

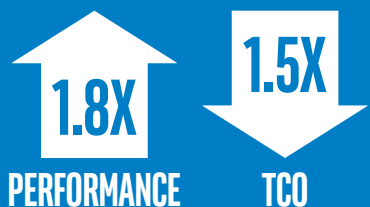


# Magnify Business Outcome: Building a Big Data Analytics Warehouse

Take control of your big data projects by harnessing the power of exclusive “better together” data center technologies driven by the Intel® Xeon® processor Scalable family, HPE ProLiant® DL Gen10 servers, and Cloudera Enterprise\*

Using this reference solution, enterprises can build a big data analytics warehouse that is optimized for their use case—increasing return on investment and lowering total cost of ownership.

## FASTER BIG DATA BATCH ANALYTICS



HPE ProLiant® DL Gen10 servers equipped with Intel® Xeon® Scalable processors provide a 1.8x increase in performance and a 1.5x decrease in total cost of ownership, compared to previous-generation processors.<sup>2</sup>

## Executive Summary

According to a Gartner survey, only 15 percent of enterprises have matured their big data projects from proof of concept (PoC) to production.<sup>1</sup> Enterprises struggle to realize the business value from big data analytics warehouse solutions because they lack the skills to effectively use the open source ecosystem and understand complex data characteristics, multiple use cases, and the nuances of distributed computing. The net result is a paralyzed state that prevents projects from moving into production, marked by inefficiencies and high costs.

For enterprises to unleash the value of analytics today and be ready for the future, they must learn the nuances of parallel computing and right-sizing their big data batch processing solutions. They can use the information in this paper to get a high return on investment from their infrastructure at a low total cost of ownership (TCO).

This paper discusses using a representative workload with large datasets in various use cases implemented on a big data analytics warehouse. HPE ProLiant® DL Gen10 servers equipped with Intel® Xeon® Scalable processors provide a 1.8x increase in performance and a 1.5x decrease in TCO, compared to previous-generation processors.<sup>2</sup>

Details include tips on sizing and optimization tactics for distributed batch processing solutions. We also discuss considerations during the PoC stage that can effectively lead to production deployment. With this information, enterprises can reduce guesswork, take control of their big data infrastructure, and build effective solutions optimized for performance and TCO.

## Introduction

Gaining insights from data has become a key differentiator for business. Enterprises that excel at big data analytics may be twice as likely to be among their industry's top 25 percent of financial performers and have a five times greater probability of making faster decisions.<sup>3</sup>

Driven by off-the-shelf hardware, open source software, and distributed compute and storage, Apache Hadoop®-based data warehousing solutions augment traditional enterprise data warehouses (EDWs) for processing large volumes of unstructured data independently, and provide the ability to combine data from a wide variety of sources. The lure of gaining comprehensive insights factoring in contextual, historical, and operational data leads many enterprises to launch big data proofs of concept (PoCs) and pilot projects.

## Table of Contents

<b>Executive Summary</b> .....	<b>1</b>
<b>Introduction</b> .....	<b>1</b>
<b>Solution Overview</b> .....	<b>3</b>
Server Hardware .....	3
Better Together With Intel® Architecture .....	4
Analytics Framework .....	5
<b>Solution Components and Design Planning</b> .....	<b>5</b>
General Considerations .....	5
SLA Requirements .....	6
Data Characteristics .....	6
Storage .....	7
Compute .....	8
Memory .....	8
Hardware and Software Accelerators .....	8
Network .....	9
Software Stack .....	9
Analytics Framework Stack .....	9
Performance .....	10
Scalability .....	11
Total Cost of Ownership .....	11
<b>Optimizations and Parameter Settings</b> .....	<b>11</b>
<b>Workload</b> .....	<b>12</b>
Introduction to TPC Express Big Bench* .....	12
Data Model .....	12
Workload Flow .....	13
Workload Kit .....	14
Quick Steps to Run the Workload .....	14
Setup and Configuration .....	15
<b>Results</b> .....	<b>15</b>
<b>Architecture and Node Configuration</b> .....	<b>16</b>
<b>Design Considerations</b> .....	<b>16</b>
<b>Summary</b> .....	<b>16</b>
<b>Appendices</b> .....	<b>17</b>

However, integrating and operationalizing big data systems can be challenging. A Gartner survey shows that most enterprises are stuck in the PoC or pilot stage and are unable to move big data projects into production.<sup>1</sup> Business groups find it difficult to build and deliver applications because the open source ecosystem lacks pre-developed solutions. IT organizations are challenged by getting solutions running reliably and effectively integrating them into the existing environment. In many cases, enterprises that are unfamiliar with scale-out big data computing have used simple Hadoop workloads such as Terasort\* to benchmark their pilot environment. These simple workloads fail to portray the full capabilities of big data—resulting in either:

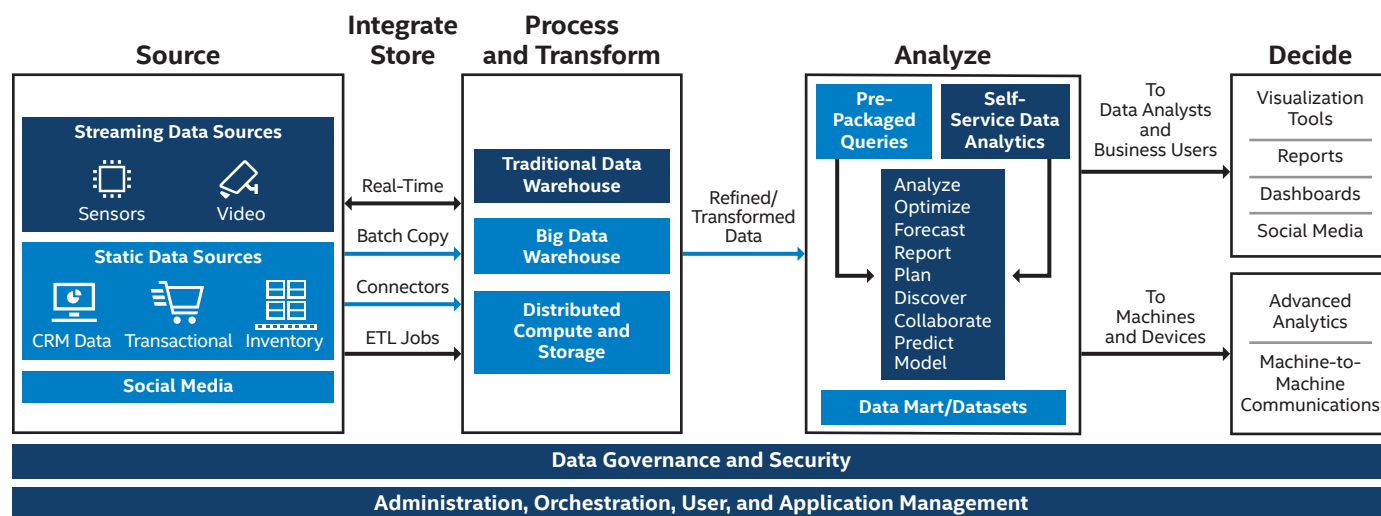
- Overbuilt, underutilized infrastructures that increase total cost of ownership (TCO).
- Underbuilt infrastructures that fail to meet service-level agreements (SLAs) and result in failed business opportunities. In some cases, enterprises try to fix these problems by scaling the nodes in an ad-hoc manner; but cluster sprawl produces both capital and operational expenditure inefficiencies.

To avoid these pitfalls, enterprises must:

- Recognize the complexity inherent in big data systems.
- Carefully plan use cases and data characteristics.
- Implement a substantial amount of optimization and tuning of both the hardware and the software stacks.
- Develop a precise understanding of workload behavior through the software stack and on the underlying hardware, factoring in dozens of variables.

Success depends on two foundational concepts:

- **Use a representative workload.** Unlike simple workloads such as Terasort, a representative workload exercises all solution components, providing a comprehensive understanding of functionality (see Figure 1), performance, and TCO.



**Figure 1.** A big data analytics batch processing solution encompasses many tasks, from data ingestion to reporting. The big data warehouse is the heart of the solution.

- **Take advantage of the latest technology.** For example, HPE ProLiant\* DL Gen10 servers equipped with Intel® Xeon® Scalable processors provide a 1.8x increase in performance and a 1.5x decrease in total cost of ownership (TCO), compared to previous-generation processors.<sup>2</sup>

## Solution Overview

The primary purpose of this reference solution is to demonstrate how to ingest, store, and load high-volume structured and unstructured data extracted from various sources such as transactional relational database management systems (RDBMS), operations data, web logs, click streams, and other external sources. Once ingested, the data is cleaned and formatted with metadata and schema. Next, the data is loaded into the analytical data warehouse with shared local storage applied with pre-defined use case logic for sorting and combining functions on distributed computing nodes. Finally, results in the form of compressed datasets are made available for business needs for consumption to run reporting, machine-learning models, and business intelligence (BI). In addition, the solution is flexible enough to support ad-hoc analysis of data when there are no pre-defined use cases.

The solution is built for horizontal applicability across all verticals. Primary use cases include customer, fraud, and cybersecurity analytics; targeted marketing; advertising campaigns; and supply chain optimization. Other use cases are possible, such as Internet of Things (IoT) and manufacturing, geo-spatial analysis, video surveillance, and click optimizations. Overall, this reference solution is an effective big data extension to an EDW analytics platform that offers the following:

- Scalability and flexibility
- Optimum performance and TCO
- Ability to satisfy business requirements for SLAs, multiple users, and future growth

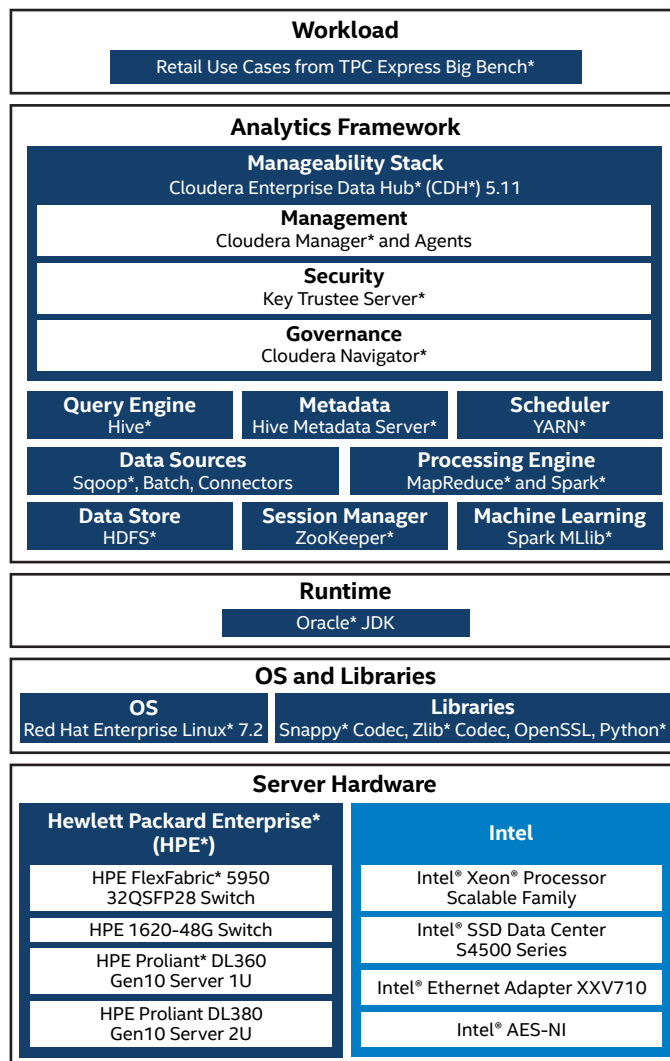
The solution can be applied to both green-field and refresh deployment scenarios, and is designed to be used to its fullest extent before scaling out by adding more nodes. The solution differentiates itself in the following ways:

- In-depth insights are derived from experiments running a workload that mimics real-world application development.
- End-to-end components are provided by established vendors with the necessary critical support structure already in place.

As shown in Figure 2, the solution consists of four major components: server hardware, OS and necessary libraries, analytics framework, and use case workload applications.

The following sections provide more information about the server hardware and analytics framework components. Table 1 provides a complete bill of materials.

## Reference Solution Overview



**Figure 2.** Four major components are required for a big data batch processing solution—server hardware, OS and libraries, an analytics framework, and the workload application.

### Server Hardware

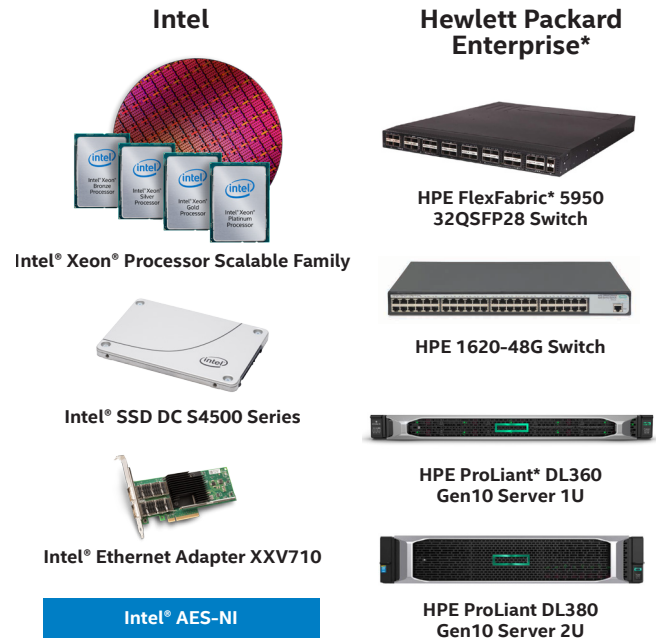
The solution is based on ProLiant DL360 Gen10 and DL380 Gen10 servers. These servers offer the latest in security, performance, and expandability. In networking, the solution implements the HPE FlexFabric\* 5950 32QSFP28 switch. This switch provides IRF bonding and sFlow\* for simplified management, monitoring, and resiliency of the Hadoop network. In addition, the solution provides an external HPE 1620-48G switch for HPE iLO\* system management.

The entire solution consists of one master node, two management nodes (which serve as failover for the master node), and 18 worker nodes, for a total of 21 nodes.

## Better Together With Intel® Architecture

Intel offers key ingredients to build a best-in-class big data analytics platform. These synergistic Intel® architecture ingredients range from the processor to acceleration libraries for building a fast, efficient, end-to-end batch processing solution that is optimized to achieve the best possible functionality, performance, power consumption, and TCO.

- **Intel® Xeon® processor Scalable family** offers unparalleled scale and performance for compute, storage, memory, network, and security.
- **Intel® Ethernet Network Adapter XXV710 (25 GbE)** accelerates the delivery of new services and capabilities through intelligent offloads, sophisticated packet processing, and quality open source drivers.
- **Intel® Solid State Drive (SSD) Data Center (DC) S4500 Series** provides highly reliable storage coupled with high performance for read-intensive applications and low energy consumption.
- **Intel® Advanced Encryption Standard - New Instructions (Intel® AES-NI)** protects data with accelerated data encryption.



**Table 1. Reference Solution Hardware Bill of Materials**

	Description	Quantity
<b>Management Nodes Server System: HPE ProLiant* DL360 Gen10 (2x)</b>		
<b>CPU</b>	Intel® Xeon® Silver 4116 processor, 2 sockets, 2.10 GHz, 48 threads	4 (2 CPUs per node)
<b>Memory</b>	32 GB DIMM DDR4	12 (6 x 32 GB DIMMs per node)
<b>OS Disk(s)</b>	Intel® Solid State Drive (SSD) Data Center Family for SATA 480 GB, Part number: 877746-B21	2 (1 SSD per node)
<b>Fabric Interconnect</b>	HPE* Ethernet 25 GB	2 (1 HPE Ethernet 25 GB adapter per node, uses Intel® network technology)
<b>Master Nodes Server System: HPE ProLiant DL360 Gen10 (1x)</b>		
<b>CPU</b>	Intel Xeon Silver 4116 processor, 2 sockets, 2.10 GHz, 48 threads	2 (2 CPUs per node)
<b>Memory</b>	32 GB DIMM DDR4	6 (6 x 32 GB DIMMs per node)
<b>OS Disk(s)</b>	Intel SSD Data Center Family for SATA 480 GB, Part number: 877746-B21	1 (1 SSD per node)
<b>Fabric Interconnect</b>	HPE Ethernet 25 GB	1 (1 HPE Ethernet 25 GB adapter per node, uses Intel network technology)
<b>Worker Nodes Server System: HPE ProLiant DL380 Gen10 (18x)</b>		
<b>CPU</b>	Intel Xeon Gold 6154 processor, 2 sockets, 3.0 GHz, 72 threads	36 (2 CPUs per node)
<b>Memory</b>	384 GB DIMM DDR4	216 (12 x 32 GB = 384 GB per node)
<b>OS Disk(s)</b>	Intel SSD Data Center Family for SATA 480 GB, Part number: 877746-B21	18 (1 SSD per node)
<b>Data SSD Disk(s)</b>	Intel SSD Data Center Family for SATA 960 GB, Part number: 877752-B21	18 (1 SSD per node)
<b>Data HDD Disk(s)</b>	HDD 600 GB SAS	432 (24 HDDs per node)
<b>Fabric Interconnect</b>	HPE Ethernet 25 GB	18 (1 HPE Ethernet 25 GB adapter per node, uses Intel network technology)
<b>Networking (All Nodes)</b>		
<b>Fabric Switch</b>	HPE FlexFabric* 5950 32QSFP28 Switch	3 Rack Switches (HPE-branded switches using Intel® Network Interface Card)
<b>External Switch</b>	HPE 1620-48G Switch	1 Rack Switch
<b>Rack</b>	HPE 42U 600 mm x 1075 mm G2 Kitted Advanced Pallet Rack	1 Server Rack

## Analytics Framework

Cloudera Enterprise Data Hub\* (CDH\*) enables deep insights on a single, integrated platform. With powerful open source tools and an active data optimization designed specifically for Hadoop, enterprises can move from big data to results quickly. Solution architects can take advantage of Cloudera features, such as Cloudera Manager\*, Key Trustee Server\*, and Cloudera Navigator\* (see Figure 3).

## Solution Components and Design Planning

Modern Hadoop-based big data systems are designed to run on commodity hardware that uses a fault-tolerant software architecture that will scale out easily when a cluster resource bottleneck occurs. However, it is not easy to create a perfect deployment and keep up with the growth in data volume and emerging use cases. Time spent in planning and on the pilot phase can pay rich long-term dividends. This section offers some general guidance to adapt this reference solution to a specific enterprise environment.

When designing the solution, these best practices can help enterprises achieve the following:

- Avoid mistakes that can significantly impact the success or failure of a big data integration.
- Keep the big data project on track from pilot to production.
- Enable the enterprise to realize optimal business value from the chosen big data solution.

The hardware configuration recommendations provided in this section are based on insights into the observed behavior of the workload pattern. Discussion is also included about several other considerations (such as data type, quality, and volume; security; and governance). Note that this is not an exhaustive list, but is an excellent starting point.

## General Considerations

Designing an effective big data analytics solution requires attention to several planning concepts, including how the solution will be used and by whom, and how data will be ingested and consumed.

### Use Case Definition

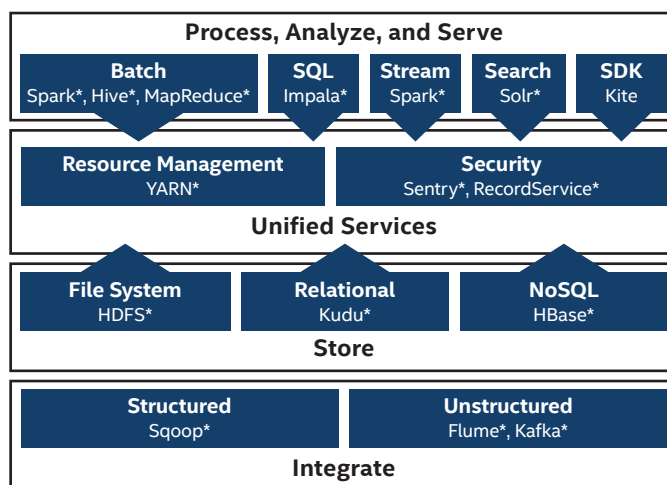
As mentioned in the [Introduction](#), a big data analytics system can be applied to a wide variety of use cases. Therefore, the first step when designing a solution is to identify the use cases that support the business objectives and meet the requirements demanded by the stakeholders. That is, start with business outcomes, then define the solution around those desired outcomes. In this reference solution we implement a set of retail use cases that are defined in the [Workload](#) section.

### Deployment Model

It is possible to deploy this reference solution in one of two ways:

- **Standalone.** The implementation is specifically built with none to minimum integration touchpoints. Typically, this approach is easy to build and maintain and is not intended to augment existing EDWs.

## Cloudera Enterprise Data Hub\*



**Figure 3.** Cloudera Enterprise Data Hub\* (CDH\*) provides a flexible, powerful analytics framework.

- **Integrated.** The implementation is primarily built to augment an existing data warehouse to provide new capabilities or shift workloads from traditional EDWs to a big data solution. This approach is the most common in an enterprise big data solution buildout. Solution complexity can be higher due to multiple integration touchpoints and requires detailed pre-deployment planning and a rigorous pilot phase.

### Primary Usage

Primary usage drives the technical complexity of the solution. Therefore, it is important to identify the technologies required to implement the solution based on targeted use cases and the deployment model. Knowing what the big data analytics warehouse is capable of (and what it cannot do) at the beginning helps prevent issues during the pilot and production deployment stages.

- **Business users** use the solution to extract insights. These users are not technically savvy and expect the solution to provide easy-to-use and ready-to-run applications for BI, reporting, and queries. Because the usage scenarios are known in advance, it is relatively easy to anticipate what to expect and allow ample time for course correction if necessary.
- **Advanced users and machines** need advanced options to interact with the solution so they can experiment and explore to create custom applications. Users must also be able to choose to use all data or subsets of the data for secondary solutions. This type of agility requires additional planning and pilot phases to verify that most scenarios are accommodated. A plan for immediate remedial actions in case of an issue is also necessary.



## Data Access

Two phases of data access must be planned for:

- **Ingestion.** Data is accumulated over time during business operations and pulled into the analytics warehouse using database connectors such as Sqoop\*, Kafka\*, Flume\*, and JDBC\*/ODBC. In some cases, the data can also be copied as a batch upload or streamed over the network. Sometimes, pre-ingestion processing is necessary at the source to reduce the incoming data to only the relevant parts. The typical enterprise can expect to have an average of five independent data sources feeding into the big data analytics warehouse solution.
- **Consumption.** Output data is enriched with business logic and combined from various sources. It can be shared and consumed in multiple modes. Typical consumption models include machine-learning algorithms, analytical applications, and data marts where businesses run pre-packaged BI queries or users run self-service ad-hoc queries. There are multiple options for staging the data for consumption. Data can be shared locally on the cluster or staged outside the solution—the appropriate choice depends on how the final output will be used, SLAs, and hardware resource utilization. For example, machine-learning algorithms can be run on the local cluster for training, where the availability of large datasets can help improve model accuracy and computing is shared among multiple nodes with node-local data speeding up training. For reporting and BI consumption, processed data can be moved to traditional systems as well as scale-out in-memory systems such as SAP HANA\*, Oracle Exadata\*, and Impala\* for fast query response time.

## SLA Requirements

This reference solution is targeted to support multiple users running parallel jobs with varying characteristics. SLA requirements depend on several factors, including the following:

- **Time to complete.** The reference solution is specifically designed to provide best-in-class time-to-insight for high-demand deployments, where the cluster is in multi-use mode with many parallel jobs being run by a variety of users. This deployment is targeted for highly active use cases, where the cluster is used virtually around the clock and is driven by strict SLAs that demand fast job completion and little tolerance for failure. (Enterprises with less demanding requirements can refer to Table 2 in the [Architecture and Node Configuration Design Considerations](#) section to adapt this reference solution to their big data analytics environment requirements, thereby improving efficiency.)
- **Job type.** In batch processing, compute utilization is proportional to the amount of data the job must process. It is expected that the cluster will handle multiple job types, where a few jobs can run for extended duration (hours or even days). These jobs must be given highest priority in allocating resources to protect against catastrophic failures. If the cluster is at peak utilization and is unable to support other jobs due to lack of resources, solution architects can optimize the query plan or change the job schedule so the

cluster is relatively free. More nodes can be added, but only after all other options are exhausted; scaling nodes in smaller increments is a good option. For example, add only three additional worker nodes that mirror the configuration of existing worker nodes.

- **Users.** While pre-packaged BI queries are useful, self-service analytics can provide significant business benefits. “Self-service analytics technology provides the features and functions relevant to and demanded by the knowledge worker,” says Brian McDonough, research manager, Business Analytics Solutions, IDC. “We have seen impressive growth in the adoption of these solutions — leading to organizations generating and deriving new business benefits as users make new insights, faster.”<sup>4</sup> To best support self-service analytics, where users are creating ad-hoc jobs that may not have been gone through the develop, test, and deploy lifecycle, solution architects should identify the number of users and the job profiles that the cluster should support and anticipate un-optimized code that may require allocating additional resources.

## Data Characteristics

The primary objective of a big data analytical warehouse based on Hadoop is to enhance structured data from traditional sources with unstructured data to provide deeper insights. Unstructured data can add a depth to data analysis that could not be achieved otherwise. Solution architects should address the following data characteristics:

- **Variety.** Structured data typically accounts for only 20 percent of all data available.<sup>5</sup> It is clean, analytical, and usually stored in databases. Databases for structured data have limited support for unformatted data such as documents, files, and video, and yet semi-structured and unstructured data comprise 80 percent of an enterprise's data volume. Because it does not follow the formal structure of data models, unstructured data is not useful when fitted to a schema or table, unless specialized techniques are used to analyze some of the data and then store it in a columnar format. The workload used in this reference solution has all three types of data.
- **Volume.** In a typical big data warehouse, just a few tables are used to store a large volume of data. For example, in the workload used in this reference solution, the four largest tables contain 90 percent of the data. It is important to understand that all data does not scale linearly. For instance, data concerning demographics and income are a fixed size, normally obtained from transactional sources, whereas product reviews and click streams can grow substantially to petabyte scale and beyond. Considering the data volume characteristics when creating the data model and application logic is extremely important because the data volume determines the performance and SLA of a big data analytics warehouse.
- **Velocity.** Data is integrated into a big data warehouse at varying speeds. Unless it is a real-time solution, initial load is done using batch processing, where custom extract, transform, and load (ETL or ELT) jobs transfer data from

multiple sources into the big data system. Subsets of incremental data are updated at pre-determined time intervals to bring in fresh data—this is referred to as a trickle update. How much velocity is needed determines the choice of software and hardware, and can dramatically alter the solution setup. For example, implementing an IoT use case may require faster data stores such as key-value (KV) stores, with a low-latency network and a large amount of DRAM. Experiments with the reference solution workload have shown that failing to understand the required velocity and attempting to drive a big data cluster with incorrect assumptions about multi-tenancy can lead to serious SLA failure.

- **Data governance and security.** It is necessary to have policy-based security enforcement when higher-level APIs such as SQL\* are used. It is important to reduce data management complexity that can negatively impact security as the cluster footprint grows over time. The reference solution uses Cloudera Navigator, Key Trustee Server, and an Apache Sentry\* service for governance and security. These powerful tools, along with an intuitive graphical user interface, define the auditing, access control, data lineage, and metadata policies.

Metadata attributes of unstructured data must be managed from the beginning, because unlike structured data there is no finite set of definitions and no concept of primary keys to govern the data. Unstructured data can vary in type as well as physical and logical formats. In addition, duplication of fields can occur. These scenarios can present a significant challenge if they are not addressed during initial planning. Issues such as data skew (a condition in which a table's data is unevenly balanced among partitions in the cluster) can negatively impact resource utilization, driving down efficiency of the cluster. Managing metadata can create clearly identifiable attributes that can be used to enforce governance rules, preserving the data's integrity, quality, and resiliency to mistakes and overwrites. Data models and schema layouts should factor in asset type, audit logs, business-specific attributes, and data lineage. Solution architects should test the developed data models on multiple nodes and make sure the PoC setup can handle large datasets.

- **Data format.** Binary file formats provide the best performance compared to running jobs on raw data, but there are additional steps involved to convert raw data to binary format. The raw data is read from the Hadoop Distributed File System\* (HDFS\*), encoded, then compressed for efficiency. Because the data is highly compressed, these file formats reduce not only disk I/O, but also the storage footprint and, hence, the cost. The resulting values are stored in contiguous memory locations in a columnar format. It is acceptable to add a conversion stage, since the time spent in conversion will reduce the elapsed time by several orders of magnitude. Solution architects can choose between multiple columnar formats, such as Optimized Row Columnar\* (ORC\*), Parquet\*, RCFile\*, and Avro\*. The encoding choice depends on the software stack. Consider a tiered HDFS architecture, where the columnar format database can be stored on an Intel SSD with NVMe\*—this is an excellent option to boost the performance without a significant cost increase.

## Storage

Because of data volume and data velocity, storage is a crucial aspect of a big data analytics warehouse solution. Considerations include the following:

- **Capacity.** Plan for adequate capacity, based on estimates of initial data volume, future data growth, and necessary data replication. Capacity should not be considered without also factoring in storage tiers, performance, and implications of fault tolerance. For an active analytical warehouse deployment similar to the one discussed in this reference solution, solution architects should focus on the SLA and the performance required to meet the SLA. For this reference solution, each node uses medium-density media to harness the data locality characteristics of Hadoop to distribute the jobs on multiple nodes. Tiered storage can help increase performance by staging the data between archive (cold storage), active working dataset (warm storage), and consumption dataset (hot storage).
- **Replication.** By enabling an HDFS replication factor of three, we take advantage of the fault tolerance and resiliency offered in HDFS to help prevent data loss in case of a media or node failure. For additional cost savings, solution architects may choose to use SATA 7.2K RPM hard disk drives (HDDs), which offer higher capacity and similar throughput to highly reliable SAS-based hard disks; however, disk failure rates may be higher.
- **Media Type.** It is important to choose the media type appropriate for the usage:
  - **OS and application storage.** The reference solution uses the Intel SSD DC S4500 Series (240 GB) across all nodes to install the OS and application stack. RAID 1 (disk mirroring) is used on management and master nodes to prevent single point of failure in these nodes (such a failure could bring down the entire cluster). Because HDFS offers built-in redundancy, where a node failure is contained with minimal impact, the reference solution worker nodes use a single Intel SSD DC S4500 Series (240 GB). Using spinning media (HDDs) is another possibility if it provides a cost advantage. It is important to monitor the log file growth to verify that storage does not run out of disk space.
  - **Metadata storage.** Essential services run on management and master nodes, such as Cloudera Management Services\*, NameNodes, ResourceManager\*, and ZooKeeper\*. These services store management databases, metadata, journaling blocks, and job tracking information. It is important that these services are housed on an effective, high-performance, and fail-proof storage setup. The reference solution uses a RAID 5 array, consisting of the Intel SSD DC S4500 Series, to store data from the above mentioned services, which contributes to seamless operation of the cluster and minimizes catastrophic failures. Using HDDs is a possibility, although careful consideration must be given to performance degradation as the deployment accumulates more data. Job logs, characteristics, metrics, and counters must be monitored and adjusted to the appropriate level (such as INFO, ERROR, or DEBUG) to minimize the risk of running out of disk space.

- **Capacity storage.** Worker nodes offer primary storage for the cluster, using HDFS. The reference solution uses a total of 24 600-GB 10k RPM SAS HDDs in JBOD mode on each worker node with a raw capacity of 14 TB per worker node. Another advantage of using SAS HDDs is that temporary shuffle I/O files share the HDD spindles with the capacity data store.
- **Intermediate I/O storage.** During job execution, shuffle generates random read/write I/O files which are temporary files that are deleted once the job completes successfully. The reference solution uses low-latency I/O access provided by NAND, using a single Intel SSD DC S4500 Series (960 GB) to absorb the intermediate I/O patterns, which results in an approximate 10 percent to 15 percent performance gain. When opting to accelerate shuffle I/O using SSDs, consider the total life-span of your solution to ensure the SSD endurance is sufficient to handle the expected writes to the SSD. This helps avoid running out of NAND life and replacing the SSD before the cluster's refresh cycle is complete. The capacity of SSDs must be determined with a full understanding of the application pattern. During the PoC stage, job counters can assist in understanding how much random shuffle I/O was generated for a given job. Adequate overhead is needed to absorb additional growth in applications and monitor the usage to avoid job failures when the SSD runs out of space (which can impact the SLA).

## Compute

The correct processor depends on the usage, as described here:

- **Master, management, and gateway nodes.** Management and master nodes can be sized appropriately to provide sufficient computing power to support worker client requests, orchestration and reporting, job and events monitoring, and service analysis. Some jobs, such as spark-submit, may use significant local resources on the gateway node. Solution architects should check for such jobs during the PoC phase and adjust the gateway node configuration as necessary.
- **Worker nodes.** Research from profiling data analytics workloads suggests that taking advantage of recent advancements in computing frameworks, as well as systems with large memory capacity, high-bandwidth network, and SSDs can enhance the performance of workloads.<sup>6</sup> The traditional means of using the core-to-disk ratio to determine compute requirements is insufficient for planning for big data compute capacity. This reference solution deployment considers the utilization pattern of a running job, user profile, job parallelism, time period when the cluster is busy, and relevant SLAs to determine compute requirements. The Intel® Xeon® Gold 6154 processor used in this reference solution offers the following advantages:
  - **High base clock frequency** helps meet the strict time to completion SLA. Also note that frequency scaling for the reference solution workload is more effective than core scaling.
  - **Moderate core count** means that each node with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled provides 36 threads per worker node. With effective DRAM sizing, each thread can be configured as a YARN\* container to support multi-user and parallel jobs.
  - **Fast vector processing** on the Intel Xeon Gold 6154 processor supports two units of Intel® Advanced Vector Extensions 512 (Intel® AVX-512), important to use cases such as machine learning and data management running on Intel AVX-512-enabled software.

Note that compression and decompression of data will impact compute utilization significantly, depending on the compression codec used. In our experience, Spark\*-based engines utilize compute resources better than MapReduce\*.

## Memory

Batch jobs are not generally bound by memory bandwidth, and exhibit moderate to occasional peak bandwidth behavior. This reference solution uses 384 GB of DDR4 per worker node. However, populating DRAM according to the [HPE population guidelines](#) will help utilize all six memory channels without any adverse bottlenecks. Each worker node has sufficient DRAM capacity to size containers to support all usable threads in the worker node when Intel HT Technology is enabled. When using Spark, we recommend increasing the memory capacity to 768 GB per node, which reduces the pressure on the storage subsystem.

## Hardware and Software Accelerators

Hardware and software accelerators can help optimize solution performance:

- **Compression codecs.** The reference solution uses compression for both intermediate shuffle data generated during job execution and for output of the job written back to HDFS. Native compression codecs have superior performance in compressing and decompressing the data compared to codecs implemented in Java\*. The Snappy\* codec can be used for both intermediate and job output purposes; however, using Snappy comes with a tradeoff—high-compression performance that consumes minimal compute cycles but provides low reduction in dataset size. Native implementations of other codecs, such as zlib\* and gzip\* are supported, which provide higher compaction of data at the expense of more compute cycles. Solution architects can decide which aspect—data compression or compute cycles—is most important. For example, if a deployment's data capacity is in jeopardy but has compute cycles to spare in the cluster, use codecs with lower compression ratios. If storage is abundant, use codecs with higher compression ratios and lower compute overhead.
- **Intel® Advanced Encryption Standard – New Instructions (Intel® AES-NI)** capabilities must be utilized for accelerating encryption. Working with OpenSSL\* cryptographic libraries, Intel AES-NI can reduce compute overhead when encryption



is enabled. The entire dataset at rest in the cluster can be encrypted with nearly zero compute overhead, freeing compute cycles for user space applications. For high-security environments, end-to-end cluster encryption can be enabled using Cloudera Key Trustee Server, which provides enterprise-grade key management for compliance and information security such as during data transfer; rpc calls between nodes; and communication between management, master, and worker nodes. However, care must be taken to avoid adverse cluster performance impacts, because `%systemtime` can steal compute cycles from user space applications. Intel AES-NI is readily available by enabling it in the BIOS; pair it with OpenSSL libraries (which are pre-installed but may require an [update or upgrade](#) to support a specific feature).

## Network

The reference solution uses the HPE Ethernet 25 GB Adapter (Intel Ethernet Network Adapter XXV710). Each worker node is capable of about 2 GB/s disk throughput; during peak disk transfers it is possible that a network bottleneck can slow down jobs. If the chosen use case does not require high network traffic, a 10 GbE network adapter can be used.

## Software Stack

This reference solution uses the following basic software:

- **Red Hat Linux\* (RHEL\*) 7.3** is a 64-bit OS with enterprise-grade support. It is possible to use a different OS as long as HPE ProLiant DL Gen10 hardware and CDH software are fully supported. In addition, free options such as CentOS\* and Fedora\* can be used if enterprise support is not a requirement. CDH does support selective versions of free operating systems; consult Cloudera documentation for compatibility.
- The management and support stack includes the following:
  - **Java runtime.** This reference solution uses Oracle JDK\* 1.8 for Java runtime. Refer to the Cloudera [documentation](#) for information about replacing the default JDK installed with the OS or the JDK installed during CDH installation. If replacing the JDK, consult the CDH [documentation](#) on supported JDK versions.
  - **Python\*.** Python is a popular language for writing big data programs and is widely used in data science modeling. In this reference solution, there are a few procedural job use cases written in the Python language. Typically, Python runtime is installed by default during OS installation.
  - **Cloudera management stack.** Cloudera Manager is an end-to-end application for managing CDH clusters installed on one server; Cloudera Manager Agents run on all the nodes in the cluster.
  - **Cloudera Navigator.** Cloudera Navigator is a fully integrated data governance solution for Hadoop, offering critical capabilities such as data discovery, continuous optimization, audit, lineage, metadata management, and policy enforcement. As part of CDH, Cloudera Navigator is critical to enabling high-performance agile analytics, supporting continuous data architecture optimization, and meeting regulatory compliance requirements.

## Analytics Framework Stack

The analytics framework stack includes fundamental tools, computing engines, storage engines, and machine-learning algorithms.

### Fundamentals

The following open source tools provide basic services:

- **YARN.** This scheduler is responsible for allocating resources. Resource Manager is the centralized service that manages resources among all the applications in the system in the form of containers and is enabled on a single node. Worker nodes run the YARN Node Manager\*, which is responsible for containers, monitoring their resource usage (CPU, memory, disk, and network) and reporting this information to the Resource Manager so capacities and queues are optimized.
- **HDFS.** This file system is the primary distributed storage used by Hadoop applications. An HDFS cluster consists of two NameNodes: a Primary NameNode, which manages the file system metadata by keeping the directory tree of all files in the file system and tracks where across the cluster the file data is kept, and a secondary NameNode to provide redundancy and high availability. Worker nodes run DataNode services to store the actual data by applying a logical file system over the storage media on the worker node.
- **ZooKeeper.** This distributed coordination service for big data applications helps with synchronization, configuration maintenance, and naming. In this reference solution, ZooKeeper is enabled on three servers with dedicated storage space to store and synchronize their quorum.

### Computing Engines

This reference solution supports two computing engines:

- **MapReduce v2 (MRv2) distributed engine.** This distributed engine is used to share the work between cluster nodes and is based on the map and reduce programming model. MRv2 is used in this reference solution as the underlying engine for all data-generation stages and SQL stages in the use cases implemented in the workload.
- **Spark distributed engine.** In the reference solution tests, we found Spark to be superior to MRv2 in both performance and efficiency; therefore, the reference solution uses Spark by default for the machine-learning stages in five use cases and all worker nodes are enabled with Spark roles. We recommend that enterprises who are currently using MRv2 as their compute engine consider switching to Spark to take advantage of similar performance gains.

### Storage Engines

Big data analytics warehouse solutions are implemented on non-relational storage engines. There are multiple options to choose from: KV stores, document stores, columnar stores, and graph and object stores. Use case, SLA, and data type and format determine the choice of engine. Non-relational storage engines have several advantages such as on-the-fly attribute edits; this flexibility makes post-deployment changes easy to implement. SQL-on-Hadoop frameworks are popular and have contributed to a reduction in application development complexity, making a big data analytical warehouse easier to use.

Because the reference solution is targeted for large batch processing where throughput of the job is emphasized and there are no strict requirements for sub-second reads and writes, the reference solution uses Hive\*, a popular SQL-on-Hadoop framework that offers data warehouse capabilities to manage large datasets residing in HDFS. Hive supports a SQL-like language called HiveQL\*, enabling a broad range of useful functions that can be expressed in SQL. Hive SQL queries are executed in the form of jobs on the cluster using MRv2 or Spark. They can process large amounts of data and support flexibility to write user-defined table functions (UDTFs) and user-defined aggregate functions (UDAFs) that use functions, operators, and procedural jobs spawned from the parent query. SQL language expressions are universally well known and users appreciate the familiarity, ease of use, and seamless integration with traditional BI tools. However, SQL-on-Hadoop frameworks such as Hive are not fully ANSI SQL compatible, so careful evaluation is required if full ANSI compatibility is required.

### Machine Learning

For the reference solution workload use cases that require machine-learning stages, the solution uses MLlib\*, Spark's scalable machine-learning library (installed with Spark). This library offers a suite of distributed machine-learning algorithms that can be invoked easily with simple calls within the code. MLlib is used by the reference solution workload use cases that involve logistical regression, k-means clustering, and classifier algorithms. Other machine-learning implementations, such as R\* and SAS\* can be used if the stage dataset is small or the users are familiar with the tools.

### Performance

The performance requirements of a solution are primarily defined by SLAs. Dividing the data pipeline into stages to gather influential factors affecting the performance is good for defining the performance requirements. Concurrent users or jobs have unique cluster utilization patterns. It is important to identify and understand these patterns to future-proof the solution and optimize future business opportunities. In general, out-of-the-box performance is usually poor; cross-solution optimizations are mandatory to improve performance (see the [Optimizations and Parameter Settings](#) section).

The following list describes some of the optimizations that may be necessary at various stages in the data pipeline:

- **Ingestion.** If the cluster is actively processing jobs while also ingesting data from multiple sources, ample network bandwidth must be available. 10 GbE is most commonly used, but 25 GbE is gaining traction in the industry and is used in this reference solution. If there is a requirement for low-latency ingestion (which is not true for this reference solution), network quality of service (QoS) should be factored in as well. Consider enabling compression to reduce the amount of data sent over network transfers.
- **Storage.** HDFS is optimized to read and write data in contiguous logical blocks where data is written or read in 64-MB to 256-MB chunks. For optimal performance, the data should fill the entire block. 24 HDDs can support

approximately 2 GB/s overall sequential throughput per node. HDFS is not performant for small files, which do not use entire HDFS blocks; therefore, during the PoC phase, solution architects should identify and eliminate small files if possible. Also, enabling compression results in significant space savings, enabling more data to be stored without expanding the storage footprint.

- **Transformation and processing.** Caution should be exercised when using high-capacity SATA drives (6 TB, for example). During a long job run, when the storage space utilization is peaking close to maximum capacity, a catastrophic failure of an HDD may result in a vast quantity of data being copied due to rebalancing, lost data, and a negative impact on other running jobs' disk and network throughput.
- **Memory management.** Java's write-once-run-everywhere programming mode can sometimes lead to performance challenges. Working around these limitations requires techniques to mitigate unsafe garbage collection and unsafe off-heap computing during the development phase. In deployment, managing large heap sizes requires tuning the garbage-collection process to reduce the long pauses. For the best throughput, we suggest configuring Garbage-First Garbage Collector (G1GC) for master node services and parallel garbage collection for mappers and reducers. A container size of 2 GB will reduce the startup time for Java virtual machines.
- **Consumption.** If the cluster is used for use cases where processed data is consumed by other services such as BI or reporting, which require fast data access, solution architects may consider implementing HDFS data tiers. Using SSDs with HDFS data tiers is not a widely used feature and may require user code changes to work properly, so solution architects should understand HDFS tiers and their implications to avoid adding complexity to the deployment.
- **Profiling the end-to-end solution.** In general, [Amdahl's Law](#) is active in the big data ecosystem. Effective use of hardware profiling tools can provide insight into bottlenecks. The 30 use cases used in the reference solution workload have unique resource utilization patterns for compute, storage, and network. The behavior patterns can be used to understand what is necessary to support all users running jobs with various patterns on the volume of data.

The following key criteria can help keep the optimization exercise time-bound yet effective:

- Verify that the solution can meet the SLA.
- Analyze and profile job logs to identify easily correctable bottlenecks.
- Monitor the basic utilization of the cluster focusing on the following metrics:
  - CPU: user time, system time, and wait times. The [PAT](#) utility is a collection of simple scripts that gather system metrics and create pivot charts for analysis.
  - I/O: disk throughput for sequential I/O and latencies for random I/O.
  - Network: throughput for large data transfer and latencies for small packet transfer.

## Scalability

Perhaps the most quoted advantage of scale-out computing is linear scaling achieved by adding nodes. Effective scalability planning should focus on performance SLA; expected growth in data, users, and use cases; and the addition of new technologies. Identify scaling issues by building a pilot environment that closely resembles the future production environment.

When planning for extension by adding more nodes, make sure the node configuration is similar or equivalent to the initial configuration, which minimizes imbalance issues (especially important when SLAs are strict). For example, by default, scheduler heuristics treat all compute resources equally. Adding a node with a different processor model with a lower frequency compared to the initial node configuration will result in straggler tasks still running on the added node while all other tasks are complete on the initial node—increasing job completion times.

Data skews and stragglers contribute to a large amount of support calls. Data volume and variety combined with the complexity of parallel computing means no matter how carefully the data modeling planning is done, it is inevitable that these issues will arise. If they are not specifically monitored and addressed, scaling the cluster by adding nodes, combined with data volume growth, can cause a cluster that was performing normally to suddenly experience significant performance issues.

## Total Cost of Ownership (TCO)

Using off-the-shelf servers may seem attractive at first, but enterprises must also consider the cost implications of software licenses, support, space, and power and cooling costs. The effort expended to set up clusters, allocate resources, and test in a pilot environment can take months before any real effort is spent to determine business value. Enterprises should also avoid over- and under-provisioning. The former leads to low average utilization, which drives up the TCO; the latter impacts SLAs and triggers expensive expansion of the cluster.

Optimal performance, SLA compliance, and low TCO require a balancing act that demands exploration of all available options and combinations for optimizations and parameter tunings; it is entirely possible to shift the bottleneck from one component to the other or inadvertently increase TCO. For example, enabling compression saves disk space and network traffic but consumes additional compute cycles, which can negatively impact user space applications. Or, adding nodes may alleviate compute bottleneck issues but comes at additional cost for required hardware and software, again driving up the TCO.

## Optimizations and Parameter Settings

Compared to open source projects, software distributors such as Cloudera, Hortonworks\*, MapR\*, and others have made deploying big data solutions easier in last few years by providing an excellent process by which to install, operate, and administer a cluster. They have masked the complexity

of working with configuration settings by providing out-of-the-box optimal settings. Even so, most analytics frameworks require adjusting parameters to achieve the best performance for a specific application, and these changes must be made by technical staff, not end users.

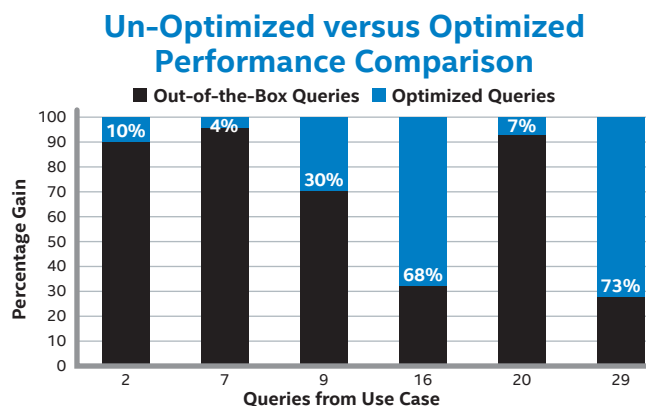
Optimization is a balancing act. It is impossible to try every single combination of hundreds of parameters to find the perfect setting; and yet, certain optimizations can provide significant performance benefits. For this reference solution, we kept these changes minimal by selectively identifying the most important parameters that provide maximum benefits. Specificity is important when setting the parameters. We have factored in the following items to identify and set optimization parameters for this reference solution:

- Dedicated cluster for running batch workloads
- Number of worker nodes in the cluster, compute threads, DRAM capacity, and disk and network throughput
- Target raw dataset size
- Working dataset size after columnar format conversion

Figure 4 shows an example of performance improvement comparing out-of-the-box, un-optimized results with results for the same use cases post-optimization. Optimizations are specific to each use case, because each query or use case has a different pattern. Performance improvements from optimizations can range from 4 percent to 73 percent, depending on the query or use case.

Some of these settings may still work in dissimilar setups—where cluster size, node configuration, and dataset sizes are not the same as for the reference solution—as long as the software stack is similar. Solution architects should exercise caution by reading the description of a parameter and modifying it to determine its effect in their specific environment.

Most parameters can be set using the Cloudera Cluster Manager\* user interface. Navigate to configuration settings for the identified component by using the search function to find the parameter and set the value. Some changes require the cluster to be restarted so that configuration files are refreshed on all nodes.



**Figure 4.** Optimizing software stack parameters can help achieve the best performance for a specific application.

Here are some general categories of parameters:

- **Basic frameworks.** These are elements in the software stack that provide fundamental capabilities. This reference solution includes YARN, HDFS, MapReduce, and Spark. Optimizations are generally applied at the global level and will impact the entire cluster and applications served by the cluster. For example, changes made to YARN will affect any jobs scheduled by YARN.
- **Application frameworks and APIs.** These are elements that provide abstraction layers that enable users to easily develop and deploy their workloads. In this reference solution, these include Hive and Spark MLlib. Optimizations are applied semi-globally, targeting only the specific API or framework. Impact of the changes is limited to only the applications using the given API. For example, settings applied to Hive will not impact user-written Java or Scala\* applications. These parameters are described in [Appendix B](#).
- **Workload-specific.** Workload-specific parameters (see [Appendix C](#)) are tailored to provide the best performance for a specific job or part of a job. This type of optimization is primarily passed during runtime and will be removed once the job completes. For example, parameters set individually for Query 1 will not impact Query 2.

Some of the global and semi-global parameters can be overridden by passing runtime parameters. While effective, this requires the testing team to closely coordinate with the DevOps team to ensure correctness and potential adverse impact on other applications. The problem can be compounded in a high-use cluster that supports multiple users.

Despite best efforts, it is not always possible to identify all issues. The dynamic nature of the Apache software ecosystem means there will be unidentified bugs and new bugs introduced with updates. Solution architects may need to address issues as they crop up later in the lifetime of the deployment. A separate sandbox environment can be maintained to closely mimic the production environment, in which to reproduce and fix the issues.

## Workload

As explained earlier, insights derived from overly simple workloads such as Terasort do not reveal the complexity of a fully functional application with multiple functions, stages, data schema, and data types. This section describes the reference solution's workload, including data model, workload flow, and a quick-start guide for running the workload.

### Introduction to TPC Express Big Bench\* (TPCx-BB\*)

This reference solution's capabilities have been tested using a representative workload based on TPC Express Big Bench\* (TPCx-BB\*), an industry-standard big data batch analytics benchmark widely used by industry and academia. The benchmark models various aspects of a commercial decision support system for a retail business. In [Appendix A](#), there are 30 use cases implemented in the form of queries of structured, semi-structured, and unstructured data.

The workload's 30 use cases simulate big data processing, analytics, and reporting. These use cases are frequently performed by big data operations at retailers with both a physical and online store presence. Historically, online retailers recorded only completed transactions, whereas today they demand much deeper insight into online consumer behavior. Simple shopping basket analysis techniques have been replaced by detailed behavior modeling; new forms of analysis have resulted in an explosion of big data analytics systems. Yet until now, there have been no mechanisms to compare disparate solutions in a real-world scenario such as this.

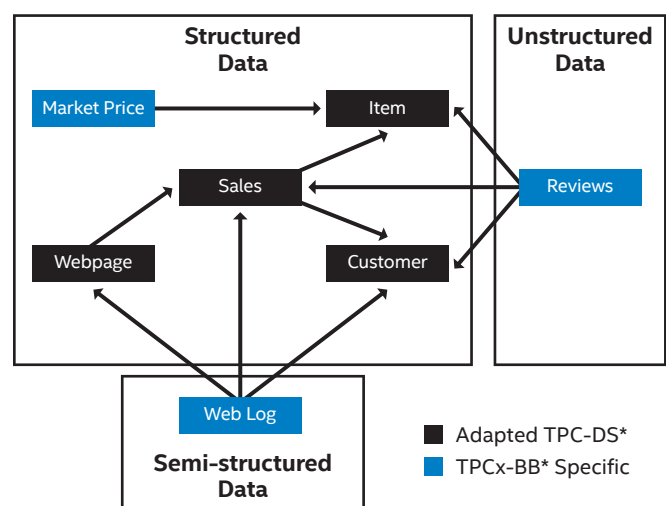
Note: Although the use case is based on the retail vertical, applicability to other verticals can be drawn as long as similar software frameworks and implementation of user code requirements are met while factoring in data schema, data quality, and data types accordingly. Figure 5 shows the simplified schema of TPCx-BB.

### Data Model

TPCx-BB is designed with a multiple-snowflake schema inspired by TPC Decision Support\* (TPC-DS\*) using a retail model consisting of five fact tables, representing three sales channels—store sales, catalog sales, and online sales—each with a sales and a returns fact table. Specific big data dimensions were added for the Big Bench data model. The market price is a traditional relational table storing competitors' prices.

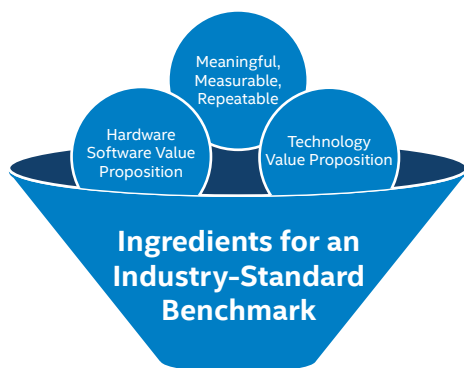
TPCx-BB includes use cases based on the TPC-DS benchmark dealing with structured data, and adds use cases to address semi-structured and unstructured data for in-store and Web sales channels.

### TPC Express Big Bench\* Schema



**Figure 5.** This simplified schema shows how structured and unstructured data are handled by the TPC Express Big Bench\* (TPCx-BB\*) benchmark.





The semi-structured data is generated to represent the user clicks from a retailer's website to enable analysis of the user's behavior. This semi-structured data represents different user actions from a clickstream log. The clickstream log contains data from URLs that are extracted from a Web server log, and therefore varies in format.

The unstructured part of the schema is generated in the form of product reviews, which are used for sentiment analysis, for example.

The 30 queries are grouped into four categories: Pure Hive Queries, Hive Queries with MapReduce programs, Hive Queries using natural language processing (NLP), and Queries using Apache Spark MLlib.

### Workload Flow

In this section, we discuss how to test the solution environment using the 30 use cases implemented as part of the TPCx-BB benchmark. This workload is fully developed, packaged, and offered as an easy-to-use self-contained kit with minimal external dependencies.

The workload can be divided into three high-level functional stages, as shown in Figure 6. The following sections describe each of these stages, relating it to the workload used to test the reference solution.

### Data Sources

Because the workload is based on a retailer with an online and physical store presence, data is accumulated from various sources including physical store and online transactions from transactional data sources which are primarily structured data. Clickstream logs (semi-structured data) from the online store help retailers understand customer interaction on the retailer's website. Finally, product reviews left by customers on the online store product pages are brought into the solution as unstructured data.

For testing purposes, gaining access to a large volume of data (tens or hundreds of TBs) is impractical. Instead, we generated the data locally using a packaged data generator, which enables us to scale to hundreds of TBs. The data generation runs as a MapReduce job and creates text-based data directly stored on HDFS. This phase is called the Data Generation phase in the workload documentation. Volume of data generated is defined by the Scale Factor; setting a nominal Scale Factor of 1000 will generate approximately 1 TB of raw data. A Scale Factor of 30000 will generate approximately 30 TB of raw data.

### Process and Transform

While it is possible to run jobs on the raw data, the performance will be unsatisfactory. Therefore, some integration optimization is required to gain performance benefits when running the jobs. In this reference solution, the raw data is further processed in the ETL stage using predefined schema creation and stored with related metadata in a Hive metadata store. This process is referred to as the Load Stage in the workload documentation.

### Deriving Insights and Running Analytics Use Cases

The final stage is running the code and starting to analyze the data. The workload implements 30 use cases; the primary mode of starting the workload is through query language, which starts applying the logic defined in the form of SQL expressions to satisfy the use case requirement. For those use cases where an additional machine-learning stage is involved, the query language stages are followed by calls

## Benchmark Workload Dataflow

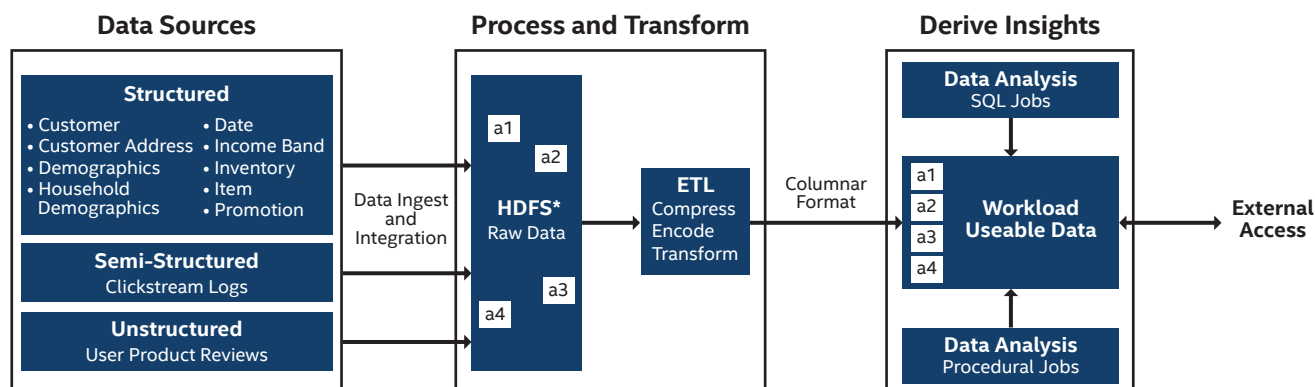


Figure 6. Three high-level functional stages comprise the benchmark workload.

to algorithms made available in Spark machine-learning libraries. In addition, procedural jobs written in Java or OpenNLP\* are invoked for selected use cases. Output and results are stored for consumption by external tools, a user interface, or other external systems.

There are two modes to run the use cases:

- **Single-user mode.** All 30 use cases are run sequentially; this is referred to as a Power Test. The primary objective for this mode is to understand the behavior of the solution when it is exclusively used by a single user. The resource utilization varies for each use case, depending on the amount of work required to complete the use case; therefore, resource utilization is a good indicator for understanding how many users running jobs with a similar profile can be supported by the solution.
- **Multi-user mode.** The workload driver invokes pre-defined logic to run multiple use cases in parallel. This is referred to as Throughput Mode. In this case, each user runs different use cases depending on how many parallel users are assigned. The primary objective for this mode is to understand cluster behavior in a multi-user setup. This is an important tool to determine the effective utilization of the cluster and understand scaling bottlenecks and limitations of the solution.

Note that the workload also supports a Refresh Mode to understand the behavior of data growth and its implications. This reference solution did not implement the refresh test.

## Workload Kit

The workload kit is specifically designed to measure the performance of big data batch analytics systems. This kit consists of a combination of the following software stack:

- **Driver.** Implemented using Java and Bash\* scripts, the versatile benchmark driver is the heart of the kit. It orchestrates the workflow involved in executing the workload.
- **Data generator.** A parallel data generator is used to create the input dataset required to run the workload. It is implemented as a Java program that runs as a MapReduce job and can generate hundreds of TBs of data in a relatively short time.
- **Use case workload.** The kit is designed to have self-contained modules for each framework capable of running the workload. All necessary binaries and configuration files reside inside the framework folders. The OpenNLP framework is also included for procedural programs invoking NLP.

In addition, to address the complexity of big data frameworks and understand the need to tune and optimize the benchmark, various configuration files provide sufficient hooks to tune the full benchmark or each individual query.

The workload requires a SQL-on-Hadoop API, distributed computing engine, and machine-learning libraries and can be run on big data systems in the following combinations:

- **Hive on MRv2:** Hive is the SQL-on-Hadoop API with MapReduce as the compute engine.

- **Hive on Spark:** Hive is the SQL-on-Hadoop API with Spark as the compute engine.
- **SparkSQL\* using Spark:** SparkSQL is the SQL-on-Hadoop API with Spark as the compute engine.

The design of the kit is modular, with options to support extensibility to new frameworks in the future. This reference solution uses Hive on MapReduce and Spark MLlib for the machine-learning stages. Please see documentation inside the kit for additional details.

## Quick Steps to Run the Workload

Running the workload is easy, assuming the cluster is already set up and functioning with all parameters set correctly as described in this document, and the workload kit is downloaded and extracted onto a gateway node.

It is strongly recommended that solution architects and testers thoroughly read the workload documentation to understand the multitude of options available to customize the tests. The [Big Bench forum](#) is also a good resource.

Prior to running the workload, it is recommended to run a simple example job to test the functionality of the setup. Examples include Pi\*, wordcount\*, sort examples from MapReduce or Spark, or Hue\* is available to create tables and Hive queries.

The following representative code snippet shows how to run the workload:

```
Open /home/user/%benchmark folder%/conf/
userSettings.conf

export BIG_BENCH_DEFAULT_DATABASE="bigbench"

export BIG_BENCH_DEFAULT_ENGINE="hive" <set SQL-
on-Hadoop API. Default is Hive but SparkSQL can be
used.>

export BIG_BENCH_DEFAULT_MAP_TASKS="96" <set this
number based on # of containers in your cluster>

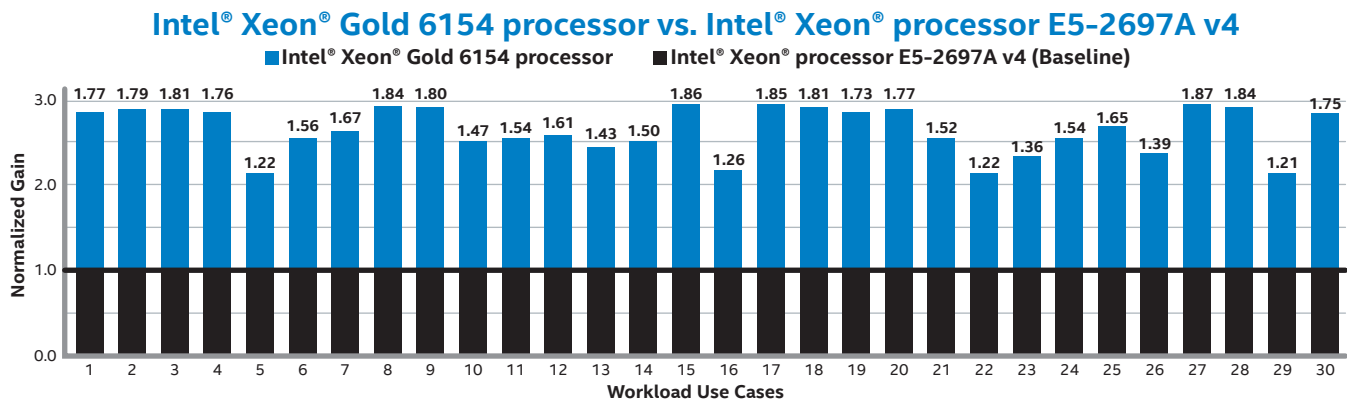
export BIG_BENCH_DEFAULT_SCALE_FACTOR="10000"
<Nominal Scale Factor 10000=~10TB>

export BIG_BENCH_DEFAULT_NUMBER_OF_PARALLEL_
STREAMS="2" <number of concurrent parallel users>

export BIG_BENCH_HADOOP_CONF="/etc/hadoop/conf.
cloudera.hdfs" <Make sure configuration directory
used by workload to read cluster config is pointing
to correct folders>

export BIG_BENCH_HADOOP_LIBS_NATIVE="/opt/cloudera/
parcels/CDH/lib/hadoop/lib/native" <path where
Hadoop stores native libraries such as compression
codecs>

/home/user/%benchmark folder%/bin runBenchmark -i
CLEAN_DATA,DATA_GENERATION,BENCHMARK_START,LOAD_
TEST,POWER_TEST,THROUGHPUT_TEST_1,BENCHMARK_
STOP,VALIDATE_POWER_TEST,VALIDATE_THROUGHPUT_TEST_1
```



**Figure 7.** The latest-generation Intel® Xeon® Scalable processor provides significant performance gains for each of the TPC Express Big Bench\* benchmark use cases.

### Setup and Configuration

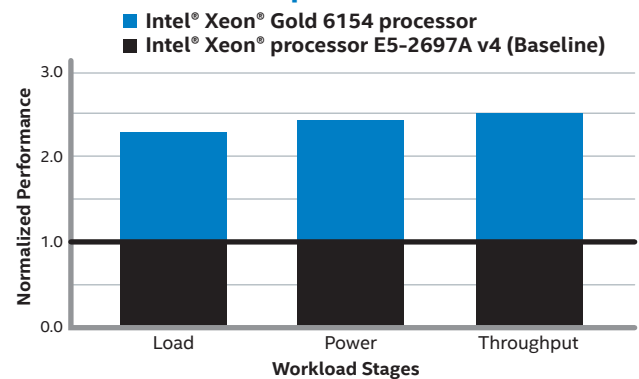
- **Hardware setup:** Follow the [guidelines](#) from HPE to set up and configure the server nodes.
- **Operating system:** Follow the [instructions](#) from Red Hat to install and configure the OS.
- **Cloudera CDH:** Follow the [instructions](#) from Cloudera to install CDH and enable the software components required to implement this reference solution.
- **Workload:** Download the workload kit archive from the [TPC](#) and extract it into a local directory on a single gateway node.

### Results

The results show that the Intel Xeon Gold 6154 processor, powering HPE ProLiant DL Gen10-based worker nodes, offers up to 1.8x percent performance gains while reducing the price-per-performance by 1.5x, when compared to an HPE ProLiant DL380 Gen9 equipped with an Intel® Xeon® processor E5-2697A v4. Also, the ProLiant DL Gen10-based nodes include enhancements such as Intel AES-NI for added security. Results also highlight the impact of effective tuning and optimizations (see [Figure 4](#), above), which help achieve the full potential of a big data warehouse.

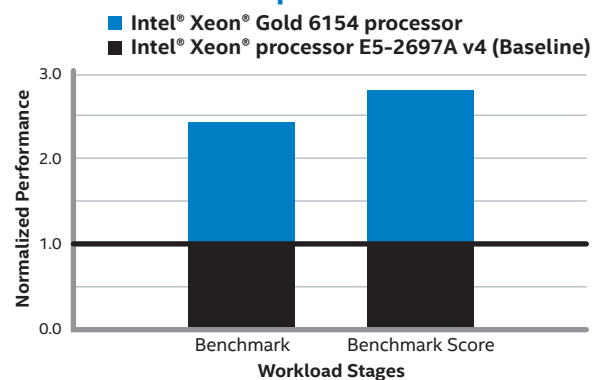
Detailed metrics are stored in the workload log folder with elapsed times for each stage, use case, and overall workload stored to determine the performance. Figure 7 shows the performance gain for each of the 30 benchmark use cases, compared to a server equipped with the previous-generation Intel Xeon processor. Figure 8 shows the performance gain for each stage (Load, Power, and Throughput) of the benchmark. Compared to the previous-generation system, the Load stage runs 28 percent faster, the Power stage runs 41 percent faster, and the Throughput stage runs nearly 46 percent faster. Overall, as shown in Figure 9, the latest-generation system completed the benchmark test 43 percent faster than the previous-generation system, and the benchmark score increased by 79 percent.

### Intel® Xeon® Gold 6154 processor vs. Intel® Xeon® processor E5-2697A v4



**Figure 8.** A big data analytics warehouse based on the latest Intel and HPE\* technology significantly accelerates each stage of the benchmark test.

### Intel® Xeon® Gold 6154 processor vs. Intel® Xeon® processor E5-2697A v4



**Figure 9.** The entire benchmark test runs 1.8x faster when running on a big data warehouse powered by HPE ProLiant\* DL Gen10 servers equipped with Intel® Xeon® Scalable processors. The benchmark score improves by 79 percent.

**Table 2.** Reference Solution Configuration Recommendations

	Good	Better	Best
<b>Business Use Case</b>	<ul style="list-style-type: none"> <li>• Exploration and development setup</li> <li>• No or non-critical SLA requirements</li> <li>• Cluster is occasionally busy</li> <li>• Data ingestion occurs during non-peak hours</li> </ul>	<ul style="list-style-type: none"> <li>• Multi-user, parallel jobs, and machine learning</li> <li>• Some SLA requirements to meet</li> <li>• Cluster is busy primarily during the day</li> <li>• Moderate data movement during peak hours</li> </ul>	<ul style="list-style-type: none"> <li>• Multi-user, parallel jobs, self-service analytics, and machine learning</li> <li>• Strict business SLA requirements</li> <li>• Cluster is busy for most of the day</li> <li>• Multiple sources are moving data during peak hours</li> </ul>
<b>Compute</b>	Intel® Xeon® Gold 5115 Processor	Intel Xeon Gold 6130 Processor	Intel Xeon Gold 6154 Processor
<b>Memory</b>	DDR4 192 GB	DDR4 384 GB	DDR4 384 GB to 768 GB
<b>HDFS* Storage</b>	12 HDDs	12 to 24 HDDs	12 to 24 HDDs, SSD Tier (Optional)
<b>Shuffle SSD</b>	Intel® SSD Data Center S4500 Series (optional)	Intel SSD Data Center S4500 Series	Intel SSD Data Center S4500 Series
<b>Network</b>	Intel® Ethernet 10 GbE	Intel Ethernet 10 GbE	Intel Ethernet 25 GbE

**GbE** – Gigabit over Ethernet; **HDD** – hard disk drive; **HDFS** – Hadoop Distributed File System; **SLA** – service-level agreement; **SSD** – Solid State Drive

## Architecture and Node Configuration Design Considerations

Worker nodes are the heart of big data EDW; they play a vital role in performance, SLA, TCO, and the ultimate success of any big data project. Table 2 shows how to tailor the reference solution worker nodes in a simple good-better-best configuration.

### Summary

Implementing a big data analytics warehouse is foundational to creating a comprehensive big data analytics solution that can process structured, unstructured, and semi-structured data. But while predictive and prescriptive big data analytic capabilities promise to deliver value, many enterprises struggle to build a cost-effective, high-performance big data warehouse. The reference solution described in this paper harnesses the power of Intel Xeon Scalable processors, HPE ProLiant DL Gen10 servers, and Cloudera Enterprise\* to provide a 1.8x increase in performance and a 1.5x decrease in TCO, compared to previous-generation processors, when running the TPCx-BB benchmark.<sup>2</sup>

The guidelines contained in this paper can help enterprises build an efficient big data warehouse, using technology from HPE and Intel, and achieve increased performance and lower TCO.

### Learn More

You may find the following resources useful:

- [HPE\\* Reference Architecture for Hadoop\\* on HPE Elastic Platform for Big Data Analytics \(EPA\)](#)
- [Intel® Xeon® Gold 6154 processor](#)
- [TPCx-BB\\* Result Highlights: Hewlett Packard Enterprise ProLiant\\* DL Gen10 for Big Data](#)



## Appendix A: TPC Express Big Bench\* (TPCx-BB\*) Benchmark Use Case Descriptions

Use Case	Implementation Method	Primary Data Type
1 Find top 100 products that are sold together frequently in given stores.	UDF or UDTF	Structured
2 Find the top 30 products that are mostly viewed together with a given product in online store.	MapReduce*	Semi-structured
3 For a given product, get a top-30 list sorted by number of views in descending order of the last five products that are mostly viewed before the product was purchased online.	MapReduce	Semi-structured
4 Shopping cart abandonment analysis: For users who added products in their shopping carts but did not check out in the online store during their session, find the average number of pages they visited during their sessions.	MapReduce	Semi-structured
5 Build a model using logistic regression for a visitor to an online store, based on existing users' online activities (interest in items of different categories) and demographics.	Machine Learning	Semi-structured
6 Identify customers shifting their purchase habit from physical store to Web sales.	Pure Query Language only	Structured
7 List top 10 states in descending order with at least 10 customers, who during a given month bought products with the price at least 20 percent higher than the average price of products in the same category.	Pure Query Language only	Structured
8 For online sales, compare the total sales amount for which customers checked online reviews before making the purchase and that of sales for which customers did not read reviews. Consider only online sales for a specific category in a given year.	MapReduce	Semi-structured
9 Aggregate total amount of sold items over different combinations of customers based on selected groups of marital status, education status, sales price, and different combinations of state and sales profit.	Pure Query Language only	Structured
10 For all products, extract sentences from reviews that have positive or negative sentiment.	UDF, UDTF, NLP	Unstructured
11 For a given product, measure the correlation of sentiments, including the number of reviews and average review ratings, on product monthly revenues within a given time frame.	Pure Query Language only	Semi-structured
12 Find all customers who viewed items of a given category on the Web in a given month and year that was followed by an in-store purchase of an item from the same category in the three consecutive months.	Pure Query Language only	Semi-structured
13 Display customers with both store and Web sales in consecutive years for which the increase in Web sales exceeds the increase in-store sales for a specified year.	Pure Query Language only	Structured
14 Calculate the ratio between the number of items sold over the Internet in the morning (7 to 8 AM) to the number of items sold in the evening (7 to 8 PM) for customers with a specified number of dependents.	Pure Query Language only	Structured
15 Find the categories with flat or declining sales for in-store purchases during a given year for a given store.	Pure Query Language only	Structured
16 Compute the impact of an item price change on store sales by computing the total sales for items in a 30-day period before and after the price change.	Pure Query Language only	Structured
17 Find the ratio of items sold with and without promotions in a given month and year. Only items in certain categories sold to customers living in a specific time zone are considered.	Pure Query Language only	Structured
18 Identify the stores with flat or declining sales in three consecutive months, and check if there are any negative reviews regarding these stores available online. Analyze the online reviews for these items to determine if there are any major negative reviews.	UDF, UDTF, NLP	Unstructured
19 Retrieve the items with the highest number of returns, where the number of returns was approximately equivalent across all store and web channels.	UDF, UDTF, NLP	Unstructured
20 Perform customer segmentation for return analysis. Segment customers according to varying criteria such as return frequency, returns to order ratio, and return amount ratio.	Machine Learning	Structured
21 Get all items that were sold in stores in a given month and year, and which were returned in the next six months and repurchased by the returning customer afterwards through the Web sales channel in the following three years.	Pure Query Language only	Structured
22 Compute the percentage change in inventory between the 30-day period before the price change and the 30-day period after the change.	Pure Query Language only	Structured
23 Calculate the coefficient of variation and mean of every item and warehouse of the given and the consecutive months.	Pure Query Language only	Structured
24 For a given product, measure the effect of competitors' prices on the product's in-store and online sales.	Pure Query Language only	Structured
25 Perform customer segmentation analysis. Segment customers according to key shopping dimensions such as how recent the last visit was, frequency of visits, and monetary amount.	Machine Learning	Structured
26 Cluster customers into book buddies or groups based on their in-store book purchasing histories.	Machine Learning	Structured
27 Extract competitor product names and model names (if any) from online product reviews for a given product.	UDF, UDTF, NLP	Unstructured
28 Build text classifier for online review sentiment classification (Positive, Negative, Neutral).	Machine Learning	Unstructured
29 Perform category affinity analysis for products purchased together online.	UDF or UDTF	Structured
30 Perform category affinity analysis for products viewed together online.	UDF, UDTF, MapReduce	Semi-structured

**NLP** – natural language processing; **UDF** – User-defined function; **UDTF** – user-defined table function

## Appendix B: Optimized Application Framework and API Parameter Settings

Parameter	Value	Description
ApplicationMaster Java* Maximum Heap Size	3072 MB	Maximum heap size, in bytes, of the Java MapReduce* ApplicationMaster, which manages all mapper and reducer tasks spawned for the job. ApplicationMaster (JVM) will terminate once the job is complete or failed.
Client Java Heap Size in Bytes	4 GB	Maximum Java process heap memory for local client processes executing the job; in our case, Hive* and spark-submit are primary clients kicking off the jobs. In an integrated environment where the big data warehouse works with external systems, consideration must be given to how many clients are active in parallel.
Compress Level of Codecs	Default compression	Compression level for the codec used to compress MapReduce outputs. Default compression is a balance between speed and compression ratio. Enabling compression reduces network and storage traffic and increases CPU utilization.
dfs.datanode.handler.count	20	Number of server threads for the DataNode to handle. More I/O requests to the DataNode require a higher value, until disk and network bandwidth reach the maximum capacity on the DataNode.
dfs.datanode.max.transfer.threads	12288	Maximum number of threads to use for transferring data in and out of the DataNode, using sockets for data connections. Setting this number too low can cause problems if a thread is blocked and is actively transferring data, which results in decreased utilization of the cluster and poor performance.
dfs.namenode.handler.count	114	Number of server threads for the NameNode Remote Procedure Call (RPC) queue per port and multiple handler (worker) threads that dequeue and process requests. A value of 100-120 is sufficient to support a cluster with 64 DataNodes.
dfs.namenode.service.handler.count	114	Number of NameNode RPC server threads that listen to requests from DataNodes and from all other non-client nodes. A value below 50 may cause issues during high-load situations.
io.file.buffer.size	128 KB	Size of buffer for read and write operations. The buffer size can be aligned with a shuffle handler size of 128 KB; however, with a few experiments, the buffer size can be increased above 1024 KB to boost disk I/O performance.
io.sort.mb	512 MB	Total amount of buffer memory to use while sorting files, in MB. By default, this parameter gives each merge stream 1 MB, which should minimize seeks. For long-running, data-intensive mapper jobs, a higher value will increase performance. However, the memory will be allocated within the mapper heap size.
mapreduce.job.reduce.slowstart.completedmaps	0.9	Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job. In a lightly loaded cluster with high-volume mapper outputs, it is possible to increase the performance by decreasing the percentage value. Increasing the value is required in a high-load cluster to prevent contention of resources between running mappers and reducers starting in parallel.
mapreduce.map.java.opts.max.heap	3072 MB	Maximum Java heap size, in bytes. The heap size should be lower than the maximum heap size allocated to mapper tasks.
mapreduce.map.memory.mb	4 GB	Amount of memory to request from the scheduler for each map task. When io.sort.mb is increased, a proportional increase to this parameter is required to prevent job failures.
mapreduce.output.fileoutputformat.compress.codec	org.apache.hadoop.io.compress.DefaultCodec	If the job outputs are compressed before they are written to the Hadoop Distributed File System* (HDFS*), the default is GZIP*, which provides a large reduction in file size, saving space on the disk. Using the Snappy* codec will speed up the compression but provides a smaller reduction in file size.
mapreduce.reduce.java.opts.max.heap	3072 MB	Maximum Java heap size, in bytes. The heap size should be lower than maximum heap size allocated to reducer tasks.
mapreduce.reduce.memory.mb	4 GB	Amount of memory to request from the scheduler for each reduce task. A higher value will help increase performance in a lightly loaded cluster.
mapreduce.reduce.shuffle.parallelcopies	50	Default number of parallel transfers run by reduce while copying shuffle output from mappers using http calls.
mapreduce.task.io.sort.factor	64	Number of streams to merge at once while sorting files. This determines the number of open file handles.
mapred.compress.map.output	TRUE	Turns MapReduce intermediate compression on or off for the whole cluster. When using Solid State Drives (SSDs) for shuffle acceleration it is absolutely necessary to compress the files to prevent running out of the disk space, resulting in job failures.
mapred.map.output.compression.codec	Snappy	Snappy is fast and reduces overhead on the CPU by compressing data before writing to the SSD.
yarn.app.mapreduce.am.resource.mb	4 GB	Amount of memory needed by the MapReduce ApplicationMaster.
yarn.nodemanager.resource.cpu-vcores	72	Number of virtual CPU cores that can be allocated for containers. If the cluster is also used for running other applications (such as an HBASE* region server), plan to allocate CPU resources to other applications.
yarn.nodemanager.resource.memory-mb	288 GB	Amount of physical memory, in MB, that can be allocated for containers. This value affects the amount of memory YARN* can utilize on this node; therefore, this parameter should be lower than the total memory of the machine.
yarn.scheduler.maximum-allocation-mb	360 GB	Largest amount of physical memory, in MB, that can be requested for a container.
zlib.compress.level	default	Compression level for the codec used to compress MapReduce outputs. Default compression is a balance between speed and compression ratio.

## Appendix C: Optimized Workload-Specific Parameter Settings

Table C1 provides the recommended values for workload-specific parameters when running use cases similar to the TPC Express Big Bench\* (TPCx-BB\*) benchmark. Table C2 provides descriptions for the parameters listed in Table C1.

**Table C1. Optimized Workload-Specific Parameters**

Query	Parameter	Value
1	hive.exec.reducers.max	1296
	hive.exec.reducers.bytes.per.reducer	4194304
	mapreduce.input.fileinputformat.split.maxsize	268435456
2	hive.exec.reducers.max	1296
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.output.fileoutputformat.compress	TRUE
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.exec.compress.intermediate	TRUE
3	hive.exec.reducers.max	1296
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	134217728
	hive.mapjoin.smalltable.filesize	600000000
4	hive.exec.reducers.max	1296
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	mapreduce.output.fileoutputformat.compress	TRUE
	hive.exec.compress.intermediate	TRUE
5	hive.exec.reducers.max	1296
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	mapreduce.output.fileoutputformat.compress	TRUE
	hive.exec.compress.intermediate	TRUE
6	hive.optimize.correlation	TRUE
	hive.exec.compress.intermediate	TRUE
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.exec.reducers.max	1296
	mapreduce.job.reduce.slowstart.completedmaps	0.99
	hive.optimize.sampling.orderby	TRUE
	hive.optimize.sampling.orderby.percent	0.2
7	hive.exec.reducers.max	1296
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	268435456

8	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
9	hive.exec.compress.intermediate	TRUE
	mapreduce.input.fileinputformat.split.maxsize	67108864
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.exec.reducers.max	1296
10	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	4194304
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.exec.reducers.bytes.per.reducer	16777216
11	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.mapjoin.smalltable.filesize	600000000
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
12	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	268435456
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
13	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
14	hive.exec.reducers.max	1296
	hive.exec.parallel	TRUE
	hive.exec.parallel.thread.number	8
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	mapreduce.input.fileinputformat.split.maxsize	536870912
15	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
16	hive.exec.reducers.max	2292
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.output.fileoutputformat.compress	TRUE
	hive.exec.compress.intermediate	TRUE
	mapreduce.input.fileinputformat.split.maxsize	268435456

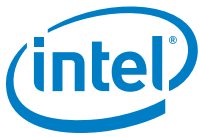


17	hive.exec.reducers.max	1296
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
18	hive.exec.reducers.max	1296
19	hive.exec.reducers.max	1296
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	mapreduce.input.fileinputformat.split.maxsize	33554432
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.exec.reducers.bytes.per.reducer	1
	hive.auto.convert.join.noconditionaltask.size	100000000
20	hive.exec.reducers.max	1296
	mapreduce.task.io.sort.factor	100
	mapreduce.task.io.sort.mb	512
	mapreduce.map.sort.spill.percent	0.99
	mapreduce.input.fileinputformat.split.maxsize	268435456
21	hive.exec.reducers.max	1296
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	268435456
22	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	67108864
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
	mapreduce.input.fileinputformat.split.maxsize	16777216
23	hive.exec.reducers.bytes.per.reducer	67108864
	hive.auto.convert.join.noconditionaltask.size	100000000
	hive.exec.mode.local.auto	TRUE
	hive.exec.max.created.files	1000000
	hive.reducers.max	1296
	hive.exec.mode.local.auto.input.files.max	900
	hive.exec.reducers.max	1296
24	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000

25	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	536870912
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
26	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
27	hive.exec.reducers.max	1296
	mapreduce.job.reduce.slowstart.completedmaps	0.01
	mapreduce.input.fileinputformat.split.maxsize	6088608
28	bigbench.hive.optimize.sampling.orderby	FALSE
	hive.exec.reducers.max	1296
29	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000
30	hive.exec.compress.intermediate	TRUE
	hive.exec.reducers.max	1296
	mapreduce.input.fileinputformat.split.maxsize	268435456
	hive.mapjoin.smalltable.filesize	600000000
	hive.optimize.correlation	TRUE
	hive.auto.convert.join.noconditionaltask	TRUE
	hive.auto.convert.join.noconditionaltask.size	100000000

**Table C2. Workload-Specific Parameter Descriptions**

Parameter	Description
hive.exec.reducers.max	Maximum number of reducers that will be used. If the one specified in the configuration property mapred.reduce.tasks is negative, Hive* will use this as the maximum number of reducers when automatically determining the number of reducers.
hive.exec.reducers.bytes.per.reducer	Defines the amount of data each reducer will process and sets parallelism during the reducer stage. This helps to improve job performance and utilization of cluster resources.
mapreduce.input.fileinputformat.split.maxsize	For splittable data, this parameter changes the portion of the data to which each mapper is assigned. By default, each mapper is assigned based on the block sizes of the source files. Entering a value larger than the block size will decrease the number of splits, which creates fewer mappers. Entering a value smaller than the block size will increase the number of splits, which creates more mappers.
hive.mapjoin.smalltable.filesize	Defines the size of tables considered small enough to cache. Caching tables allows for mapper-only joins, which are much faster than common joins for small tables in the big data context.
hive.optimize.correlation	Exploits intra-query correlations. For details, see the <a href="#">Correlation Optimizer</a> design document.
hive.auto.convert.join.noconditionaltask	Controls whether Hive enables the optimization for converting a common join into a mapjoin, based on the input file size. If this parameter is on, and the sum of size for n-1 of the tables/partitions for an n-way join is smaller than the size specified by hive.auto.convert.join.noconditionaltask.size, the join is directly converted to a mapjoin (there is no conditional task).
hive.auto.convert.join.noconditionaltask.size	If hive.auto.convert.join.noconditionaltask is off, this parameter does not take effect. However, if it is on, and the sum of size for n-1 of the tables/partitions for an n-way join is smaller than the value for this parameter, the join is directly converted to a mapjoin (there is no conditional task). The default is 10 MB.
hive.exec.compress.intermediate	Determines whether the output of the intermediate map/reduce jobs in a query is compressed or not.
hive.optimize.sampling.orderby	When hive.optimize.sampling.orderby=true, this parameter specifies the total number of samples to be obtained to calculate partition keys.
hive.optimize.sampling.orderby.percent	When hive.optimize.sampling.orderby=true, this parameter specifies the probability with which a row will be chosen.
hive.exec.parallel	Specifies whether to execute jobs in parallel. Applies to MapReduce* jobs that can run in parallel, such as jobs processing different source tables before a join. As of Hive v0.14, it also applies to move tasks that can run in parallel; for example, moving files to insert targets during multi-insert.
hive.exec.parallel.thread.number	How many jobs at most can be executed in parallel.
mapreduce.task.io.sort.factor	The number of streams to merge at once while sorting files. This determines the number of open file handles.
mapreduce.task.io.sort.mb	Total amount of buffer memory to use while sorting files, in MB. By default, gives each merge stream 1 MB, which should minimize seeks.
mapreduce.map.sort.spill.percent	Soft limit in the serialization buffer. Once reached, a thread will begin to spill the contents to disk in the background. Note that collection will not block if this threshold is exceeded while a spill is already in progress, so spills may be larger than this threshold when it is set to less than 0.5.
hive.exec.mode.local.auto	Allows Hive to determine whether to run in local mode automatically.
hive.exec.max.created.files	Maximum number of HDFS* files created by all mappers/reducers in a MapReduce job.
hive.reducers.max	Maximum number of reducers that can be defined for the job; used for increasing efficiency in job completion.
hive.exec.mode.local.auto.input.files.max	When hive.exec.mode.local.auto=true, the number of tasks should be less than this for local mode.
bigbench.hive.optimize.sampling.orderby	If the query will not run when hive.optimize.sampling.orderby=true because of a Hive bug, disable orderby sampling locally in this file.



<sup>1</sup> Gartner Survey Reveals Investment in Big Data Is Up but Fewer Organizations Plan to Invest, [gartner.com/newsroom/id/3466117](http://gartner.com/newsroom/id/3466117)

<sup>2</sup> TPCx-BB Result Highlights: Hewlett Packard Enterprise ProLiant DL Gen10 for Big Data, [tpc.org/tpcx-bb/results/tpcxbb\\_result\\_detail.asp?id=117071001](http://tpc.org/tpcx-bb/results/tpcxbb_result_detail.asp?id=117071001)

<sup>3</sup> The Value of Big Data: How Analytics Differentiates Winners, [bain.com/publications/articles/the-value-of-big-data.aspx](http://bain.com/publications/articles/the-value-of-big-data.aspx)

<sup>4</sup> IDC PeerScape: Self-Service Analytics Practices for Ensuring Successful Deployment and Use, [idc.com/getdoc.jsp?containerId=256758](http://idc.com/getdoc.jsp?containerId=256758)

<sup>5</sup> A Quick Guide to Structured and Unstructured Data, [smartdatacollective.com/quick-guide-structured-and-unstructured-data](http://smartdatacollective.com/quick-guide-structured-and-unstructured-data)

<sup>6</sup> Making Sense of Performance in Data Analytics Frameworks, [usenix.org/system/files/conference/nsdi15/nsdi15-paper-ousterhout.pdf](http://usenix.org/system/files/conference/nsdi15/nsdi15-paper-ousterhout.pdf)

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: [Learn About Intel® Processor Numbers](#).

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference [www.intel.com/performance/resources/benchmark\\_limitations.htm](http://www.intel.com/performance/resources/benchmark_limitations.htm) or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

THE INFORMATION PROVIDED IN THIS PAPER IS INTENDED TO BE GENERAL IN NATURE AND IS NOT SPECIFIC GUIDANCE. RECOMMENDATIONS (INCLUDING POTENTIAL COST SAVINGS) ARE BASED UPON INTEL'S EXPERIENCE AND ARE ESTIMATES ONLY. INTEL DOES NOT GUARANTEE OR WARRANT OTHERS WILL OBTAIN SIMILAR RESULTS.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others. © Intel Corporation

0118/BGOW/KC/PDF

336553-001US