cloud security alliance®

BIG DATA WORKING GROUP

ビッグデータの分類

2014 年 9 月

日本語版の提供について

本書「**ビッグデータの分類**」は、CSA Big Data Working Groupが公開している「**Big Data Taxonomy**」の日本語訳です。
本書は、原文をそのまま翻訳したものです。従って、日本独自の法令や基準に関する記述は含まれておりません。また、本書内で参照されている URL 等は、すべて英語版へのリンクとなっております。原文と日本語版の内容に相違があった場合には、原文が優先されます。

また、この翻訳版は予告なく変更される場合があります。以下の変更履歴（日付、バージョン、変更内容）をご確認ください。

変更履歴

日付	バージョン	変更内容
2014 年 12 月 16 日	日本語バージョン 1.0	

本書は、一般社団法人　日本クラウドセキュリティアライアンスの以下の有志により作成されています。
（敬称略、順不同）

笹原　英司
勝見　勉
諸角　昌宏

日本クラウドセキュリティアライアンスに関する情報は、以下の URL より参照可能ですのでご覧ください。
http://cloudsecurityalliance.jp

2014 年 12 月 16 日

謝辞

寄稿者

Praveen Murthy

Anurag Bharadwaj

P. A. Subrahmanyam

Arnab Roy

Sree Rajan

解説者

Nrupak Shah, Kapil Assudani, Grant Leonard, Gaurav Godhwani, Joshua Goldfarb, Zulfikar, Aaron Alva

デザイン/編集

Tabitha Alterman, Copyeditor

Frank Guanco, Project Manager, CSA

Luciano J.R. Santos, Global Research Director, CSA

Kendall Cline Scoboria, Graphic Designer, Shea Media

Evan Scoboria, Co-Founder, Shea Media; Webmaster, CSA

John Yeoh, Senior Research Director, CSA

概要

本書では、ビッグデータの 6 次元の分類を提案している。この分類の主な目的は、データ分析技術やセキュリティとプライバシーのフレームワークと同様に、計算処理とストレージ基盤の無数の選択肢に対して意思決定者が判断できるようにしている。分類は、分析されるデータの特性に軸を置いている。

目次

序論

ビッグデータという用語は、企業や政府機関が我々および我々の周辺から収集した大量のデジタル情報のことを意味する。このデータは、デスクトップコンピュータや携帯電話などを介した伝統的な情報交換やソフトウェア利用によって生成されるだけでなく、都市の街路（カメラ、マイクロホン）、ジェットエンジン（温度センサ）、また、最近急増中の仮想的にすべての電子機器がインターネットに接続されてデータを生成する IoT（Internet of Things）など、様々な環境に埋め込まれた無数のセンサからも生成される。

毎日 2,500 京バイトのデータが生成されているが、今日の世界のデータの 90%は、この 2 年間に生成されたものである(2011[1]年時点)。保存、処理、セキュリティとプライバシー、解析の問題は全て、大規模なクラウド基盤、データソースと形式の多様性、データ収集のストリーミング特性、大容量のクラウド間移動など、ビッグデータの速度、容量、種類によって拡大している。

6 つの軸から成る分類を図 1 に示す。

これらの 6 つの軸が、ビッグデータ基盤を確立するのに必要となる重要な側面を示している。本書では、それぞれの軸について説明していく。

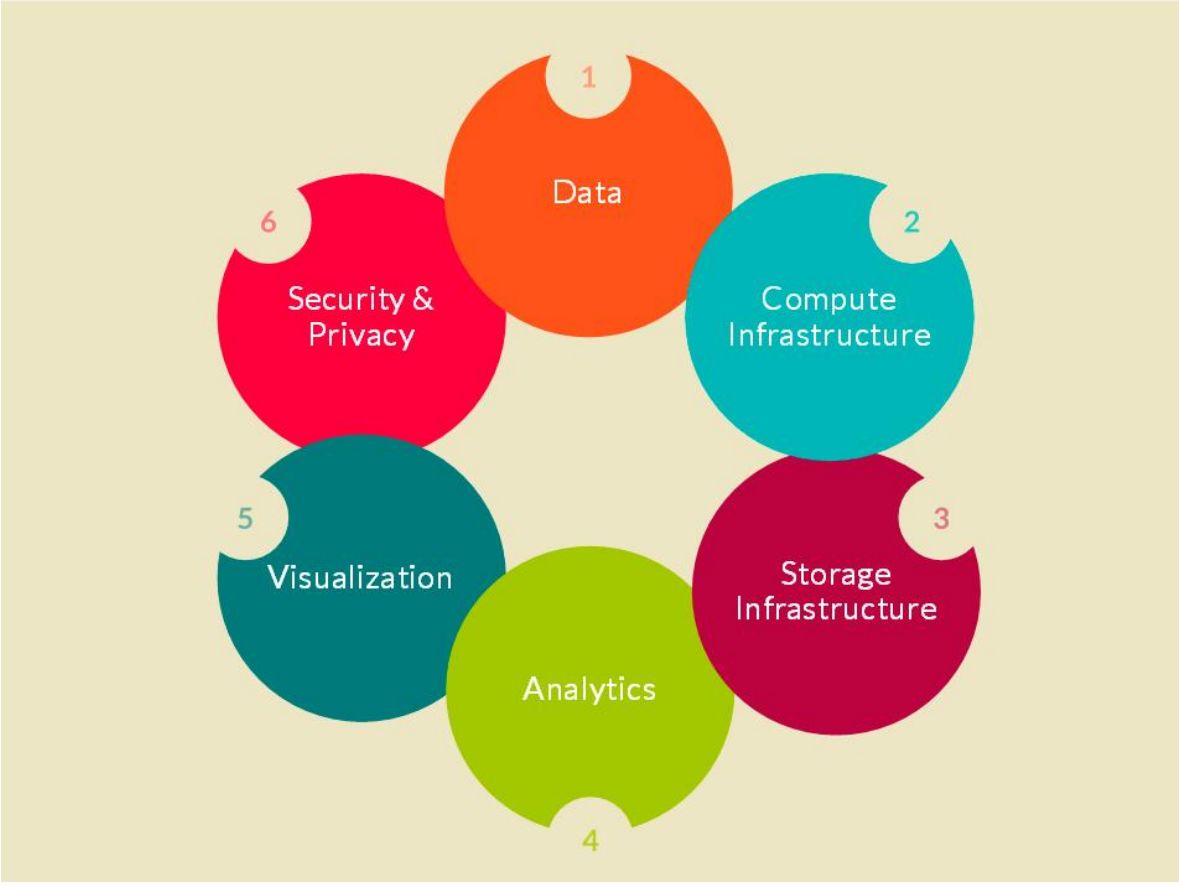

図 1: ビッグデータの 6 つの軸の分類

データ

最初の質問: ビッグデータが生まれる様々な領域[1]は何だろうか？ データが発生する領域を分類する理由は、特定のタイプのデータを生成するために必要な基盤の選択とその要件を理解するためである。すべての「データ」が同等であるというわけではない。データが発生する特定の領域は、それを保存し、処理し、解析を行うのに必要となるアーキテクチャの形式を決定するであろう。データ領域に関するこの質問について考える方法がいくつかある。

レイテンシー要件[2]

データを特徴付ける最初の方法は、分析に必要となる時間の間隔に基づくものである:

- **リアルタイム**（財務的なストリーム、複雑なイベント処理(CEP)[3]、侵入検知、不正検知）
- **準リアルタイム**（広告表示[4]）
- **バッチ**（小売、フォレンジック、バイオインフォマティクス、地理データ、さまざまなタイプの履歴データ）

「リアルタイム」アプリケーションの例

「リアルタイム」に到着するデータを扱うアプリケーションには以下のようなものがある:

- オンライン広告の最適化（リアルタイム入札を含む）
- 高い頻度のオンライン取引プラットホーム
- セキュリティイベントモニター
- 金融取引モニターと不正検知
- ウェブ分析やその他の様々なダッシュボード
- オンラインゲームあるいは電子商取引のための推移予測[5]
- 行動や利用状況に基づいて、デバイス、産業プラント、ロジスティクスシステムを最適化
- 制御システムに関連したタスク；　例えば、スマートグリッド、原子力発電所
- 話題に関係するツイートの感情・評判分析

これらのアプリケーションの大半で、データは絶えず変化している。特定のイベントに反応するためには、過去の全体のデータを考慮する代わりに、ある時間枠（「直近のページビュー」あるいは「直近の時/日/週/月のトランザクション…」）における関連データだけを考えることが必要であり実用的である。

ビッグデータテクノロジーソリューションに影響を与えるリアルタイムアプリケーションの主な属性

[1] Domain
[2] Latency requirements:　レイテンシー要件：　アプリケーション等のパフォーマンスに対する要件
[3] complex event processing (CEP)
[4] ad placement
[5] Churn prediction: 推移予測（顧客の入退会など）

当面の問題に対して最適な方法とビッグデータの技術的ソリューションを選択するためには、意思決定に影響を与える主な属性を理解することが重要である。レイテンシー要件（結果を計算するために使用可能な時間）に加えて、以下のことが含まれる:

- イベント特性
  - アプリケーションに必要な入出力データ転送速度が含まれる。
- イベントレスポンスの複雑さ
  - 処理の複雑性:
    - 各イベント処理作業における計算の複雑性は何か?
  - データ領域の複雑性:
    - このような処理をサポートするためにアクセスしなければならないデータのサイズはどのくらいか?
    - それはメモリに入るか、あるいは複数の場所やストレージ媒体に分散しているか?

当然のことであるが、計算処理およびデータ領域の双方で高度なイベントレスポンスの複雑性は、高い入出力データ転送速度と低いレイテンシー要件と共に、下層の基盤に対して最も厳しい課題を突き付ける。

レイテンシーは、入力イベントが発生し、そのイベントに必要な応答を行うまでの間、利用可能な時間である。別の言い方をすると、計算処理を実行して意思決定に至るまでの時間である。

ここで考えなければならない 2 つの広義のカテゴリとして、低いレイテンシーおよび高いレイテンシーの要件がある。

- ここでは、数十ミリ秒程度の応答時間を必要とするものを「低いレイテンシー」のアプリケーションと定義する。高頻度の取引、リアルタイムの入札、オンライン広告の最適化のようなアプリケーションには、アプリケーションとの関連で許容できるレイテンシーの上限がある。オンライン広告の最適化システムは、20-50 ミリ秒程度かかることがしばしばある。高頻度の取引システムでは、潜在的にリアルタイム応答の点でさらに厳しい条件が必要となる。特定のレイテンシーは、アプリケーションと基盤の機能であり、時間がたつにつれて改善していくものであるが、このカテゴリに含まれるアプリケーションは「リアルタイム」の応答を必要とする。

- 数秒から数分程度、潜在的には数時間の応答時間を必要とするものを「中から高のレイテンシー」のアプリケーションと定義する。例えば、ユーザとのやり取りやダッシュボードに関わるアプリケーションでは、通常、数秒あるいは数分単位で結果が更新されることも許容される。ほとんどの形式のレポーティングやより長い時間を要するデータ分析は、数分、あるいは数時間程度のレイテンシーを許容できる。

ここでの本当の問題は、アプリケーションがリアルタイムで反応できる（あるいは、反応しなければならない）かどうかということである。データが 1 秒あたり 10 万イベント来たとしても、管理者が収集したデータを数日に一度、手作業で検証するようなビジネス行動をとっている場合には、低いレイテンシーは大きなビジネス上の問題とはならない。一方、工程管理システムが一連のセンサからのデータに基づいて行われる場合や、企業がウェブサイトの新しい待ちうけページを展開していて訪問者数が急に落ちることを検知するような場合には、より速い応答が必要である。

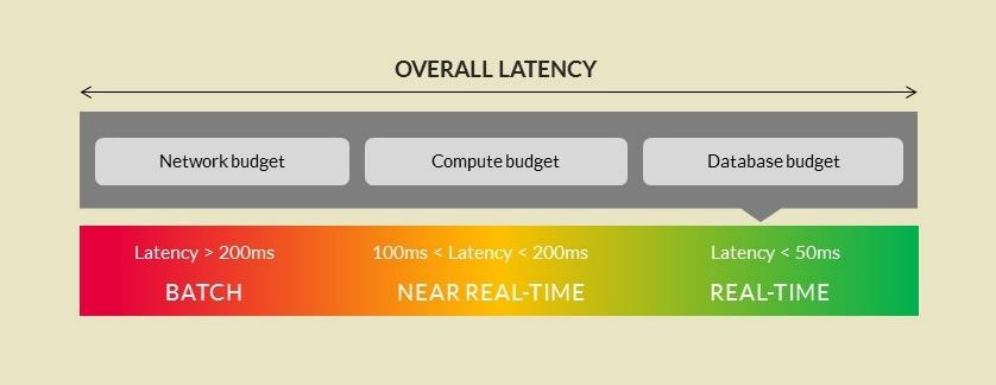

図 2: レイテンシー要件の特性

上記の図 2 に示しているように、全体のレイテンシーの計算は、大まかに言って通信ネットワークレイテンシー、計算処理レイテンシー、データベースレイテンシーから構成される。3 つの広義のカテゴリ（ネットワーク、計算処理、データベース）のそれぞれの正確な予算は、アプリケーションに大きく依存する。計算処理中心のアプリケーションは、ネットワークやデータベース操作に残している余地は少なくなる。

リアルタイムアプリケーションのためのビッグデータテクノロジーソリューション

適切なビッグデータの技術基盤を考えるとき、主要な問題の 1 つはレイテンシー要件である。低いレイテンシーが必要でない場合には、最初に、ディスクまたはメモリにデータを収集し、後でこのデータについて計算を行うという伝統的なアプローチで十分であろう。反対に、低いレイテンシー要件では、データが入ってくると同時に処理を行わなければならない。

構造

ビッグデータについて、様々な領域をマッピングする別の方法としては、データの構造または組織の度合いがある。

- **構造化**[6]（小売、金融、バイオインフォマティクス、地理データ）
- **準構造化**[7]（ウェブログ、電子メール、ドキュメント)
- **非構造化**[8]（画像、動画、センサーデータ、ウェブページ)。

データを、構造化、非構造化、準構造化として考えると役に立つ。これらの分類を正確に示す公式の定義がわかりにくい点に注意しながら、以下にいくつかの例を挙げる。

構造化データ

[6] **Structured**
[7] **Semi-structured**
[8] **Unstructured**

構造化データの例として、リレーショナルデータベースやスプレッドシートに含まれるデータが挙げられる。構造化データは、データが属する様々なフィールド（名前、住所、年齢など）と、各フィールドのデータタイプ（数値、通貨、文字、名前、日付、住所）によって大部分が特徴付けられたデータベースモデルに従う。また、モデルは、それぞれのフィールドの制限や制約の概念（たとえば、特定の範囲の整数）と、一貫性の概念に使用される様々なフィールドの要素間の制約（重複なし、2 つの異なった場所を同時に扱わないなど）を有する。

非構造化データ

非構造化データ（あるいは、非構造化情報）は、 事前に定義されたデータモデルがないか、あるいは事前に定義された方法で組織化されていない情報を示す。非構造化情報は、通常、大部分がテキストであるが、日付、数、ファクト[9]などのデータを含む場合もある。その他の例としては、写真、グラフィックイメージ、ビデオ、ストリーミングのセンサーデータ、ウェブページ、PDF ファイル、パワーポイントプレゼンテーション、電子メール、ブログのエントリー、wiki、ワープロ文書など、「生（タグ付けされていない）」データを含んでいる。

準構造化データ

準構造化データは、構造化データと非構造化データの中間に位置する。一種の構造化データであるが、下位のデータモデルによって課された厳格な構造を持っていない。準構造化データでは、タグあるいは他のタイプのマーカーがデータの中のある要素を特定するのに使用されるが、データに厳密な構造はなく、完全にセマンティックな意味を引き出すためには更なる処理が必要である。例えば、現在のワープロソフトは作者の名前と作成した日付を示したメタデータを含むことができる。一方、ドキュメントの大半は非構造化テキストを含んでいる （テキストをきちんとしたカテゴリに分類するモデルは存在しないので、高度な学習アルゴリズムにより、テキストが何について書かれているかを理解するためにテキストを分析する必要がある）。 追加のニュアンスとして、ドキュメントの中のテキストは、目次、章、節を含んでタグ付けされているかもしれない。電子メールには、送信者、受信者、日付、時刻、その他の固定フィールドに加えてメールメッセージの中味や添付の非構造化データがある。写真やその他のグラフィックは、作者、日付、場所、その他の内容特有のキーワード（写真の中の人名など）をタグ付けし、グラフィックスを体系化し位置付けることを可能にしている。XML とその他のマークアップ言語が、準構造化データを管理するためにしばしば使用される。

他に、領域を特徴付ける方法として、データから情報を生成・抽出する必要のある業界の類型を見るやり方がある。

- 金融サービス
- 小売
- ネットワークセキュリティ
- 大規模科学
- ソーシャルネットワーキング
- Internet of Things/センサーネットワーク
- 視覚メディア

[9] Fact: 処理されていない生の一次情報データ

図 3 は、ビッグデータ処理問題が発生する様々な領域と特定のサブ領域を示している。

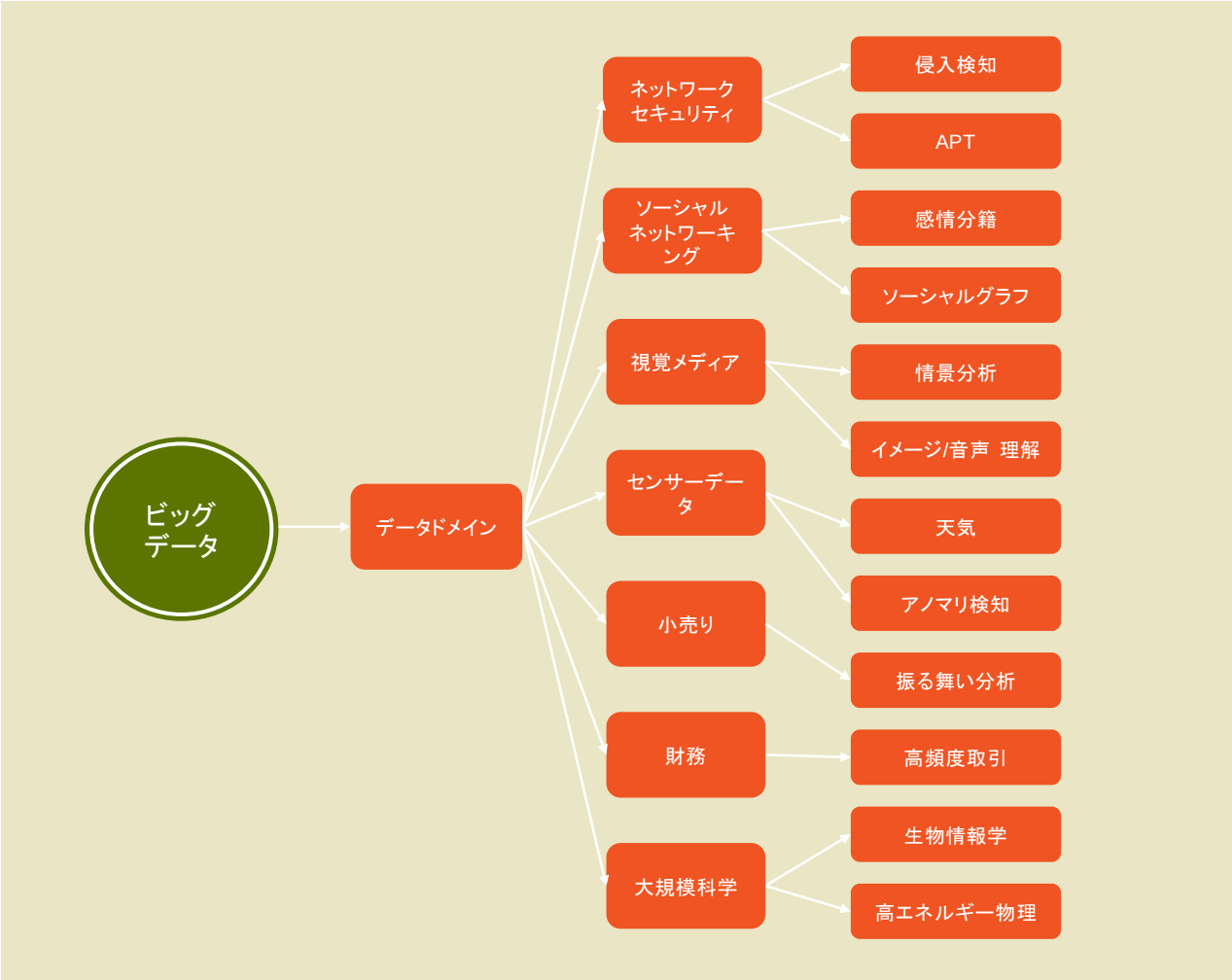

図 3: データ領域

図 4 は、ビッグデータの業務分野をどのように時間軸と組織の軸にマッピングするかを示している。実際、ここに含まれる全業務分野があらゆる組織レベルでデータに相対するユースケースを有しており、すべての応答時間につながる処理に対するニーズを有するケースが起き得る。その場合、業務分野領域は、ビッグデータの領域空間を特徴付けるためのもう 1 つの直交した軸になる。そして、一般的なユースケースによってこれらの領域を視覚化し、業務分野、時間、構造にマッピングする。

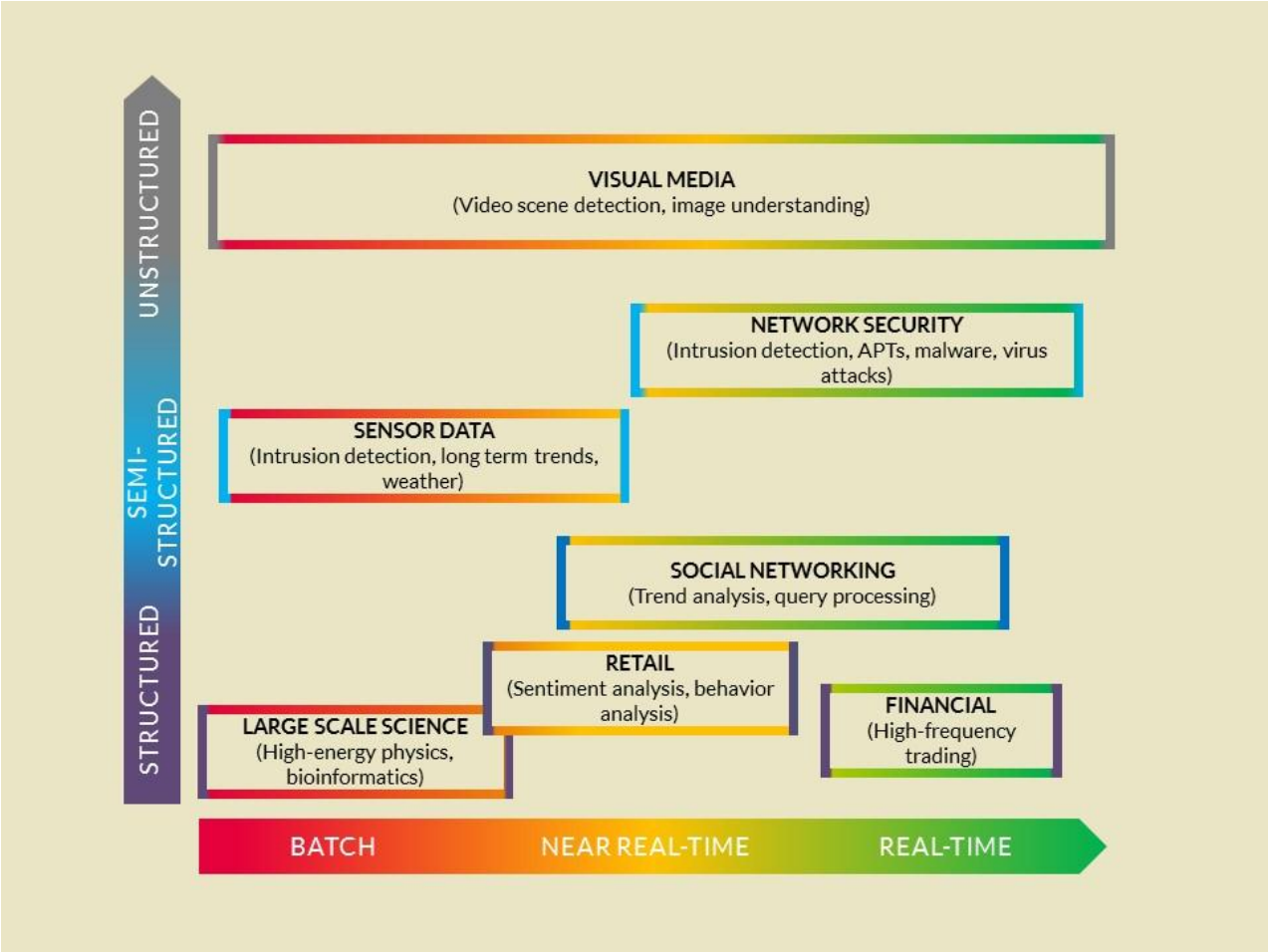

図 4: ビッグデータ業務分野マップ

計算処理基盤

Hadoop エコシステムは、コモディティ化した計算処理資源を使用して大規模なデータセットを並列処理できることで人気のある選択肢であるが、その他にも様々な領域で使用することができる計算処理基盤がある。図 5 は、様々なスタイルの処理のアーキテクチャの分類を示している。現在、ビッグデータに関する計算処理パラダイムは、処理をバッチモードで行うか、あるいはリアルタイム/準リアルタイムでストリーミングデータ（データが絶えず入って来て、すぐに処理する必要がある）を処理するかという最初の抽象化のレベルで異なる。本節では、バッチ処理のための Hadoop とリアルタイム処理のための Spark という 2 つの特定の基盤に焦点を当てる。

MapReduce は、大規模なデータセットを処理し生成するためのプログラミングモデルとそれに関連した実装である。ユーザは、1 組の中間の鍵/値ペアを生成するために鍵/値ペアを処理する map 機能と、同じ中間の鍵に関連したすべての中間の値をマージする reduce 機能を指定する。[2]の資料で参照されているように、多くの現実の世界の作業がこのモデルで表現できる。

この機能のスタイルで書かれたプログラムは、大規模のコモディティ化したマシンのクラスタ上で自動的に並列処理される。ランタイムシステムは、入力データを分割し、1 セットのマシン上でプログラムの実行を計画

し、故障したマシンを処理し、必要なマシン間の通信を管理する。これにより、平列処理や分散システムの経験のないプログラマでも、大規模な分散システムのリソースを簡単に利用できるようになる。

バルク同期並列処理[3]は、最初に Leslie Valiant によって提案されたモデルである。このモデルにおいて、プロセッサーは、いくつかのステップに渡り独立してローカルデータ上で実行する。処理の間、他のプロセッサーと通信することもできる。しかし、同期するために、実行中のいくつかのポイントですべて停止する;これらのポイントは、バリア同期点と呼ばれている。この方法により、デッドロックやライブロックの問題を簡単に見つけることができる。

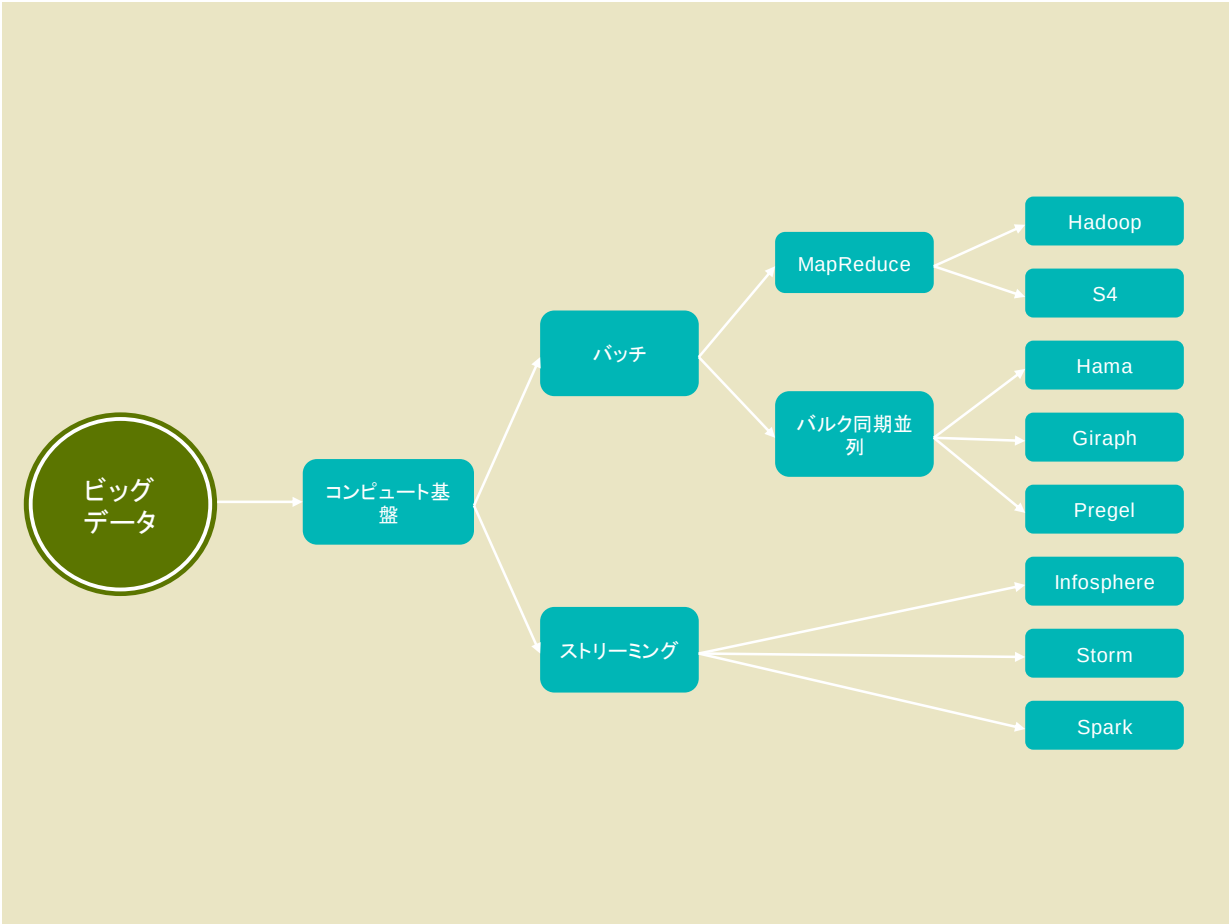

図 5: コンピュート基盤

低いレイテンシー: ストリーム処理

イベントが起こった時にアプリケーションが「すぐ」の応答を要求する場合、何らかの形のストリーム処理が必要になる。これは、基本的にデータが入った時に処理を行う。一般的な方法は、それぞれのイベントを別々に処理する小さなコードになる。処理を早くするために、ストリームは細分化され、クラスタに分散して処理される。

Apache Storm は、Twitter 用に開発され、リアルタイム処理のパラダイムを必要とする Twitter やその他の企業によって広められた人気のあるイベント処理フレームワークである。その他の例としては、Amazon の Kinesis、MapR のストリーミング機能がある。これらのフレームワークは、複数のクラスタノード上に拡張性を持たせ、レジリエンスとフォールトトレランスを様々なレベルでサポートしている。例えば、チェックポイントの設定を通して、システムが故障から復旧することができるようになる。

ストリーム処理フレームワークは、主に計算負荷を並列化させることだけを行う；追加のストレージレイヤーが、それらの結果を保存しクエリできるようにするために必要である。計算の「現在の状況」は、ストリーム処理フレームワークに含まれているが、通常、この情報にアクセスまたは参照するきれいな方法はない。特に、外部モジュールから行う方法はない。処理しているデータ量によって、より大きく高性能なストレージバックエンドが必要になるかもしれない。

かいつまんで言うと、Apache Spark（詳細は以下に記述）は、ハイブリッド手法を採用する。イベントはバッファに集められて、一定間隔（たとえば数秒ごと）にバッチ処理される。 Spark がメモリ内にデータを保持するので、原理的には受信したデータストリームについて行くことができる速いバッチ処理ができる。

まとめると、低いレイテンシーの処理は、コンピュートとストレージに関連した基盤で何らかの形のストリーム処理を行う。アプリケーション要件が非常に高い場合には（たとえば、ミリ秒以下のレイテンシーが必要な場合）、伝統的なソフトウェアスタックでは十分でない可能性があるということに注意すべきだ。 専門のソフトウェアスタックかコンポーネントが、アプリケーションに組み込まれていなければならないかもしれない。

高いレイテンシー: バッチ処理

アプリケーションコンテキストが高いレイテンシーを許容できる場合（例えば、数秒や数分で結果を生成することを必要としない）、バッチ指向のコンピューティングアプローチを取ることができる。

最も簡単な例では、アプリケーションが、必要なことを行うためにログファイルをスキャンすることできる。あるいは、アプリケーションが期待される結果を計算するためにデータをクエリした後、データベースにすべてのデータを入れることができる。ここで使用されるデータベースは、古典的な SQL データベース、Cassandra のようなピュアストレージデータベース[10]、あるいは CouchDB のような集約ジョブ[11]を実行するデータベースである。

バッチ処理は、Apache Hadoop（以下に詳細を記述）のようなフレームワークを使用することで効果的に拡張することができ、下位の処理に map-reduce パラダイムを組み込むことができる。ログデータは、クラスタ上に分散させた形で保存できる。アプリケーションは、応答時間を短縮するためにクエリを並列して実行できる。

Hadoop の成熟度に応じて、多くのプロジェクトが Hadoop 上に作られ進化してきた。その一例が、Apache Drill で、BigQuery がベースにした Google の Dremel と同様の垂直型データベース[12]（カラム指向のデータベース）である。垂直型データベースは、テーブル全体をスキャンし何らかの基準に合ったエントリーの数を数えなければならないタスクのために最適化されている。伝統的なデータベースのように行にデータを保存する代わりに、データはカラムに保存される。ログファイルのように 1 エントリーあたりに 1 行のデータを保存する代わりに、それぞれのフィールドのデータを取り出してそれを一緒に保存することで、はるかに優れた IO 特性をもたらしている。その他の垂直型データベースとして、HP Vertica や ParStream がある。

いくつかのプロジェクトや製品は、ストレージメディアとしてデータベースの下位に位置するディスクをメモリ（あるいは、フラッシュとメモリの組み合わせ）にリプレースし始めた。これは、 ディスクスピードの限

[10]全てのデータ記憶領域をフラッシュメモリで構成する
[11] aggregation job
[12] Vertical database

界を超えるために SAP Hana で最も顕著に行われ、GridGain や Apache Spark でも行われている。これらのシステムは、クエリのターンアラウンドタイムをかなり減少しているが、本質的には依然としてバッチ処理である。フラッシュベースのメモリを使用した高性能ソリューションの他の例には、Aerospike がある。

Hadoop 1.0

Hadoop[4]は、Google の独創的な MapReduce [2]と Google ファイルシステム[5]の文献から起こったオープンソースの分散プログラミングとストレージ基盤である。これは、”map reduce”コンピューティングのパラダイムをベースとし、最初に計算処理タスクへの入力を分割して map を実行し、いくつかの Worker ノード[13]に割り振る。Worker ノードは、入力のサブセットを個別に処理する。そして、”reduce”ステップでは、すべての map サブプロセスの結果を集め、一定の法則に従って全体の計算タスク（図 6）の出力を形付けるために結合する。データは、非構造化データの Hadoop ファイルシステム、あるいは、構造化データのデータベースに格納される。Hadoop は、データが非常に大きく 1 つのコンピュータのディスクサイズに収まらない場合でも、処理できるように設計されている。MapReduce パラダイムは、計算を行う所にデータを移動させるのではなく、データが格納されるところで計算を行うように設計されている。フレームワークは、データの複製を通してフォールトトレラントが高くなるように設計されており、ポーリングを通して Worker ノードの進捗を追跡するアーキテクチャを介して、いくつかのノードが落ちた場合には他のノードに作業を再割り当てする。さらに、フレームワークは、自動的にコンピュート/ストレージネットワークにわたって”map”, “reduce”に分割される。全ては Hadoop ランタイムによって自動的に行われ、開発者は計算タスクのために map および reduce のルーチンを書くだけでよい。

全体の Hadoop エコシステムは、以下のプロジェクトから構成される:

- **Pig –**高級言語を提供するプラットホームで、大規模データセットを解析するプログラムを表す。pig は、Pig プログラムを Hadoop フレームワークが実行する一連の MapReduce ジョブに翻訳するためのコンパイラを備えている。
- **Hive –** Hadoop 環境の上で作られたデータウエアハウスソリューション。テーブル、カラム、パーティションという良く知られているリレーショナルデータベースの概念、および、Hadoop の非構造型の世界のための SQL のサブセット（HiveQL）を導入している。Hive クエリは、MapReduce ジョブにコンパイルされ、Hadoop を使用して実行される。
- **HBase –** カラム指向の NoSQL データストレージ環境で、Hadoop において大規模でまばらに存在するテーブルをサポートするように設計されている。
- **Flume –** 生成されたままの大容量データを効率的に移動するための分散型で、信頼性が高く、可用性が高いサービス。Flume は、ログが生成されると複数のシステムからそれを集め、Hadoop 分散ファイルシステム (HDFS)に挿入する。
- **Lucene –** 高性能でフル機能を備えたテキスト検索を提供するサーチエンジンライブラリ。

[13] worker node

- **Avro –** データタイプとプロトコルを定義するための JSON を使用したデータシリアライズ技術で、コンパクトなバイナリ形式でデータをシリアライズする。
- **ZooKeeper –** 構成情報と名前付けを維持するための集中管理サービスで、分散環境の同期とグループサービスを提供する。
- **Oozie –** Apache Hadoop ジョブの実行を管理し調整するワークフロースケジューラシステム。

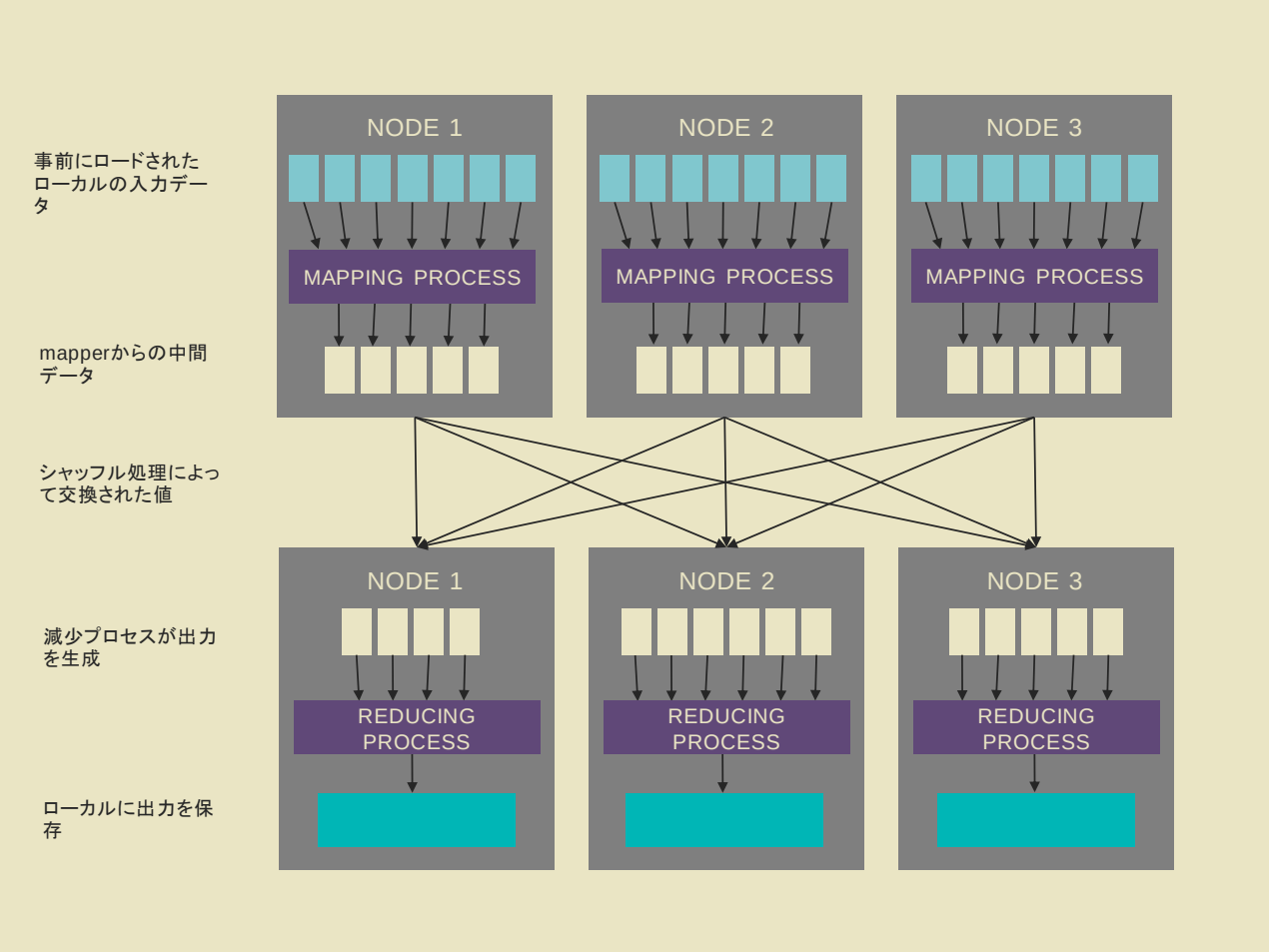

図 6: Hadoop MapReduce フロー[7]

Hadoop はバッチ処理に向いているので、一般的には終端の無いデータストリームには向いていないと考えられている。これは、Hadoop ジョブが、すべてのデータが様々なノード上のファイルに存在し、Map と Reduce フェーズが、固定量の入力に対して行われ固定量の出力を生成するように開始されると仮定しているからである。終端の無い一定のデータストリームに対するストリーミングアプリケーションにおいては、このモデルでは不十分である。新たに入ってきた入力を処理するために新たな map タスクを動的に追加すること（一方、ストリームデータを処理するシステムとして潜在的に古いデータの処理を取り除いていくには、スライディン

グウインド[14]を動作させる必要がある）は、非常に多いオーバーヘッドと、非常に多くのパフォーマンスに不利な条件を作り出す。

また、Hadoop は、反復型や前に計算された値に依存するアルゴリズムに対しては適していない。このクラスのアルゴリズムには、いくつかのタイプの機械学習アルゴリズムがあり、これは、オンライン学習アルゴリズムなどのような高度なデータ解析には不可欠のものである。[6].

Hadoop は、共有されたグローバルな状態に依存するアルゴリズムにも適していない。というのも、全体の MapReduce モデルが、並列に稼働する独立した map タスクに依存し、ロック、セマフォ、ネットワーク遅延による激しい性能のボトルネックを伴う、共有された状態へのアクセスを必要としないからである。 これが起こる例として、確率モデルにおける推論を実行するのに使用される Monte-Carlo のシミュレーションがある[6]。

図 7 は、Hadoop エコシステムの様々な要素がどのように絡んでいるかを示している。

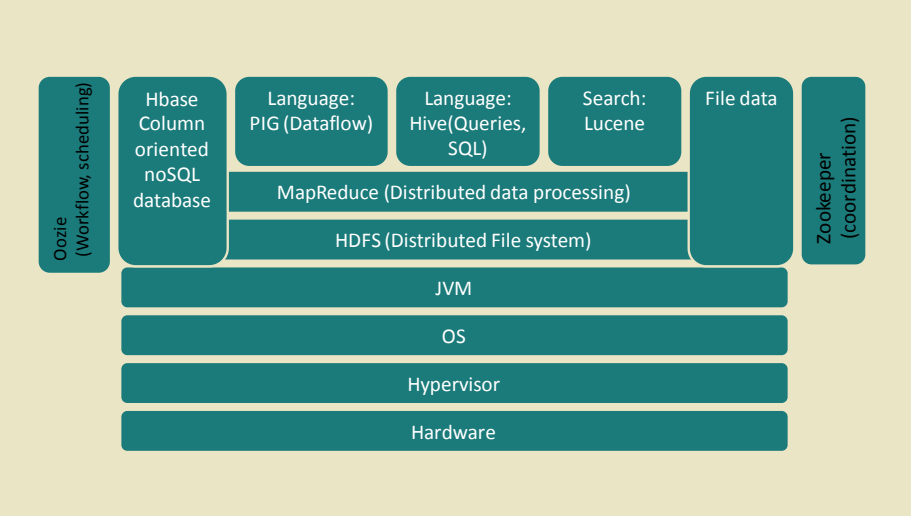

図 7:Hadoop1.0 スタック

Hadoop 2.0

これらの問題を解決するため、新しく Hadoop2.0 が開発された（図 8）。重要なのは、このフレームワークが、HDFS、リソース管理、MapReduce プログラミングを切り離し、YARN と呼ばれるリソース管理レイヤーを導入したことである。YARN は、下位のレベルのリソースを扱うものである。Hadoop2.0 のアプリケーションは、現在、Hadoop が管理するストレージとコンピュートリソース上で、それ自身のアプリケーションレベルスケジューリングルーチンを展開できる。

[14] sliding windows:　通信の高速化を図ったフロー制御の一つ。ウィンドウと呼ばれる概念を設け、そのウィンドウに空きがある限り、受信側からの応答を待たずに送信側が送信を行なうというもの

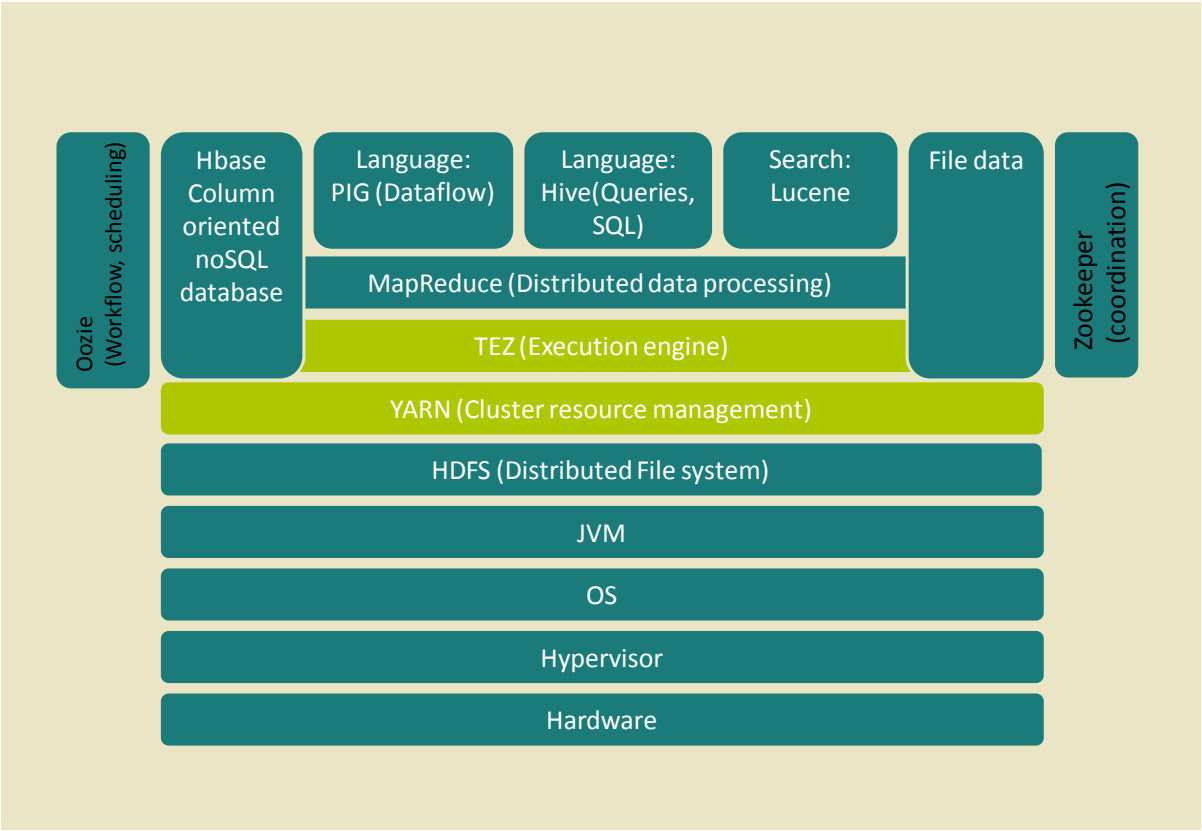

図 8: リソース管理のための YARN 層と実行のための TEZ がある Hadoop2.0 スタック

Berkeley Spark [39,40]

Spark は、カリフォルニア大学バークレー校で考案されたオープンソースのクラスタコンピューティングシステムで、実行時と開発時の双方でデータ解析を高速化することを目的としている。プログラムを高速化するために、Spark は単純なインメモリクラスタコンピューティングを提供する：これにより、Hadoop MapReduce のようなディスクベースのシステムよりもはるかに高速に、ジョブがデータをメモリにロードしクエリを繰り返し実行するジョブが可能になる。また、Spark は、処理スタックを統合することを目的としている。これは、MapReduce を使用したバッチ処理、HBase を使用した対話型クエリ、Twitter の Storm のようなフレームワークを使用してリアルタイムに解析を行うストリーム処理を統合する。これらの 3 つのスタックは、一貫した測定基準で維持することが難しい。また、ストリーミングデータに対して対話型クエリを実行することも難しい。統合された Spark スタックは、これらの要件を効率的かつ拡張性を持って扱うことができるように設計されている。

Spark の重要な概念は、レジリエンスのある分散データセット(RDD)[15]である。これは、RAM あるいはディスクに保存されたクラスタ全体に広がっているオブジェクトの集まりである。Spark のアプリケーションはクラスタのノードのメモリに RDD をロードし、Spark ランタイムが自動的にデータの分割とランタイム中の局所参照性を取り扱うことができる。これにより、高速な繰り返し処理が可能になる。受信データのストリームを一連

[15] resilient distributed dataset

のバッチに分け、一連の小さなバッチジョブとして処理できる。Spark アーキテクチャは、1 つのシステムでストリームとバッチ処理をシームレスに組み合わせることができる。

プログラミングをより高速化するために、Spark は、Scala、 Java、Python に対し、クリーンで簡潔な API を提供する。Spark は、大規模なデータセットのクエリを素早く行うために、Scala や Python シェルからインタラクティブに使用できる。Spark は、当初、繰り返しの機械学習アルゴリズムと会話型のデータマイニングという、メモリにデータを維持することが必要な 2 つのアプリケーションのために開発された。どちらの場合も、Spark は、Hadoop MapReduce より **100 倍以上**速く実行できた。

また、Spark は、Shark の背後で動くエンジンである。Shark は、 Apache Hive と完全に互換性のあるデータウエアハウスシステムで、Hive より 100 倍以上速く実行できる。Spark は新しいエンジンであるが、Hadoop がサポートしているどのようなデータソースにもアクセスできる。

Spark は、YARN や HDFS などと同様な下位のインフラストラクチャを使用しつつ、MapReduce の代わりに Hadoop2.0 エコシステムにシームレスに適合している。GraphX や MLlib ライブラリは、リアルタイムに実行できる高度なグラフアルゴリズムや機械学習アルゴリズムを搭載している。BlinkDB は、会話型 SQL クエリを実行する、大規模な並列処理で近似値を求めるクエリエンジンであり、応答時間をクエリの正確性より優先させ、その結果有意の誤差の限定という注釈が付けられる。BlinkDB は、2-10%の誤差の範囲内で Hive より 200 倍速く実行されてきた。

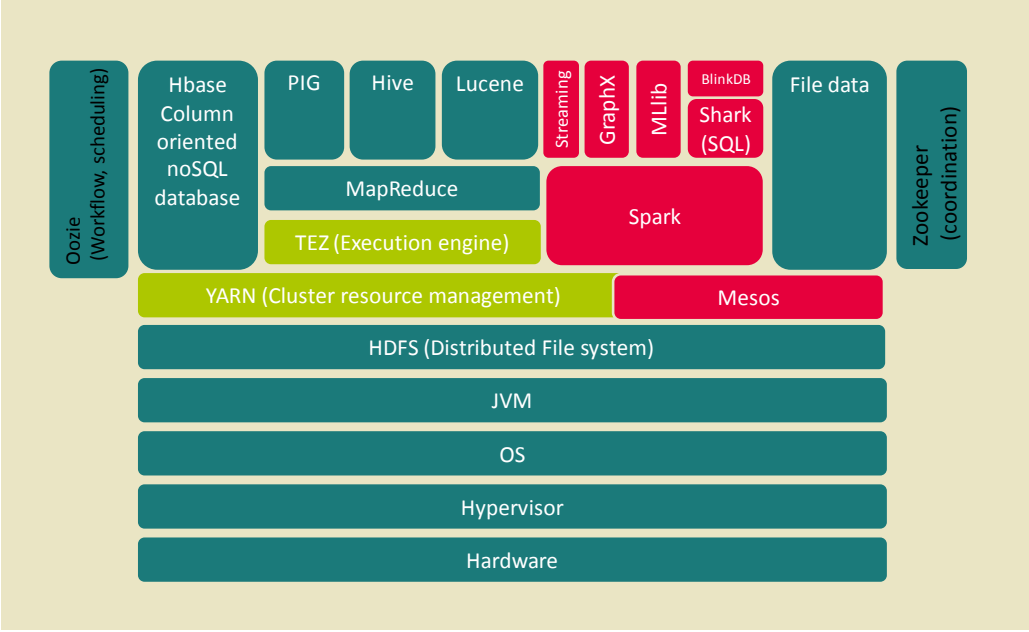

図 9: Hadoop2.0 エコシステム内の Spark

ストレージインフラストラクチャ

カラムと行のデータベース構造に簡単には収まらない様々な種類のマルチメディアやテキストなどの形式で、大容量のデータが非常に高速にやってくる。これらの要素の下で、大規模なデータセット[8]を持ったソーシャル、解析、ゲーム、金融、医療のアプリケーションを構築する時に、開発者が必要とする規模とスピードを提供するソリューションが生まれてきた。図 10 は、ビッグデータストレージに使用される様々なタイプのデータベースの分類を示している。

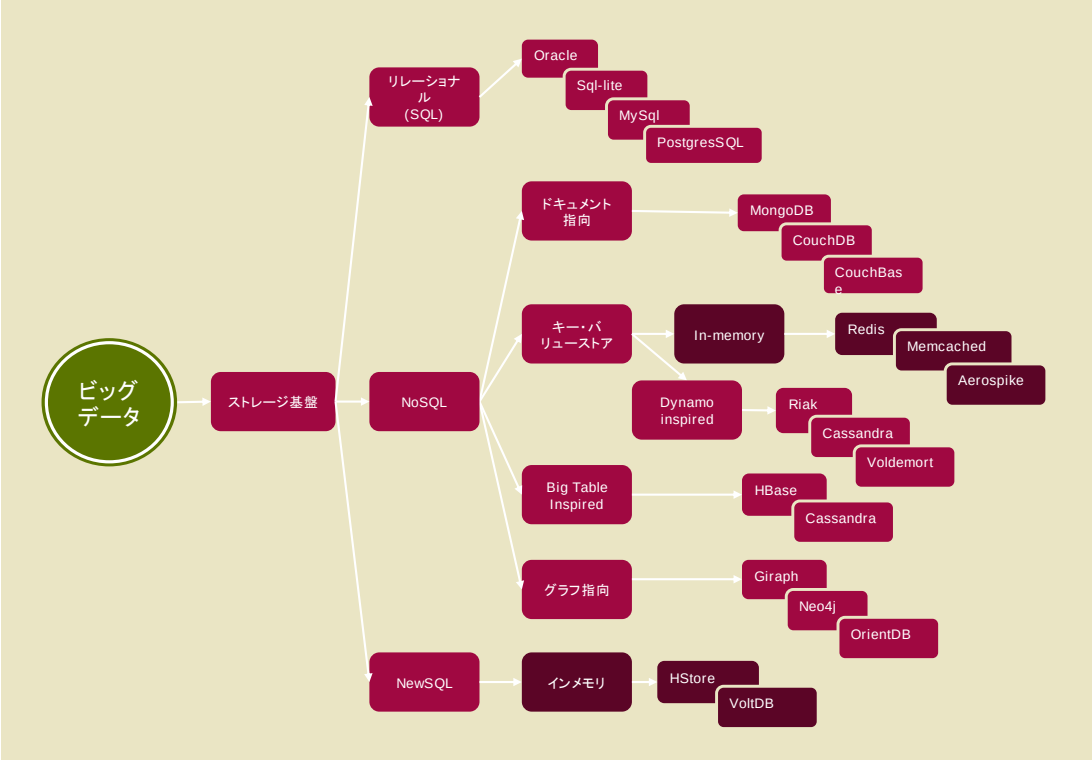

図 10: ストレージインフラストラクチャ

ここで、データの容量、速度、種類を扱うデータベースを拡張するためには、1 つのサーバをアップグレードする（RAM を追加したり HDD の容量を増加したりする）垂直型の拡張ではなく、複数のサーバに水平に拡張することが必要になる。しかし、水平に拡張することは、データが異なった場所に存在する分散アーキテクチャを意味する。この仕組みは、すべての分散されたコンピュータシステムで共有を行うという類のない挑戦、すなわち CAP 定理 [9]につながる。CAP 定理に従って、分散記憶装置は、**分断耐性**[16]（システムは、システムのどこか一部のメッセージの喪失または失敗にもかかわらず動作し続ける）を維持する一方で、**一貫性**[17]（誰でも同じデータを見る）あるいは**可用性**[18]（常に読んだり書いたりすることができる）のどちらかを犠牲にしなければならない。分断耐性は、ノードクラッシュ（いかなる理由においても）あるいはネットワークが任意の数のパケットをドロップする（スイッチの障害あるいはその他の理由）時に起こる分散処理システムの分断と

[16] **partition tolerance**
[17] **consistency**
[18] **availability**

してのオプションではないことに注意してほしい。回避不能な「分断」が発生し、分散処理システムが互いに通信することができない部分を持つ時、問題は分散処理システムが一貫性（一貫性を保証することができない時すなわち可用性を犠牲にする時、一貫性を満たすクエリだけに対応することを意味する）を満たすか、あるいは、可用性を満たす（たとえいくつかが矛盾しているとしても、すべてのクエリに回答できることを意味する）かどうかということになる。

データベースが判断される標準の測定基準が **ACID 特性**である:

- **原始性**[19]は、それぞれのトランザクションが、すべて実行されるか一つも実行されないかのどちらかになることを要求する。トランザクションの一部が失敗した場合、全体のトランザクションは失敗となり、データベースの状態は変わらないままになる。
- **一貫性**[20]は、それぞれのトランザクションが、データベースを 1 つの有効な状態から別の有効な状態になることを保証する。
- **独立性**[21]は、トランザクションの同時実行が、トランザクションの逐次実行で得られるシステム状態と同じになることを保証する。
- **永続性**[22]は、トランザクションが一度コミットされると、停電、クラッシュ、エラーが発生したとしてもそのままの状態でいることを意味する。

ACID が焦点を当てている一貫性は、伝統的なリレーショナルデータベースのアプローチである。インターネットやクラウドベースのストレージモデルのニーズを扱うために、その対極にある設計理念が Eric Brewer[10]によって作られ、BASE（Basically Available, Soft state, Eventually consistent）と名づけられた。NoSQL データベースの大半は、BASE の理念に基づいており、CAP 理論の C あるいは A が選択される。

以下のテーブルは、最も一般的なデータベースモデルを、強み、および、CAP 理論の一貫性と可用性のどちらに従うかという観点からまとめたものである[8]:

[19] **Atomicity**
[20] **Consistency**
[21] **Isolation**
[22] **Durability**

	データベースタイプ						
	1.0 リレーショナル	2.0 ドキュメント	key Value 型	Big Table 誘発型	Dynamo 誘発型	グラフ	NewSQL
何ができるか	行/カラムにデータを保存する。サーバ上で、親と子のレコードをリモートで統合できる。スケール以上のスピードを提供する。垂直スケーリングの能力は多少あるが、水平スケーリングの能力は弱い。このタイプのデータベースは、ほとんどの人の起点となる。	ドキュメントにデータを保存する。親と子のレコードは、同じドキュメントに保存され、統合することなく一つのフェッチ操作で結果を返す。サーバは、ドキュメントの中に保存されたフィールドを意識し、それを参照し、その属性を選択して返す。	１つのキーに任意の値を保存する。大部分は、一つの値のみに簡単な操作を実行できる。通常、Redis は、例外としてレコードのそれぞれの属性を複数のステップでフェッチする。非常に単純で非常に高速である。	データは、Google の BigTable に関する論文[11]で考案されたカラム指向の保存場所に置かれる。CAP パラメータはチューニング可能で、一貫性あるいは可用性のどちらか望む方に調整できる。どちらの調整も運用上の負荷は大きい。	分散されたキー/バリューは、Amazon の Dynamo に関する論文で[12]で考案されたように保存される。Dynamo リングに書かれたキーは、書き込みが成功と報告されるまでいくつかのノードに保存される。また Riak は、ネイティブの MapReduce を実装する。	データを表現および保存するために、ノード、エッジ、プロパティを持ったグラフ構造を使用する。インデックスに左右されない隣接関係を提供する：これは、あらゆる要素が隣接した要素へのダイレクトなポインタを含んでおり、インデックス参照が必要ないことを意味する。どのようなグラフも保存できる一般的なグラフデータベースは、triplestore やネットワークデータベースなどの専門特化したグラフデータベースとは異なっている[13]。	これらのデータベースが高性能とスケーラビリティを提供する点を除くとリレーショナルと同様に伝統的な ACID 概念を維持する。SQL の高レベル言語のクエリ能力を保持していないが、高いスループットのオンライントランザクション処理が可能である[14]。

水平スケーリング	一貫性を保証するために冗長ノードとの間でデータを共有するというレプリケーションを通して水平スケーリングが可能である。そして、水平にパーティション化されたシャーディングを実現しているが、その技術は複雑になる。	レプリケーションあるいはレプリケーションとシャーディングを通して水平スケーリングを提供する。 また、通常、ドキュメント指向のデータベースは特別なクエリのために比較的低性能な MapReduce になる。	シャーディングを通して水平スケールを提供する。	優れたスピードと非常に広い水平スケール機能を提供する。	通常、最も良いスケールで、非常に強いデータの永続性を提供する。	Titan を除いて、今までのところ最も弱い水平スケーリングを提供する。	シャーディングを通して提供する。
CAP バランス	可用性よりも一貫性を優先する。	一般的に、可用性よりも一貫性を優先する。	一般的に、可用性よりも一貫性を優先する。	可用性よりも一貫性を優先する。	一貫性よりも可用性を優先する。	一貫性よりも可用性を優先する。	可用性よりも一貫性を優先する。
いつ使用するか	高度に構造化したデータがあり、何を保存するかを知っている時。実稼働環境のクエリが予測できる時。	「レコード」の概念が、比較的限定的な増加で、単一のドキュメントに関連する特性のすべてを保存できる時。	非常に単純なスキームで、上流のクエリの結果をキャッシュする、若しくは極端な速度のシナリオ（例. リアルタイムのカウンタ）の場合。	単一マシンの能力を超えたスケールを持った一貫性と書き込み性能を必要とする時。特に、Hbase は、実稼働環境でおよそ 1,000 ノード使用する。	システムが、書き込みのために常に利用可能な必要があり、事実上データを失うことができない時。	固定スキームや関係による結びつきが欠如したオブジェクトの集まりを保存する必要がある時。 SQL の複雑なクエリの代わりに、グラフデータベースを横断することによって、データに関する推論を実行することができる。	SQL クエリを効率的に処理できる、リレーショナルデータベースの拡張可能なバージョンが必要な時に、水平に拡張して強力な ACID を保証することもできる。
製品例	Oracle, SQLite, PostgreSQL, MySQL, VoltDB	MongoDB, CouchDB, BigCouch, Cloudant	CouchBase, Redis, PostgreSQL HStore, LevelDB	Hbase, Cassandra (inspired by both BigTable and Dynamo)	Cassandra, Riak, BigCouch	Neo4j, OrientDB, Giraph, Titan	VoltDB, SQLfire

テーブル 1: ビッグデータのためのデータベースタイプ

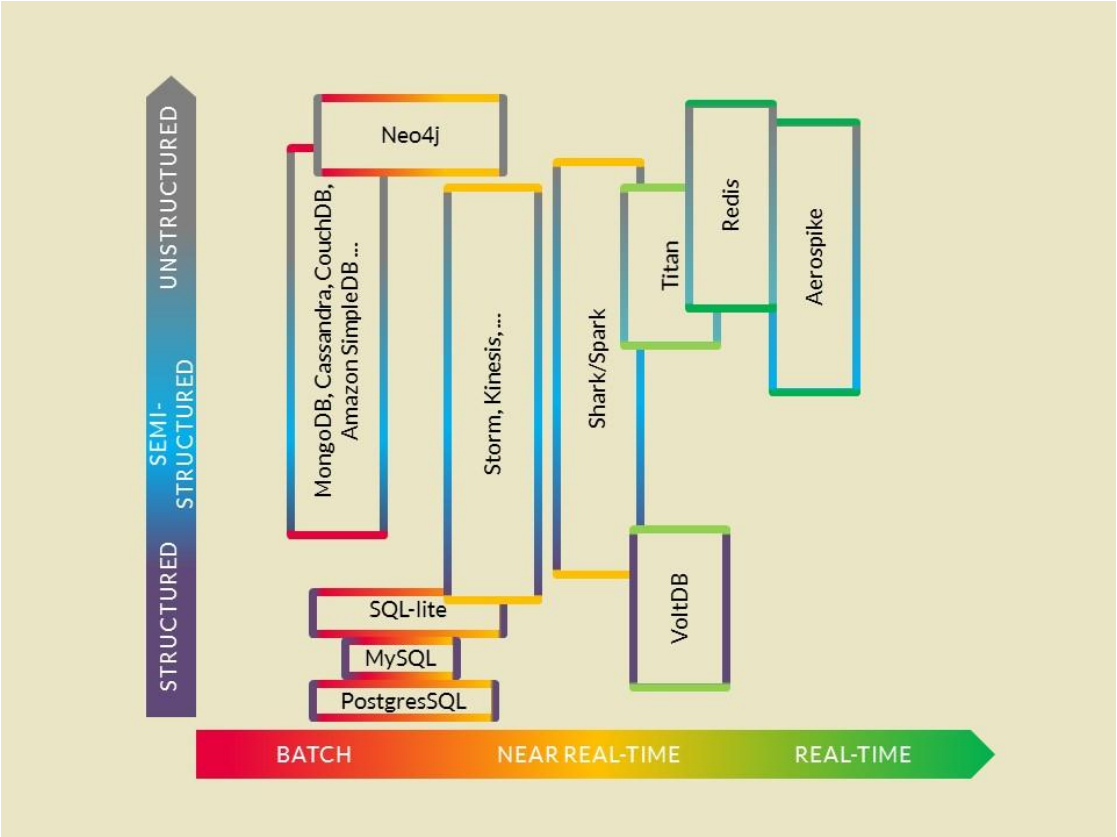

図 11: ストレージ技術のマップ

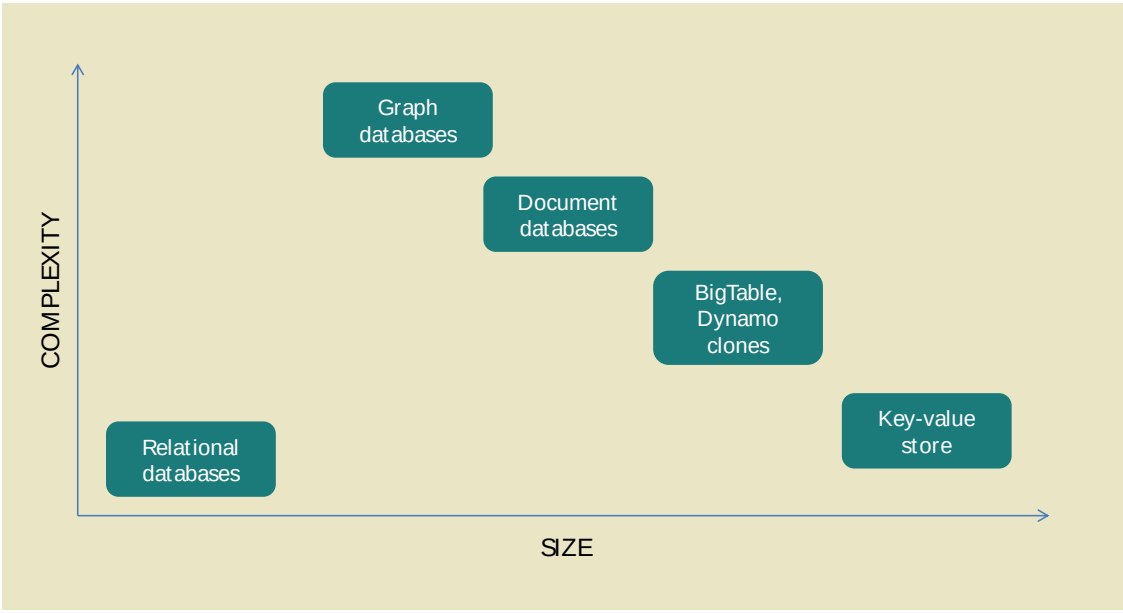

図 12: 様々なクラスのデータベースが扱うことができるデータの複雑さ対データ量[15]

パフォーマンス評価と代表的なベンチマーク

この節の最初に検討したように、過去数年間にわたって様々なタイプのデータベースや NoSQL ソリューションが急増し、キーバリューストア、文書データベース、グラフデータベース、NewSQL に細分化されてきた。

これらのソリューションがさまざまな特定分野を扱っているため、特定の部類の問題についてデータベースの状況をを評価することは、重要な反面ますます難しいタスクになっている。

どのような形でデータを保存しアクセスするかについては、様々なデータベースが様々な手法を使用している。Couchbase や MongoDB などのデータベースは、ワーキングセットの大部分が RAM にキャッシュされるハードウェアで動くことを意図している。Aerospike などその他のデータベースは、ソリッドステートディスク（SSD）への直接書き込みのために最適化されている。それは、たくさんの insert 処理に関わるベンチマークで有利になる。インメモリデータベースは、レイテンシーとスループットで利点がある一方、大規模なデータセットを扱うときにはスケーリングの問題が起きる可能性がある。SSD は、より高い密度で RAM よりも 1 ギガバイトあたりのコストが低いので、より少ない SSD データベースのノード上に非常に大きなデータセットを載せることができるようにすべきである。これは、非常に多量のデータを話題にする時に価値がある。より多くの RAM ベースのマシンを加えることによってスケーリングすると、より多くのマシンで故障率が高まるため、結果として故障からの復旧コストを高めることになる。

ベンチマークにおいては、データを保存しフォールトトレランスを達成するのに使用される様々な技術に加え、現実の世界で使用される際、データがアクセスされ信頼される方法に関して、様々なタイプのシナリオがあるという事実に取り組む必要がある。あるベンチマークの研究では、2 つの共通したシナリオが調査された：すべてのトランザクションがディスク上にコミットされ、ノード障害に備えてレプリケートされることが要求されるような強い持続性を必要とするアプリケーションと、最大限可能なスピードを達成するためにこれらの要件を緩和しても構わないと考えられているアプリケーションである[16]。別の研究では、読み出し専用、読み出しと書き込み、書き込み専用などのトランザクションを含むさまざまなシナリオで達成可能なスループットとレイテンシーが調査された[17]。 UC バークレーの AmpLab の別の研究では、特定のタイプのクエリのための 3 つの異なったシナリオが、いくつかの代表的なデータベースと Shark で比較された[18]。

一般的に、この進化の速い状況の中で、性能に関して決定的な声明を示すことは難しい。パフォーマンスは、使用されるクエリエンジン、ストレージアーキテクチャ、データが保存される方法に大きく依存している。図 11 は、相対的なパフォーマンスの幅広いカテゴリを示すことを目的としており、最も信頼のおける順位と解釈してはならない。図 12 は、データベースの様々なカテゴリが、それらが扱うことができるデータのサイズに対して保存されているデータ項目の複雑さをどのように比較するかを示している。

分析法

機械学習アルゴリズム

機械学習技術は、大規模で多次元のデータからの洞察を収集できる自動的でスケーラブルな方法を可能にする。機械学習は、コンピュータが自動的にパターンを学習し、データから推論を行う能力である。 いくつかの異なった軸に沿ってアルゴリズムを分類できる。

アルゴリズムタイプ

図 13 に、この分類を示す。

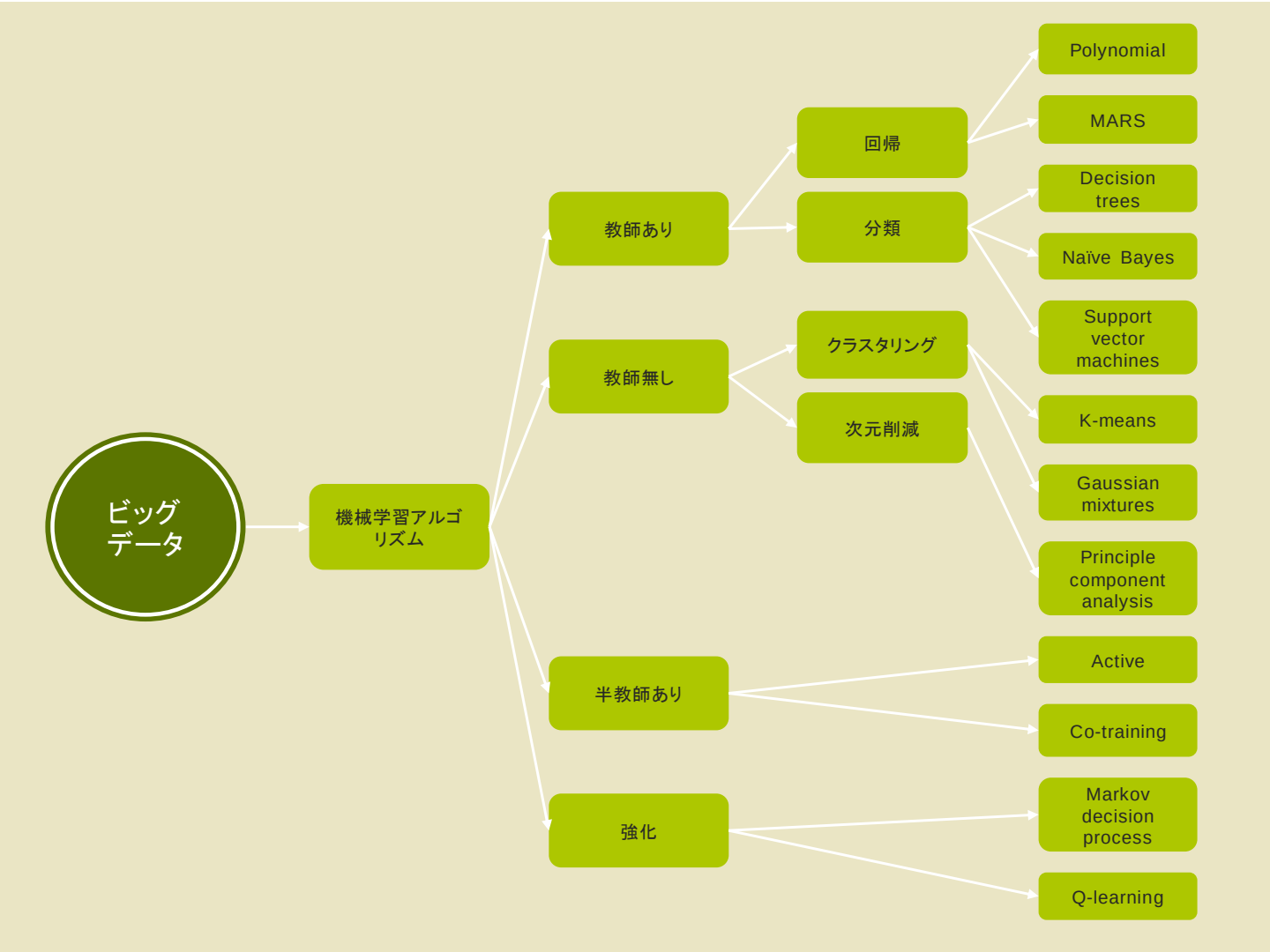

図 13: 機械学習アルゴリズム

教師あり学習[23] - このカテゴリは、入力データを、与えられた目標値あるいはクラスラベルにマッピングするすべての機械学習アルゴリズムである。一般的に知られているアルゴリズムは、カテゴリラベル（クラス）の予測である**分類**[24]と、連続値変数の予測である**回帰/予測**[25]を含んでいる。これらのアルゴリズムにおいて、学習者は、多くのトレーニングデータ上で特徴ベクトル[26]から 1 セットのクラス（分類の場合）あるいは値（回帰の場合）へのマッピング関数を近似させる。トレーニングデータのための必要条件は、あらゆるデータポイントにおける人間のラベルの存在である。何百万に及ぶ可能性のあるデータポイントのために人間のラベルを入手するのは高コストであり、様々なビッグデータアプリケーションのために、そのようなラベル付けをすることは容易でない。しかしながら、大規模でラベル付けされていない実社会のデータセット上で最先端の性能を達成するために、より小さなトレーニングデータセットを利用して成功したビッグデータアプリケーション

[23] **Supervised Learning**
[24] **classification**
[25] **regression/prediction**
[26] feature vector

が今日多く存在する。このカテゴリに属する最も広く利用されているツールは、分類すると以下の通りである:ニューラルネットワーク[27]、決定木[28]、サポートベクターマシン[29]、ナイーブベイズ[30]；また、回帰/予測としては以下がある:線形回帰[31]、多項式回帰[32]、放射基底関数[33]、MARS[34]、多重線形補間[35]。

教師なし学習[36] - このカテゴリは、関連する人間によるラベルを必要とせずに、入力データの隠された構造を学習するすべての機械学習アルゴリズムである。一般的に知られているアルゴリズムとして、**クラスタリング**[37]と**ソース信号分離**[38]がある。入力データのための必要条件は、意味のある知識発見に利用できる代表的な特徴が存在することである。この学習フレームワーク上で処理可能な豊富な大規模のラベル付けされていないデータセットに、アプリケーションが簡単にアクセスできるので、この技術は特にビッグデータに適している。このカテゴリに属する最も広く使用されているツールとしては、K-Mean クラスタリング[39]、ガウス混合モデリング[40]、スペクトラルクラスタリング[41]、階層的クラスタリング[42]、主成分分析[43]、独立成分分析[44]がある。

強化学習[45] - このカテゴリは、報酬関数[46]を最大にするために観察と行動の間のマッピング関数を学習するすべての機械学習アルゴリズムを含む。学習アルゴリズムは、通常、一定期間に最大の報酬をもたらす行動を取るように最適化される。そのような学習アルゴリズムとしては、ロボティクス[47]が人気であるが、ビッグデータコミュニティでは限定的な探索になっている。このカテゴリに属する 2 つの広く使用されているツールとしてはマルコフ決定過程[48]と Q 学習[49]がある。

半教師あり学習[50] -このカテゴリは、少量のラベル付けされたデータを使用して、この情報を大規模のラベル付けされていないデータセットに融合させ、適切な学習アルゴリズムに近づける。この類型のアルゴリズムは、ビッグデータコミュニティにとっては特に興味深いものになる。なぜなら、大大規模のラベル付けされていないデータセットが恒常的に存在すると、伝統的な教師あり学習アルゴリズムではうまく解決できないからであ

---

[27] Neural Networks
[28] Decision Trees
[29] Support Vector Machines
[30] Naïve Bayes
[31] Linear regression
[32] Polynomial regression
[33] Radial basis functions
[34] 多変量適応回帰スプライン
[35] Multilinear interpolation
[36] **Unsupervised Learning**
[37] **clustering**
[38] **source signal separation**
[39] K-Means Clustering
[40] Gaussian Mixture Modeling
[41] Spectral Clustering
[42] Hierarchical Clustering
[43] Principal Component Analysis
[44] Independent Component Analysis
[45] **Reinforcement Learning**
[46] reward function
[47] Robotics
[48] Markov Decision Process
[49] Q-Learning
[50] **Semi-Supervised Classification**

る。 他方、半教師あり学習技術は、ラベル付けされているデータとラベル付けされていないデータの間の構造的な共通点を効率的な方法で利用し、大きい大規模なデータセット上の機能的マッピングを一般化する。この分類のアルゴリズムにおけるサブカテゴリとしては、生成モデル[51]、グラフベースモデル[52]、マルチビュモデル[53]がある。このカテゴリにおいて最も広く使用されているツールとしては、能動学習[54]、転移学習[55]、共訓練[56]がある。

種々のデータマイニング

上記の分類は、単純で構造化されたデータセットに対しては適切であるが、複雑なむ非構造化データセットに対しては更なる特徴付けが必要で、それから得られるメリットがある。データの種類を分類できる 6 つの広義のカテゴリがあり、各種タイプの構造化されていないデータセットに適用するために前項で触れた様々な機械学習アルゴリズムを適合させ、強化することが可能である。

- **時系列データ[57]** - ビッグデータアプリケーションの大部分が時間に依存しているので、時系列データに対して拡張性のある機械学習アルゴリズムを適用することは重要なタスクとなる。時系列データとは、例えば株式市場のデータのように、繰り返し時間を測定して得られる一連の値かイベントである。時系列データをモデル化できる既存アルゴリズムとしては、隠れマルコフモデル[58]、マルコフ確率場[59]（Spatio-Temporal Modeling）、条件付き確率場[60]などである。これらのアルゴリズムをビッグデータに合わせて拡張することは、活発な研究トピックになっている。最近、研究者は、伝統的な動的時間伸縮法[61](DTW)アルゴリズムを何兆データポイントに拡張することに成功している。

- **ストリーミングデータ[62]** - ここで言うデータは、例えばリモートセンサ、小売のトランザクション、監視システム、インターネットトラフィック、電子通信ネットワークから絶えず到着している。このタイプのデータ要件を扱うために、機械学習アルゴリズムはオンラインで適用されなければならない。ほとんどの機械学習アルゴリズムは、クラスタリングのようなバッチデータ処理を必要とし、意味のある集まりを学習するためにすべてのデータを 1 つのパスで見る必要がある。そのような技術は、過去のデータを保存することが計算上不可能なストリーミングデータに対して、拡張性のあるものになっていない。最近、そのようなオフラインアルゴリズムにストリーミングデータを近似させる多くの分散型機械学習アルゴリズムが提案された。それには、分配型 k 平均法[63]や分散型特異値分解（SVD）がある。これらのアルゴリズムを

---

[51] Generative Models
[52] Graph-Based Models
[53] Multi-View Model
[54] Active Learning
[55] Transfer Learning
[56] Co-Training
[57] **Time Series Data**
[58] Hidden Markov Model
[59] Markov Random Fields
[60] Conditional Random Field
[61] Dynamic time warping
[62] **Streaming Data**
[63] distributed k-means

実装するいくつかのツールが、Apache Mahout と Vowpal Wabbit である。他方、アルゴリズムの観点から取組んでいるものとして、線形 SVM[64]、カーネル SVM[65]、並列木学習[66]などのアルゴリズムがある。

- **シーケンスデータ**[67] - シーケンスデータは、具体的な時間の概念の有無にかかわらず記録された、一連の順序づけられた要素若しくはイベントから構成される[19]。シーケンシャルデータの分析は、小売りデータ分析（1 種類のアイテムを買う顧客がさらにもう一種類のアイテムを買うかどうかを決定する）、あるいは、DNA とタンパク質配列の分析のような、いくつもの異なるコンテキストから発生する。しばしば、ここで使われる機械学習アルゴリズムとしては、隠れマルコフモデルとシーケンス調整アルゴリズム[68]（例えば DNA 配列のローカル調整のための BLAST）がある。

- **グラフデータ**[69] - 多くの問題が、グラフとして自然にモデル化される。それには、ソーシャルネットワークの分析、WWW の分析、生体ネットワークの分析、タンパク質構造の分析あるいは統合などを必要とする問題が含まれる。ほぼすべての大規模なグラフマイニングアルゴリズムは、大規模マトリクスの解法に必要なマトリクス形式で効率的に表現することができる。そのような大規模な処理を行うために提示された既存の技術としては、協調 フィルタリング[70]、特異値分解[71] (SVD)、ページランク[72]がある。これを扱うことを目的とした 2 つのツールが、Presto 法と GraphLab である。

- **空間データ**[73] - 空間データベースは、地図、医療画像処理データ、遠隔測定データ、VLSI チップレイアウトデータなどの空間に関連するデータを保持する。データ間の空間的な関係性のため、データは完全に独立しているということができない；学習アルゴリズムはこの事実を利用できる。

- **マルチメディアデータ**[74] - このカテゴリは、画像、動画、音声、テキストマークアップを含んでいる。このカテゴリに関するマイニングアルゴリズムは、画像分割、運動ベクトル解析、モデル構築のためのデジタル信号処理技術を多く含んでいる。

種類別のデータの分類は、特定のユースケースを持った異なる業界セグメントに簡単にマッピングできる。例えば、金融には時系列を、ソーシャルネットワーキングにはグラフデータをマッピングする。いくつかのカテゴリが、複数のタイプのデータにまたがる可能性がある。 例えば、大規模な科学は、ほとんどすべてのタイプのデータとマイニングアルゴリズムを伴う可能性がある。

統計的手法

[64] Linear SVM
[65] Kernel SVM
[66] Parallel Tree Learning
[67] **Sequence Data**
[68] Sequence alignment algorithms
[69] **Graph Data**
[70] Collaborative Filtering
[71] Singular Value Decomposition
[72] Page Rank
[73] **Spatial Data**
[74] **Multimedia Data**

機械学習は、ビッグデータを理解するための唯一のパラダイムではない。統計的手法は、長い間データを分析する標準的な方法だった。統計的手法と機械学習の技術の唯一の違いは用語であると主張した人もいた。 テーブル 2 は、2 つのコミュニティ（機械学習コミュニティと統計コミュニティ）の呼び方の違いについて、一般的な概念をまとめている[20]。

機械学習	統計
ネットワーク、グラフ	モデル
例/インスタンス	データポイント
ラベル	応答
重さ	パラメタ
特徴	共変量[75]
学習	フィッティング/推定
汎化[76]	テストセットパフォーマンス
教師あり学習法	回帰[77]/分類
教師なし学習法	密度推定、クラスタリング

テーブル 2 機械学習、統計の比較用語集

様々な類似性や見せかけの新しい考案があるが、確率や統計を全く伴わない機械学習アルゴリズムがあることも事実である。例えば、ベクトルマシンと人工のニューラルネットワークのサポートがある。さらに、ビッグデータ解析は、多量のデータだけではなく、高い次元性にもかかわることが多いので、計算上の問題は重要な検討課題になる。機械学習アルゴリズムは、伝統的な統計的定式化を無視した計算上の問題にも焦点を当てなければならない。従って、様々な意味で、統計的手法とパラダイムは機械学習技術の部分集合であると言うことができる。モデル化と推論への統計学者の関心について、アルゴリズムと予測に対する機械学習者の関心と比べてより哲学的な議論に入ることは可能であるが、このドキュメントの範囲外とする。

文献で頻繁に使用される別の用語として、「データマイニング」がある。これは、データから推論を導き出し予測を行うための全体を網羅した技術を示したより一般的な用語である。従って、機械学習技術もまた、推論と予測を行う人間によって使用される可視化技術を含んだデータマイニング技術の部分集合になる。

[75] Covariate
[76] Generalization
[77] Regression

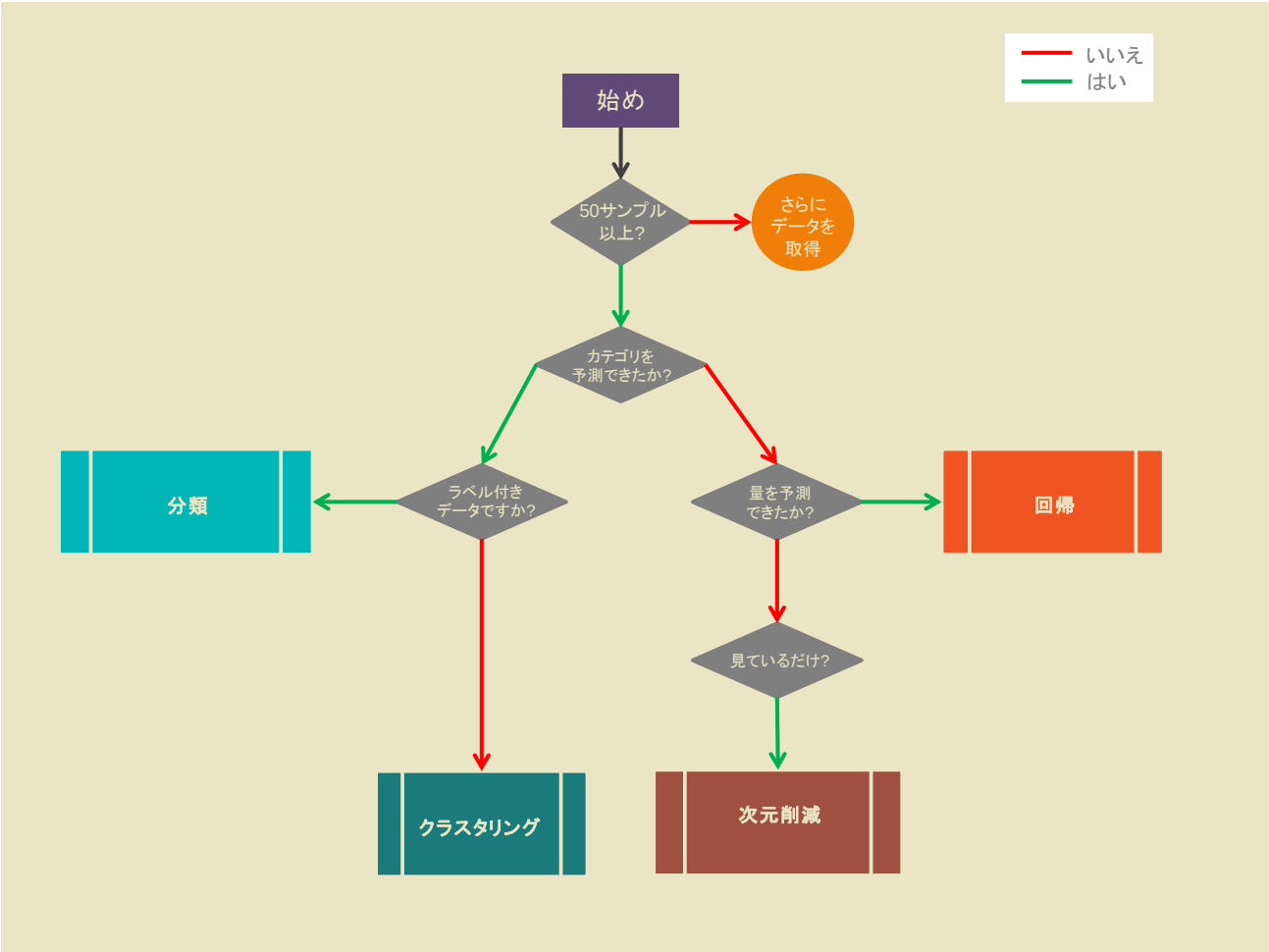

図 14: 機械学習フローチャート

図 14 は、データ分析を始めるとき、どのクラスの学習アルゴリズムを使用するかを決めるための基本的なフローチャートを示している。

分類ツールと予測ツールを評価するための測定基準

分類ツールと予測ツールは、以下の 5 つの一般的なパラメータで評価できる:

- **正確性**: 与えられた分類と推測が、新しいデータあるいは以前は見えていなかったデータ（すなわち、クラスラベル情報のない順組[78]）のクラスラベルと価値を正しく予測するための能力
- **速さ**: 与えられた分類あるいは推測を生成し使用するのに必要な計算できるコスト
- **堅牢性**:ノイズの多いデータや値が不足したデータを、正しい予測にするための分類あるいは予測の能力
- **拡張性**:与えられた大量のデータを効率的に分類あるいは予測する能力
- **解釈可能性**: 分類あるいは予測（解釈可能性は主観的なので評価するのは難しい）によって提供される理解と洞察のレベル

[78] Tuples: 順祖。複数の構成要素からなる組

与えられたテストセットの分類ツールの正確性は、正しく分類された順組の割合（これらラベルありのテストセットの順組は、分類ツール[79]を訓練強化するために使用すべきではない）になる。同様に、予測ツールの正確性は、所与の予測が、新しい若しくは以前見えていなかったデータのために予測された属性の値を、どれくらいよく推測できるかを示している。分類ツールの**誤り率**[80]あるいは**分類誤りレート**[81]とは、単純に正しく分類されなかった順組の残っている割合である。

m クラスの分類ツールにおいて、混同行列[82]は *m x m* のテーブルとなる。テーブルの中の *CM_i_j* エントリーは、クラス *j* にラベル付けされたクラス *i* の順組の数を示している。分類ツールの正確性が高い場合、テーブルの非対角線のエントリーの大部分はほとんどゼロになる。混同行列からは、正確性や感度のような別の測定基準で計算されることができる。

予測ツールに関して、正確性は測定基準によって計算される。これは、1 つのテストセットの二乗平均平方根誤差[83]のようなものである。正確性は、トレーニングセットから独立した 1 つ以上のテストセットを使用することで見積もることができる。クロス確認[84]やブートストラップ法[85]などの見積り技術に関しては、議論する必要がある。

学習者が、トレーニングデータでは 100%の正確性を示すが、テストデータでは 50%の正確性しか示さない予測ツールを出力する場合、両方で 75%の正確性になるとすると、学習者は過剰適合であると言う。**過剰適合**[86]は、バイアスおよび変化を使用して測定できる。バイアスは、同様の間違いから学ぶという学習者の傾向である。他方、変化は、本当かどうかに関わりなくランダムに学ぶ傾向である[38]。 .

学習者のための汎化誤差[87]は、二乗されたバイアスと変化の合計として表現できる。従って、同時にバイアスの最小化と変化の最小化の間の汎化誤差を最小にする場合、トレードオフが存在する。汎化誤差が、変化を最小にすることによって最小にされる場合、バイアスは高くなる可能性があり、極端に**過少適合**[88]となる。バイアスが低いが変化が高い場合、過剰適合になる。

可視化

最も一般的に使用されているビッグデータの可視化技術は、以下の 3 つのカテゴリに広く分類できる（図 15）。

空間レイアウト可視化 - このクラスの可視化技術は、データオブジェクトを座標空間上のある一点にマッピングする形式を参照する。このような技術の主な動機としては、空間基盤として組織された情報を容易に解釈す

[79] classifier
[80] **error rate**
[81] **misclassification rate**
[82] confusion matrix
[83] the root mean squared error
[84] cross-validation
[85] bootstrapping
[86] **Overfitting**
[87] generalization error
[88] **underfitting**

るための人間の認識能力がある。一般的に使用されている空間レイアウト可視化技術は、折れ線グラフ、棒グラフ、散布図などである。しかしながら、このようなグラフィックは、しばしばデータの複雑な関係が想像できないために制限されることがある。複雑な関係に関する一つの例は、データオブジェクトでの階層構造の存在であり、しばしばツリーマップ[89][21]を使用して可視化される。その他、一般的な空間レイアウト可視化技術の例は、グラフあるいはネットワークレイアウト可視化であり、ノードとエッジの存在がデータからのおもしろい洞察につながる。 力指向グラフ描画アルゴリズム[90][22]は、可視化アルゴリズムの 1 つの例である。

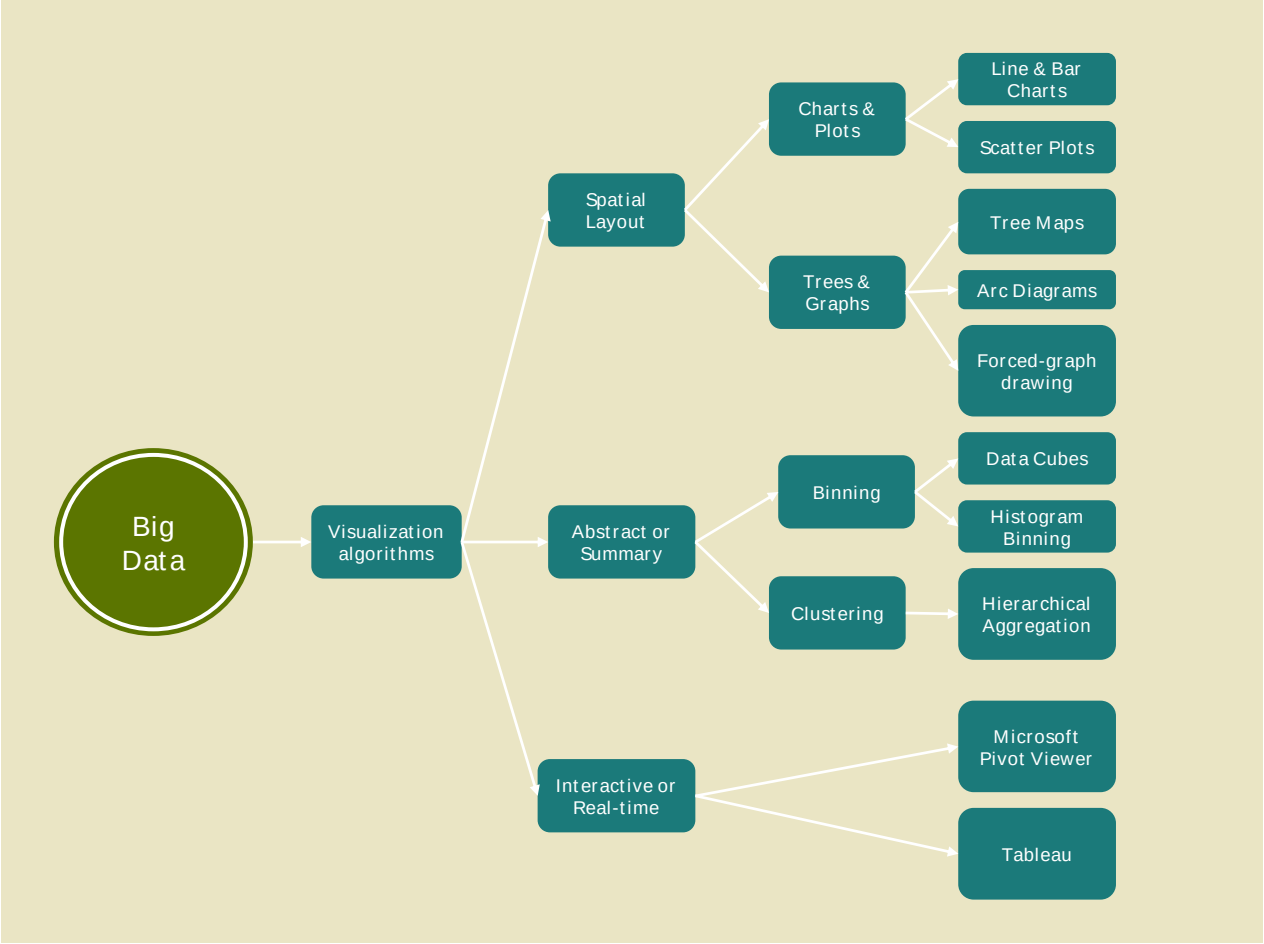

図 15: 可視化技術のための分類

抽象化/集約可視化[91] -ビッグデータ分析法は、重要な相関関係を発見できるようになる前に、データが拡張性を持って処理される（例えば、何十億レコード、テラバイトのデータ）ことが必要である。既存の可視化技術をこのレベルに拡張することは、重要なタスクになる。新しい類型の可視化技術が最近提案された。それは、大規模データを可視化のルーチン[23]に委ねる前に、処理し、抽象化し、集約するものである。この技術は、インタラクティブな可視化法の分類に入る。データ抽象化の一般的な例は、ヒストグラムあるいはデータキューブとしてグループ分けされる。これらは、コンパクトかつデータの 次元表現を削減するという点で、優位性を持っている。

[89] treemaps
[90] Force-directed graph drawing algorithm
[91] **Summary Visualization**

インタラクティブ/リアルタイム可視化 - 最近の技術の類型は、インタラクティブ可視化に分類されており、ユーザとの相互関係をリアルタイムに行わなければならない。このような技術では、複雑な可視化メカニズムにより、ユーザによるデータのリアルタイムナビゲーションを 1 秒以内に実行することが必要になる[24]。この技術により、ユーザは素早くデータの中にある重要な洞察を見つけ出すことができ、その洞察上の異なったデータサイエンス理論を証明できるまたは証明できないという意味で非常に強力である。このような技術は、また、データ指向の洞察を非常に信頼している業界にとって重要である。今日、Microsoft の Pivot Table や Tableau のような多くの業界向けソフトウェアがインタラクティブ可視化のために同様の戦略を採用している。

セキュリティとプライバシー

ビッグデータのためのセキュリティとプライバシーの課題は、図 16 に示されているように、ビッグデータエコシステムの 4 つの見方に整理できるであろう：

1. インフラストラクチャセキュリティ
2. データプライバシー
3. データ管理
4. 完全性とリアクティブなセキュリティ

ビッグデータシステムのインフラストラクチャをセキュアにするためには、分散化した計算処理とデータストアをセキュアにすることが必要である。データ自体をセキュアにすることが最も重要なので、情報の配布がプライバシーを確実に保護し、機微なデータが暗号化や粒度の高いアクセス制御を介して確実に保護される必要がある。膨大な容量のデータの管理には、データストアをセキュアにするだけでなく、データの由来に対する効率的な監査や調査を可能にする、拡張性があり分散化されたソリューションが必要である。最後に、多様なエンドポイントから出現するストリーミングデータについては、完全性をチェックし、セキュリティインシデントのためのリアルタイム分析を実行して、インフラストラクチャの健全性を保証するために利用できるようにしておく必要がある。

ここでは、今まで議論してきた様々な形式のデータに関するセキュリティ問題について議論する。

ストリーミングデータ -データが公開されているか否かによって、ストリーミングデータのための 2 つの派生的なセキュリティ問題が存在する。 公開されているデータは、機密性について問題ないかもしれないが、政府機関のように個々のクライアントに適用されるフィルタリングの基準によって分類される可能性がある。個人的なデータは、機密性が心配であるかもしれないが、同時に、データの適切に変更されたバージョンが、予測分析[92]のような特定の有益性を実現するために公開される可能性がある。

「Private Searching On Streaming Data」[25]で、Ostrovsky と Skeith は、ストリーミングデータ上のプライベート検索の問題を検討している。その中で、彼らは、様々な暗号の仮定の下で秘密の基準（例えば、隠れたキーワードの隠れた組合せの有無）を満たす文書の検索を効率的に実装している。彼らのスキームでは、クライアン

[92] predictive analytics

トが、機密基準に関する不明瞭な変換をフィルターノードに送信することができる。フィルターノードは、入力データに対して不明瞭な基準を適用し、結果として暗号化されフィルターされたメッセージになるが、クライアントしか復号することができない。これは、フィルタリングされた実際のデータだけでなく評価基準を効果的に隠す。

ストリーミング時系列データの連続した性質は、このようなデータの機微な側面を隠す際に、特別な技術的課題となる。特に、データポイントの単純で無作為な摂動は、厳格なプライバシー保証を提供しない。「Time series compressibility and privacy」[26]において、Papadimitriou 等は、時系列の圧縮性および部分的な情報隠蔽の間のトレードオフ、また、個人に関する不確実性の導入におけるこのトレードオフの意味合いを研究している。

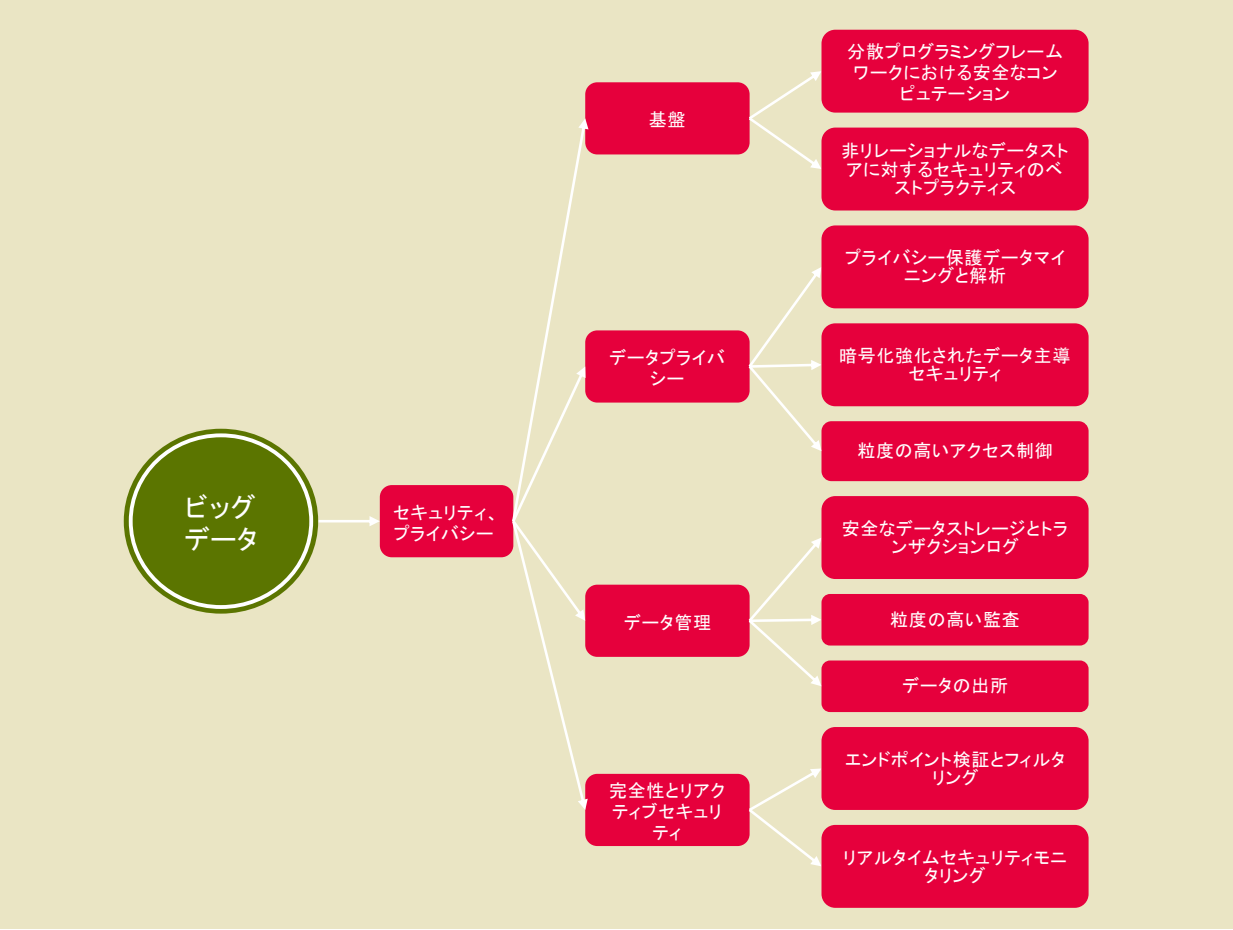

図 16: セキュリティとプライバシーの分類

グラフデータ -「Private analysis of graph structure」[27]において、Karwa 等は、厳格なプライバシーの保証を提供する一方で、グラフデータについて役に立つ統計を公表するための効率的なアルゴリズムを提示している。社会的絆または電子メールコミュニケーションのような個人の間の関係を反映するデータセットに、彼らのアルゴリズムは取り組んでいる。アルゴリズムは、どのような特定の関係の有無も効果的に隠す。具体的には、アルゴリズムは、クエリを数える部分グラフに対するおよその答えを出力した。三角形または星のようなクエリグラフがあれば、ゴールはクエリグラフと同一構造のコピーの数を入力グラフの中に返す。

科学的 -科学的なデータは、多くの場合秘密にしておく必要はない。にもかかわらず、データの完全性、特にリモートセンサノードから来るデータを保証しなければならない。 軽量で短い署名は、文字通り提案される

べきで、そのようなタスクに使用することができる。一方、デジタル署名に関する膨大な文献があるが、いくつかの例は[28], [29], [30]で、短い署名の生成に適合している。

しかしながら、多くの科学的なアプリケーションにおいてセンサはオープンな環境に展開されるため、物理的な攻撃に対して脆弱であり、センサの持つ暗号化鍵を破壊される可能性がある。[31]で、著者は大規模なセンサーネットワークに安全な情報を集約するためのフレームワークを提案している。それは、攻撃者によって鍵が破壊されることに対応している。そのフレームワークにおいて、アグリゲーターと呼ばれるセンサーネットワークの特定のノードに、クエリによって要求された情報を集約する。効率的な無作為抽出メカニズムとインタラクティブ証明を作ることにより、ユーザは、アグリゲーターとセンサーノードの一部が破損した時でさえ、アグリゲーターによって与えられる答えが正確な値の良い近似値であることを確かめることができる。

ウェブ -多くの標準的な暗号プロトコルが、ウェブ上の安全な通信として展開されている。いくつかの例としては、TLS、ケルベロス、OAuth、PKI、Secure BGP、DNSSEC（Secure DNS）、IEEE802 シリーズプロトコル、IPSec、Secure Re-routing in Mobile IPv6 がある。

最近、ビッグデータ分析法は、セキュリティインテリジェンスやフォレンジックのためにウェブサーバによって生成されたログを分析することを可能にした。NIST Special Publication 800-92 [32]において、著者は組織に対してログデータのためのベストプラクティスを推奨している。ネットワーク化されたサーバ、ワークステーション、その他のコンピューティングデバイスの広範囲にわたる配備のため、そして、ネットワークとシステムに対する脅威の増加のため、コンピュータセキュリティログの数、量、多様性は大いに増加している。この増加は、コンピュータセキュリティログ管理の必要性を確立した。そして、それは、コンピュータセキュリティログデータを生成し、送信し、保存し、分析し、廃棄するプロセスである。一般的には、組織がポリシーとログ管理の手順を確立し、組織を通して適切にログ管理を優先順位付けし、ログ管理基盤を作成し維持し、ログ管理責任のあるすべてのスタッフに対する適切なサポートを提供し、標準的なログ管理運用プロセスを構築することが推奨される。

小売と金融データ -小売や金融の個人データは個人的性質の高いものなので、データを保存して送信するベストプラクティスに常に従うべきである。既存のプラクティスのいくつかが、データにアクセスする実体の適切な承認と認証を保証するために、停止中、移動中、インフラストラクチャ内においてデータを暗号化したままにすべきである。金融業界のための PCI DSS などのコンプライアンス規格は、セキュリティのベストプラクティスと法的必要条件を提供している。

データの高容量化の到来は、非常に多くの解析技術を可能にしている。それは、製品に適切なデモグラフィックスを与えようとするサードパーティ組織にとって高い価値の情報を生み出す。実際、データは、明らかに特定を可能にする識別子を匿名化と集約のプロセスによって確実に削除した後、共有される。このプロセスはその場その場のもので、しばしば経験則 [33]に頼っており、公開されているデータ[34]と紐付けることで非匿名化が可能となる事例がしばしばある。個人情報を持っていない解析の共有は、実際に最終目標として考えられる。しかしながら、サードパーティや研究機関は、しばしばそれ自身のデータの研究および解析を行おうとする。そのため、利用価値があり正確なデータの抽出は、引き続き課題となる。

プライバシー保護データ公開を扱ったいくつかの形式的モデルが提案されている [35]。最も強力なモデルの 1 つは、差分プライバシー[93] [36]のフレームワークである。「GUPT: privacy preserving data analysis made easy」[37]において、著者は、プログラムに対して差分プライバシーを保証し、プライバシーを考慮した開発を行わないという GUPT と呼ばれているシステムのデザインと評価を提示している。GUPT は、時間とともにデータのプライバシーを低下させるデータの機微性のモデルを使用する。このことにより、プライバシーを全体的に一定のレベルで保証し、各々のアプリケーションの有用性を最大にしている一方で、異なるユーザアプリケーションのために異なるレベルのプライバシーの効果的な配置を可能にする。

結論

ここでは、6 つの最も重要な次元に沿ってビッグデータの分類の序論を示した。 6 つの次元は、データ領域、計算処理基盤、ストレージアーキテクチャ、分析法、可視化、セキュリティとプライバシーである[94]。ビッグデータ基盤と方法論は、速いペースで進化し続けているが、ベースにしている基盤技術は多くの場合数年前に考案されたものである。人間行動とマシンツーマシン通信のデジタル化の著しい増大は、大規模で安価なハードウェアと組み合わされて、水平かつ分散型の計算処理という以前からあるアカデミックな考えを実用化し、新しく実世界のアプリケーションに役に立つように取り入れられている。

参考文献

[1] IBM Study, “StorageNewsletter » Every Day We Create 2.5 Quintillion Bytes of Data,” 21-Oct-2011. http://www.storagenewsletter.com/rubriques/market-reportsresearch/ibm-cmo-study/.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, Berkeley, CA, USA, 2004, pp. 10–10.

[3] L. Valiant, “A bridging model for parallel computation,” *CACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[4] “Welcome to Apache™ Hadoop®!” http://hadoop.apache.org/.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, 2003, pp. 29–43.

[6] J. Lin and C. Dyer, *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool, 2010.

[7] “Hadoop Tutorial - YDN.” http://developer.yahoo.com/hadoop/tutorial/module4.html#dataflow.

[8] “Breaking Down ‘Big Data’ – Database Models – SoftLayer Blog.” http://blog.softlayer.com/2012/breaking-down-big-data-database-models/.

[9] “CAP theorem,” *Wikipedia, the free encyclopedia*. 16-Aug-2013.

[10] “CAP Twelve Years Later: How the ‘Rules’ Have Changed.” http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed.

[11] “Google Research Publication: BigTable.” http://research.google.com/archive/bigtable.html.

[12] “Amazon’s Dynamo - All Things Distributed.” http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html.

[13] “Graph database,” *Wikipedia, the free encyclopedia*. 21-Aug-2013.

[93] Differential Privacy

[94] 原文は、” The six dimensions are data domains, compute infrastructure, storage architectures, analytics, visualization, security and privacy, and data domains.”となっていますが、最後の”data domains”は重複となるのでタイプミスと思われます。

[14] "New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps." http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext.
[15] Max De Marzi, "Introduction to Graph Databases," 29-Apr-2012. http://www.slideshare.net/maxdemarzi/introduction-to-graph-databases-12735789.
[16] "Thumbtack Technology - Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs." http://thumbtack.net/whitepapers/ultra-high-performance-nosql-benchmark.html.
[17] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, "Solving Big Data Challenges for Enterprise Application Performance Management," *Proc VLDB Endow*, vol. 5, no. 12, pp. 1724–1735, Aug. 2012.
[18] "Big Data Benchmark." https://amplab.cs.berkeley.edu/benchmark/.
[19] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Burlington, MA: Elsevier, 2012.
[20] R. Tibshirani, "Glossary of Machine learning and statistical terms." Stanford University, 2012.
[21] B. Bederson and et. al., "Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies." https://www.researchgate.net/publication/220184592_Ordered_and_quantum_treemaps_Making_effective_use_of_2D_space_to_display_hierarchies.
[22] "Force-directed graph drawing - Wikipedia, the free encyclopedia." http://en.wikipedia.org/wiki/Force-directed_graph_drawing.
[23] B. Shneiderman, "Extreme Visualization: Squeezing a Billion Records into a Million Pixels." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.2521.
[24] Z. Liu and et. al., "Stanford Vis Group | imMens: Real-time Visual Querying of Big Data." http://vis.stanford.edu/papers/immens
[25] R. Ostrovsky and et. al., "Private Searching On Streaming Data." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.157.1127.
[26] S. Papadimitriou and et. al., "Time series compressibility and privacy." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.7150.
[27] V. Karwa and et. al., "Private analysis of graph structure." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.227.8815.
[28] D. Boneh and et. al., "Short Group Signatures." http://crypto.stanford.edu/~dabo/abstracts/groupsigs.html.
[29] P. Gaborit and et. al., "Lightweight code-based identification and signature." http://citeseer.uark.edu:8080/citeseerx/viewdoc/summary;jsessionid=410E9FDC4CC2BF40183250734B93BE7D?doi=10.1.1.131.3620.
[30] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *Advances in Cryptology – ASIACRYPT 2001*, C. Boyd, Ed. Springer Berlin Heidelberg, 2001, pp. 514–532.
[31] B. Przydatek and et. al., "SIA: secure information aggregation in sensor networks." http://dl.acm.org/citation.cfm?id=958521.
[32] K. Kent and et. al., "Guide to Computer Security Log Management. SP 800-92." http://dl.acm.org/citation.cfm?id=2206303.
[33] L. Sweeney, "k-anonymity: a model for protecting privacy." http://dataprivacylab.org/dataprivacy/projects/kanonymity/kanonymity.html.
[34] A. Narayanan and et. al., "Robust De-anonymization of Large Sparse Datasets." http://dl.acm.org/citation.cfm?id=1398064.
[35] B. Fung and et. al., "Privacy-Preserving Data Publishing: A Survey on Recent Developments." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.150.6812.
[36] C. Dwork, "Differential Privacy." http://research.microsoft.com/apps/pubs/default.aspx?id=64346.
[37] P. Mohan and et. al., "GUPT: privacy preserving data analysis made easy." http://dl.acm.org/citation.cfm?id=2213876.
[38] P. Domingos, "A few useful things to know about machine learning." http://dl.acm.org/citation.cfm?id=2347755.
[39] "Spark | AMPLab – UC Berkeley," AMPLab - UC Berkeley. https://amplab.cs.berkeley.edu/publication/.
[40] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, Berkeley, CA, USA, 2010, pp. 10–10.

