Microsoft SQL Server 2019
Case Study:
SQL Workloads running on Apache Spark in MS SQL Server 2019 Big Data Cluster

Technical White Paper
Published: November 2019
Applies to: Microsoft SQL Server 2019 Big Data Cluster

Abstract

In October 2019, Microsoft and Intel conducted performance and scalability testing using workloads based on TPC-DS Schema with data sets 1TB, 3TB, 10TB, 30TB, and 100TB running on the first Microsoft SQL Server 2019 Big Data Cluster solution, utilizing Apache Spark. We showcase the ability of Microsoft SQL Server 2019 Big Data Cluster running on Intel-powered platforms to handle Big Data Sets at various data sizes.

This white paper presents the definitions of these configurations and the benefits that Microsoft SQL Server 2019 brings as a solution for your Big Data problems at scale. It is confirming that Microsoft SQL Server 2019 Big Data Cluster is your choice for Big Data storage and processing large volumes of data and workloads. For your review, we detail the cluster environment, storage, workload, and Microsoft SQL Server 2019 Big Data Cluster configurations.

Table of Contents

Introduction

Big data refers to the large, diverse sets of information that grow at ever-increasing rates. (www.investopedia.com)

Big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time. (wikipedia.org)

The ever-evolving digital world is rapidly scaling the demands for flexible compute, networking, and storage. Future workloads will necessitate infrastructures that can seamlessly scale to support immediate responsiveness and widely diverse performance requirements. The exponential growth of data generation and consumption require that your data centers urgently evolve – or left behind in a highly competitive environment. These demands are driving the architecture of modernized, future-ready data centers and networks that can quickly fix and scale.

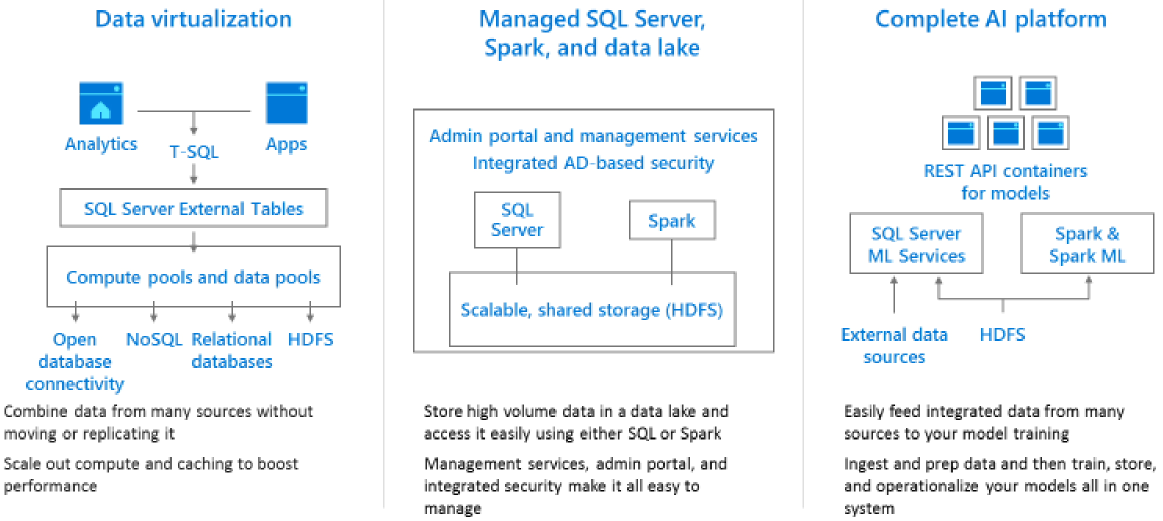
With an increasing amount of data, there is an increasing demand for flexibility to use the data from various sources. Microsoft SQL Server 2019 Big Data Cluster integrates Microsoft SQL Server and the best of big data open-source solutions. It deploys today's big data solutions on scalable clusters using Spark, HDFS containers with Kubernetes and SQL Server. This is Microsoft SQL Big Data Cluster response to offer a perfect balance of cutting-edge software and hardware, performance and scalability, deployment efficiency and simplified data management/analysis. It enables intelligence overall customers' data and represents the best platform to securely manage your big data at all data sets.

This paper showcases Microsoft SQL Server 2019 Big Data Cluster as a choice to answer your questions about finding the platform to store, manage, and process big data sets. In this study, we are providing insights into two systems to address ever-increasing data demands: Fueled by Intel® Xeon® processors and Intel® Data Center Storage Solutions, we put Microsoft SQL Server Big Data Cluster to test.

Technology

Microsoft SQL Server 2019 Big Data Cluster

Microsoft SQL Server 2019 Big Data Cluster is a versatile platform that seamlessly meets the requirements of the ever-expanding data sets. Its first version is built on top of Kubernetes to offer extreme scalability with today's best orchestration. With embedded HDFS storage, its elastic solution leverages large volumes of structured and unstructured data, while the best in class Microsoft SQL Server engine processes the relational data sets. Thanks to tuned integration with Kubernetes, Microsoft SQL Server 2019 Big Data Cluster is the ideal Big Data solution for AI, ML, M/R, Streaming, BI, T-SQL, and Spark.


Data virtualization
Analytics
T-SQL
Apps
SQL Server External Tables
Compute pools and data pools
Open database connectivity
NoSQL
Relational databases
HDFS
Managed SQL Server, Spark, and data lake
Admin portal and management services
Integrated AD-based security
SQL Server
Spark
Scalable, shared storage (HDFS)
Complete AI platform
REST API containers for models
SQL Server ML Services
Spark & Spark ML
External data sources
HDFS

Combine data from many sources without moving or replicating it

Scale out compute and caching to boost performance

Store high volume data in a data lake and access it easily using either SQL or Spark

Management services, admin portal, and integrated security make it all easy to manage

Easily feed integrated data from many sources to your model training

Ingest and prep data and then train, store, and operationalize your models all in one system

Figure 1: Microsoft SQL Server 2019 Big Data Cluster and Analytics Features

The Big Data Cluster's beating heart and brain is Microsoft SQL Server 2019, which creates the perfect environment for marriage between structured and unstructured data. The simplified deployment with containers and Kubernetes is putting the elasticity and the portability at the core of the platform and enabling easy on-prem and on-cloud deployments. The development and management experience are consistent regardless of where you run: on-prem or any of the major cloud providers.

As Big Data refers to decision support at scale, we have deployed today's best decision support benchmark, based on TPC-DS Schema, on two reference clusters. Between the two configurations, we deployed 1TB, 3TB, 10TB, 30TB, and 100TB data sets to challenge our Microsoft SQL Server 2019 Big Data Cluster deployments. With this document, we want to present these use cases and our current findings for your reviews before your deployments.

Before preparing the deployment, we should initially consider:

1- The Microsoft SQL Server 2019 Big Data architecture and its components (see Figure 2),
2- How the control, data pool, storage pool and compute pool components are laid out on the actual cluster master(s) and worker nodes (see Figure 3),
3- Specifically, how pools get composed of these functional pods (see Figure 4).

We provide insights for deploying SQL Server Big Data Cluster in our environments based on these considerations throughout the paper.

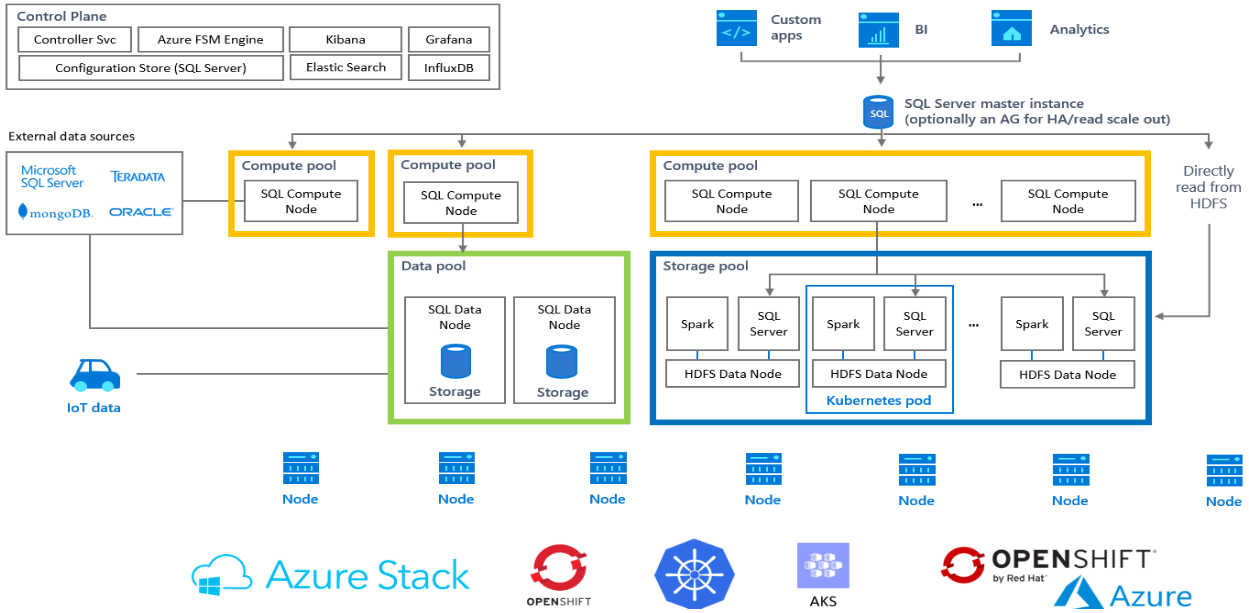


Figure 2: Microsoft SQL Server 2019 Big Data Cluster Architecture – Overview

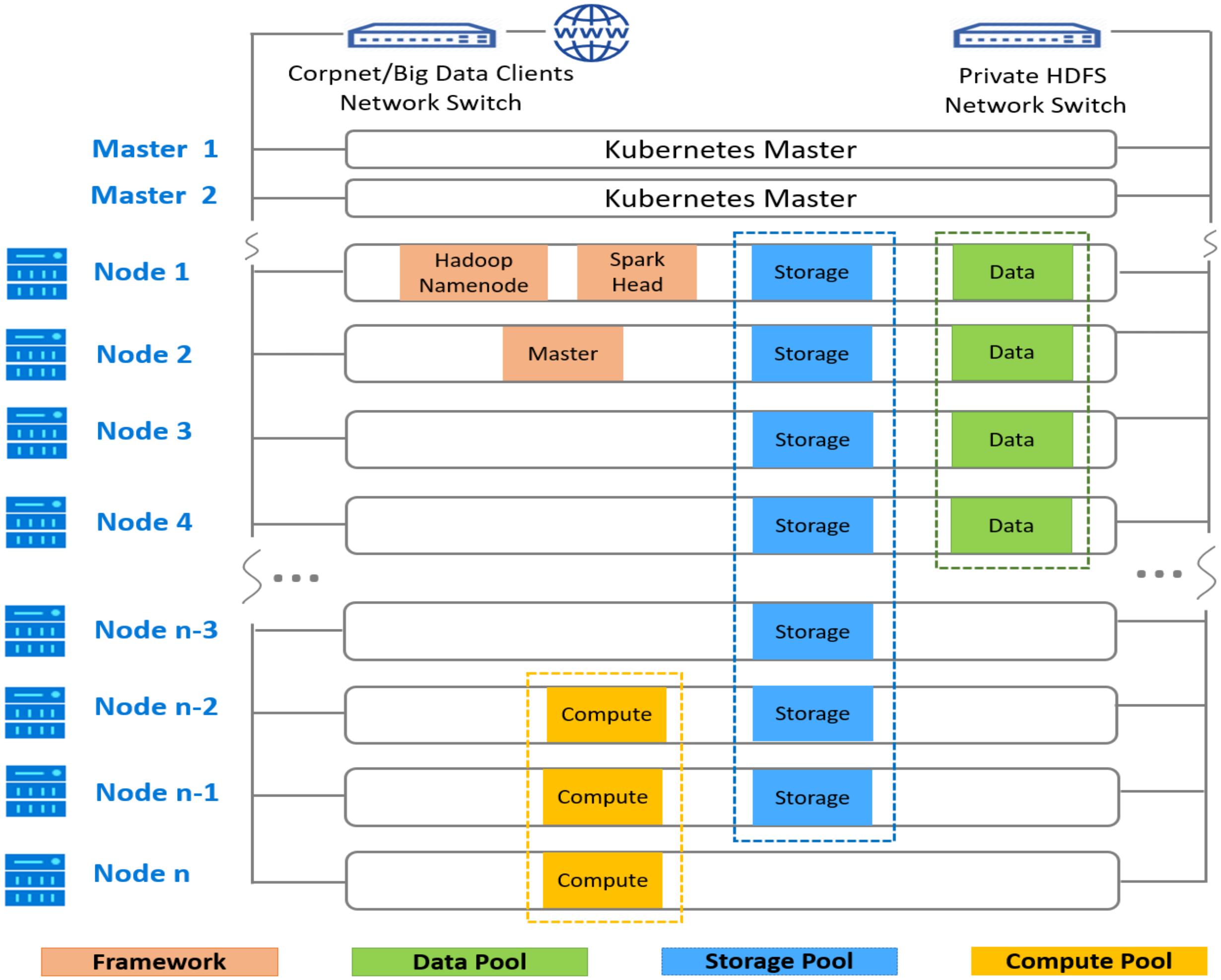


Figure 3: Microsoft SQL Server 2019 Big Data Cluster Architecture – Logical View

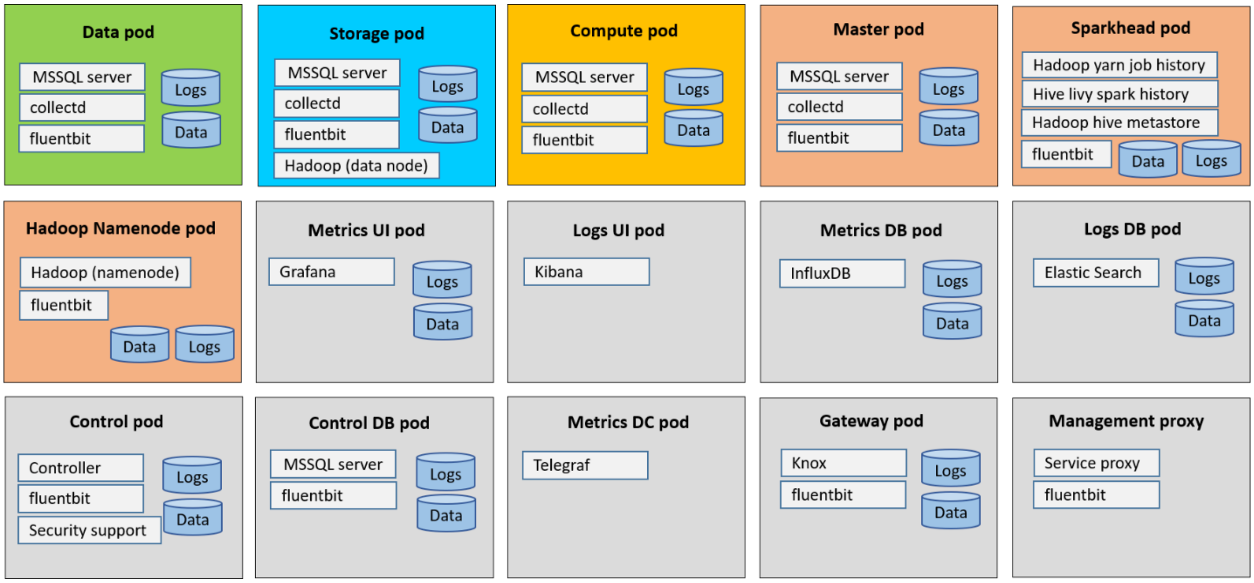
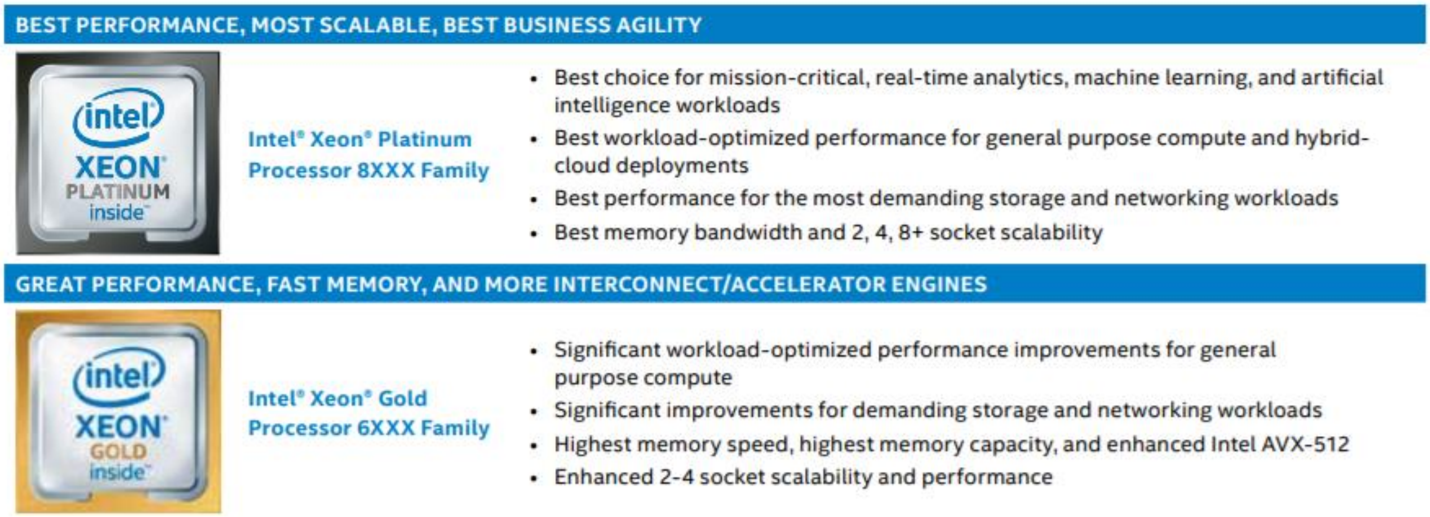


Figure 4: Microsoft SQL Server 2019 Big Data Cluster - Pod Level View

Intel® Xeon® Based Platforms

The Intel® Xeon® Scalable platform provides the foundation for a powerful data center. Disruptive by design, this innovative processor sets a new level of platform convergence and capabilities across compute, storage, memory, network, and security. Across infrastructures, Intel® Xeon® Scalable platform is designed for data center modernization to drive operational efficiencies to lead to improved total cost of ownership (TCO) and higher productivity for users. From its new Intel® Mesh Architecture and widely expanded resources to its hardware-accelerating and newly integrated technologies, the Intel® Xeon® Scalable platform enables a new level of consistent, pervasive and breakthrough performance.


intel
XEON
PLATINUM
inside
intel
XEON
GOLD
inside

BEST PERFORMANCE, MOST SCALABLE, BEST BUSINESS AGILITY

Intel® Xeon® Platinum Processor 8XXX Family

- Best choice for mission-critical, real-time analytics, machine learning, and artificial intelligence workloads
- Best workload-optimized performance for general purpose compute and hybrid-cloud deployments
- Best performance for the most demanding storage and networking workloads
- Best memory bandwidth and 2, 4, 8+ socket scalability

GREAT PERFORMANCE, FAST MEMORY, AND MORE INTERCONNECT/ACCELERATOR ENGINES

Intel® Xeon® Gold Processor 6XXX Family

- Significant workload-optimized performance improvements for general purpose compute
- Significant improvements for demanding storage and networking workloads
- Highest memory speed, highest memory capacity, and enhanced Intel AVX-512
- Enhanced 2-4 socket scalability and performance

Figure 5: Intel® Xeon® Processors

Intel® Data Center SSDs

Reliable Intel® SSD D3-S4510 Series and DC P4510 series, based on 64-layer Intel® 3D NAND TLC, meet demanding service level requirements while increasing server efficiency. Innovative SATA firmware and the latest generation of Intel®3D NAND make D3-S4510 SSDs compatible with existing SATA setups for an easy storage upgrade, whereas it also enables scalable performance and low latency via PCIe/NVMe based DC P4510 family. Simply by integrating SSDs into the solution, organizations improve server agility and scale for more users and better services, supporting larger data without expanding the server footprint.

Table 1: Intel® Data Center SSD Technology Overview

Features At-a-Glance		
Capacity	**S4510**	2.5in: 240GB, 480GB, 960GB, 2TB, 4TB, 8TB; M.2 240GB, 480GB, 960GB
	P4510	1TB, 2TB, 4TB, 8TB
Performance	**S4510**[1]	128K Sequential Read/Write – up to 560/510 MB/s 4KB Random Read/Write – up to 97,000/36,000 IOPS
	P4510[2]	128K Sequential Read/Write – up to 3200/3000 MB/s 4KB Random Read/Write – up to 637K/139K IOPS
Reliability	Designed for end-to-end data protection from silent data corruption, uncorrectable bit error rate < 1 sector per 10^17 bits read	
Power	**S4510**	Active power up to 3.6W; Idle power up to 1.1W
	P4510	Up to 16W
Interface	**S4510**	SATA 6Gb/s
	P4510	PCIe 3.1 x4, NVMe 1.2
Form Factor	**S4510**	2.5in x 7mm; M.2 2280
	P4510	U.2 2.5in x 15mm
Media	Intel® 3D NAND TLC	
Endurance	**S4510**	up to 2 DWPD
	P4510	up to 1 DWPD (JESD219 workload)
Warranty	5-year limited warranty	

Microsoft Reference Cluster Configurations

Great deployment flexibility – hardware reusability

Microsoft SQL Server 2019 Big Data Reference Cluster has 9 servers running Linux: one master node and 8 storage nodes. In this document, the presented use case scenarios are dictating the configuration of the Big Data Cluster, and its implementation (deployment) is done via Kubernetes instructions embedded into YAML files.

A YAML file is describing the set of common pods that are pulled as images out of the public Docker repository and deployed in a very easy, automated way. The existing parameters of the deployed configuration could be changed afterward. A new configuration can be deployed if the new use case

---

[1] System Configuration: Motherboard - H270N-WIFI-CF (Gigabyte Technology Co.); CPU – Intel © Core ™ i7-17700 @ 3.6GHz; BIOS version – F8d (American Megatrends Inc.); Memory – 8052144KB; OS version – CentOS 7.5; Kernel version – 4.17; FIO version – 3.1

[2] System Configuration: FIO* was used with this configuration: Intel® Server Board S2600WTTR, Intel® Xeon® E5-2699 v4, Speed: 2.30GHz, Intel BIOS: Internal Release, DRAM: DDR4 – 32GB, OS: Linux Centos* 7.2 kernel 4.8.6. SSD firmware version VDV10120. Testing performed by Intel.

arises from the changes in the customer's business model or the usage pattern. The previous configuration must be removed, and then the new configuration can be deployed. We have used this flexibility to deploy 1TB, 10TB, and 100TB benchmarks. This demonstrates fantastic flexibility and hardware re-use for different business purposes.

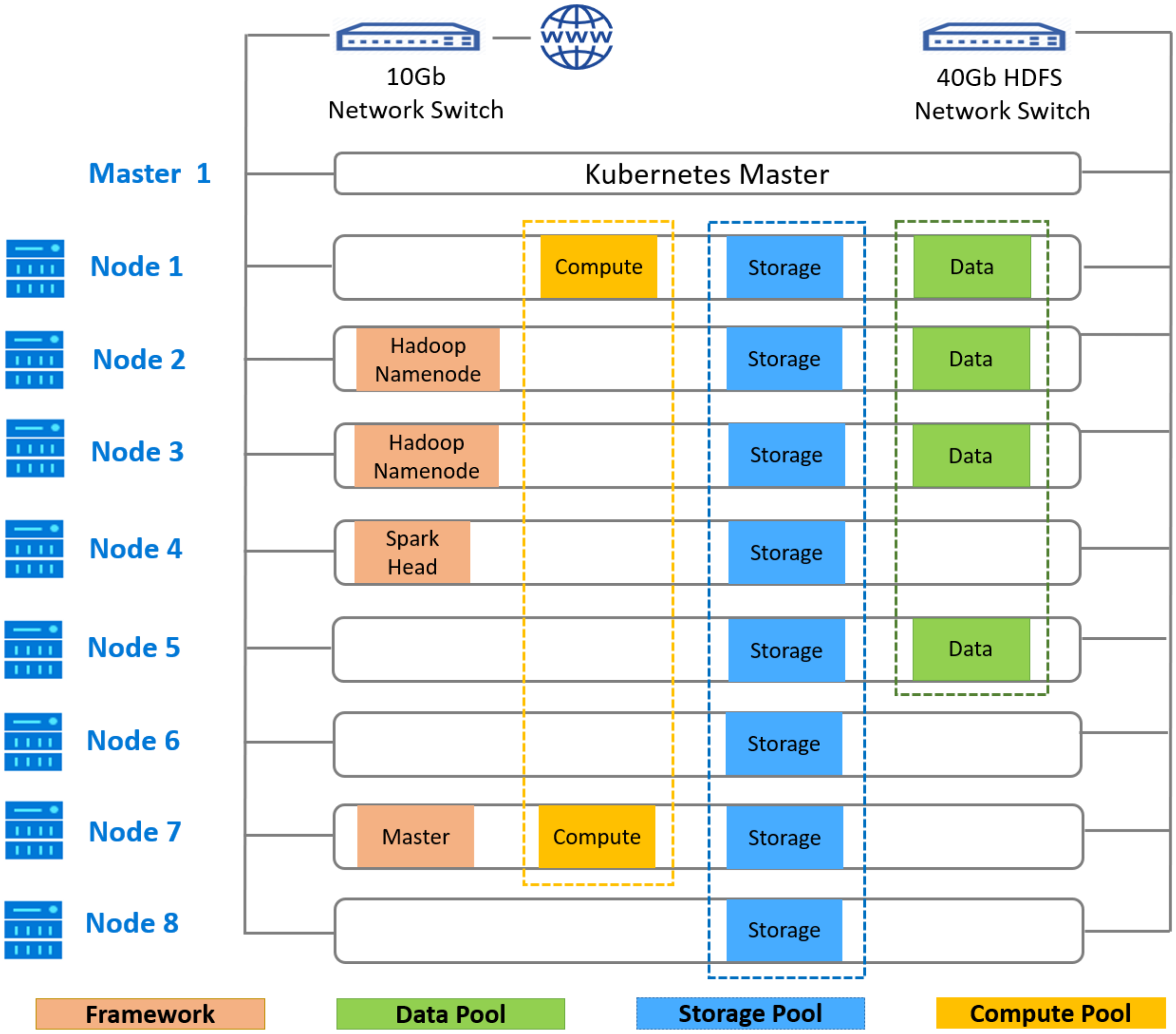


Figure 6: Microsoft Reference Cluster – Physical View

For the Big Data use case scenario, we have configured the Big Data Cluster with 8 physical nodes as the storage nodes to be able to store a large amount of data into the platform embedded HDFS file system. Here, the powerful software meets powerful hardware (see Table 1Table 2).

For each agent node storage of the Microsoft reference cluster, we created 3 logical volumes `/data`, `/mssql` and `/general` that the Kubernetes persistent volumes (PV) mount. These volumes were created using the Linux mdadm utility. The logical volumes are striped across partitions of physical disks. The detailed steps to configure these logical volumes are provided in the Microsoft Reference Cluster - Logical Volume for storagethe Appendix.

Table 2: Microsoft Reference Cluster - Configurations

OVERVIEW	
Sockets/Cores/Threads (per node)	2/48/48
Nodes	1 master + 8 agents
Cluster Network Switch Vendor	Mellanox SX1701
Server Vendor	Lenovo ThinkSystem SR650
BIOS Operating Mode	Max. Performance
SYSTEM DETAILS – Server configurations per node	
Processor	Intel® Xeon®(R) Platinum 8160 CPU @ 2.10GHz
Memory (RAM)	768GB (32GBx24)
OS Disk	480GB; Micron 5100 PRO
Data Disk	5x 8TB; Intel® DC P4510
Network	Corpnet: 10Gbps, HDFS: 40Gbps
SOFTWARE	
OS Distribution	Ubuntu 16.04 LTS
SQL Big Data Cluster Version	RC1
Framework	Spark 2.4.2
WORKLOAD	
Data Set Size	1TB, 10TB, 100TB

Intel Reference Cluster Configurations

The Intel Reference Cluster is set up similarly to the Microsoft Reference Cluster. It has 1 master and 8 agent nodes powered by Intel® Xeon® technology. The cluster has 8 Storage pods, 4 Data pods and 4 Compute pods forming the Storage, Data, and Compute pools. Each node is packed with Intel® Xeon Gold processors, Intel® Data Center SSDs, and other hardware, as mentioned in the

Table 3, to deliver the best performance offering a stable system viable to 1TB, 3TB, 10TB and 30TB data sets. The reference cluster is setup as shown in Figure 7.

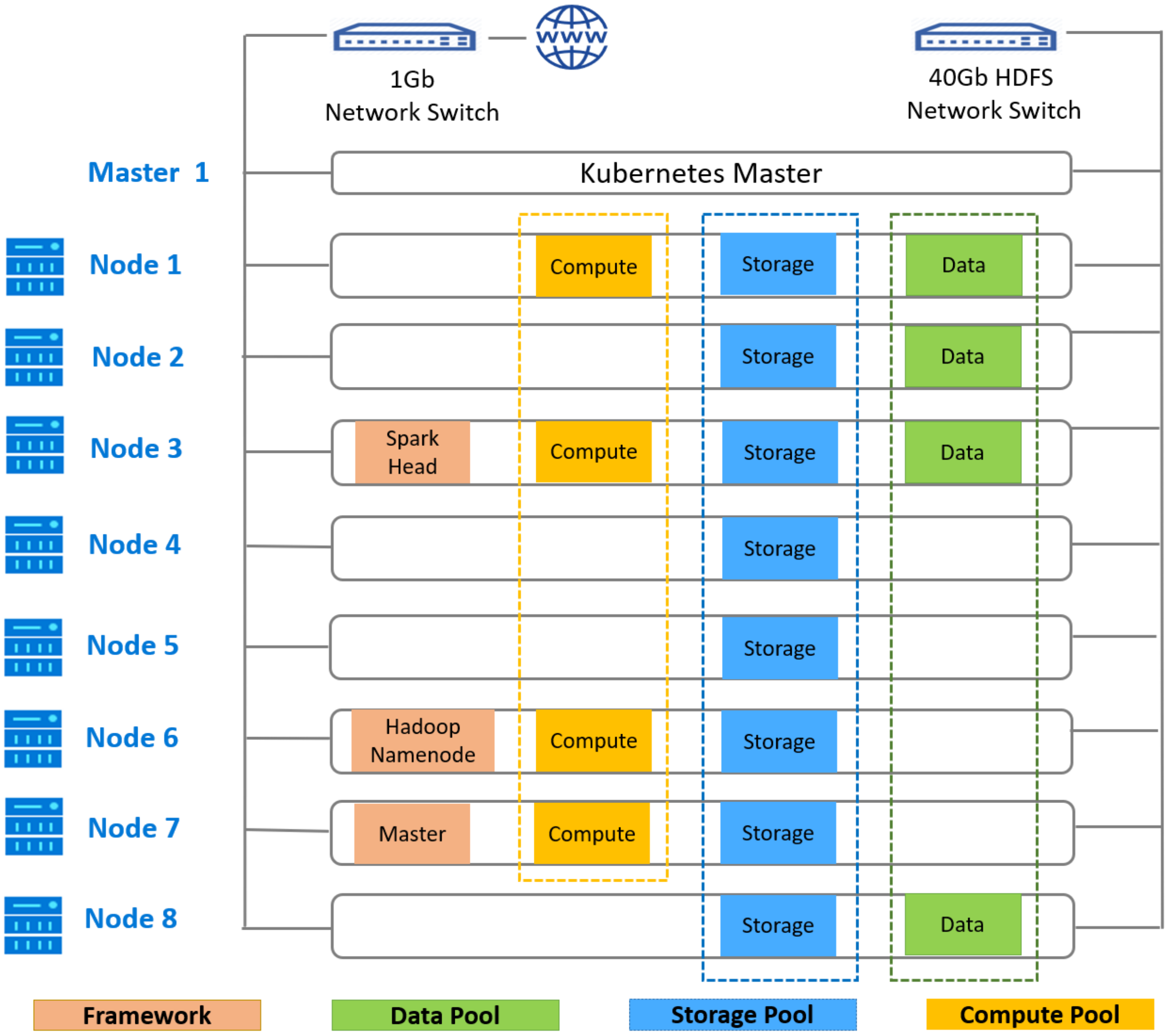


Figure 7: Intel Reference Cluster - Physical View

Table 3: Intel Reference Cluster - Configurations

OVERVIEW	
Sockets/Cores/Threads (per node)	2/36/72
Nodes	1 master + 8 agents
BIOS Settings	HT on, Turbo On
SYSTEM DETAILS - Server Configuration per node:	
Processor	Intel® Xeon® Gold Processor
Memory (RAM)	768GB (32GBx24)
OS Disk	200GB; Intel® SSD 710
Data Disk (Agent nodes only)	3x 3.84TB; Intel® D3-S4510
Network	1Gbps, 40Gbps
SOFTWARE	
OS Distribution	Ubuntu 16.04 LTS
SQL Big Data Cluster Version	RC1
Framework	Spark 2.4.2
WORKLOAD	
Data Set Size	1TB, 3TB, 10TB, 30TB

In the Intel Reference clusters, we configured the storage as follows: On each agent node, the Kubernetes persistent volumes (PV) are mounted on a logical volume mounted at `/mnt/local-storage`. This logical volume is created using Logical Volume Management (LVM). As shown in Figure 8, physical volumes corresponding to SSD (SATA or PCIe) disks map together into a volume group and then a logical volume. The logical volume is striped across the disks, resulting in superior performance. Detailed steps to configure logical volumes are provided in the Logical Volume for storage section of the Appendix.

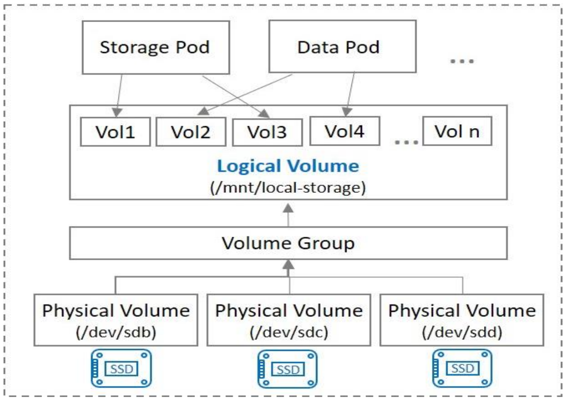


Figure 8: Logical Volumes in Agent Nodes - Configuration

In both the Microsoft and Intel Reference Clusters, to improve the performance of Docker and Kubernetes, on our agent nodes, we mounted the Docker's /var/lib/docker directory and Kubernetes' /var/lib/kubelet directory on the striped logical volume as well. Without this step, the OS disk was the performance bottleneck at higher data sets. Hence this step is necessary to run queries at higher data sets. Detailed steps are provided in the Appendix.

Test Data Sets

TPC-DS is the world's first industry-standard benchmark designed to measure the performance of SQL-based Big Data implementations. In this study, we have used data sets based on TPC-DS schema producing 1TB, 10TB, 30TB and 100TB worth of raw structured and semi-structured data. TPC-DS schema describes a data model of a retail enterprise selling through 3 channels (stores, catalogs, and web). It is comprised of 99 queries that scan large volumes of data by utilizing Spark SQL and gives answers to real-world business questions. It challenges the cluster configurations to extract maximum efficiency in the areas of CPU, memory, and I/O utilization along with operating system and the big data solution.

Both reference clusters utilize the publicly available kit from Databricks for data generation, data load, simple table statistics generation, and query execution. It generates data and then loads it into HDFS of Microsoft SQL Server 2019 Big Data Cluster as a set of Parquet files. The used schema contains 24 tables of data that are distributed over 8 Storage nodes.

To illustrate these data sets to the reader, we present the following 10TB and 100TB data sets as examples:

10TB Data set

The 10TB data set consists of 56 billion rows with 2 largest tables: catalog_sales and store_sales. Figure 9 shows the cardinality distribution of our data set.

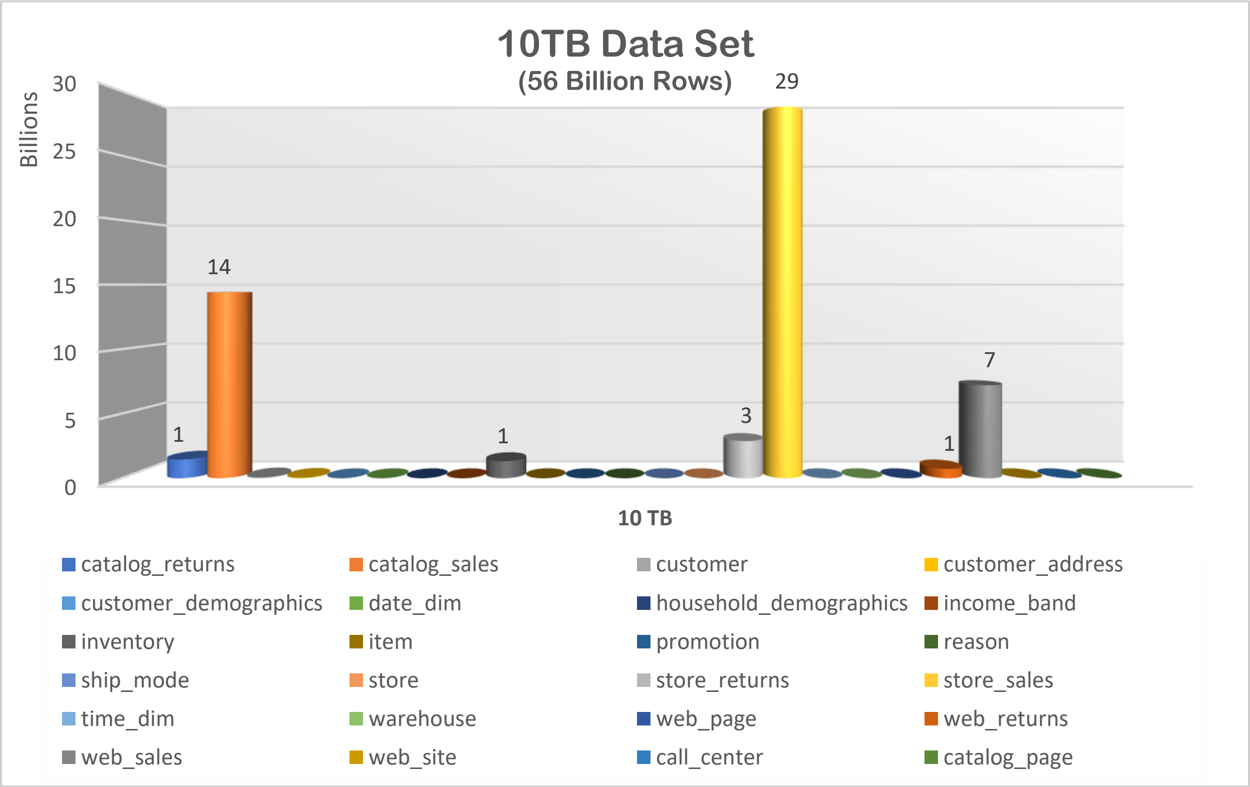


Figure 9: 10TB Data Set - Data Distribution

We observe that 99% of data is stored in 7 tables with a corresponding partition key, which is used for fast data retrieval. The data load and statistics generation for the 10TB data set was done in 3.69 hours on the Microsoft Reference cluster. This is the simple 10TB data set statistics for NULL value count of the used partition keys for 3 tables.

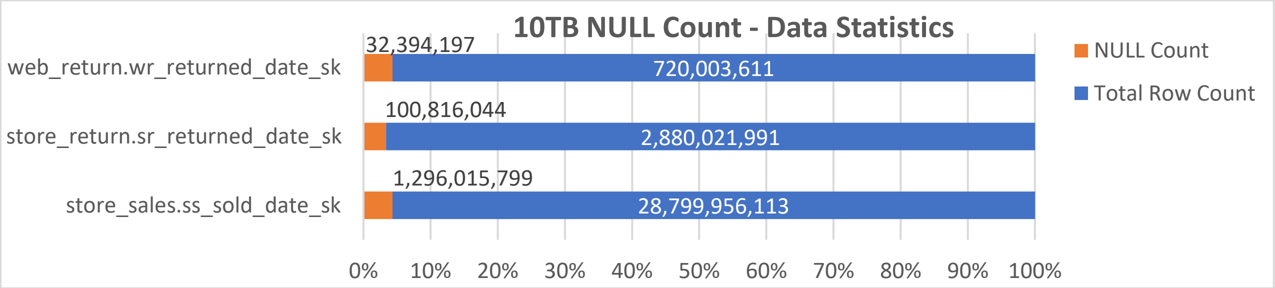


Figure 10: 10TB NULL Count - Data Statistics

100TB Data Set

The 100TB dataset consists of 556 billion rows with 2 largest tables: catalog_sales and store_sales. Figure 11 shows the distribution of our data set.

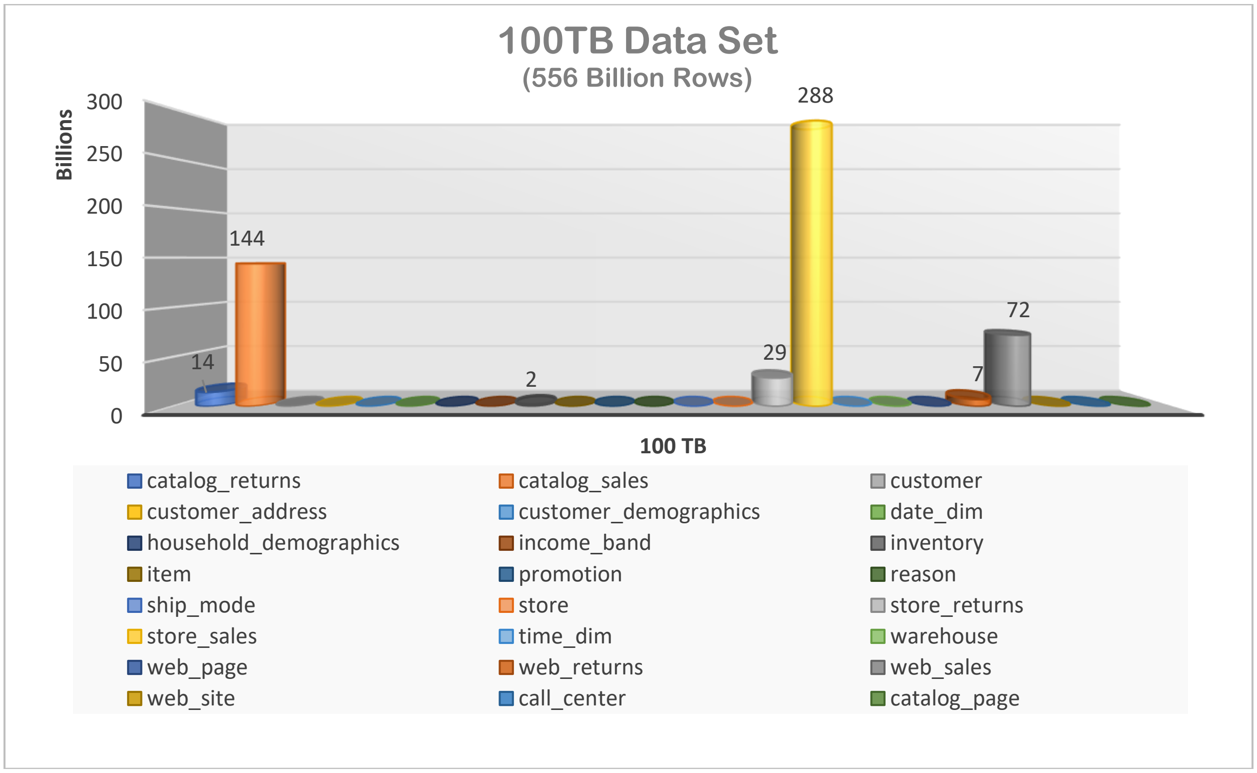


Figure 11: 100TB Data Set - Distribution

The data load and the statistics generation for 100TB data set was done in 35.25 hours on Microsoft Reference Cluster. The composition of the same partition key NULL value statistics for 100TB data set is shown in Figure 12.

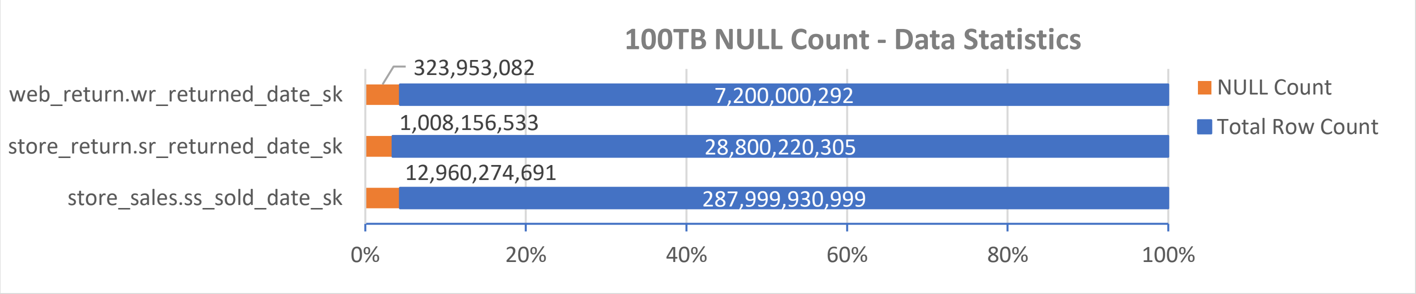


Figure 12: 100TB NULL Count - Data Statistics

It is worth noting that the total number of store_sales table partitions is 1824, and table takes 14.1TB of disk space, and the NULL partition size is around 535 GB, and that constitutes 3.8% of that space. It is a similar situation with store_return and web_return tables their NULL partitions. These statistics show that data in these tables is skewed on the NULL partition, and that is reflected as slower processing of more complex queries where these tables are referenced and/or partition keys used, as well as the longer data load time. Due to Spark's limitation of processing one partition with one thread, most of the data load time is spent on loading the NULL partitions of the mentioned tables. It takes around 18 hours to load the NULL partition of the largest schema table store_sales and that in combination with load times of other two table NULL partitions determine the data loading time of the data set.

Data Load

On the Microsoft reference cluster, we observe that the data load and the compute table statistics scales linearly from 1TB to 10TB to 100TB data sets. It takes 0.5 hours, 3.69 hours, and 35.28 hours respectively. The data generation and the data load are shown in Figure 13, but they are considered as a single step.

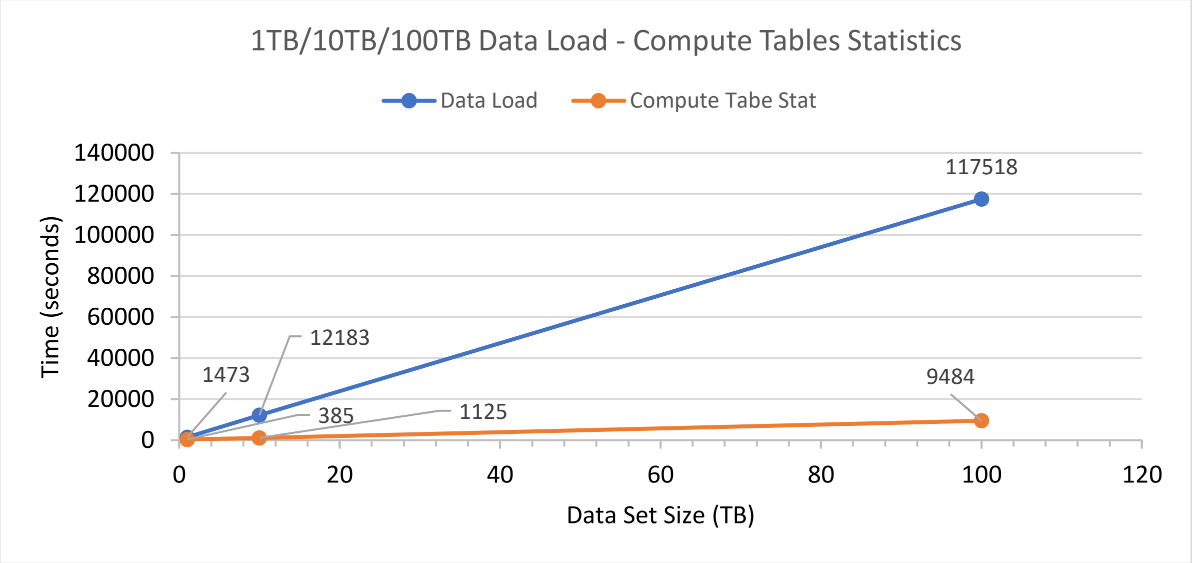


Figure 13: 1TB/10TB/100TB Data Load - Compute Tables Statistics

Similar observations were made on the Intel reference cluster, where the data load time is scaling, as illustrated in Figure 14.

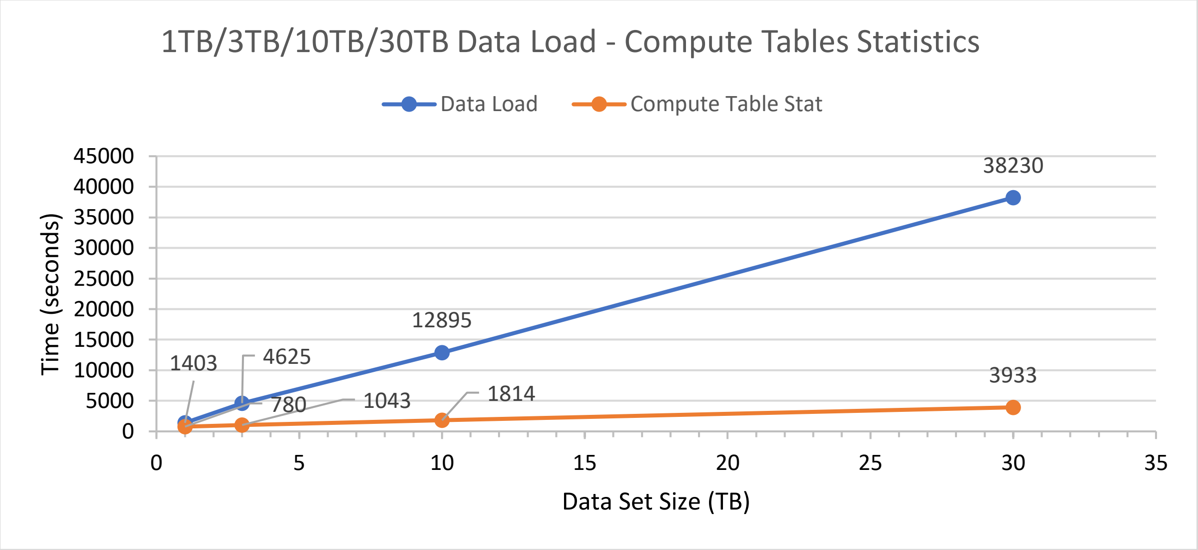


Figure 14: 1TB/3TB/10TB/30TB Data Load - Compute Tables Statistics

Run Methodology

The following are the detailed steps involved to perform the experiments in this study.

1. Microsoft SQL Server 2019 Big Data Cluster setup and deployment

   First, for each node in the cluster, we installed Linux and tuned the BIOS settings for optimal performance (see Tables 2 and 3). Then we installed Docker 18.09 and Kubernetes 1.15. Then, we setup the Kubernetes network across all the nodes, with one node being the master. After validating the setup, we deploy the Microsoft SQL Server 2019 Big Data Cluster using the cluster configuration tool ***azdata***. Once the cluster is deployed, we ensure that each storage pod comes up on a separate node. This is critical to performance scaling since they provide the spark execution environment for our experiments.

   As described in our cluster configuration, our persistent volumes are mounted on the logical volume that is striped across our data disks. This ensures high IO bandwidth and utilization across the cluster storage subsystem.

2. Data set creation

   To generate the data set, we use the Databricks' kit without any modifications. For all data sets, it dumps the partitioned data to the HDFS volumes configured across the storage pods in a parquet file format.

   For both the Microsoft and Intel Reference Clusters, we increase the HDFS replication factor to 3. Further details of the impact of this parameter on overall IO and network characteristics will be discussed in a future whitepaper.

3. Test run

We use Databricks' TPC-DS Spark SQL kit to run the read-only suite of 99 queries, some of which are split into queries a and b, resulting in 104 Spark SQL queries (see TPC-DS Schema Based Queries section of Appendix). We copy the kit's JAR file on the Big Data Cluster sparkhead pod or one of the storage pool pods. Then we launch the spark application using the spark parameters discussed in the

Spark SQL Configuration section. In all our performance runs, the queries are executed in sequence 1 to 99. Once the run is complete, we collect the query time results from the HDFS. These results are analyzed offline together, along with some system-level performance counters we collect to measure system characteristics.

For each data set size, we repeat steps 2 and 3 and re-adjust spark parameters for the data set size. Given this is a big-data environment, there is no need to restart the nodes or the SQL BDC software in between the performance runs.

Spark SQL Configuration

Two sets of spark optimizations configurations are shown in Tables 4 and 5. We choose the number of executors and executor memory allocated based on the hardware resources (CPU, Memory, and Storage) available on the cluster.

Table 4: Microsoft Reference Cluster - Spark Parameters

Microsoft Reference Cluster Spark Parameters
spark.sql.statistics.histogram.enabled=true
spark.sql.cbo.enabled=true
spark.sql.cbo.joinReorder.enabled=true
spark.sql.cbo.joinReorder.dp.threshold=18
spark.driver.maxResultSize=16g
Spark.yarn.executor.memoryOverhead=6g
num-executors = 80
executor memory='65g'
executor-cores = 4

Table 5: Intel Reference Cluster - Spark Parameters

Intel Reference Cluster Spark Parameters	
Baseline	**Optimized**
spark.sql.autoBroadcastJoinThreshold=20971520	spark.sql.autoBroadcastJoinThreshold=20971520
spark.sql.statistics.histogram.enabled=true	spark.sql.statistics.histogram.enabled=true
spark.driver.maxResultSize=16g	spark.driver.maxResultSize=16g
num-executors = 128	num-executors = 128
executor-memory = 41g	executor-memory = 41g
executor-cores = 4	executor-cores = 4
	spark.sql.cbo.enabled=true
	spark.sql.cbo.joinReorder.enabled=true
	spark.sql.cbo.joinReorder.dp.star.filter=false
	spark.sql.cbo.starSchemaDetection=true
	spark.sql.optimizer.nestedSchemaPruning.enabled=true
	spark.sql.cbo.joinReorder.dp.threshold=18

Based on our experiments, we find the best performance with 4 cores per executor. We also identified a set of spark optimizations that improve query performance. These optimizations include Cost Based Optimizations (CBO), modifying the Join Reorder threshold, Broadcast Join Threshold, and the Join Types (SortMerge Join, Broadcast Hash Join). In both reference clusters, we enable sql.statistics.histogram for accurate cardinality estimates for filter and join predicates. The flag spark.sql.cbo.joinReorder.enabled is recommended to be true when CBO is enabled.

For both reference clusters, we have performance runs with two configurations, called ***baseline*** and ***optimized,*** respectively. The ***optimized*** configuration has CBO, and related flags enabled.

In Microsoft Reference Cluster, we increased the joinReorder.dp.threshold to 18 (from the default value of 12) to enable join-reorder optimization if more than 12 tables are joined. This tremendously improves the performance of query 64 for the 100TB data set. However, the optimization process creates over 38k physical plans, which increases the optimization time to around 20 mins, and reduces performance for smaller data sets. Thus, this parameter is not applicable for 10TB data set.

In the Intel Reference Cluster, we increased the autoBroadcastJoinThreshold to 20MB (from default value of 10MB) to encourage queries to perform more efficient broadcast join, leading to improved performance in selected queries. For all data sets, we increase joinReorder.dp.threshold to 18 (from default value of 12) for reasons described above on this platform.

Results and Analysis

Scaling Performance

For the Microsoft Reference Cluster configuration, we have performed characterizations with 1TB, 10TB, and 100TB data sets.

For the Intel Reference Cluster, characterizations were executed with 1TB, 3TB, 10TB and 30TB data sets.

Microsoft reference cluster

10TB with Spark Optimizer enabled

The test results of the 10TB data set for all the queries based on TPC-DS schema are shown in Figure 15. Apache Spark optimizer (Catalysts) brings a benefit of the shorter execution time for more than 50 % of queries.

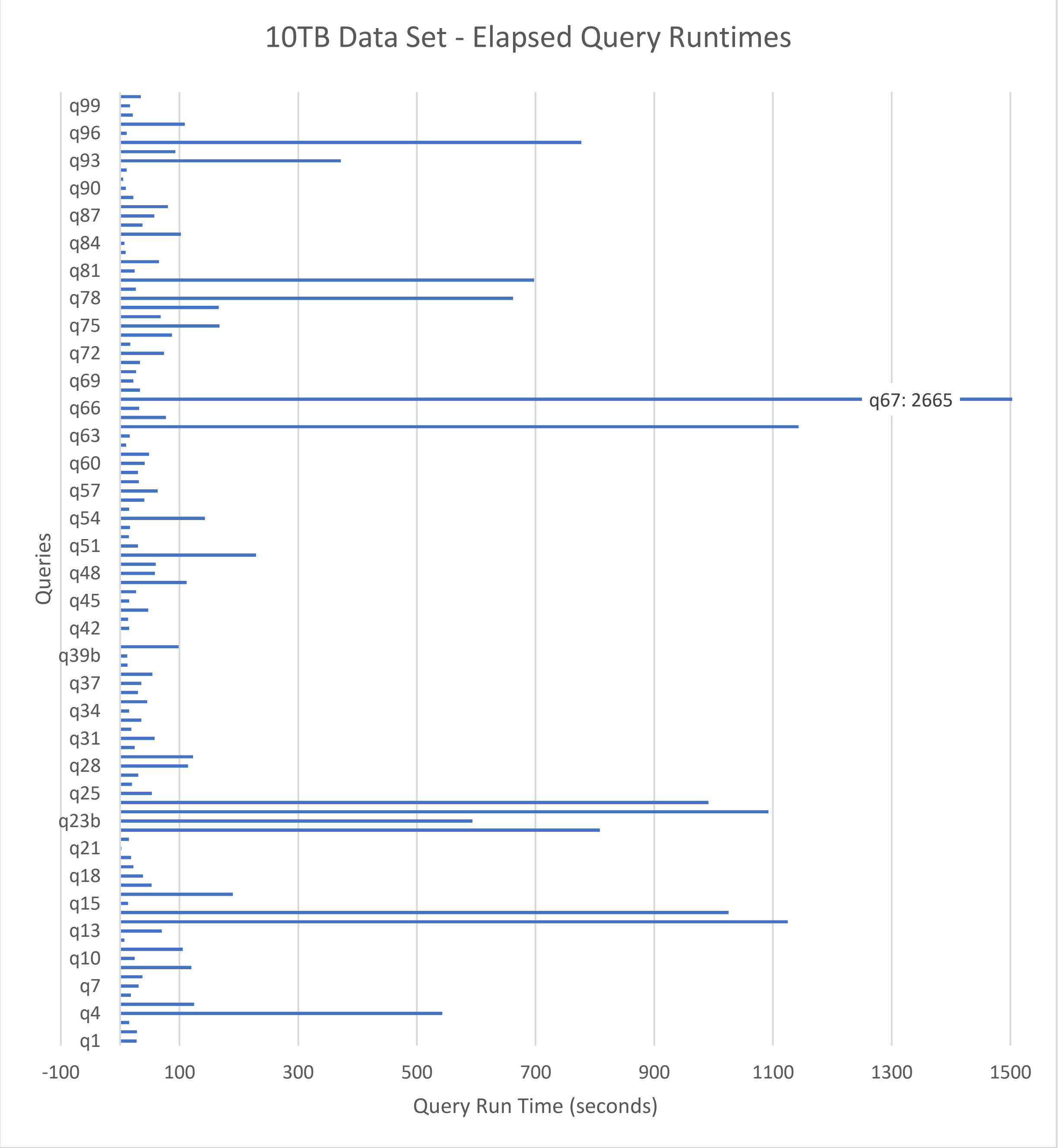


Figure 15: 10TB Data Set - Elapsed Query Runtimes

1TB – 10TB – 100TB with Spark Optimizer enabled

The 94-query subset of TPC-DS is used to demonstrate the scalability and performance of the Microsoft SQL Server 2019 Big Data Reference Cluster. The described hardware configuration presents “the sweet spot” from the cost perspective. It is a configuration that offers great performance, scales very well as data grow, and it could be used for typical customer deployments. Additional investments into memory would be required to enable rest of the queries to run.

Join-reorder optimization plays an important role in long running queries when there are joins between multiple tables. The notable winner is query 72. Apache Spark join-reorder will not bring benefits across all queries and in some cases, there is execution degradation when Spark Optimizer is used compared to when the query execution when Spark Optimizer is disabled. This execution degradation is usually with shorter runtime queries, and it modestly fluctuates between successive executions of the test runs which is normal with this type of processing that involves Big Data sets.

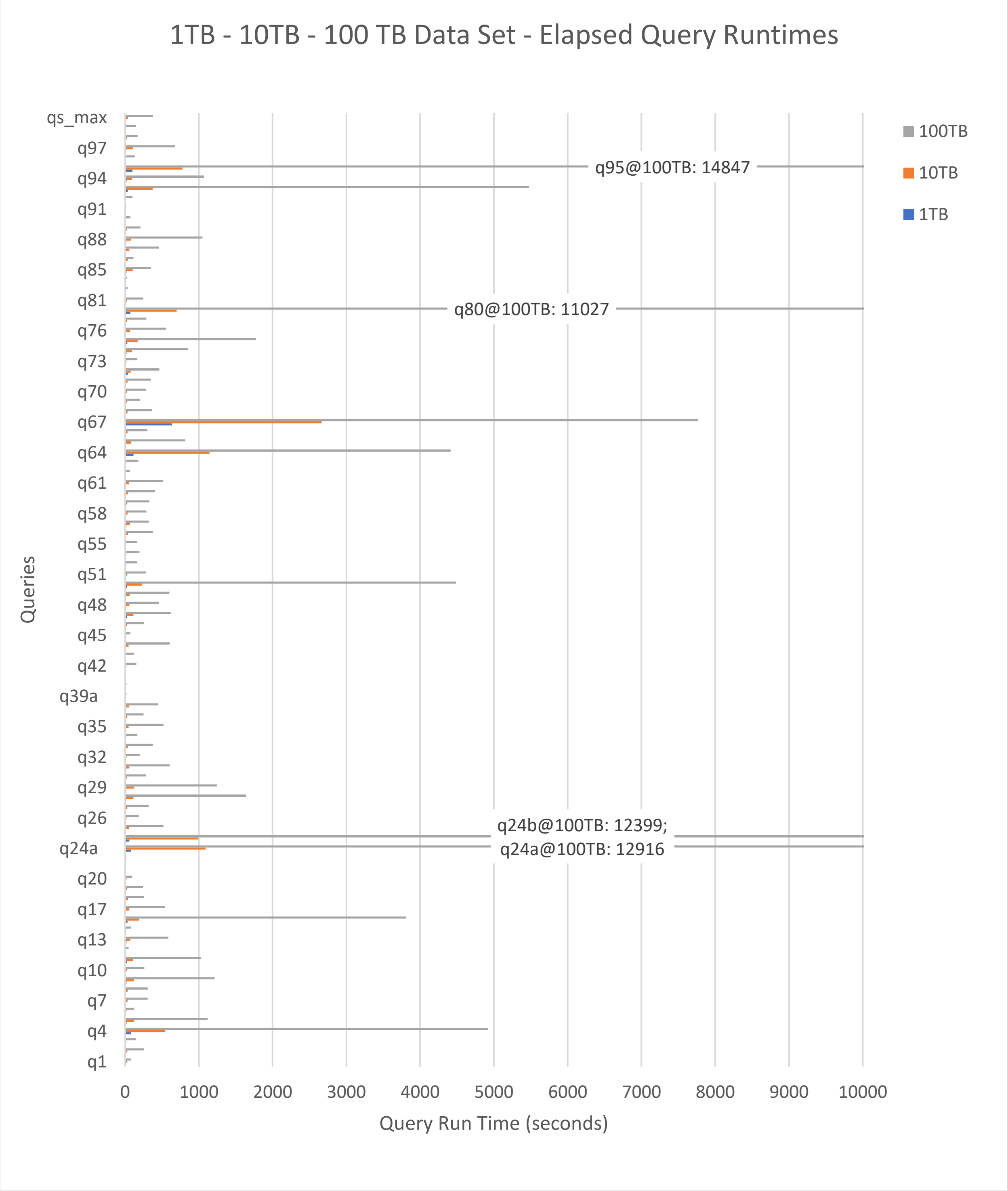


Figure 16: 1TB - 10TB - 100TB Data Set - Elapsed Query Runtimes

Query Runtimes

The following two graphs show the excellent scaling capabilities of Microsoft SQL Server 2019 Big Data Reference Cluster as the data grows and demonstrates powerful elasticity and performance of the entire platform. The total runtime of 94 queries and Geo Mean are linearly changing as the data set grows.

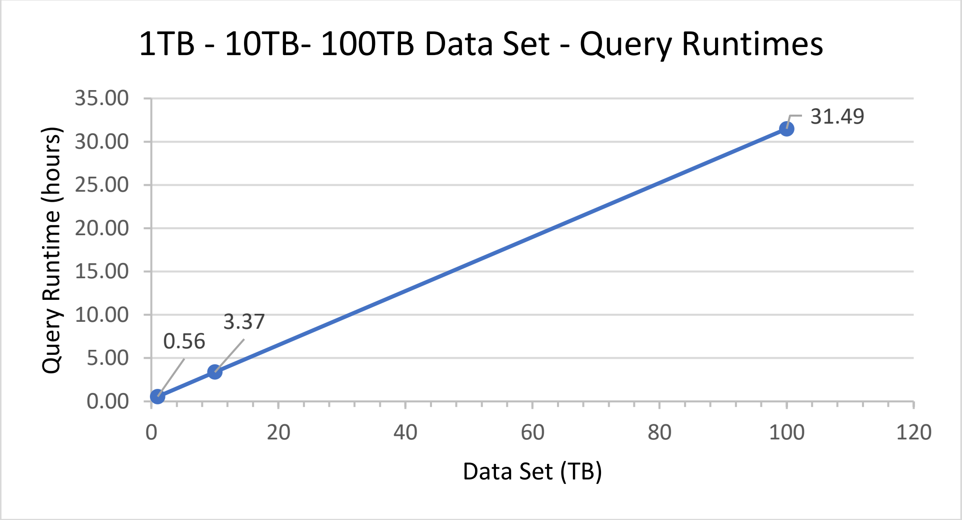


Figure 17: 1TB - 10TB - 100TB - Query Runtimes

Geometric Mean is calculated for 94 queries using the formula shown below.

$$Geomean\ of\ Query\ runtime = \sqrt[94]{\prod_{i=1}^{94} runtime(Qi)}$$

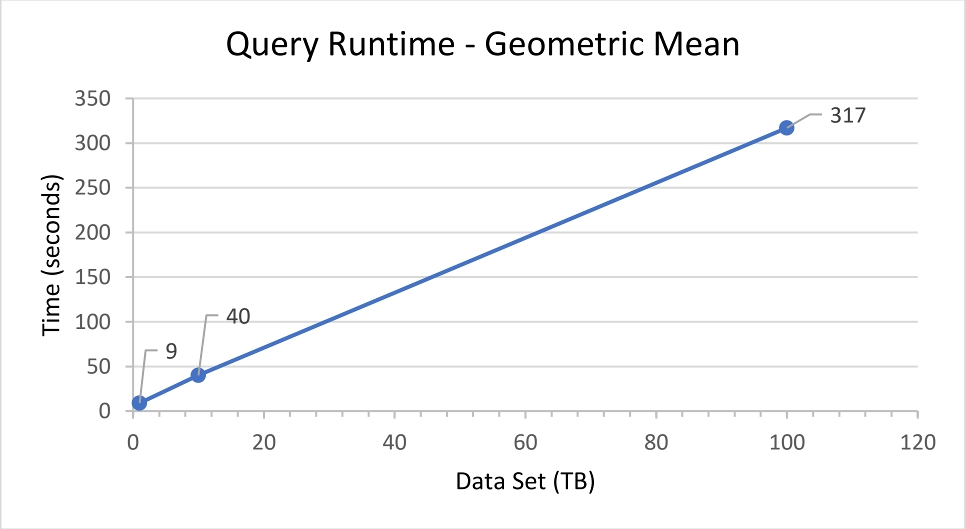


Figure 18: Query Runtime - Geometric Mean

Performance Analysis

From the master node, we monitor the system performance on each agent node using the free and open-sourced Linux tools sysstat SAR and Intel® Performance Analysis Tool (PAT). The following figures show the average CPU utilization, disk bandwidth, and network bandwidth across the entire runs, averaged across the nodes, at different data sets. The performance scales linearly across all data sets.

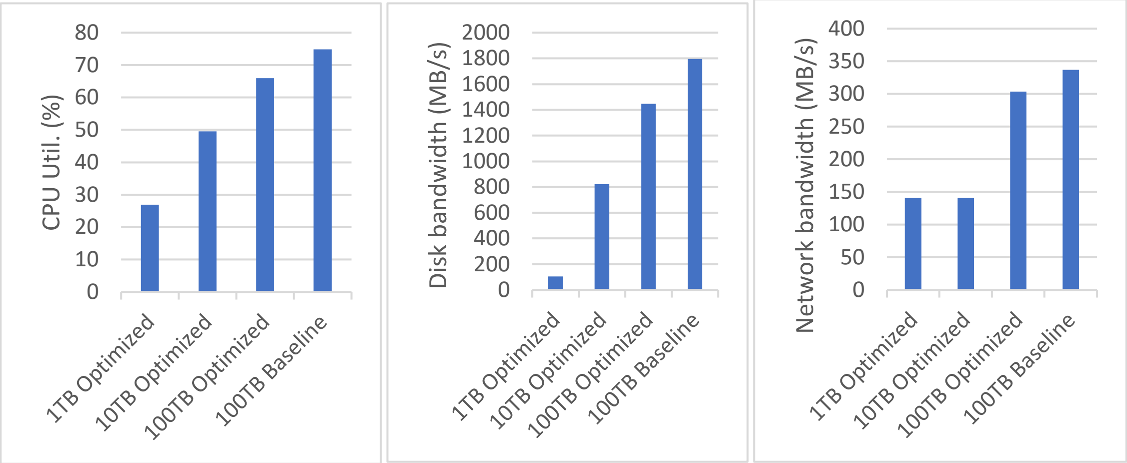


Figure 19: System Performance per Node – Microsoft Reference Cluster

For the memory allocation, the following graph presents the percentage of used memory in average per node for one of these 100TB Optimized runs:

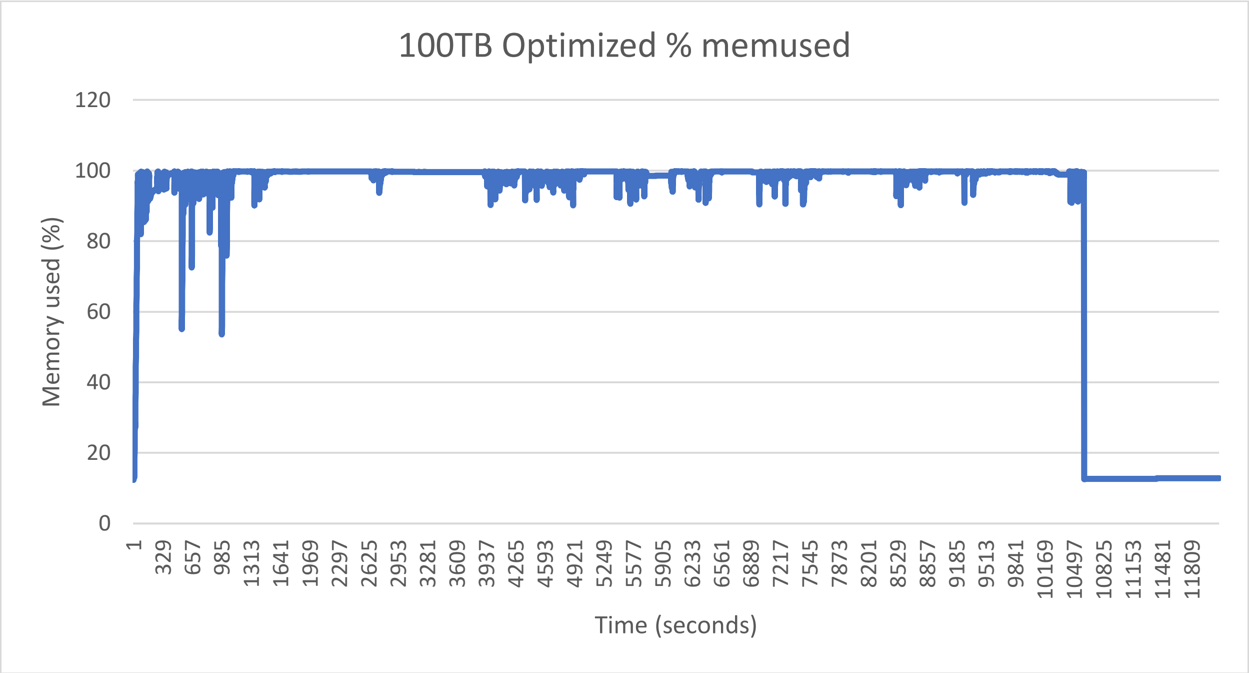


Figure 200: 100TB % of Memory Usage per Node

We also have graphic representations of various system resources under these workloads shared in System Performance section of Appendix. They should help you understand the updates we applied to the hardware, Microsoft SQL Server 2019 Big Data Cluster deployment, and Spark settings.

Intel Reference Cluster

In this setup, we increase the data sets from 1TB, 3TB, 10TB and 30TB and demonstrate that the same cluster configuration offers great performance across different data sets.

Scaling Query Runtimes

We compare the performance of our baseline configuration with the optimized configuration and observe that the optimized Spark configuration results in significant performance improvement for all data sets, with more than 50% improvement for the30TB data set.

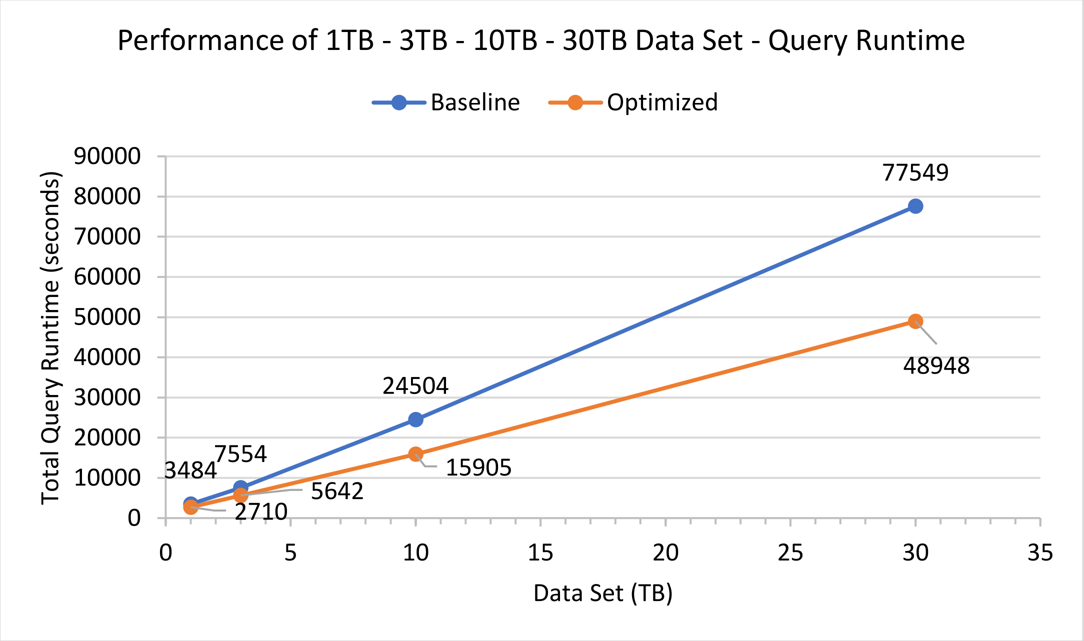


Figure 211: Performance of 1TB - 3TB - 10TB - 30TB Data Set – Query Runtime

The plot in Figure 21 shows the performance difference between baseline and optimized configuration for all queries, excluding 14a, 14b, 64 for reasons discussed in the TPC-DS Schema Based Queries section of the Appendix.

The performance improvement is contributed majorly by the following:
1) Enabling CBO: CBO significantly improves the performance of the 16 queries (q04, q06, q11, q13, q15, q17, q24a, q24b, q25, q29, q45, q49, q72, q74, q85, q91). The prime example being query 72, in which we observe a 47x performance Improvement for the 30TB data set.
2) Tuning spark.sql.autoBroadcastJoinThreshold: This configures the maximum size in bytes for a table that will be broadcasted to all worker nodes when performing a join. Increasing this parameter to 20MB (default is 10MB) promotes the queries to perform efficient broadcast joins.
3) Enabling spark.sql.cbo.joinReorder.enabled: The execution engine selects the optimal plan using the plan statistics. Enabling this parameter (default is False) makes the query plan optimized to have more Broadcast Joins over Hash Joins.

For both the baseline and optimized configurations, the query runtime scales linearly with an increasing data set.

Performance Analysis

From the master node, we monitor the system performance at each agent node using the Intel® PAT and analyze the data to estimate various system metrics across the entire performance run. Figure 22 shows the average CPU utilization, disk bandwidth, and network bandwidth across the entire run, averaged across the nodes, at different data sets. We observe that the performance metrics scales well with increasing data set. Thus, this configuration is viable to implement different data sets without any performance degradation.

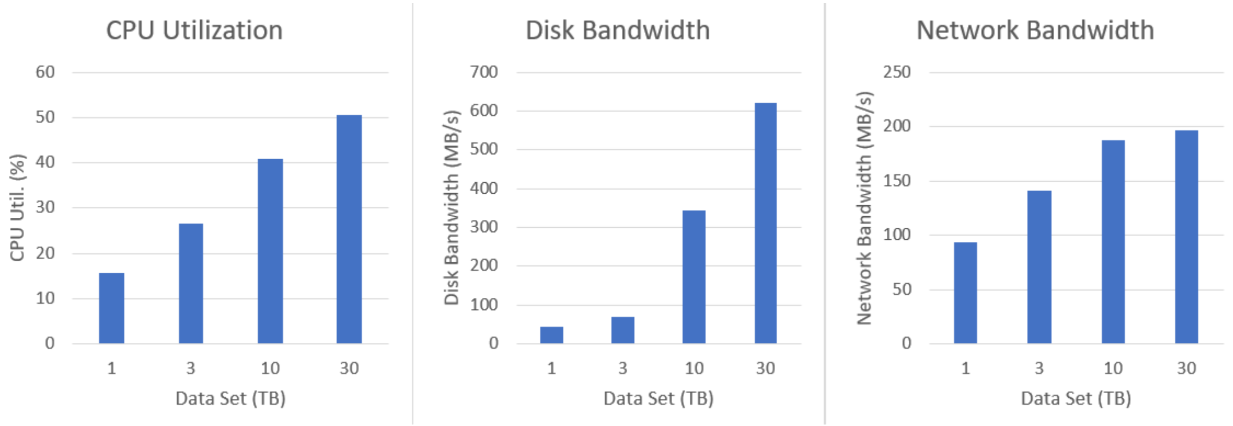


Figure 222: System Performance Per Node – Intel reference Cluster

Going from 3TB to 10TB, we observe that the disk bandwidth increases significantly. This happens since the data size exceeds the total DRAM memory across the cluster. Thus the data set needs to be fetched from the disks more often.

It is worth mentioning that we observe high disk queue length for 10TB and 30TB data set, resulting in high CPU IO wait (~4% for 10TB and ~10% for 30TB data set). Upgrading the storage with high performant Intel® NVMe based drives would improve the disk bandwidth with lower latencies at higher queue depths for modern cloud storage systems.

We also gathered detailed performance metrics corresponding to each query run and present the data as a scatter plot in Figure for the 30TB data set. The plot shows the average CPU utilization, storage bandwidth, and network bandwidth, averaged across the nodes, during the run duration of each query. We find that most queries are in the upper right quadrant of the plot, indicating high CPU utilization and storage bandwidth. As illustrated, the Microsoft SQL Server 2019 Big Data Cluster leverages the high performance of Intel® Xeon® processors and Intel® SSDs to deliver great performance for complex queries.

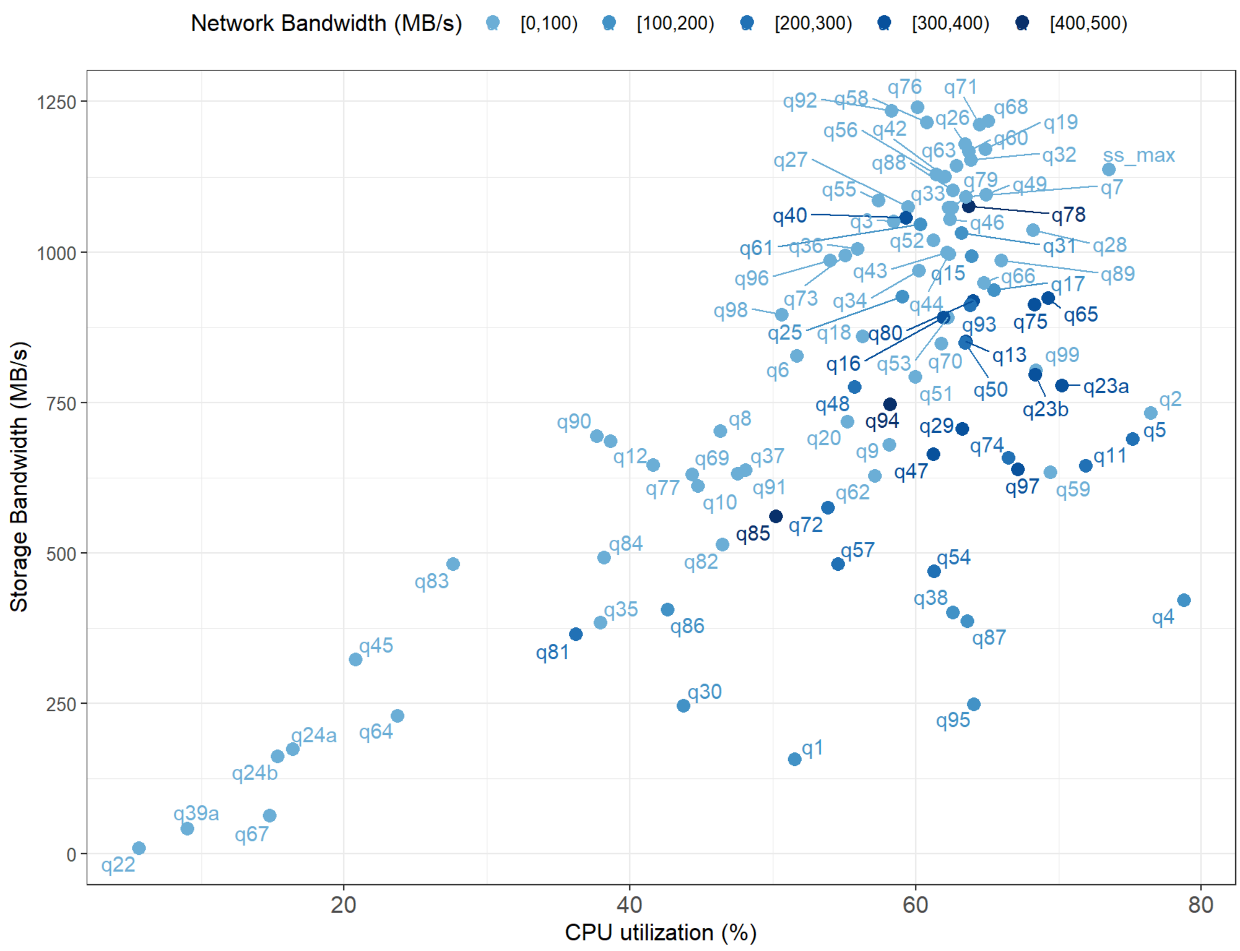


Figure 23: Performance Metrics per query - 30TB

System Performance

Built-in Grafana Monitoring

Microsoft SQL Server 2019 Big Data Cluster has built-in ready-to-use Grafana interface (see Figure 23). Grafana offers a unified view of system monitoring and performance. It is a web-based, full-featured, interactive dashboard that gives the cluster administrator full overview of the system usage. It enables creation of custom dashboards for nice and easy presentation of system metrics and visualization of the most important system information instantaneously.

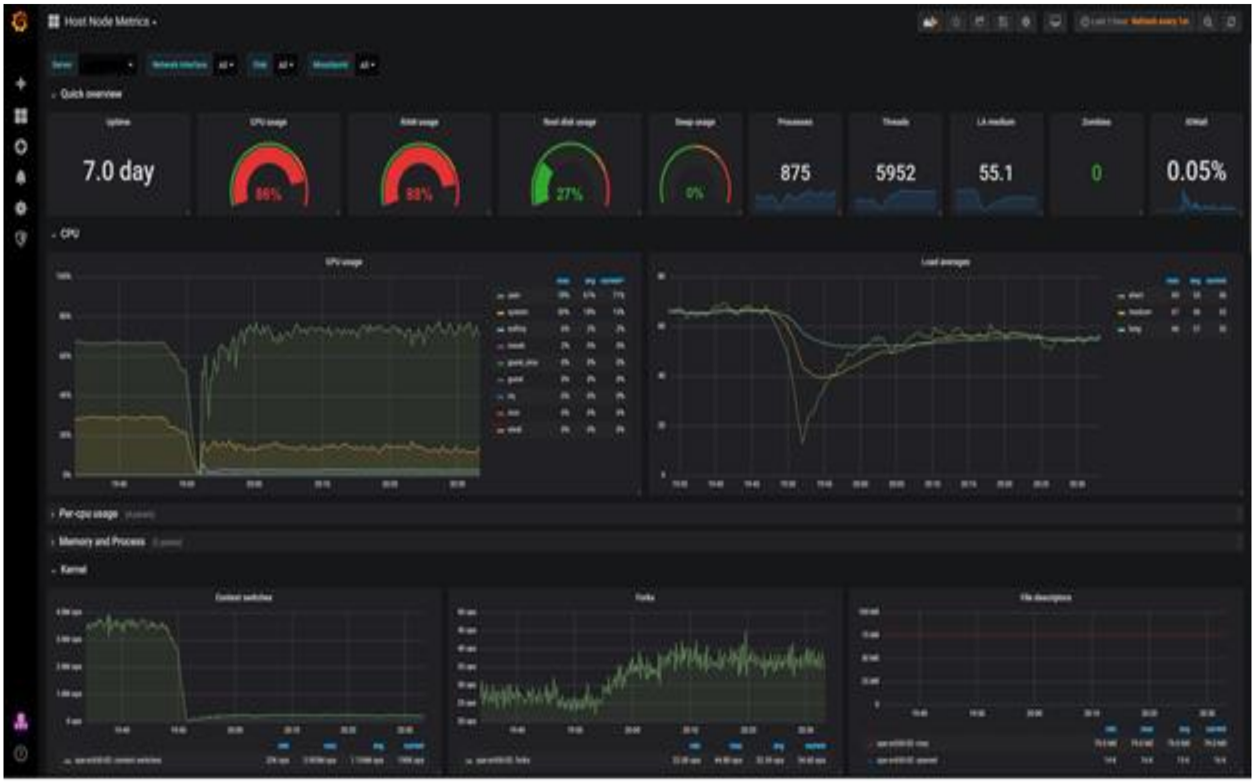


Figure 234: Built-in Grafana Monitoring Interface Showcasing System Metrics

SAR

For our measurements and, specifically, for the post-processing of these measurements, we also use Linux sysstat SAR measurements, which we detail in sar-multinodes-collect.sh script in the Appendix**.**

The processing of these measurements allowed us to compare the runs: per cluster node for all CPUs, per CPU, for memory, per storage device, and per network interface. Profiling measurements allowed us to get per-container, per module and per function.

Performance Analysis Tool (PAT)

Intel Reference cluster used the Performance Analysis Tool (PAT) to monitor and gather system-level performance metrics, including CPU, Memory, Disk, and Network. It can be configured for a single machine as well as an entire cluster of machines. We ran the PAT tool on the master node and collect metrics from all the agent nodes. We analyzed the data collected to derive the average system metrics across all the agent nodes during the workload run. We also gathered metrics to analyze individual query performance.

Summary

This first version of Microsoft SQL Server Big Data Cluster is ready for deployment for your data sets. The technology effectively utilizes Docker Containers, Kubernetes container orchestration, Apache HDFS, Apache Spark, SQL Server 2019 on Ubuntu Linux (Version 16.04) and Intel® Xeon® Processors on Intel® Data Center SSDs. We have used these reference configurations with various Big Data set sizes to characterize and tune the cluster nodes, pods and Spark parameters to help you get a head start at deploying Microsoft SQL Server 2019 Big Data Cluster, using best in class Intel® Xeon® Processors and Intel® Data Center SSDs/NVMe. With this case study, we have demonstrated that the performance scales linearly from 1TB to 100TB datasets seamlessly and the various system resources are effectively utilized. We also analyze performance with optimized SPARK tunings (SPARK Optimizer) and parameters that improve the performance of most of the queries for all the datasets. The data shows that with the SPARK Optimizer settings, customers are getting the most out of the hardware resources like CPU, memory, and Intel® Data Center SSDs.

Studies are underway to improve the performance at larger dataset sizes. Our future study would involve Intel accelerators to offload key compute operators for optimal utilization of hardware resources.

As the Microsoft SQL Server Big Data Cluster solution evolves to fulfill our customers' Big Data needs by continuously integrating Microsoft SQL Server latest solutions and Big Data cluster innovations, we will continue updating these deployment references with our performance characterizations.

References

1. https://docs.microsoft.com/en-us/sql/big-data-cluster/big-data-cluster-overview?view=sqlallproducts-allversions
2. https://docs.microsoft.com/en-us/sql/big-data-cluster/deployment-guidance?view=sqlallproducts-allversions#configfile
3. https://info.microsoft.com/ww-landing-SQL-Server-2019-Big-Data-WhitePaper.html
4. http://www.tpc.org/information/benchmarks.asp
5. https://github.com/databricks/spark-sql-perf
6. https://databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html
7. https://docs.microsoft.com/en-us/sql/big-data-cluster/concept-data-persistence
8. https://github.com/microsoft/sql-server-samples/tree/master/samples/features/sql-big-data-cluster/deployment/kubeadm/ubuntu
9. https://blog.cloudera.com/how-to-tune-your-apache-spark-jobs-part-2/
10. https://github.com/intel-hadoop/PAT
11. https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/platinum-processors/platinum-8160.html
12. https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/gold-processors.html
13. https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/data-center-ssds/d7-series/dc-p4510-series.html
14. https://www.lenovo.com/us/en/data-center/servers/racks/ThinkSystem-SR650/p/77XX7SRSR65
15. https://spark.apache.org/docs/latest/sql-performance-tuning.html
16. https://arch-long.cn/articles/spark%20sql/Spark-SQL-Configurations.html

Appendix

TPC-DS Schema Based Queries

We use Databricks’ TPC-DS Spark SQL kit to run the read-only suite of 99 queries, some of which are split into queries a and b, resulting in 104 Spark SQL queries as listed in Table below:

Table 6: List of TPC-DS Queries

Query-1	**Query - 14a**	Query - 24b	**Query - 37**	Query - 49	Query - 62	Query - 75	Query - 88
Query-2	**Query - 14b**	Query - 25	Query - 38	Query - 50	Query - 63	Query - 76	Query - 89
Query - 3	Query - 15	Query - 26	Query - 39a	Query - 51	Query - 64	Query - 77	Query - 90
Query - 4	Query - 16	Query - 27	Query - 39b	Query - 52	Query - 65	**Query - 78**	Query - 91
Query - 5	Query - 17	Query - 28	**Query - 40**	Query - 53	Query - 66	**Query - 79**	Query - 92
Query - 6	Query - 18	Query - 29	Query - 41	**Query - 54**	Query - 67	Query - 80	Query - 93
Query - 7	Query - 19	Query - 30	Query - 42	Query - 55	Query - 68	Query - 81	Query - 94
Query - 8	Query - 20	Query - 31	Query - 43	Query - 56	Query - 69	**Query - 82**	Query - 95
Query - 9	Query - 21	Query - 32	Query - 44	Query - 57	Query - 70	Query - 83	Query - 96
Query - 10	Query - 22	Query - 33	Query - 45	Query - 58	Query - 71	Query - 84	Query - 97
Query - 11	Query - 23a	Query - 34	Query - 46	Query - 59	Query - 72	Query - 85	Query - 98
Query - 12	**Query - 23b**	Query - 35	Query - 47	Query - 60	Query - 73	Query - 86	Query - 99
Query - 13	**Query - 24a**	Query - 36	Query - 48	Query - 61	Query - 74	Query - 87	Query - s_max

The TPC-DS queries are designed to scan large volumes of data. The operation that naturally fits this design is join operation. The Apache Spark, by default, is using Sort Merge Join, and in some of the query it is consuming all available Spark executor memory when query execution is done against large data sets like in the case of the 100TB data set.

Microsoft Reference cluster

In the Microsoft reference cluster, the 94 queries not colored in blue, listed in the above table are used with 1TB/10TB/100TB data sets. Tests are executed with Open Source Apache Spark 2.4 running on the Microsoft SQL Server 2019 Big Data Cluster. The collected results show excellent performance and scaling capabilities of Microsoft SQL Server Big Data Cluster when different data sets are used.

Intel Reference Cluster

In Intel reference cluster, queries 14a,14b did not run for the30TB data set due to memory limitations. Regarding query 64, as discussed earlier, increasing spark.sql.cbo.joinReorder.The threshold significantly improves performance for the 30TB data set. However, at lower data sets, high optimization time reduces its performance, significantly affecting the total runtime. To keep the metrics comparable across data sets, we removed the results for queries 14a,14b,64 across all data sets.

Storage setup

Logical Volume for storage

This section outlines the steps for creating an LVM across three physical disks mounted at `/dev/sd[b,c,d]`, respectively.

1. Create physical volumes for three disks
   ```
   sudo pvcreate /dev/sdb /dev/sdc /dev/sdd
   ```
2. Create volume group
   ```
   sudo vgcreate local-storage /dev/sdb /dev/sdc /dev/sdd
   ```
3. Create logical volume with default stripe size
   ```
   sudo lvcreate --extents 100%FREE -n test local-storage --stripes 3
   ```
4. Format logical volume
   ```
   sudo mkfs.ext4 /dev/mapper/local--storage-test
   ```
5. Mount it on '/mnt/local-storage'
   ```
   sudo mount /dev/mapper/local--storage-test /mnt/local-storage
   ```
6. Add the following entry to `/etc/fstab'
   ```
   /dev/mapper/local--storage-test /mnt/local-storage ext4 defaults 0 0
   ```
7. Verify the logical volume
   ```
   sudo lvdisplay -vm
   ```

Move Docker and Kubelet working directory

For best storage IO performance, we moved the Docker and kubelet working directory from OS disk (default) to the striped logical volume.

Docker

Run all the following commands in superuser mode

1. Create directory
   ```
   mkdir -p /mnt/local-storage/docker
   ```
2. Change Docker daemon path
   In file `/lib/systemd/system/docker.service`

   Change from:
   ```
   ExecStart=/usr/bin/dockerd -H fd://  --containerd=/run/containerd/containerd.sock
   ```
   To:
   ```
   ExecStart=/usr/bin/dockerd  -g /mnt/local-storage/docker -H fd://  --containerd=/run/containerd/containerd.sock
   ```
3. Stop all docker processes
   ```
   systemctl stop docker
   ```
   Check if the process stopped - `ps aux | grep -i docker | grep -v grep`
4. Reload and rsync folders
   ```
   rsync -aqxP /var/lib/docker/ /mnt/local-storage/docker
   ```
5. Start docker
   ```
   systemctl start docker
   ```
   Check that it runs with the new path - `ps aux | grep -i docker | grep -v grep`

Kubernetes

Run all the following commands in superuser mode

1. Create directory
   ```
   mkdir -p /mnt/local-storage/kubelet/
   chmod 777 /mnt/local-storage/kubelet
   ```
2. Change kubelet root directory
   ```
   vim /etc/default/kubelet
   #Add line
   KUBELET_EXTRA_ARGS=--root-dir=/mnt/local-storage/kubelet/
   ```
3. Restart kubelet
   ```
   systemctl daemon-reload
   systemctl restart kubelet
   ```

sar-multinodes-collect.sh

```
#!/bin/bash
if [ -z $1 ]; then
    # default to 30 seconds collection.
    DURATION=00:00:30
else
    if [[ $1 =~ [0-9]+:[0-9]+:[0-9]+ ]];
    then
        DURATION=$1
    else
        echo “Usage: $0 HH:MM:SS”
        exit 1
    fi
fi
DURATION_IN_SECONDS=`echo $DURATION | awk -F’:’ ‘{print $1 * 60 * 60 + $2 * 60 + $3}’`
export WCOLL=/tmp/cluster/clusternodes
export PDSH_RCMD_TYPE=ssh
export SAR_DIR=/data/sar
export SAR_OPTS=”-dpqrwWB -u ALL -P ALL -n DEV -n EDEV -I SUM”
export SAR_INTERVAL=10
export SAR_DURATION_IN_COUNTS=`echo “$DURATION_IN_SECONDS/$SAR_INTERVAL” |bc`
TAG=`date +%m%d_%H%M`
pdsh “mkdir -p $SAR_DIR”
pdsh “sar $SAR_OPTS $SAR_INTERVAL $SAR_DURATION_IN_COUNTS > $SAR_DIR.$TAG 2>&1”
rpdcp “$SAR_DIR.$TAG” $SAR_DIR
pdsh “rm -f $SAR_DIR.$TAG”
```

Microsoft Reference Cluster – Logical Volumes Configuration Details

Per Server block physical partitions, with 5 data disks:

```
brw-rw---- 1 root disk 259, 16 Oct 11 16:35 /dev/nvme0n1
brw-rw---- 1 root disk 259,  0 Oct 11 16:35 /dev/nvme1n1
brw-rw---- 1 root disk 259,  2 Oct 11 16:35 /dev/nvme2n1
brw-rw---- 1 root disk 259,  1 Oct 11 16:35 /dev/nvme3n1
brw-rw---- 1 root disk 259, 12 Oct 11 16:35 /dev/nvme4n1
```

```
/dev/sda1            Partition
/dev/sda2            Partition
/dev/sda3            Partition
/dev/nvme0n1p1       Partition
```

```
/dev/nvme0n1p2       Partition
/dev/nvme0n1p3       Partition
/dev/nvme1n1p1       Partition
/dev/nvme1n1p2       Partition
/dev/nvme1n1p3       Partition
/dev/nvme2n1p1       Partition
/dev/nvme2n1p2       Partition
/dev/nvme2n1p3       Partition
/dev/nvme3n1p1       Partition
/dev/nvme3n1p2       Partition
/dev/nvme3n1p3       Partition
/dev/nvme4n1p1       Partition
/dev/nvme4n1p2       Partition
/dev/nvme4n1p3       Partition
```

The logical partitions were created with:

```
# sudo mdadm --create --verbose /dev/md0 --level=raid0 --raid-devices=5
/dev/nvme0n1p1 /dev/nvme1n1p1 /dev/nvme2n1p1 /dev/nvme3n1p1
/dev/nvme4n1p1
# sudo mdadm --create --verbose /dev/md1 --level=raid0 --raid-devices=5
/dev/nvme0n1p2 /dev/nvme1n1p2 /dev/nvme2n1p2 /dev/nvme3n1p2
/dev/nvme4n1p2
# sudo mdadm --create --verbose /dev/md2 --level=raid0 --raid-devices=5
/dev/nvme0n1p3 /dev/nvme1n1p3 /dev/nvme2n1p3 /dev/nvme3n1p3
/dev/nvme4n1p3
#
# sudo mount /dev/md0 /data
# sudo mount /dev/md1 /mssql
# sudo mount /dev/md2 /general
```

Resulting in the following logical partitions:

```
NAME        MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sda           8:0    0 447.1G  0 disk
├─sda1        8:1    0   512M  0 part  /boot/efi
├─sda2        8:2    0 445.7G  0 part  /
└─sda3        8:3    0   977M  0 part
nvme0n1     259:0    0   7.3T  0 disk
├─nvme0n1p1 259:1    0     5T  0 part
│ └─md0       9:0    0    25T  0 raid0 /data
├─nvme0n1p2 259:2    0 465.7G  0 part
│ └─md1       9:1    0   2.3T  0 raid0 /mssql
└─nvme0n1p3 259:3    0   1.8T  0 part
  └─md2       9:2    0   9.1T  0 raid0 /general
nvme1n1     259:6    0   7.3T  0 disk
├─nvme1n1p1 259:11   0     5T  0 part
│ └─md0       9:0    0    25T  0 raid0 /data
├─nvme1n1p2 259:13   0 465.7G  0 part
│ └─md1       9:1    0   2.3T  0 raid0 /mssql
└─nvme1n1p3 259:16   0   1.8T  0 part
  └─md2       9:2    0   9.1T  0 raid0 /general
nvme2n1     259:4    0   7.3T  0 disk
├─nvme2n1p1 259:7    0     5T  0 part
│ └─md0       9:0    0    25T  0 raid0 /data
├─nvme2n1p2 259:8    0 465.7G  0 part
│ └─md1       9:1    0   2.3T  0 raid0 /mssql
└─nvme2n1p3 259:9    0   1.8T  0 part
  └─md2       9:2    0   9.1T  0 raid0 /general
nvme3n1     259:5    0   7.3T  0 disk
├─nvme3n1p1 259:12   0     5T  0 part
```

```
│ └─md0        9:0    0    25T  0 raid0 /data
├─nvme3n1p2 259:14   0 465.7G  0 part
│ └─md1        9:1    0   2.3T  0 raid0 /mssql
└─nvme3n1p3 259:15   0   1.8T  0 part
  └─md2        9:2    0   9.1T  0 raid0 /general
nvme4n1     259:10   0   7.3T  0 disk
├─nvme4n1p1 259:17   0     5T  0 part
│ └─md0        9:0    0    25T  0 raid0 /data
├─nvme4n1p2 259:18   0 465.7G  0 part
│ └─md1        9:1    0   2.3T  0 raid0 /mssql
└─nvme4n1p3 259:19   0   1.8T  0 part
```

Examples of system resources consumptions under these workloads

Captured on the Microsoft cluster configuration:

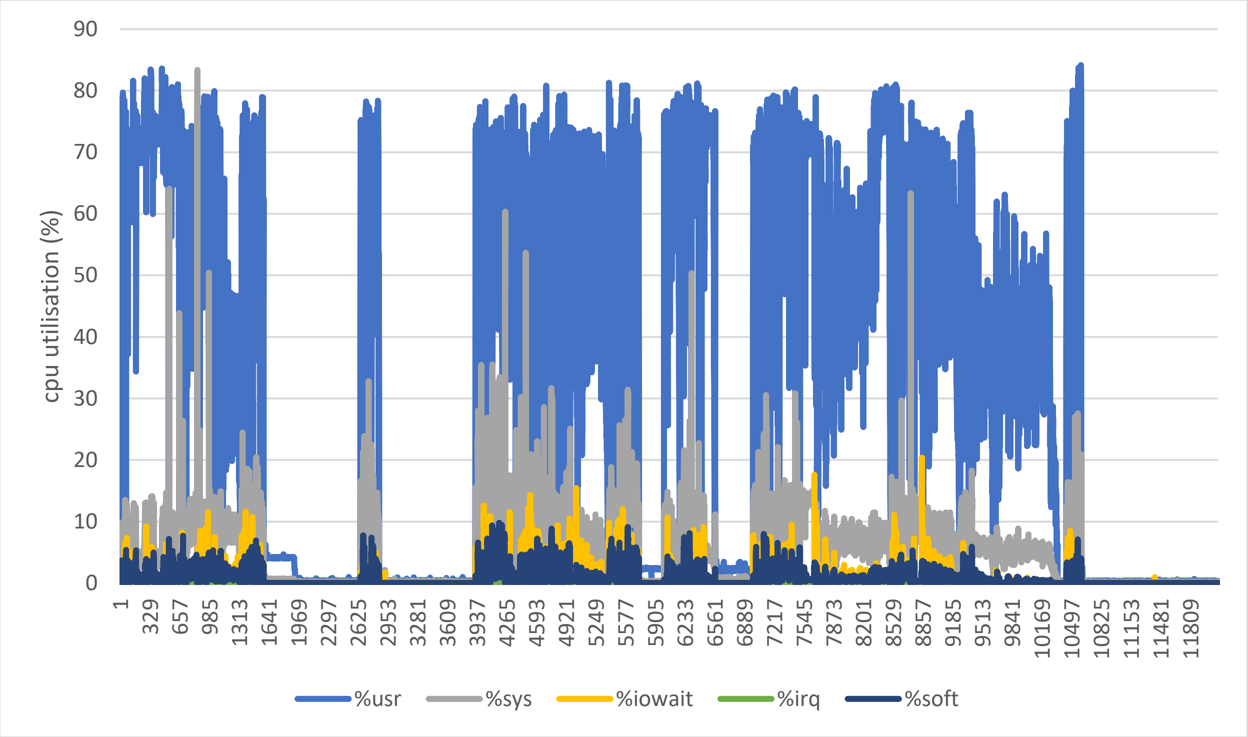


Figure 245: 100TB Optimized – Agent node CPU utilization

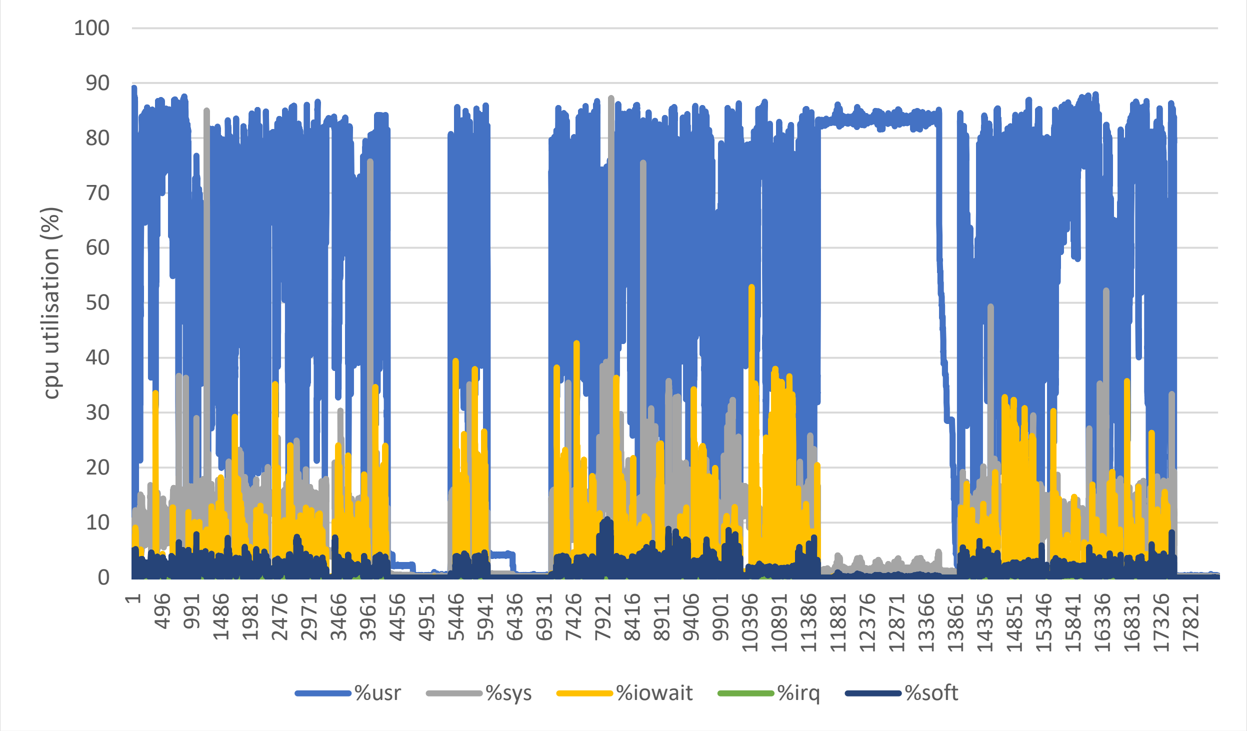


Figure 256: 100TB Baseline - Agent node cpu utilization

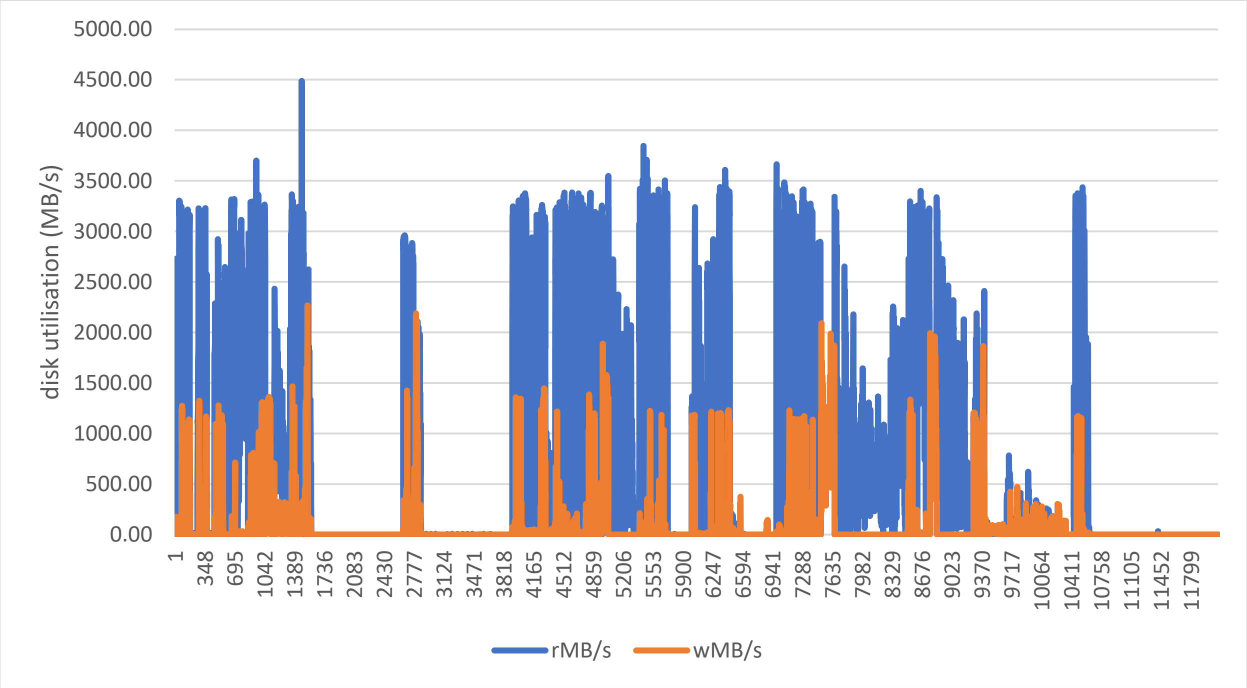


Figure 267: 100TB Optimized - Agent node disk utilization

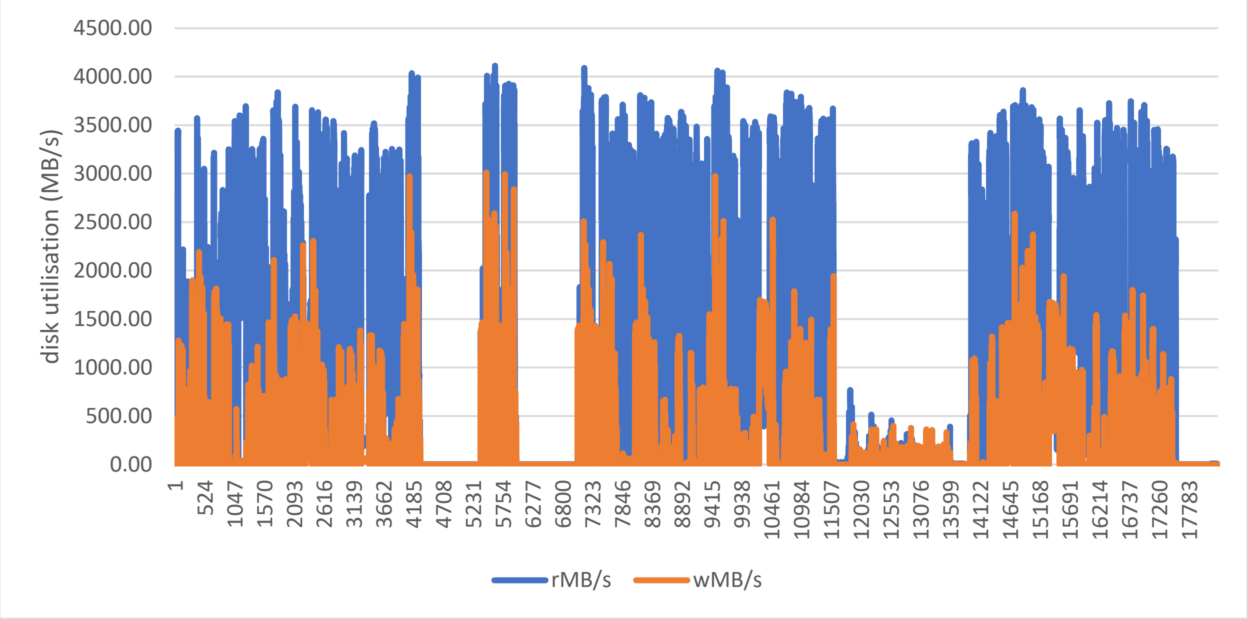


Figure 278: 100TB Baseline – Agent node disk utilization

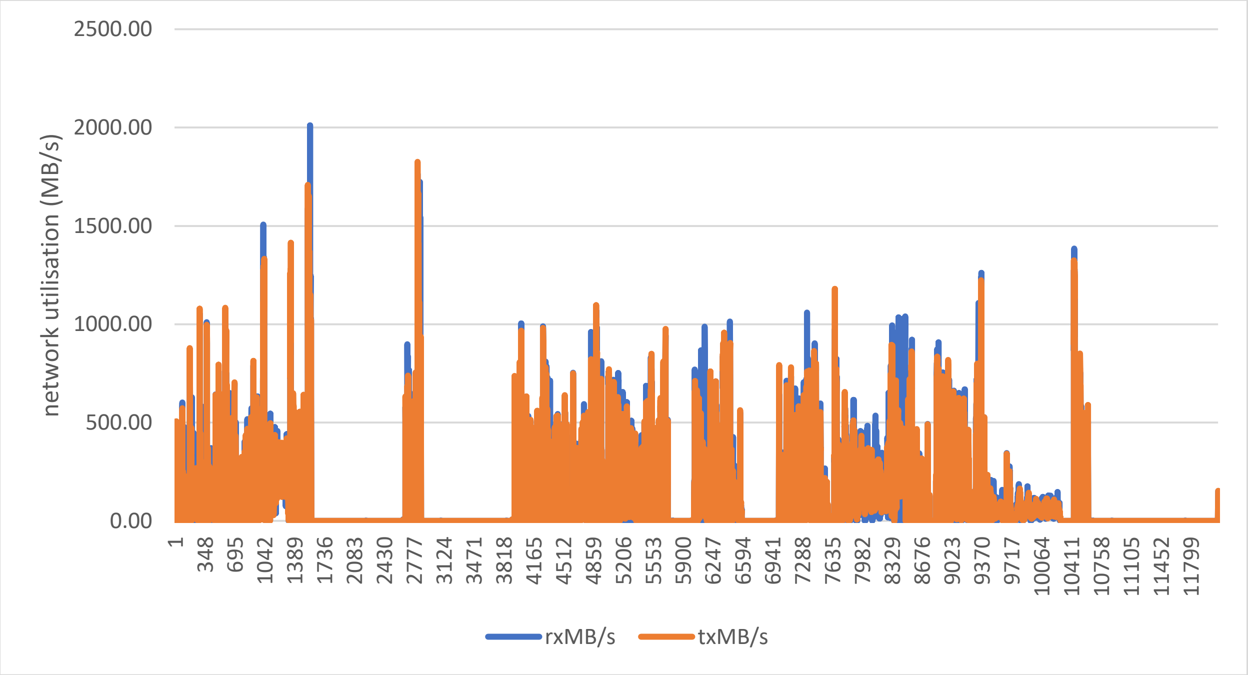


Figure 28: 100TB Optimized - Agent node network utilization

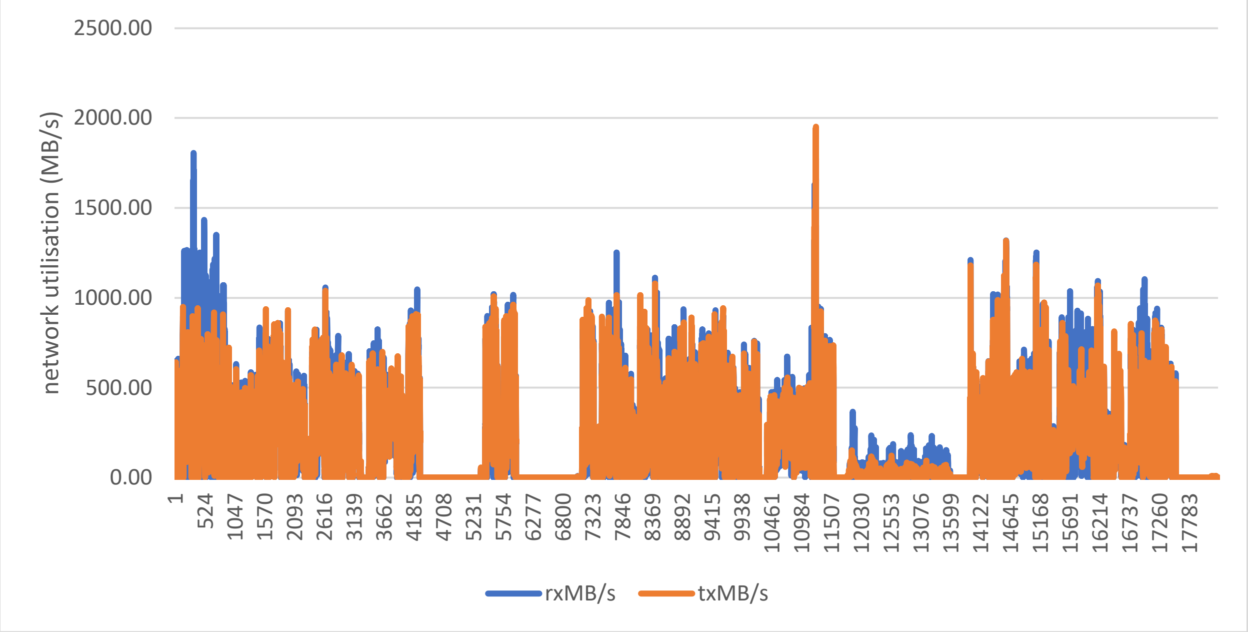


Figure 290: 100TB Baseline – Agent node network utilization

To provide references for the 10TB Optimized runs:

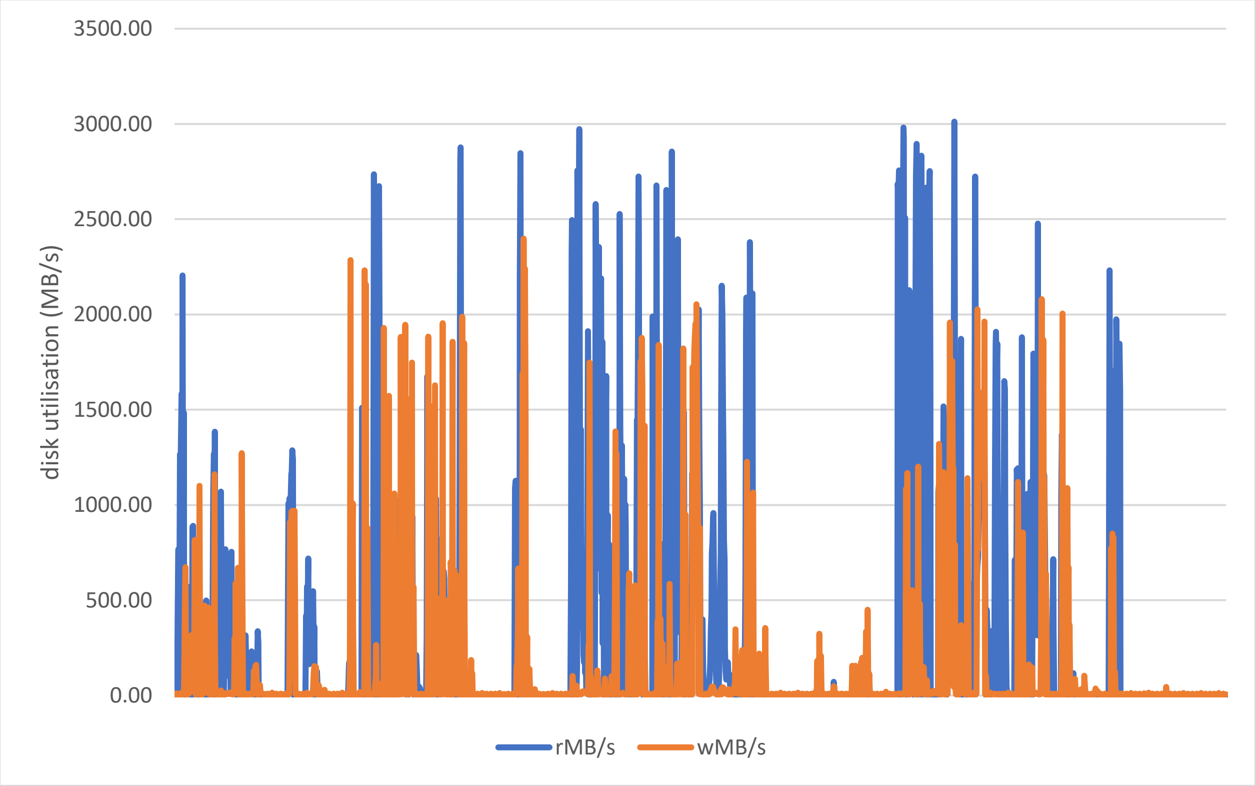


Figure 301: 10TB Optimized – Agent node disk utilization

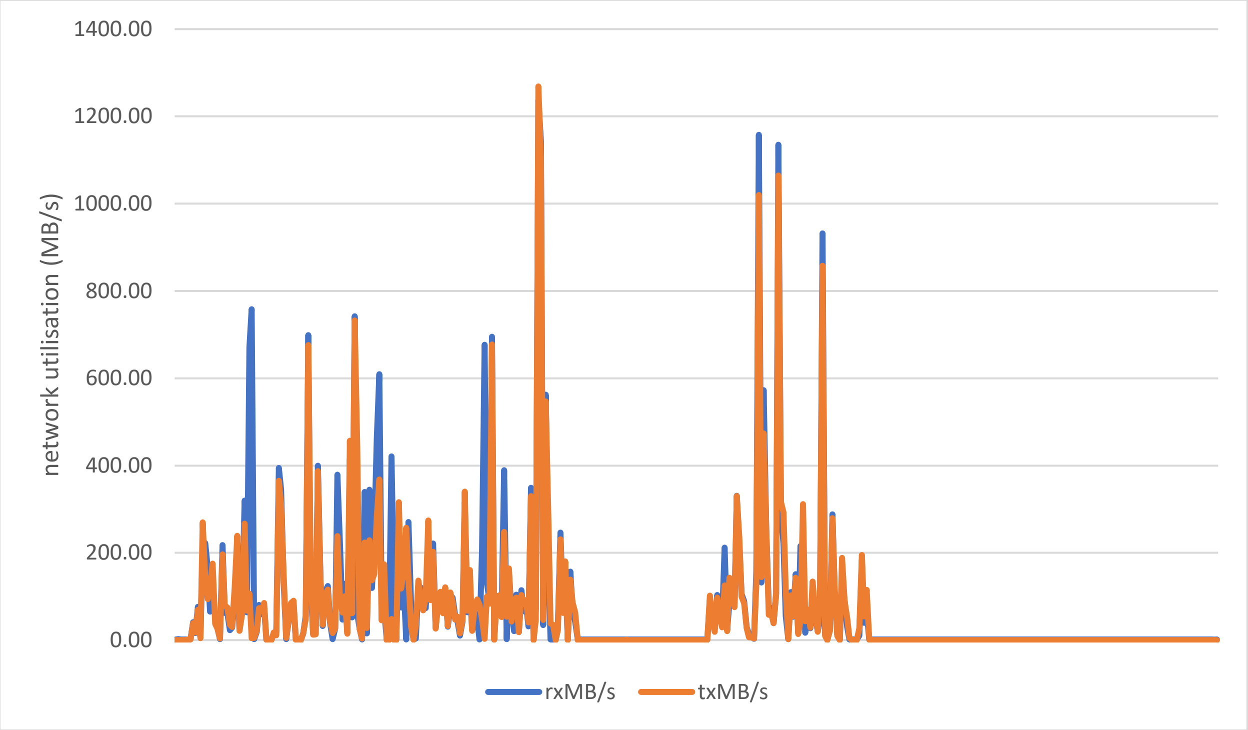


Figure 312: 10TB Baseline - Agent node network utilization

List of Figures

List of Tables