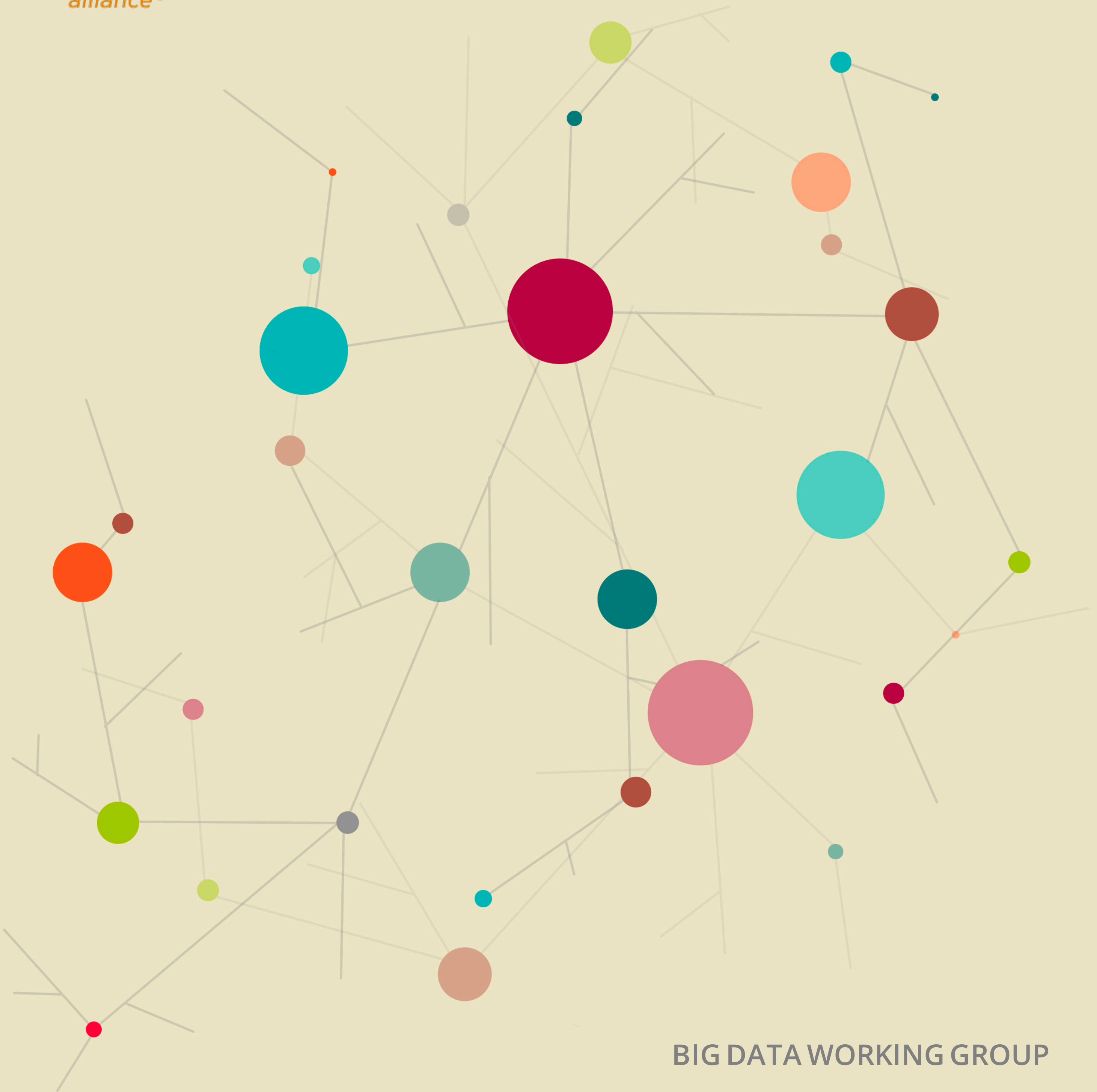

BIG DATA WORKING GROUP

Big Data Taxonomy

September 2014

© 2014 Cloud Security Alliance – All Rights Reserved

All rights reserved. You may download, store, display on your computer, view, print, and link to the Cloud Security Alliance "Big Data Taxonomy" paper at https://cloudsecurityalliance.org/research/big-data/, subject to the following: (a) the Document may be used solely for your personal, informational, non-commercial use; (b) the Document may not be modified or altered in any way; (c) the Document may not be redistributed; and (d) the trademark, copyright or other notices may not be removed. You may quote portions of the Document as permitted by the Fair Use provisions of the United States Copyright Act, provided that you attribute the portions to the Cloud Security Alliance "Big Data Taxonomy" (2014).

Acknowledgements

Contributors

Praveen Murthy

Anurag Bharadwaj

P. A. Subrahmanyam

Arnab Roy

Sree Rajan

Commentators

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

Design/Editing

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

Abstract

In this document, we propose a six-dimensional taxonomy for big data. The main objective of this taxonomy is to help decision makers navigate the myriad choices in compute and storage infrastructures as well as data analytics techniques, and security and privacy frameworks. The taxonomy has been pivoted around the nature of the data to be analyzed.

© 2014 Cloud Security Alliance - All Rights Reserved.

Table of Contents

Acknowledgements....3
Abstract....3
Introduction....5
Data....6
Compute Infrastructure....10
Storage Infrastructure....17
Analytics....22
Visualization....27
Security and Privacy....29
Conclusion....31
References....32

© 2014 Cloud Security Alliance - All Rights Reserved.

Introduction

The term big data refers to the massive amount of digital information companies and governments collect about us and our surroundings. This data is not only generated by traditional information exchange and software use via desktop computers, mobile phones and so on, but also from the myriads of sensors of various types embedded in various environments, whether in city streets (cameras, microphones) or jet engines (temperature sensors), and the soon-to-proliferate Internet of Things, where virtually every electrical device will connect to the Internet and produce data.

Every day, we create 2.5 quintillion bytes of data--so much that 90% of the data in the world today has been created in the last two years alone (as of 2011 [1]). The issues of storing, computing, security and privacy, and analytics are all magnified by the velocity, volume, and variety of big data, such as large-scale cloud infrastructures, diversity of data sources and formats, streaming nature of data acquisition and high volume inter-cloud migration.

The six-dimensional taxonomy is shown in Figure 1. These six dimensions arise from the key aspects that are needed to establish a big data infrastructure. We will describe each of the dimensions in the rest of the document.

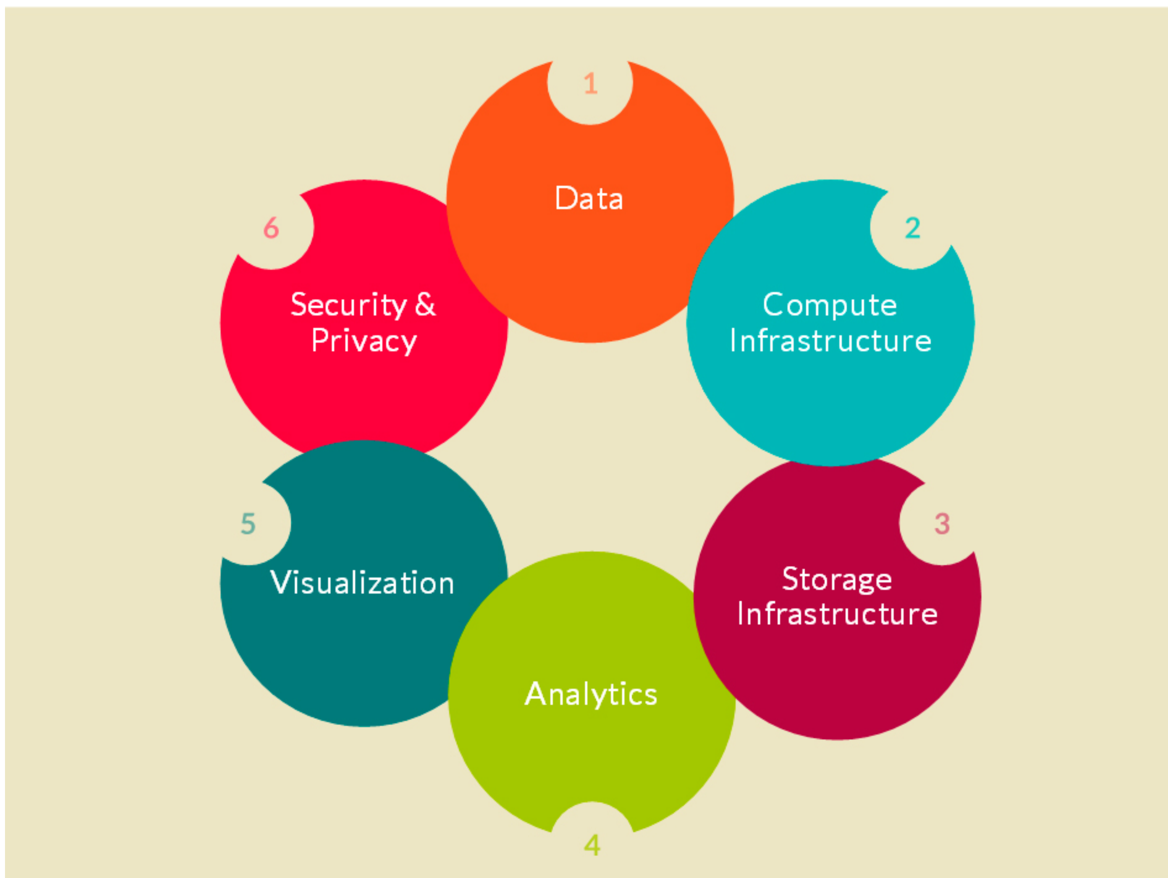

Figure 1: Big data 6-D taxonomy

© 2014 Cloud Security Alliance - All Rights Reserved.

Data

The first question: What are the various domains in which big data arise? The reason for categorizing the domains in which data arise is in order to understand the infrastructural choices and requirements that need to be made for particular types of data. All "data" is not equivalent. The particular domain in which data arises will determine the types of architecture that will be required to store it, process it, and perform analytics on it. There are several ways in which we can think about this question of data domains.

Latency Requirements

The first way to characterize data would be according to time span in which it needs to be analyzed:

- **Real-time** (financial streams, complex event processing (CEP), intrusion detection, fraud detection)
- **Near real-time** (ad placement)
- **Batch** (retail, forensics, bioinformatics, geodata, historical data of various types)

Examples of "Real-Time" Applications

Some of the many applications that involve data arriving "in real-time" include the following:

- On-line ad optimization (including real-time bidding)
- High frequency online trading platforms
- Security event monitoring
- Financial transaction monitoring and fraud detection
- Web analytics and other kinds of dashboards
- Churn prediction for online games or e-commerce
- Optimizing devices, industrial plants or logistics systems based on behavior and usage
- Control systems related tasks; e.g., the SmartGrid, nuclear plants
- Sentiment analysis of tweets pertaining to a topic

In most of these applications, data is constantly changing. To react to certain events, it is necessary and/or practical to consider only relevant data over a certain time frame ("page views in the last hour" or "transactions in the last hour/day/week/month…"), instead of taking the entirety of past data into account.

Key Attributes of Real-Time Applications Impacting Big Data Technology Solutions

In order to select the appropriate approach and big data technology solution that is best suited to a problem at hand, it is important to understand some of the key attributes that impact this decision. In addition to latency requirements (the time available to compute the results), these could include the following:

- Event Characteristics
    - Including input/output data rate required by the application.
- Event Response Complexity
    - Processing complexity:
        - What is the computational complexity of the processing task for each event?
    - Data Domain Complexity:
        - What is the size of the data that has to be accessed to support such processing?

© 2014 Cloud Security Alliance - All Rights Reserved.

- Does it fit in memory, or is it scattered over multiple locations and storage media?

As might be expected, event response complexity that is high in both the compute and data domain aspects, when coupled with high input/output data rates and low latency requirements poses the most severe challenges on the underlying infrastructure.

Latency is the time available between when an input event occurs and the response to that event is needed. Stated differently, it is the time that it takes to perform the computation that leads to a decision.

There are two broad categories that we consider here: low and high latency requirements.

- We will here define "low latency" applications to be those that require a response time on the order of a few tens of milliseconds. For applications like high frequency trading and real-time bidding or on-line ad optimization, there is an upper limit on the latency that is acceptable in the context of the application. Often, this is on the order of 20-50 milliseconds for on-line ad optimization systems, with high frequency trading systems being potentially even more stringent in terms of real-time response required. While the specific latencies are a function of the application and infrastructure, and will evolve over time, applications that we include in this category are those requiring a "real-time" response.
- We define "medium to high latency" applications to be those that need a response time on the order of a few seconds to minutes, to potentially a few hours. For example, in the context of applications that involve user interaction and dashboards, it is normally acceptable if the results are updated every few seconds or even every few minutes. Most forms of reporting and longer duration data analysis can tolerate latencies on the order of several minutes, and sometimes even hours or days.

The real question here is whether the application can (or has to) react in real-time. If data comes in at 100k events per second, but the main business action is taken by a manager who manually examines the aggregate data once every few days to adjust some business strategy, then low latency is not a key business issue. On the other hand, if a process control system is being driven by data from a set of sensors, or an enterprise is deploying a new landing page for a website and the objective is to detect any sudden drop in the number of visitors, a more immediate response is needed.

As shown in Figure 2 below, the overall latency of a computation at a high level is comprised of communication network latency, computation latency, and database latency. The precise budget for each of the three broad categories (network, compute, and database) depends highly on the application. Compute-intensive applications will leave less room for network and database operations.

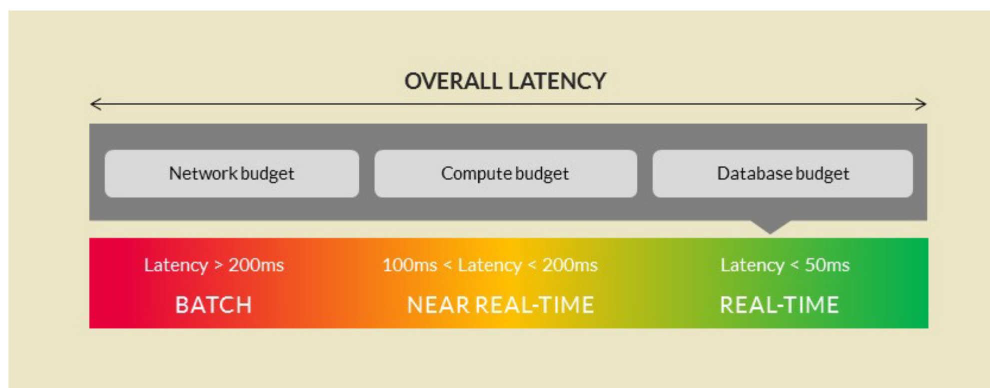

Figure 2: Characterization of latency requirements

© 2014 Cloud Security Alliance - All Rights Reserved.

Big Data Technology Solutions for Real-Time Applications

When considering an appropriate big data technology platform, one of the main considerations is the latency requirement. If low latency is not required, more traditional approaches that first collect data on disk or in memory and then perform computations on this data later will suffice. In contrast, low latency requirements generally imply that the data must be processed as it comes in.

Structure

Another way to map various domains for big data is in the degree of structure or organization they come with.

- **Structured** (retail, financial, bioinformatics, geodata)
- **Semi-structured** (web logs, email, documents)
- **Unstructured** (images, video, sensor data, web pages)

It is useful to think of data as being structured, unstructured or semi-structured. We provide some examples below, with the caveat that a formal definition that precisely delineates these categories may be elusive.

Structured Data

Structured data is exemplified by data contained in relational databases and spreadsheets. Structured data conforms to a database model, which is largely characterized by the various fields that data belongs to (name, address, age and so forth), and the data type for each field (numeric, currency, alphabetic, name, date, address). The model also has a notion of restrictions or constraints on each field (for example, integers in a certain range), and constraints between elements in the various fields that are used to enforce a notion of consistency (no duplicates, cannot be scheduled in two different places at the same time, etc.)

Unstructured Data

Unstructured Data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a predefined manner. Unstructured information is typically text-heavy, but may also contain data such as dates, numbers, and facts. Other examples include the “raw” (untagged) data representing photos and graphic images, videos, streaming sensor data, web pages, PDF files, PowerPoint presentations, emails, blog entries, wikis, and word processing documents.

Semi-Structured Data

Semi-structured data lies in between structured and unstructured data. It is a type of structured data, but lacks a strict structure imposed by an underlying data model. With semi-structured data, tags or other types of markers are used to identify certain elements within the data, but the data doesn’t have a rigid structure from which complete semantic meaning can be easily extracted without much further processing. For example, word processing software now can include metadata showing the author's name and the date created, while the bulk of the document contains unstructured text. (Sophisticated learning algorithms would have to mine the text to understand what the text was about, because no model exists that classifies the text into neat categories). As an additional nuance, the text in the document may be further tagged as including table of contents, chapters, and sections. Emails have the sender, recipient, date, time and other fixed fields added to the unstructured data of the email message content and any attachments. Photos or other graphics can be tagged with keywords such as the creator, date, location and other

© 2014 Cloud Security Alliance - All Rights Reserved.

content-specific keywords (such as names of people in the photos), making it possible to organize and locate graphics. XML and other markup languages are often used to manage semi-structured data.

Yet another way to characterize the domains is to look at the types of industries that generate and need to extract information from the data

- Financial services
- Retail
- Network security
- Large-scale science
- Social networking
- Internet of Things/sensor networks
- Visual media

Figure 3 illustrates the various domains and specific subdomains in which big data processing issues arise.

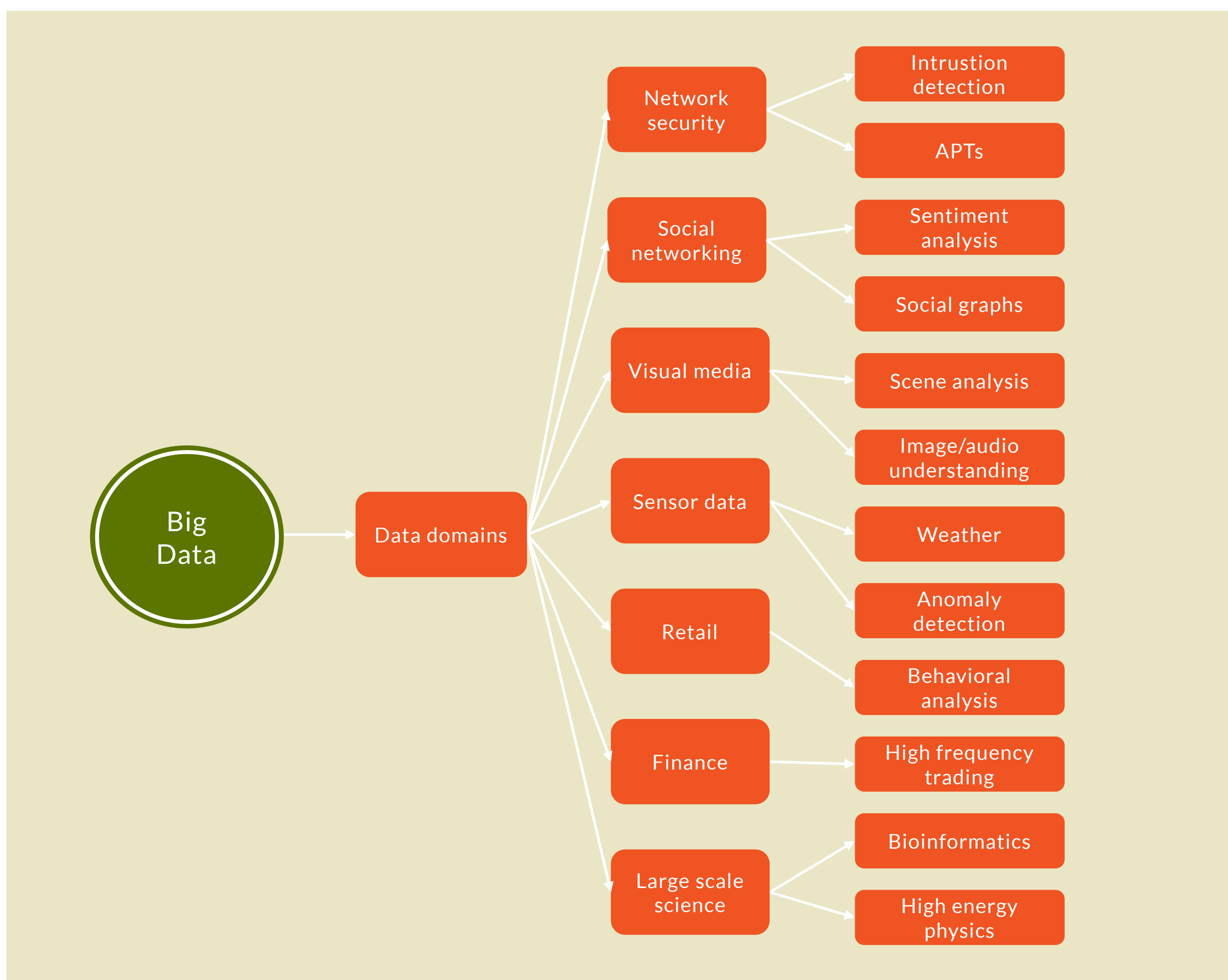

Figure 3: Data domains

Figure 4 illustrates how the big data verticals map to the time and organization axes. A case can be made that in fact all of the industries included here have use cases that encounter data at all levels of organization, and have processing needs that span all response times. In that case, the industry domain is another orthogonal axis for characterizing the

© 2014 Cloud Security Alliance - All Rights Reserved.

domain space of big data. We would then visualize these domains by particular common use cases, and map them to industry, time, and structure.

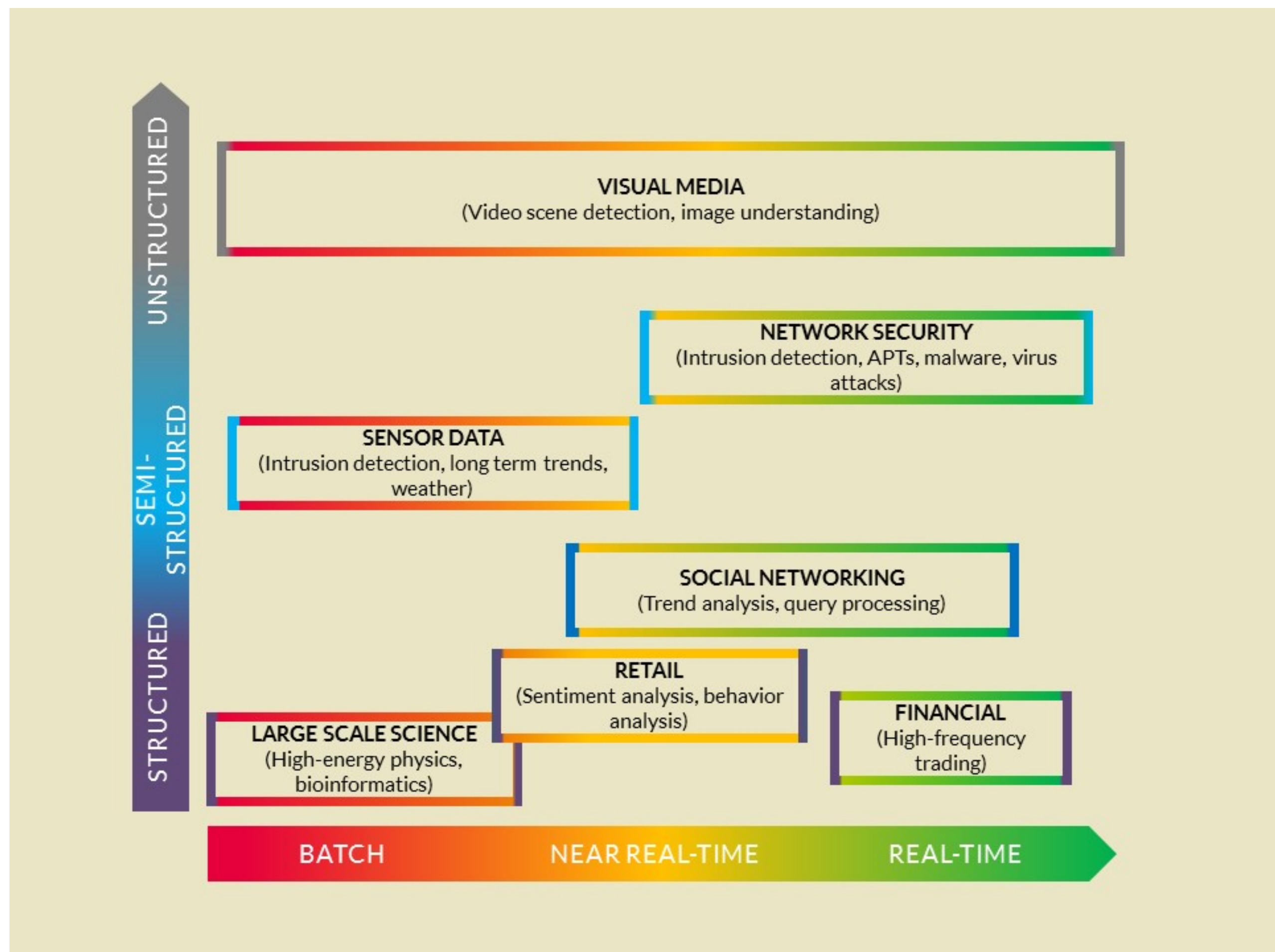

Figure 4: Mapping the big data verticals

Compute Infrastructure

While the Hadoop ecosystem is a popular choice for processing large datasets in parallel using commodity computing resources, there are several other compute infrastructures to use in various domains. Figure 5 shows the taxonomy for the various styles of processing architectures. Computing paradigms on big data currently differ at the first level of abstraction on whether the processing will be done in batch mode, or in real-time/near real-time on streaming data (data that is constantly coming in and needs to be processed right away). In this section, we highlight two specific infrastructures: Hadoop for batch processing, and Spark for real-time processing.

MapReduce is a programming model and an associated implementation for processing and generating large datasets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper referenced in [2].

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's

© 2014 Cloud Security Alliance - All Rights Reserved.

execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to utilize the resources of a large distributed system easily.

Bulk synchronous parallel processing [3] is a model proposed originally by Leslie Valiant. In this model, processors execute independently on local data for a number of steps. They can also communicate with other processors while computing. But they all stop to synchronize at known points in the execution; these points are called barrier synchronization points. This method ensures that deadlock or livelock problems can be detected easily.

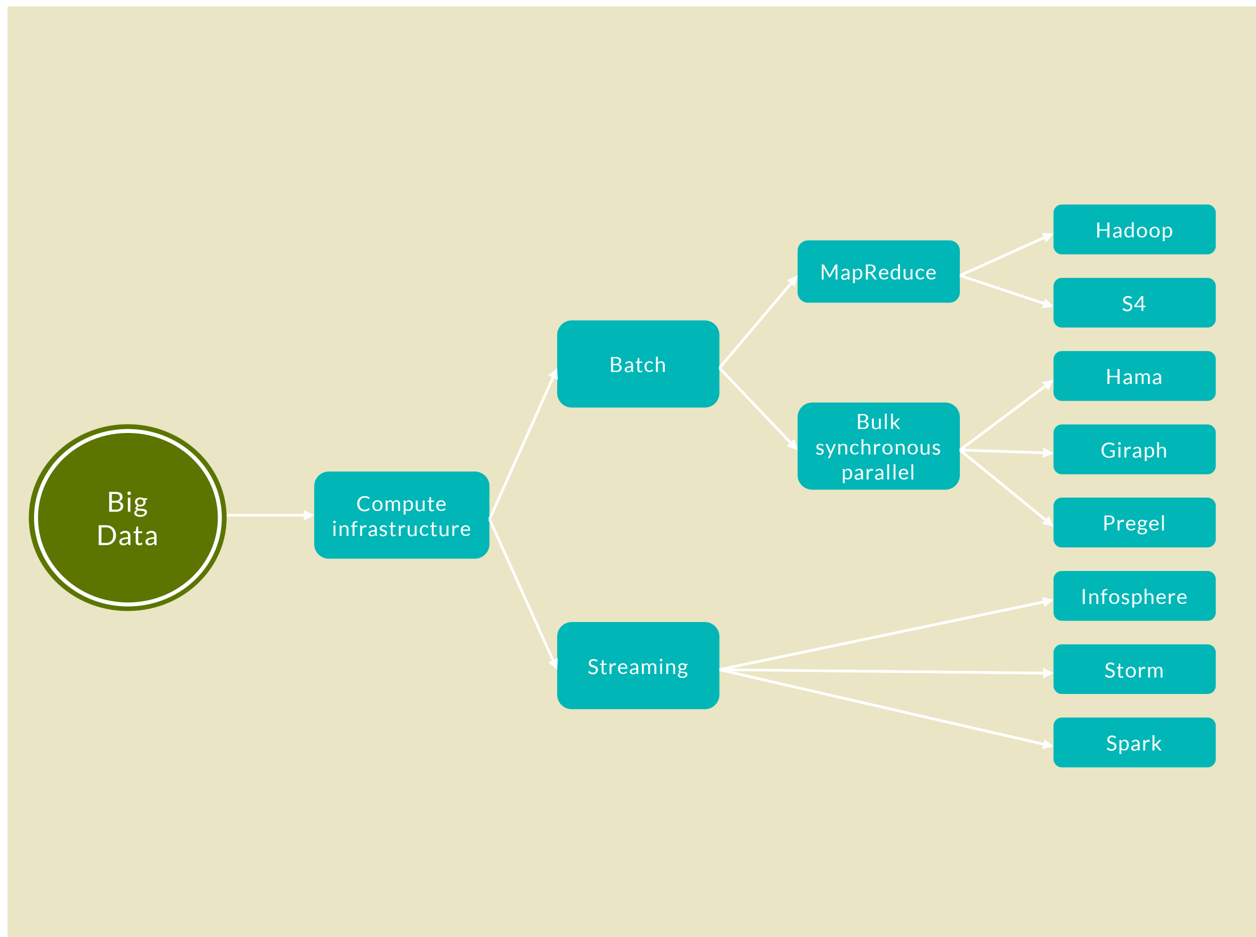

Figure 5: Compute infrastructure

Low Latency: Stream Processing

If an application demands "immediate" response to each event as it occurs, some form of stream processing is needed, which essentially processes the data as it comes in. The general approach is to have a little bit of code that processes each of the events separately. In order to speed up the processing, the stream may be subdivided, and the computation distributed across clusters.

Apache Storm is a popular framework for event processing that was developed at Twitter and promulgated by Twitter and other companies that required this paradigm of real-time processing. Other examples are Amazon's Kinesis, or the streaming capabilities of MapR. These frameworks take care of the scaling onto multiple cluster nodes and come with varying degrees of support for resilience and fault tolerance, for example, through checkpointing, to make sure the system can recover from failure.

© 2014 Cloud Security Alliance - All Rights Reserved.

These stream processing frameworks primarily address only parallelization of the computational load; an additional storage layer is needed to store the results in order to be able to query them. While the "current state" of the computation is contained in the stream processing framework, there is usually no clean way to access or query this information, particularly from external modules. Depending on the amount of data that is being processed, this might further entail the need for a large, high-performance storage backend.

Apache Spark (discussed in more detail below), simplistically speaking, takes a hybrid approach. Events are collected in a buffer and then processed at fixed intervals (say every few seconds) in a batch fashion. Since Spark holds the data in memory, it can in principle process the batches fast enough to keep up with the incoming data stream.

In summary, low latency processing generally entails some form of stream processing, with associated infrastructure for computation and storage. It is important to note that if the application requirements are extreme (for example, if sub-millisecond latencies are needed), then traditional software stacks may not be good enough for the task. Specialized software stacks or components may be have to be custom built for the application.

High Latency: Batch Processing

If the application context can tolerate high latency (for example, it does not require that the results be generated within a few seconds, or even minutes), a batch-oriented computing approach can be adopted.

In the simplest example, the application can scan through log files to do what is needed. Alternatively, all of the data can be put into a database after which the application queries this data to compute desired results. Databases used here can be classical SQL databases, or pure storage databases such as Cassandra, or databases that can also run aggregation jobs, such as CouchDB.

Batch processing can be scaled effectively using frameworks such as Apache Hadoop (discussed in more detail below), provided that the underlying processing can be cast into a map-reduce paradigm. Log data can be stored in a distributed fashion on the cluster. The application can then run queries in a parallel fashion to reduce response times.

As Hadoop has matured, a number of projects have evolved that build on top of Hadoop. One such example is Apache Drill, a vertical database (or column-oriented database) similar to Google's Dremel on which BigQuery is based. Vertical databases are optimized for tasks in which you have to scan whole tables and count entries matching some criterion. Instead of storing the data by row as in traditional databases, data is stored by column. So instead of storing data as in a log file, one line per entry, one takes each field of the data and stores it together, resulting in much better IO characteristics. HP Vertica and ParStream are other vertical databases.

Some projects and products have started to replace the disk underlying the database by memory (or a combination of flash and memory) as a storage medium, most notably SAP Hana but also GridGain and Apache Spark to get around the disk speed limitation. These systems are still essentially batch processing in nature, although turnaround times between queries can be reduced considerably. Another example of a high performance solution that leverages flash based memory systems is Aerospike.

Hadoop 1.0

Hadoop [4] is the open source distributed programming and storage infrastructure that grew out of Google's seminal MapReduce [2] and Google file system [5] papers. It is based on the paradigm of "map reduce" computing, where the input to a computational task is first "mapped" by splitting it across various worker nodes that work on subsets of the input independently, and a "reduce" step where the answers from all of the map sub-problems are collected and

© 2014 Cloud Security Alliance - All Rights Reserved.

combined in some manner to form the output of the overall computational task (Figure 6). The data stored can either be in the Hadoop filesystem as unstructured data, or in a database as structured data. Because Hadoop is designed to work on problems whose input data is very large and cannot fit in the disk size of a single computer, the MapReduce paradigm is designed to take computation to where the data is stored, rather than move data to where the computation occurs. The framework is also designed to be highly fault tolerant via data replication, and to have an architecture that keeps track of the worker nodes' progress via polling, and reassigns tasks to other nodes if some of the nodes should fail. In addition, the framework will automatically partition the "map" and "reduce" computations across the compute/storage network. All of that is done automatically by the Hadoop runtime, and the developer has only to write the map and reduce routines for the computational task at hand.

The entire Hadoop ecosystem includes the following projects:

- **Pig** – A platform that provides a high-level language for expressing programs that analyze large datasets. Pig is equipped with a compiler that translates Pig programs into sequences of MapReduce jobs that the Hadoop framework executes.
- **Hive** – A data-warehousing solution built on top of the Hadoop environment. It brings familiar relational-database concepts, such as tables, columns, and partitions, and a subset of SQL (HiveQL) to the unstructured world of Hadoop. Hive queries are compiled into MapReduce jobs executed using Hadoop.
- **HBase** – A column-oriented NoSQL data-storage environment designed to support large, sparsely populated tables in Hadoop.
- **Flume** – A distributed, reliable, available service for efficiently moving large amounts of data as it is produced. Flume is well-suited to gathering logs from multiple systems and inserting them into the Hadoop Distributed File System (HDFS) as they are generated.
- **Lucene** – A search-engine library that provides high-performance and full-featured text search.
- **Avro** – A data-serialization technology that uses JSON for defining data types and protocols, and serializes data in a compact binary format.
- **ZooKeeper** – A centralized service for maintaining configuration information and naming, providing distributed synchronization and group services.
- **Oozie** – A workflow scheduler system for managing and orchestrating the execution of Apache Hadoop jobs.

© 2014 Cloud Security Alliance - All Rights Reserved.

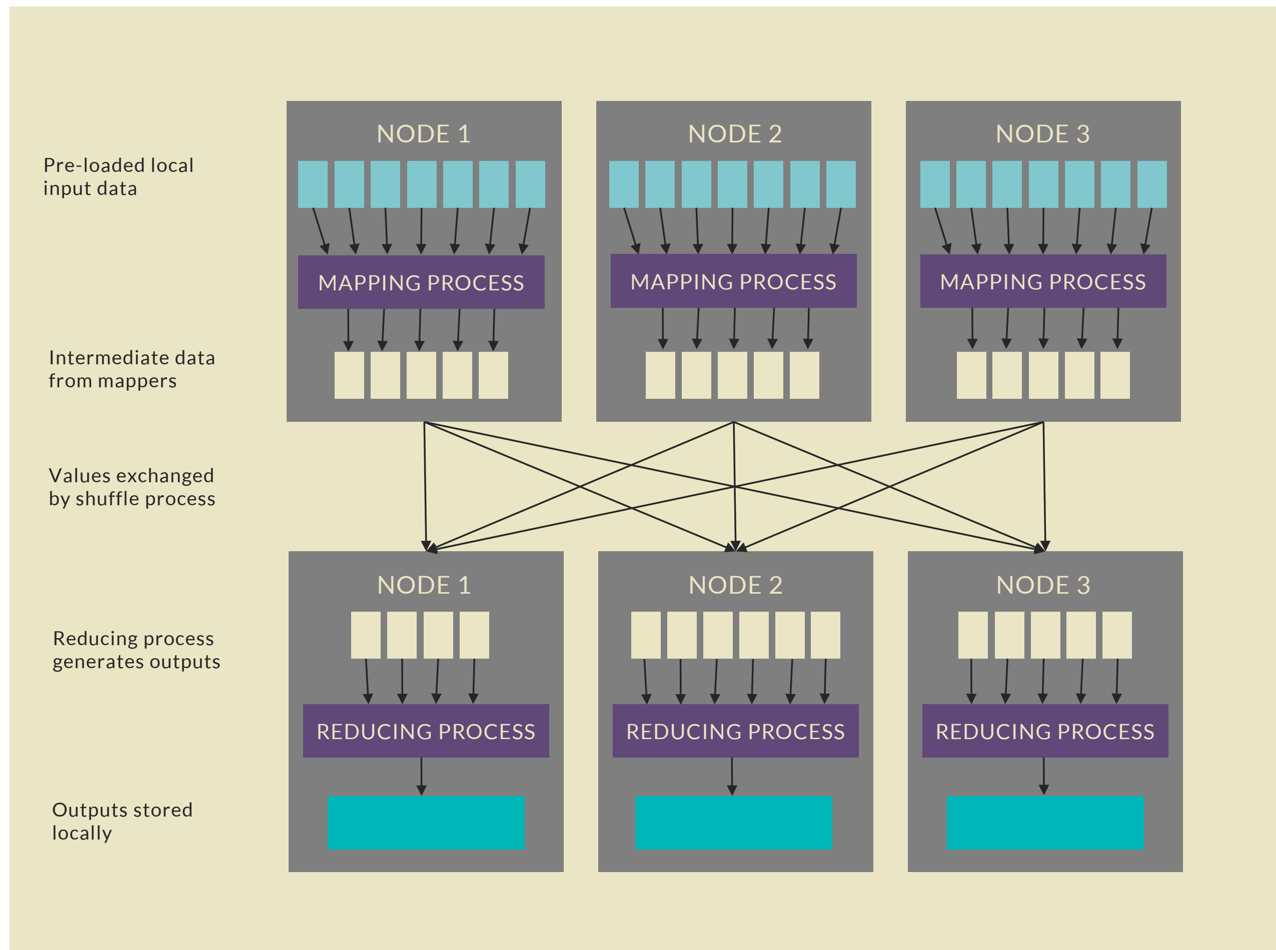

Figure 6: Hadoop MapReduce flow [7]

While Hadoop is good for batch processing, it is generally considered unsuitable for data streams that are non-terminating. This is because a Hadoop job assumes that all of the data exists in files on various nodes, and will start its Map and Reduce phases on a fixed amount of input to generate a fixed amount of output. For streaming applications, where there is a steady stream of data that never stops, this model is clunky and suboptimal. Adding new map tasks dynamically to process newly arrived inputs (while potentially removing the processing of old data as any system for processing streams of data has to work on sliding windows) creates too much overhead and too many performance penalties.

Hadoop is also not suitable for algorithms that are iterative and depend on previously computed values. This class of algorithms includes many types of machine learning algorithms that are critical for sophisticated data analytics, such as online learning algorithms [6].

Hadoop is also unsuitable for algorithms that depend on a shared global state, since the entire MapReduce model depends on independent map tasks running in parallel without needing access to a shared state that would entail severe performance bottlenecks due to locks, semaphores, and network delays. An example of where this occurs is in Monte-Carlo simulations which are used to perform inferences in probabilistic models [6].

Figure 7 shows how the various elements of the Hadoop ecosystem fit together.

© 2014 Cloud Security Alliance - All Rights Reserved.

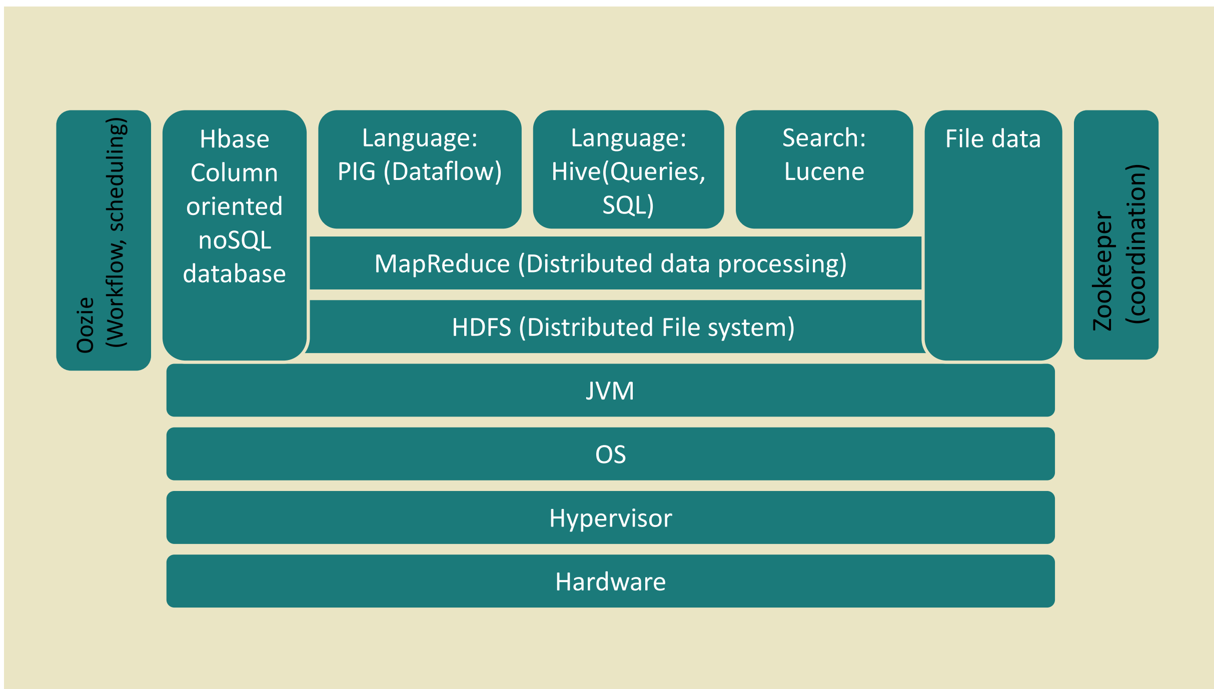

Figure 7: Hadoop 1.0 stack

Hadoop 2.0

Because of these issues, a new Hadoop 2.0 has been developed (Figure 8). Crucially, this framework decouples HDFS, resource management, and MapReduce programming, and introduces a resource management layer called YARN that takes care of the lower level resources. An application in Hadoop 2.0 can now deploy its own application-level scheduling routines on top of the Hadoop-managed storage and compute resources.

© 2014 Cloud Security Alliance - All Rights Reserved.

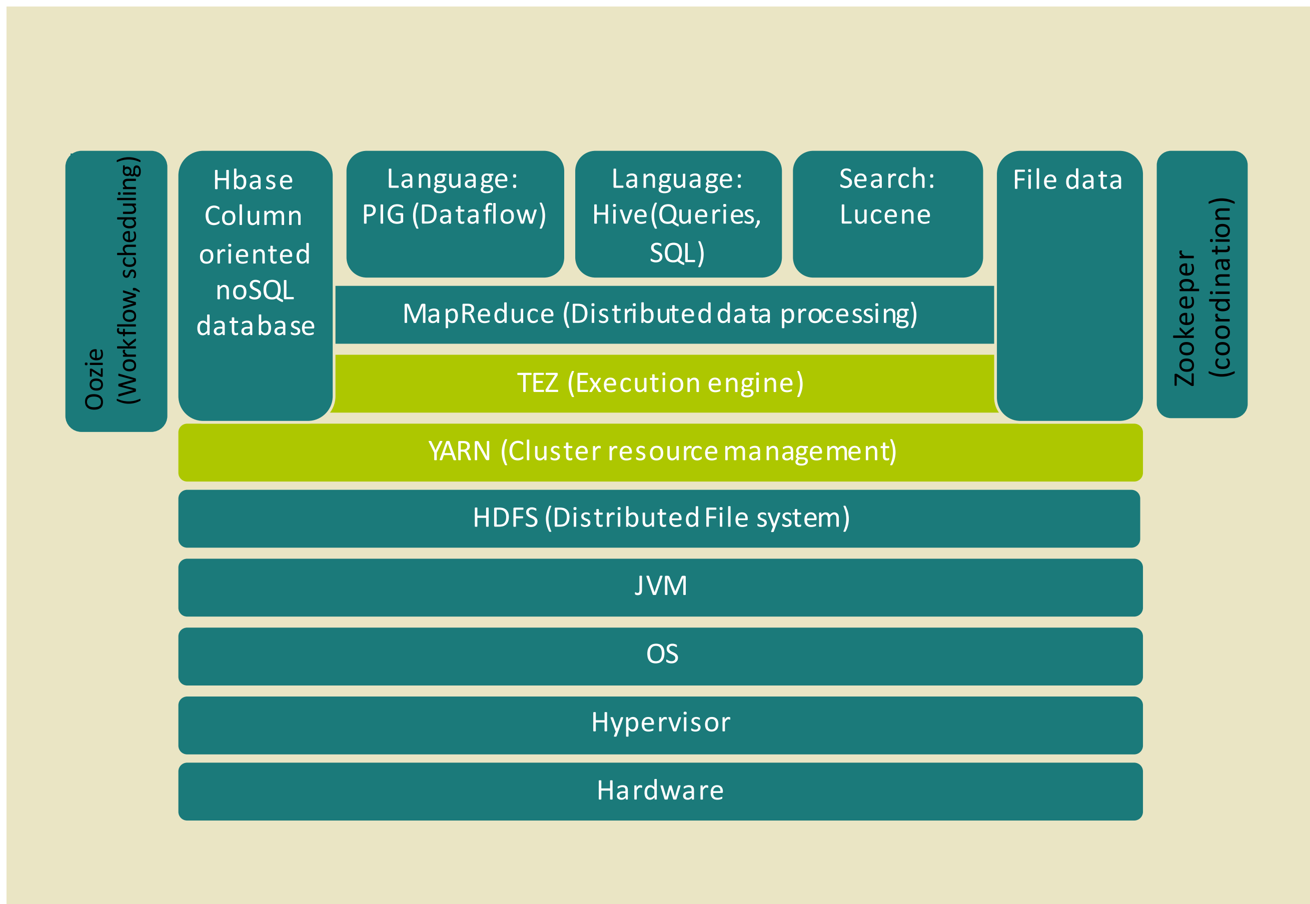

Figure 8: Hadoop 2.0 stack with YARN layer for resource management and TEZ for execution

Berkeley Spark [39,40]

Spark is an open source cluster computing system invented at the University of California at Berkeley that is aimed at speeding up data analytics – both in runtime and in development. To run programs faster, Spark provides primitives for in-memory cluster computing: a job can load data into memory and query it repeatedly much more quickly than with disk-based systems such as Hadoop MapReduce. Spark is also intended to unify the processing stack, where currently batch processing is done using MapReduce, interactive queries using HBase, and the processing of streams for real-time analytics using other frameworks such Twitter's Storm. These three stacks are difficult to maintain for consistent metrics. Also, it is difficult to perform interactive queries on streaming data. The unified Spark stack is designed to handle these requirements efficiently and scalably.

A key concept in Spark is the resilient distributed dataset (RDD), which is a collection of objects spread across a cluster stored in RAM or disk. Applications in Spark can load these RDDs into the memory of a cluster of nodes and let the Spark runtime automatically handle the partitioning of the data and its locality during runtime. This enables fast iterative processing. A stream of incoming data can be split up into a series of batches and processed as a sequence of small-batch jobs. The Spark architecture allows this seamless combination of streaming and batch processing in one system.

To make programming faster, Spark provides clean, concise APIs in Scala, Java, and Python. Spark can be used interactively from the Scala and Python shells to rapidly query big data sets. Spark was initially developed for two applications where keeping data in memory helps: iterative machine learning algorithms and interactive data mining. In both cases, Spark has been shown to run up to **100x** faster than Hadoop MapReduce.

© 2014 Cloud Security Alliance - All Rights Reserved.

Spark is also the engine behind Shark, a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive. While Spark is a new engine, it can access any data source supported by Hadoop.

Spark fits in seamlessly with the Hadoop 2.0 ecosystem (Figure 9) as an alternative to MapReduce, while using the same underlying infrastructure such as YARN and the HDFS. The GraphX and MLlib libraries include sophisticated graph and machine learning algorithms that can be run in real-time. BlinkDB is a massively parallel, approximate query engine for running interactive SQL queries that trades off query accuracy for response time, with results annotated by meaningful error bars. BlinkDB has been demonstrated to run 200x faster than Hive within an error of 2-10%.

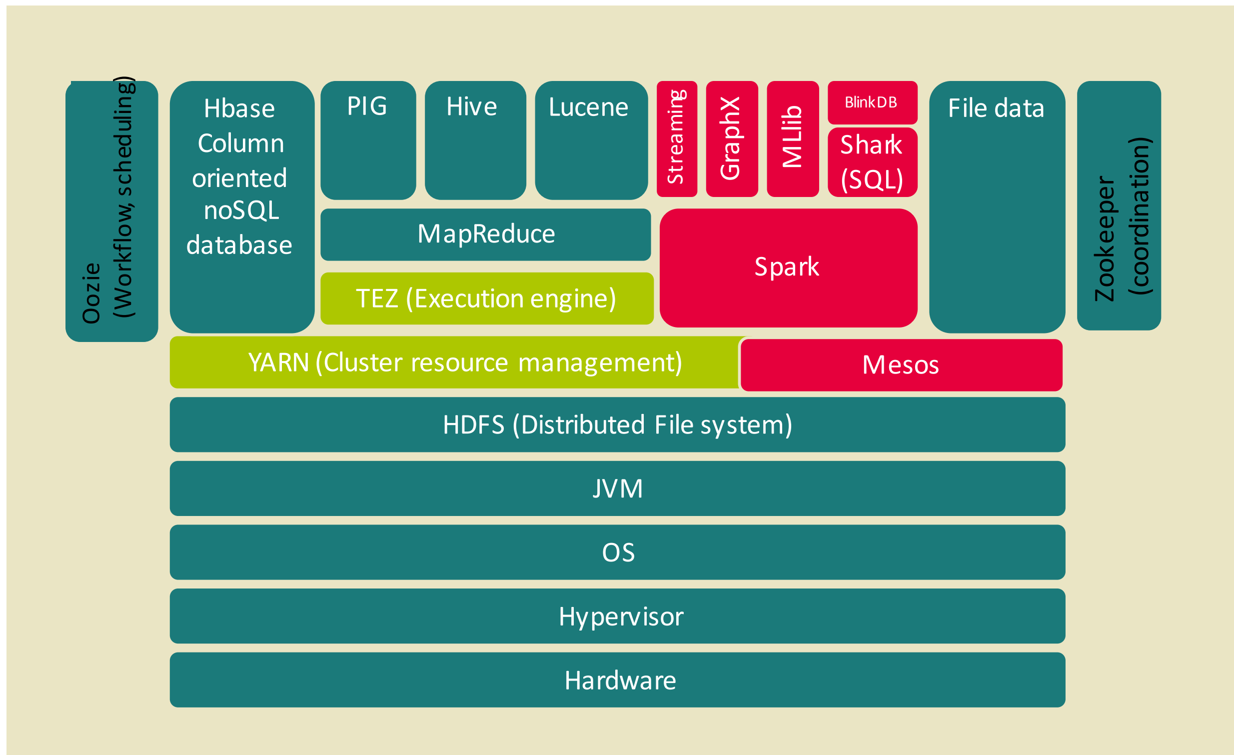

Figure 9: Spark in the Hadoop 2.0 ecosystem

Storage Infrastructure

Large volumes of data are coming at a much faster velocity in varieties of formats such as multimedia and text that don't easily fit into a column-and-row database structure. Given these factors, many solutions have been created that provide the scale and speed developers need when they build social, analytics, gaming, financial or medical apps with large datasets [8]. Figure 10 shows the taxonomy for the various types of databases that are used for big data storage.

© 2014 Cloud Security Alliance - All Rights Reserved.

Figure 10: Storage infrastructure

In order to scale databases here to handle the volume, velocity, and variety of data, we need to scale horizontally across multiple servers rather than scaling vertically by upgrading a single server (by adding more RAM or increasing HDD capacity). But scaling horizontally implies a distributed architecture in which data resides in different places. This setup leads to a unique challenge shared by all distributed computer systems: The CAP Theorem [9]. According to the CAP theorem, a distributed storage system must choose to sacrifice either **consistency** (that everyone sees the same data) or **availability** (that you can always read/write) *while* having **partition tolerance** (where the system continues to operate despite arbitrary message loss or failure of part of the system). Note that partition tolerance is not an option as partitions in a distributed system happen when nodes crash (for whatever reason) or the network drops an arbitrary number of packets (due to switch failures or other reasons). When the inevitable "partition" happens and a distributed system has parts that cannot communicate with each other, the question is whether a distributed system is going to favor consistency (meaning it will respond only to queries that satisfy consistency, and not respond when consistency cannot be guaranteed, meaning that availability is sacrificed) or whether availability will be favored (meaning all queries are answered even if some are inconsistent).

A standard metric by which databases are judged is by their **ACID properties**:

- **Atomicity** requires that each transaction be all or nothing. If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.
- **Consistency** ensures that each transaction will bring the database from one valid state to another.
- **Isolation** ensures that concurrent execution of transactions results in a system state that would be obtained if the transactions were executed serially.

© 2014 Cloud Security Alliance - All Rights Reserved.

- **Durability** means that once a transaction is committed it will remain so, even in the event of power loss, crashes, or errors.

The ACID focus on consistency is the traditional approach taken by relational databases. In order to handle the needs of internet and cloud based models of storage, a design philosophy at the other end of the spectrum was coined by Eric Brewer [10], and called BASE: Basically Available, Soft state, Eventually consistent. Most NoSQL databases are based on BASE principles, while choosing C or A in the CAP theorem.

The following table summarizes the most common database models, by their strengths, and their CAP theorem compromise of consistency versus availability [8]:

	DATABASE TYPE						
	Relational	**Document**	**Key-value**	**Big Table-Inspired**	**Dynamo-Inspired**	**Graph**	**NewSQL**
WHAT THEY DO	Stores data in rows/columns. Parent-child records can be joined remotely on the server. Provides speed over scale. Some capacity for vertical scaling, poor capacity for horizontal scaling. This type of database is where most people start.	Stores data in documents. Parent-child records can be stored in the same document and returned in a single fetch operation with no join. The server is aware of the fields stored within a document, can query on them, and return their properties selectively.	Stores an arbitrary value at a key. Most can perform simple operations on a single value. Typically, each property of a record must be fetched in multiple trips, with Redis being an exception. Very simple and very fast.	Data put into column-oriented stores inspired by Google's BigTable paper [11]. It has tunable CAP parameters, and can be adjusted to prefer either consistency or availability. Both of these adjustments are operationally intensive.	Distributed key/value stores inspired by Amazon's Dynamo paper [12]. A key written to a dynamo ring is persisted in several nodes at once before a successful write is reported. Riak also provides a native MapReduce implementation.	Uses graph structures with nodes, edges, and properties to represent and store data. Provides index-free adjacency; This means that every element contains a direct pointer to its adjacent element and no index lookups are necessary. General graph databases that can store any graph are distinct from specialized graph databases such as triplestores and network databases [13].	Like relational, except these databases offer high performance and scalability while preserving traditional ACID notions. They are capable of high throughput online transaction processing requirements, while preserving the high-level language query capabilities of SQL [14].
HORIZONTAL SCALING	Horizontal scaling is possible via replication – sharing data between redundant nodes to ensure consistency – and some people have success sharding – horizontally	Horizontal scaling is provided via replication, or replication and sharding. Document-oriented databases also usually support relatively low-performance MapReduce for ad-hoc querying.	Horizontal scale is provided via sharding.	Good speed and very wide horizontal scale capabilities.	Usually provide the best scale and extremely strong data durability.	Poor horizontal scaling so far, except for Titan.	Provided through sharding.

	partitioning data – but those techniques add complexity.						
CAP BALANCE	Prefer consistency over availability.	Generally prefer consistency over availability.	Generally prefer consistency over availability.	Prefer consistency over availability	Prefer availability over consistency.	Prefer availability over consistency	Prefer consistency over availability.
WHEN TO USE	When you have highly structured data, and you know what you'll be storing. Great when production queries will be predictable.	When your concept of a "record" has relatively bounded growth, and can store all of its related properties in a single doc.	Very simple schemas, caching of upstream query results, or extreme speed scenarios (like real-time counters).	When you need consistency and write performance that scales past the capabilities of a single machine. Hbase in particular has been used with approximately (or about) 1,000 nodes in production.	When the system must always be available for writes and effectively cannot lose data.	When you need to store collections of objects that lack a fixed schema and are linked together by relationships. Reasoning about the data can be done via traversals in a graph database instead of complex queries in SQL.	When you want a scalable version of a relational database that handles SQL queries efficiently but can also scale horizontally and provide strong ACID guarantees
EXAMPLE PRODUCT	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

Table 1: Database types for big data

© 2014 Cloud Security Alliance - All Rights Reserved.

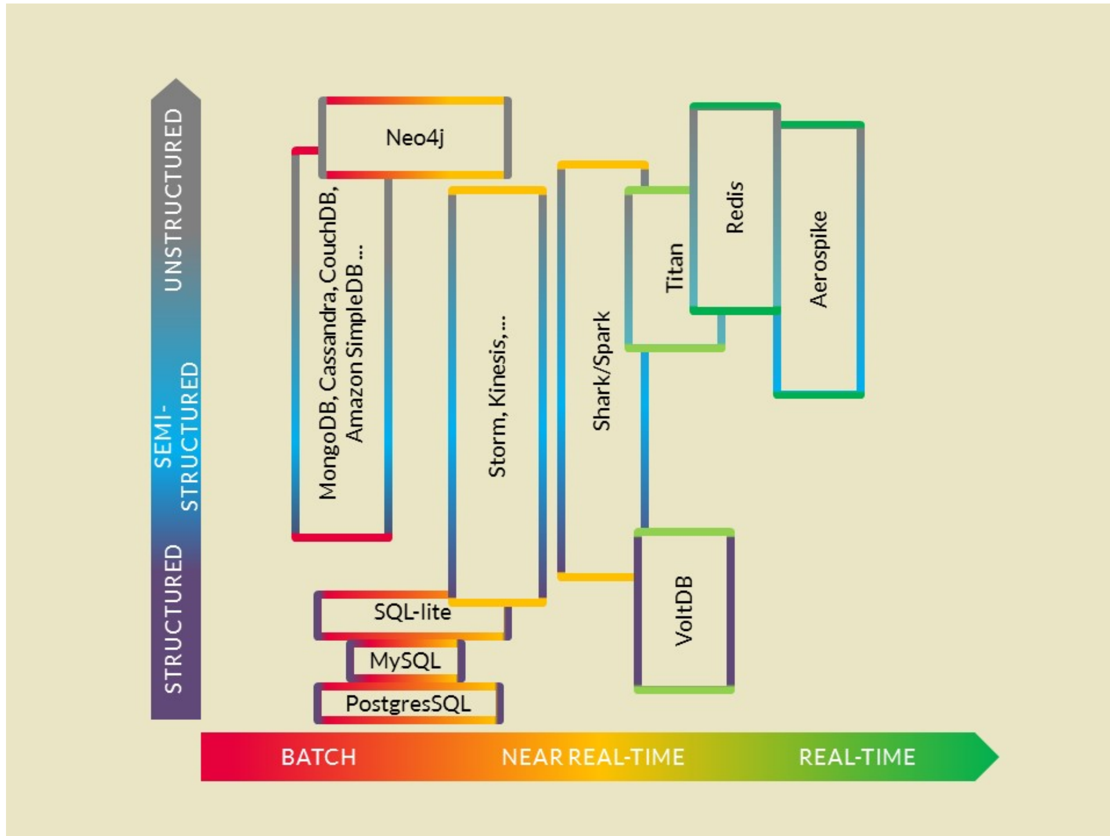

Figure 11: Storage technologies map

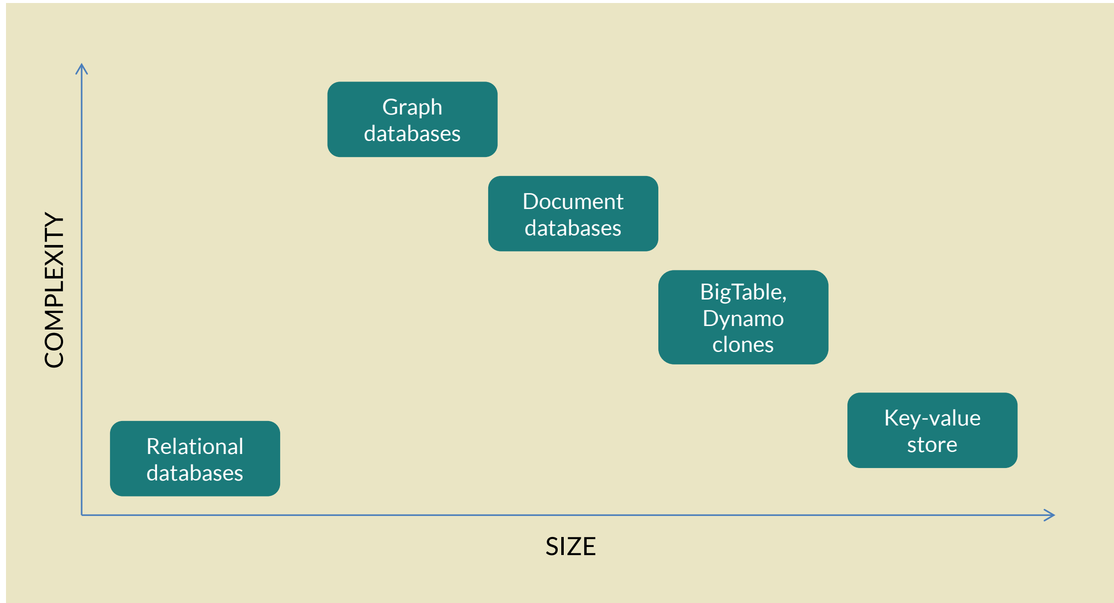

Figure 12: Data complexity vs amount of data various classes of databases can handle [15]

Performance Assessment and Representative Benchmarks

As discussed earlier in this section, over the past few years various types of database and NoSQL solutions have proliferated and differentiated themselves into key-value stores, document databases, graph databases, and NewSQL. Because of the different niches addressed by these solutions, trying to evaluate the database landscape for a particular class of problem is an important but increasingly difficult task.

© 2014 Cloud Security Alliance - All Rights Reserved.

Different databases use differing methods in how data is stored and accessed. Some databases, such as Couchbase and MongoDB, are intended to run on hardware where most of the working set is cached in RAM. Other databases, such as Aerospike, are optimized for direct writes to solid state disk (SSD), which would give them an advantage for benchmarks that involve lot of inserts. While in-memory databases have latency and throughput advantages, there may be scaling issues that come into play when large datasets are being handled. SSDs have higher densities and lower per-gigabyte costs than RAM, so it should be possible to scale much larger datasets on fewer nodes for SSD databases – valuable when talking about very large amounts of data. Scaling by adding more RAM-based machines may end up incurring a higher recovery-from-failure cost as a greater number of machines incur a higher rate of failures.

In addition to the differing technologies used for storing data and achieving fault-tolerance, benchmarks also need to contend with the fact that real-world usage will have all types of scenarios for the manner in which data has to be accessed and be reliable. In one benchmark's study, two common scenarios were examined: applications that have strong durability needs, in which every transaction must be committed to disk and replicated in case of node failure, and applications that are willing to relax these requirements in order to achieve the highest possible speed [16]. Another study examined the throughputs and latencies achievable in a variety of scenarios including read-only transactions, reads and writes, writes only and so forth [17]. Another study from AmpLab at UC Berkeley compared Shark with some representative databases in three different scenarios for specific types of queries [18].

In general, it is difficult to make definitive statements about performance in this fast evolving landscape. Performance is highly dependent on the query engine used, the storage architecture, and the manner in which data has been stored. Figure 11 is meant to be suggestive of broad categories of relative performance, and should not be read as a definitive ordering. Figure 12 shows how the various categories of databases compare regarding the complexity of the data items they store versus the size of the data they can handle.

Analytics

Machine Learning Algorithms

Machine learning techniques allow automatic and scalable ways in which insights from large, multi-dimensional data can be gleaned. Broadly, machine learning is the ability for computers to automatically learn patterns and make inferences from data. These algorithms can be classified along many different axes.

Algorithm Type

This classification is shown in Figure 13.

© 2014 Cloud Security Alliance - All Rights Reserved.

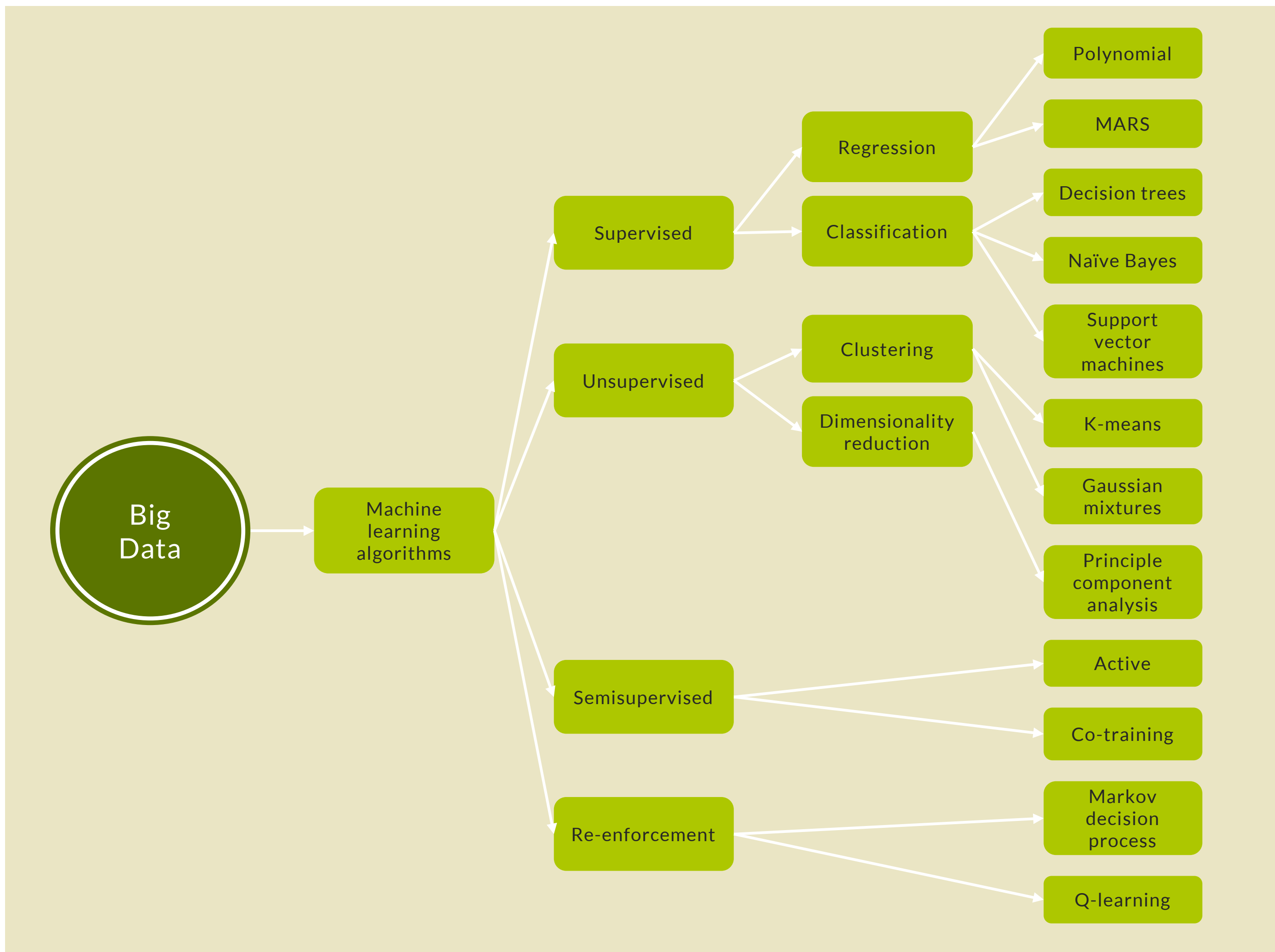

Figure 13: Machine learning algorithms

Supervised Learning – This category involves all machine learning algorithms that map input data to a given target value or class label(s). Commonly known algorithms include **classification,** which is the prediction of categorical labels (classes), and **regression/prediction**, which is the prediction of continuous-valued variables. In these algorithms, the learner approximates a mapping function from a feature vector to a set of classes (in the case of classification) or value (in the case of regression) over a large set of training data. The necessary condition for the training data is the presence of human labels for every data point. Such labeling is not entirely feasible for various big data applications in which it is expensive to obtain human labels for potentially millions of data points. However, a number of big data applications exist today that have successfully utilized smaller training datasets to achieve state-of-the-art performance on large, unlabeled real world datasets. Some of the most widely used tools in this category include the following for classification: Neural Networks, Decision Trees, Support Vector Machines, and Naïve Bayes; and the following for regression/prediction: Linear regression, Polynomial regression, Radial basis functions, MARS, and Multilinear interpolation.

Unsupervised Learning - This category involves all machine learning algorithms that learn the hidden structure of input data without requiring associated human labels. Commonly known algorithms include **clustering** and **source signal separation**. The necessary condition for the input data is the presence of representative features that can be exploited for meaningful knowledge discovery. This technique is especially suited to big data as applications have easy access to an abundance of large unlabeled datasets that can be processed with this learning framework. Some of the most widely

© 2014 Cloud Security Alliance - All Rights Reserved.

used tools in this category include K-Means Clustering, Gaussian Mixture Modeling, Spectral Clustering, Hierarchical Clustering, Principal Component Analysis and Independent Component Analysis.

Reinforcement Learning - This category involves all machine learning algorithms that learn a mapping function between observations and actions so as to maximize a reward function. The learning algorithm is typically optimized to take an action that will yield maximum reward over a period of time. Such learning algorithms, though popular in Robotics, have seen limited exploration in big data community. Two widely used tools in this category include Markov Decision Process and Q-Learning.

Semi-Supervised Classification - This category uses small amounts of labeled data and fuses this information with large, unlabeled datasets to approximate an appropriate learning algorithm. This class of algorithms is specifically interesting for the big data community as the invariable presence of large unlabeled datasets is not well exploited by traditional supervised learning algorithms. On the other hand, semi-supervised learning techniques exploit structural commonality between labeled and unlabeled data in an efficient manner to generalize the functional mapping over large datasets. A few subcategories of algorithms in this taxonomy include Generative Models, Graph-Based Models and Multi-View Models. Some of the most widely used tools in this category include Active Learning, Transfer Learning and Co-Training.

Data Mining by Variety

While the above classification is appropriate for simple, structured datasets, complex, unstructured datasets require and benefit from further characterization. There are six broad categories in which this data variety can be categorized, and the machine learning algorithms mentioned in the previous section can be adapted and/or strengthened in various ways in order to apply to these various types of unstructured datasets.

- **Time Series Data** – A vast majority of big data applications are sensitive to time, so applying scalable machine learning algorithms to time series data becomes an important task. Time series data are sequences of values or events obtained over repeated measurements of time, for instance, stock market data. Existing algorithms that can successfully model time series data include Hidden Markov Models, Markov Random Fields (Spatio-Temporal Modeling) and Conditional Random Fields. Scaling these algorithms to big data is an active research topic. Recently, researchers have successfully scaled a traditional Dynamic time warping (DTW) algorithm to trillions of data points.
- **Streaming Data** – Data here is constantly arriving, for instance, from remote sensors, retail transactions, surveillance systems, Internet traffic, and telecommunication networks. To handle this type of data requirement, machine learning algorithms have to be applied in an online fashion. Most machine learning algorithms require batch processing of data such as clustering that needs to look at whole data in one pass to learn meaningful clusters. Such techniques are not scalable to streaming data where it is computationally infeasible to store past data. Recently, a number of distributed machine learning algorithms have been proposed that approximate such offline algorithms to streaming data, including distributed k-means and distributed SVD. A few tools that implement these algorithms include Apache Mahout and Vowpal Wabbit. On the other hand, there are a few algorithms that address these challenges algorithmically such as Linear SVM, Kernel SVM, and Parallel Tree Learning.
- **Sequence Data** – Sequence data consists of sequences of ordered elements or events that are recorded with or without a concrete notion of time [19]. The analysis of sequential data arises in many different contexts, including retail data analysis (determining whether customers that buy one type of item are more likely to buy another type of item), or analysis of DNA and protein sequences. Machine learning algorithms used here frequently include Hidden Markov Models and Sequence alignment algorithms (such as BLAST for local alignment for DNA sequences).

© 2014 Cloud Security Alliance - All Rights Reserved.

- **Graph Data** – Many problems are naturally modeled as graphs, including problems that require the analysis of social networks, the analysis of the World Wide Web, the analysis of biological networks, and the analysis or synthesis of protein structures. Almost all large-scale graph mining algorithms can be efficiently represented in matrix format thereby necessitating large scale matrix solvers. Existing techniques that have been shown to perform at such large scale include Collaborative Filtering, Singular Value Decomposition (SVD), and Page Rank. Two tools that aim to address this challenge are Presto and GraphLab.
- **Spatial Data** – Spatial databases hold space-related data such as maps, medical imaging data, remote sensing data, VLSI chip layout data and so forth. Because of spatial relationships between data, the data cannot be assumed to be completely independent; learning algorithms can exploit this fact.
- **Multimedia Data** – This category includes images, videos, audio, and text markups. Mining algorithms for this category will include many digital signal processing techniques for image segmentation, motion vector analysis, and model construction.

The classification of data by variety can easily be mapped to different vertical segments of industry where specific use cases arise; for instance, financial with time series or social networking with graph data. Some categories may span multiple types of data; for instance, large-scale science may involve nearly all types of these data and mining algorithms.

Statistical Techniques

Machine learning is not the only paradigm for making sense of big data. Statistical techniques have been the standard way of analyzing data for a long time indeed. Some people have argued that the only difference between statistical techniques and machine learning techniques is terminology. Table 2 summarizes some of the common concepts that have been called by different names by these two communities (the machine learning community and the statistics community) [20]:

MACHINE LEARNING	STATISTICS
Network, Graphs	Model
Example/instance	Data point
Label	Response
Weights	Parameters
Feature	Covariate
Learning	Fitting/Estimation
Generalization	Test set performance
Supervised learning	Regression/Classification
Unsupervised learning	Density estimation, Clustering

© 2014 Cloud Security Alliance - All Rights Reserved.

Table 2 Glossary of terms in machine learning versus statistics

While there are many similarities and seeming re-inventions, it is also true that there are machine learning algorithms that do not involve probability or statistics at all; for instance, support vector machines and artificial neural networks. In addition, because big data analysis frequently involves not only large amounts of data, but also high dimensionality, computational issues are important considerations. Machine learning algorithms also have to focus on computational issues that traditional statistical formulations have ignored. Hence, in many ways, it could be argued that statistical techniques and paradigms are a subset of machine learning techniques. More philosophical discussions can be entered into vis-à-vis the statistician's interest in modeling and inference as compared to the machine learner's interest in algorithms and prediction, but that is beyond the scope of this document.

Another term that is frequently used in the literature is "data mining." This is a more general term that refers to the entire range of techniques for drawing inferences and making predictions from data. Hence, machine learning techniques are a subset of data mining techniques that may also include visualization techniques used by human beings to make inferences and predictions.

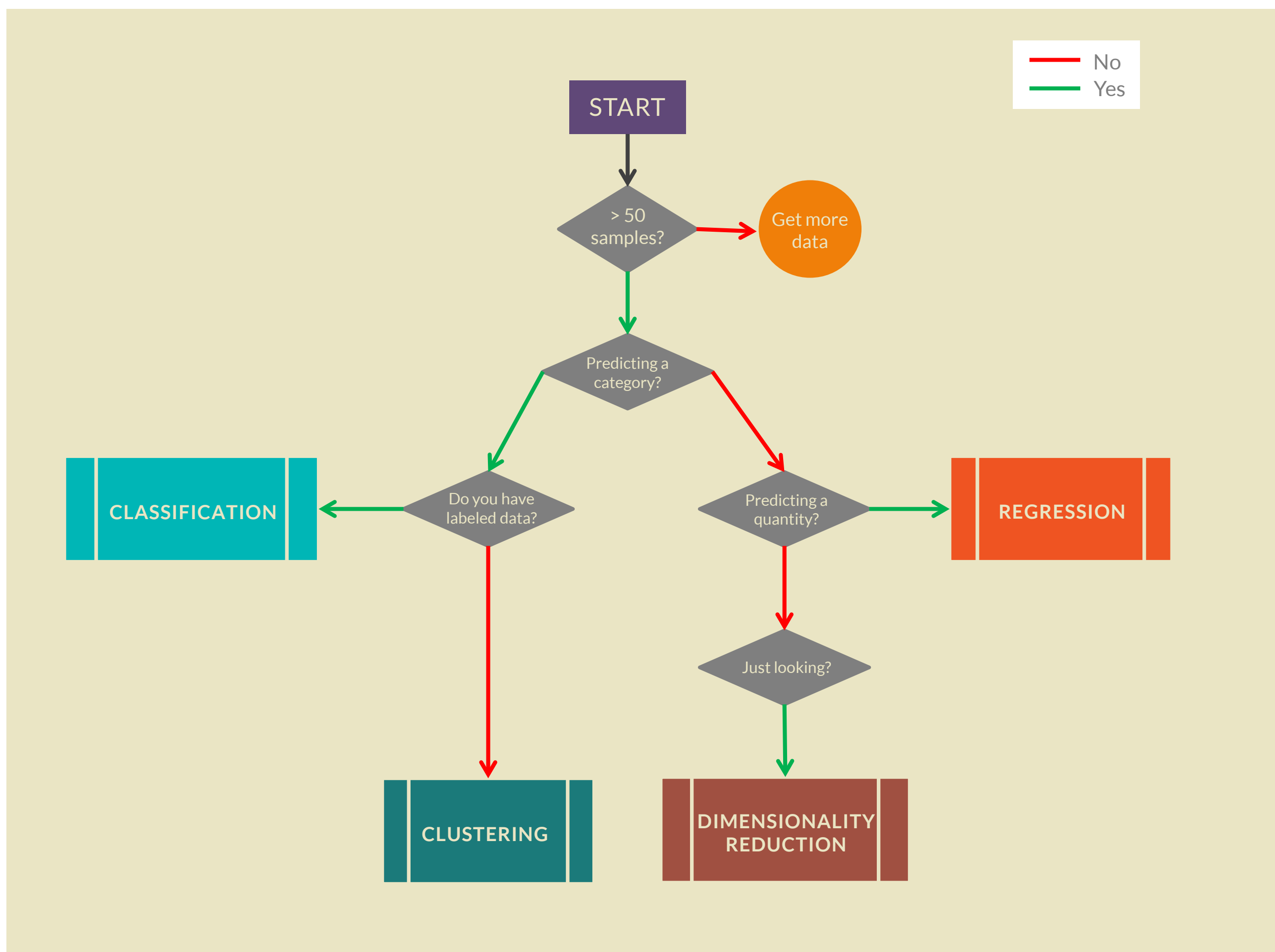

Figure 14: Machine learning flow chart

Figure 14 shows a basic flow chart for deciding which class of learning algorithms to use when embarking on data analysis.

© 2014 Cloud Security Alliance - All Rights Reserved.

Metrics for Evaluating Classifiers and Predictors

Classifiers and predictors can be evaluated on the following five broad parameters:

- **Accuracy**: the ability of the given classifier or predictor to correctly predict the class label or value of new data or previously unseen data (i.e., tuples without class label information)
- **Speed**: the computational costs involved in generating and using the given classifier or predictor
- **Robustness**: the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values
- **Scalability**: the ability to construct the classifier or predictor efficiently given large amounts of data
- **Interpretability**: the level of understanding and insight that is provided by the classifier or predictor(interpretability is subjective and therefore more difficult to assess)

The accuracy of a classifier on a given test set is the percentage of tuples that are classified correctly (these test set tuples with labels should not have been used to train the classifier). Similarly, the accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data. The **error rate** or **misclassification rate** of a classifier is simply the remaining percentage of tuples that were not classified correctly.

For a classifier with m classes, a confusion matrix is an *m* x *m* table. An entry *CM_i_j* in the table indicates the number of tuples of class *i* that were labeled as class *j*. If a classifier has good accuracy, most of the non-diagonal entries of the table should be close to zero. From the confusion matrix, other metrics can be computed, such as precision and sensitivity.

For a predictor, the accuracy is computed by a metric, such as the root mean squared error over a test set. Accuracy can be estimated using one or more test sets that are independent of the training set. Estimation techniques, such as cross-validation and bootstrapping, need to be discussed.

When a learner outputs a classifier that is 100% accurate on training data but, say, 50% accurate on test data, when it could have been 75% accurate on both, then we say that the learner has overfit. **Overfitting** can be measured using bias and variance. Bias is the learner's tendency to learn the same wrong thing, while variance is the tendency to learn random things irrespective of the real thing [38].

The generalization error for a learner can be expressed as the sum of bias squared and the variance. Hence, there is a tradeoff when minimizing the generalization error between minimizing bias and variance simultaneously. If the generalization error is minimized by minimizing the variance, then the bias might be high and we get extreme **underfitting**. If the bias is low, but the variance high, then we get overfitting.

Visualization

Most commonly used big data visualization techniques can be broadly classified into the following three categories (Figure 15).

Spatial Layout Visualization – This class of visualization techniques refer to formulations that uniquely map a data object to a specific point on the coordinate space. The primary motivation of such techniques is the cognitive ability of humans to easily interpret information organized as a *spatial substrate*. Commonly used spatial layout visualization techniques include line charts, bar charts, scatter plots, etc. However, these graphics are often limited by their inability

© 2014 Cloud Security Alliance - All Rights Reserved.

to visualize complex relationships in data. One such example of a complex relationship is the presence of hierarchy in data objects, often visualized using *treemaps* [21]. Another example of a popular spatial layout visualization technique is a graph or network layout visualization in which the presence of nodes and edges leads to interesting insights from data. Force-directed graph drawing algorithm [22] is an example of a visualization algorithm.

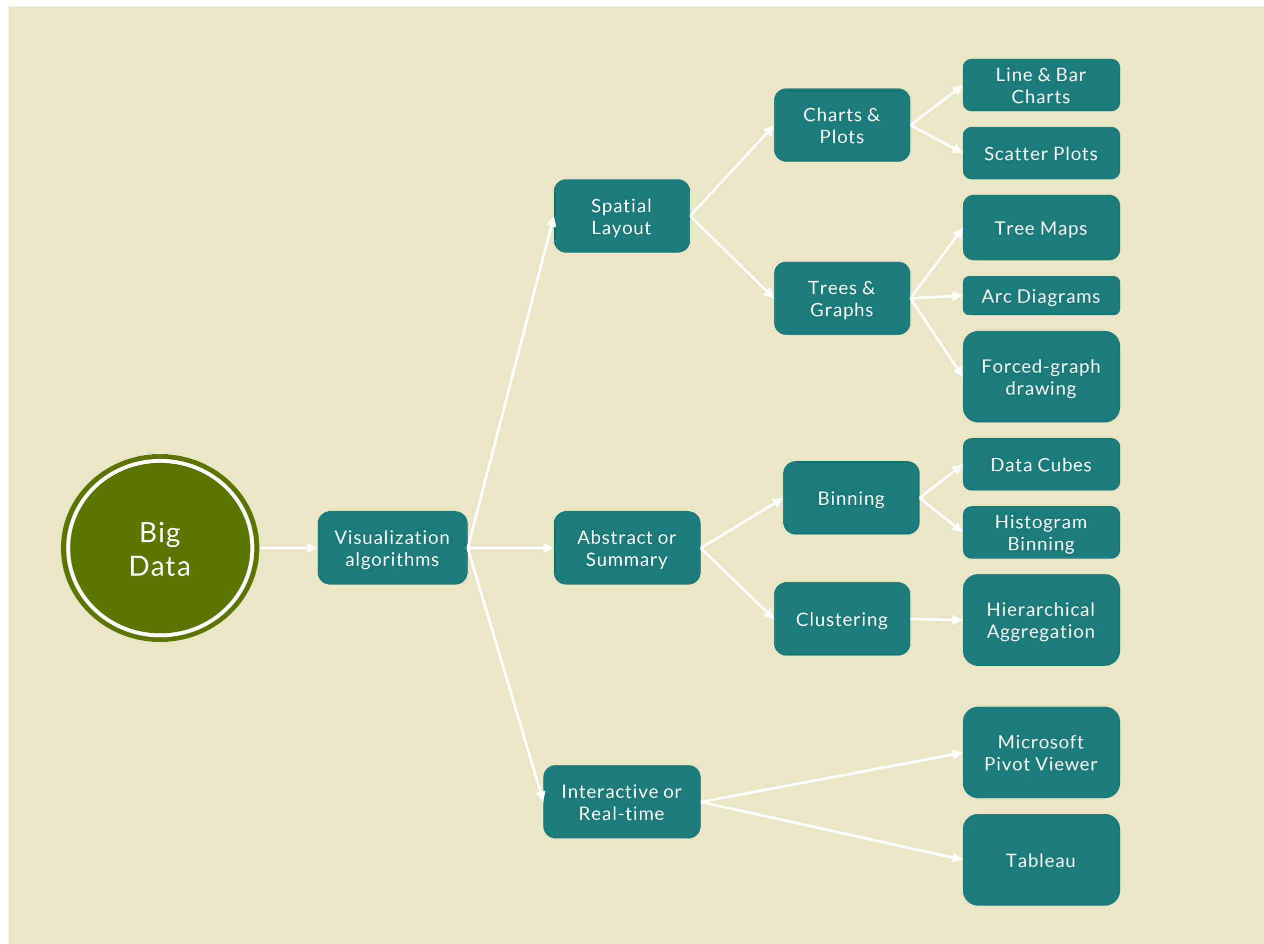

Figure 15: Taxonomy for visualization techniques

Abstract/Summary Visualization – Often big data analytics require data to be processed at scale (e.g. billions of records, terabytes of data) before any meaningful correlations can be discovered. Scaling existing visualization techniques at this level becomes a non-trivial task. A new class of visualization techniques has been proposed lately that process and abstract or summarize such large-scale data before rendering it to visualization routines [23]. These techniques fall under Interactive Visualization methods. Common examples of data abstraction is binning it into histograms or presenting them as data cubes. A number of clustering algorithms have also been proposed that extend binning-based summarization of data to novel concepts. They have the added advantage of providing a compact, reduced dimension representation of data.

Interactive/Real-Time Visualization – A more recent class of techniques fall under interactive visualization that have to adapt to user interactions in real-time. Such techniques necessitate that even complex visualization mechanisms take less than a second for a real-time navigation of data by a user [24]. These techniques are quite powerful in the sense that they allow users to rapidly discover important insights in the data and prove or disprove different data science theories on top of such insights. Such techniques are also crucial to industries that rely greatly on data-driven insights. Today a number of industry software, such as Microsoft Pivot Table and Tableau, employ similar strategies for interactive visualization.

© 2014 Cloud Security Alliance - All Rights Reserved.

Security and Privacy

The Security and Privacy challenges for big data may be organized into four aspects of the big data ecosystem as depicted in Figure 16:

1. Infrastructure Security
2. Data Privacy
3. Data Management
4. Integrity and Reactive Security

Securing the infrastructure of big data systems involves securing distributed computations and data stores. Securing the data itself is of paramount importance, so we have to ensure that information dissemination is privacy-preserving and that sensitive data is protected through the use of cryptography and granular access control. Managing enormous volumes of data necessitates scalable and distributed solutions for not only securing data stores but also enabling efficient audits and investigations of data provenance. Finally, the streaming data that is coming in from diverse end-points has to be checked for integrity and can be used to perform real-time analytics for security incidents to ensure the health of the infrastructure.

We will also discuss the security issues that arise for the various forms of data discussed earlier.

Streaming Data – There are two complementary security problems for streaming data depending on whether the data is public or not. For public data, confidentiality may not be an issue, but the filtering criteria applied by individual clients, such as governments, may be classified. For private data, confidentiality may be a concern, while at the same time suitably modified version of the data may be disclosed to achieve specific utilities, such as predictive analytics.

In “Private Searching On Streaming Data” [25], Ostrovsky and Skeith consider the problem of private searching on streaming data, where they efficiently implement searching for documents that satisfy a secret criterion (such as presence or absence of a hidden combination of hidden keywords) under various cryptographic assumptions. In their scheme, the client can send a garbled transformation of the secret criteria to the filter nodes. The filter nodes can apply the garbled criteria to the incoming data resulting in encrypted filtered messages, which only the client can decrypt. This effectively hides the criteria as well as the actual data that were filtered.

The continuous nature of streaming time series data presents special technical challenges in hiding the sensitive aspects of such data. In particular, simple-minded random perturbation of data points does not provide rigorous privacy guarantees. In “Time series compressibility and privacy” [26], Papadimitriou, et al. study the trade-offs between time series compressibility and partial information hiding and the implication of this tradeoff on the introduction of uncertainty about individual values by perturbing them.

© 2014 Cloud Security Alliance - All Rights Reserved.

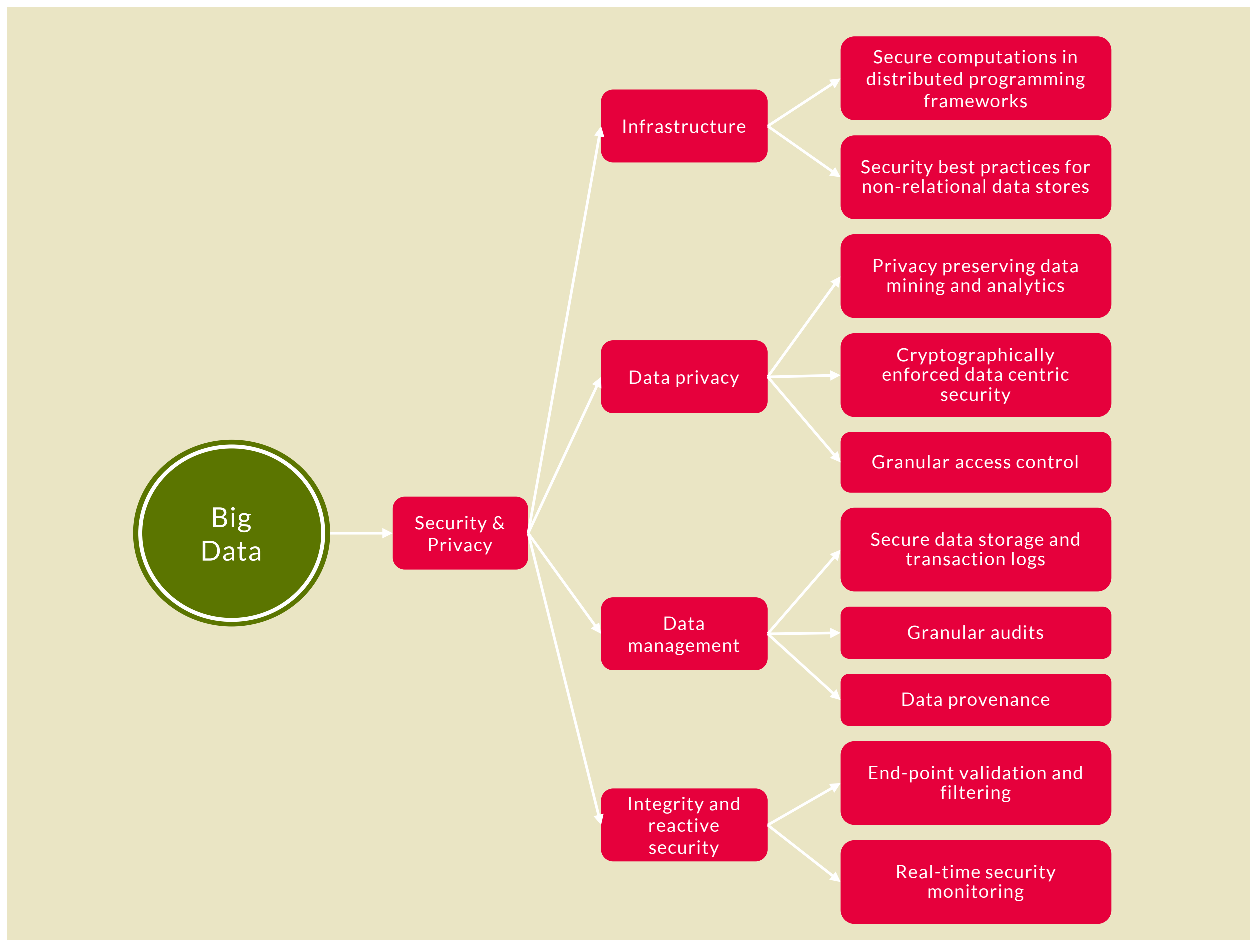

Figure 16: Security and privacy classification

Graph Data – In "Private analysis of graph structure" [27], Karwa, et al. present efficient algorithms for releasing useful statistics about graph data while providing rigorous privacy guarantees. Their algorithms work on datasets that reflect relationships between individuals, such as social ties or email communication. The algorithms effectively hide the presence or absence of any particular relationship. Specifically, the algorithms output approximate answers to subgraph counting queries. Given a query graph such as a triangle or a star, the goal is to return the number of isomorphic copies of the query graph in the input graph.

Scientific – Often scientific data is not required to be confidential. Nevertheless, integrity of data, especially coming from remote sensor nodes, has to be ensured. Lightweight, short signatures have been proposed in the literature, which can be used for such tasks. While there is a vast literature on digital signatures, some examples are [28], [29], and [30], which are geared towards producing short signatures.

However, in many scientific applications, sensors are deployed in open environments, and hence are vulnerable to physical attacks, potentially compromising the sensor's cryptographic keys. In [31], the authors propose a framework for secure information aggregation in large sensor networks, which addresses key compromise by attackers. In their framework, certain nodes in the sensor network – called aggregators – help aggregate information requested by a query. By constructing efficient random sampling mechanisms and interactive proofs, the user is enabled to verify that the answer given by the aggregator is a good approximation of the true value even when the aggregator and a fraction of the sensor nodes are corrupted.

© 2014 Cloud Security Alliance - All Rights Reserved.

Web – Several standardized cryptographic protocols are in deployment to secure communication on the web. A few examples are TLS, Kerberos, OAuth, PKI, Secure BGP, DNSSEC (Secure DNS), IEEE 802 Series protocols, IPSec and Secure Re-routing in Mobile IPv6.

Recently, big data analytics has also enabled analyzing logs generated by web servers for security intelligence and forensics. In the NIST Special Publication 800-92 [32], the authors recommend best practices for organizations to log data. Because of the widespread deployment of networked servers, workstations, and other computing devices, and the ever-increasing number of threats against networks and systems, the number, volume, and variety of computer security logs has increased greatly. This increase has created the need for computer security log management, which is the process for generating, transmitting, storing, analyzing, and disposing of computer security log data. The general recommendations are that organizations should establish policies and procedures for log management, prioritize log management appropriately throughout the organization, create and maintain a log management infrastructure, provide proper support for all staff with log management responsibilities, and establish standard log management operational processes.

Retail and Financial Data – Due to the highly personal nature of retail and financial data of individuals, best practices to store and transmit the data should always be followed. Some of the existing practices are to keep data encrypted at rest, in transit and in infrastructure to ensure proper authorization and authentication of entities accessing the data. Compliance standards such as PCI DSS for the financial industry also provide security best practices and legal requirements.

The advent of high volumes of such data has enabled a plethora of analytics techniques that generate information of high value for third party organizations who desire to target the right demographics with their products. In practice, such data is shared after sufficient removal of apparently unique identifiers by the processes of anonymization and aggregation. This process is ad-hoc, often based on empirical evidence [33] and has led to many instances of "de-anonymization" in conjunction with publicly available data [34]. Sharing of analytics that is devoid of personal information can actually be seen as an end goal. However, third parties and research organizations often want to do studies and analytics of their own, and for that, distilling useful but rigorously sanitized data remains a challenge.

Several formal models to address privacy preserving data disclosure have been proposed [35]. One of the strongest models is the framework of Differential Privacy [36]. In "GUPT: privacy preserving data analysis made easy" [37], the authors present the design and evaluation of a system called GUPT that guarantees differential privacy to programs not developed with privacy in mind. GUPT uses a model of data sensitivity that degrades privacy of data over time. This enables efficient allocation of different levels of privacy for different user applications while guaranteeing an overall constant level of privacy and maximizing the utility of each application.

Conclusion

We have given the beginning of the taxonomy of the big data landscape along six of the most important dimensions. The six dimensions are data domains, compute infrastructure, storage architectures, analytics, visualization, security and privacy, and data domains. Big data infrastructure and methodology continue to evolve at a fast pace, but the underlying technologies they are based on have, in many cases, been invented many years ago. The greatly increased digitization of human activity and machine-to-machine communications, combined with large scale inexpensive hardware, is making practical many previously academic ideas of parallel and distributed computing, along with new tweaks necessary to make them even more useful in real world applications.

© 2014 Cloud Security Alliance - All Rights Reserved.

References

[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.

[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.

© 2014 Cloud Security Alliance - All Rights Reserved.

[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.

© 2014 Cloud Security Alliance - All Rights Reserved.