが、他フレームワークに比べ損失が小さい



＜事実のみ＞　※Caffeは、学習率0.1、0.5で学習失敗（発散）したため、学習時間を反映していない。

・Caffe、TensorFlowは、学習率に依らず学習時間は一定

（Caffeは、各学習率において、他フレームワークにより4.2～4.8倍高速）

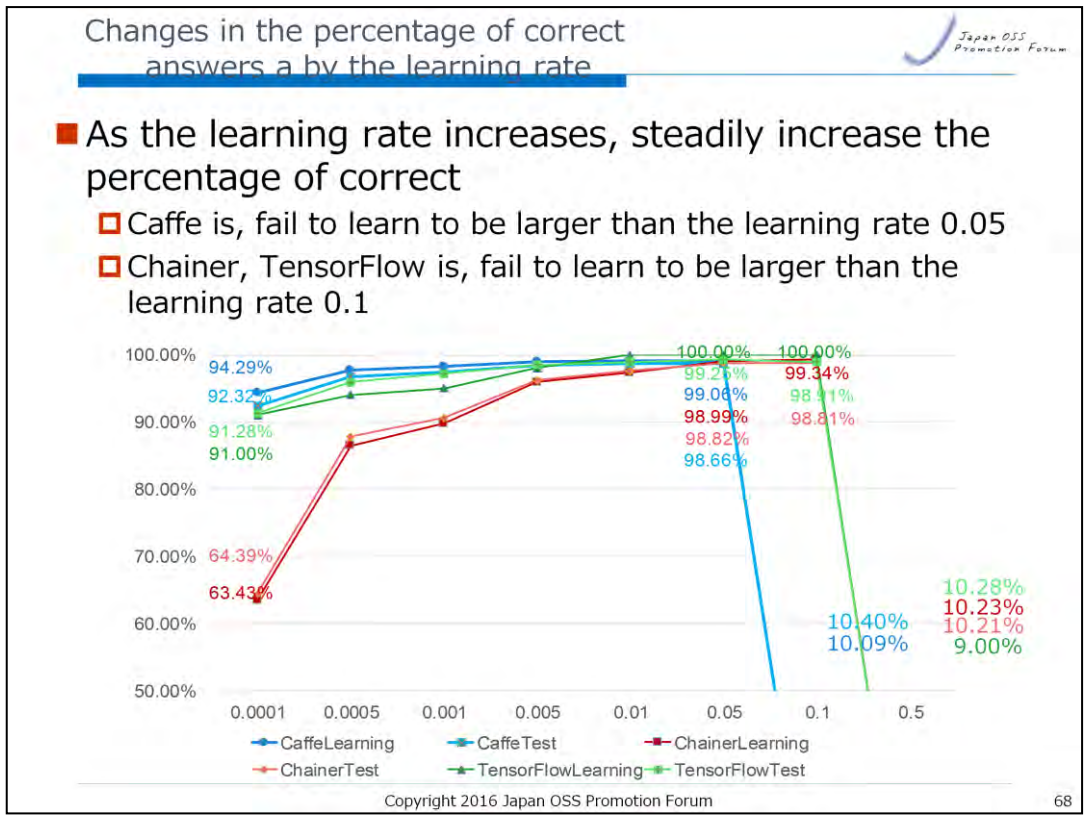
・Chainerは、学習時間に揺らぎがあり安定しない

※一般的に、学習率が大きくなると過学習、小さくなると収束に時間がかかるようになる。

※一般的に学習率が小さすぎると学習が遅いが、Caffe,TensorFlowの場合は、常に一定の学習時間となっている。Chainerは、学習時間が波打っているが・・・

※確率的勾配降下法（SGD）だと学習率が大きい場合に、全てがNanになってしまう（wが行方不明になる）つまり、Caffeの場合、学習率を大きくしすぎて学習失敗（発散）してしまった。http://www.slideshare.net/ToruUenoyama/tensorflow-gdg

※Caffe は、学習率0.1、0.5で学習に失敗



※一般的に、学習率が大きくなると過学習、小さくなると収束に時間がかかるようになる。

※一般的に学習率が小さすぎると学習が遅いが、Caffe,TensorFlowの場合は、常に一定の学習時間となっている。Chainerは、学習時間が波打っているが・・・

※確率的勾配降下法（SGD）だと学習率が大きい場合に、全てがNanになってしまう（wが行方不明になる）つまり、Caffeの場合、学習率を大きくしすぎて学習失敗（発散）してしまった。http://www.slideshare.net/ToruUenoyama/tensorflow-gdg

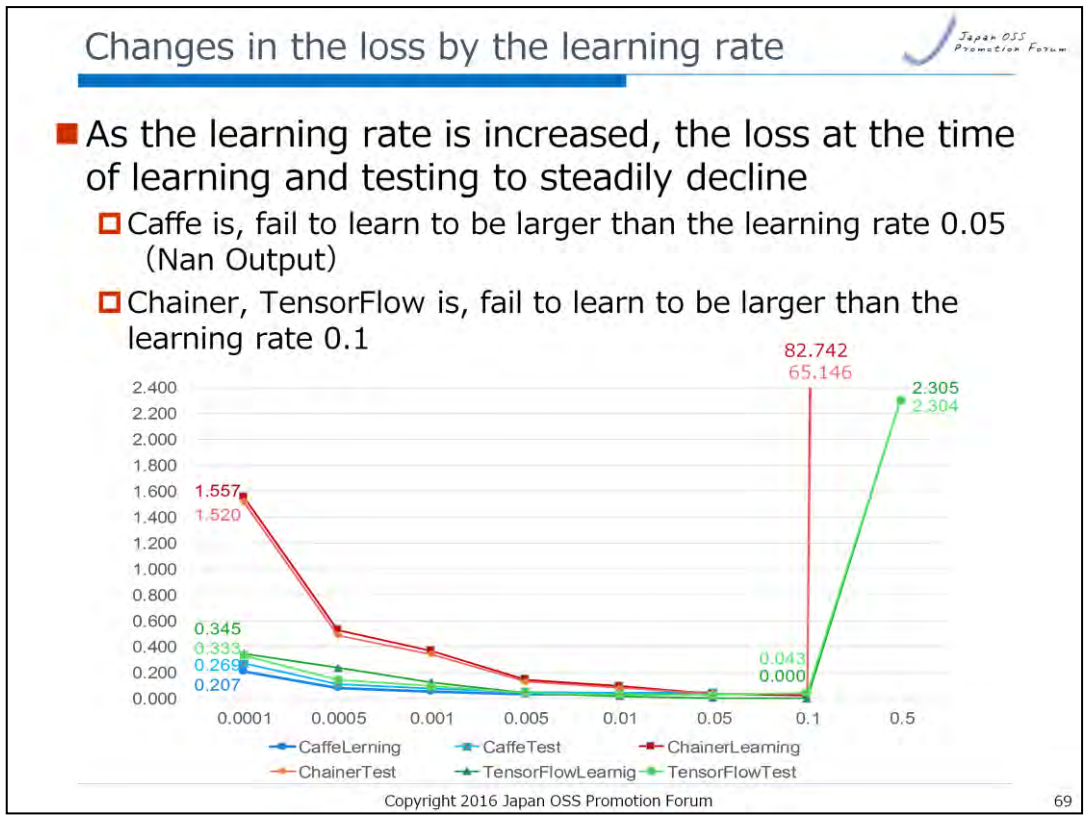
※Caffe は、学習率0.1、0.5で学習に失敗

＜事実のみ＞

・Caffeは、学習率を0.05より大きくすると学習に失敗（発散）

・Chainerのみ、学習率0.5において高い正答率（学習に失敗しない）

・TensorFlowは、学習率を0.1より大きくすると学習に失敗（発散）

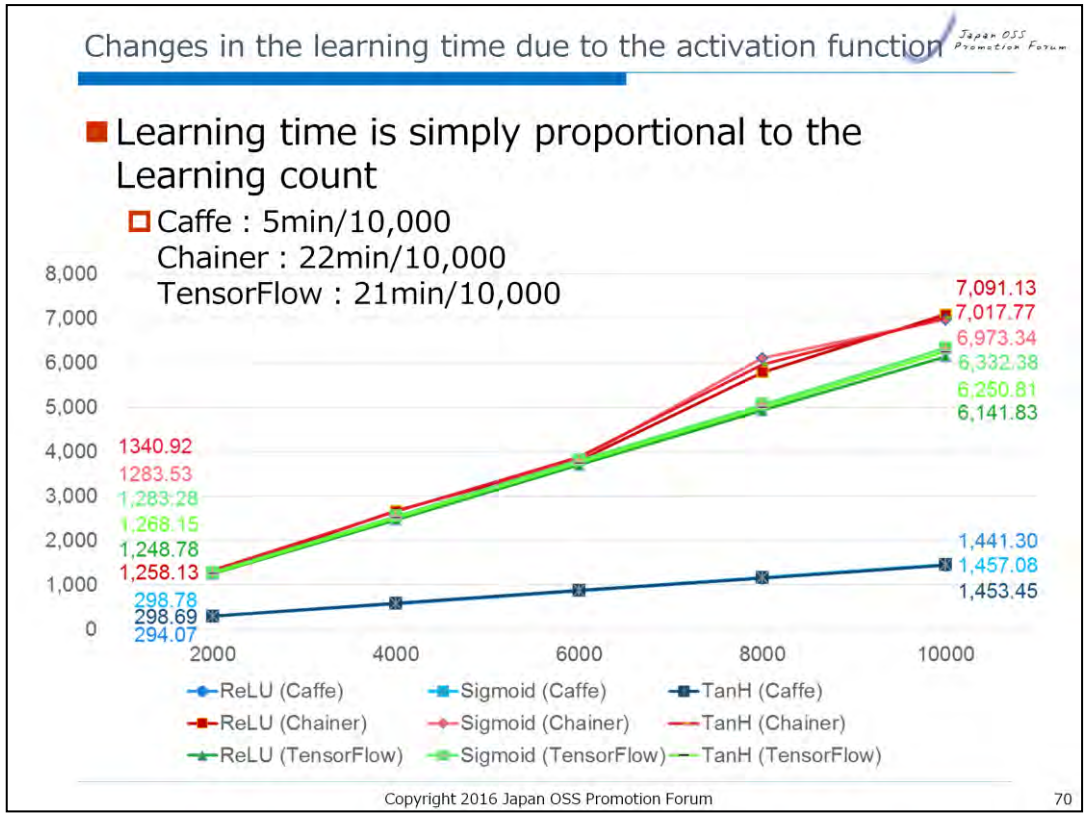


＜事実のみ＞

・Caffeは、学習率0.0.5より大きくなると学習に失敗（Nanで怒られる）

・Chainerのみ学習率0.5まで順調に低下

・TensorFlowは、学習率0.1より大きくなると学習に失敗（発散）

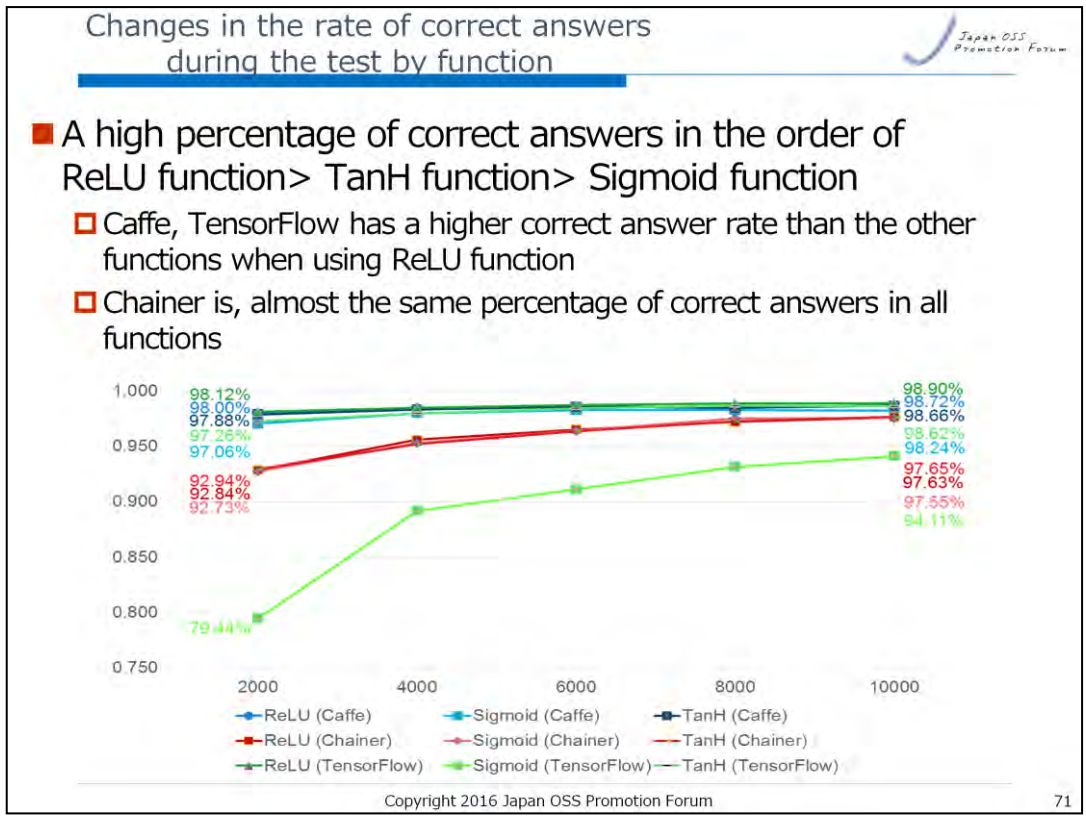


＜事実のみ＞

・Caffeは、活性化（非線形）関数を変更しても学習時間はほぼ同じ（学習回数に単純比例）

・Chainerは、学習回数が6,000より大きくなると、何らかの原因で学習時間が増加

・TensorFlowは、活性化関数を変更しても学習時間はほぼ同じ（学習回数に単純比例）



＜事実のみ＞

・Sigmoid（Chainer）は、正答率が常に最も低い

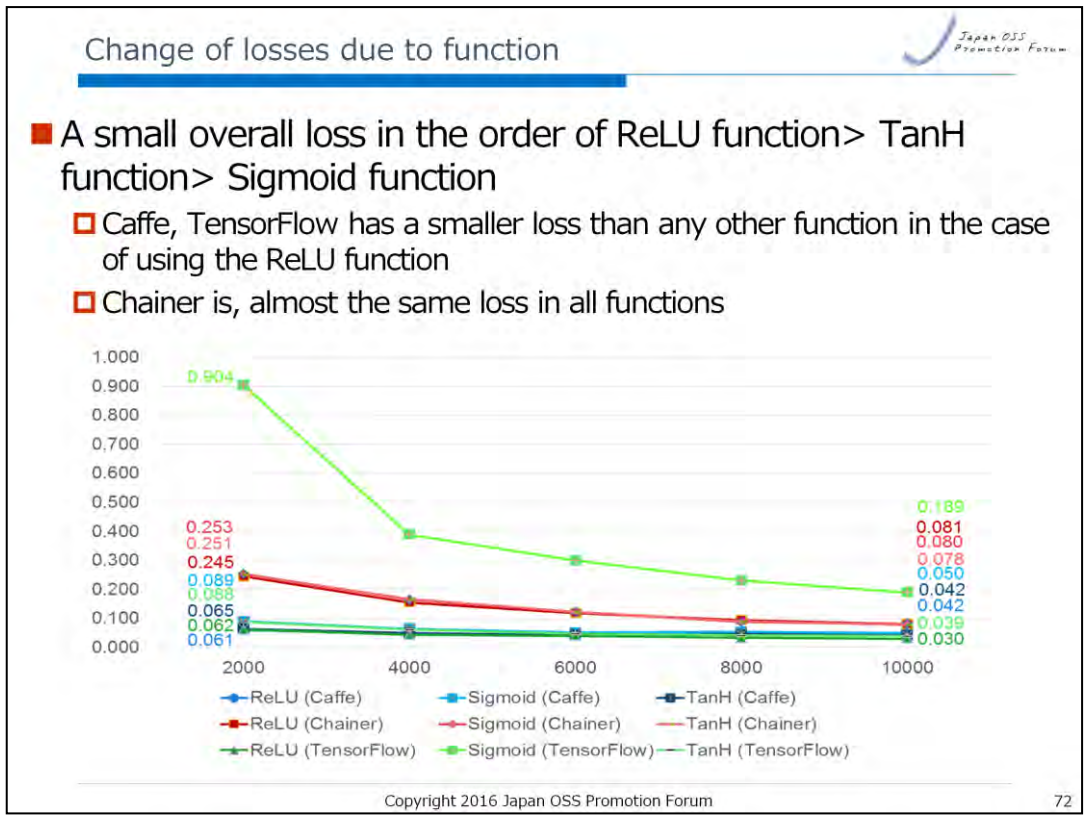
・Sigmoid（TensorFlow）は、正答率が他フレームワークよりも低い

※Sigmoid(Chainer)を除く

・Sigmoid関数は、全てのフレームワークで最も正答率が低い

・ReLU関数は、全てのフレームワークで最も正答率が高い

※一般的に、昔から使われているsigmoid、tanhは勾配消滅問題あり、ReLUは最近流行りで学習が速く勾配消失問題の心配なし



＜事実のみ＞

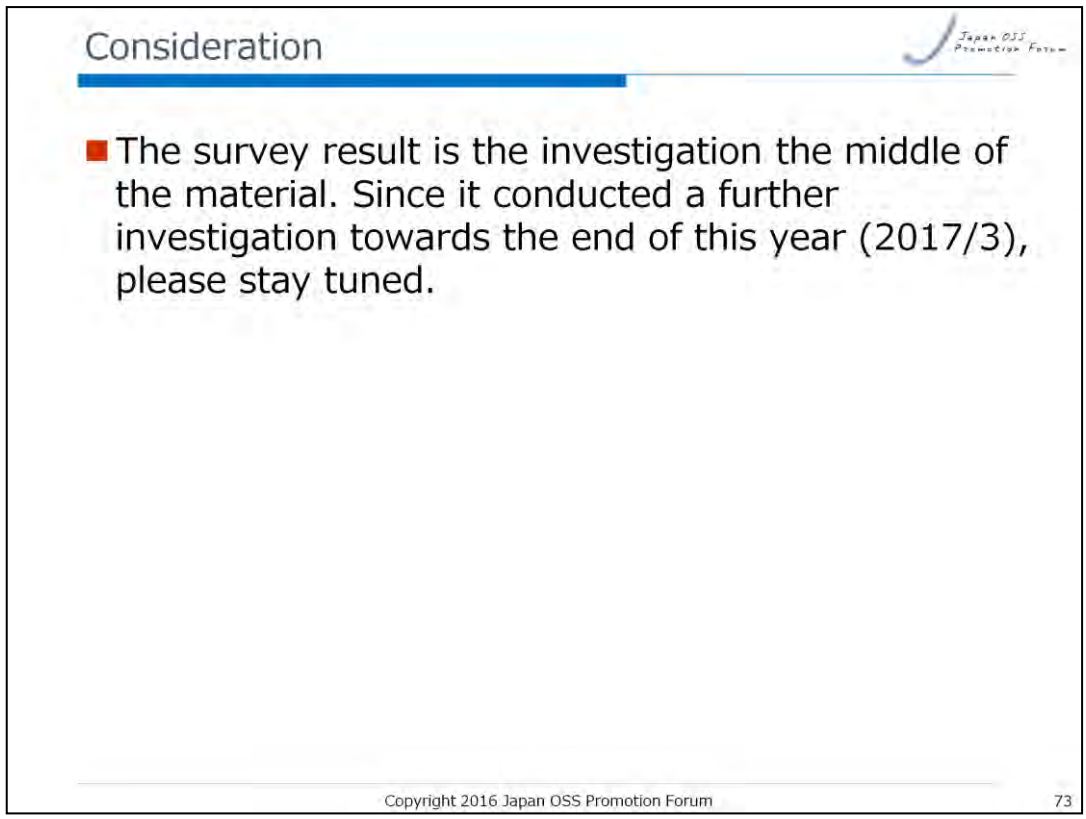
・Sigmoid（Chainer）は、損失が常に最も大きい

・Sigmoid（TensorFlow）は、損失が他フレームワークよりも大きい

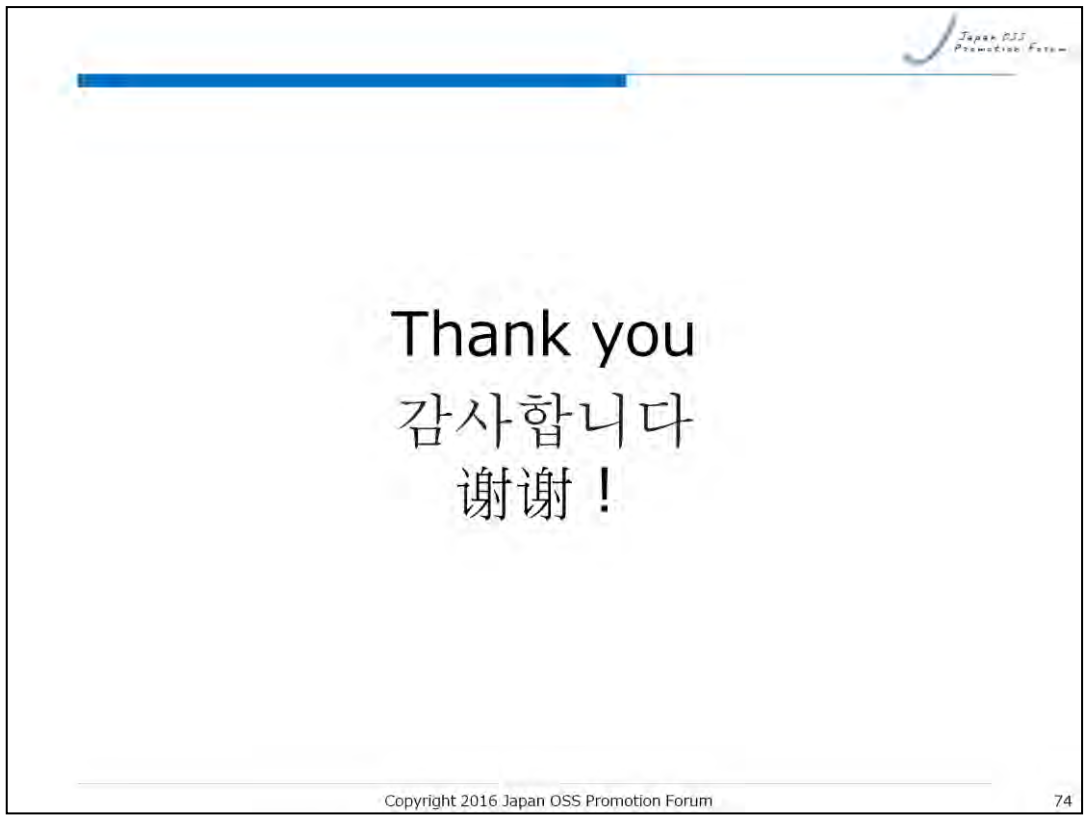
※Sigmoid（Chainer）を除く

・Sigmoid関数は、全てのフレームワークで最も損失が大きい

・ReLU関数は、全てのフレームワークで最も損失が小さい



本日、ご紹介した資料は、調査途中の資料です。
2017/3の年度末に向けてさらなる調査を実施しますので、
ご期待ください。



ありがとうございました。