DELL EMC

Running Spark & Hadoop with Dell EMC Isilon

Boni Bruno, CISSP, CISM, CGEIT

Chief Solutions Architect, Analytics

ABSTRACT

This white paper describes the benefits of running Spark and Hadoop with Dell EMC PowerEdge Servers and Gen6 Isilon Scale-out Network Attached Storage (NAS). Solution architecture and configuration guidelines are presented. Various performance benchmarks are included for reference.

April 2018

Contents

PREFACE

This Dell EMC ® publication covers topics to help the technical community take advantage of the resilience, scalability, and performance of combining Dell EMC PowerEdge™ Servers with Dell EMC Isilon™ Scale-Out Network Attached Storage (NAS) to implement or integrate a data lake for Hadoop and Spark providing analytics solutions to access, manage, and analyze data sets to improve business insights.

This publication is targeted at technical professionals (analytics consultants, technical support staff, IT Architects, and IT Specialists) that are responsible for delivering analytics solutions and support on Dell EMC PowerEdge and Isilon.

AUTHOR

Boni Bruno, CISSP, CISM, CGEIT is a Chief Solutions Architect for Dell EMC Unstructured Data Solutions Team. His area of knowledge includes HPC, Storage Architecture, Data Analytics, Security, and Enterprise Management. Boni is a regular speaker at numerous conferences on the subject of Enterprise Architecture, Security, and Analytics.

INTRODUCTION

This section provides an introduction to Dell EMC PowerEdge and Isilon for Hadoop and Spark solutions. Running both Hadoop and Spark with Dell EMC PowerEdge and Isilon provides an excellent platform for big data and analytics.

The following topics are discussed in this section:

- Impact of Big Data
- Challenges in Big Data Analytics
- Apache Spark
- Why use Dell EMC Isilon for Big Data and Analytics

Impact of Big Data

As the planet becomes more integrated, the rate of data growth is increasing exponentially. This data explosion is rendering commonly accepted practices of data management inadequate. As a result, this growth gives birth to a new wave of business challenges regarding data management and analytics. I'll cover how Dell EMC is poised to lead the next generation of technology to meet and conquer the data management challenges that Big Data presents. The increasing volume and detail of information, the rise of multimedia and social media, and the *Internet of Things* are expected to fuel continued exponential data growth for the foreseeable future. The Internet of Things is generating large volumes and various data from numerous varied sources as sensors, vehicles, games, set-top boxes, and household appliances. Capturing, correlating, and analyzing this data can produce valuable insights for a company.

The number of insights and trends that can be extracted from the stored but unexplored data, which is a new resource for many organizations, is unlimited. Executives have relied on experience and intuition to formulate critical business decisions in past decades, but in the new era of big data, key decisions can be supported by decision support systems.

Challenges in Big Data Analytics

It is estimated that a staggering 70% of the time that is spent on analytics projects is concerned with identifying, cleansing, and integrating data because of the following issues:

- *Data is often difficult to locate because it is scattered among many business applications and business systems.*
- *Data often needs reformatting to make it easier to analyze.*

- *Data often needs to be refreshed regularly to keep it up-to-date.*

Acquiring data for analytics in an *ad hoc* manner creates a huge burden on the teams that own the systems that supply the data. Often, the same type of data is repeatedly requested and the original information owner finds it hard to track who has copies of which data.

As a result, Dell EMC advise many organizations to consider implementing a ***data lake*** solution. A data lake is a set of one or more data repositories that are created to support data discovery, analytics, *ad hoc* investigations, and reporting. The data lake contains data from many different sources. People in the organization are welcome to add data to the data lake and access any updates as necessary. I'll discuss later how Dell EMC Isilon is the ideal storage solution for implementing a data lake.

Apache Spark

Apache Spark is an open source cluster computing framework for large-scale data processing. Like MapReduce, Apache Spark provides parallel distributed processing, fault tolerance on commodity hardware, and scalability. With its in-memory computing capabilities, analytic applications can run faster on Apache Spark compared to Hadoop MapReduce. Spark compliments Hadoop as it provides a unified solution for big data requirements.

Apache Spark is highly versatile and known for its ease of use in creating algorithms that harness insight from complex data. In addition to its ease of use, this framework covers a wide range of workloads through its different modules:

- *Interactive queries through Apache Spark SQL*
- *Streaming data with Apache Spark Streaming*
- *Machine Learning with the MLlib module*
- *Graph processing with GraphX*

Applications can be built using simple APIs for Scala, Python, and Java:

- *Batch applications leveraging MapReduce*
- *Iterative algorithms that build upon each other*
- *Interactive queries and data manipulation through "Notebooks" (a web-based interface)*

Apache Spark runs on Hadoop YARN, Apache Mesos, stand-alone, and starting with version Spark 2.3 Kubernetes. This paper focuses on running Spark with Hadoop YARN as currently that represents a majority of the deployments most enterprise customers have adopted. The security frame work of Hadoop is more mature, but Mesos and Kubernetes are definitely gaining traction, it will be interesting to see which technology gains the most market share in the long run.

Figure 1 presents the Apache Spark architecture.

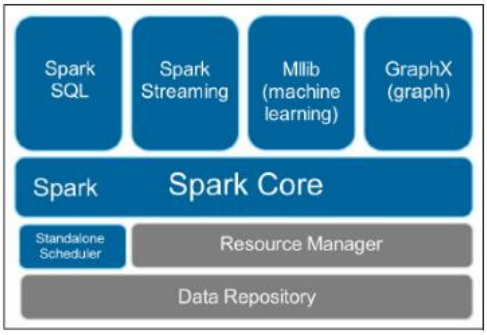


Figure 1 Apache Spark software stack

Why use Dell EMC Isilon for Big Data & Analytics

The Dell EMC® Isilon® scale-out network-attached storage (NAS) platform provides Apache Spark clients with direct access to big data through a Hadoop File System (HDFS) interface or Network File System (NFS) depending on whether you installed Spark with Hadoop or in Stand-alone mode. Powered by the distributed Dell EMC Isilon OneFS® operating system, a Dell EMC Isilon cluster delivers a scalable pool of storage with a global namespace.

Spark clients access data that is stored in an Isilon cluster by using the HDFS or NFS protocols. Every Isilon node can act as a NameNode and a DataNode for a Hadoop/Spark cluster or single scalable NFS mount point for a Spark Standalone cluster. Each node boosts performance and expands the cluster's capacity. For big data analytics, the Isilon scale-out distributed architecture minimizes bottlenecks, rapidly serves big data, and optimizes performance for analytic jobs.

An Isilon cluster fosters data analytics without ingesting data into an HDFS based file system. With a Dell EMC Isilon cluster, you can store data on an enterprise storage platform with your existing workflows and standard protocols, including SMB, HTTP, FTP, REST, and NFS as well as HDFS. Regardless of whether you write the data with SMB or NFS, you can analyze it with either Hadoop or Spark compute clusters through HDFS. There is no need to set up an HDFS file system and then load data into it with tedious HDFS copy commands or inefficient Hadoop connectors.

An Isilon cluster simplifies data management while cost-effectively maximizing the value of data. Although high-performance computing with Hadoop has traditionally stored data locally in compute clusters HDFS file system, the following use cases make a compelling case for coupling Hadoop or Spark based analytics with Isilon scale-out NAS:

- *Store data in a POSIX-compliant file system with SMB and NFS workflows and then access it through HDFS*
- *Scale storage independently of compute as your data sets grow*
- *Protect data more reliably and efficiently instead of replicating it with HDFS 3X mirror replication*
- *Eliminate HDFS copy operations to ingest data and Hadoop fs commands to manage data*
- *Implement distributed fault-tolerant NameNode and DataNode services*
- *Manage data with enterprise storage features such as deduplication and snapshots*

Storing data in an Isilon scale-out NAS cluster instead of HDFS clients streamlines the entire analytics workflow. The Isilon OneFS HDFS interface eliminates extracting the data from a storage system and loading it into an HDFS file system. Isilon OneFS multiprotocol data access with SMB and NFS eliminates exporting the data after you analyze it. The result is that you can not only increase the ease and flexibility with which you analyze data, but also reduce capital expenditures and operating expenses.

SOLUTION OVERVIEW

This section covers the Dell EMC solution for Spark and Hadoop from both a software and a hardware perspective.

The following topics are described in this chapter:

- *Overview of Isilon for Big Data*
- *Hardware components*
- *Software components*
- *How Hadoop and Spark works with Isilon*

Overview of Isilon for Big Data

The Dell EMC Isilon scale-out platform combines modular hardware with unified software to provide the storage foundation for data analysis. Isilon scale-out NAS is a fully distributed system that consists of nodes of modular hardware arranged in a cluster. The distributed Isilon OneFS operating system combines the memory, I/O, CPUs, and disks of the nodes into a cohesive storage unit to present a global namespace as a single file system.

The nodes work together as peers in a shared-nothing hardware architecture with no single point of failure. Every node adds capacity, performance, and resiliency to the cluster, and each node acts as a NameNode and DataNode. The NameNode daemon is a distributed process that runs on all the nodes in the cluster. A compute client can connect to any node in the cluster to access NameNode services.

As nodes are added, the file system expands dynamically and redistributes data, eliminating the work of partitioning disks and creating volumes. The result is a highly efficient and resilient storage architecture that brings all the advantages of an enterprise scale-out NAS system to storing data for analysis.

Unlike traditional storage, Spark's ratio of CPU, RAM, and disk space depends on the workload—factors that make it difficult to size a Spark cluster before you have had a chance to measure your Spark workload. Expanding data sets also makes sizing decisions upfront problematic. Isilon scale-out NAS lends itself perfectly to this scenario: Isilon scale-out NAS lets you increase CPUs, RAM, and disk space by adding nodes to dynamically match storage capacity and performance with the demands of a dynamic Spark workload.

An Isilon cluster optimizes data protection. OneFS more efficiently and reliably protects data than HDFS. The HDFS file system, by default, replicates a block of data three times. In contrast, OneFS stripes the data across the cluster and protects the data with forward error correction codes, which consume less space than replication with better protection.

An Isilon cluster also includes enterprise features to back up your data and to provide high availability. For example, in managing your DataNode data, a best practice with a traditional Hadoop system is to back up your data to another system—an operation that must be performed with brute force by using a tool like DistCP. OneFS includes support for NDMP backups, cluster synchronization, geo-replication, snapshots, file system journal, virtual hot spare, antivirus, Integrity Scan, dynamic sector repair, and accelerated drive rebuilds.

The enterprise features of OneFS ease data management. OneFS includes storage pools, deduplication, automated tiering, quotas, high-performing SSDs, capacity-optimized HDDs, and cluster monitoring and forecasting with InsightIQ.

SmartPools, for example, provides tiered storage so that you can store current data in a high-performance storage pool while storing older data in a lower, more cost-effective pool in case you need to analyze it again later.

For security, OneFS can authenticate HDFS connections with Kerberos and includes support for Ranger. SmartLock can protect sensitive data from malicious, accidental, or premature alteration or deletion to help comply with SEC 17a-4 regulations.

Hardware Components

Selecting a hardware architecture to support the deployment of a big data and analytics solution requires an understanding of the relevant components and how they impact many aspects: performance, reliability, availability, costs, and management.

Dell EMC PowerEdge R640 Server: Get scalable computing and shuffle storage in a 1U, 2-socket platform for an ideal mix of performance and cost.

Dell EMC Isilon Hybrid Scale-out NAS: Isilon hybrid platforms include Isilon H600 for high performance, Isilon H500 for a versatile balance of performance and capacity, and Isilon H400 to support a wide range of enterprise file workloads. With SSD technology for caching, Isilon hybrid systems offer additional performance gains for metadata-intensive operations. These models support dual 10GbE or 40GbE ports per node on the front-end and dual InfiniBand or 40GbE ports on the back-end.

Dell EMC Networking S4048-ON 10/40GbE Switch: Top-of-Rack (TOR), high-density 1U switch, with 48 10GbE SFP+ and 6 x 40GbE ports and up to 720 Gbps with ultra-low latency and line-rate performance.

Dell EMC IB Switch: IB SWITCH, used for Isilon back-end. Note: 40GbE back-end is also available. Contact Isilon for specific 40GbE model options.

Network Diagram – Solution Example

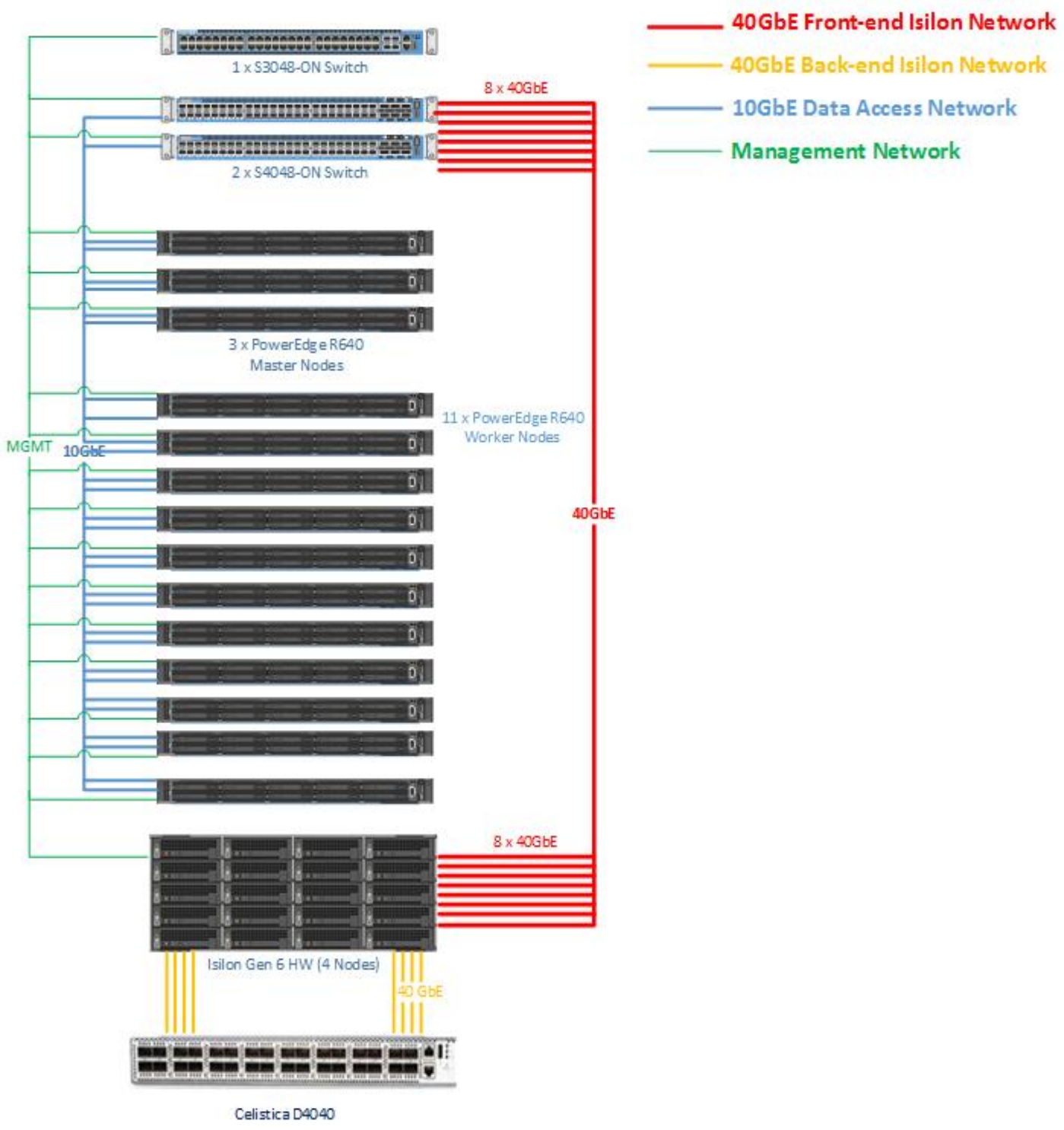


Figure 2 – Example Network Architecture with Isilon NAS

Network Diagram – Test Lab Configuration

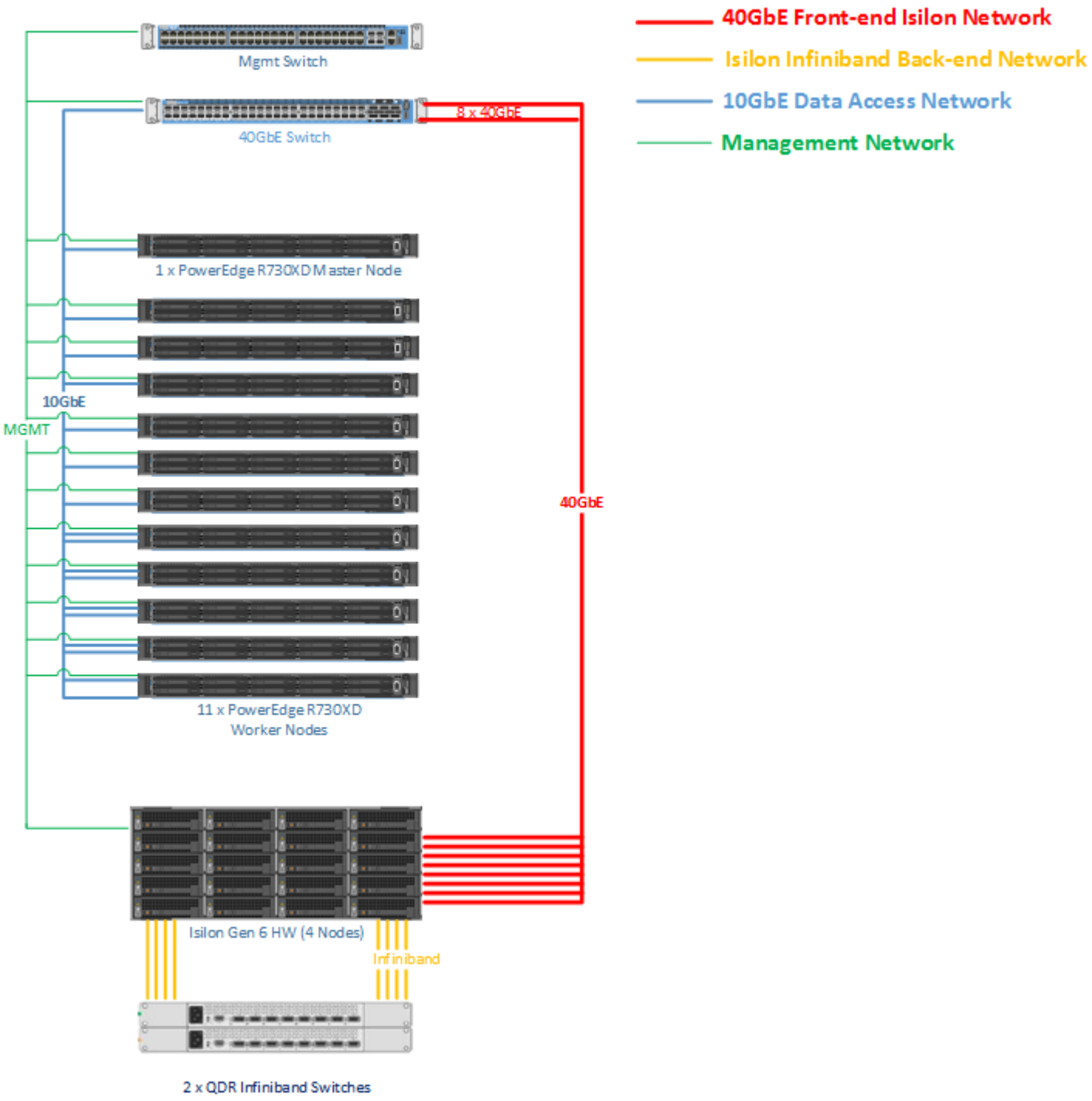


Figure 3 – Configuration used for this paper

Software Components

Figure 4 shows the software components on the Dell EMC hardware for Hadoop and Spark:

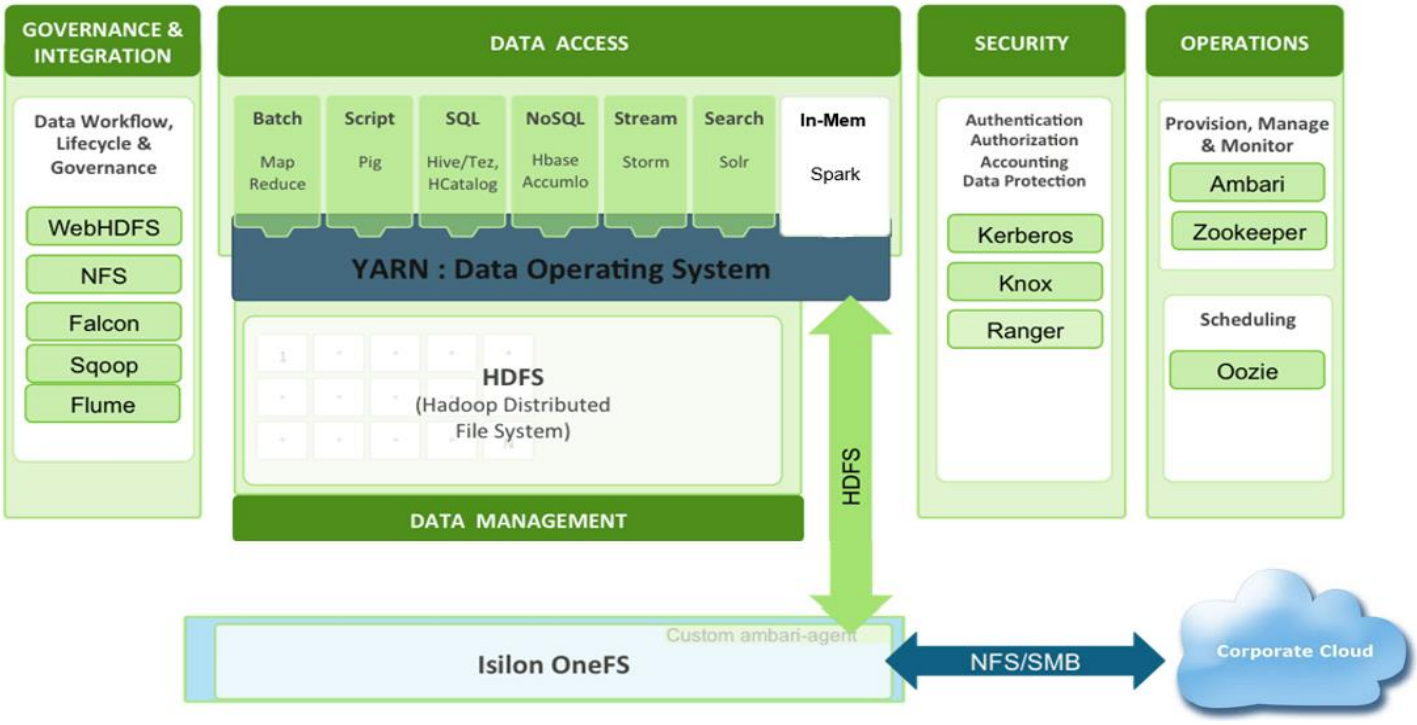


Figure 4 – Software Stack

How Spark & Hadoop work with Isilon

An Isilon cluster separates data from compute. As Spark & Hadoop clients execute jobs, the clients access the data stored on an Isilon cluster over HDFS. OneFS becomes the HDFS file system for all the clients. If using NFS, Isilon would be the NFS server for all the clients as well. Scale-out performance and multiprotocol support are key benefits of using Dell EMC Isilon for Spark and Hadoop workloads.

OneFS implements the server-side operations of the HDFS protocol on every node, and each node functions as both a NameNode and a DataNode. An Isilon node, however, does not act as a YARN Resource Manager or Node Manager; those functions remain with Hadoop. OneFS contains no concept of a secondary NameNode: Every Isilon node functions as a NameNode, the function of the secondary NameNode—checking pointing the internal NameNode transaction log—is unnecessary and inefficient. For more details, see the Hadoop Performance with Isilon presentation which covers NameNode Benchmark results showing Isilon with 37% better NNBench (NameNode Bench) performance over tradition DAS Hadoop implementations.

OneFS load balances HDFS connections across all the nodes in the Isilon cluster. Because OneFS stripes Hadoop data across the cluster and protects it with parity blocks at the file level, any node can simultaneously serve DataNode traffic as well as NameNode requests for file blocks. This is a unique engineering feature Isilon provides that is not available with other shared storage solutions in the market.

A virtual racking feature mimics data locality. You can, for example, create a virtual rack of nodes to assign compute clients to the nodes that are closest to a client's network switch if doing so is necessary to work with your network topology or to optimize performance.

Client computers can access any node in the cluster through dual 10 GigE or 40GigE on newer node network connections. A SmartConnect license adds additional network resilience with IP address pools that support multiple DNS zones in a subnet as well as IP failover. In OneFS, an IP address pool appears as a: ***groupnet.subnet:poolname***. A best practice is to bind multiple IP addresses to each node interface in a Dell EMC Isilon SmartConnect™ network pool. This offers much better fault tolerance and resilience. For more information, read my post at *EMC Community Network*.

Spark & Hadoop Configuration Guidance

Spark can be deployed with different cluster managers – Spark's standalone cluster manager, Apache Mesos, Kubernetes (starting with Spark version 2.3) or Hadoop YARN. Most organizations that already have deployed Hadoop will look to integrate Spark with Hadoop's YARN framework to manage all data processing under one resource manager.

One of the most significant benefits of Hadoop YARN is to separate processing from resource management. This enables a variety of new and familiar tools like Spark SQL to identify data of value interactively and in real time, without being hampered by the often I/O intensive, high latency MapReduce framework.

YARN has the concept of labels for groupings of compute nodes. Jobs submitted through YARN can be flagged to perform work on a particular set of nodes when the appropriate label name is included with the job. Thus, an organization can create groups of compute resources that are designed, built, or optimized for particular types of work, allowing for jobs to be passed out to subsets of the hardware that are optimized to the type of computation.

Spark on YARN is an optimal way to schedule and run Spark jobs on a Hadoop cluster alongside a variety of other data-processing frameworks, leveraging existing clusters using queue placement policies, and enabling security by running on Kerberos-enabled clusters.

Spark 2.1

Apache Spark 2.x is a major release update of Spark 1.x and includes significant updates in the areas of API usability, SQL support, performance improvements, structured streaming, R User-Defined Functions support, as well as operational improvements.

One of the largest changes in Spark 2.x is the new updated APIs including unifying DataFrame and Dataset APIs providing type safety for DataFrames, the new Spark Session API with a new entry point that replaces the old SQL Context and Hive Context for DataFrame and Dataset APIs, a new streamlined configuration API for Spark Session, and a new improved Aggregator API for typed aggregation in Datasets.

Spark 2.x substantially improved SQL functionality with ANSI SQL 2003 support which enables running all 99 queries from the Hive testbench, which is similar to TPC-DS benchmark. Major improvements to Spark SQL include: a native SQL parser that supports both ANSI-SQL as well as Hive QL, native DDL command implementations, and subquery support. Additional new Spark SQL 2.x improvements include native CSV data source, off-heap memory management for both caching and runtime execution, and Hive-style bucketing support.

Spark 2.x also made substantial performance improvements over Spark 1.x. By implementing a new technique called "whole stage code generation", Spark 2.x improves the performance 2-10 times for common operators in SQL and DataFrames. Other performance improvements include: improved Parquet scan throughput through vectorization, improved ORC performance, many improvements in the Catalyst query optimizer for common workloads, and improved window function performance via native implementations for all window functions.

In the area of the Spark Machine Learning API, Spark 2.x replaces the RDD-based APIs in the spark.mllib package with the DataFrame-based API in the spark.ml package. New features in the Spark 2.x Machine Learning API include: ML persistence to support saving and loading ML models and Pipelines, new MLlib APIs in R for generalized linear models, naive Bayes, k-means clustering, survival regression, new MLlib APIs in Python for LDA, Gaussian Mixture Model, Generalized Linear Regression, etc.

In the area of Spark Streaming, Spark 2.x introduced a new high-level streaming API, called Structured Streaming, built on top of Spark SQL and the Catalyst optimizer. Structured Streaming enables users to program against streaming sources and sinks using the same DataFrame/Dataset API as in static data sources. Starting with Spark version 2.3, support for continuous processing for structured streaming and the ability to join two streams has been added. There is also an experimental Streaming API (v2) that works with batch, micro batch, and continuous execution. See New Spark Release for more information. This document focuses on Spark release 2.1 as that is what is included in the Hortonworks HDP 2.6 release I tested.

Spark Hardware Guidelines

General guidance running Spark with Dell EMC PowerEdge Servers

Sizing hardware for Spark and scaling a Spark cluster depends on the use case, but Spark is primarily a CPU-bound workload that can benefit from more CPU cores, higher memory, and either flash or SAS disks for temporary storage. Following are some general guidelines based on the recommendations from the Apache Software Foundation:

CPU cores

Spark scales well to tens of CPU cores per machine because it performs minimal sharing between threads. Provisioning at least 8-16 cores per machine is a good start; and, based on the workload impact on CPU, more cores may be required. Once data is in memory, most applications are either CPU- or network-bound. Special use cases like machine or deep learning will require significantly higher numbers of cores or special purpose processors like GPUs that have 100s or 1000s of cores.

For general workloads such as Spark SQL the recommended hardware guidelines for CPU (Intel Xeon processor) and memory (768GB) are optimal, but for demanding workloads such as machine learning, customers might want to consider more powerful CPUs such as Intel Xeon processors with 20 cores at 2.4 GHz.

Memory

In general, Spark can run well with anywhere from 32GB to hundreds of gigabytes of memory per machine. In all cases, it is recommended to allocate only 75% of the memory for Spark, leaving the rest for the operating system and buffer cache. Since Spark is optimized for memory, it can make use of as much memory as is available. Actual memory requirement depends on the application and data set size.

Dell EMC PowerEdge R640 with 768GB RAM, should be sufficient for the majority of Spark workloads; R640 server can accommodate up to 1.5TB RAM for the most memory demanding workloads and up to two Intel Xeon Scalable processors with up to 28 cores per processor.

Storage systems

Most Spark jobs will likely have to read input data from a storage system (e.g., the Hadoop File System, or NoSQL data stores), so having fast access to data from Spark's compute nodes is important. Since the Gen6 Isilon models offers high performance scale-out architecture to independently scale compute and storage, to accommodate data and workloads growing at different rates, Isilon makes use of either 40GbE or InfiniBand networks to minimize latency between Isilon nodes. Most Hadoop solutions are based on 10GbE only.

Local disks

While Spark can perform a lot of its computation in memory, it still uses local disks to store data that doesn't fit in RAM, as well as to preserve intermediate output between stages. It is recommended to have at least two disks per node, in addition to the OS disks, that are dedicated for the intermediate data and large enough to fit the workload. These disks should be configured without RAID (just as separate mount points).

The PowerEdge R640 provides a lot of disk options. Front drive bays: Up to 10 x 2.5" SAS/SATA (HDD/SSD) with up to 8 NVMe SSD max 58TB or up to 4 x 3.5" SAS/SATA HDD max 48TB. Rear drive bays: Up to 2 x 2.5" SAS/SATA (HDD/SSD), NVMe SSD max 12TB, more than ideal for intermediate data for Spark jobs.

Network

When data is cached in memory, Spark applications can be network-bound in clusters with older networks. Spark applications such as group-bys, reduce-bys, and SQL joins result in a lot of network traffic between the compute nodes; moving to a 10GbE or, ideally, 25GbE network will help avoid network bottlenecks. Dell EMC PowerEdge R640 supports many network daughter card options: 4 x 1GE or 2 x 10GE + 2 x 1GE or 4 x 10GE or 2 x 25GE. Isilon Gen6 models support multiple 40GbE interfaces per node to provide high throughput on the storage network.

Hadoop Configuration Guidelines

The configuration changes mentioned in this section are intended to assist in optimizing the setup of the various Hadoop services when separating storage from compute using Dell EMC Isilon Scale-out NAS. Customizations may be required depending on your particular deployment and use case.

Mandatory HADOOP HDFS Configuration Changes

As stated before, Isilon acts as the NameNode and DataNode for the entire cluster. The http address PORT of the NameNode and DataNode for the HDFS interface on Isilon OneFS is TCP PORT 8082. Make sure to make the following changes to your hdfs-site.xml configuration file:

dfs.namenode.http-address = <Isilon-SmartConnect-ZoneName:8082>

- dfs.namenode.https-address = <Isilon-SmartConnect-ZoneName:8080>

Recommended Performance Configuration Changes for HDFS on Hadoop

- dfs.client-write-packet-size = 131072 Note: This is the write packet size Isilon OneFS uses.
- dfs.blocksize = 134217728 Bytes

  Note: This is the client-side block size, this determines how an HDFS client writes a block of data to the cluster.
- On Isilon OneFS, you can configure the HDFS Block in the GUI as shown below:

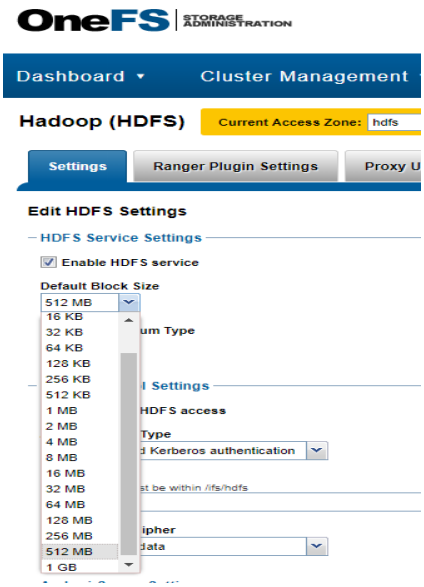


Figure 5 – Isilon OneFS UI for HDFS Protocol

Isilon OneFS support HDFS block sizes from 4KB to 1GB. Most deployments seem to do well at 128MB or 256MB. In some cases when datasets are really large (TB range), a block size of 512MB may be appropriate. The Isilon block size determines how the OneFS HDFS daemon returns data to read requests. The HDFS clients must contain enough memory to accept the specified Isilon HDFS block size. If the Isilon HDFS block size is set to 512 MB, and if a client's maximum map JVM size is set to 512 MB, the client will be unable to pull files from the cluster. Make sure clients have sufficient map memory for JVM before increasing HDFS block size on Isilon.

Recommended YARN Configuration Changes for Spark

When using YARN as the resource manager for Spark, it is important to align the YARN and Spark configurations accordingly. Spark executor memory and executor cores settings result in allocation requests to YARN with the same values, configure YARN to accommodate your Spark configuration.

It is recommended that 75% of memory be allocated for Spark, leaving the rest for OS and buffer cache. For a recommended configuration of 768GB memory, this means Node Managers should be configured to use 576GB memory.

Configure enough physical memory and vcores for YARN:

- Amount of physical memory that can be allocated for containers: **yarn.nodemanager.resource.memory-mb = 576 GB**
- Amount of vcores available on a compute node that can be allocated for containers: **yarn.nodemanager.resource.cpu-vcores = 64**

Configuring the number of YARN containers depends on the nature of the workload. For example, a Spark workload that needs 24GB containers will result in 24 total containers being available on the node. Similarly, a workload that needs 48GB containers will result in 12 containers being available on the node. The general guideline is to configure containers in a way that maximizes the utilization of the vcores and memory on each node in the cluster.

Assuming identical compute nodes, the total number of containers available for the application can be written as (subtracting 1 to account for the YARN application master that will be started):

• Number of 24GB containers = 24 * Number of compute nodes – 1

• Number of 48GB containers = 12 * Number of compute nodes – 1

If you had 12 PowerEdge R640 compute nodes, this will result in a total of 287 Spark executors with 24GB memory or 143 Spark executors with 48GB memory.

- The node-locality-delay specifies how many scheduling intervals to let pass attempting to find a *node local slot* to run on prior to searching for a *rack local slot*. Since data on Isilon will always be remote, setting a value other than zero will just slow down the scheduler as nothing will ever reside in a node local slot. Setting the delay to 0 will improve performance: **yarn.scheduler.capacity.node-locality-delay = 0**

Specify the location of YARN local log files and directories to use local disks on the compute nodes:

- ***yarn.nodemanager.log-dirs: /data1/hadoop/yarn/log, /data2/hadoop/yarn/log***
- ***yarn.nodemanager.local-dirs: /data1/hadoop/yarn/local, /data2/hadoop/yarn/local***

The Isilon HDFS storage should be for actual data only to limit network traffic, *always keep logs and shuffle space local to the compute nodes*. The impact of Shuffling on Spark Performance will be discussed more in the next section.

Spark Configuration Guidelines

As mentioned before, running Spark with YARN is recommended as it leverages YARN services for resource allocation, runs Spark executors in YARN containers, and supports workload management and security features.

Spark on Hortonworks Data Platform (HDP) 2.6

The Spark version with HDP comes with YARN support prebuilt. HDP 2.6 supports Spark versions 1.6 and 2.1. It also comes with Spark Thrift server for JDBC and ODBC clients to run Spark SQL queries, Livy for local and remote access to Spark through the Livy REST API, and Apache Zeppelin for browser-based notebook access to Spark.

Spark 1.x and Spark 2.x can be installed on the same HDP 2.6 Dell EMC Isilon cluster. If you currently use Spark 1.x, you can also install Spark 2.1 and test jobs on Spark 2.1 in parallel with a Spark 1.x working environment. After verifying that all scripts and jobs run successfully with Spark 2.1, you can migrate jobs from Spark 1.x to Spark 2.1.

If more than one version of Spark is installed on a node, your job runs with the default version. In HDP 2.6, the default is Spark version 1.6.

If you want to run jobs on a specific version of Spark, use one of the following approaches:

• Use full paths in your scripts:

  – *To use Spark 1.6, use /usr/hdp/current/spark-client/bin/spark-submit*

  – *To use Spark 2.1, use /usr/hdp/current/spark2-client/bin/spark-submit*

• Set the SPARK_MAJOR_VERSION environment variable to the desired version of Spark before you launch the job:

  – *To use Spark 1.6, set SPARK_MAJOR_VERSION=1*

  – *To use Spark 2.1, set SPARK_MAJOR_VERSION=2*

Spark Memory Management

Because of the in-memory nature of most Spark computations, Spark programs can be bottlenecked by any resource in the cluster: CPU, network bandwidth, or memory. Memory usage in Spark largely falls under one of two categories: execution and storage. Execution memory refers to that used for computation in shuffles, joins, sorts and aggregations, while storage memory refers to that used for caching and propagating internal data across the cluster. The best way to size the amount of memory consumption a dataset will require is to create an RDD, put it into cache, and look at the "Storage" page in the web UI as shown below in Figure 6:

Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
MapPartitionsRDD	Memory Deserialized 1x Replicated	55	79%	422.8 MB	0.0 B

RDD Storage Info for MapPartitionsRDD

Storage Level: Memory Deserialized 1x Replicated
Cached Partitions: 68
Total Partitions: 70
Memory Size: 522.8 MB
Disk Size: 0.0 B

Data Distribution on 12 Executors

Host	Memory Usage	Disk Usage
hdp26-wrk7.solarch.lab.emc.com:39072	53.8 MB (10.4 GB Remaining)	0.0 B
10.246.21.136:36376	0.0 B (10.5 GB Remaining)	0.0 B
hdp26-wrk11.solarch.lab.emc.com:57918	46.1 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk2.solarch.lab.emc.com:55015	38.4 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk6.solarch.lab.emc.com:43912	46.1 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk1.solarch.lab.emc.com:36504	38.5 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk3.solarch.lab.emc.com:47344	46.1 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk9.solarch.lab.emc.com:60946	53.8 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk5.solarch.lab.emc.com:60056	46.1 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk4.solarch.lab.emc.com:41034	46.1 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk10.solarch.lab.emc.com:45898	53.8 MB (10.4 GB Remaining)	0.0 B
hdp26-wrk8.solarch.lab.emc.com:45102	53.8 MB (10.4 GB Remaining)	0.0 B

Figure 6 - RDD Storage Information

Sometimes, you will get an *OutOfMemoryError* not because your RDDs didn't fit in memory, but because the working set of one of your tasks, such as one of the reduce tasks in *groupByKey*, was too large. Spark's shuffle operations (*sortByKey, groupByKey, reduceByKey, join, etc*) build a hash table within each task to perform the grouping, which can often be large.

The simplest fix here is to *increase the level of parallelism*, so that each task's input set is smaller. Spark can efficiently support tasks as short as 200 ms, because it reuses one executor JVM across many tasks and it has a low task launching cost, so you can safely increase the level of parallelism to more than the number of cores in your clusters.

Spark2 History Server User Interface

The Spark Application UI (http://<spark2-server.:4040) and the Spark2 History Server UI (http://<spark2-server>:18081) can be very helpful and show not only the history of your Spark2 application runs, but also provide important Spark statistics for each application run. Figure 4 below shows a snippet from the main page of the Spark 2 History Server UI. This information is pulled from the Spark 2 logs stored by default on the HDFS ***/spark2-history*** directory on Isilon. This directory should be set to have user:spark and user group: hadoop.

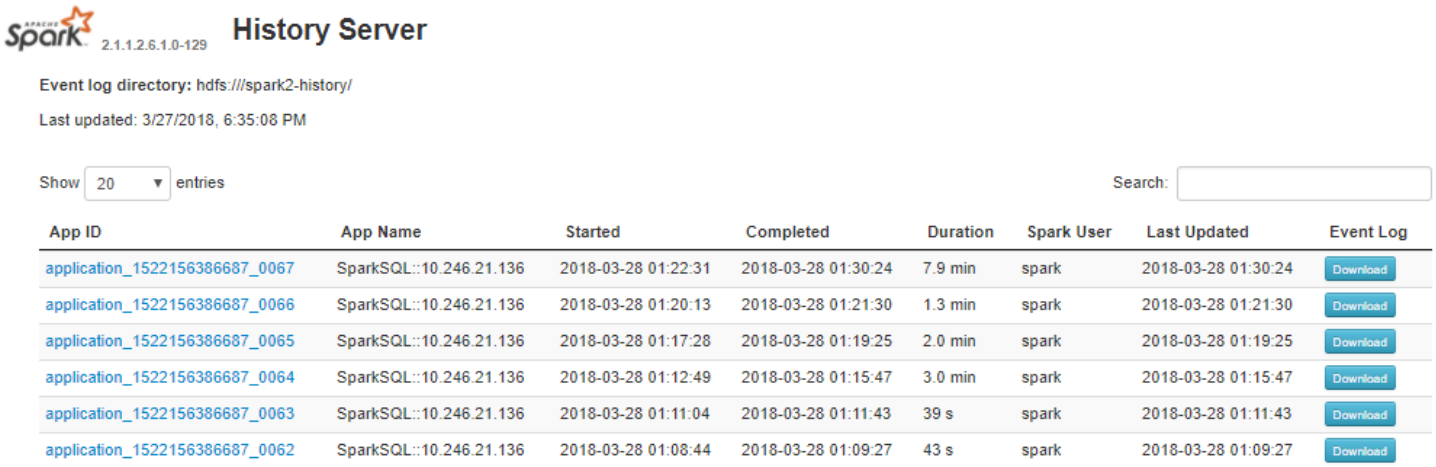


Figure 7 – Spark2 History Server UI

From the History Server UI you can quickly identify application start times, duration, user information, and you can download the event log if needed. If you select a given application id, you can get details on the associated Job ID's as shown below:

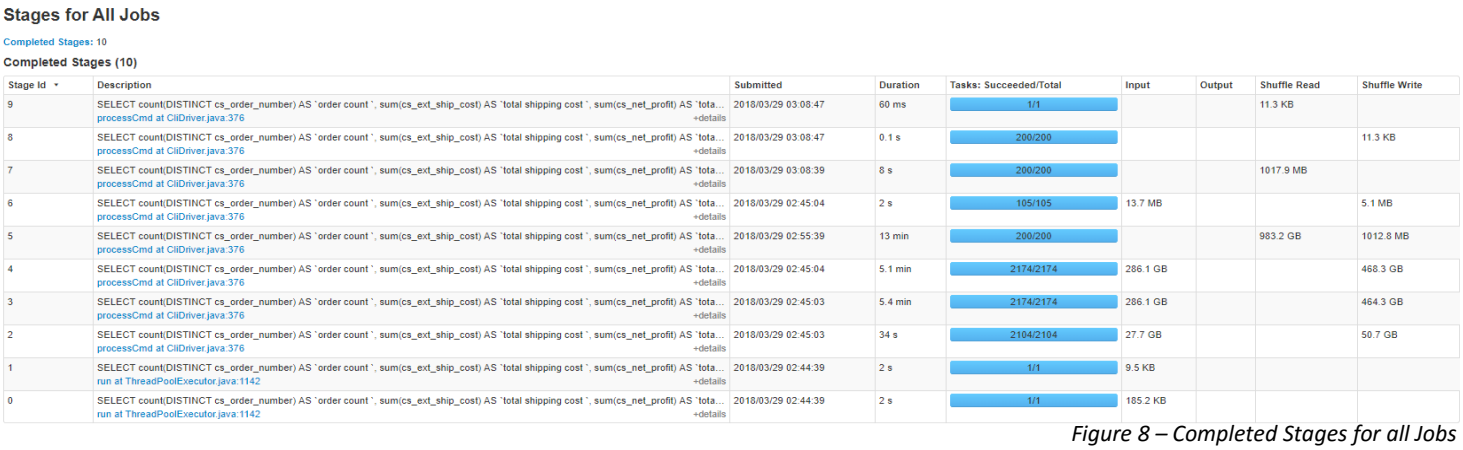


Figure 8 – Completed Stages for all Jobs

Figure 8 shows Spark Stages report from the Spark2 History Server UI. The key thing to note here are succeeded totals for all stages and duration times. Keep a look out for retries and failures on any stage. You want all stages to succeed without retries or failures, if you see any, you will likely have to revise your Spark configuration parameters for the associated job or see the application logs to help identify problems.

Another area of concern is Shuffling. Even if all stages are successful without failures, Shuffle is an expensive operation since it involves disk I/O, data serialization, and network I/O. When data does not fit in memory Spark will spill these tables to local disk, incurring the additional overhead of disk I/O and increased garbage collection. Operations which can cause a shuffle include repartition operations like *repartition* and *coalesce*, *ByKey* operations (except for counting) like *groupByKey* and *reduceByKey*, and join operations like *cogroup* and *join*.

Use the History UI to identify Spark jobs that are shuffle heavy. I specifically chose a Shuffle heavy job to include in Figure 7 above, you can see that Stage ID 5 is extremely shuffle heavy, almost 1TB in Shuffle Read and 1GB Shuffle Write with a duration of 13min. You can drill down into a stage to get additional metrics. Figure 9 shows the additional metrics for Stage 5.

Details for Stage 5 (Attempt 0)

Total Time Across All Tasks: 17.4 h
Locality Level Summary: Process local: 200
Shuffle Read: 983.2 GB / 9034383386
Shuffle Write: 1012.8 MB / 53689329

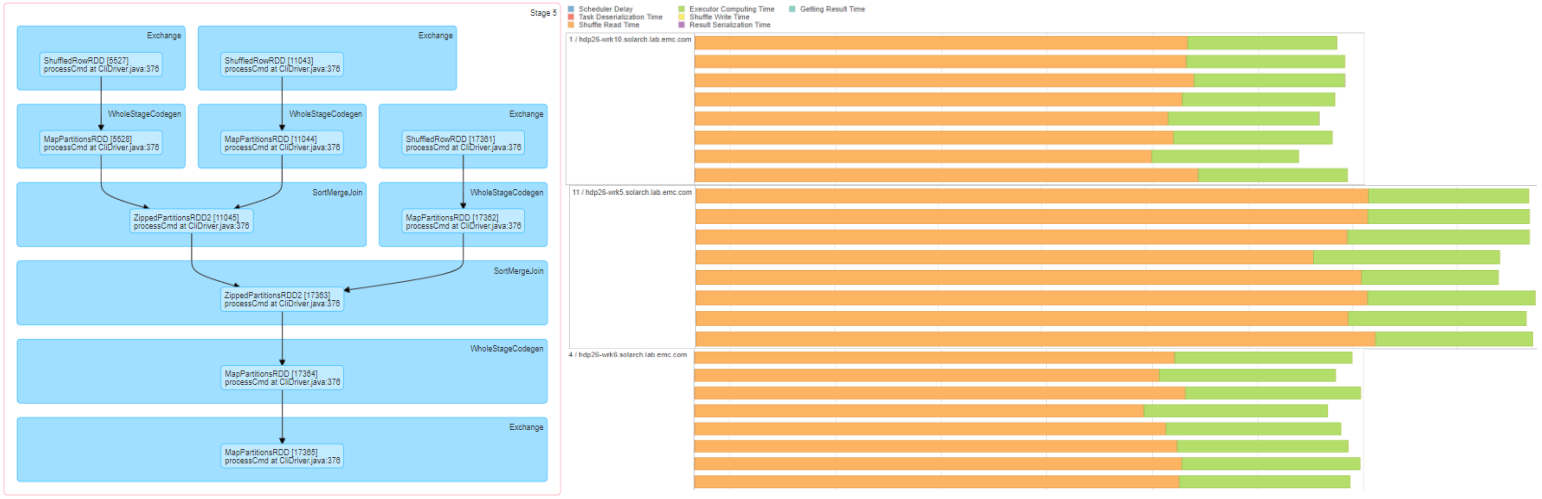
▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1.4 min	4.3 min	5.2 min	6.0 min	8.4 min
Scheduler Delay	0 ms	2 ms	5 ms	15 ms	30 ms
Task Deserialization Time	1 ms	2 ms	4 ms	0.1 s	0.2 s
GC Time	0.8 s	1 s	2 s	4 s	5 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	30 s	2.6 min	3.8 min	4.5 min	6.9 min
Shuffle Read Size / Records	4.9 GB / 45084730	4.9 GB / 45150900	4.9 GB / 45173100	4.9 GB / 45195307	4.9 GB / 45246414
Shuffle Remote Reads	4.5 GB	4.5 GB	4.5 GB	4.5 GB	4.5 GB
Shuffle Write Size / Records	5.0 MB / 264429	5.0 MB / 267631	5.1 MB / 268511	5.1 MB / 269256	5.1 MB / 272445

Figure 9 – Stage Details from Spark History Serer UI

"Shuffle Read Blocked Time" is the time that tasks spend in a blocked waiting state for shuffle data to be read from remote machines. Shuffling is bad enough, to have to wait to shuffle data to be read really has an impact on performance. You want the block time to be at 0, ranges in the minutes as shown here requires code optimization. The DAG for stage 5 as well as the Shuffle Read Time for some of the executors are shown below for reference:



For this job, the throughput on Isilon averaged around 2Gb/s per node (~8Gb/s for the cluster) and the CPU usage stayed well below 20%. This is well below the capacity and throughput for Isilon, the overhead is associated with shuffle read time from local disks increases job run time.

Figure 10 shows Isilon Cluster statistics from the Isilon OneFS GUI, it is helpful to monitor these statistics during Spark job runs to gain insights on storage loads and cluster utilization. Isilon Insight IQ is a free monitoring software package you can install that provides much more cluster statistics. I provide examples of Isilon IIQ in the benchmark results of this paper.

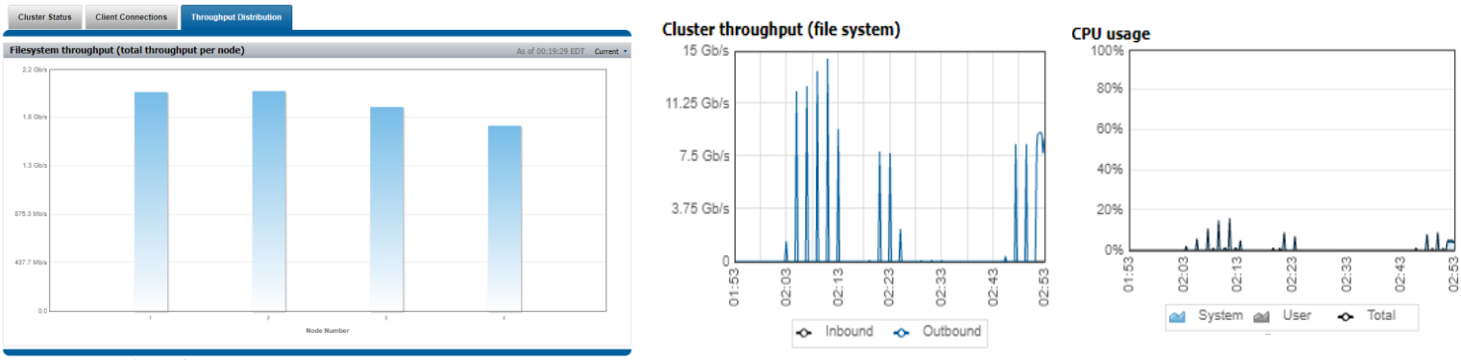


Figure 10 – Isilon Cluster Statistics

Figure 11 presents information on the Spark Environment. Whether you submitted the application or you're analyzing someone else's Spark job, knowing the settings used for each application run can be critical. The Spark Environment in the Spark2 History UI provides all the settings information for a given application run.

For example, cross joins are not enabled by default in Spark SQL, if you are creating a SQL script that will perform cross joins and the Spark configuration has not been updated, you will need to pass the configuration option ***spark.sql.crossJoin.enabled=true*** when executing your Spark SQL script. The Spark Environment will show all configuration settings, environment variables, version info, and etcetera for each application run. A snippet of the Spark Environment from the Spark History UI is shown below for reference.

spark.driver.memory	20G
spark.driver.port	39440
spark.eventLog.dir	hdfs:///spark2-history/
spark.eventLog.enabled	true
spark.executor.cores	8
spark.executor.extraLibraryPath	/usr/hdp/current/hadoc
spark.executor.id	driver
spark.executor.instances	11
spark.executor.memory	20G
spark.history.fs.logDirectory	hdfs:///spark2-history/
spark.master	yarn
spark.org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.param.PROXY_HOSTS	hdp26-master.solarch.
spark.org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter.param.PROXY_URI_BASES	http://hdp26-master.so
spark.scheduler.mode	FIFO
spark.sql.catalogImplementation	hive
spark.sql.crossJoin.enabled	true
spark.submit.deployMode	client
spark.ui.filters	org.apache.hadoop.ya
spark.yarn.driver.memoryOverhead	4096
spark.yarn.executor.memoryOverhead	4096
spark.yarn.historyServer.address	hdp26-master.solarch.
spark.yarn.queue	default

Figure 11 – Spark Environment Table from Spark History UI

The Spark History UI also provides Executors information. Here you can see the driver details and number of executors, task information, cores used, input size, garbage collection (GC) time, as well as shuffle information.

Long-running Spark jobs may consume a large amount of disk space. When using Spark with Yarn, the temporary storage space for Spark is the same as the local directories defined in the Yarn configuration. Make sure to create many local directories for Yarn on different drives for each compute

node as stated in the *Recommended Yarn Configuration* section presented earlier. The key is to spread the Spark Shuffle Space out as much as possible to help limit the impact of shuffling to job performance.

When looking at the Executors Report in the History UI (Figure 12), make sure you Task Time is spread evenly across all the executors. Make sure there are no failed tasks. If you ever see the driver listed by itself with no executors, the job was not run in yarn mode, i.e. the job was run in default mode without the ***–master yarn*** option. Jobs can run in driver mode only, but performance results will be better using multiple Executors with YARN. Make sure all the executors are listed and the status show active, if you see any dead executors listed, it could be due to something as simple as a node or Spark process restart, but in cases where an out-of-memory situation occurred, executors can die. You may have to increase *spark.memoryOverhead* or repartition the data or quite possibly grow your cluster if needed. Check the error logs for additional information.

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(12)	104	4.9 MB / 135.2 GB	0.0 B	88	88	0	2306	2394	1.0 h (1.4 min)	20.3 GB	445.1 MB	42.6 GB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(12)	104	4.9 MB / 135.2 GB	0.0 B	88	88	0	2306	2394	1.0 h (1.4 min)	20.3 GB	445.1 MB	42.6 GB

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.246.21.136:35280	Active	15	535.6 KB / 11.3 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	hdp26-wrk11.solarch.lab.emc.com:34046	Active	9	425.5 KB / 11.3 GB	0.0 B	8	8	0	265	273	5.5 min (6 s)	1.7 GB	36.5 MB	3.7 GB	stdout stderr	Thread Dump
2	hdp26-wrk4.solarch.lab.emc.com:37895	Active	9	425.5 KB / 11.3 GB	0.0 B	8	8	0	256	264	5.7 min (8 s)	1.8 GB	51.3 MB	3.8 GB	stdout stderr	Thread Dump
3	hdp26-wrk7.solarch.lab.emc.com:54873	Active	6	341.8 KB / 11.3 GB	0.0 B	8	8	0	161	169	5.6 min (8 s)	1.9 GB	57.6 MB	3.9 GB	stdout stderr	Thread Dump
4	hdp26-wrk3.solarch.lab.emc.com:53954	Active	6	341.8 KB / 11.3 GB	0.0 B	8	8	0	148	156	5.6 min (8 s)	1.8 GB	28.9 MB	3.8 GB	stdout stderr	Thread Dump
5	hdp26-wrk2.solarch.lab.emc.com:59207	Active	10	459.2 KB / 11.3 GB	0.0 B	8	8	0	247	255	5.6 min (9 s)	1.8 GB	34.9 MB	3.9 GB	stdout stderr	Thread Dump
6	hdp26-wrk8.solarch.lab.emc.com:58032	Active	6	341.8 KB / 11.3 GB	0.0 B	8	8	0	156	164	5.7 min (9 s)	1.9 GB	36 MB	4 GB	stdout stderr	Thread Dump
7	hdp26-wrk6.solarch.lab.emc.com:49150	Active	9	425.5 KB / 11.3 GB	0.0 B	8	8	0	237	245	5.7 min (8 s)	1.9 GB	39.2 MB	3.9 GB	stdout stderr	Thread Dump
8	hdp26-wrk5.solarch.lab.emc.com:35816	Active	6	341.8 KB / 11.3 GB	0.0 B	8	8	0	154	162	5.7 min (8 s)	1.9 GB	36 MB	3.9 GB	stdout stderr	Thread Dump
9	hdp26-wrk10.solarch.lab.emc.com:59543	Active	9	425.5 KB / 11.3 GB	0.0 B	8	8	0	218	226	5.6 min (8 s)	1.8 GB	60.5 MB	3.8 GB	stdout stderr	Thread Dump
10	hdp26-wrk1.solarch.lab.emc.com:41400	Active	9	425.5 KB / 11.3 GB	0.0 B	8	8	0	246	254	5.7 min (8 s)	2 GB	25.9 MB	4.1 GB	stdout stderr	Thread Dump
11	hdp26-wrk9.solarch.lab.emc.com:41403	Active	10	459.2 KB / 11.3 GB	0.0 B	8	8	0	218	226	5.7 min (8 s)	1.8 GB	38.3 MB	3.8 GB	stdout stderr	Thread Dump

Figure 12– Executors Report from Spark History UI

The next section covers various Spark benchmarks with Isilon as the backend data store.

Spark Benchmarks with Isilon

The benchmarks presented in this paper were generated in a lab environment using previous generation PowerEdge servers and not the newer generation PowerEdge servers shown in Figure 2 – Networking Diagram Solution Example, a single Gen6 Isilon H600 chassis with four nodes was configured to support twelve PowerEdge R730xd servers.

Hardware specifications are the same for each of the twelve R730xd servers used to conduct the benchmark testing – 2 Intel Xeon CPUs (40 Cores Total) @ 2.2GHz, 256G RAM, 25 x 1TB SAS HDD, 4 port 10G Intel network card. The Isilon Gen6 model tested is a single H600-4U-Chassis with 4 nodes, each node contains 256GB, 1GE for Management, 2x40GE SFP+ front-end interfaces, 2x10G IB back-end interfaces, 30 x 1TB SAS HHD, 1x6TB SSD. Hortonworks HDP version 2.6 with YARN version 2.7.3 and Spark version 2.1.1 provides the Hadoop/Spark software stack for the test cluster.

For all the benchmarks presented in this paper, it's important to remember that data storage has been separated from all compute resources with Isilon, i.e. all data is stored and provided by Isilon. Local storage on each compute node is for the Operating System and intermediary storage used by Hadoop/Spark only, e.g. /tmp, logging directories, shuffle space, etc.

TPC-DS Benchmark

The TPC Benchmark DS (**TPC-DS**) is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. TPC-DS with a scale factor of 3000 (3TB size) was configured on the Dell EMC test cluster, ORC format was used to store the data on Isilon H600.

Spark version 2.1.1 ran all 99 TPC-DS test queries with Hive version 1.2.1000. The list of TPC-DS SQL queries can be found at the Apache Spark GitHub for reference. The SQL parser and sub-query support added in Spark 2 makes it possible to run all 99 queries. Spark 1.6, also available with Hortonworks HDP 2.6, can only run approximately half the number of queries. If your require ANSI SQL compatibility, don't waste your time with Spark 1.6, start directly with Spark 2. The 3TB Spark2 TPC-DS benchmark results for Isilon H600 with PowerEdge R730xd are shown in Figure 13.

Query #	Time (sec)	Query #	Time (sec)	Query #	Time (sec)	Query #	Time (sec)	Query #	Time (sec)
Q1	44	Q21	15	Q43	55	Q62	33	Q81	73
Q2	59	Q24	26	Q44	108	Q63	60	Q82	74
Q3	80	Q25	451	Q45	95	Q64	756	Q83	43
Q4	819	Q26	85	Q46	53	Q65	114	Q84	59
Q5	266	Q27	116	Q47	110	Q66	102	Q85	134
Q6	166	Q28	230	Q48	146	Q67	618	Q86	30
Q7	105	Q29	365	Q49	178	Q68	68	Q87	111
Q8	64	Q30	67	Q50	269	Q69	653	Q88	161
Q9	461	Q31	206	Q51	88	Q70	110	Q89	41
Q10	787	Q32	69	Q52	56	Q71	159	Q90	34
Q11	407	Q33	108	Q53	63	Q72	1059	Q91	29
Q12	25	Q34	44	Q54	165	Q73	39	Q92	39
Q13	300	Q35	790	Q55	55	Q74	298	Q93	410
Q15	166	Q36	96	Q56	90	Q75	298	Q94	604
Q16	1128	Q37	57	Q57	205	Q76	77	Q95	459
Q17	371	Q38	151	Q58	63	Q77	104	Q96	33
Q18	124	Q40	147	Q59	42	Q78	639	Q97	105
Q19	99	Q41	9	Q60	107	Q79	64	Q98	39
Q20	43	Q42	54	Q61	92	Q80	588	Q99	31

Figure 13 – 3TB TPC-DS Benchmark with Isilon H600 and 12 x PowerEdge R730xd servers

Spark MLlib Benchmarks

MLlib is Spark's machine learning library, focusing on learning algorithms and utilities, including collaborative filtering, classification, regression, clustering, dimensionality reduction, as well as underlying optimization primitives. Spark MLlib Supports writing applications in Java, Scala, or Python. All the benchmarks shown here have all the input and output data stored on Isilon, local disk storage on the compute nodes are used for shuffling and temporary files.

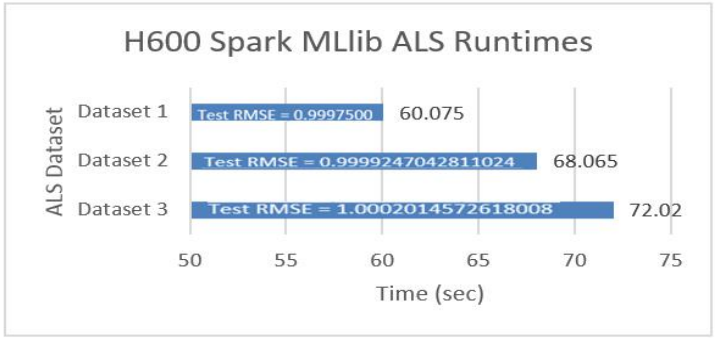
Alternating Least Squares

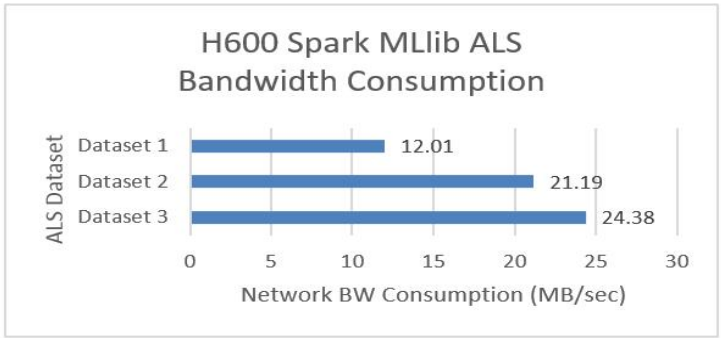
Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. Spark MLlib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. Spark MLlib uses the alternating least squares (ALS) algorithm to learn these latent factors. The implementation in spark.ml has the following parameters:

- *numBlocks* is the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10).
- *rank* is the number of latent factors in the model (defaults to 10).
- *maxIter* is the maximum number of iterations to run (defaults to 10).
- *regParam* specifies the regularization parameter in ALS (defaults to 1.0).
- *implicitPrefs* specifies whether to use the *explicit feedback* ALS variant or one adapted for *implicit feedback* data (defaults to false which means using *explicit feedback, implicit feedback* is enabled for the benchmark shown below).
- *alpha* is a parameter applicable to the implicit feedback variant of ALS that governs the *baseline* confidence in preference observations (defaults to 1.0).
- *nonnegative* specifies whether or not to use nonnegative constraints for least squares (defaults to false).

Note: The DataFrame-based API for ALS currently only supports integers for user and item ids. Other numeric types are supported for the user and item id columns, but the ids must be within the integer value range.

Figure 14 shows benchmark results for running ALS against product recommendation data stored on Isilon H600 NAS using Spark MLlib for different data sets. Root Mean Square Error (**RMSE**) measures how much error there is between two data sets. In other words, it compares a predicted value (test data set) and an observed or known value (training data set). As the sample size increases (assuming good data), the RMSE usually improves as shown below.





Dataset 1	Dataset 2	Dataset 3
NUM_ITERATIONS_ALS=20	NUM_ITERATIONS_ALS=20	NUM_ITERATIONS_ALS=20
NUM_PRODUCTS_ALS=40000	NUM_PRODUCTS_ALS=60000	NUM_PRODUCTS_ALS=60000
NUM_RECOMMENDS_ALS=20	NUM_RECOMMENDS_ALS=20	NUM_RECOMMENDS_ALS=20
NUM_USERS_ALS=30000	NUM_USERS_ALS=40000	NUM_USERS_ALS=50000
Training set: 41924313, Test set: 17957187	Training set: 84103306, Test set: 36036704	Training set: 105416981, Test set: 45184715

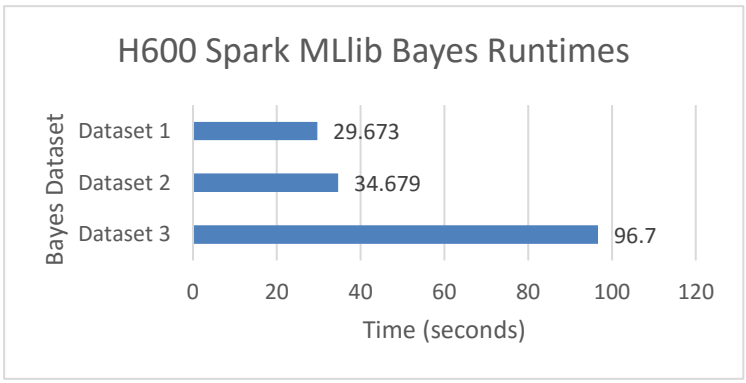
Figure 14 – Isilon H600 Spark MLlib ALS Test Results

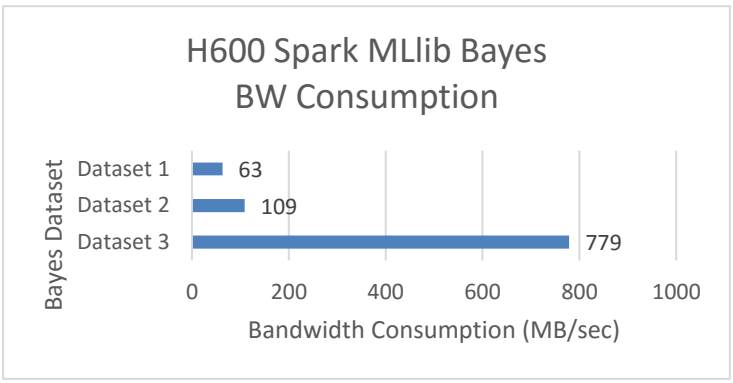
Naive Bayes

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.

Spark MLlib supports multinomial naive Bayes and Bernoulli naive Bayes. These models are typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes). Feature values must be nonnegative. The model type is selected with an optional parameter "multinomial" or "bernoulli" with "multinomial" as the default. Additive smoothing can be used by setting the parameter λ (the default is 1).

Figure 15 presents the benchmark test for Bayes machine learning algorithm on Isilon H600 using Spark MLlib with different data set sizes determined by number of pages, classes, and n-grams. An ***n*-gram** is a contiguous sequence of *n* items from a given sample of text and classes corresponds to the text categories in the data set. The test workload uses generated documents whose words follow the zipfian distribution.





Bayes Dataset 1

Classes	100
N-grams	2
Pages	500K

Bayes Dataset 2

Classes	100
N-grams	2
Pages	1M

Bayes Dataset 3

Classes	20000
N-grams	2
Pages	20M

Figure 15 – Isilon H600 Spark MLlib Naive Bayes Test Results

Linear Regression

Spark's Generalized Linear Regression interface allows for flexible specification of general linear models (GLMs) which can be used for various types of prediction problems including linear regression, Poisson regression, logistic regression, and others. Currently, Spark MLlib only supports a subset of the exponential family distributions, Figure 16 shows the current supported distributions:

Family	Response Type	Supported Links
Gaussian	Continuous	Identity*, Log, Inverse
Binomial	Binary	Logit*, Probit, CLogLog
Poisson	Count	Log*, Identity, Sqrt
Gamma	Continuous	Inverse*, Idenity, Log
Tweedie	Zero-inflated continuous	Power link function

* Canonical Link

Figure 16 - Supported Exponential Distributions for Spark MLlib Linear Regression Interface

Spark currently supports up to 4096 features through its Generalized Linear Regression interface, Spark will throw an exception if this constraint is exceeded. For linear and logistic regression, models with an increased number of features can be trained using the Linear Regression and Logistic Regression estimators. The three datasets generated for Linear Regression testing with Isilon contained 100K features/1M examples (800 GB dataset), 80K features/500K examples (320 GB dataset), and 50K features/300K examples (120 GB dataset).

Figure 17 shows the job run times and throughput of the Spark MLlib Linear Regression tests on the three datasets stored on Isilon H600.

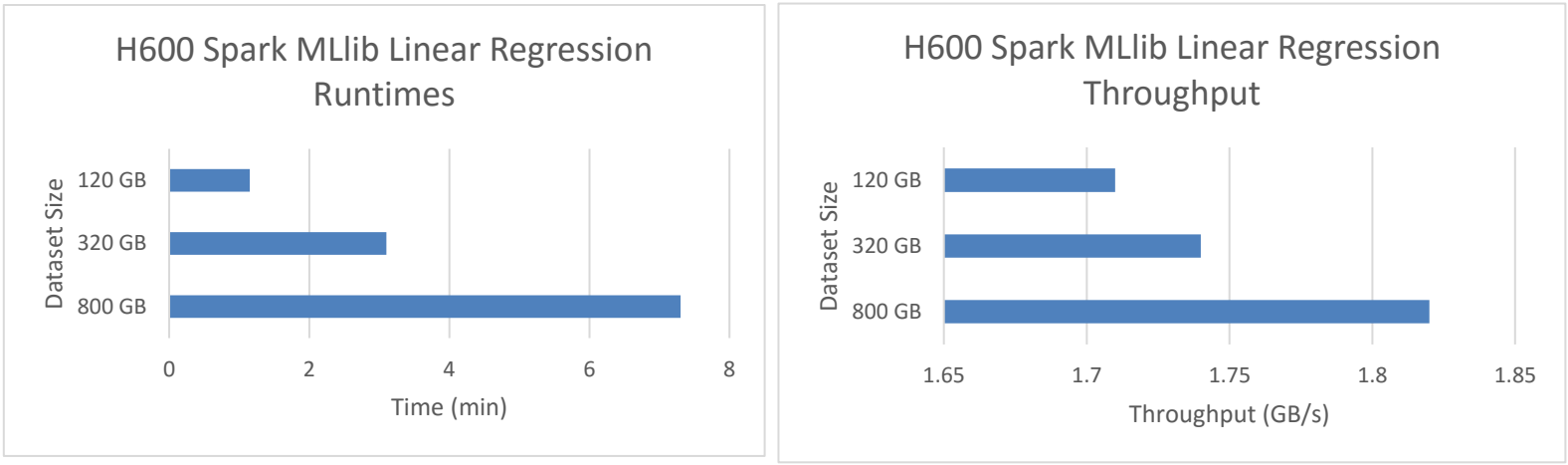


Figure 17 – Isilon H600 Spark MLlib Linear Regression Test Results (iterations=100, stepSize=.00001)

The previous Spark MLlib workloads are primarily CPU and Memory bound workloads. The Linear Regression tests provide a good example of how Spark MLlib can also create significant I/O demand on storage. Figure 18 provides Insight IQ graphs showing how well Isilon provides load balancing and I/O throughput for the 800 GB Linear Regression Spark MLlib job.

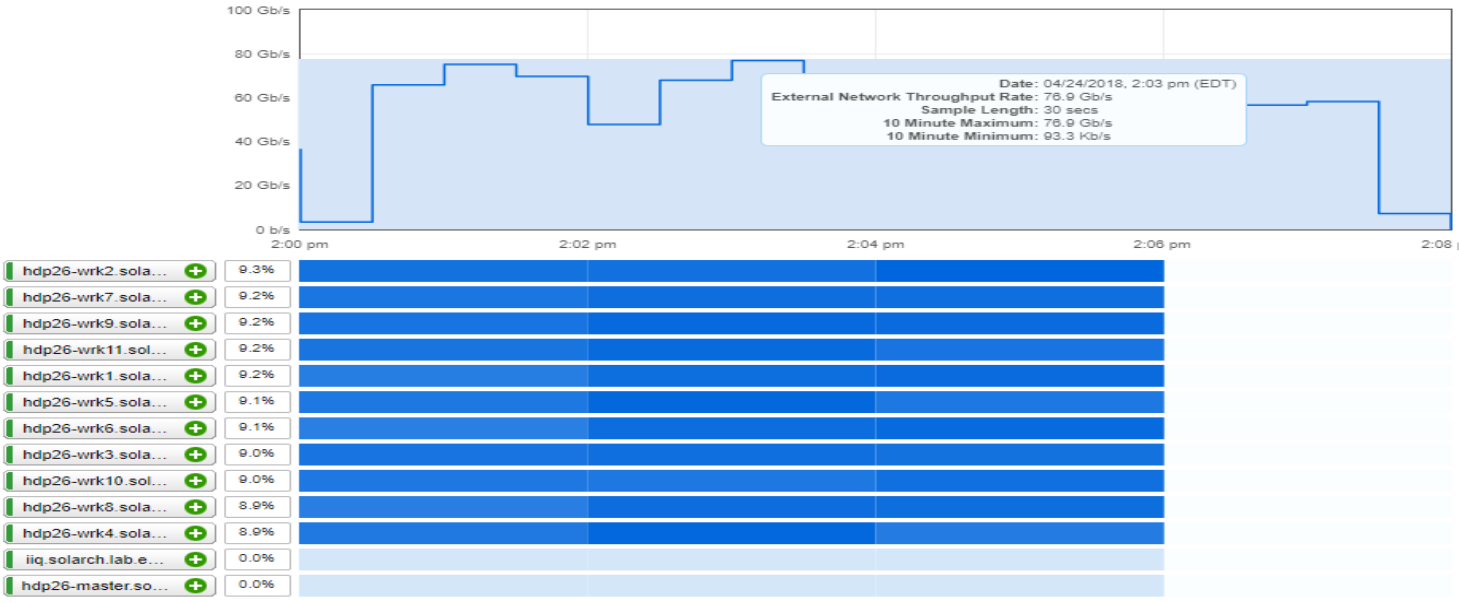


Figure 18 – Isilon H600 Cluster Throughput for 800 GB Spark MLlib Linear Regression

Isilon Smart Connect provides excellent parallel I/O load balancing as shown in Figure 18. All eleven Spark clients are evenly load balanced at 9% of the ~80 Gb/s aggregate throughput observed for the Spark MLlib Linear Regression job.

The distribution of the Spark clients during the 800 GB Spark MLlib Linear Regression job run is shown in Figure 19.

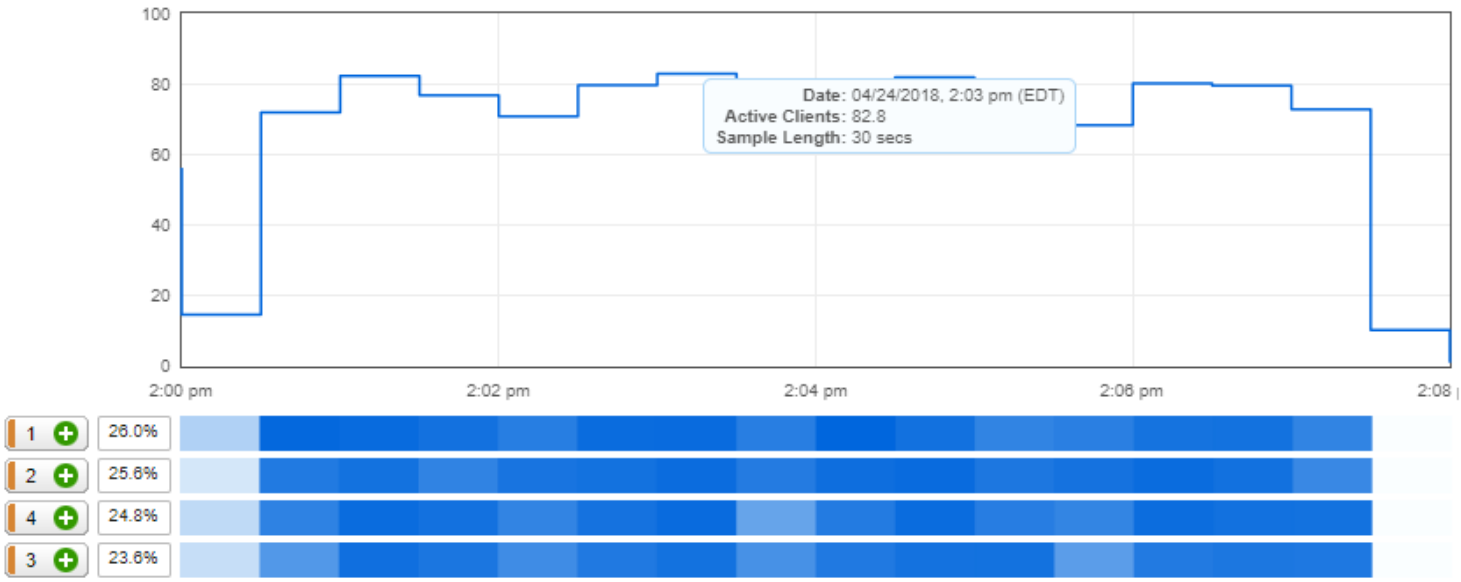


Figure 19 – Spark Client Distribution on Isilon H600 during Spark MLlib Linear Regression

The average number of active Spark clients during the Spark MLlib Linear Regression on Isilon was ~82. Each of the four Isilon H600 nodes handled ~25 percent of the active client connections. The Spark configuration for the Linear Regression job utilized 44 executors, 8 cores, 20 GB of RAM, and 200 partitions. The observed load balancing is attributed to the Isilon OneFS Smart Connect feature that is unique to Dell EMC Isilon and a key software component to provide the high level of I/O throughput presented in Figure 18.

As I/O requirements increase, simply add another Isilon chassis and the storage capacity and throughput immediately increase with no configuration changes required in the Hadoop/Spark cluster. This is a key value proposition of the Dell EMC Isilon solution for Hadoop and Spark deployments.

Figure 20 shows the total number of connections observed from the Spark cluster to Isilon during the Spark MLlib Linear Regression job.

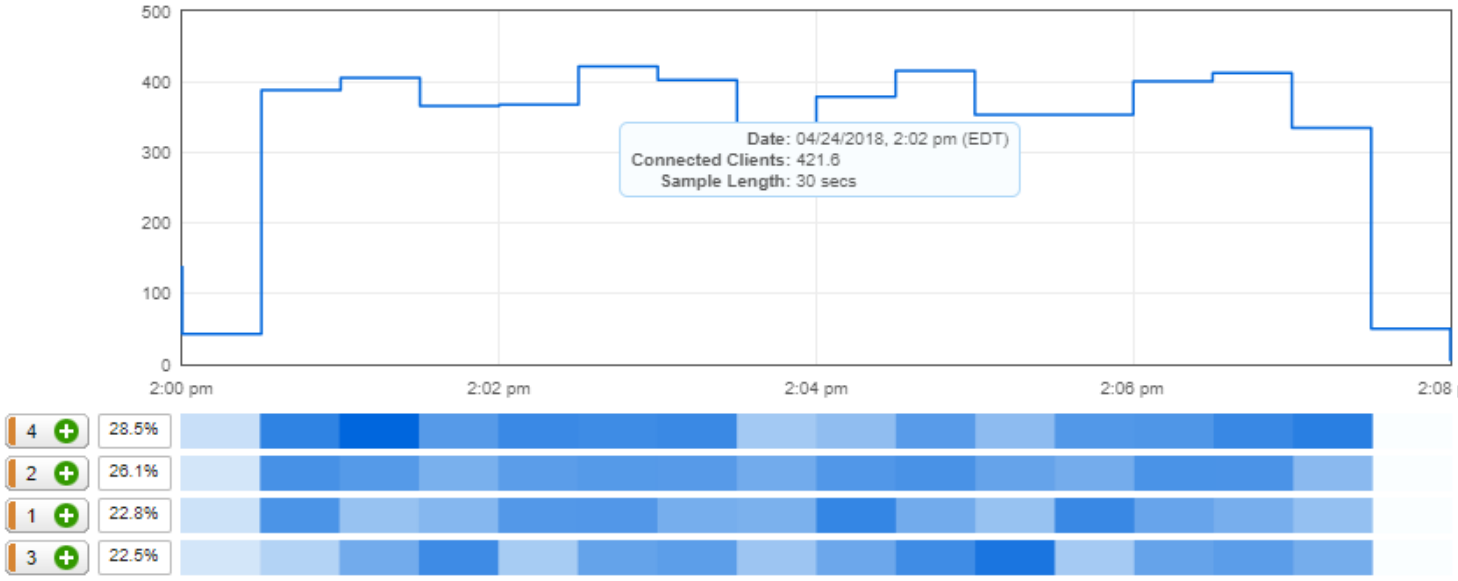


Figure 20 – Total number of Spark client connections to Isilon during Spark MLlib Linear Regression

It's interesting to see the total number of client connections during Spark job runs. Although there are 11 physical Spark clients, the number of client connections can be much higher as shown in Figure 20. During the Spark MLlib Linear Regression job, over 400 connections where observed on Isilon. The Spark parameters used for each Spark job dictate the overall client connections.

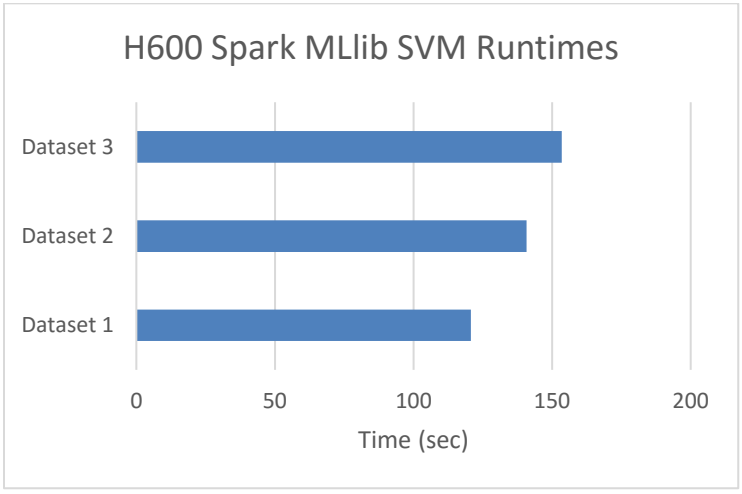
Support Vector Machine

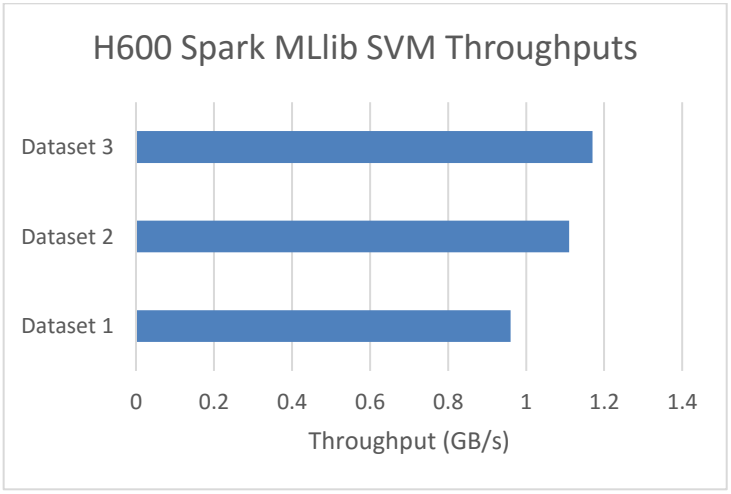
The most common classification type is binary classification, where there are two categories, usually named positive and negative. Spark MLlib supports linear Support Vector Machines (SVMs) with L1 and L2 regularized variants. The purpose of the regularizer is to encourage simple models and avoid overfitting. Overfitting is often caused by having too many factors or parameters compared to the number of data points. For additional information on SVM (with R-code), see my R Publication. Figure 21 provides information on L1 and L2 regularizers supported by Spark MLlib for SVM. Sign (**w**) is a vector consisting of the signs (±1) of all the entries of **w** (the variable vector).

	regularizer $R(\mathbf{w})$	**gradient or sub-gradient**
zero (unregularized)	0	$\mathbf{0}$
L2	$\frac{1}{2}\|\mathbf{w}\|_2^2$	$\mathbf{w}$
L1	$\|\mathbf{w}\|_1$	$\mathrm{sign}(\mathbf{w})$

Figure 21 – Spark MLlib supported Regularizers for SVM

The SVM testing results for Isilon H600 is shown below in Figure 22.





Dataset 1: 120K Examples, 300K Features Dataset 2: 140K Examples, 300K Features Dataset 3: 150K Examples, 300K Features

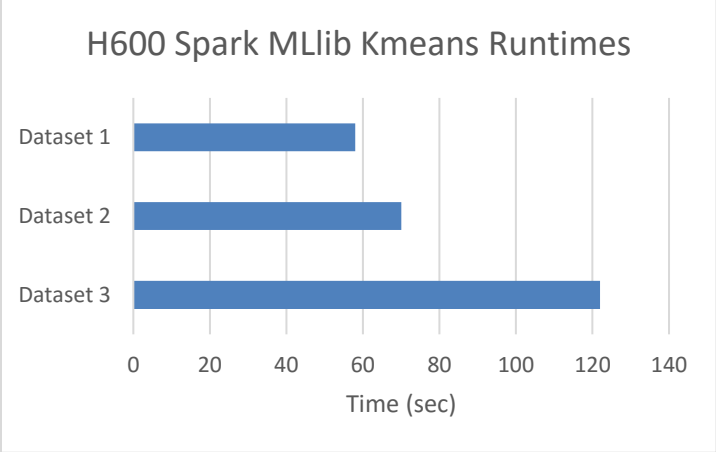
Figure 22 – Isilon H600 Spark MLlib SVM Test Results

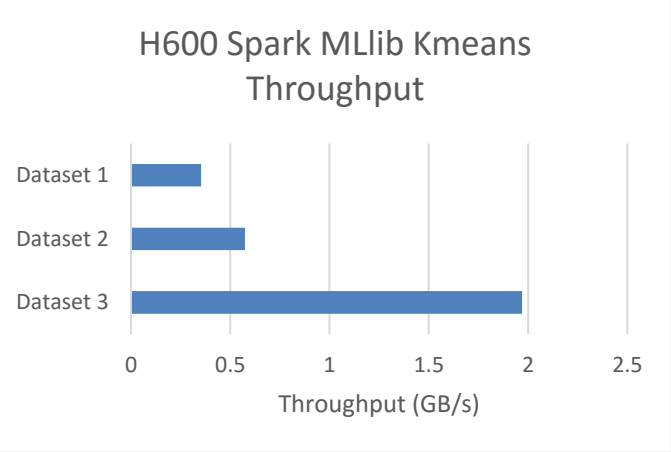
K-means Clustering

K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The implementation in Spark MLlib has the following parameters:

- *k* is the number of desired clusters. Note that it is possible for fewer than k clusters to be returned, for example, if there are fewer than k distinct points to cluster.
- *maxIterations* is the maximum number of iterations to run.
- *initializationMode* specifies either random initialization or initialization via k-means||.
- *initializationSteps* determines the number of steps in the k-means|| algorithm.
- *epsilon* determines the distance threshold within which we consider k-means to have converged.
- *initialModel* is an optional set of cluster centers used for initialization. If this parameter is supplied, only one run is performed.

The input data on Isilon is based on Uniform Distribution and Guassian Distribution. The Kmeans testing results for Isilon H600 is shown below in Figure 23.





Dataset 1

Clusters	5
Samples	100000000
Samples Per File	20000000
k	10
Dimensions	20
Max Iterations	5
Epsilon	0.5

Dataset 2

Clusters	5
Samples	200000000
Samples Per File	40000000
k	10
Dimensions	20
Max Iterations	5
Epsilon	0.5

Dataset 3

Clusters	5
Samples	1200000000
Samples Per File	40000000
k	10
Dimensions	20
Max Iterations	10
Epsilon	0.5

Figure 23 – Isilon H600 Spark MLlib Kmeans Test Results

Spark GraphX Benchmarks

GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

The property graph is a directed multigraph with user defined objects attached to each vertex and edge. A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex. The ability to support parallel edges simplifies modeling scenarios where there can be multiple relationships (e.g., co-worker and friend) between the same vertices. Each vertex is keyed by a *unique* 64-bit long identifier (VertexId). GraphX does not impose any ordering constraints on the vertex identifiers. Similarly, edges have corresponding source and destination vertex identifiers.

NWeight is used to test Spark GraphX with Isilon. NWeight is an iterative graph-parallel algorithm implemented by Spark GraphX and pregel. The algorithm computes associations between two vertices that are n-hop away. Three NWeight configurations were constructed for the Spark GraphX tests.

NWeight Configuration 1:	NWeight Configuration 2:	NWeight Configuration 3:
Number of edges : 1000000	Number of edges : 10000000	Number of edges : 100000000
Number of degrees: 3	Number of degrees: 3	Number of degrees: 3
Max Out Edges: 30	Max Out Edges: 30	Max Out Edges: 30

Figure 24 show the Spark GraphX test results.

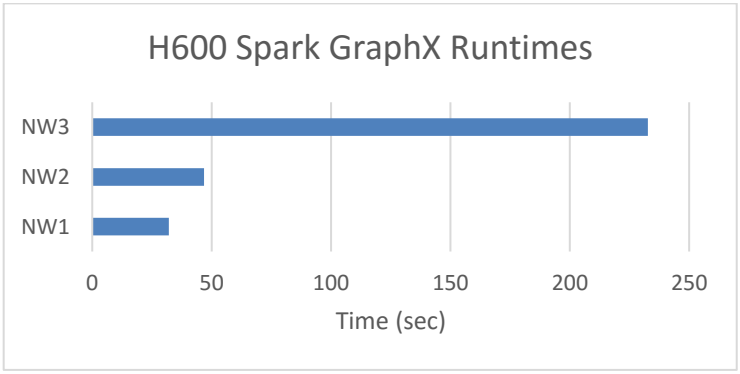


Figure 24 – Isilon H600 Spark GraphX Test Results

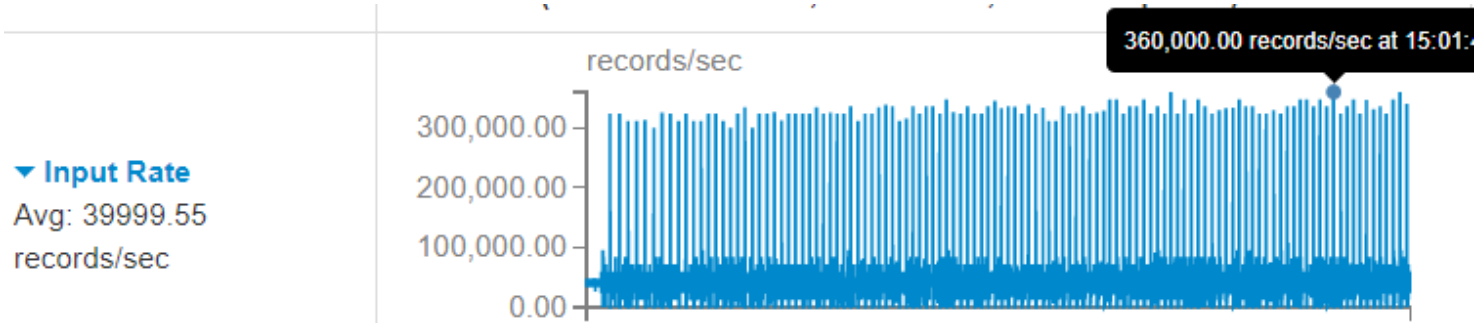
Spark Streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window.

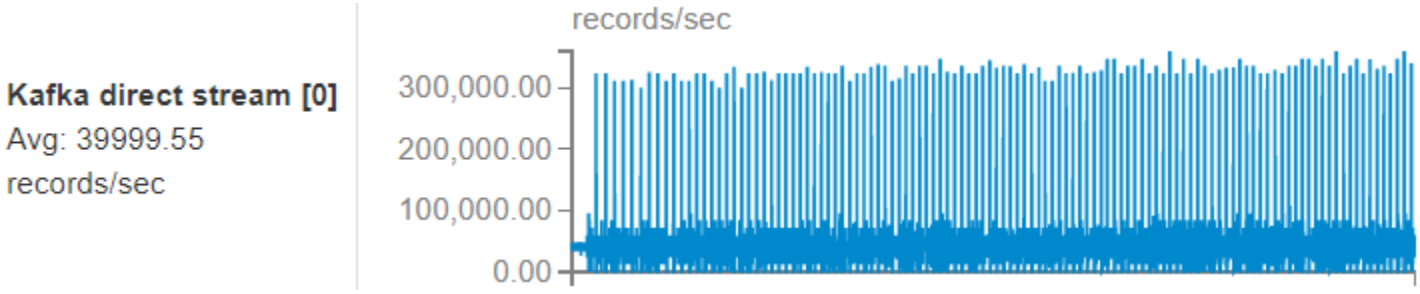
To test Spark Streaming with Isilon, a continuous Kafka stream was created, the Spark Streaming workload counts the words cumulatively every few seconds from the Kafka stream. The details of the interval span, records per interval, record length, and number of Kafka Producers are shown below:

```
Interval Span        : 30 ms
Record Per Interval  : 600
Record Length        : 200 bytes
Producer Number      : 2
Total Records         : -1 [Infinity]
Total Rounds          : -1 [Infinity]
Kafka Topic           : wordcount
```

The average input rate into Spark Streaming is approximately 40,000 records/sec with peak streams reaching 360,000 records/sec as shown below:



The Kafka direct stream [0] is shown below for reference, no records where lost between Kafka and Spark Streaming.



Conclusions

Dell EMC Isilon Scale-out NAS provides a high-performance solution to efficiently separate storage from compute for both Hadoop and Spark cluster deployments. Various Spark benchmarks and testing scenarios were conducted with Isilon and the results show that all Spark API's (SQL, MLlib, GraphX, and Streaming) work and perform well with Dell EMC Isilon. Also presented are hardware and software configuration guidelines to help ensure successful Hadoop and Spark deployments with Isilon.