NIST Special Publication 1500-6r2

NIST Big Data Interoperability Framework:
Volume 6, Reference Architecture

Version 3

NIST Big Data Public Working Group
Definitions and Taxonomies Subgroup

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.1500-6r2

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Special Publication 1500-6r2

NIST Big Data Interoperability Framework: Volume 6, Reference Architecture

Version 3

NIST Big Data Public Working Group
Definitions and Taxonomies Subgroup
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.1500-6r2

October 2019

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Walter Copan, NIST Director and Undersecretary of Commerce for Standards and Technology

National Institute of Standards and Technology (NIST) Special Publication 1500-6r2
75 pages (October 2019)

NIST Special Publication series 1500 is intended to capture external perspectives related to NIST standards, measurement, and testing-related efforts. These external perspectives can come from industry, academia, government, and others. These reports are intended to document external perspectives and do not represent official NIST positions.

Certain commercial entities, equipment, or materials may be identified in this document to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all publications during public comment periods and provide feedback to NIST. All NIST publications are available at http://www.nist.gov/publication-portal.cfm.

Copyrights and Permissions

Comments on this publication may be submitted to Wo Chang

National Institute of Standards and Technology
Attn: Wo Chang, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8900) Gaithersburg, MD 20899-8930
Email: SP1500comments@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at NIST promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology (IT). ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. This document reports on ITL's research, guidance, and outreach efforts in IT and its collaborative activities with industry, government, and academic organizations.

Abstract

Big Data is a term used to describe the large amount of data in the networked, digitized, sensor-laden, information-driven world. While opportunities exist with Big Data, the data can overwhelm traditional technical approaches, and the growth of data is outpacing scientific and technological advances in data analytics. To advance progress in Big Data, the NIST Big Data Public Working Group (NBD-PWG) is working to develop consensus on important fundamental concepts related to Big Data. The results are reported in the *NIST Big Data Interoperability Framework* (NBDIF) series of volumes. This volume, Volume 6, summarizes the work performed by the NBD-PWG to characterize Big Data from an architecture perspective, presents the NIST Big Data Reference Architecture (NBDRA) conceptual model, discusses the roles and fabrics of the NBDRA, presents an *activities view* of the NBDRA to describe the activities performed by the roles, and presents a *functional component view* of the NBDRA containing the classes of functional components that carry out the activities.

Keywords

Activities view; Big Data Application Provider; Big Data; Big Data characteristics; Data Consumer; Data Provider; Framework Provider; functional component view; Management Fabric; reference architecture; Security and Privacy Fabric; System Orchestrator; use cases.

Acknowledgements

This document reflects the contributions and discussions by the membership of the NBD-PWG, co-chaired by Wo Chang (NIST ITL), Bob Marcus (ET-Strategies), and Chaitan Baru (San Diego Supercomputer Center; National Science Foundation). For all versions, the Subgroups were led by the following people: Nancy Grady (SAIC), Natasha Balac (SDSC), and Eugene Luster (R2AD) for the Definitions and Taxonomies Subgroup; Geoffrey Fox (Indiana University) and Tsegereda Beyene (Cisco Systems) for the Use Cases and Requirements Subgroup; Arnab Roy (Fujitsu), Mark Underwood (Krypton Brothers; Synchrony Financial), and Akhil Manchanda (GE) for the Security and Privacy Subgroup; David Boyd (InCadence Strategic Solutions), Orit Levin (Microsoft), Don Krapohl (Augmented Intelligence), and James Ketner (AT&T) for the Reference Architecture Subgroup; and Russell Reinsch (Center for Government Interoperability), David Boyd (InCadence Strategic Solutions), Carl Buffington (Vistronix), and Dan McClary (Oracle), for the Standards Roadmap Subgroup.

The editors for this document were the following:

- ***Version 1***: Orit Levin (Microsoft), David Boyd (InCadence Strategic Solutions), and Wo Chang (NIST)
- ***Version 2***: David Boyd (InCadence Strategic Solutions) and Wo Chang (NIST)
- ***Version 3***: David Boyd (InCadence Strategic Solutions) and Wo Chang (NIST)

Laurie Aldape (Energetics Incorporated) and Elizabeth Lennon (NIST) provided editorial assistance across all NBDIF volumes.

NIST SP1500-6, Version 3 has been collaboratively authored by the NBD-PWG. As of the date of this publication, there are over six hundred NBD-PWG participants from industry, academia, and government. Federal agency participants include the National Archives and Records Administration (NARA), National Aeronautics and Space Administration (NASA), National Science Foundation (NSF), and the U.S. Departments of Agriculture, Commerce, Defense, Energy, Health and Human Services, Homeland Security, Transportation, Treasury, and Veterans Affairs.

NIST would like to acknowledge the specific contributions[1] to this volume, during Version 1, Version 2, and/or Version 3 activities, by the following NBD-PWG members:

Chaitan Baru
University of California, San Diego, Supercomputer Center

Janis Beach
Information Management Services, Inc.

David Boyd
InCadence Strategic Solutions

Scott Brim
Internet2

Gregg Brown
Microsoft

Carl Buffington
Vistronix

Pavithra Kenjige
PK Technologies

James Kobielus
IBM

Donald Krapohl
Augmented Intelligence

Orit Levin
Microsoft

Eugene Luster
DISA/R2AD

Serge Manning
Huawei USA

Robert Marcus
ET-Strategies

Felix Njeh
U.S. Department of the Army

Gururaj Pandurangi
Avyan Consulting Corp.

Linda Pelekoudas
Strategy and Design Solutions

Dave Raddatz
SiliconGraphics International Corp.

Russell Reinsch
Center for Government Interoperability

John Rogers
HP

[1] "Contributors" are members of the NIST Big Data Public Working Group who dedicated great effort to prepare and gave substantial time on a regular basis to research and development in support of this document.

Yuri Demchenko
University of Amsterdam

Jill Gemmill
Clemson University

Nancy Grady
SAIC

Ronald Hale
ISACA

Keith Hare
JCC Consulting, Inc.

Richard Jones
The Joseki Group LLC

Gary Mazzaferro
AlloyCloud, Inc.

Shawn Miller
U.S. Department of Veterans Affairs

Sanjay Mishra
Verizon

Vivek Navale
NARA

Quyen Nguyen
U.S. Census Bureau

Arnab Roy
Fujitsu

Michael Seablom
NASA

Rupinder Singh
McAfee, Inc.

Anil Srivastava
Open Health Systems Laboratory

Glenn Wasson
SAIC

Timothy Zimmerlin
Consultant

Alicia Zuniga-Alvarado
Consultant

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 (NBDIF): Volume 6, Reference Architecture* document to provide a vendor-neutral, technology- and infrastructure-agnostic conceptual model and examine related issues. The NIST Big Data Reference Architecture (NBDRA) which consists of a conceptual model and two architectural views, was a collaborative effort within the Reference Architecture 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

The *NIST Big Data Interoperability Framework* (NBDIF) was released in three versions, which correspond to the three stages of the NBD-PWG work. Version 3 (current version) of the NBDIF volumes resulted from Stage 3 work with major emphasis on the validation of the NBDRA Interfaces and content enhancement. Stage 3 work built upon the foundation created during Stage 2 and Stage 1. The current effort documented in this volume reflects concepts developed within the rapidly evolving field of Big Data. The three stages (in reverse order) aim to achieve the following with respect to the NIST Big Data Reference Architecture (NBDRA).

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

The *NBDIF* consists of nine volumes, each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The nine volumes are as follows:

- Volume 1, Definitions [1]
- Volume 2, Taxonomies [2]
- Volume 3, Use Cases and General Requirements [3]
- Volume 4, Security and Privacy [4]
- Volume 5, Architectures White Paper Survey [5]
- Volume 6, Reference Architecture (this volume)
- Volume 7, Standards Roadmap [6]
- Volume 8, Reference Architecture Interfaces [7]
- Volume 9, Adoption and Modernization [8]

During Stage 1, Volumes 1 through 7 were conceptualized, organized, and written. The finalized Version 1 documents can be downloaded from the V1.0 Final Version page of the NBD-PWG website (https://bigdatawg.nist.gov/V1_output_docs.php).

During Stage 2, the NBD-PWG developed Version 2 of the NBDIF Version 1 volumes, with the exception of Volume 5, which contained the completed architecture survey work that was used to inform Stage 1 work of the NBD-PWG. The goals of Stage 2 were to enhance the Version 1 content, define general interfaces between the NBDRA components by aggregating low-level interactions into high-level general interfaces, and demonstrate how the NBDRA can be used. As a result of the Stage 2 work, the need for NBDIF Volume 8 and NBDIF Volume 9 was identified and the two new volumes were created. Version 2 of the NBDIF volumes, resulting from Stage 2 work, can be downloaded from the V2.0 Final Version page of the NBD-PWG website (https://bigdatawg.nist.gov/V2_output_docs.php).

1 INTRODUCTION

1.1 BACKGROUND

There is broad agreement among commercial, academic, and government leaders about the 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:

- How is Big Data defined?
- What attributes define Big Data solutions?
- What is new in Big Data?
- What is the difference between Big Data and *bigger data* that has been collected for years?
- 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, secure Big Data solutions?

Within this context, on March 29, 2012, the White House announced the Big Data Research and Development Initiative [9]. 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 analysts' 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 Standards Roadmap. 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.

On June 19, 2013, the NIST Big Data Public Working Group (NBD-PWG) was launched with extensive participation by industry, academia, and government from across the nation. The scope of the NBD-PWG involves forming a community of interests from all sectors—including industry, academia, and government—with the goal of developing consensus on definitions, taxonomies, secure reference architectures, security and privacy, and, from these, a standards roadmap. Such a consensus would create a vendor-neutral, technology- and infrastructure-independent framework that would enable Big Data stakeholders to identify and use the best analytics tools for their processing and visualization requirements on the most suitable computing platform and cluster, while also allowing added value from Big Data service providers.

The *NIST Big Data Interoperability Framework* (NBDIF) was released in three versions, which correspond to the three stages of the NBD-PWG work. Version 3 (current version) of the NBDIF volumes resulted from Stage 3 work with major emphasis on the validation of the NBDRA Interfaces and content enhancement. Stage 3 work built upon the foundation created during Stage 2 and Stage 1. The current effort documented in this volume reflects concepts developed within the rapidly evolving field of Big Data. The three stages (in reverse order) aim to achieve the following with respect to the NIST Big Data Reference Architecture (NBDRA).

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

The *NBDIF* consists of nine volumes, each of which addresses a specific key topic, resulting from the work of the NBD-PWG. The nine volumes are as follows:

- Volume 1, Definitions [1]
- Volume 2, Taxonomies [2]
- Volume 3, Use Cases and General Requirements [3]
- Volume 4, Security and Privacy [4]
- Volume 5, Architectures White Paper Survey [5]
- Volume 6, Reference Architecture (this volume)
- Volume 7, Standards Roadmap [6]
- Volume 8, Reference Architecture Interfaces [7]
- Volume 9, Adoption and Modernization [8]

During Stage 1, Volumes 1 through 7 were conceptualized, organized, and written. The finalized Version 1 documents can be downloaded from the V1.0 Final Version page of the NBD-PWG website (https://bigdatawg.nist.gov/V1_output_docs.php).

During Stage 2, the NBD-PWG developed Version 2 of the NBDIF Version 1 volumes, with the exception of Volume 5, which contained the completed architecture survey work that was used to inform Stage 1 work of the NBD-PWG. The goals of Stage 2 were to enhance the Version 1 content, define general interfaces between the NBDRA components by aggregating low-level interactions into high-level general interfaces, and demonstrate how the NBDRA can be used. As a result of the Stage 2 work, the need for NBDIF Volume 8 and NBDIF Volume 9 was identified and the two new volumes were created. Version 2 of the NBDIF volumes, resulting from Stage 2 work, can be downloaded from the V2.0 Final Version page of the NBD-PWG website (https://bigdatawg.nist.gov/V2_output_docs.php).

1.2 SCOPE AND OBJECTIVES OF THE REFERENCE ARCHITECTURES SUBGROUP

Reference architectures provide "an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions [10]." Reference architectures generally serve as a foundation for solution architectures and may also be used for comparison and alignment of instantiations of architectures and solutions.

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.

The NBDRA is a high-level conceptual model crafted to serve as a tool to facilitate open discussion of the requirements, design structures, and operations inherent in Big Data. The NBDRA is intended to facilitate the understanding of the operational intricacies in Big Data. It does not represent the system architecture of a specific Big Data system, but rather is a tool for describing, discussing, and developing system-specific architectures using a common framework of reference. The model is not tied to any specific vendor products, services, or reference implementation, nor does it define prescriptive solutions that inhibit innovation.

The NBDRA does not address the following:

- Detailed specifications for any organization's operational systems;
- Detailed specifications of information exchanges or services; and
- Recommendations or standards for integration of infrastructure products.

1.3 REPORT PRODUCTION

A wide spectrum of Big Data architectures has been explored and developed as part of various industry, academic, and government initiatives. The development of the NBDRA and material contained in this volume involved the following steps:

1. Announce that the NBD-PWG Reference Architecture Subgroup is open to the public to attract and solicit a wide array of subject matter experts and stakeholders in government, industry, and academia;
2. Gather publicly available Big Data architectures and materials representing various stakeholders, different data types, and diverse use cases;[2]
3. Examine and analyze the Big Data material to better understand existing concepts, usage, goals, objectives, characteristics, and key elements of Big Data, and then document the

[2] Many of the architecture use cases were originally collected by the NBD-PWG Use Case and Requirements Subgroup and can be accessed at http://bigdatawg.nist.gov/usecases.php.

findings using NIST's Big Data taxonomies model (presented in *NBDIF: Volume 2, Taxonomies*);

4. Develop a technology-independent, open reference architecture based on the analysis of Big Data material and inputs received from other NBD-PWG subgroups;
5. Identify workflow and interactions from the System Operator to the rest of the NBDRA components; and
6. Develop an Activities View and a Functional Component View of the NBDRA to describe the activities performed by the roles and fabrics along with the functional components that carry out the activities.

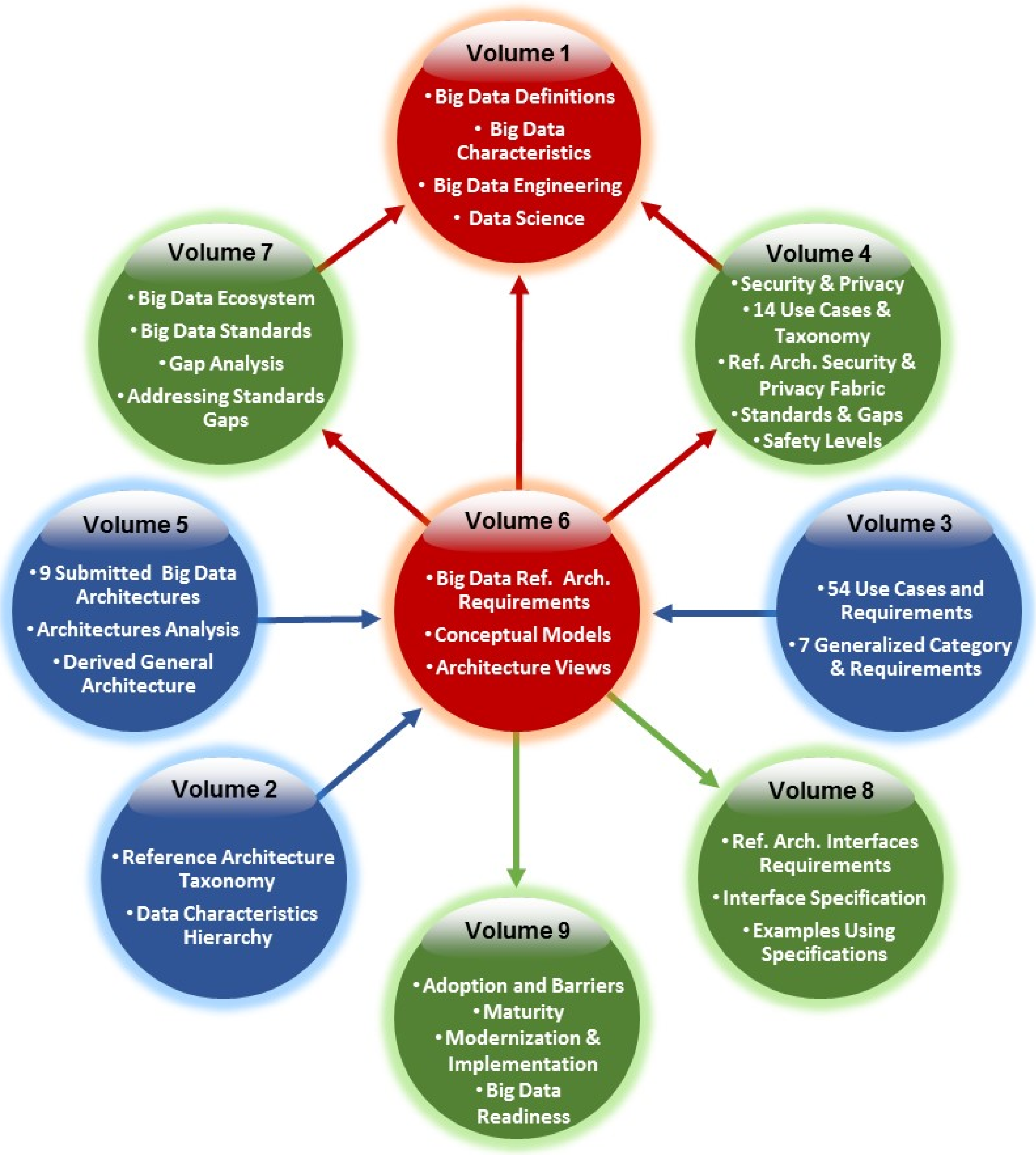
To achieve technical and high-quality document content, this document will go through a public comment period along with NIST internal review.

1.4 REPORT STRUCTURE

The organization of this document roughly corresponds to the process used by the NBD-PWG to develop the NBDRA. Following the introductory material presented in Section 1, the remainder of this document is organized as follows:

- Section 2 summarizes the work of other NBD-PWG Subgroups that informed the formation of the NBDRA.
- Section 3 presents the NBDRA conceptual model, which is a vendor- and technology-agnostic Big Data conceptual model.
- Section 4 explores two different views of the NBDRA, the activities view, which examines the activities carried out by the NBDRA roles, and the functional component view, which examines the functional components that carry out the activities
- Section 5 summarizes conclusions of this volume.

While each NBDIF volume was created with a specific focus within Big Data, all volumes are interconnected. During the creation of the volumes, information from some volumes was used as input for other volumes. Broad topics (e.g., definition, architecture) may be discussed in several volumes with each discussion circumscribed by the volume's particular focus. Arrows shown in Figure 1 indicate the main flow of information input and/or output from the volumes. Volumes 2, 3, and 5 (blue circles) are essentially standalone documents that provide output to other volumes (e.g., to Volume 6). These volumes contain the initial situational awareness research. During the creation of Volumes 4, 7, 8, and 9 (green circles), input from other volumes was used. The development of these volumes took into account work on the other volumes. Volumes 1 and 6 (red circles) were developed using the initial situational awareness research and continued to be modified based on work in other volumes. The information from these volumes was also used as input to the volumes in the green circles.


Volume 1
• Big Data Definitions
• Big Data Characteristics
• Big Data Engineering
• Data Science
Volume 7
• Big Data Ecosystem
• Big Data Standards
• Gap Analysis
• Addressing Standards Gaps
Volume 4
• Security & Privacy
• 14 Use Cases & Taxonomy
• Ref. Arch. Security & Privacy Fabric
• Standards & Gaps
• Safety Levels
Volume 5
• 9 Submitted Big Data Architectures
• Architectures Analysis
• Derived General Architecture
Volume 6
• Big Data Ref. Arch. Requirements
• Conceptual Models
• Architecture Views
Volume 3
• 54 Use Cases and Requirements
• 7 Generalized Category & Requirements
Volume 2
• Reference Architecture Taxonomy
• Data Characteristics Hierarchy
Volume 8
• Ref. Arch. Interfaces Requirements
• Interface Specification
• Examples Using Specifications
Volume 9
• Adoption and Barriers
• Maturity
• Modernization & Implementation
• Big Data Readiness

215

216 *Figure 1: NBDIF Documents Navigation Diagram Provides Content Flow Between Volumes*

217

2 HIGH-LEVEL REFERENCE ARCHITECTURE REQUIREMENTS

The development of a Big Data reference architecture requires a thorough understanding of current techniques, issues, and concerns. To this end, the NBD-PWG collected use cases to gain an understanding of current applications of Big Data, conducted a survey of reference architectures to understand commonalities within Big Data architectures in use, developed a taxonomy to understand and organize the information collected, and reviewed existing technologies and trends relevant to Big Data. The results of these NBD-PWG activities were used in the development of the NBDRA and are briefly summarized in this section extracted from the corresponding other parts of the NBDIF.

2.1 USE CASES AND REQUIREMENTS

To develop the use cases, publicly available information was collected for various Big Data architectures in nine broad areas, or application domains. Participants in the NBD-PWG Use Case and Requirements Subgroup and other interested parties provided the use case details via a template, which helped to standardize the responses and facilitate subsequent analysis and comparison of the use cases. However, submissions still varied in levels of detail, quantitative data, or qualitative information. The *NBDIF: Volume 3, Use Cases and General Requirements* document presents the original use cases, an analysis of the compiled information, and the requirements extracted from the use cases.

The extracted requirements represent challenges faced in seven characterization categories (Table 1) developed by the Subgroup. Requirements specific to the use cases were aggregated into high-level generalized requirements, which are vendor and technology neutral.

The use case characterization categories were used as input in the development of the NBDRA and map directly to NBDRA components and fabrics as shown in Table 1.

Table 1: Mapping Use Case Characterization Categories to Reference Architecture Components and Fabrics

USE CASE CHARACTERIZATION CATEGORIES		REFERENCE ARCHITECTURE COMPONENTS AND FABRICS
Data sources	→	Data Provider
Data transformation	→	Big Data Application Provider
Capabilities	→	Big Data Framework Provider
Data consumer	→	Data Consumer
Security and privacy	→	Security and Privacy Fabric
Life cycle management	→	System Orchestrator; Management Fabric
Other requirements	→	To all components and fabrics

The high-level generalized requirements are presented below. The development of these generalized requirements is presented in the *NBDIF: Volume 3, Use Cases and Requirements* document.

Data Source Requirements (DSR)

- DSR-1: Reliable, real-time, asynchronous, streaming, and batch processing to collect data from centralized, distributed, and cloud data sources, sensors, or instruments
- DSR-2: Slow, bursty, and high throughput data transmission between data sources and computing clusters
- DSR-3: Diversified data content ranging from structured and unstructured text, documents, graphs, websites, geospatial, compressed, timed, spatial, multimedia, simulation, and instrumental (i.e., system managements and monitoring) data

Transformation Provider Requirements (TPR)

- TPR-1: Diversified, compute-intensive, statistical and graph analytic processing and machine-learning techniques
- TPR-2: Batch and real-time analytic processing
- TPR-3: Processing large diversified data content and modeling
- TPR-4: Processing data in motion (e.g., streaming, fetching new content, data tracking, traceability, data change management, and data boundaries)

Capability Provider Requirements (CPR)

- CPR-1: Legacy software and advanced software packages
- CPR-2: Legacy and advanced computing platforms
- CPR-3: Legacy and advanced distributed computing clusters, co-processors, input/output (I/O) processing
- CPR-4: Advanced networks (e.g., software-defined network [SDN]) and elastic data transmission, including fiber, cable, and wireless networks (e.g., local area network, wide area network, metropolitan area network, Wi-Fi)
- CPR-5: Legacy, large, virtual, and advanced distributed data storage
- CPR-6: Legacy and advanced programming executables, applications, tools, utilities, and libraries

Data Consumer Requirements (DCR)

- DCR-1: Fast searches from processed data with high relevancy, accuracy, and recall
- DCR-2: Diversified output file formats for visualization, rendering, and reporting
- DCR-3: Visual layout for results presentation
- DCR-4: Rich user interface for access using browser, visualization tools
- DCR-5: High-resolution, multidimensional layer of data visualization
- DCR-6: Streaming results to clients

Security and Privacy Requirements (SPR)

- SPR-1: Protect and preserve security and privacy of sensitive data.
- SPR-2: Support sandbox, access control, and multi-tenant, multilevel, policy-driven authentication on protected data and ensure that these are in line with accepted governance, risk, and compliance (GRC) and confidentiality, integrity, and availability (CIA) best practices.

LIFE CYCLE MANAGEMENT REQUIREMENTS (LMR)

- LMR-1: Data quality curation, including preprocessing, data clustering, classification, reduction, and format transformation
- LMR-2: Dynamic updates on data, user profiles, and links
- LMR-3: Data life cycle and long-term preservation policy, including data provenance
- LMR-4: Data validation
- LMR-5: Human annotation for data validation
- LMR-6: Prevention of data loss or corruption
- LMR-7: Multisite (including cross-border, geographically dispersed) archives
- LMR-8: Persistent identifier and data traceability
- LMR-9: Standardization, aggregation, and normalization of data from disparate sources

OTHER REQUIREMENTS (OR)

- OR-1: Rich user interface from mobile platforms to access processed results
- OR-2: Performance monitoring on analytic processing from mobile platforms
- OR-3: Rich visual content search and rendering from mobile platforms
- OR-4: Mobile device data acquisition and management
- OR-5: Security across mobile devices and other smart devices such as sensors

2.2 REFERENCE ARCHITECTURE SURVEY

The NBD-PWG Reference Architecture Subgroup conducted a survey of current reference architectures to advance the understanding of the operational intricacies in Big Data and to serve as a tool for developing system-specific architectures using a common reference framework. The Subgroup surveyed currently published Big Data platforms by leading companies or individuals supporting the Big Data framework and analyzed the collected material.

This effort revealed a consistency between Big Data architectures that served in the development of the NBDRA. Survey details, methodology, and conclusions are reported in *NBDIF: Volume 5, Architectures White Paper Survey*.

2.3 TAXONOMY

The NBD-PWG Definitions and Taxonomy Subgroup focused on identifying Big Data concepts, defining terms needed to describe the new Big Data paradigm, and defining reference architecture terms. The reference architecture taxonomy presented below provides a hierarchy of the components of the reference architecture. Additional taxonomy details are presented in the *NBDIF: Volume 2, Taxonomy* document.

Figure 2 outlines potential actors for the seven roles developed by the NBD-PWG Definition and Taxonomy Subgroup. The blue boxes contain the name of the role at the top with potential actors listed directly below.

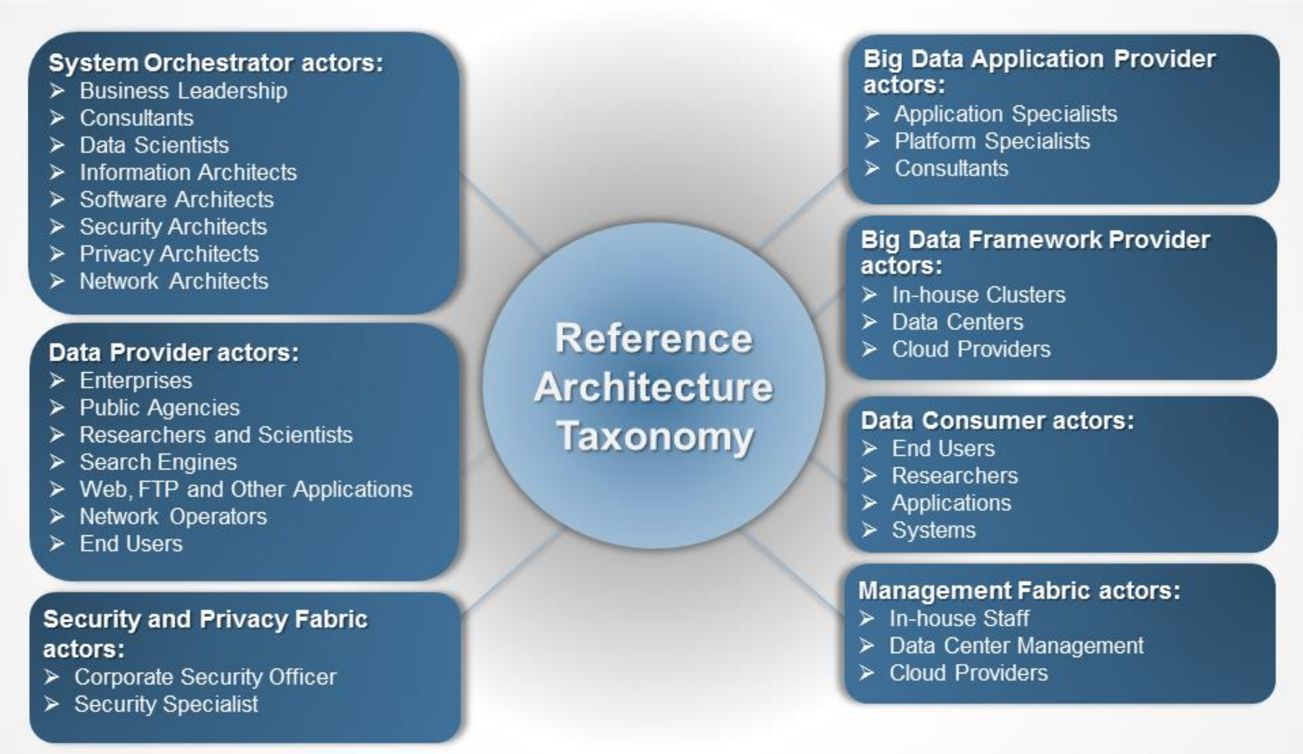


Figure 2: NBDRA Taxonomy

SYSTEM ORCHESTRATOR

The System Orchestrator provides the overarching requirements that the system must fulfill, including policy, governance, architecture, resources, and business requirements, as well as monitoring or auditing activities to ensure that the system complies with those requirements. The System Orchestrator role provides system requirements, high-level design, and monitoring for the data system. While the role predates Big Data systems, some related design activities have changed within the Big Data paradigm.

DATA PROVIDER

A Data Provider makes data available to itself or to others. In fulfilling its role, the Data Provider creates an abstraction of various types of data sources (such as raw data or data previously transformed by another system) and makes them available through different functional interfaces. The actor fulfilling this role can be part of the Big Data system, internal to the organization in another system, or external to the organization orchestrating the system. While the concept of a Data Provider is not new, the greater data collection and analytics capabilities have opened up new possibilities for providing valuable data.

BIG DATA APPLICATION PROVIDER

The Big Data Application Provider executes the manipulations of the data life cycle to meet requirements established by the System Orchestrator. This is where the general capabilities within the Big Data framework are combined to produce the specific data system. While the activities of an application provider are the same whether the solution being built concerns Big Data or not, the methods and techniques have changed because the data and data processing is parallelized across resources.

BIG DATA FRAMEWORK PROVIDER

The Big Data Framework Provider has general resources or services to be used by the Big Data Application Provider in the creation of the specific application. There are many new components from which the Big Data Application Provider can choose in using these resources and the network to build the specific system. This is the role that has seen the most significant changes because of Big Data.

The Big Data Framework Provider consists of one or more instances of the three subcomponents: infrastructure frameworks, data platforms, and processing frameworks. There is no requirement that all instances at a given level in the hierarchy be of the same technology and, in fact, most Big Data implementations are hybrids combining multiple technology approaches. These provide flexibility and can meet the complete range of requirements that are driven from the Big Data Application Provider. Due to the rapid emergence of new techniques, this is an area that will continue to need discussion.

DATA CONSUMER

The Data Consumer receives the value output of the Big Data system. In many respects, it is the recipient of the same type of functional interfaces that the Data Provider exposes to the Big Data Application Provider. After the system adds value to the original data sources, the Big Data Application Provider then exposes that same type of functional interfaces to the Data Consumer.

SECURITY AND PRIVACY FABRIC

Security and privacy issues affect all other components of the NBDRA. The Security and Privacy Fabric interacts with the System Orchestrator for policy, requirements, and auditing and also with both the Big Data Application Provider and the Big Data Framework Provider for development, deployment, and operation. The *NBDIF: Volume 4, Security and Privacy* document discusses security and privacy topics.

MANAGEMENT FABRIC

The Big Data characteristics of volume, velocity, variety, and variability demand a versatile system and software management platform for provisioning, software and package configuration and management, along with resource and performance monitoring and management. Big Data management involves system, data, security, and privacy considerations at scale, while maintaining a high level of data quality and secure accessibility.

3 NBDRA CONCEPTUAL MODEL

As discussed in Section 2, the NBD-PWG Reference Architecture Subgroup used a variety of inputs from other NBD-PWG subgroups in developing a vendor-neutral, technology- and infrastructure-agnostic conceptual model of Big Data architecture. This conceptual model, the NBDRA, is shown in Figure 3 and represents a Big Data system comprised of five logical functional components connected by interoperability interfaces (i.e., services). Two fabrics envelop the components, representing the interwoven nature of management and security and privacy with all five of the components.

The NBDRA is intended to enable system engineers, data scientists, software developers, data architects, and senior decision makers to develop solutions to issues that require diverse approaches due to convergence of Big Data characteristics within an interoperable Big Data ecosystem. It provides a framework to support a variety of business environments, including tightly integrated enterprise systems and loosely coupled vertical industries, by enhancing understanding of how Big Data complements and differs from existing analytics, business intelligence, databases, and systems.

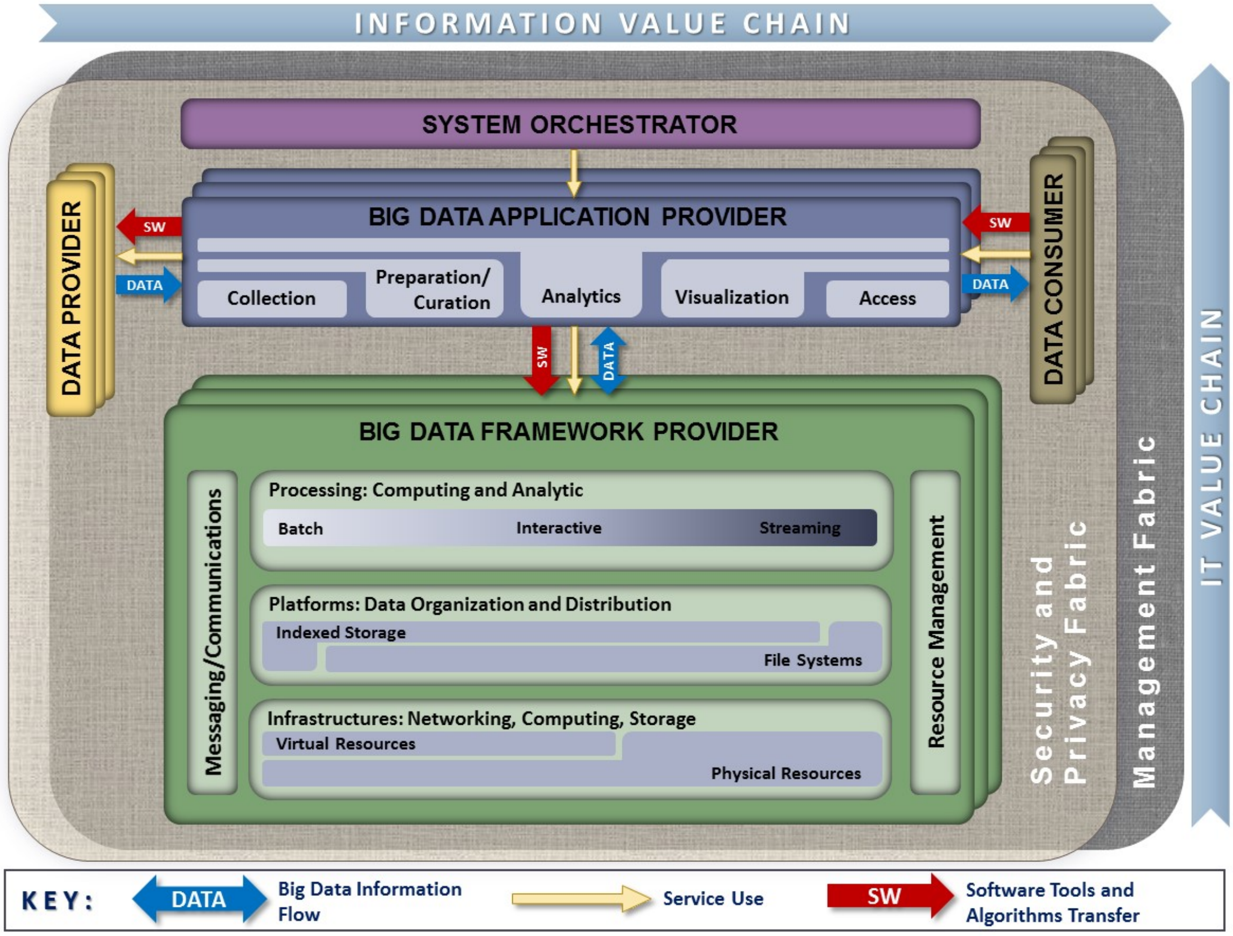


Figure 3: NIST Big Data Reference Architecture (NBDRA)

Note: None of the terminology or diagrams in these documents is intended to imply any business or deployment model. The terms *provider* and *consumer* as used are descriptive of general roles and are meant to be informative in nature.

The NBDRA is organized around five major roles and multiple sub-roles aligned along two axes representing the two Big Data value chains: Information Value (horizontal axis) and Information Technology (IT; vertical axis). Along the Information Value axis, the value is created by data collection, integration, analysis, and applying the results following the value chain. Along the IT axis, the value is created by providing networking, infrastructure, platforms, application tools, and other IT services for hosting of and operating the Big Data in support of required data applications. At the intersection of both axes is the Big Data Application Provider role, indicating that data analytics and its implementation provide the value to Big Data stakeholders in both value chains. The term *provider* as part of the Big Data Application Provider and Big Data Framework Provider is there to indicate that those roles provide or implement specific activities and functions within the system. It does not designate a service model or business entity.

The five main NBDRA roles, shown in Figure 3 and discussed in detail in Section 3, represent different technical roles that exist in every Big Data system. These roles are the following:

- System Orchestrator,
- Data Provider,
- Big Data Application Provider,
- Big Data Framework Provider, and
- Data Consumer.

The two fabric roles shown in Figure 3 encompassing the five main roles are:

- Management, and
- Security and Privacy.

These two fabrics provide services and functionality to the five main roles in the areas specific to Big Data and are crucial to any Big Data solution.

The **DATA** arrows in Figure 3 show the flow of data between the system's main roles. Data flows between the roles either physically (i.e., by value) or by providing its location and the means to access it (i.e., by reference). The **SW** arrows show transfer of software tools for processing of Big Data *in situ*. The **Service Use** arrows represent software programmable interfaces. While the main focus of the NBDRA is to represent the run-time environment, all three types of communications or transactions can happen in the configuration phase as well. Manual agreements (e.g., service-level agreements) and human interactions that may exist throughout the system are not shown in the NBDRA.

Within a given Big Data Architecture implementation, there may be multiple instances of elements performing the Data Provider, Data Consumer, Big Data Framework Provider, and Big Data Application Provider roles. Thus, in a given Big Data implementation, there may be multiple Big Data applications which use different frameworks to meet requirements. For example, one application may focus on ingestion and analytics of streaming data and would use a framework based on components suitable for that purpose, while another application may perform data warehouse style batch analytics which would leverage a different framework. Figure 4 below shows how such multiple instances may interact as part of a larger integrated system. As illustrated in the conceptual model, there should be a common Security and Privacy, and Management roles across the architecture. The crosscutting roles are sometimes referred to as fabrics because they must touch all the other roles and sub-roles within the Architecture.

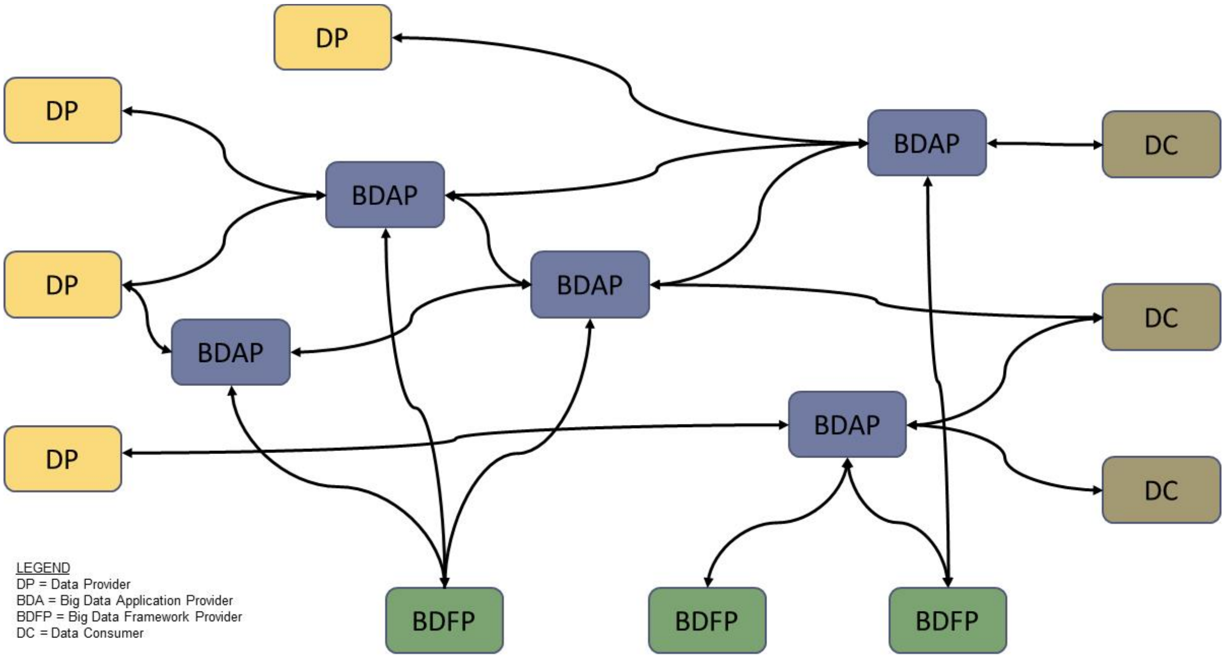


Figure 4: Multiple Instances of NBDRA Components Interact as Part of a Larger System

The roles in the Big Data ecosystem perform activities and are implemented via functional components. In system development, actors and roles have the same relationship as in the movies, but system development actors can represent individuals, organizations, software, or hardware. According to the Big Data taxonomy, a single actor can play multiple roles, and multiple actors can play the same role. The NBDRA does not specify the business boundaries between the participating actors or stakeholders, so the roles can either reside within the same business entity or can be implemented by different business entities. Therefore, the NBDRA is applicable to a variety of business environments, from tightly integrated enterprise systems to loosely coupled vertical industries that rely on the cooperation of independent stakeholders. As a result, the notion of internal versus external functional components or roles does not apply to the NBDRA. However, for a specific use case, once the roles are associated with specific business stakeholders, the functional components and the activities they perform would be considered as internal or external—subject to the use case's point of view.

The NBDRA does support the representation of stacking or chaining of Big Data systems. For example, a Data Consumer of one system could serve as a Data Provider to the next system down the stack or chain. Figure 5 below shows how a given Big Data Architecture implementation would operate in context with other systems, users, or Big Data implementations.

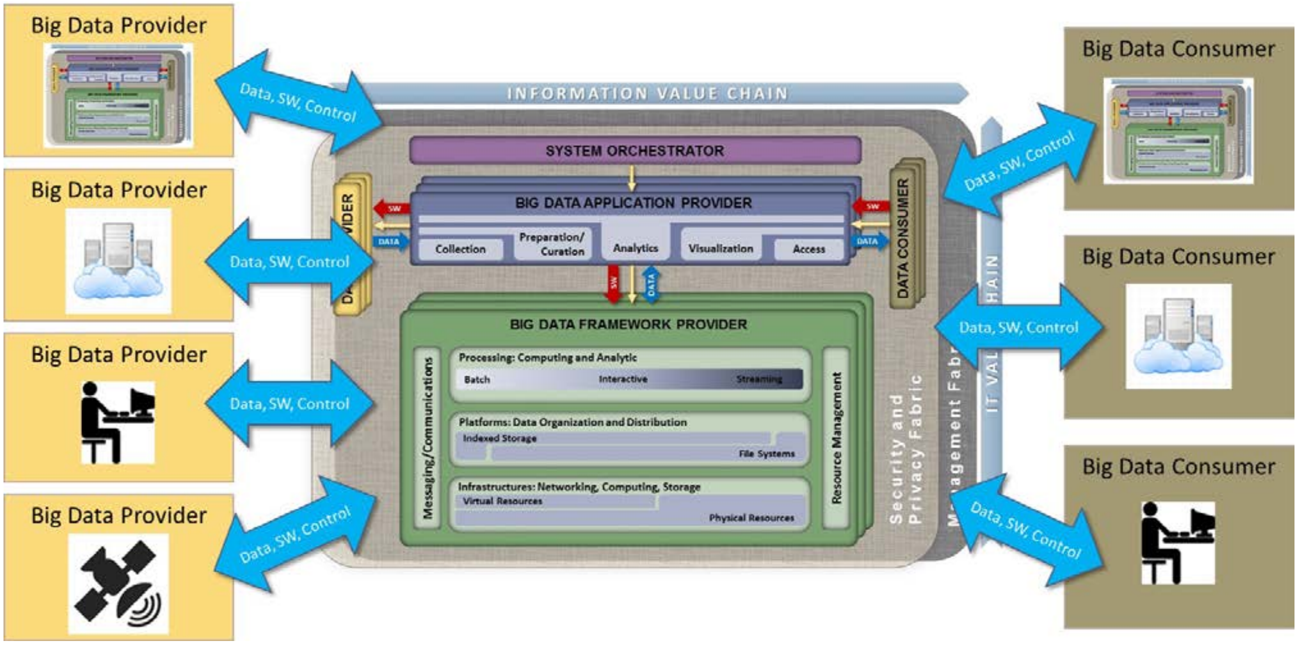


Figure 5: Big Data System within a System of Systems View

The following paragraphs provide high-level descriptions of the primary roles within the NBDRA. Section 4 contains more detailed descriptions of the sub-roles, activities, and functional components.

3.1 SYSTEM ORCHESTRATOR

The System Orchestrator role includes defining and integrating the required data application activities into an operational vertical system. Typically, the System Orchestrator involves a collection of more specific roles, performed by one or more actors, which manage and orchestrate the operation of the Big Data system. These actors may be human components, software components, or some combination of the two.

The function of the System Orchestrator is to configure and manage the other components of the Big Data architecture to implement one or more workloads that the architecture is designed to execute. The workloads managed by the System Orchestrator may be assigning/provisioning framework components to individual physical or virtual nodes at the lower level or providing a graphical user interface that supports the specification of workflows linking together multiple applications and components at the higher level.

The System Orchestrator may also, through the Management Fabric, monitor the workloads and system to confirm that specific quality of service requirements are met for each workload, and may actually elastically assign and provision additional physical or virtual resources to meet workload requirements resulting from changes/surges in the data or number of users/transactions.

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 bounded by agreements and standard or standard-de-facto interfaces.

In an enterprise environment, the System Orchestrator role is typically centralized and can be mapped to the traditional role of system governor that provides the overarching requirements and constraints, which the system must fulfill, including policy, architecture, resources, or business requirements. A system governor works with a collection of other roles (e.g., data manager, data security, and system manager) to implement the requirements and the system’s functionality.

In a loosely coupled vertical system, the System Orchestrator role is typically decentralized. Each independent stakeholder is responsible for its own system management, security, and integration, as well as integration within the Big Data distributed system using the interfaces provided by other stakeholders.

3.2 DATA PROVIDER

The Data Provider role introduces new data or information feeds into the Big Data system for discovery, access, and transformation by the Big Data system. New data feeds are distinct from the data already in use by the system and residing in the various system repositories. Similar technologies can be used to access both new data feeds and existing data. The Data Provider actors can be anything from a sensor, to a human inputting data manually, to another Big Data system.

One of the important characteristics of a Big Data system is the ability to import and use data from a variety of data sources. Data sources can be internal or public records, tapes, images, audio, videos, sensor data, web logs, system and audit logs, HyperText Transfer Protocol (HTTP) cookies, and other sources. Humans, machines, sensors, online and offline applications, Internet technologies, and other actors can also produce data sources.

The roles of Data Provider and Big Data Application Provider often belong to different organizations, unless the organization implementing the Big Data Application Provider owns the data sources. Consequently, data from different sources may have different security and privacy considerations. In fulfilling its role, the Data Provider creates an abstraction of the data sources. In the case of raw data sources, the Data Provider can potentially clean, correct, and store the data in an internal format that is accessible to the Big Data system that will ingest it.

The Data Provider can also provide an abstraction of data previously transformed by another system (i.e., legacy system, another Big Data system). In this case, the Data Provider would represent a Data Consumer of the other system. For example, Data Provider 1 could generate a streaming data source from the operations performed by Data Provider 2 on a dataset at rest.

Data Provider activities include the following, which are common to most systems that handle data:

- Collecting the data;
- Persisting the data;
- Providing transformation functions for data scrubbing of sensitive information such as personally identifiable information (PII);
- Creating the metadata describing the data source(s), usage policies/access rights, and other relevant attributes;
- Enforcing access rights on data access;
- Establishing formal or informal contracts for data access authorizations;
- Making the data accessible through suitable programmable push or pull interfaces;
- Providing push or pull access mechanisms; and
- Publishing the availability of the information and the means to access it.

The Data Provider exposes a collection of interfaces (or services) for discovering and accessing the data. These interfaces would typically include a registry so that applications can locate a Data Provider, identify the data of interest it contains, understand the types of access allowed, understand the types of analysis supported, locate the data source, determine data access methods, identify the data security requirements, identify the data privacy requirements, and other pertinent information. Therefore, the interface would provide the means to register the data source, query the registry, and identify a standard set of data contained by the registry.

Subject to Big Data characteristics (i.e., volume, variety, velocity, and variability) and system design considerations, interfaces for exposing and accessing data would vary in their complexity and can include both push and pull software mechanisms. These mechanisms can include subscription to events, listening to data feeds, querying for specific data properties or content, and the ability to submit a code for execution to process the data *in situ*. Because the data can be too large to economically move across the network, the interface could also allow the submission of analysis requests (e.g., software code implementing a certain algorithm for execution), with the results returned to the requestor. Data access may not always be automated, but might involve a human role logging into the system and providing directions where new data should be transferred (e.g., establishing a subscription to an email-based data feed).

The interface between the Data Provider and Big Data Application Provider typically will go through three phases: initiation, data transfer, and termination. The initiation phase is started by either party and often includes some level of authentication/authorization. The phase may also include queries for metadata about the source or consumer, such as the list of available topics in a publish/subscribe (pub/sub) model and the transfer of any parameters (e.g., object count/size limits or target storage locations). Alternatively, the phase may be as simple as one side opening a socket connection to a known port on the other side.

The data transfer phase may be a push from the Data Provider or a pull by the Big Data Application Provider. It may also be a singular transfer or involve multiple repeating transfers. In a repeating transfer situation, the data may be a continuous stream of transactions/records/bytes. In a push scenario, the Big Data Application Provider must be prepared to accept the data asynchronously but may also be required to acknowledge (or negatively acknowledge) the receipt of each unit of data. In a pull scenario, the Big Data Application Provider would specifically generate a request that defines through parameters of the data to be returned. The returned data could itself be a stream or multiple records/units of data, and the data transfer phase may consist of multiple request/send transactions.

The termination phase could be as simple as one side simply dropping the connection or could include checksums, counts, hashes, or other information about the completed transfer.

3.3 BIG DATA APPLICATION PROVIDER

The Big Data Application Provider role executes a specific set of operations along the data life cycle to meet the requirements established by the System Orchestrator, as well as meeting security and privacy requirements. The Big Data Application Provider is the architecture component that encapsulates the business logic and functionality to be executed by the architecture. The Big Data Application Provider activities include the following:

- Collection,
- Preparation,
- Analytics,
- Visualization, and
- Access.

These activities are represented by the subcomponents of the Big Data Application Provider as shown in Figure 3. The execution of these activities would typically be specific to the application and, therefore, are not candidates for standardization. However, the metadata and the policies defined and exchanged between the application's subcomponents could be standardized when the application is specific to a vertical industry.

While many of these activities exist in traditional data processing systems, the data volume, velocity, variety, and variability present in Big Data systems radically change their implementation. Traditional algorithms and mechanisms of traditional data processing implementations need to be adjusted and optimized to create applications that are responsive and can grow to handle ever-growing data collections.

As data propagates through the ecosystem, it is being processed and transformed in different ways in order to extract the value from the information. Each activity of the Big Data Application Provider can be implemented by independent stakeholders and deployed as stand-alone services.

The Big Data Application Provider can be a single instance or a collection of more granular Big Data Application Providers, each implementing different steps in the data life cycle. Each of the activities of the Big Data Application Provider may be a general service invoked by the System Orchestrator, Data Provider, or Data Consumer, such as a web server, a file server, a collection of one or more application programs, or a combination. There may be multiple and differing instances of each activity or a single program may perform multiple activities. Each of the activities is able to interact with the underlying Big Data Framework Providers as well as with the Data Providers and Data Consumers. In addition, these 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 blocks behind the Big Data Framework Providers in Figure 3 indicate that multiple Big Data Framework Providers can support a single Big Data Application Provider.

3.4 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 3). 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 sub-roles (from the bottom to the top):

- Infrastructure Frameworks,
- Data Platform Frameworks, and
- Processing Frameworks.

3.5 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, and
- 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.

3.6 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.

3.7 SECURITY AND PRIVACY FABRIC OF THE NBDRA

Security and privacy considerations form a fundamental aspect of the NBDRA. This is geometrically depicted in Figure 3 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.

4 NBDRA ARCHITECTURE VIEWS

As outlined in Section 3, the five main roles and two fabrics of the NBDRA represent the different categories of technical activities and functional components within a Big Data system. In order to apply the NBDRA to a particular system, it is necessary to construct architecture views of these activities and the functional components that implement them. In constructing these views, the following definitions apply:

> ***Role:*** *A related set of functions performed by one or more actors.*
>
> ***Sub-Role****: A closely related sub-set of functions within a larger role.*
>
> ***Activity:*** *A class of functions performed to fulfill the needs of one or more roles. Example: Data Collection is a class of activities through which a Big Data Application Provider obtains data. Instances of such would be web crawling, File Transfer Protocol (FTP) site, web services, database queries, etc.*
>
> ***Functional Component:*** *A class of physical items which support one or more activities within a role. Example: Stream Processing Frameworks are a class of computing frameworks which implement processing of streaming data. Instances of such frameworks would include SPARK and STORM.*

In order to promote consistency and the ability to easily compare and contrast the views of different architecture implementations, the NBDRA is proposing the conventions shown in Figure 6 for the activities and functional component views.

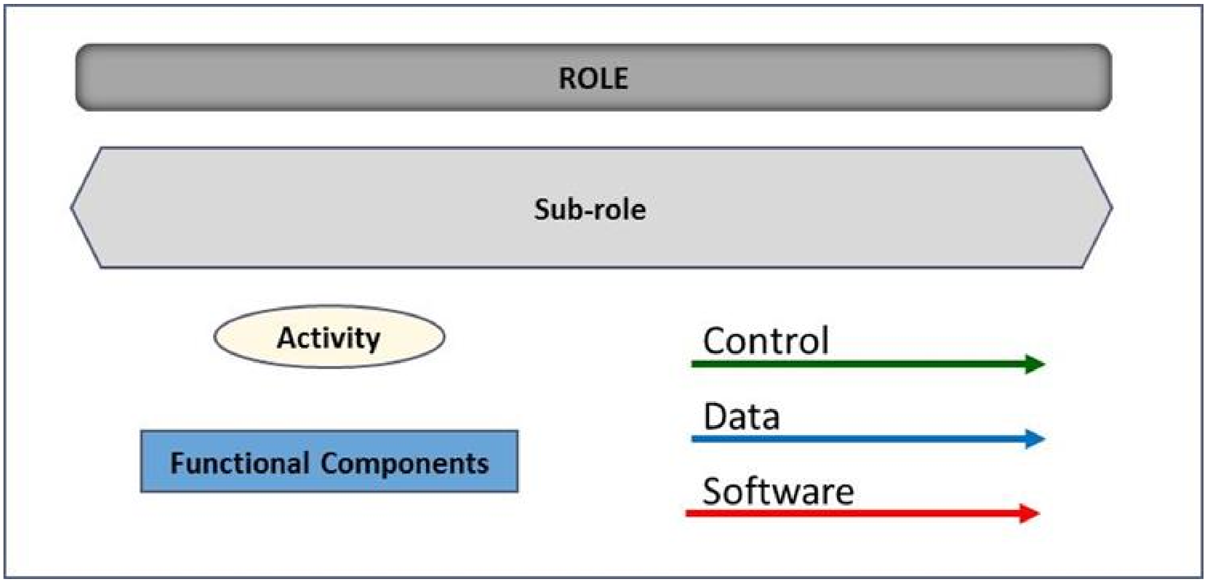


Figure 6: NBDRA View Conventions

The process of applying the NBDRA to a specific architecture implementation involves creating two views of the architecture. The first view is the Activities View where one would enumerate the activities to be accomplished by each role and sub-role within the system. Since there could be multiple instances of different roles within a given system architecture, it would be appropriate to construct separate architecture views for each instance since the role would likely be performing different activities though different functional components.

Figure 7 below provides a broad skeleton for construction of the activity views in terms of the roles and fabrics which anchor each view into a common framework. Depending on the specifics of a particular architecture, it may helpful to visually rearrange these components, show multiple instances where appropriate, and even construct separate sub-view diagrams for each role. These choices are entirely dependent on the specific architecture requirements.

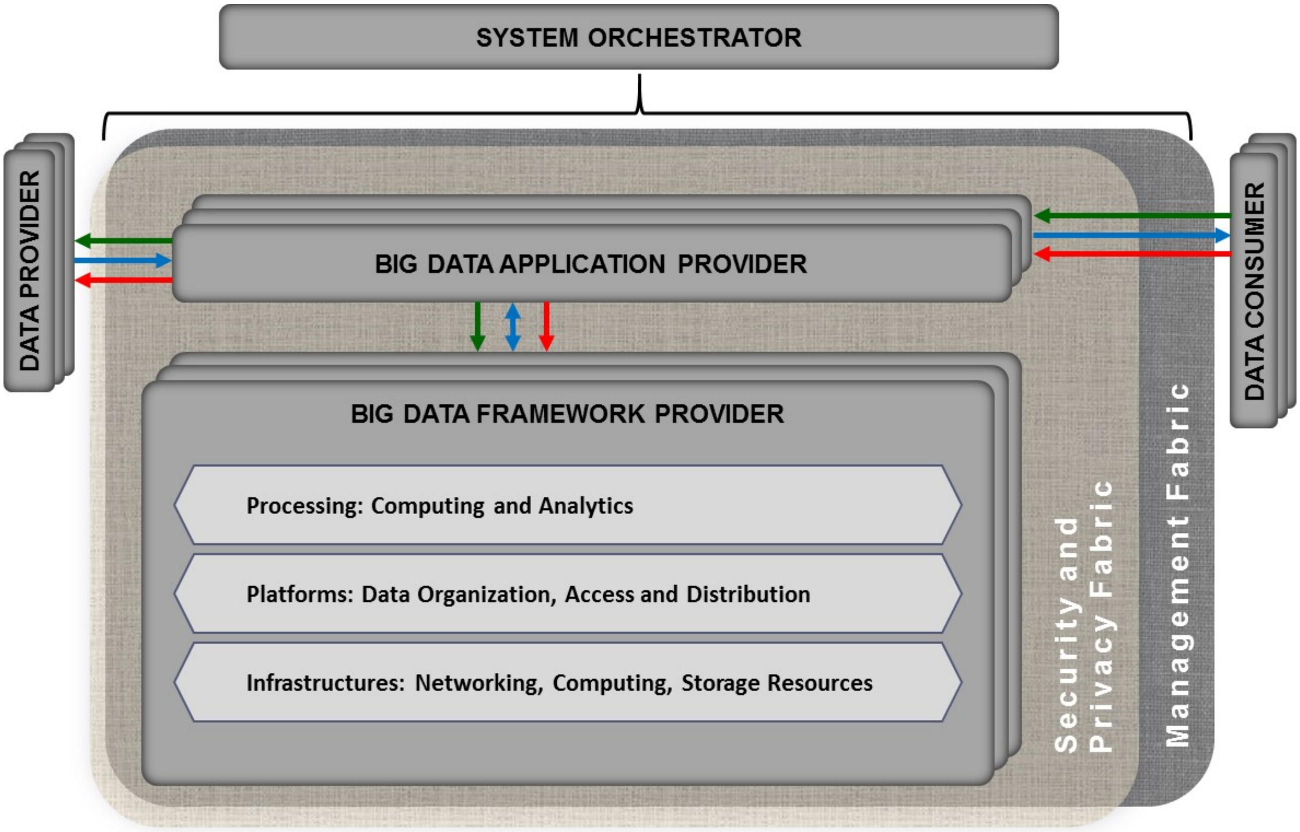

SYSTEM ORCHESTRATOR
DATA PROVIDER
BIG DATA APPLICATION PROVIDER
DATA CONSUMER
BIG DATA FRAMEWORK PROVIDER
Processing: Computing and Analytics
Platforms: Data Organization, Access and Distribution
Infrastructures: Networking, Computing, Storage Resources
Security and Privacy Fabric
Management Fabric

Figure 7: Top Level Roles and Fabrics

Sections 4.1 and 4.2 provide high-level examples of the types and classes of activities and functional components, respectively, that may be required to support a given architecture implementation. General classes and descriptions are provided in both cases because across the range of potential Big Data applications and architectures, the potential specific activities would be too numerous to enumerate and the rapid evolution of software/hardware functional components makes a complete list impractical.

It should also be noted that as one goes lower down the IT value chain of the architecture, the diversity and details of the activities and functional components would be less varied.

Finally, the sections below do not attempt to provide activity or functional component details for the Data Provider or Data Consumer roles. There are two reasons for this. First, a Data Provider could be anything from a simple sensor to a full-blown Big Data system itself. Providing a comprehensive list would be impractical as shown in the System of Systems View in Figure 5 above. Second, often the Data Provider and Data Consumer roles are supported by elements external to the architecture being developed and, thus are outside the control of the architect. The user of this report should enumerate and document those activities and functions to the extent it makes sense for their specific architecture. In cases where the Data Provider and Data Consumer roles are within the architecture boundary, the user is advised to create views based on similar roles, activities, and functional components found in the sections below. In cases where those roles are external to the architecture, the user should document any activities or components on which the architecture is dependent. For example, activities and components related to authentication or service-level agreements should be captured.

4.1 ACTIVITIES VIEW

As described above, the activities view is meant to describe what is performed or accomplished by various roles in the Big Data system. As per the definitions, an activity can be something performed by a person, organization, software, or hardware. Figure 8 below provides some top-level classes of activities by roles and sub-roles which may be applicable to a Big Data architecture implementation. The following paragraphs describe the roles and the classes of activities associated with those roles. The user is advised to use these examples primarily as guides and to create more specific classes of activities and associated descriptions as required to document their architecture.

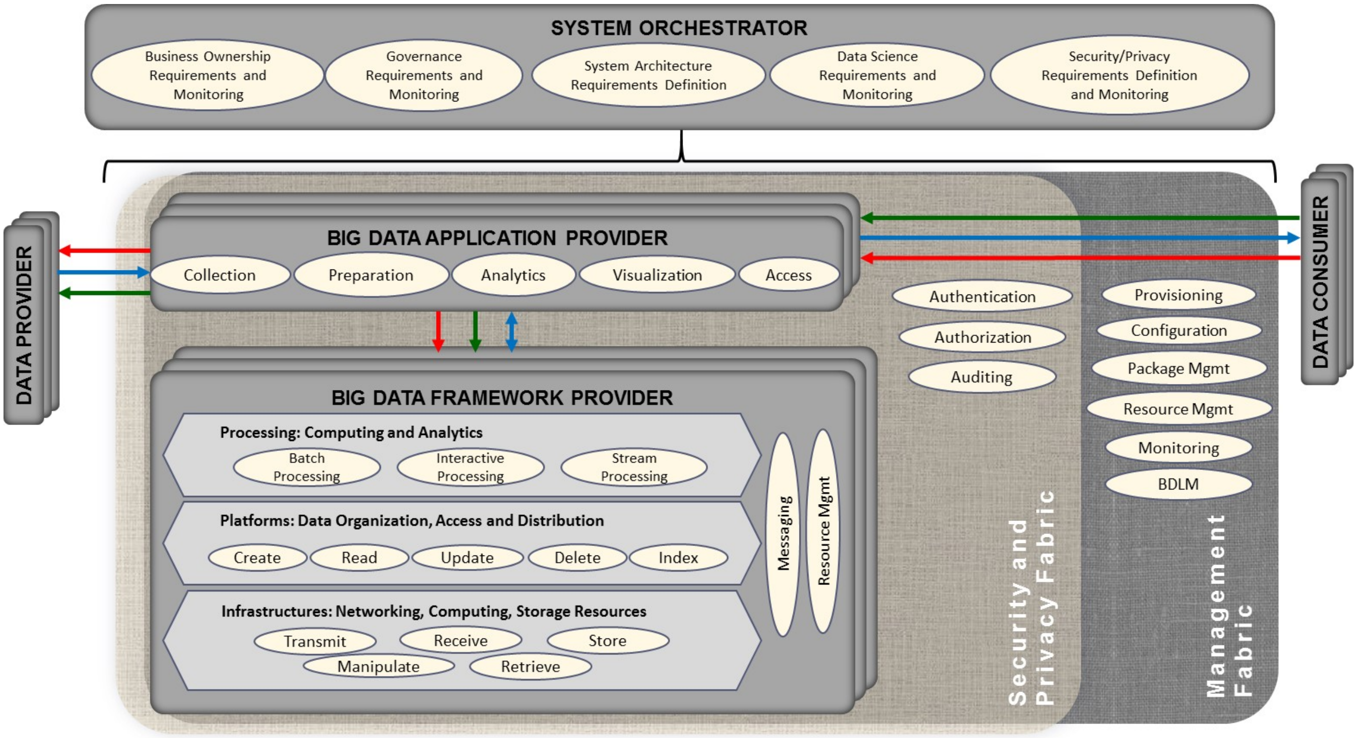

SYSTEM ORCHESTRATOR
Business Ownership Requirements and Monitoring
Governance Requirements and Monitoring
System Architecture Requirements Definition
Data Science Requirements and Monitoring
Security/Privacy Requirements Definition and Monitoring
DATA PROVIDER
BIG DATA APPLICATION PROVIDER
Collection
Preparation
Analytics
Visualization
Access
DATA CONSUMER
Authentication
Authorization
Auditing
Provisioning
Configuration
Package Mgmt
Resource Mgmt
Monitoring
BDLM
BIG DATA FRAMEWORK PROVIDER
Processing: Computing and Analytics
Batch Processing
Interactive Processing
Stream Processing
Platforms: Data Organization, Access and Distribution
Create
Read
Update
Delete
Index
Infrastructures: Networking, Computing, Storage Resources
Transmit
Receive
Store
Manipulate
Retrieve
Messaging
Resource Mgmt
Security and Privacy Fabric
Management Fabric

Figure 8: Top-Level Classes of Activities Within the Activities View

Because the Data Provider and Data Consumer roles can represent anything such as another computer system, a Big Data system, a person sitting at a keyboard, or remote sensors, the sub-roles and classes of activities associated with these roles can encompass any of the activity classes defined below or others. Users of the NBDRA should define the classes of activities and particular activities that address specific concerns related to their architecture implementation.

The following paragraphs describe the general classes of activities implemented within the roles, sub-roles, and fabrics of the NBDRA.

4.1.1 SYSTEM ORCHESTRATOR

The activities within the System Orchestrator role set the overall ownership, governance, and policy functions for the Big Data system by defining the appropriate requirements. These activities take place primarily during the system definition phase but must be revisited periodically throughout the life cycle of the system. The other primary aspect of activities under this role is the monitoring of compliance with the associated requirements.

Some classes of activities that could be defined for this role in the architecture include requirements definition and compliance monitoring for:

- Business Ownership: This activity class defines which stakeholders own and have responsibility for the various parts of the Big Data System. This activity would define the ownership and responsibility for the activities and functional components of the rest of the system and how that ownership will be monitored.
- Governance: This activity class would define the policies and process for governance of the overall system. These governance requirements would in turn be executed and monitored by the stakeholders defined as owners for the respective parts of the system.
- System Architecture: This class of activities involves defining the overall requirements that must be met by the system architecture. In general, activities in this class establish the technical guidelines that the overall system must meet and then provide the policies for monitoring the overall architecture to verify that it remains in compliance with the requirements.
- Data Science: Activities in this class would define many of the requirements that must be met by individual algorithms or applications within the system. These could include accuracy of calculations or the precision/recall of data mining algorithms.
- Security/Privacy: While no classes of activities are considered mandatory, this class is certainly the most critical and any architecture without well-defined security and privacy requirements and associated monitoring is bound to be at extreme risk. Security deals with the control of access to the system and its data and is required to ensure the privacy of personal or corporate information. Privacy relates to both securing personal information but also defining the policies and controls by which that information or derived information may or may not be shared.

Other classes of activities that may be addressed include the following:

- Quality Management,
- Service Management, and
- Audit Requirements.

4.1.2 BIG DATA APPLICATION PROVIDER

4.1.2.1 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.

4.1.2.2 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), cleaning (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 dataset.

4.1.2.3 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.

4.1.2.4 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.

4.1.2.5 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.

4.1.3 BIG DATA FRAMEWORK PROVIDER

The Big Data Framework Provider role supports classes of activities associated with providing management and communications between the subordinate sub-roles (i.e., Processing, Platforms, and Infrastructures) and their classes of activities. Two common classes of activities associated with this role are the following:

- Messaging: This activity class provides the necessary message queues and other communication mechanisms that support communications between the activities within the Big Data Framework Provider sub-roles and the Big Data Application Provider activities.
- Resource Management: Resources available to a given Big Data system are finite, so activities that manage the allocation of resources to other sub-roles and activities are necessary. Such activities would ensure that resources are allocated an appropriate priority status relative to other activities and that resources, such as memory and central processing unit (CPU), are not oversubscribed.

4.1.3.1 Infrastructure Activities

Classes of activities within the Infrastructure sub-role support the underlying computing, storage, and networking functions required to implement the overall system. These activity classes reflect the underlying operations performed on data within the system to include: Transmission, Reception, Storage, Manipulation, and Retrieval. These activities may be associated with physical or virtual infrastructure resources. In defining the specific activities for a given system, the focus should be on specific types of activities. For example, a system which requires highly parallel processing of large matrices or data may specify an activity which supports Single Instruction Multiple Data computing, such as that provided by Graphic Processing Units (GPUs). Transmission activities may include descriptions of data transmission requirements which define the required throughput and latency. Storage and retrieval activities might describe performance of volatile or non-volatile storage.

4.1.3.2 Platform Activities

The Big Data Platform Provider sub-role is associated with activities which manage the organization and distribution of data within the Big Data system. Since many Big Data systems are horizontally distributed across multiple infrastructure resources, specific activities related to creating data elements can specify that data will be replicated across a number of nodes and will be eventually consistent when accessed from any node in the cluster. Other activities should describe how data will be accessed and what type of indexing is required to support that access. For example, geospatial data requires specialized indexing for efficient retrieval. So a related activity might describe maintaining a z-curve type of index.

4.1.3.3 Processing Activities

Processing activities describe how data will be processed in support of Big Data applications. This processing generally falls into a continuum, from long-running batch jobs to responsive processing, and supports interactive applications of continuous stream processing. The types of processing activities described for a given architecture would be dependent on the characteristics (volume and velocity primarily) of the data processed by the Big Data Application Providers and their requirements. Depending on the type of processing required, an activity might describe MapReduce or Bulk Synchronous Parallel (BSP) processing for batch-oriented requirements. Streaming activities might specify the performance requirements necessary to handle the volume or velocity of data.

4.1.4 MANAGEMENT FABRIC ACTIVITIES

4.1.4.1 System Management

To address the challenge of daily demands of operating multiple Big Data applications, a Big Data Management Fabric may be needed by planners, operators, and data center owners. Stated broadly, Big Data creates a need for larger or novel forms of operational intelligence. These include the following:

- Configuration activities associated with management of potential accountability and traceability for data access associated with individual subjects / consumers, as well as their associated organizations.
- Resource management activities to support burst and peak demand tied to both planned and unplanned usage changes. Specific activities would be defined to support the automated allocation of resources to meet demand. By predicting the fluctuations in load, the impact of those fluctuations can be smoothed through simulation, predictive load analytics, more intelligent monitoring, and practical experience. Modeling and simulation for operational intelligence may become essential in some settings [11], [12].
- Monitoring activities to support operational mitigation and resilience for both centralized and decentralized services. These activities may also support load balancing in conjunction with resource management activities to avoid outages during unexpected peak loads and reduce costs during off-peak times. Real-time monitoring, gating, filtering, and throttling of streaming data requires new approaches due to the “variety of tasks, such as performance analysis, workload management, capacity planning, and fault detection. Applications producing Big Data make the monitoring task very difficult at high-sampling frequencies because of high computational and communication overheads [13].”
- Provisioning and package management activities to support automated deployment and configuration of software and services. This class of activities is frequently associated with the emerging Dev/Ops movement designed to automate the frequent deployment of capabilities into production. Movement toward automated methods for ensuring information assurance (versus training and governance: they may not scale). See references [14] and [15].
- BDLM activities support the overall life cycle of data throughout its existence within the Big Data system. Of all the classes of management fabric activities, the BDLM activities are the most affected by the Big Data characteristics and merit the additional discussion below.

4.1.4.2 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. The 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, which 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, unconstrained data storage 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:** This involves the change of data accessibility over time. For example, census data can be made available to the public after 72 years. 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 controls access to the functions and data usage produced by the Big Data system. This data access control can be informed by the metadata, which is managed and updated by BDLM.

4.1.5 SECURITY AND PRIVACY FABRIC ACTIVITIES

The Security and Privacy Fabric provides the activities necessary to manage the access to system data and services. The primary classes of activities associated with this fabric are:

- **Authentication:** This class of activities includes validation that the user or process is who they claim to be. The specific authentication activities may specify the type of authentication, such as two-factor or private key.
- **Authorization:** This class of activities ensures that the user or process has the rights to access resources or services. Access controls may define the specific access privileges (e.g., create, update, delete) for the data or services. The authorization activities may specify broad role-based access controls or more granular attribute-based access controls.
- **Auditing:** These activities record events that happen within the system to support both forensic analysis in the event of a breach or corruption of data, as well as providing for maintenance of providence and pedigree for data.

Depending on the allocation of responsibilities, the Security and Privacy Fabric may also support certain provisioning and configuration activities. For example, activities for regular monitoring of system or application configuration files to ensure that there have been no unauthorized changes may be allocated to this fabric. In reality, the activities in the Security and Privacy Fabric and Management Fabric must, at a minimum, interact and will frequently involve shared responsibilities.

4.2 FUNCTIONAL COMPONENT VIEW

The functional component view of the reference architecture should define and describe the functional components (e.g., software, hardware, people, organizations) that perform the various activities outlined in the activities view. Activities and functional components need not map one-to-one and in fact, many functional components may be required to execute a single activity and multiple activities may be performed by a single functional component. The user of this model is recommended to maintain a mapping of activities to functional components to support verification that all activities can be performed by some component and that only components that are necessary are included within the architecture. Figure 9 below shows classes of functional components common to the various roles, sub-roles, and fabrics of the NBDRA. These classes are described in the following paragraphs.

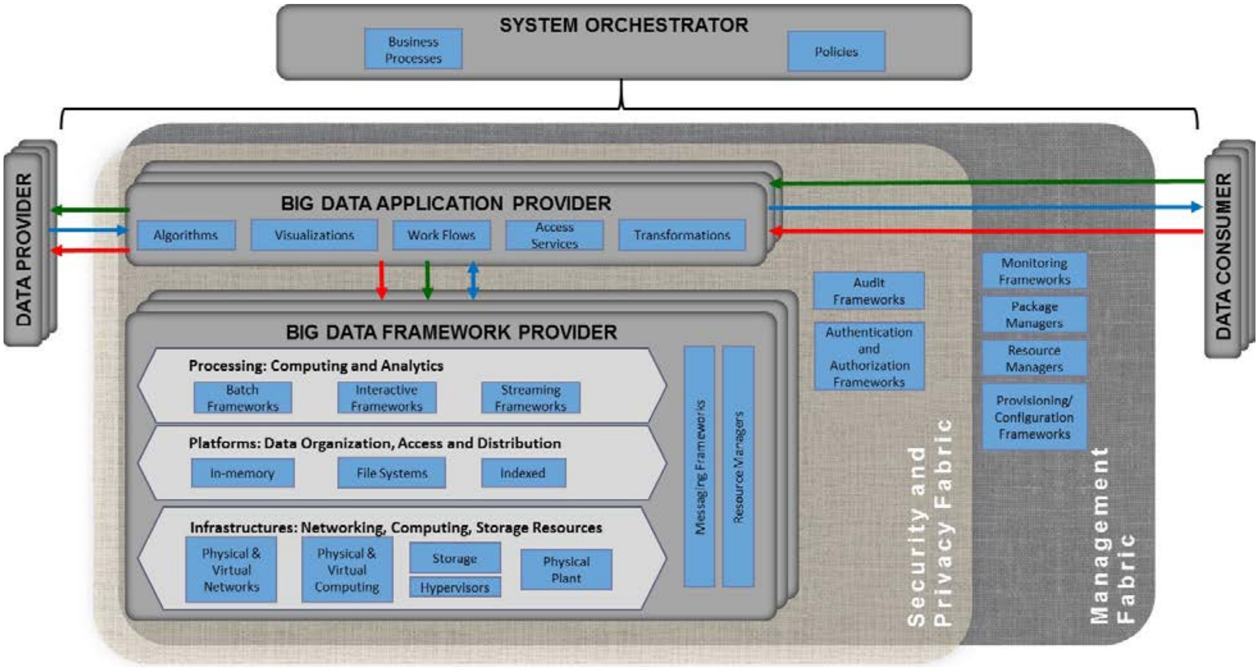


Figure 9: Common Classes of Functional Components

4.2.1 SYSTEM ORCHESTRATOR

The classes of functional components for the system orchestrator revolve around the policies and processes that govern the operation of the Big Data system. These policies and processes define the requirements for how other functional components must behave and interact. Often the policies and processes are derived from community best practices or standards such as International Organization of Standardization (ISO) 20000 for IT Services Management or ISO 27000 for Information Technology Security. Other classes of processes and policies may include ones for data sharing, external system access, and how privacy-sensitive data is to be handled.

4.2.2 BIG DATA APPLICATION PROVIDER

The functional components within the Big Data Application Provider implement the specific functionality of the Big Data system. The classes for components within a Big Data application include:

- **Work Flows:** These components would control how data and/or users go through the functions of the system. These are often implemented within frameworks or enterprise service bus components that would also be included here.
- **Transformations:** These components are responsible for reformatting data to meet the needs of the algorithms or visualizations. The transformations may also invoke algorithms to support the transformation. These may be embedded in other components, such as ETL tools.

- **Visualizations:** The visualization components are responsible for formatting data to present to an end user. These visualizations may be textual or graphic and are frequently implemented with other framework or tool functional components. For example, textual visualizations may be implemented using report writer components while a graphic visualization of the output of a clustering algorithm may be implemented by a charting framework component.
- **Access Services:** These components provide access to the Big Data system to the Data Consumers and may be designed for use by humans or other systems. Frequently, these specific components are implemented within other frameworks or components such as web services containers.
- **Algorithms:** This class of components is the heart of the application functionality. They can range from simple summarization and aggregation algorithms to more complex statistical analysis such as clustering, or graph traversal/analysis algorithms.

Algorithms themselves can be classified into general classes which may be defined as functional components. In 2004, a list of algorithms for simulation in the physical sciences was developed that became known as the *Seven Dwarfs* [16]. The original list of seven dwarfs was modified in 2006 and extended to 13 algorithms (Table 2) based on the following definition: “A dwarf is an algorithmic method that captures a pattern of computation and communication.”[3]

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**
MapReduce	

Notes:
* Indicates one of the original seven dwarfs. The following modifications to the original list of seven algorithms were made in 2006: Fast Fourier Transform, Particles, and Monte Carlo were removed. MapReduce was added.

Many other algorithms or processing models have been defined over the years. MapReduce, and Bulk Synch Processing (BSP) are perhaps the two best known models in the Big Data space today. These are described in the following subsections.

4.2.2.1 MapReduce

Several major Internet search providers popularized the MapReduce model as they worked to implement their search capabilities. In general, MapReduce 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.

[3] Patterson, David; Yelick, Katherine. Dwarf Mind. A View from Berkeley. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

While there is a single output, nothing in the model prohibits multiple input datasets. It is extremely common for complex analytics to be built as workflows of multiple MapReduce jobs. While the MapReduce programming model is best suited to aggregation-type analytics (e.g., sum, average, group-by), a wide variety of analytic algorithms have been implemented within processing frameworks. MapReduce 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 that the entire dataset be read, output, and then moved or copied over the original dataset. 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 dataset do not tend to scale or perform well under MapReduce.

Due to its shared-nothing approach, the usability of MapReduce 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 MapReduce 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.

4.2.2.2 Bulk Synchronous Parallel

The BSP programming model, originally developed by Leslie Valiant [17], 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 of the 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 MapReduce 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 interconnected 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 superstep 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 superstep 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.

4.2.3 BIG DATA FRAMEWORK PROVIDER

The Big Data Framework Provider provides the infrastructure required to support the Big Data Application Provider. Components within the Big Data Framework Provider fall within three overall sub-roles (i.e., processing, platforms, infrastructures) along with some specific crosscutting roles, which support the communication and integration of components within the overall provider.

4.2.3.1 Infrastructure Frameworks

This Infrastructure Frameworks sub-role of the Big Data Framework Provider provides all of the resources necessary to host/run the activities of the other roles 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.
- Physical Plant: These are the environmental 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.

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

4.2.3.1.1 Hypervisors

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 implemented via *hypervisors* that are 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 because in the event that the container controls/limits fail, one application may take down every application sharing those physical resources.

4.2.3.1.2 Physical and Virtual Networks

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 addressed.

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

4.2.3.1.2.1 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.

4.2.3.1.2.2 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.

4.2.3.1.3 Physical and Virtual Computing

The logical distribution of cluster/computing infrastructure may vary from a tightly coupled high performance computing (HPC) 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 (GPU) and Field Programmable Gate Arrays (FPGA), which can provide dynamically programmed massively parallel computing capabilities to individual nodes in the infrastructure.

4.2.3.1.4 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.

4.2.3.1.5 Physical Plant

Environmental resources, such as power and heating, ventilation, and air conditioning provided by physical plant components, 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 3, 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 (e.g., badge/biometrics), surveillance, and perimeter alarms all are necessary to maintain control of the data being processed.

4.2.3.2 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 10 presents a brief overview of data organization approaches for Big Data.

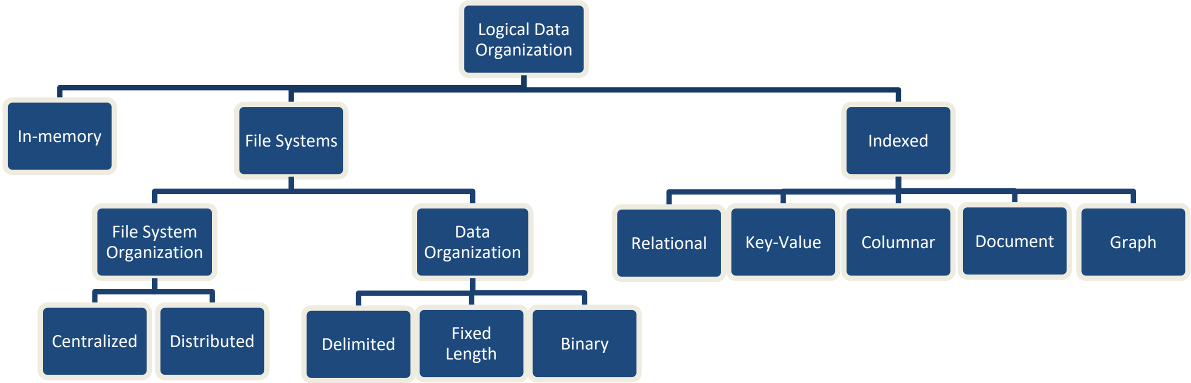


Figure 10: Data Organization Approaches

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.

4.2.3.2.1 In-memory

The infrastructure illustrated in the NBDRA (Figure 3) 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 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 supercomputer 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.

4.2.3.2.2 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.

4.2.3.2.2.1 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 datasets 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.

4.2.3.2.2.2 In File Data Organization

Very little is different for in file data organization in Big Data. 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.

4.2.3.2.3 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 dataset) 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 dataset into memory or being able to distribute the application and dataset 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 11. 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.

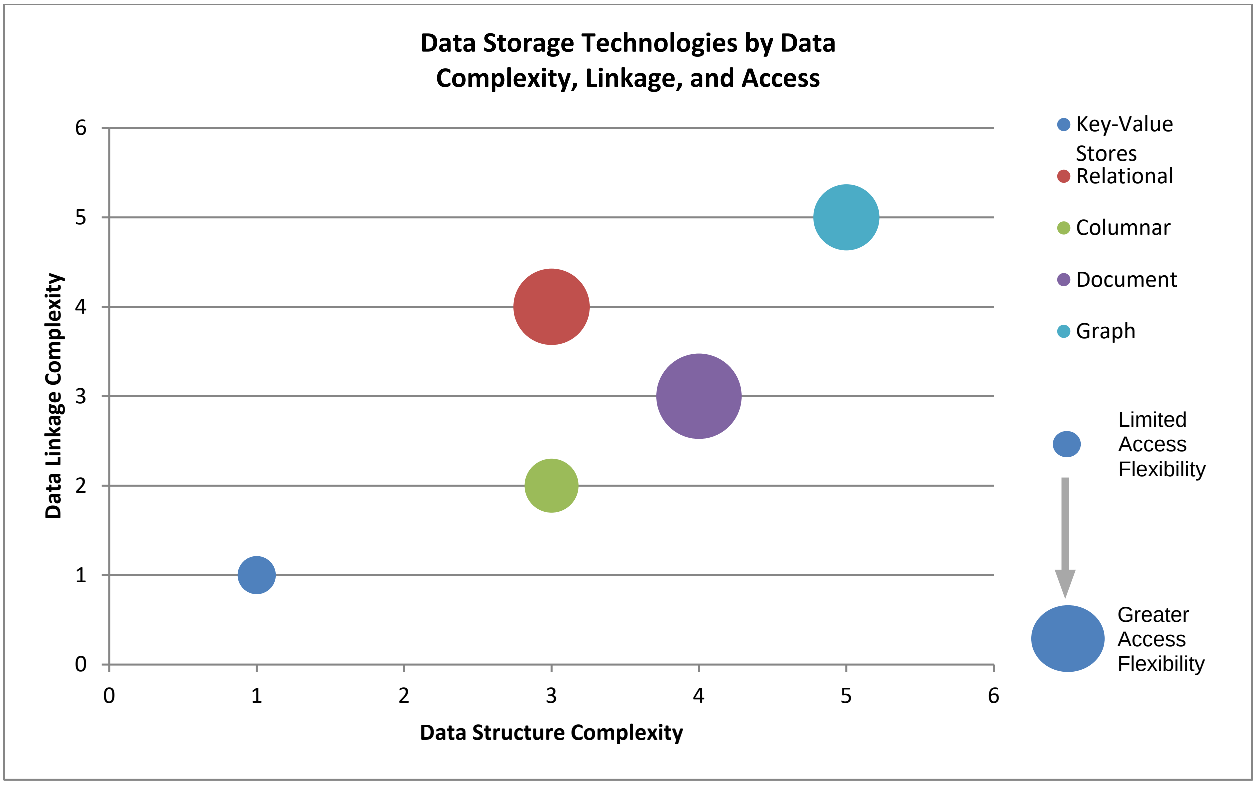


Figure 11: Data Storage Technologies

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

- Relational storage platforms,
- Key-value storage platforms,
- Wide columnar storage platforms,
- Document storage platforms, and
- Graph storage platforms.

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).

4.2.3.2.3.1 Relational Storage Platforms

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 two 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 MapReduce 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 require 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 provided only 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 is 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.

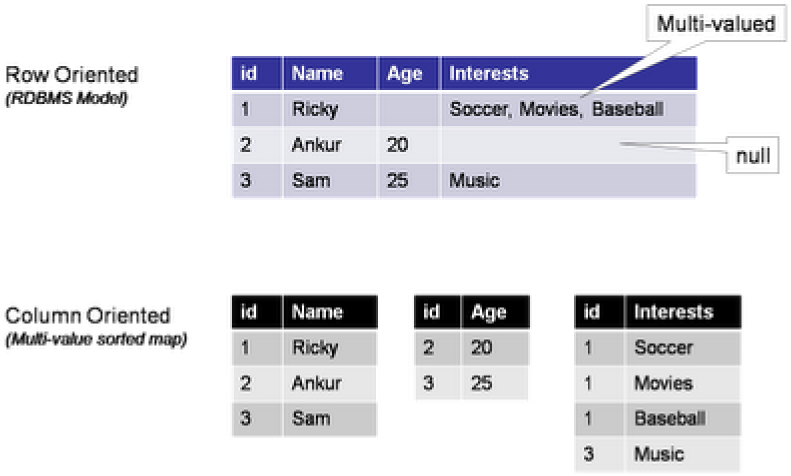
4.2.3.2.3.2 Key-Value Storage Platforms

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 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 dataset.

4.2.3.2.3.3 Wide Columnar Storage Platforms

Much of the hype associated with Big Data came with the publication of the BigTable paper in 2006 [18] 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 12 below shows the basic differences between row-oriented and column-oriented stores.



Row Oriented
(RDBMS Model)

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

Figure 12: 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 13 below.

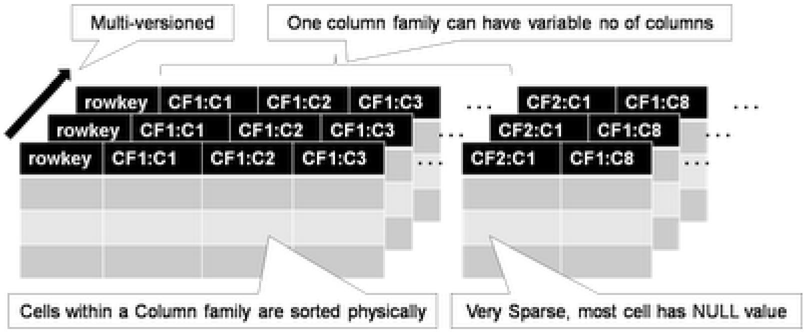

Multi-versioned
One column family can have variable no of columns
rowkey
CF1:C1
CF1:C2
CF1:C3
...
CF2:C1
CF1:C8
Cells within a Column family are sorted physically
Very Sparse, most cell has NULL value

Figure 13: 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 a 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.

4.2.3.2.3.4 Document Storage Platforms

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. This is one reason that document stores are frequently popular for datasets 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. For 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., JavaScript Object Notation [JSON], XML).

4.2.3.2.3.5 Graph Storage Platforms

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 counterterrorism 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 14 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 that 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.

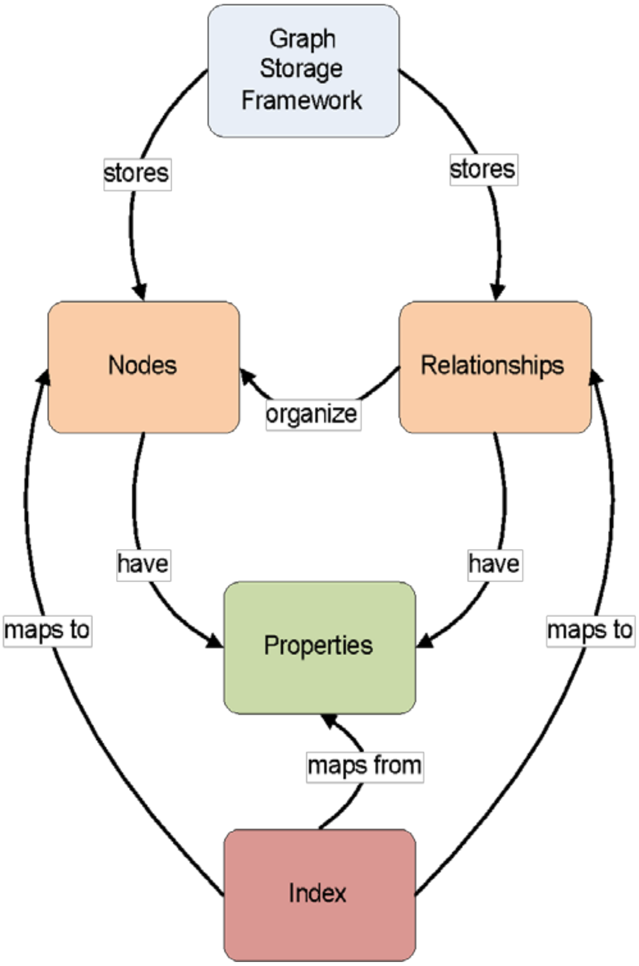


Figure 14: Object Nodes and Relationships of Graph Databases

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 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 a directed labeled graph. The contents of RDF stores are frequently described using formal ontology languages like the W3C Web Ontology Language (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 datasets, 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 [19], [20].

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.

4.2.3.3 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 the NBDRA (Figure 3), 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 15.

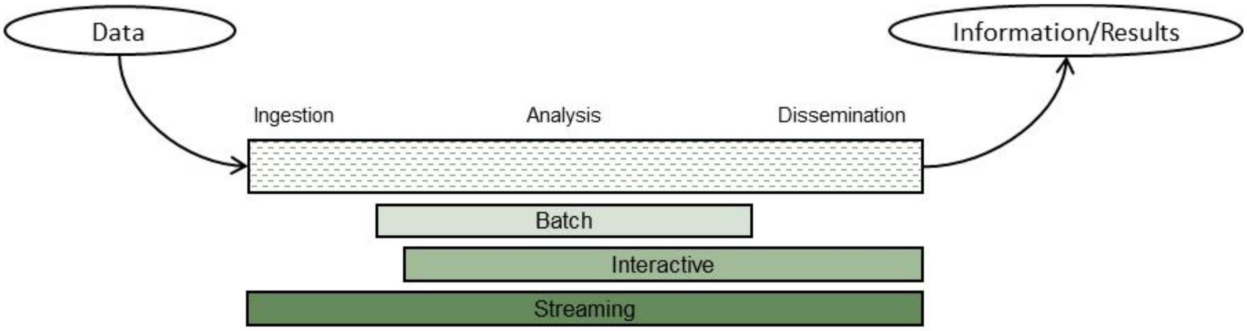


Figure 15: Information Flow

The green coloring in Figure 15 illustrates the general sensitivity of that processing style 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. The darker the shade, the more sensitive to latency. 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 Complex Event Processing) may or may not require NRT processing. The Data Consumer generally is located at the far right of Figure 15. 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 provide 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.

4.2.3.3.1 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.

4.2.3.3.2 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.

4.2.3.3.2.1 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 identification (ID) to uniquely identify each message in the stream.

4.2.3.3.2.2 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.

4.2.3.3.2.3 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.

4.2.3.4 Crosscutting Components

Because the components within the three sub-roles within the Big Data Framework Provider must share resources and communicate, two major classes of crosscutting components are needed: Messaging/Communications Frameworks and Resource Management Frameworks.

4.2.3.4.1 Messaging/Communications Frameworks

Messaging and communications frameworks have their roots in the 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.

4.2.3.4.2 Resource Management Frameworks

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.

4.2.4 MANAGEMENT FABRIC

The management fabric encompasses components responsible for the establishing and continuing operation of the system.

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, MapReduce 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.

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.

4.2.4.1 Monitoring Frameworks

To monitor 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 cloud service providers 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 monitoring.

4.2.4.2 Provisioning/Configuration Frameworks

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. Such frameworks also interact with the Security and Privacy Fabric to ensure that the system configuration continually meets the security requirements outlined in the policies specified by the System Orchestrator.

4.2.4.3 Package Managers

Package management components support the installation and updates of other components within the Big Data system. This class of components is often provided by the underlying operating system component and is invoked by the provisioning /configuration frameworks to install and update components within the system. Components within this class generally leverage a central network repository to ensure that the correct component version is deployed consistently across the cluster. In many Big Data systems, this same repository is leveraged to support the deployment of application components and, in some cases, even data components.

4.2.4.4 Resource Managers

Resource management components within the Management Framework provide the system with the overall resources necessary to support the system. These components will work with external resource providers such as Cloud Service Providers to acquire the resources necessary to provision the other components of the system. They will handle requests for additional resources from resource managers within the Big Data Framework Provider when required and coordinate with the Provisioning/Configuration Frameworks to properly configure other components across those resources.

4.2.4.5 Data Life Cycle Managers

Life Cycle Data Management components are necessary to manage the life cycle of the data ingested into the system, stored and preserved in the system, and accessed for processing or dissemination purposes:

Metadata Catalog is the inventory of all datasets in the system. It should contain the model for the foundational concept of “unit” of data, whether it is a database record (e.g., key-value pair or relational table row), or a dataset (e.g., database export file). Each data unit has characteristics maintained in the associated metadata, which should include at least a unique identifier and timestamp indicating when the data was created and/or ingested. These timestamps will help the Data Life Cycle Manager to monitor the “age” of the data within the system. Moreover, the Metadata Catalog will have to support data discovery that is necessary for data access and data governance. There are numerous international and national standards which govern the content, model, and interfaces for metadata catalogs.

The Data Tracker tracks the movement of data throughout the system, from the ingestion point to the dissemination or destruction point. The Data Tracker component handles the Volume and Variety characteristics inherent to Big Data. The two kinds of movements are as follows:

- ***Ingress and egress movement***: tracks data entering and exiting the system. Data exiting means that the data are dispositioned to satisfy the retention policy, which can originate from either the need of the Big Data application or preservation policy. Indeed, some applications may require “fresh” data for analytical purposes. The degree of freshness depends on the specific requirements of the business applications, and can be influenced by policy and regulations. For instance, while the visual analytics application monitoring the approval or disapproval feedback during a presidential election debate requires real-time data and most recent tweet and blog data, the study of the trend of household income over the past 50 years needs both recent and archived Census data. On the other hand, records management laws and policies may dictate the retention time for the data, and hence impact the Data Preservation.
- ***Intra-system movement:*** Due to the large volume of Big Data, the Big Data Framework Provider will likely have multitiered storage for cost-efficiency and scalability. Within that storage environment, data is made available to the analytics processes managed by the Big Data Application Provider. Commercial infrastructure vendors offer different storage categories with different pricing models. The action of making data available to processes and applications may be realized by physically moving the data to storage where the processing software can operate. However, a recent paradigm is to move computation and processing capabilities to where data are located to circumvent the large data transfer between storage tiers.

The Data Tracker may interface with the Data Preservation component to implement preservation and long-term storage policies.

The Data Preservation component is applied to both permanent and temporary data. Its responsibility is to continuously inspect the “age” of data in the system, and operate on the data based on the retention policy. For permanent data, Data Preservation will perform the Preservation Plan, which can consist of migrating data to a long-term preservation format, periodically refreshing the storage hardware, or maintaining emulation environments used to read the archived data. Data Preservation will leverage the multitiered storage which satisfies data durability requirement, and achieves cost-efficiency. If data are deemed to have limited lifetime, then Data Preservation will apply appropriate disposition methods to purge them from the system. The purge methods will depend on the security policy to ensure data confidentiality.

4.2.5 SECURITY AND PRIVACY FABRIC

The components within the Security and Privacy Fabric implement the core activities supporting the overall security and privacy requirements outlined by the policies and processes of the System Orchestrator.

4.2.5.1 Authentication and Authorization Frameworks

Components within this class must interface and interact with all other components within the Big Data system to support access control to the data and services of the system. This support includes authenticating the user or service attempting to access the system resource to validate their identity. This class of components provides APIs to other services and components for collecting the identity information, and validating that information against a trusted store of identities. Frequently these components will provide an identification token back to the invoking component that defines allowed access for the life of a session. This token can also be used to retrieve authorizations for the users/components detailing what data and service resources they may access. These authorizations can be used by the components to limit access to data or even filter data provided in response to requests by components. Typically, a component will pass the identification token as part of the request which the receiving component will use to look up authorizations from a trusted store to manage the access to the underlying resources (data or services).

4.2.5.2 Audit Frameworks

Audit Framework components are responsible for collecting, managing, consolidating, and in some cases monitoring events from across the system that reflect access to and changes to data and services across the system. The scope and nature of the events collected is based on the requirements specified by the policies within the System Orchestrator. Typically, these components will collect and store this data within a secure centralized repository within the system and manage the retention of this data based on the policies. The data maintained by these components can be leveraged during system operation to provide providence and pedigree for data to users or application components as well as for forensic analysis in the response to security or data breaches. Because of the number and frequency of operations and events which may be generated by a large Big Data system, the framework itself must deal with the Big Data characteristics of volume and velocity. To handle this, many Big Data system architectures implement a Big Data system instance specifically for management and storage of this data. Monitoring frameworks within the Management Fabric may execute algorithms within this Big Data system instance to provide alerts to potential security or data issues.

5 SUMMARY

This document (Version 3) presents the overall NBDRA conceptual model along with architecture views for the activities performed by the architecture and the functional components that would implement the architecture.

The purpose of these views is to provide the system architect a framework to efficiently categorize the activities that the Big Data system will perform and the functional components which must be integrated to perform those activities. During the architecture process, the architect is encouraged to collaborate closely with the system stakeholders to ensure that all required activities for the system are captured in the activities view. Those activities should then be mapped to functional components within that view using a traceability matrix. This matrix will serve to validate that components will be integrated into the architecture to accomplish all required activities and that all integrated functional components have a purpose within the architecture.

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 allow it to be deployed using multiple infrastructure elements as described in Section 4.2.3. 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 required.

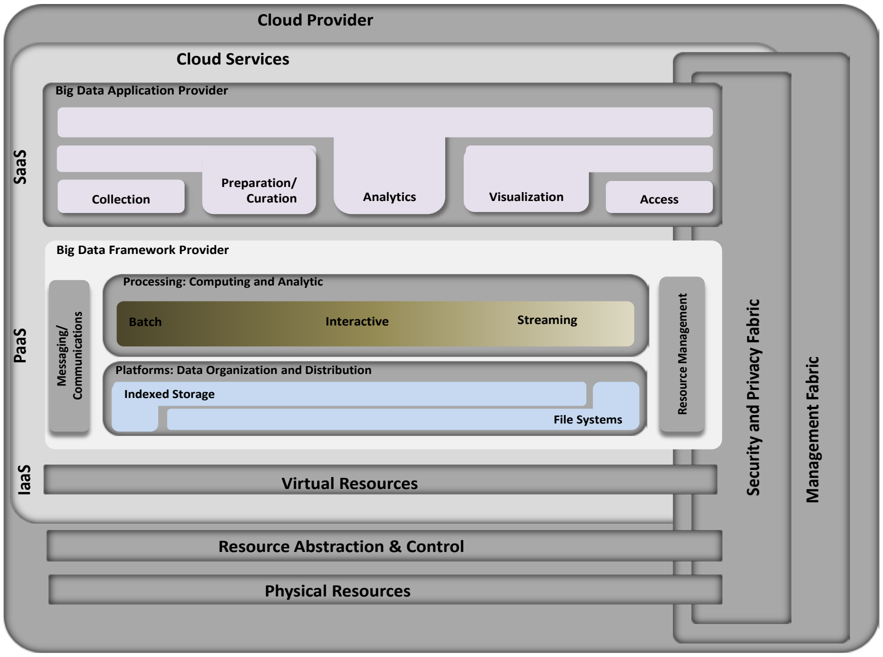


Figure A-1: Big Data Framework Deployment Options

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 with the NIST Cloud Reference architecture [21]. 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) [22].”
- 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, operating systems, or storage, but has control over the deployed applications and possibly” application-hosting environment configurations [22].
- SaaS: “The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. … The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings [22].”

Appendix C: 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
CPR	Capability Provider Requirements
CPU	central processing unit
CRUD	create/read/update/delete
CSP	Cloud Service Provider
CSV	comma separated values
DCR	Data Consumer Requirements
DDF	Data Description Framework
DLM	data life cycle management
DNS	Domain Name Server
DSR	Data Source Requirements
ELT	extract, load, transform
ETL	extract, transform, load
FPGA	Field Programmable Gate Arrays
FTP	file transfer protocol
GB	gigabyte
GPU	graphic processing units
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
ID	identification
ISO	International Organization of Standardization
IT	information technology
ITL	Information Technology Laboratory
JSON	JavaScript Object Notation
LMR	Life Cycle Management Requirements
NARA	National Archives and Records Administration
NAS	network-attached storage
NASA	National Aeronautics and Space Administration
NBDIF	NIST Big Data Interoperability Framework
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
OR	Other Requirements
OS	operating system
OWL	W3C Web Ontology Language
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
SPR	Security and Privacy Requirements
SQL	Structured Query Language
TCP	Transmission Control Protocol
TPR	Transformation Provider Requirements
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Appendix D: Resources and Bibliography

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, 2012, http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_fact_sheet_final_1.pdf.

BIBLIOGRAPHY

[1] W. L. Chang (Co-Chair), N. Grady (Subgroup Co-chair), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 1, Big Data Definitions (NIST SP 1500-1 VERSION 3),” Gaithersburg MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-1r2

[2] W. L. Chang (Co-Chair), N. Grady (Subgroup Co-chair), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 2, Big Data Taxonomies (NIST SP 1500-2 VERSION 3),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-2r2

[3] W. L. Chang (Co-Chair), G. Fox (Subgroup Co-chair), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 3, Big Data Use Cases and General Requirements (NIST SP 1500-3 VERSION 3),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-3r2

[4] W. L. Chang (Co-Chair), A. Roy (Subgroup Co-chair), M. Underwood (Subgroup Co-chair), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 4, Big Data Security and Privacy (NIST SP 1500-4 VERSION 3),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-4r2

[5] W. L. Chang (Co-Chair), S. Mishra (Editor), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 5, Big Data Architectures White Paper Survey (NIST SP 1500-5 VERSION 1),” Sep. 2015.

[6] W. L. Chang (Co-Chair), R. Reinsch (Subgroup Co-chair), D. Boyd (Version 1 Subgroup Co-chair), C. Buffington (Version 1 Subgroup Co-chair), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 7, Big Data Standards Roadmap (NIST SP 1500-7 VERSION 3),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-7r2

[7] W. L. Chang (Co-Chair), G. von Laszewski (Editor), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 8, Big Data Reference Architecture Interfaces (NIST SP 1500-9 VERSION 2),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-9r1

[8] W. L. Chang (Co-Chair), R. Reinsch (Subgroup Co-chair), C. Austin (Editor), and NIST Big Data Public Working Group, “NIST Big Data Interoperability Framework: Volume 9, Adoption and Modernization (NIST SP 1500-10 VERSION 2),” Gaithersburg, MD, Sep. 2019 [Online]. Available: https://doi.org/10.6028/NIST.SP.1500-10r1

[9] T. White House Office of Science and Technology Policy, “Big Data is a Big Deal,” *OSTP Blog*, 2012. [Online]. Available: http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal. [Accessed: 21-Feb-2014]

[10] N. and I. I. (OASD/NII) Office of the Assistant Secretary of Defense, “Reference Architecture Description,” 2010 [Online]. Available:

http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf

[11] A. D. N. Sarma, “Architectural Framework for Operational Business Intelligence System,” *Int. J. Innov. Manag. Technol.*, vol. 5, no. 4, p. 7, 2014 [Online]. Available: http://www.ijimt.org/papers/529-E318.pdf

[12] S. C. L. Koh and K. H. Tan, “Operational intelligence discovery and knowledge-mapping approach in a supply network with uncertainty,” *J. Manuf. Technol. Manag.*, vol. 17, no. 6, pp. 687–699, 2006.

[13] M. Andreolini, M. Colajanni, M. Pietri, and S. Tosi, “Adaptive, scalable and reliable monitoring of big data on clouds,” *J. Parallel Distrib. Comput.*, vol. 79–80, pp. 67–79, 2015.

[14] V. Lemieux, B. Endicott-Popovsky, K. Eckler, T. Dang, and A. Jansen, “Visualizing an information assurance risk taxonomy,” in *VAST 2011 - IEEE Conference on Visual Analytics Science and Technology 2011, Proceedings*, 2011, pp. 287–288.

[15] L. Duboc, E. Letier, D. S. Rosenblum, and T. Wicks, “A case study in eliciting scalability requirements,” in *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE’08*, 2008, pp. 247–252.

[16] P. Colella, “Defining software requirements for scientific computing (Slide in ‘Can Computer Architecture Affect Scientific Productivity?’),” in *Salishan Conference on High-speed Computing, 2005*, 2004 [Online]. Available: http://www.lanl.gov/orgs/hpc/salishan/salishan2005/davidpatterson.pdf

[17] L. G. Valiant, “A bridging model for parallel computation,” *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990 [Online]. Available: http://portal.acm.org/citation.cfm?doid=79173.79181

[18] F. Chang *et al.*, “Bigtable: A distributed storage system for structured data,” *7th Symp. Oper. Syst. Des. Implement. (OSDI ’06), Novemb. 6-8, Seattle, WA, USA*, pp. 205–218, 2006 [Online]. Available: http://research.google.com/archive/bigtable-osdi06.pdf

[19] B. Smith, T. Malyuta, W. S. Mandirck, C. Fu, K. Parent, and M. Patel, “Horizontal Integration of Warfighter Intelligence Data,” in *Semantic Technology in Intelligence, Defense and Security (STIDS)*, 2012, p. 8 [Online]. Available: http://ontology.buffalo.edu/smith/articles/Horizontal-integration.pdf

[20] S. Yoakum-Stover and T. Malyuta, “Unified data integration for situation management,” in *Proceedings - IEEE Military Communications Conference MILCOM*, 2008.

[21] F. Liu *et al.*, “NIST Cloud Computing Reference Architecture, SP 500-292,” *Spec. Publ. 500-292*, p. 35, 2011 [Online]. Available: http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=909505

[22] P. Mell and T. Grance, “NIST SP 800-145: The NIST Definition of Cloud Computing,” 2011 [Online]. Available: http://www.mendeley.com/research/the-nist-definition-about-cloud-computing/