Table of Contents

Figures

Tables

Executive Summary

The NIST Big Data Public Working group (NBD-PWG) Reference Architecture Subgroup prepared this *NIST Big Data Interoperability Framework: Volume 6, Reference Architecture* to provide a vendor-neutral, technology- and infrastructure-agnostic conceptual model and examine related issues. The conceptual model, referred to as the NIST Big Data Reference Architecture (NBDRA), was crafted by examining publicly available Big Data architectures representing various approaches and products. Inputs from the other NBD-PWG subgroups were also incorporated into the creation of the NBDRA. It is applicable to a variety of business environments, including tightly integrated enterprise systems, as well as loosely coupled vertical industries that rely on cooperation among independent stakeholders. The NBDRA captures the two known Big Data economic value chains: information, where value is created by data collection, integration, analysis, and applying the results to data-driven services, and the information technology (IT), where value is created by providing networking, infrastructure, platforms, and tools in support of vertical data-based applications.

The *NIST Big Data Interoperability Framework* consists of seven volumes, each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The seven volumes are:

- Volume 1, Definitions
- Volume 2, Taxonomies
- Volume 3, Use Cases and General Requirements
- Volume 4, Security and Privacy
- Volume 5, Architectures White Paper Survey
- Volume 6, Reference Architecture
- Volume 7, Standards Roadmap

The *NIST Big Data Interoperability Framework* will be released in three versions, which correspond to the three development stages of the NBD-PWG work. The three stages aim to achieve the following with respect to the NIST Big Data Reference Architecture (NBDRA).

Stage 1: Identify the high-level Big Data reference architecture key components, which are technology-, infrastructure-, and vendor-agnostic.
Stage 2: Define general interfaces between the NBDRA components.
Stage 3: Validate the NBDRA by building Big Data general applications through the general interfaces.

Potential areas of future work for the Subgroup during stage 2 are highlighted in Section 1.5 of this volume. The current effort documented in this volume reflects concepts developed within the rapidly evolving field of Big Data.

1 INTRODUCTION

1.1 BACKGROUND

There is broad agreement among commercial, academic, and government leaders about the remarkable potential of Big Data to spark innovation, fuel commerce, and drive progress. Big Data is the common term used to describe the deluge of data in today's networked, digitized, sensor-laden, and information-driven world. The availability of vast data resources carries the potential to answer questions previously out of reach, including the following:

- How can a potential pandemic reliably be detected early enough to intervene?
- Can new materials with advanced properties be predicted before these materials have ever been synthesized?
- How can the current advantage of the attacker over the defender in guarding against cyber-security threats be reversed?

There is also broad agreement on the ability of Big Data to overwhelm traditional approaches. The growth rates for data volumes, speeds, and complexity are outpacing scientific and technological advances in data analytics, management, transport, and data user spheres.

Despite widespread agreement on the inherent opportunities and current limitations of Big Data, a lack of consensus on some important fundamental questions continues to confuse potential users and stymie progress. These questions include the following:

- What attributes define Big Data solutions?
- How is Big Data different from traditional data environments and related applications?
- What are the essential characteristics of Big Data environments?
- How do these environments integrate with currently deployed architectures?
- What are the central scientific, technological, and standardization challenges that need to be addressed to accelerate the deployment of robust Big Data solutions?

Within this context, on March 29, 2012, the White House announced the Big Data Research and Development Initiative.[1] The initiative's goals include helping to accelerate the pace of discovery in science and engineering, strengthening national security, and transforming teaching and learning by improving the ability to extract knowledge and insights from large and complex collections of digital data.

Six federal departments and their agencies announced more than $200 million in commitments spread across more than 80 projects, which aim to significantly improve the tools and techniques needed to access, organize, and draw conclusions from huge volumes of digital data. The initiative also challenged industry, research universities, and nonprofits to join with the federal government to make the most of the opportunities created by Big Data.

Motivated by the White House initiative and public suggestions, the National Institute of Standards and Technology (NIST) has accepted the challenge to stimulate collaboration among industry professionals to further the secure and effective adoption of Big Data. As one result of NIST's Cloud and Big Data Forum held on January 15–17, 2013, there was strong encouragement for NIST to create a public working group for the development of a Big Data Interoperability Framework. Forum participants noted that this roadmap should define and prioritize Big Data requirements, including interoperability, portability, reusability, extensibility, data usage, analytics, and technology infrastructure. In doing so, the roadmap would accelerate the adoption of the most secure and effective Big Data techniques and technology.

activities may execute in parallel or in any number of sequences and will frequently communicate with each other through the messaging/communications element of the Big Data Framework Provider. Also, the functions of the Big Data Application Provider, specifically the collection and access activities, will interact with the Security and Privacy Fabric to perform authentication/authorization and record/maintain data provenance.

Each of the functions can run on a separate Big Data Framework Provider or all can use a common Big Data Framework Provider. The considerations behind these different system approaches would depend on potentially different technological needs, business and/or deployment constraints (including privacy), and other policy considerations. The baseline NBDRA does not show the underlying technologies, business considerations, and topological constraints, thus making it applicable to any kind of system approach and deployment.

For example, the infrastructure of the Big Data Application Provider would be represented as one of the Big Data Framework Providers. If the Big Data Application Provider uses external/outsourced infrastructures as well, it or they will be represented as another or multiple Big Data Framework Providers in the NBDRA. The multiple grey blocks behind the Big Data Framework Providers in Figure 2 indicate that multiple Big Data Framework Providers can support a single Big Data Application Provider.

4.3.1 COLLECTION

In general, the collection activity of the Big Data Application Provider handles the interface with the Data Provider. This may be a general service, such as a file server or web server configured by the System Orchestrator to accept or perform specific collections of data, or it may be an application-specific service designed to pull data or receive pushes of data from the Data Provider. Since this activity is receiving data at a minimum, it must store/buffer the received data until it is persisted through the Big Data Framework Provider. This persistence need not be to physical media but may simply be to an in-memory queue or other service provided by the processing frameworks of the Big Data Framework Provider. The collection activity is likely where the extraction portion of the Extract, Transform, Load (ETL)/Extract, Load, Transform (ELT) cycle is performed. At the initial collection stage, sets of data (e.g., data records) of similar structure are collected (and combined), resulting in uniform security, policy, and other considerations. Initial metadata is created (e.g., subjects with keys are identified) to facilitate subsequent aggregation or look-up methods.

4.3.2 PREPARATION

The preparation activity is where the transformation portion of the ETL/ELT cycle is likely performed, although analytics activity will also likely perform advanced parts of the transformation. Tasks performed by this activity could include data validation (e.g., checksums/hashes, format checks), cleansing (e.g., eliminating bad records/fields), outlier removal, standardization, reformatting, or encapsulating. This activity is also where source data will frequently be persisted to archive storage in the Big Data Framework Provider and provenance data will be verified or attached/associated. Verification or attachment may include optimization of data through manipulations (e.g., deduplication) and indexing to optimize the analytics process. This activity may also aggregate data from different Data Providers, leveraging metadata keys to create an expanded and enhanced data set.

4.3.3 ANALYTICS

The analytics activity of the Big Data Application Provider includes the encoding of the low-level business logic of the Big Data system (with higher-level business process logic being encoded by the System Orchestrator). The activity implements the techniques to extract knowledge from the data based on the requirements of the vertical application. The requirements specify the data processing algorithms for processing the data to produce new insights that will address the technical goal. The analytics activity will leverage the processing frameworks to implement the associated logic. This typically involves the activity providing software that implements the analytic logic to the batch and/or streaming elements of

the processing framework for execution. The messaging/communication framework of the Big Data Framework Provider may be used to pass data or control functions to the application logic running in the processing frameworks. The analytic logic may be broken up into multiple modules to be executed by the processing frameworks which communicate, through the messaging/communication framework, with each other and other functions instantiated by the Big Data Application Provider.

4.3.4 VISUALIZATION

The visualization activity of the Big Data Application Provider prepares elements of the processed data and the output of the analytic activity for presentation to the Data Consumer. The objective of this activity is to format and present data in such a way as to optimally communicate meaning and knowledge. The visualization preparation may involve producing a text-based report or rendering the analytic results as some form of graphic. The resulting output may be a static visualization and may simply be stored through the Big Data Framework Provider for later access. However, the visualization activity frequently interacts with the access activity, the analytics activity, and the Big Data Framework Provider (processing and platform) to provide interactive visualization of the data to the Data Consumer based on parameters provided to the access activity by the Data Consumer. The visualization activity may be completely application-implemented, leverage one or more application libraries, or may use specialized visualization processing frameworks within the Big Data Framework Provider.

4.3.5 ACCESS

The access activity within the Big Data Application Provider is focused on the communication/interaction with the Data Consumer. Similar to the collection activity, the access activity may be a generic service such as a web server or application server that is configured by the System Orchestrator to handle specific requests from the Data Consumer. This activity would interface with the visualization and analytic activities to respond to requests from the Data Consumer (who may be a person) and uses the processing and platform frameworks to retrieve data to respond to Data Consumer requests. In addition, the access activity confirms that descriptive and administrative metadata and metadata schemes are captured and maintained for access by the Data Consumer and as data is transferred to the Data Consumer. The interface with the Data Consumer may be synchronous or asynchronous in nature and may use a pull or push paradigm for data transfer.

4.4 BIG DATA FRAMEWORK PROVIDER

The Big Data Framework Provider typically consists of one or more hierarchically organized instances of the components in the NBDRA IT value chain (Figure 2). There is no requirement that all instances at a given level in the hierarchy be of the same technology. In fact, most Big Data implementations are hybrids that combine multiple technology approaches in order to provide flexibility or meet the complete range of requirements, which are driven from the Big Data Application Provider.

Many of the recent advances related to Big Data have been in the area of frameworks designed to scale to Big Data needs (e.g., addressing volume, variety, velocity, and variability) while maintaining linear or near-linear performance. These advances have generated much of the technology excitement in the Big Data space. Accordingly, there is a great deal more information available in the frameworks area compared to the other components, and the additional detail provided for the Big Data Framework Provider in this document reflects this imbalance.

The Big Data Framework Provider comprises the following three subcomponents (from the bottom to the top):

- Infrastructure Frameworks
- Data Platform Frameworks
- Processing Frameworks

4.4.1 INFRASTRUCTURE FRAMEWORKS

This Big Data Framework Provider element provides all of the resources necessary to host/run the activities of the other components of the Big Data system. Typically, these resources consist of some combination of physical resources, which may host/support similar virtual resources. These resources are generally classified as follows:

- Networking: These are the resources that transfer data from one infrastructure framework component to another.
- Computing: These are the physical processors and memory that execute and hold the software of the other Big Data system components.
- Storage: These are resources which provide persistence of the data in a Big Data system.
- Environmental: These are the physical plant resources (e.g., power, cooling, security) that must be accounted for when establishing an instance of a Big Data system.

While the Big Data Framework Provider component may be deployed directly on physical resources or on virtual resources, at some level all resources have a physical representation. Physical resources are frequently used to deploy multiple components that will be duplicated across a large number of physical nodes to provide what is known as horizontal scalability. Virtualization is frequently used to achieve elasticity and flexibility in the allocation of physical resources and is often referred to as infrastructure as a service (IaaS) within the cloud computing community. Virtualization is typically found in one of three basic forms within a Big Data Architecture.

- Native: In this form, a hypervisor runs natively on the bare metal and manages multiple virtual machines consisting of operating systems (OS) and applications.
- Hosted: In this form, an OS runs natively on the bare metal and a hypervisor runs on top of that to host a client OS and applications. This model is not often seen in Big Data architectures due to the increased overhead of the extra OS layer.
- Containerized: In this form, hypervisor functions are embedded in the OS, which runs on bare metal. Applications are run inside containers, which control or limit access to the OS and physical machine resources. This approach has gained popularity for Big Data architectures because it further reduces overhead since most OS functions are a single shared resource. It may not be considered as secure or stable since in the event that the container controls/limits fail, one application may take down every application sharing those physical resources.

The following subsections describe the types of physical and virtual resources that comprise Big Data infrastructure.

4.4.1.1 NETWORKING

The connectivity of the architecture infrastructure should be addressed, as it affects the velocity characteristic of Big Data. While, some Big Data implementations may solely deal with data that is already resident in the data center and does not need to leave the confines of the local network, others may need to plan and account for the movement of Big Data either into or out of the data center. The location of Big Data systems with transfer requirements may depend on the availability of external network connectivity (i.e., bandwidth) and the limitations of Transmission Control Protocol (TCP) where there is low latency (as measured by packet Round Trip Time) with the primary senders or receivers of Big Data. To address the limitations of TCP, architects for Big Data systems may need to consider some of the advanced non-TCP based communications protocols available that are specifically designed to transfer large files such as video and imagery.

Overall availability of the external links is another infrastructure aspect relating to the velocity characteristic of Big Data that should be considered in architecting external connectivity. A given connectivity link may be able to easily handle the velocity of data while operating correctly. However, should the quality of service on the link degrade or the link fail completely, data may be lost or simply

back up to the point that it can never recover. Use cases exist where the contingency planning for network outages involves transferring data to physical media and physically transporting it to the desired destination. However, even this approach is limited by the time it may require to transfer the data to external media for transport.

The volume and velocity characteristics of Big Data often are driving factors in the implementation of the internal network infrastructure as well. For example, if the implementation requires frequent transfers of large multi-gigabyte files between cluster nodes, then high speed and low latency links are required to maintain connectivity to all nodes in the network. Provisions for dynamic quality of services (QOS) and service priority may be necessary in order to allow failed or disconnected nodes to re-synchronize once connectivity is restored. Depending on the availability requirements, redundant and fault tolerant links may be required. Other aspects of the network infrastructure include name resolution (e.g., Domain Name Server [DNS]) and encryption along with firewalls and other perimeter access control capabilities. Finally, the network infrastructure may also include automated deployment, provisioning capabilities, or agents and infrastructure wide monitoring agents that are leveraged by the management/communication elements to implement a specific model.

Security of the networks is another aspect that must be addressed depending on the sensitivity of the data being processed. Encryption may be needed between the network and external systems to avoid man in the middle interception and compromise of the data. In cases, where the network infrastructure within the data center is shared encryption of the local network should also be considered. Finally, in conjunction with the security and privacy fabric auditing and intrusion detection capabilities need to be address.

Two concepts, SDN and Network Function Virtualization (NFV), have recently been developed in support of scalable networks and scalable systems using them.

4.4.1.1.1 Software Defined Networks

Frequently ignored, but critical to the performance of distributed systems and frameworks, and especially critical to Big Data implementations, is the efficient and effective management of networking resources. Significant advances in network resource management have been realized through what is known as SDN. Much like virtualization frameworks manage shared pools of CPU/memory/disk, SDNs (or virtual networks) manage pools of physical network resources. In contrast to the traditional approaches of dedicated physical network links for data, management, I/O, and control, SDNs contain multiple physical resources (including links and actual switching fabric) that are pooled and allocated as required to specific functions and sometimes to specific applications. This allocation can consist of raw bandwidth, quality of service priority, and even actual data routes.

4.4.1.1.2 Network Function Virtualization

With the advent of virtualization, virtual appliances can now reasonably support a large number of network functions that were traditionally performed by dedicated devices. Network functions that can be implemented in this manner include routing/routers, perimeter defense (e.g., firewalls), remote access authorization, and network traffic/load monitoring. Some key advantages of NFV include elasticity, fault tolerance, and resource management. For example, the ability to automatically deploy/provision additional firewalls in response to a surge in user or data connections and then un-deploy them when the surge is over can be critical in handling the volumes associated with Big Data.

4.4.1.2 COMPUTING

The logical distribution of cluster/computing infrastructure may vary from a tightly coupled high performance computing cluster to a dense grid of physical commodity machines in a rack, to a set of virtual machines running on a cloud service provider (CSP), or to a loosely coupled set of machines distributed around the globe providing access to unused computing resources. Computing infrastructure also frequently includes the underlying OSs and associated services used to interconnect the cluster

resources via the networking elements. Computing resources may also include computation accelerators such as Graphic Processing Units (GPUS) and Field Programmable Gate Arrays (FPGAS) which can provide dynamically programmed massively parallel computing capabilities to individual nodes in the infrastructure.

4.4.1.3 STORAGE

The storage infrastructure may include any resource from isolated local disks to storage area networks (SANs) or network-attached storage (NAS).

Two aspects of storage infrastructure technology that directly influence their suitability for Big Data solutions are capacity and transfer bandwidth. Capacity refers to the ability to handle the data volume. Local disks/file systems are specifically limited by the size of the available media. Hardware or software redundant array of independent disks (RAID) solutions—in this case local to a processing node—help with scaling by allowing multiple pieces of media to be treated as a single device. However, this approach is limited by the physical dimension of the media and the number of devices the node can accept. SAN and NAS implementations—often known as shared disk solutions—remove that limit by consolidating storage into a storage specific device. By consolidating storage, the second aspect—transfer bandwidth—may become an issue. While both network and I/O interfaces are getting faster and many implementations support multiple transfer channels, I/O bandwidth can still be a limiting factor. In addition, despite the redundancies provided by RAID, hot spares, multiple power supplies, and multiple controllers, these boxes can often become I/O bottlenecks or single points of failure in an enterprise. Many Big Data implementations address these issues by using distributed file systems within the platform framework.

4.4.1.4 ENVIRONMENTAL RESOURCES

Environmental resources, such as power and heating, ventilation, and air conditioning, are critical to the Big Data Framework Provider. While environmental resources are critical to the operation of the Big Data system, they are not within the technical boundaries and are, therefore, not depicted in Figure 2, the NBDRA conceptual model.

Adequately sized infrastructure to support application requirements is critical to the success of Big Data implementations. The infrastructure architecture operational requirements range from basic power and cooling to external bandwidth connectivity (as discussed above). A key evolution that has been driven by Big Data is the increase in server density (i.e., more CPU/memory/disk per rack unit). However, with this increased density, infrastructure—specifically power and cooling—may not be distributed within the data center to allow for sufficient power to each rack or adequate air flow to remove excess heat. In addition, with the high cost of managing energy consumption within data centers, technologies have been developed that actually power down or idle resources not in use to save energy or to reduce consumption during peak periods.

Also important within this element are the physical security of the facilities and auxiliary (e.g. power sub-stations). Specifically perimeter security to include credential verification (badge/biometrics), surveillance, and perimeter alarms all are necessary to maintain control of the data being processed.

4.4.2 DATA PLATFORM FRAMEWORKS

Data Platform Frameworks provide for the logical data organization and distribution combined with the associated access application programming interfaces (APIs) or methods. The frameworks may also include data registry and metadata services along with semantic data descriptions such as formal ontologies or taxonomies. The logical data organization may range from simple delimited flat files to fully distributed relational or columnar data stores. The storage mediums range from high latency robotic tape drives, to spinning magnetic media, to flash/solid state disks, or to random access memory. Accordingly, the access methods may range from file access APIs to query languages such as Structured Query Language (SQL.) Typical Big Data framework implementations would support either basic file system style storage or in-memory storage and one or more indexed storage approaches. Based on the

specific Big Data system considerations, this logical organization may or may not be distributed across a cluster of computing resources.

In most aspects, the logical data organization and distribution in Big Data storage frameworks mirrors the common approach for most legacy systems. Figure 3 presents a brief overview of data organization approaches for Big Data.

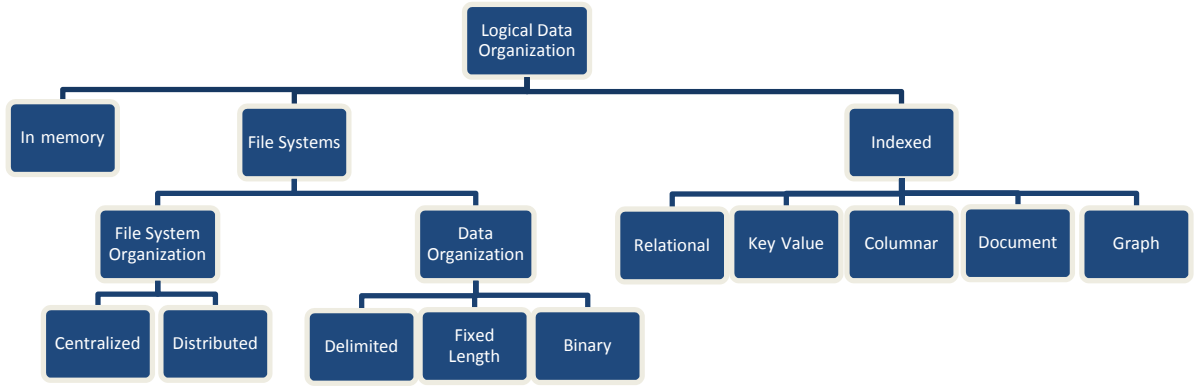


Figure 3: Data Organization Approaches

Many Big Data logical storage organizations leverage the common file system concept—where chunks of data are organized into a hierarchical namespace of directories—as their base and then implement various indexing methods within the individual files. This allows many of these approaches to be run both on simple local storage file systems for testing purposes or on fully distributed file systems for scale.

4.4.2.1 IN-MEMORY

The infrastructure illustrated in the NBDRA (Figure 2) indicates that physical resources are required to support analytics. However, such infrastructure will vary (i.e., will be optimized) for the Big Data characteristics of the problem under study. Large, but static, historical datasets with no urgent analysis time constraints would optimize the infrastructure for the volume characteristic of Big Data, while time-critical analyses such as intrusion detection or social media trend analysis would optimize the infrastructure for the velocity characteristic of Big Data. Velocity implies the necessity for extremely fast analysis and the infrastructure to support it—namely, very low latency, in-memory analytics.

In-memory storage technologies, many of which were developed to support the scientific high performance computing (HPC) domain, are increasingly used due to the significant reduction in memory prices and the increased scalability of modern servers and OSs. Yet, an in-memory element of a velocity-oriented infrastructure will require more than simply massive random-access memory (RAM). It will also require optimized data structures and memory access algorithms to fully exploit RAM performance. Current in-memory database offerings are beginning to address this issue. Shared memory solutions common to HPC environments are often being applied to address inter-nodal communications and synchronization requirements.

Traditional database management architectures are designed to use spinning disks as the primary storage mechanism, with the main memory of the computing environment relegated to providing caching of data and indexes. Many of these in-memory storage mechanisms have their roots in the massively parallel processing and super computer environments popular in the scientific community.

These approaches should not be confused with solid state (e.g., flash) disks or tiered storage systems that implement memory-based storage which simply replicate the disk style interfaces and data structures but with faster storage medium. Actual in-memory storage systems typically eschew the overhead of file system semantics and optimize the data storage structure to minimize memory footprint and maximize the data access rates. These in-memory systems may implement general purpose relational and other not only

(or no) Structured Query Language (NoSQL) style organization and interfaces or be completely optimized to a specific problem and data structure.

Like traditional disk-based systems for Big Data, these implementations frequently support horizontal distribution of data and processing across multiple independent nodes—although shared memory technologies are still prevalent in specialized implementations. Unlike traditional disk-based approaches, in-memory solutions and the supported applications must account for the lack of persistence of the data across system failures. Some implementations leverage a hybrid approach involving write-through to more persistent storage to help alleviate the issue.

The advantages of in-memory approaches include faster processing of intensive analysis and reporting workloads. In-memory systems are especially good for analysis of real time data such as that needed for some complex event processing (CEP) of streams. For reporting workloads, performance improvements can often be on the order of several hundred times faster—especially for sparse matrix and simulation type analytics.

4.4.2.2 FILE SYSTEMS

Many Big Data processing frameworks and applications access their data directly from underlying file systems. In almost all cases, the file systems implement some level of the Portable Operating System Interface (POSIX) standards for permissions and the associated file operations. This allows other higher-level frameworks for indexing or processing to operate with relative transparency as to whether the underlying file system is local or fully distributed. File-based approaches consist of two layers, the file system organization and the data organization within the files.

4.4.2.2.1 File System Organization

File systems tend to be either centralized or distributed. Centralized file systems are basically implementations of local file systems that are placed on a single large storage platform (e.g., SAN or NAS) and accessed via some network capability. In a virtual environment, multiple physical centralized file systems may be combined, split, or allocated to create multiple logical file systems.

Distributed file systems (also known as cluster file systems) seek to overcome the throughput issues presented by the volume and velocity characteristics of big data combine I/O throughput across multiple devices (spindles) on each node, with redundancy and failover mirroring or replicating data at the block level across multiple nodes. Many of these implementations were developed in support of HPC computing solutions requiring high throughput and scalability. Performance, in many HPC implementations is often achieved through dedicated storage nodes using proprietary storage formats and layouts. The data replication is specifically designed to allow the use of heterogeneous commodity hardware across the Big Data cluster. Thus, if a single drive or an entire node should fail, no data is lost because it is replicated on other nodes and throughput is only minimally affected because that processing can be moved to the other nodes. In addition, replication allows for high levels of concurrency for reading data and for initial writes. Updates and transaction style changes tend to be an issue for many distributed file systems because latency in creating replicated blocks will create consistency issues (e.g., a block is changed but another node reads the old data before it is replicated.) Several file system implementations also support data compression and encryption at various levels. One major caveat is that, for distributed block based file systems, the compression/encryption must be able to be split and allow any given block to be decompressed/ decrypted out of sequence and without access to the other blocks.

Distributed object stores (also known as global object stores) are a unique example of distributed file system organization. Unlike the approaches described above, which implement a traditional file system hierarchy namespace approach, distributed object stores present a flat name space with a globally unique identifier (GUID) for any given chunk of data. Generally, data in the store is located through a query against a metadata catalog that returns the associated GUIDs. The GUID generally provides the underlying software implementation with the storage location of the data of interest. These object stores

are developed and marketed for storage of very large data objects, from complete data sets to large individual objects (e.g., high resolution images in the tens of gigabytes [GBs] size range). The biggest limitation of these stores for Big Data tends to be network throughput (i.e., speed) because many require the object to be accessed in total. However, future trends point to the concept of being able to send the computation/application to the data versus needing to bring the data to the application.

From a maturity perspective, two key areas where distributed file systems are likely to improve are (1) random write I/O performance and consistency, and (2) the generation of de facto standards at a similar or greater level as the Internet Engineering Task Force Requests for Comments document series, such as those currently available for the network file system (NFS) protocol. Distributed object stores, while currently available and operational from several commercial providers and part of the roadmap for large organizations such as the National Geospatial Intelligence Agency (NGA), currently are essentially proprietary implementations. For Distributed object stores to become prevalent within Big Data ecosystems, there should be: some level of interoperability available (i.e., through standardized APIs); standards-based approaches for data discovery; and, most importantly, standards-based approaches that allow the application to be transferred over the grid and run locally to the data versus transferring the data to the application.

4.4.2.2.2 In File Data Organization

Very little is different in Big Data for in file data organization. File based data can be text, binary data, fixed length records, or some sort of delimited structure (e.g., comma separated values [CSV], Extensible Markup Language [XML]). For record oriented storage (either delimited or fixed length), this generally is not an issue for Big Data unless individual records can exceed a block size. Some distributed file system implementations provide compression at the volume or directory level and implement it below the logical block level (e.g., when a block is read from the file system, it is decompressed/decrypted before being returned).Because of their simplicity, familiarity, and portability, delimited files are frequently the default storage format in many Big Data implementations. The trade-off is I/O efficiency (i.e., speed). While individual blocks in a distributed file system might be accessed in parallel, each block still needs to be read in sequence. In the case of a delimited file, if only the last field of certain records is of interest with perhaps hundreds of fields, a lot of I/O and processing bandwidth is wasted.

Binary formats tend to be application or implementation specific. While they can offer much more efficient access due to smaller data sizes (i.e., integers are two to four bytes in binary while they are one byte per digit in ASCII [American Standard Code for Information Interchange]), they offer limited portability between different implementations. At least one popular distributed file system provides its own standard binary format, which allows data to be portable between multiple applications without additional software. However, the bulk of the indexed data organization approaches discussed below leverage binary formats for efficiency.

4.4.2.3 INDEXED STORAGE ORGANIZATION

The very nature of Big Data (primarily the volume and velocity characteristics) practically drives requirements to some form of indexing structure. Big Data volume requires that specific data elements be located quickly without scanning across the entire dataset. Big Data velocity also requires that data can be located quickly either for matching (e.g., incoming data matches something in an existing data set) or to know where to write/update new data.

The choice of a particular indexing method or methods depends mostly on the data and the nature of the application to be implemented. For example, graph data (i.e., vertices, edges, and properties) can easily be represented in flat text files as vertex-edge pairs, edge-vertex-vertex triples, or vertex-edge list records. However, processing this data efficiently would require potentially loading the entire data set into memory or being able to distribute the application and data set across multiple nodes so a portion of the graph is in memory on each node. Splitting the graph across nodes requires the nodes to communicate when graph sections have vertices that connect with vertices on other processing nodes. This is perfectly

acceptable for some graph applications—such as shortest path—especially when the graph is static. Some graph processing frameworks operate using this exact model. However, this approach is infeasible for large scale graphs requiring a specialized graph storage framework, where the graph is dynamic or searching or matching to a portion of the graph is needed quickly.

Indexing approaches tend to be classified by the features provided in the implementation, specifically: the complexity of the data structures that can be stored; how well they can process links between data; and, how easily they support multiple access patterns as shown in Figure 4. Since any of these features can be implemented in custom application code, the values portrayed represent approximate norms. For example, key-value stores work well for data that is only accessed through a single key, whose values can be expressed in a single flat structure, and where multiple records do not need to be related. While document stores can support very complex structures of arbitrary width and tend to be indexed for access via multiple document properties, they do not tend to support inter-record relationships well.

It is noted that the specific implementations for each storage approach vary significantly enough that all of the values for the features represented here are really ranges. For example, relational data storage implementations are supporting increasingly complex data structures and ongoing work aims to add more flexible access patterns natively in BigTable columnar implementations. Within Big Data, the performance of each of these features tends to drive the scalability of that approach depending on the problem being solved. For example, if the problem is to locate a single piece of data for a unique key, then key-value stores will scale really well. However, if a problem requires general navigation of the relationships between multiple data records, a graph storage model will likely provide the best performance.

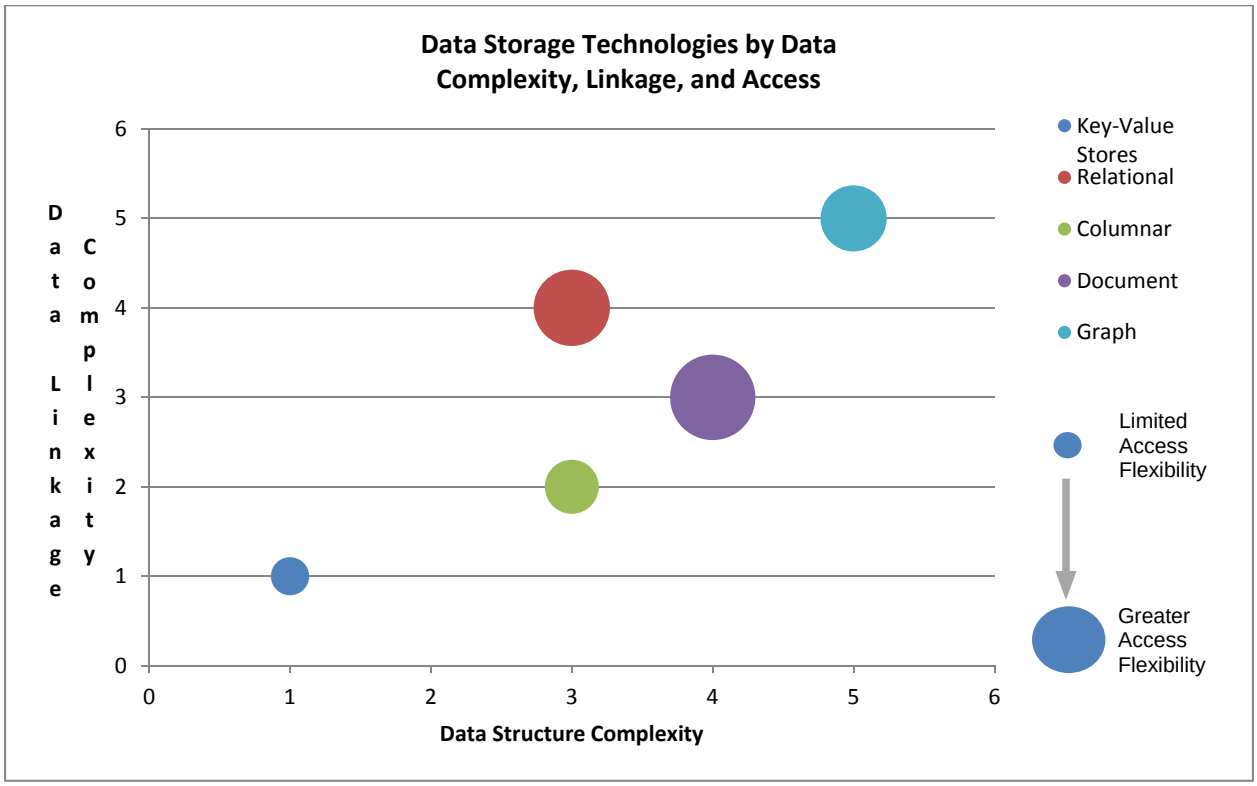


Figure 4: Data Storage Technologies

4.4.3 PROCESSING FRAMEWORKS

The processing frameworks for Big Data provide the necessary infrastructure software to support implementation of applications that can deal with the volume, velocity, variety, and variability of data. Processing frameworks define how the computation and processing of the data is organized. Big Data applications rely on various platforms and technologies to meet the challenges of scalable data analytics and operation.

Processing frameworks generally focus on data manipulation, which falls along a continuum between batch and streaming oriented processing. However, depending on the specific data organization platform, and actual processing requested, any given framework may support a range of data manipulation from high latency to near real time (NRT) processing. Overall, many Big Data architectures will include multiple frameworks to support a wide range of requirements.

Typically, processing frameworks are categorized based on whether they support batch or streaming processing. This categorization is generally stated from the user perspective (e.g., how fast does a user get a response to a request). However, Big Data processing frameworks actually have three processing phases: data ingestion, data analysis, and data dissemination, which closely follow the flow of data through the architecture. The Big Data Application Provider activities control the application of specific framework capabilities to these processing phases. The batch-streaming continuum, illustrated in the processing subcomponent in Figure 2, can be applied to the three distinct processing phases. For example, data may enter a Big Data system at high velocity and the end user must quickly retrieve a summary of the prior day's data. In this case, the ingestion of the data into the system needs to be NRT and keep up with the data stream. The analysis portion could be incremental (e.g., performed as the data is ingested) or could be a batch process performed at a specified time, while retrieval (i.e., read visualization) of the data could be interactive. Specific to the use case, data transformation may take place at any point during its transit through the system. For example, the ingestion phase may only write the data as quickly as possible, or it may run some foundational analysis to track incrementally computed information such as minimum, maximum, average. The core processing job may only perform the analytic elements required by the Big Data Application Provider and compute a matrix of data or may actually generate some rendering like a heat map to support the visualization component. To permit rapid display, the data dissemination phase almost certainly does some rendering, but the extent depends on the nature of the data and the visualization.

For the purposes of this discussion, most processing frameworks can be described with respect to their primary location within the information flow illustrated in Figure 5.

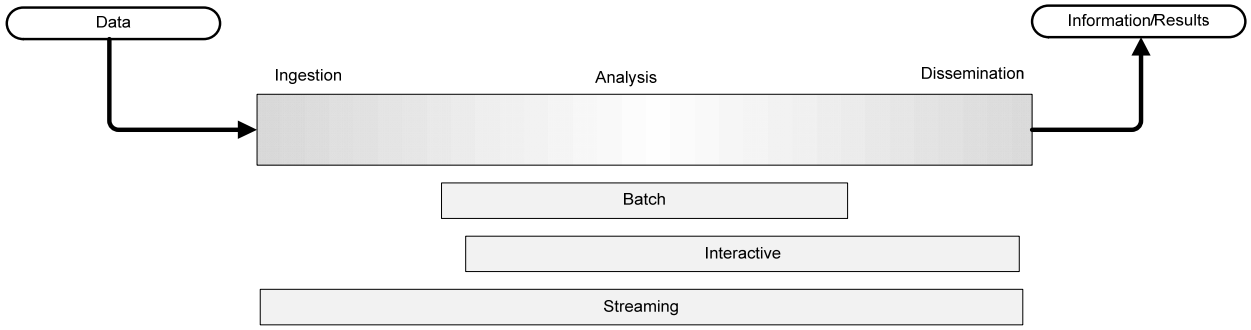


Figure 5: Information Flow

The green shading in Figure 5 illustrates the general sensitivity of that processing phase to latency, which is defined as the time from when a request or piece of data arrives at a system until its processing/delivery

is complete. For Big Data, the ingestion may or may not require NRT performance to keep up with the data flow. Some types of analytics (specifically those categorized as CEP) may or may not require NRT processing. The Data Consumer generally is located at the far right of Figure 5, depending upon the use case and application batch responses (e.g., a nightly report is emailed) may be sufficient. In other cases, the user may be willing to wait minutes for the results of a query to be returned, or they may need immediate alerting when critical information arrives at the system. In general, batch analytics tend to better support long term strategic decision making, where the overall view or direction is not affected by the latest small changes in the underlying data. Streaming analytics are better suited for tactical decision making, where new data needs to be acted upon immediately. A primary use case for streaming analytics would be electronic trading on stock exchanges where the window to act on a given piece of data can be measured in microseconds. Messaging and communication provides the transfer of data between processing elements and the buffering necessary to deal with the deltas in data rate, processing times, and data requests.

Typically, Big Data discussions focus around the categories of batch and streaming frameworks for analytics. However, frameworks for retrieval of data that provide interactive access to Big Data are becoming a more prevalent. It is noted that the lines between these categories are not solid or distinct, with some frameworks providing aspects of each category.

4.4.3.1 BATCH FRAMEWORKS

Batch frameworks, whose roots stem from the mainframe processing era, are some of the most prevalent and mature components of a Big Data architecture because the historically long processing times for large data volumes. Batch frameworks ideally are not tied to a particular algorithm or even algorithm type, but rather provide a programming model where multiple classes of algorithms can be implemented. Also, when discussed in terms of Big Data, these processing models are frequently distributed across multiple nodes of a cluster. They are routinely differentiated by the amount of data sharing between processes/activities within the model.

In 2004, a list of algorithms for simulation in the physical sciences was developed that became known as the "Seven Dwarfs".[3] The list was recently modified and extended to the 13 algorithms (Table 2), based on the following definition: "A dwarf is an algorithmic method that computes a pattern of computation and communication."[4]

Table 2: 13 Dwarfs—Algorithms for Simulation in the Physical Sciences

Dense Linear Algebra*	Combinational Logic
Sparse Linear Algebra*	Graph Traversal
Spectral methods	Dynamic Programming
N-Body Methods	Backtrack and Branch-and-Bound
Structured Grids*	Graphical Models
Unstructured Grids*	Finite State Machines
Map/Reduce	

Notes:
* Indicates one of the original seven dwarfs. The recent list modification removed three of the original seven algorithms: Fast Fourier Transform, Particles, and Monte Carlo.

Many other algorithms or processing models have been defined over the years two of the best-known models in the Big Data space, Map/Reduce and Bulk Synchronous Parallel (BSP), are described in the following subsections.

4.4.3.1.1 Map/Reduce

Several major Internet search providers popularized the Map/Reduce model as they worked to implement their search capabilities. In general, Map/Reduce programs follow five basic stages:

1. Input preparation and assignment to mappers;
2. Map a set of keys and values to new keys and values: Map(k1,v1) → list(k2,v2);
3. Shuffle data to each reducer and each reducer sorts its input—each reducer is assigned a set of keys (k2);
4. Run the reduce on a list(v2) associated with each key and produce an output: Reduce(k2, list(v2)) → list(v3); and
5. Final output the lists(v3) from each reducer are combined and sorted by k2.

While there is a single output, nothing in the model prohibits multiple input data sets. It is extremely common for complex analytics to be built as workflows of multiple Map/Reduce jobs. While the Map/Reduce programming model is best suited to aggregation (e.g., sum, average, group-by) type analytics, a wide variety of analytic algorithms have been implemented within processing frameworks. Map/Reduce does not generally perform well with applications or algorithms that need to directly update the underlying data. For example, updating the values for a single key would require the entire data set be read, output, and then moved or copied over the original data set. Because the mappers and reducers are stateless in nature, applications that require iterative computation on parts of the data or repeated access to parts of the data set do not tend to scale or perform well under Map/Reduce.

Due to its shared nothing approach, the usability of Map/Reduce for Big Data applications has made it popular enough that a number of large data storage solutions (mostly those of the NoSQL variety) provide implementations within their architecture. One major criticism of Map/Reduce early on was that the interfaces to most implementations were at too low of a level (written in Java or JavaScript.) However many of the more prevalent implementations now support high-level procedural and declarative language interfaces and even visual programming environments are beginning to appear.

4.4.3.1.2 Bulk Synchronous Parallel

The BSP programming model, originally developed by Leslie Valiant[5], combines parallel processing with the ability of processing modules to send messages to other processing modules and explicit synchronization of the steps. A BSP algorithm is composed of what are termed "supersteps," which comprise the following three distinct elements.

- Bulk Parallel Computation: Each processor performs the calculation/analysis on its local chunk of data.
- Message Passing: As each processor performs its calculations it may generate messages to other processors. These messages are frequently updates to values associated with the local data of other processors but may also result in the creation of additional data.
- Synchronization: Once a processor has completed processing its local data it pauses until all other processors have also completed their processing.

This cycle can be terminated by all the processors "voting to stop", which will generally happen when a processor has generated no messages to other processors (e.g., no updates). All processors voting to stop in turn indicates that there are no new updates to any processors' data and the computation is complete. Alternatively, the cycle may be terminated after a fixed number of supersteps have been completed (e.g., after a certain number of iterations of a Monte Carlo simulation).

The advantage of BSP over Map/Reduce is that processing can actually create updates to the data being processed. It is this distinction that has made BSP popular for graph processing and simulations where computations on one node/element of data directly affect values or connections with other nodes/elements. The disadvantage of BSP is the high cost of the synchronization barrier between supersteps. Should the distribution of data or processing between processors become highly unbalanced then some processors may become overloaded while others remain idle. While high-performance interconnect technologies help to reduce the cost of this synchronization through faster data exchange

between nodes and can allow for re-distribution of data during a super-step skewing of the processing requirements, the fastest possible performance of any given super step is lower bounded by the slowest performance of any processing unit. Essentially, if the data is skewed such that the processing of a given data element (say traversal of the graph from that element) is especially long-running, the next super step cannot begin until that nodes processing completes.

Numerous extensions and enhancements to the basic BSP model have been developed and implemented over the years, many of which are designed to address the balancing and cost of synchronization problems.

4.4.3.2 STREAMING FRAMEWORKS

Streaming frameworks are built to deal with data that requires processing as fast or faster than the velocity at which it arrives into the Big Data system. The primary goal of streaming frameworks is to reduce the latency between the arrival of data into the system and the creation, storage, or presentation of the results. CEP is one of the problem domains frequently addressed by streaming frameworks. CEP uses data from one or more streams/sources to infer or identify events or patterns in NRT.

Almost all streaming frameworks for Big Data available today implement some form of basic workflow processing for the streams. These workflows use messaging/communications frameworks to pass data objects (often referred to as events) between steps in the workflow. This frequently takes the form of a directed execution graph. The distinguishing characteristics of streaming frameworks are typically organized around the following three characteristics: event ordering and processing guarantees, state management, and partitioning/parallelism. These three characteristics are described below.

4.4.3.2.1 Event Ordering and Processing Guarantees

This characteristic refers to whether stream processing elements are guaranteed to see messages or events in the order they are received by the Big Data System, as well as how often a message or event may or may not be processed. In a non-distributed and single stream mode, this type of guarantee is relatively trivial. Once distributed and/or multiple streams are added to the system, the guarantee becomes more complicated. With distributed processing, the guarantees must be enforced for each partition of the data (partitioning and parallelism as further described below.) Complications arise when the process/task/job dealing with a partition dies. Processing guarantees are typically divided into the following three classes:

- At-most-once delivery: This is the simplest form of guarantee and allows for messages or events to be dropped if there is a failure in processing or communications or if they arrive out of order. This class of guarantee is applicable for data where there is no dependence of new events on the state of the data created by prior events.
- At-least-once delivery: Within this class, the frameworks will track each message or event (and any downstream messages or events generated) to verify that it is processed within a configured time frame. Messages or events that are not processed in the time allowed are re-introduced into the stream. This mode requires extensive state management by the framework (and sometimes the associated application) to track which events have been processed by which stages of the workflow. However, under this class, messages or events may be processed more than once and also may arrive out of order. This class of guarantee is appropriate for systems where every message or event must be processed regardless of the order (e.g., no dependence on prior events), and the application either is not affected by duplicate processing of events or has the ability to de-duplicate events itself.

- Exactly-once delivery: This class of framework processing requires the same top level state tracking as At-least-once delivery but embeds mechanisms within the framework to detect and ignore duplicates. This class often guarantees ordering of event arrivals and is required for applications where the processing of any given event is dependent on the processing of prior events. It is noted that these guarantees only apply to data handling within the framework. If data is passed outside the framework processing topology, then by an application then the application must ensure the processing state is maintained by the topology or duplicate data may be forwarded to non-framework elements of the application.

In the latter two classes, some form of unique key must be associated with each message or event to support de-duplication and event ordering. Often, this key will contain some form of timestamp plus the stream ID to uniquely identify each message in the stream.

4.4.3.2.2 State Management

A critical characteristic of stream processing frameworks is their ability to recover and not lose critical data in the event of a process or node failure within the framework. Frameworks typically provide this state management through persistence of the data to some form of storage. This persistence can be: local, allowing the failed process to be restarted on the same node; a remote or distributed data store, allowing the process to be restarted on any node; or, local storage that is replicated to other nodes. The trade-off between these storage methods is the latency introduced by the persistence. Both the amount of state data persisted and the time required to assure that the data is persisted contribute to the latency. In the case of a remote or distributed data store, the latency required is generally dependent on the extent to which the data store implements ACID (Atomicity, Consistency, Isolation, Durability) or BASE (Basically Available, Soft state, Eventual consistency)style consistency. With replication of local storage, the reliability of the state management is entirely tied to the ability of the replication to recover in the event of a process or node failure. Sometimes this state replication is actually implemented using the same messaging/communication framework that is used to communicate with and between stream processors. Some frameworks actually support full transaction semantics, including multi-stage commits and transaction rollbacks. The trade-off is the same one that exists for any transaction system is that any type of ACID-like guarantee will introduce latency. Too much latency at any point in the stream flow can create bottlenecks and, depending on the ordering or processing guarantees, can result in deadlock or loop states—especially when some level of failure is present.

4.4.3.2.3 Partitioning and Parallelism

This streaming framework characteristic relates to the distribution of data across nodes and worker tasks to provide the horizontal scalability needed to address the volume and velocity of Big Data streams. This partitioning scheme must interact with the resource management framework to allocate resources. The even distribution of data across partitions is essential so that the associated work is evenly distributed. The even data distribution directly relates to selection of a key (e.g., user ID, host name) that can be evenly distributed. The simplest form might be using a number that increments by one and then is processed with a modulus function of the number of tasks/workers available. If data dependencies require all records with a common key be processed by the same worker, then assuring an even data distribution over the life of the stream can be difficult. Some streaming frameworks address this issue by supporting dynamic partitioning where the partition of overloaded workers is split and allocated to existing workers or newly created workers. To achieve success—especially with a data/state dependency related to the key—it is critical that the framework have state management, which allows the associated state data to be moved/transitioned to the new/different worker.

4.4.4 MESSAGING/COMMUNICATIONS FRAMEWORKS

Messaging and communications frameworks have their roots in the High Performance Computing (HPC) environments long popular in the scientific and research communities. Messaging/Communications

Frameworks were developed to provide APIs for the reliable queuing, transmission, and receipt of data between nodes in a horizontally scaled cluster. These frameworks typically implement either a point-to-point transfer model or a store-and-forward model in their architecture. Under a point-to-point model, data is transferred directly from the sender to the receivers. The majority of point-to-point implementations do not provide for any form of message recovery should there be a program crash or interruption in the communications link between sender and receiver. These frameworks typically implement all logic within the sender and receiver program space, including any delivery guarantees or message retransmission capabilities. One common variation of this model is the implementation of multicast (i.e., one-to-many or many-to-many distribution), which allows the sender to broadcast the messages over a "channel", and receivers in turn listen to those channels of interest. Typically, multicast messaging does not implement any form of guaranteed receipt. With the store-and-forward model, the sender would address the message to one or more receivers and send it to an intermediate broker, which would store the message and then forward it on to the receivers. Many of these implementations support some form of persistence for messages not yet delivered, providing for recovery in the event of process or system failure. Multicast messaging can also be implemented in this model and is frequently referred to as a pub/sub model.

4.4.5 RESOURCE MANAGEMENT FRAMEWORK

As Big Data systems have evolved and become more complex, and as businesses work to leverage limited computation and storage resources to address a broader range of applications and business challenges, the requirement to effectively manage those resources has grown significantly. While tools for resource management and "elastic computing" have expanded and matured in response to the needs of cloud providers and virtualization technologies, Big Data introduces unique requirements for these tools. However, Big Data frameworks tend to fall more into a distributed computing paradigm, which presents additional challenges.

The Big Data characteristics of volume and velocity drive the requirements with respect to Big Data resource management. Elastic computing (i.e., spawning another instance of some service) is the most common approach to address expansion in volume or velocity of data entering the system. CPU and memory are the two resources that tend to be most essential to managing Big Data situations. While shortages or over-allocation of either will have significant impacts on system performance, improper or inefficient memory management is frequently catastrophic. Big Data differs and becomes more complex in the allocation of computing resources to different storage or processing frameworks that are optimized for specific applications and data structures. As such, resource management frameworks will often use data locality as one of the input variables in determining where new processing framework elements (e.g., master nodes, processing nodes, job slots) are instantiated. Importantly, because the data is big (i.e., large volume), it generally is not feasible to move data to the processing frameworks. In addition, while nearly all Big Data processing frameworks can be run in virtualized environments, most are designed to run on bare metal commodity hardware to provide efficient I/O for the volume of the data.

Two distinct approaches to resource management in Big Data frameworks are evolving. The first is intra-framework resource management, where the framework itself manages allocation of resources between its various components. This allocation is typically driven by the framework's workload and often seeks to "turn off" unneeded resources to either minimize overall demands of the framework on the system or to minimize the operating cost of the system by reducing energy use. With this approach, applications can seek to schedule and request resources that—much like main frame OSs of the past—are managed through scheduling queues and job classes.

The second approach is inter-framework resource management, which is designed to address the needs of many Big Data systems to support multiple storage and processing frameworks that can address and be optimized for a wide range of applications. With this approach, the resource management framework actually runs as a service that supports and manages resource requests from frameworks, monitoring

framework resource usage, and in some cases manages application queues. In many ways, this approach is like the resource management layers common in cloud/virtualization environments, and there are efforts underway to create hybrid resource management frameworks that handle both physical and virtual resources.

Taking these concepts further and combining them is resulting in the emerging technologies built around what is being termed software-defined data centers (SDDCs). This expansion on elastic and cloud computing goes beyond the management of fixed pools of physical resources as virtual resources to include the automated deployment and provisioning of features and capabilities onto physical resources. For example, automated deployment tools that interface with virtualization or other framework APIs can be used to automatically stand up entire clusters or to add additional physical resources to physical or virtual clusters.

4.5 DATA CONSUMER

Similar to the Data Provider, the role of Data Consumer within the NBDRA can be an actual end user or another system. In many ways, this role is the mirror image of the Data Provider, with the entire Big Data framework appearing like a Data Provider to the Data Consumer. The activities associated with the Data Consumer role include the following:

- Search and Retrieve
- Download
- Analyze Locally
- Reporting
- Visualization
- Data to Use for Their Own Processes

The Data Consumer uses the interfaces or services provided by the Big Data Application Provider to get access to the information of interest. These interfaces can include data reporting, data retrieval, and data rendering.

This role will generally interact with the Big Data Application Provider through its access function to execute the analytics and visualizations implemented by the Big Data Application Provider. This interaction may be demand-based, where the Data Consumer initiates the command/transaction and the Big Data Application Provider replies with the answer. The interaction could include interactive visualizations, creating reports, or drilling down through data using business intelligence functions provided by the Big Data Application Provider. Alternately, the interaction may be stream- or push-based, where the Data Consumer simply subscribes or listens for one or more automated outputs from the application. In almost all cases, the Security and Privacy fabric around the Big Data architecture would support the authentication and authorization between the Data Consumer and the architecture, with either side able to perform the role of authenticator/authorizer and the other side providing the credentials. Like the interface between the Big Data architecture and the Data Provider, the interface between the Data Consumer and Big Data Application Provider would also pass through the three distinct phases of initiation, data transfer, and termination.

5 MANAGEMENT FABRIC OF THE NBDRA

The Big Data characteristics of volume, velocity, variety, and variability demand a versatile management platform for storing, processing, and managing complex data. Management of Big Data systems should handle both system- and data-related aspects of the Big Data environment. The Management Fabric of the NBDRA encompasses two general groups of activities: system management and Big Data life cycle management (BDLM). System management includes activities such as provisioning, configuration, package management, software management, backup management, capability management, resources management, and performance management. BDLM involves activities surrounding the data life cycle of collection, preparation/curation, analytics, visualization, and access.

As discussed above, the NBDRA represents a broad range of Big Data systems—from tightly coupled enterprise solutions integrated by standard or proprietary interfaces to loosely coupled vertical systems maintained by a variety of stakeholders or authorities bound by agreements, standard interfaces, or de facto standard interfaces. Therefore, different considerations and technical solutions would be applicable for different cases.

5.1 SYSTEM MANAGEMENT

The characteristics of Big Data pose system management challenges on traditional management platforms. To efficiently capture, store, process, analyze, and distribute complex and large datasets arriving or leaving with high velocity, a resilient system management is needed.

As in traditional systems, system management for Big Data architecture involves provisioning, configuration, package management, software management, backup management, capability management, resources management, and performance management of the Big Data infrastructure, including compute nodes, storage nodes, and network devices. Due to the distributed and complex nature of the Big Data infrastructure, system management for Big Data is challenging, especially with respect to the capability for controlling, scheduling, and managing the processing frameworks to perform the scalable, robust, and secure analytics processing required by the Big Data Application Provider. The Big Data infrastructure may contain SAN or NAS storage devices, cloud storage spaces, NoSQL databases, Map/Reduce clusters, data analytics functions, search and indexing engines, and messaging platforms. The supporting enterprise computing infrastructure can range from traditional data centers, cloud services, and dispersed computing nodes of a grid. To manage the distributed and complex nature of the Big Data infrastructure, system management relies on the following:

- Standard protocols such as Simple Network Management Protocol (SNMP), which are used to transmit status about resources and fault information to the management fabric components; and
- Deployable agents or management connectors which allow the management fabric to both monitor and also control elements of the framework.

These two items aid in monitoring the health of various types of computing resources and coping with performance and failures incidents while maintaining the quality of service levels required by the Big Data Application Provider. Management connectors are necessary for scenarios where the CSPs expose management capabilities via APIs. It is conceivable that the infrastructure elements contain autonomic, self-tuning, and self-healing capabilities, thereby reducing the centralized model of system management. In large infrastructures with many thousands of computing and storage nodes, the provisioning of tools and applications should be as automated as possible. Software installation, application configuration, and regular patch maintenance should be pushed out and replicated across the nodes in an automated fashion, which could be done based on the topology knowledge of the infrastructure. With the advent of

virtualization, the utilization of virtual images may speed up the recovery process and provide efficient patching that can minimize downtime for scheduled maintenance.

In an enterprise environment, the management platform would typically provide enterprise-wide monitoring and administration of the Big Data distributed components. This includes network management, fault management, configuration management, system accounting, performance management, and security management.

In a loosely coupled vertical system, each independent stakeholder is responsible for its own system management, security, and integration. Each stakeholder is responsible for integration within the Big Data distributed system using the interfaces provided by other stakeholders.

5.2 BIG DATA LIFE CYCLE MANAGEMENT

BDLM faces more challenges compared to traditional data life cycle management (DLM), which may require less data transfer, processing, and storage. However, BDLM still inherits the DLM phases in terms of data acquisition, distribution, use, migration, maintenance, and disposition—but at a much bigger processing scale. Big Data Application Providers may require much more computational processing for collection, preparation/curation, analytics, visualization, and access to be able to use the analytic results. In other words, the BDLM activity includes verification that the data are handled correctly by other NBDRA components in each process within the data life cycle—from the moment they are ingested into the system by the Data Provider, until the data are processed or removed from the system.

The importance of BDLM to Big Data is demonstrated through the following considerations:

- Data volume can be extremely large that may overwhelm the storage capacity, or make storing incoming data prohibitively expensive.
- Data velocity, the rate at which data can be captured and ingested into the system, can overwhelm available storage space at a given time. Even with the elastic storage service provided by cloud computing for handling dynamic storage needs, uncontrolled data management may also be unnecessarily costly for certain application requirements.
- Different Big Data applications will likely have different requirements for the lifetime of a piece of data. The differing requirements have implications on how often data must be refreshed so that processing results are valid and useful. In data refreshment, old data are dispositioned and not fed into analytics or discovery programs. At the same time, new data is ingested and taken into account by the computations. For example, real-time applications will need very short data lifetime but a market study of consumers' interest in a product line may need to mine data collected over a longer period of time.

Because the task of BDLM can be distributed among different organizations and/or individuals within the Big Data computing environment, coordination of data processing between NBDRA components has greater difficulty in complying with policies, regulations, and security requirements. Within this context, BDLM may need to include the following sub-activities:

- **Policy Management:** Captures the requirements for the data life cycle that allows old data to be dispositioned and new data to be considered by Big Data applications. Maintains the migration and disposition strategies that specify the mechanism for data transformation and dispositioning, including transcoding data, transferring old data to lower-tier storage for archival purpose, removing data, or marking data as in situ.
- **Metadata Management:** Enables BDLM, since metadata are used to store information that governs the management of the data within the system. Essential metadata information includes persistent identification of the data, fixity/quality, and access rights. The challenge is to find the minimum set of elements to execute the required BDLM strategy in an efficient manner.

- **Accessibility Management** Change of data accessibility over time: For example, census data can be made available to the public after 72 years.[6] BDLM is responsible for triggering the accessibility update of the data or sets of data according to policy and legal requirements. Normally, data accessibility information is stored in the metadata.
- **Data Recovery**: BDLM can include the recovery of data that were lost due to disaster or system/storage fault. Traditionally, data recovery can be achieved using regular backup and restore mechanisms. However, given the large volume of Big Data, traditional backup may not be feasible. Instead, replication may have to be designed within the Big Data ecosystem. Depending on the tolerance of data loss—each application has its own tolerance level—replication strategies have to be designed. The replication strategy includes the replication window time, the selected data to be replicated, and the requirements for geographic disparity. Additionally, in order to cope with the large volume of Big Data, data backup and recovery should consider the use of modern technologies within the Big Data Framework Provider.
- **Preservation Management:** The system maintains data integrity so that the veracity and velocity of the analytics process are fulfilled. Due to the extremely large volume of Big Data, preservation management is responsible for disposition-aged data contained in the system. Depending on the retention policy, these aged data can be deleted or migrated to archival storage. In the case where data must be retained for years, decades, and even centuries, a preservation strategy will be needed so the data can be accessed by the provider components if required. This will invoke long-term digital preservation that can be performed by Big Data Application Providers using the resources of the Big Data Framework Provider.

In the context of Big Data, BDLM contends with the Big Data characteristics of volume, velocity, variety, and variability. As such, BDLM and its sub-activities interact with other components of the NBDRA as shown in the following examples:

- **System Orchestrator:** BDLM enables data scientists to initiate any combination of processing including accessibility management, data backup/recovery, and preservation management. The process may involve other components of the NBDRA, such as Big Data Application Provider and Big Data Framework Provider. For example, data scientists may want to interact with the Big Data Application Provider for data collection and curation, invoke the Big Data Framework Provider to perform certain analysis, and grant access to certain users to access the analytic results from the Data Consumer.
- **Data Provider:** BDLM manages ingestion of data and metadata from the data source(s) into the Big Data system, which may include logging the entry event in the metadata by the Data Provider.
- **Big Data Application Provider:** BDLM executes data masking and format transformations for data preparation or curation purpose.
- **Big Data Framework Provider:** BDLM executes basic bit-level preservation and data backup and recovery according to the recovery strategy.
- **Data Consumer:** BDLM ensures that relevant data and analytic results are available with proper access control for consumers and software agents to consume within the BDLM policy strategy.
- **Security and Privacy Fabric:** Keeps the BDLM up to date according to new security policy and regulations.

The Security and Privacy Fabric also uses information coming from BDLM with respect to data accessibility. The Security and Privacy Fabric control access to the functions and data usage produced by the Big Data system. This data access control can be informed by the can use metadata, which is managed and updated by BDLM.

6 SECURITY AND PRIVACY FABRIC OF THE NBDRA

Security and privacy considerations form a fundamental aspect of the NBDRA. This is geometrically depicted in Figure 2 by the Security and Privacy Fabric surrounding the five main components, indicating that all components are affected by security and privacy considerations. Thus, the role of security and privacy is correctly depicted in relation to the components but does not expand into finer details, which may be more accurate but are best relegated to a more detailed security and privacy reference architecture. The Data Provider and Data Consumer are included in the Security and Privacy Fabric since, at the least, they may often nominally agree on security protocols and mechanisms. The Security and Privacy Fabric is an approximate representation that alludes to the intricate interconnected nature and ubiquity of security and privacy throughout the NBDRA. Additional details about the Security and Privacy Fabric are included in the *NIST Interoperability Framework: Volume 4, Security and Privacy* document.

7 CONCLUSION

The NBD-PWG Reference Architecture Subgroup prepared this *NIST Big Data Interoperability Framework: Volume 6, Reference Architecture* to provide a vendor-neutral, technology- and infrastructure-agnostic conceptual model and examine related issues. The conceptual model, referred to as the NBDRA, was a collaborative effort within the Subgroup and with the other NBD-PWG subgroups. The goal of the NBD-PWG Reference Architecture Subgroup is to develop an open reference architecture for Big Data that achieves the following objectives:

- Provides a common language for the various stakeholders;
- Encourages adherence to common standards, specifications, and patterns;
- Provides consistent methods for implementation of technology to solve similar problem sets;
- Illustrates and improves understanding of the various Big Data components, processes, and systems, in the context of a vendor- and technology- agnostic Big Data conceptual model;
- Provides a technical reference for U.S. government departments, agencies, and other consumers to understand, discuss, categorize, and compare Big Data solutions; and
- Facilitates analysis of candidate standards for interoperability, portability, reusability, and extendibility.

This document presents the results of the first stage of NBDRA development. The *NIST Big Data Interoperability Framework* will be released in three versions, which correspond to the three stages of the NBD-PWG work. The three stages aim to achieve the following:

Stage 1: Identify the common reference architecture components of Big Data implementations and formulate the technology-independent NBDRA;
Stage 2: Define general interfaces between the NBDRA components; and
Stage 3: Validate the NBDRA by building Big Data general applications through the general interfaces.

This document (Version 1) presents the overall NBDRA components and fabrics with high-level description and functionalities.

Version 2 activities will focus on the definition of general interfaces between the NBDRA components by achieving the following:

- Select use cases from the 62 (51 general and 11 security and privacy) submitted use cases or other, to be identified, meaningful use cases.
- Work with domain experts to identify workflow and interactions from the System Orchestrator to the rest of the NBDRA components.
- Implement small scale, manageable, and well-defined confined environment and model interaction between NBDRA components and fabrics.
- Aggregate the common data workflow and interaction between NBDRA components/fabrics and package them into general interfaces.

Version 3 activities will focus on validation of the NBDRA through the use of the defined NBDRA interfaces to build general Big Data applications. The validation strategy will include the following:

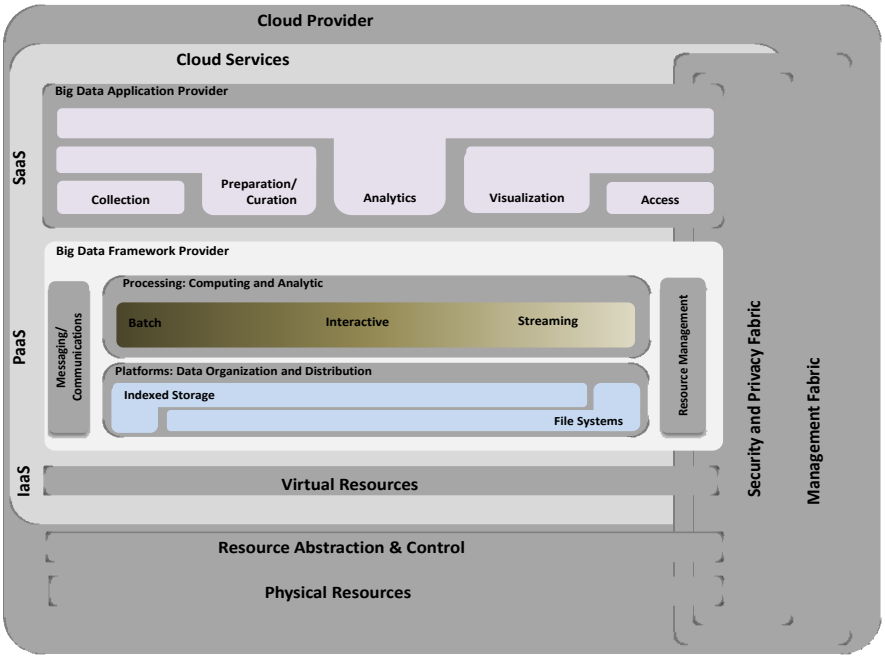
- Implement the same set of use cases used in Version 2 by using the defined general interfaces.
- Identify and implement a few new use cases outside the Version 2 scenarios.
- Enhance general NBDRA interfaces through lessons learned from the implementations in Version 3 activities.

The general interfaces developed during Version 3 activities will offer a starting point for further refinement by any interested parties and is not intended to be a definitive solution to address all implementation needs.

Appendix A: Deployment Considerations

The NIST Big Data Reference Architecture is applicable to a variety of business environments and technologies. As a result, possible deployment models are not part of the core concepts discussed in the main body of this document. However, the loosely coupled and distributed natures of Big Data Framework Provider functional components allows it to be deployed using multiple infrastructure elements as described in Section 4.4.1. The two most common deployment configurations are directly on physical resources or on top of an IaaS cloud computing framework. The choices between these two configurations are driven by needs of efficiency/performance and elasticity. Physical infrastructures are typically used to obtain predictable performance and efficient utilization of CPU and I/O bandwidth since it eliminates the overhead and additional abstraction layers typical in the virtualized environments for most IaaS implementations. IaaS cloud based deployments on are typically used when elasticity is needed to support changes in workload requirements. The ability to rapidly instantiate additional processing nodes or framework components allows the deployment to adapt to either increased or decreased workloads. By allowing the deployment footprint to grow or shrink based on workload demands this deployment model can provide cost savings when public or shared cloud services are used and more efficient use and energy consumption when a private cloud deployment is used. Recently, a hybrid deployment model known as Cloud Bursting has become popular. In this model a physical deployment is augmented by either public or private IaaS cloud services. When additional processing is needed to support the workload additional the additional framework component instances are established on the IaaS infrastructure and then deleted when no longer require.

Figure A-1: Big Data Framework Deployment Options



In addition to providing IaaS support, cloud providers are now offering Big Data Frameworks under a platform as a service (PaaS) model. Under this model, the system implementer is freed from the need to establish and manage the complex configuration and deployment typical of many Big Data Framework components. The implementer simply needs to specify the size of the cluster required, and the cloud provider manages the provisioning, configuration, and deployment of all the framework components. There are even some nascent offerings for specialized software as a service (SaaS) Big Data applications appearing in the market that implement the Big Data Application Provider functionality within the cloud

environment. Figure A-1 illustrates how the components of the NBDRA might align onto the NIST Cloud Reference architecture.[7] The following sections describe some of the high-level interactions required between the Big Data Architecture elements and the CSP elements.

CLOUD SERVICE PROVIDERS

Recent data analytics solutions use algorithms that can utilize and benefit from the frameworks of the cloud computing systems. Cloud computing has essential characteristics such as rapid elasticity and scalability, multi-tenancy, on-demand self-service and resource pooling, which together can significantly lower the barriers to the realization of Big Data implementations.

The **CSP** implements and delivers **cloud services**. Processing of a service invocation is done by means of an instance of the service implementation, which may involve the composition and invocation of other services as determined by the design and configuration of the service implementation.

Cloud Service Component

The cloud service component contains the implementation of the cloud services provided by a CSP. It contains and controls the software components that implement the services (but not the underlying hypervisors, host OSs, device drivers, etc.).

Cloud services can be described in terms of service categories.

Cloud services are also grouped into categories, where each service category is characterized by qualities that are common between the services within the category. The NIST Cloud Computing Reference Model defines the following cloud service categories:

- Infrastructure as a services (IaaS)
- Platform as a service (PaaS)
- Software as a service (SaaS)

Resource Abstraction and Control Component

The Resource Abstraction and Control component is used by CSPs to provide access to the physical computing resources through software abstraction. Resource abstraction needs to assure efficient, secure, and reliable usage of the underlying physical resources. The control feature of the component enables the management of the resource abstraction features.

The Resource Abstraction and Control component enables a CSP to offer qualities such as rapid elasticity, resource pooling, on-demand self-service and scale-out. The Resource Abstraction and Control component can include software elements such as hypervisors, virtual machines, virtual data storage, and time-sharing.

The Resource Abstraction and Control component enables control functionality. For example, there may be a centralized algorithm to control, correlate, and connect various processing, storage, and networking units in the physical resources so that together they deliver an environment where IaaS, PaaS or SaaS cloud service categories can be offered. The controller might decide which CPUs/racks contain which virtual machines executing which parts of a given cloud workload, and how such processing units are connected to each other, and when to dynamically and transparently reassign parts of the workload to new units as conditions change.

Security and Privacy and Management Functions

In almost all cases, the Cloud Provider will provide elements of the Security, Privacy, and Management functions. Typically the provider will support high-level security/privacy functions that control access to the Big Data Applications and Frameworks while the frameworks themselves must control access to their underlying data and application services. Many times the Big Data specific functions for security and

privacy will depend on and must interface with functions provided by the CSP. Similarly, management functions are often split between the Big Data implementation and the Cloud Provider implementations. Here the cloud provider would handle the deployment and provisioning of Big Data architecture elements within its IaaS infrastructure. The cloud provider may provide high-level monitoring functions to allow the Big Data implementation to track performance and resource usage of its components. In, many cases the Resource Management element of the Big Data Framework will need to interface to the CSP's management framework to request additional resources.

PHYSICAL RESOURCE DEPLOYMENTS

As stated above, deployment on physical resources is frequently used when performance characteristics are paramount. The nature of the underlying physical resource implementations to support Big Data requirements has evolved significantly over the years. Specialized, high-performance super computers with custom approaches for sharing resources (e.g., memory, CPU, storage) between nodes has given way to shared nothing computing clusters built from commodity servers. The custom super computing architectures almost always required custom development and components to take advantage of the shared resources. The commodity server approach both reduced the hardware investment and allowed the Big Data frameworks to provide higher-level abstractions for the sharing and management of resources in the cluster. The Recent trends now involve density, power, cooling optimized server form factors that seek to maximize the available computing resources while minimizing size, power and/or cooling requirements. This approach retains the abstraction and portability advantages of the shared nothing approaches while providing improved efficiency.

Appendix B: Terms and Definitions

NBDRA COMPONENTS

- **Big Data Engineering:** Advanced techniques that harness independent resources for building scalable data systems when the characteristics of the datasets require new architectures for efficient storage, manipulation, and analysis.
- **Data Provider:** Organization or entity that introduces information feeds into the Big Data system for discovery, access, and transformation by the Big Data system.
- **Big Data Application Provider:** Organization or entity that executes a generic vertical system data life cycle, including: (a) data collection from various sources, (b) multiple data transformations being implemented using both traditional and new technologies, (c) diverse data usage, and (d) data archiving.
- **Big Data Framework Provider:** Organization or entity that provides a computing fabric (such as system hardware, network, storage, virtualization, and computing platform) to execute certain Big Data applications, while maintaining security and privacy requirements.
- **Data Consumer:** End users or other systems that use the results of data applications.
- **System Orchestrator:** Organization or entity that defines and integrates the required data transformations components into an operational vertical system.

OPERATIONAL CHARACTERISTICS

- Interoperability: The capability to communicate, to execute programs, or to transfer data among various functional units under specified conditions.
- Portability: The ability to transfer data from one system to another without being required to recreate or reenter data descriptions or to modify significantly the application being transported.
- Privacy: The assured, proper, and consistent collection, processing, communication, use and disposition of data associated with personal information and PII throughout its life cycle.
- Security: Protecting data, information, and systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide:
  - Integrity: guarding against improper data modification or destruction, and includes ensuring data nonrepudiation and authenticity;
  - Confidentiality: preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary data; and
  - Availability: ensuring timely and reliable access to and use of data.
- Elasticity: The ability to dynamically scale up and down as a real-time response to the workload demand. Elasticity will depend on the Big Data system, but adding or removing "software threads" and "virtual or physical servers" are two widely used scaling techniques. Many types of workload demands drive elastic responses, including web-based users, software agents, and periodic batch jobs.
- Persistence: The placement/storage of data in a medium design to allow its future access.

PROVISIONING MODELS

- IaaS: The capability provided to the consumer to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include OS and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over OSs, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).[8]

- PaaS: The capability provided to the consumer to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, OS, or storage, but has control over the deployed applications and possibly application hosting environment configurations. (Source: NIST CC Definition)
- SaaS: The capability provided to the consumer to use applications running on a cloud infrastructure. The consumer does not manage or control the underlying cloud infrastructure including network, servers, OSs, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. (Source: NIST CC Definition)

Appendix C: Examples of Big Data Organization Approaches

This appendix provides an overview of several common Big Data Organization Approaches as follows:

- Relational storage models
- Key-value storage models
- Columnar storage models
- Document Storage
- Graph Stores

The reader should keep in mind that new and innovative approaches are emerging regularly, and that some of these approaches are hybrid models that combine features of several indexing techniques (e.g., relational and columnar, or relational and graph).

RELATIONAL STORAGE MODELS

This model is perhaps the most familiar to folks as the basic concept has existed since the 1950s and the SQL is a mature standard for manipulating (search, insert, update, delete) relational data. In the relational model, data is stored as rows with each field representing a column organized into Table based on the logical data organization. The problem with relational storage models and Big Data is the join between one or more tables. While the size of 2 or more tables of data individually might be small the join (or relational matches) between those tables will generate exponentially more records. The appeal of this model for organizations just adopting Big Data is its familiarity. The pitfalls are some of the limitations and more importantly, the tendency to adopt standard relational database management system (RDBMS) practices (high normalization, detailed and specific indexes) and performance expectations.

Big data implementations of relational storage models are relatively mature and have been adopted by a number of organizations. They are also maturing very rapidly with new implementations focusing on improved response time. Many Big Data implementations take a brute force approach to scaling relational queries. Essentially, queries are broken into stages but more importantly processing of the input tables is distributed across multiple nodes (often as a Map/Reduce job). The actual storage of the data can be flat files (delimited or fixed length) where each record/line in the file represents a row in a table. Increasingly however these implementations are adopting binary storage formats optimized for distributed file systems. These formats will often use block level indexes and column oriented organization of the data to allow individual fields to be accessed in records without needing to read the entire record. Despite this, most Big Data Relational storage models are still "batch oriented" systems designed for very complex queries which generate very large intermediate cross-product matrices from joins so even the simplest query can required 10s of seconds to complete. There is significant work going on and emerging implementations that are seeking to provide a more interactive response and interface.

Early implementations only provided limited data types and little or no support for indexes. However, most current implementations have support for complex data structures and basic indexes. However, while the query planners/optimizers for most modern RDBMS systems are very mature and implement cost-based optimization through statistics on the data the query planners/optimizers in many Big Data implementations remain fairly simple and rule-based in nature. While for batch oriented systems this generally acceptable (since the scale of processing the Big Data in general can be orders of magnitude more an impact) any attempt to provide interactive response will need very advanced optimizations so that (at least for queries) only the most likely data to be returned is actually searched. This of course leads to the single most serious drawback with many of these implementations. Since distributed processing and storage are essential for achieving scalability, these implementations are directly limited by the CAP (Consistency, Availability, and Partition Tolerance) theorem. Many in fact provide what is generally

referred to as a t-eventual consistency which means that barring any updates to a piece of data, all nodes in the distributed system will eventually return the most recent value. This level of consistency is typically fine for Data Warehousing applications where data is infrequently updated and updates are generally done in bulk. However, transaction-oriented databases typically require some level of ACID compliance to ensure that all transactions are handled reliably and conflicts are resolved in a consistent manner. There are a number of both industry and open source initiatives looking to bring this type of capability to Big Data relational storage frameworks. One approach is to essentially layer a traditional RDBMS on top of an existing distributed file system implementation. While vendors claim that this approach means that the overall technology is mature, a great deal of research and implementation experience is needed before the complete performance characteristics of these implementations are known.

Key-Value Storage Models

Key-value stores are one of the oldest and mature data indexing models. In fact, the principles of key value stores underpin all the other storage and indexing models. From a Big Data perspective, these stores effectively represent random access memory models. While the data stored in the values can be arbitrarily complex in structure all the handling of that complexity must be provided by the application with the storage implementation often providing back just a pointer to a block of data. Key-value stores also tend to work best for 1-1 relationships (e.g., each key relates to a single value) but can also be effective for keys mapping to lists of homogeneous values. When keys map multiple values of heterogeneous types/structures or when values from one key need to be joined against values for a different or the same key, then custom application logic is required. It is the requirement for this custom logic that often prevents Key-value stores from scaling effectively for certain problems. However, depending on the problem, certain processing architectures can make effective use of distributed key-value stores. Key-value stores generally deal well with updates when the mapping is one to one and the size/length of the value data does not change. The ability of key-value stores to handle inserts is generally dependent on the underlying implementation. Key-value stores also generally require significant effort (either manual or computational) to deal with changes to the underlying data structure of the values.

Distributed key-value stores are the most frequent implementation utilized in Big Data applications. One problem that must always be addressed (but is not unique to key-value implementations) is the distribution of keys across over the space of possible key values. Specifically, keys must be chosen carefully to avoid skew in the distribution of the data across the cluster. When data is heavily skewed to a small range, it can result in computation hot spots across the cluster if the implementation is attempting to optimize data locality. If the data is dynamic (new keys being added) for such an implementation, then it is likely that at some point the data will require rebalancing across the cluster. Non-locality optimizing implementations employ various sorts of hashing, random, or round-robin approaches to data distribution and don't tend to suffer from skew and hot spots. However, they perform especially poorly on problems requiring aggregation across the data set.

Columnar Storage Models

Much of the hype associated with Big Data came with the publication of the Big Table paper in 2006[9] but column oriented storage models like BigTable are not new to even Big Data and have been stalwarts of the data warehousing domain for many years. Unlike traditional relational data that store data by rows of related values, columnar stores organize data in groups of like values. The difference here is subtle but in relational databases, an entire group of columns are tied to some primary key (frequently one or more of the columns) to create a record. In columnar, the value of every column is a key and like column values point to the associated rows. The simplest instance of a columnar store is little more than a key-value store with the key and value roles reversed. In many ways, columnar data stores look very similar to indexes in relational databases. Figure 5 below shows the basic differences between row-oriented and column-oriented stores.

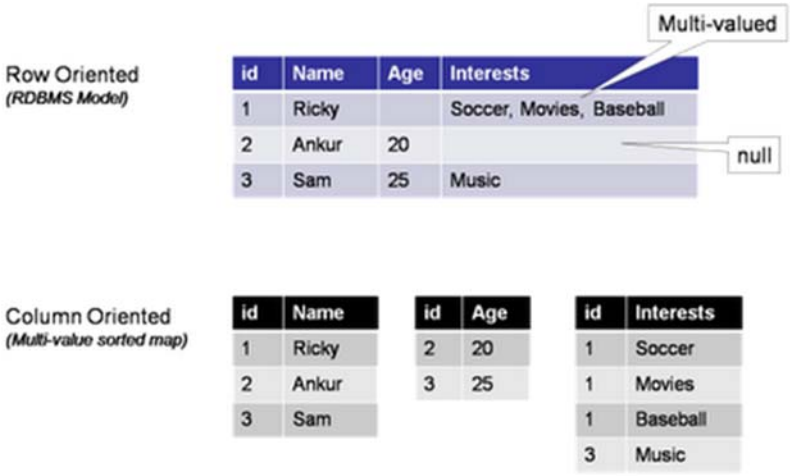


Figure B-1: Differences Between Row Oriented and Column Oriented Stores

In addition, implementations of columnar stores that follow the BigTable model introduce an additional level of segmentation beyond the table, row, and column model of the relational model. That is called the column family. In those implementations, rows have a fixed set of column families but within a column family, each row can have a variable set of columns. This is illustrated in Figure 6 below.

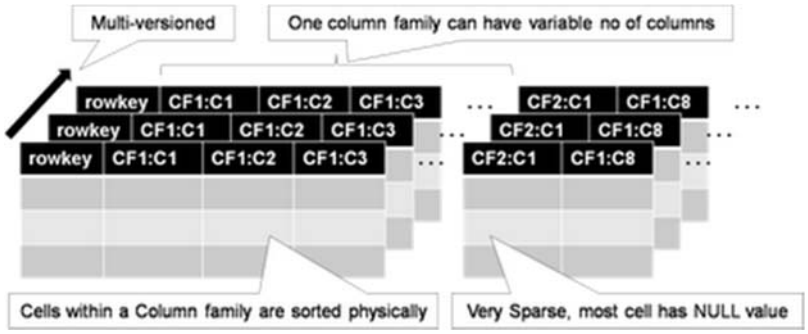


Figure B-2: Column Family Segmentation of the Columnar Stores Model

The key distinction in the implementation of columnar store over relational stores is that data is high de-normalized for column stores and that while for relational stores every record contains some value (perhaps NULL) for each column, in columnar store the column is only present if there is data for one or more rows. This is why many column-oriented stores are referred to as sparse storage models. Data for each column family is physically stored together on disk sorted by rowed, column name, and timestamp. The last (timestamp) is there because the BigTable model also includes the concept of versioning. Every RowKey, Column Family, Column triple is stored with either a system-generated or user-provided Timestamp. This allows users to quickly retrieve the most recent value for a column (the default), the specific value for a column by timestamp, or all values for a column. The last is most useful because it permits very rapid temporal analysis on data in a column.

Because data for a given column is stored together, two key benefits are achieved. First, aggregation of the data in that column requires only the values for that column to be read. Conversely, in a relational system, the entire row (at least up to the column) needs to be read (which if the row is long and the column at the end it could be lots of data). Secondly, updates to a single column do not require the data for the rest of the row to be read/written. Also, because all the data in a column is uniform, data can be compressed much more efficiently. Often only a single copy of the value for a column is stored followed by the row keys where that value exists. And while deletes of an entire column is very efficient, deletes of

an entire record are extremely expensive. This is why historically column-oriented stores have been applied to online analytical processing (OLAP) style applications while relational stores were applied to online transaction processing (OLTP) requirements.

Recently, security has been a major focus of existing column implementations primarily due to the release by the National Security Agency (NSA) of its BigTable implementation to the open source community. A key advantage of the NSA implementation and other recently announced implementations is the availability of security controls at the individual cell level. With these implementations, a given user might have access to only certain cells in group based potentially on the value of those or other cells.

There are several very mature distributed column-oriented implementations available today from both open source groups and commercial foundations. These have been implemented and operational across a wide range of businesses and government organizations. Emerging are hybrid capabilities that implement relational access methods (e.g., SQL) on top of BigTable/Columnar storage models. In addition, relational implementations are adopting columnar oriented physical storage models to provide more efficient access for Big Data OLAP like aggregations and analytics.

Document Storage

Document storage approaches have been around for some time and popularized by the need to quickly search large amounts of unstructured data. Modern document stores have evolved to include extensive search and indexing capabilities for structured data and metadata and why they are often referred to as semi-structured data stores. Within a document-oriented data store, each "document" encapsulates and encodes the metadata, fields, and any other representations of that record. While somewhat analogous to a row in a relational table, one-reason document stores evolved and have gained in popularity is that most implementations do not enforce a fixed or constant schema. While best practices hold that groups of documents should be logically related and contain similar data, there is no requirement that they be alike or that any two documents even contain the same fields. That is one reason that document stores are frequently popular for data sets which have sparsely populated fields since there is far less overhead normally than traditional RDBMS systems where null value columns in records are actually stored. Groups of documents within these types of stores are generally referred to as collections, and like key-value stores, some sort of unique key references each document.

In modern implementations, documents can be built of arbitrarily nested structures and can include variable length arrays and, in some cases, executable scripts/code (which has significant security and privacy implications). Most document-store implementations also support additional indexes on other fields or properties within each document with many implementing specialized index types for sparse data, geospatial data, and text.

When modeling data into document-stores, the preferred approach is to de-normalize the data as much as possible and embed all one-to-one and most one-to-many relationships within a single document. This allows for updates to documents to be atomic operations which keep referential integrity between the documents. The most common case where references between documents should be used is when there are data elements that occur frequently across sets of documents and whose relationship to those documents is static. As an example, the publisher of a given book edition does not change, and there are far fewer publishers than there are books. It would not make sense to embed all the publisher information into each book document. Rather the book document would contain a reference to the unique key for the publisher. Since for that edition of the book, the reference will never change and so there is no danger of loss of referential integrity. Thus information about the publisher (address for example) can be updated in a single atomic operation the same as the book. Were this information embedded, it would need to be updated in every book document with that publisher.

In the Big Data realm, document stores scale horizontally through the use of partitioning or sharding to distribute portions of the collection across multiple nodes. This partitioning can be round robin-based,

ensuring an even distribution of data or content/key based so that data locality is maintained for similar data. Depending on the application required, the choice of partitioning key like with any database can have significant impacts on performance especially where aggregation functions are concerned.

There are no standard query languages for document store implementations with most using a language derived from their internal document representation (e.g., JSON, XML).

Graph Stores

While social networking sites like Facebook and LinkedIn have certainly driven the visibility of and evolution of graph stores (and processing as discussed below), graph stores have been a critical part of many problem domains from military intelligence and counter terrorism to route planning/navigation and the semantic web for years. Graph stores represent data as a series of nodes, edges, and properties on those. Analytics against graph stores include very basic shortest path and page ranking to entity disambiguation and graph matching.

Graph databases typically store two types of objects nodes and relationships as show in Figure 7 below. Nodes represents objects in the problem domain that are being analyzed be they people, places, organizations, accounts, or other objects. Relationships describe those objects in the domain relate to each other. Relationships can be non-directional/bidirectional but are typically expressed as unidirectional in order to provide more richness and expressiveness to the relationships. Hence, between two people nodes where they are father and son, there would be two relationships. One "is father of" going from the father node to the son node, and the other from the son to the father of "is son of". In addition, nodes and relationships can have properties or attributes. This is typically descriptive data about the element. For people it might be name, birthdate, or other descriptive quality. For locations it might be an address or geospatial coordinate. For a relationship like a phone call, it could be the date, time of the call, and the duration of the call. Within graphs, relationships are not always equal or have the same strength. Thus relationship often has one or more weight, cost, or confidence attributes. A strong relationship between people might have a high weight because they have known each other for years and communicate every day. A relationship where two people just met would have a low weight. The distance between nodes (be it a physical distance or a difficulty) is often expressed as a cost attribute on a relation in order to allow computation of true shortest paths across a graph. In military intelligence applications, relationships between nodes in a terrorist or command and control network might only be suspected or have not been completely verified, so those relationships would have confidence attributes. Also, properties on nodes may also have confidence factors associated with them although in those cases the property can be decomposed into its own node and tied with a relationship. Graph storage approaches can actually be viewed as a specialized implementation of a document storage scheme with two types of documents (nodes and relationships). In addition, one of the most critical elements in analyzing graph data is locating the node or edge in the graph where the analysis is to begin. To accomplish this, most graph databases implement indexes on the node or edge properties. Unlike relational and other data storage approaches, most graph databases tend to use artificial/pseudo keys or guides to uniquely identify nodes and edges. This allows attributes/properties to be easily changed due to both actual changes in the data (someone changed their name) or as more information is found out (e.g., a better location for some item or event) without needing to change the pointers two/from relationships.

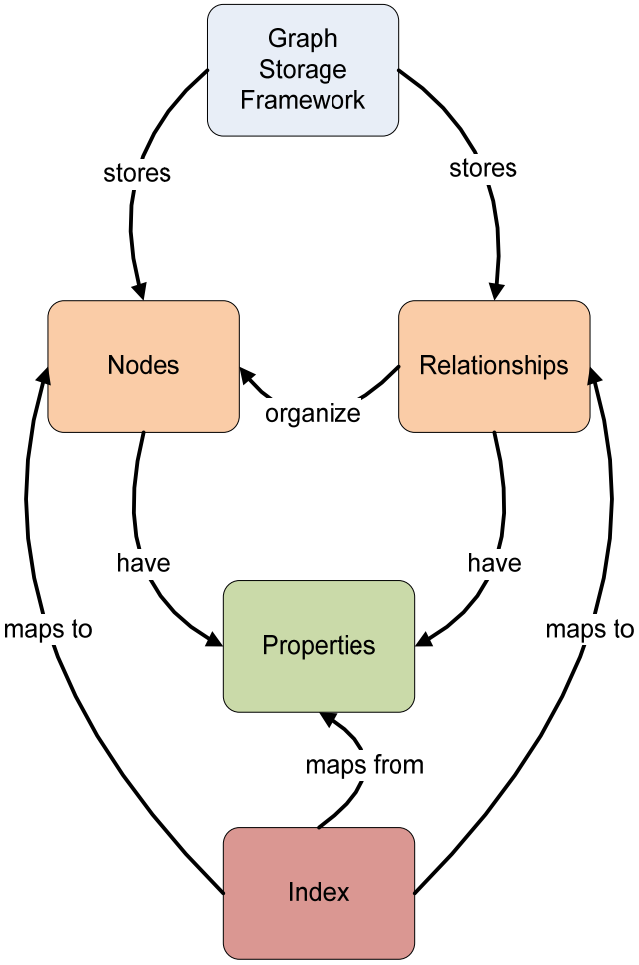


Figure B-3: Object Nodes and Relationships of Graph Databases

The problem with graphs in the Big Data realm is that they grow to be too big to fit into memory on a single node and by their typically chaotic nature (few real-world graphs follow well-defined patterns) makes their partitioning for a distributed implementation problematic. While distance between or closeness of nodes would seem like a straightforward partitioning approach, there are multiple issues which must be addressed. First would be balancing of data. Graphs often tend to have large clusters of data very dense in a given area, thus leading to essentially imbalances and hot spots in processing. Second, no matter how the graph is distributed, there are connections (edges) that will cross the boundaries. That typically requires that nodes know about or how to access the data on other nodes and requires inter-node data transfer or communication. This makes the choice of processing architectures for graph data especially critical. Architectures that do not have inter-node communication/messaging tend not to work well for most graph problems. Typically, distributed architectures for processing graphs assign chunks of the graph to nodes, then the nodes use messaging approaches to communicate changes in the graph or the value of certain calculations along a path.

Even small graphs quickly elevate into the realm of Big Data when one is looking for patterns or distances across more than one or two degrees of separation between nodes. Depending on the density of the graph, this can quickly cause a combinatorial explosion in the number of conditions/patterns that need to be tested.

A specialized implementation of a graph store known as the Resource Description Framework (RDF) is part of a family of specifications from the World Wide Web Consortium (W3C) that is often directly

associated with Semantic Web and associated concepts. RDF triples as they are known consist of a Subject (Mr. X), a predicate (lives at), and an object (Mockingbird Lane). Thus a collection of RDF triples represents and directed labeled graph. The contents of RDF stores are frequently described using formal ontology languages like OWL or the RDF Schema (RDFS) language, which establish the semantic meanings and models of the underlying data. To support better horizontal integration of heterogeneous data sets, extensions to the RDF concept such as the Data Description Framework (DDF) have been proposed which add additional types to better support semantic interoperability and analysis.[10] [11]

Graph data stores currently lack any form of standardized APIs or query languages. However, the W3C has developed the SPARQL query language for RDF, which is currently in a recommendation status, and there are several frameworks such as Sesame which are gaining popularity for working with RDF and other graph-oriented data stores.

Appendix D: Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
API	application programming interface
ASCII	American Standard Code for Information Interchange
BASE	Basically Available, Soft state, Eventual consistency
BDLM	Big Data life cycle management
BSP	Bulk Synchronous Parallel
CAP	Consistency, Availability, and Partition Tolerance
CEP	complex event processing
CIA	confidentiality, integrity, and availability
CRUD	create/read/update/delete
CSP	Cloud Service Provider
CSV	comma separated values
DDF	Data Description Framework
DLM	data life cycle management
DNS	Domain Name Server
ELT	extract, load, transform
ETL	extract, transform, load
GB	gigabyte
GRC	governance, risk management, and compliance
GUID	globally unique identifier
HPC	High Performance Computing
HTTP	HyperText Transfer Protocol
I/O	input/output
IaaS	infrastructure as a service
IT	information technology
ITL	Information Technology Laboratory
NARA	National Archives and Records Administration
NAS	network-attached storage
NASA	National Aeronautics and Space Administration
NBD-PWG	NIST Big Data Public Working Group
NBDRA	NIST Big Data Reference Architecture
NFS	network file system
NFV	network function virtualization
NGA	National Geospatial Intelligence Agency
NIST	National Institute of Standards and Technology
NoSQL	not only (or no) Structured Query Language
NRT	near real time
NSA	National Security Agency
NSF	National Science Foundation
OLAP	online analytical processing
OLTP	online transaction processing
OS	operating systems
PaaS	platform as a service
PII	personally identifiable information
POSIX	Portable Operating System Interface
RAID	redundant array of independent disks
RAM	random-access memory
RDBMS	relational database management system

RDF	Resource Description Framework
RDFS	RDF Schema
SaaS	software as a service
SAN	storage area network
SDDC	software-defined data center
SDN	software-defined network
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Appendix E: Resources and References

GENERAL RESOURCES

The following resources provide additional information related to Big Data architecture.

Big Data Public Working Group, "NIST Big Data Program," *National Institute for Standards and Technology,* June 26, 2013, http://bigdatawg.nist.gov .

Doug Laney, "3D Data Management: Controlling Data Volume, Velocity, and Variety," *Gartner,* February 6, 2001, http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

Eberhardt Rechtin, "The Art of Systems Architecting," *CRC Press*, January 6, 2009.

International Organization of Standardization (ISO), "ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description," *ISO,* November 24, 2011, http://www.iso.org/iso/catalogue_detail.htm?csnumber=50508.

Mark Beyer and Doug Laney, "The Importance of 'Big Data': A Definition," *Gartner,* June 21, 2012, http://www.gartner.com/DisplayDocument?id=2057415&ref=clientFriendlyUrl.

Martin Hilbert and Priscilla Lopez, "The World's Technological Capacity to Store, Communicate, and Compute Information," *Science*, April 1, 2011.

National Institute of Standards and Technology [NIST], "Big Data Workshop," *NIST,* June 13, 2012, http://www.nist.gov/itl/ssd/is/big-data.cfm.

National Science Foundation, "Big Data R&D Initiative," *National Institute for Standards and Technology,* June 2012, http://www.nist.gov/itl/ssd/is/upload/NIST-BD-Platforms-05-Big-Data-Wactlar-slides.pdf.

Office of the Assistant Secretary of Defense, "Reference Architecture Description," *U.S. Department of Defense,* June 2010, http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf.

Office of the White House Press Secretary, "Obama Administration Unveils "Big Data" Initiative," *White House Press Release*, March 29, 2012, http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf.

White House, "Big Data Across the Federal Government," *Executive Office of the President,* March 29, 012, http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_fact_sheet_final_1.pdf.

DOCUMENT REFERENCES

[1] The White House Office of Science and Technology Policy, "Big Data is a Big Deal," *OSTP Blog*, accessed February 21, 2014, http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal.

[2] Office of the Assistant Secretary of Defense, "Reference Architecture Description," U.S. Department of Defense, June 2010, http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf.

[3] Colella, Phillip, Defining software requirements for scientific computing. Slide of 2004 presentation included in David Patterson's 2005 talk. 2004. http://www.lanl.gov/orgs/hpc/salishan/salishan2005/davidpatterson.pdf

[4] Patterson, David; Yelick, Katherine. Dwarf Mind. A View From Berkeley. http://view.eecs.berkeley.edu/wiki/Dwarf_Mine

[5] Leslie G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, Volume 33 Issue 8 (1990).

[6] United States Census Bureau, The "72-Year Rule." https://www.census.gov/history/www/genealogy/decennial_census_records/the_72_year_rule_1.html. Accessed March 3, 2015.

[7] NIST SP 500-292, *NIST Cloud Computing Reference Architecture*. September 2011. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505

[8] The NIST Definition of Cloud Computing, Special Publication 800-145, September 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] Chang, Fay; Dean, Jeffrey; Ghemawat; Sanjay; Hsieh, Wilson C.; Wallach, Deborah A; Burrows, Mike; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E., Bigtable: A distributed storage system for structured data. Proceedings of the 7th Conference on Usenix Symposium On Operating Systems Design And Implementation, 2006. Pages 205-218.

[10] Smith, Barry; Malyuta, Tatiana; Mandirck, William S.; Fu, Chia; Parent, Kesny; Patel, Milan, Horizontal Integration of Warfighter Intelligence Data. Semantic Technology in Intelligence, Defense and Security (STIDS), 2012.

[11] Yoakum-Stover, S; Malyuta, T., Unified Data Integration for Situation Management. Military Communications, 2008.