

Accelerating Large-Scale Business Analytics

Using a 100TB dataset powered by the Intel® Xeon® Processor E7 Family and Microsoft® SQL Server 2016

Authors Executive Summary

Mahmut Aktasoglu
Intel Corporation

Avaneesh Shetty
Intel Corporation

Hamesh Patel
Intel Corporation

Kevin Farlee
Microsoft Corporation

Sadashivan Krishnamurthy
Microsoft Corporation

The creation, organization, and distribution of content across data centers continues to rise at an ever-increasing pace. With an increasing amount of data being stored in modern data warehouses, organizations need solutions that scale proportionally. Although building clusters of servers and distributing data across the systems is one of the more common approaches to scale, it is not desirable from either a performance or TCO point of view.

In this paper, we show how Microsoft® SQL Server technology in a single Intel® Xeon® processor-based platform can effectively perform advanced analytics on a large dataset (100TB in this case), using Microsoft® SQL Server 2016 and the Intel® Xeon® processor E7-8890 v4.¹

Microsoft®
SQL Server®

The work is the culmination of a thoroughly tested configuration including meticulous design aspects that took care in sizing key hardware components and optimal Microsoft® SQL Server parameters to provide meaningful insights on large datasets. Results shared in this paper include:

- The system loaded a 100TB dataset—the largest scale factor allowed by TPC-H
- The SQL Server 2016 columnstore engine compressed the data by 3x and loaded the data at a rate of 1.6TB/hour
- Using the Intel® Xeon® processor E7-8890 v4, we observed up to 1.36x average speed up—compared to the Intel® Xeon® processor E7-8890 v3—for complex ad-hoc queries against a 100TB dataset
- Using the Intel® AVX2 instruction set in the columnstore engine provides up to 2.3x performance boost for in-memory execution
- It took **less than 2.5 minutes**, processing **4.3B rows/second** when data is in memory

The intended audience for this paper includes data analytics experts, IT executives and managers, solution architects, and infrastructure planners.

Contents

Executive Summary 1

Technology 2

 Intel® Xeon® Processor-Based Platforms..... 2

 Intel® Data Center SSDs 2

 Intel® RAID Controllers 2

Microsoft® SQL Server 2016: Data Warehousing with Improved Columnstore Technology 2

System Configuration..... 3

Database Creation and Performance Testing 6

 Building the Database 6

 Test Queries 6

Results & Analysis 7

 The New Intel® Xeon® Processor E7 v4 Family Performance Combined with Improved SQL Server 2016 Columnstore Engine 7

 SQL Server 2016 Optimized for Intel® AVX2 8

Summary 9

Acknowledgements 10

Appendix A 11

Test Queries 11

 Test Query 1 11

 Test Query 2 11

 Test Query 3 11

 Test Query 4 12

 Test Query 5 12

 Test Query 6 13

 Test Query 7 13

 Test Query 8 14

 Test Query 9 14

 Test Query 10 15

 Test Query 11 15

 Test Query 12 15

Appendix B 16

 Building the Database 16

 Performance Tests with the Intel® Xeon® Processor E7-8890 v3 .. 16

In-Memory Performance of Columnstore Engine 17

Performance Scaling with Data Size..... 17

Performance of Running Queries Concurrently 18

Technology

Intel® Xeon® Processor-Based Platforms

In our configuration, we rely on the advantage of in-memory computing for fast insights and decision making. Infrastructure built on the Intel® Xeon® processor E7-8800 v4 product family can deliver real-time analytics services and open up new data-driven business opportunities. Designed for the most mission-critical workloads and the always-on enterprise, the processor combines large memory capacities with high performance, reliability, and virtualization capabilities to keep data centers supplying business advantage without interruption.²

Intel® Data Center SSDs

To ensure that data centers can keep up with the speed of business, all that's needed is an I/O-optimized infrastructure using the extraordinary performance and reliability of Intel® Data Center SSDs.³ By pairing Intel® Data Center SSDs with Intel® CPUs, chipsets, firmware, and drivers, we built a seamless system, enabling large amounts of data to be transferred to processors quickly, eliminating I/O bottlenecks. Faster data servers give organizations a competitive edge as they deal with ever-increasing infrastructure demands.²

Intel® RAID Controllers

SAS-based Intel® RAID controllers, featuring the dual-core LSI SAS3108 RAID-on-Chip (ROC) processor, offer significant performance enhancements for solutions architected with 12Gb/s or 6Gb/s SAS drives.⁴ The read/write performance ideally suits the controllers for a broad range of application workloads, such as enterprise data center applications, cloud computing, and content applications. The massive growth of cloud and big data applications requires enterprise features to manage, optimize, and improve the efficiency of growing data centers. The 12Gb/s SAS Intel® RAID controller StorCLI utility features server storage management, either locally or remotely, from a single pane of glass with the RAID Web Console application. This generation of storage, spurred by the increased adoption and trending lower costs of flash, requires more bandwidth and I/O per second (IOPS) capability than the previous generation. In addition to outstanding performance, the Intel® 12Gb/s SAS product family supports advanced features such as FastPath I/O and RAID SSD Cache, dual-level advanced RAID types, and revertible hot spare.²

Microsoft® SQL Server 2016: Data Warehousing with Improved Columnstore Technology⁵

Microsoft® SQL Server 2016 has made significant improvements in data warehousing technologies and performance, including columnstore features, as well as many other improvements.

Columnstore indices offer great advantages over traditional row stores for analytics and data warehousing queries. They are ideally suited for star schemas and tables with billions of rows, which are commonly seen. Among their advantages for analytics are:

Up to 10x Compression in Data Size

Data warehouses are very large by nature, and the compression offered by columnstore index technologies offers not only space and cost savings, but also significantly increased performance. These benefits are due to the dramatically reduced I/O requirements given by the compression, coupled with the ability to scan only the specific columns required by each query. This compression also reduces the amount of memory required to hold a given number of rows from the source data warehouse.

Additional Indices

SQL Server 2016 adds the capability to add additional (B-Tree) indices to columnstore-based tables, which enables efficient single-row lookup. In addition to these architectural features, we have further optimized the processing of queries in columnstore indices in the following ways:

Operator Pushdown

Pushdown refers to moving both filter and aggregation query operations closer to the data so that many of the filters and calculations can be done in the scan operators, dramatically reducing the volume of data that needs to be handled further on in query processing.

Batch-Mode Processing

SQL Server 2016 includes enhancements in batch-mode processing, which processes many rows at a time rather than doing calculations serially on each individual row. These batch operations are further optimized by leveraging Single Instruction, Multiple Data (SIMD) vector-processing CPU instructions in the Intel® architectures.

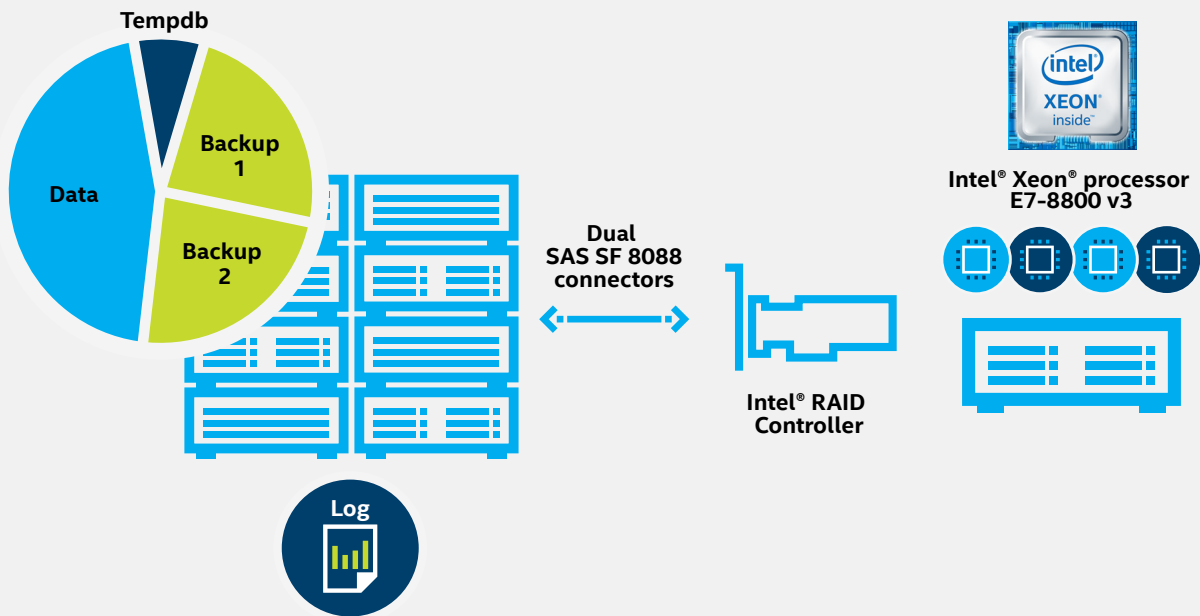


Figure 1. Hardware Configuration⁶

System Configuration

The Intel® technology-powered white box used is a 4U Intel® Server System that supports four Intel® Xeon® processor E7-8890 v4 product family processors.

Attached to the platform, we use 8 LSI 620J JBOD storage enclosures (see Figure 1⁶), which are dual-connected to the Intel® RAID controllers via SAS SF-8088 cables, which provides 4 external lanes with a 12Gb/s transfer rate per lane. We used the directly attached storage units for storing data (i.e., raw data and database files), tempdb (i.e., SQL Server system database used for query execution), and backups. The most crucial performance aspect of our storage subsystem for our workload was read bandwidth for data and tempdb drives. Before sizing and configuring the drives, we ran Iometer tests for 512K sequential reads to measure I/O bandwidth of each storage unit, populated by 1.6TB Intel® SSDs. In the graph below, you can observe the amount of I/O bandwidth that can be attained as we add more disks to the enclosure.

BANDWIDTH (MB/s) vs. NUMBER OF DISKS IN A BAY

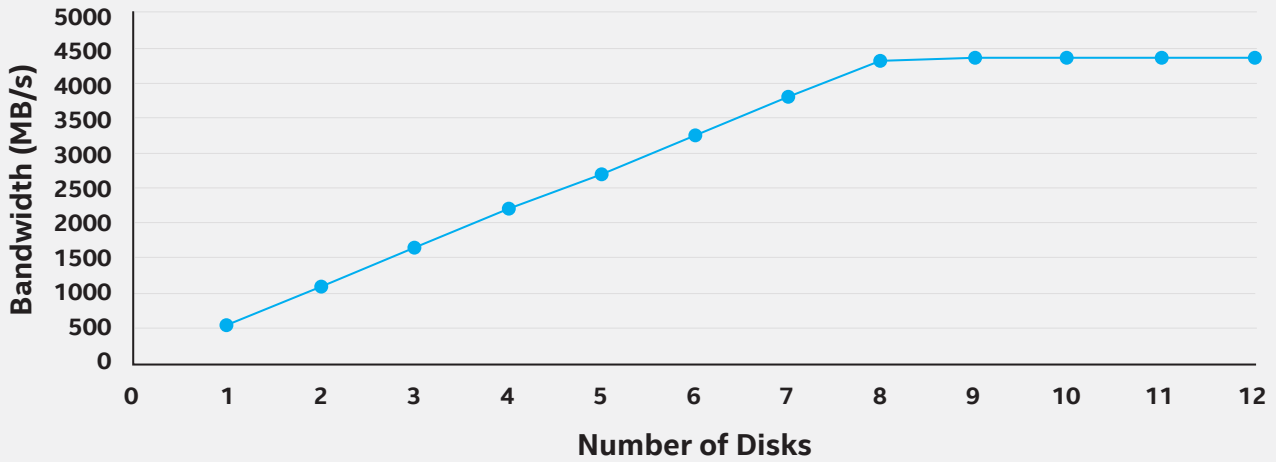


Figure 2. Bandwidth Scaling with Number of Drives

As can be seen from Figure 2, Intel® RAID cards can easily scale up to an approximate 4.5GB/s data transfer rate with 8 SSDs. Beyond that point, we are limited by SAS transfer rates. Based on our observation, we used 8 SSDs for our data drives per enclosure to provide peak I/O bandwidth and ample storage (about 11TB per data drive, 88TB total capacity). We measured total I/O bandwidth of **over 25GB/s** across all data drives with our configuration detailed below.

Our storage configuration is as follows: Each storage bay is populated with 24 x 1.6TB 2.5" Intel® SSD DC S3500 Series drives. We created RAID0 across 8 SSDs for data drives and 4 SSDs for tempdb drives. We created RAID6 over the remaining 12 SSDs to hold our database backup files, ensuring good performance with adequate redundancy (i.e., around 14.5TB per bay, which allows us to keep 2 sets of backups). Lastly, for SQL log files, we included a storage unit with 24 x 800GB 2.5" Intel® SSDs daisy-chained to one of the main storage bays. Figure 1 illustrates the storage layout of our system.

As for system memory, we fully populated our DIMM sockets with 64GB DDR4 LRDIMMs, giving us a total memory capacity of 6TB. To make use of the whole physical address space, we used Microsoft® Windows Server 2016 RTM build, which supports memory capacity up to 12TB.

Table 1 lists the specifics about the components we used in our configuration.

CPU		
Processor	Intel® Xeon® processor E7-8890 v4	
# Cores/Threads	24/48	
Last Level Cache	60MB	
Base/Turbo Frequency	2.2GHz/3.4GHz	
QPI	3 links, 9.6GT/s	
Instruction Set Extensions	AVX2	
PLATFORM		
# Sockets	4	
Chipset	C600/X79 Express Chipset	
# PCI Express® (PCIe®) ports	12	
Networking	2x Intel® Network Adapter X540-T2 (10Gb/s)	
STORAGE		
SSDs – Data + Tempdb + Backups	Intel® SSD DC S3500 Series (1.6TB, 2.5" SATA 3.0)	x192
SSDs – Log	Intel® SSD DC S3700 Series (800GB, 2.5" SATA 3.0)	x24
RAID Controllers	Intel® RAID Controller RS3SC008 PCIe x8 Gen3	x8
Storage Enclosure	LSI LSI00217 / 620J 24-Bay 2U Rackmount JBOD	x9
MEMORY		
Memory Type	DDR4	
DIMM Size and Type	64GB LRDIMM, dual rank	
Memory Frequency (MHz)	2133	
DIMMs/Channel	3	
System Memory Capacity (GB)	6144	
SOFTWARE		
Database Software	Microsoft® SQL Server 2016 Enterprise RTM 64-bit Edition (build no: 13.0.1601.5)	
OS Distribution	Microsoft® Windows Server 2016 TP5 64-bit (build no: 14300)	

Table 1. System Configuration Details

Database Creation and Performance Testing

Our choice of workload to put our system to the test is a 100TB data warehouse database. We constructed our database based on the TPC-H schema. The TPC-H benchmark* models a decision-support system for worldwide sales and distribution of arbitrary products. It provides guidelines to define size of business (i.e., scale factor), which in turn dictates the total size of the data. The size of the database we chose, 100TB, is the biggest scale factor allowed by TPC-H. There are nearly 600 billion rows in the fact table (i.e., LINEITEM) representing orders data over the course of 7 years. Table 2 summarizes cardinality of each table in the database.

Building the Database

Using SQL Server 2016, we created the file group, consisting of 512 data files of 100GB size, and mapped the database table onto this file group.⁷ Next, we defined a clustered columnstore index (CCI) on the tables and used a week-based partitioning scheme applied to the largest two tables of the schema (i.e., LINEITEM and ORDERS). Following database creation, we generated data in batches of 10TB. Note that we assume the data for the largest two tables is presorted and preprocessed (i.e., split into subsets of one-week data) to optimize parallel loading. Generating 10TB worth of data at a time allowed us to build the database on the same configuration with no extra storage overhead required to keep the raw data. Once data is generated, we loaded data into columnstore tables using BULK INSERT (BI) statements. As data is loaded into the database, the SQL Server 2016 columnstore engine **compresses the data by 3x**, which greatly reduces storage requirements. It took about 6 hours to load 10TB of data onto our system. This is a very impressive result, showing that SQL Server 2016 is capable of loading 100TB worth of data into a columnstore database in 60 hours at a rate of **1.6 TB/hour**.

Once all data was loaded, we created non-clustered indices (NCI) in addition to the CCIs on 5 tables (PART, CUSTOMER, SUPPLIER, NATION, and REGION), which took about 1 hour.

TABLE NAME	CARDINALITY
LINEITEM	599,999,969,200
ORDERS	150,000,000,000
PARTSUPP	80,000,000,000
PART	20,000,000,000
CUSTOMER	15,000,000,000
SUPPLIER	1,000,000,000
NATION	25
REGION	5

Table 2. Table Cardinality

Support for additional indices on tables with columnstore indices is a new capability introduced in SQL Server 2016. Having multiple indices—optimized for different logical operations—allows the SQL Server query optimizer to come up with the most efficient query plan, improving query performance. Furthermore, we generated statistics to improve estimations made by the query optimizer, which is crucial for optimized query plans and better performance. After the 4.7 hours of creating the statistics, the ETL phase was complete. The database consumed about 45TB of storage space. We finally created the database backup to be able then to seamlessly restore for continuous measurements as shown in Figure 3.

Test Queries

To test performance with the Intel® Xeon® processor E7-8890 v4, we restored the database created from our backup files and ran a subset of ad-hoc queries derived from the TPC-H Benchmark standard (see Appendix A for details) against the 100TB columnstore database. In this paper, we provide results of three tests to highlight various performance aspects of the system under test. Brief descriptions of the queries are provided in Table 3.

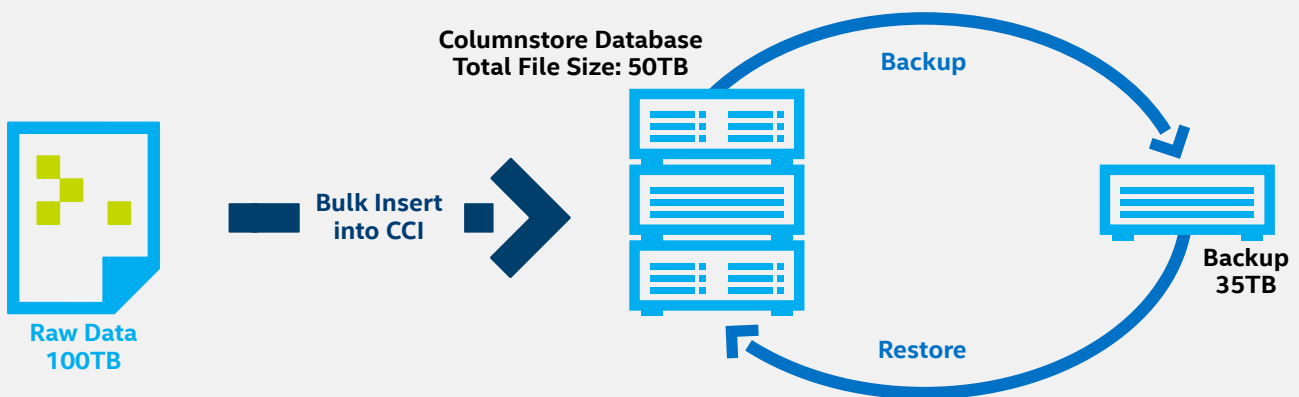


Figure 3. Load and Backup/Restore Process

Results & Analysis

The New Intel® Xeon® Processor E7 v4 Family Performance Combined with Improved SQL Server 2016 Columnstore Engine

The Intel® Xeon® processor E7 v4 family is the newest member of data center processors designed for the performance needs in data platforms. The Intel® Xeon® processor E7 v4 family has up to 48 logical cores—a 33% increase compared to its predecessor—in terms of the maximum degree of parallelism that can be attained.

In our first experiment, we explore the benefits of the columnstore engine—first introduced in SQL Server 2012—in SQL Server 2016. A common characteristic of data warehouse applications is the need to scan large amounts of data, in some cases the whole table. The SQL Server columnstore engine provides technology optimized to perform such operations. By building a columnstore index, SQL Server converts row-based data into a compressed columnar format. Once data is stored in columnstore format, scanning the whole table becomes very efficient in terms of disk reads. A scan operation issues I/O requests for only relevant table columns as opposed to reading the whole row of data. Moreover, all persistent data is compressed. Once data is read from disk, it is “cached” in memory within a columnstore object pool (think buffer pool for columnstore data). SQL Server decompresses the cached data as needed using **Intel® AVX2 instruction set-optimized batch mode scan operator**. These optimizations enable SQL Server to efficiently access large amounts of data and keep them in memory for reuse.

TEST QUERY ID	NOTES
TQ1	Pricing summary report
TQ2	Revenue change forecast
TQ3	Important supplier stock identification
TQ4	Promotion effect
TQ5	Top supplier
TQ6	Product type profit measure
TQ7	Returned item report
TQ8	Minimum cost supplier
TQ9	Shipping priority
TQ10	Local supplier volume
TQ11	Shipping modes and order priority
TQ12	Small quantity order revenue

Table 3. Query Descriptions

Figure 4 shows the performance of the first 7 test queries executed on the same configuration but with different processors. Each query was run twice, clearing all SQL Server caches between each run, to warm up the SQL Server columnstore object pool cache. We plotted the query run times from the second run in Figure 4, showing performance gains across the board when using the new Intel® Xeon® processor E7-8890 v4.

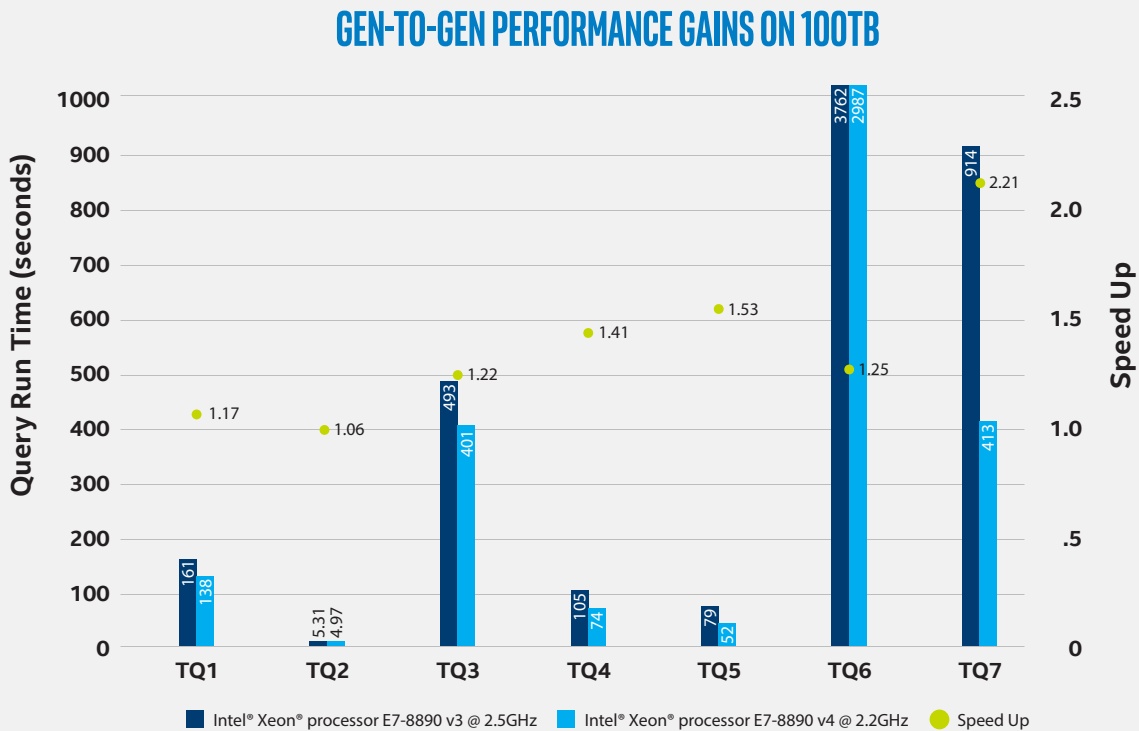


Figure 4. Query Execution Time Comparison

A comparison of the query execution times between the previous-generation Intel® Xeon® processor E7-8890 v3 and the new Intel® Xeon® processor E7-8890 v4; lower run times are better. Note that y-axis is scaled down to 1,000 seconds.

For the 7 test queries we used in our previous work (See Appendix B), we observed **up to 1.36x average speed up compared to the Intel® Xeon® processor E7-8890 v3 for complex ad-hoc queries against a 100TB dataset.**⁸

Of all queries in our test, TQ1 is especially of interest in terms of representing real-world scenarios: This query scans 97% of the fact table to calculate an aggregate value—a very common operation for most DW applications (see Appendix A for implementation details of TQ1). Figure 5 shows the TQ1 query plan statistics from the scan operator. The plan shows the number of scanned rows that are passed down to the aggregate operator and the number of rows that have been aggregated within the scan operator itself. The total of the highlighted fields gives the total number of rows processed. The SQL Server columnstore engine, optimized for Intel® architecture with Intel® AVX2 instruction set support, can run TQ1 in just **less than 2.5 minutes**, processing **4.3B rows/second** when data is in memory.

SQL Server 2016 Optimized for Intel® AVX2

Our next experiment demonstrates yet another value of the Intel® Xeon® processor family. SQL Server 2016 has different versions of the same algorithms using vector instructions, and the latest release includes Intel® Streaming SIMD Extensions (Intel® SSE) instruction set support in columnstore batch-mode operations. This means that when database software is updated to SQL Server 2016, the database software will detect all vector instruction set extensions available on the CPU and select the algorithms that provide the best performance.⁹

To show the performance benefits of using vector instructions in SQL Server’s columnstore engine, we implemented a SQL Server 2016 prototype that can override selection of algorithms via a trace flag. Using this prototype engine for the experiment, we ran the same test queries on the same platform, but with a different instruction set. For the first set of runs, we used a trace flag to turn off Intel® AVX2 instruction set support, choosing algorithms using a scalar Intel® architecture instruction set instead. In the second run, we let SQL Server 2016 choose the best version of algorithms, which in this case was the Intel® AVX2 instruction set.

Similar to the previous experiment, we ran the queries twice, clearing SQL Server caches between each query run. Using the run time from the second execution of the query, we plotted the results in Figure 6.

Figure 6 shows a **performance boost of up to 2.3x by utilizing Intel® AVX2 instruction set.**

Columnstore Index Scan (Clustered)	
Scan a columnstore index, entirely or only a range.	
Physical Operation	Columnstore Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows	168099251
Actual Number of Batches	186948
Estimated I/O Cost	94985.6
Estimated Operator Cost	97030.2 (86%)
Estimated CPU Cost	2044.66
Estimated Subtree Cost	97030.2
Number of Executions	144
Estimated Number of Executions	1
Estimated Number of Rows	59415700000
Estimated Row Size	48 B
Actual Rebinds	0
Actual Rewinds	0
Partitioned	True
Actual Partition Count	351
Ordered	True
Actual Number of Locally Aggregated Rows	594265518673
Node ID	5

Figure 5. TQ1 Execution Plan Properties of the Batch-Mode Columnstore Scan

INTEL® AVX2 PERFORMANCE QUERY RUN TIMES & SPEED UP

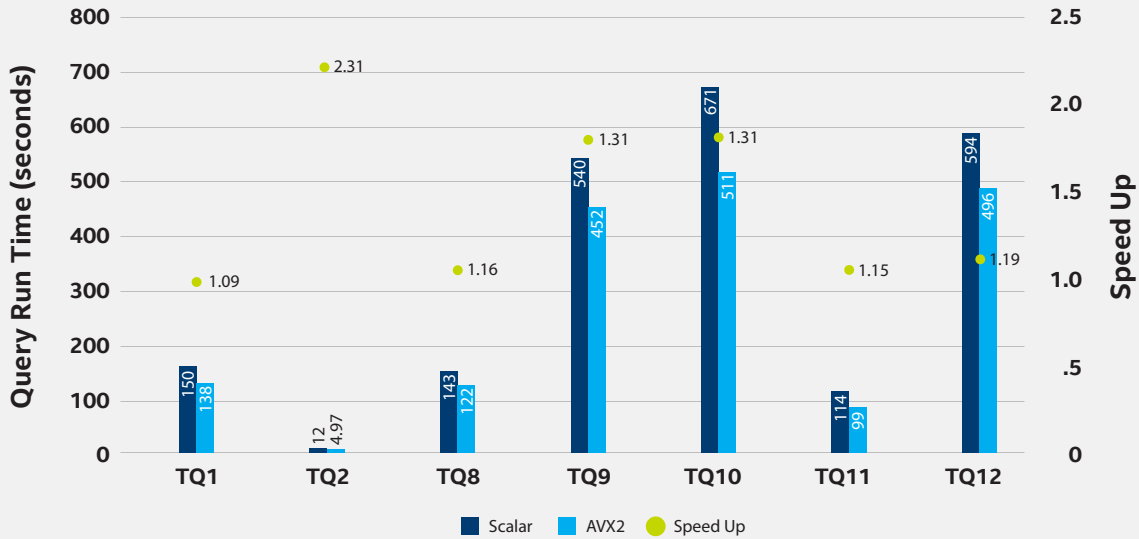


Figure 6. Query Execution Time Comparison

Test query execution times (in seconds) on 100TB using the Intel® Xeon® processor E7-8890 v4 with and without Intel® AVX2; lower run time is better.

TQ2 is the perfect value-proposition scenario in our experiments. It is a query with a relatively simple query plan, and it spends most of its query execution time in Intel® AVX2-optimized scan operator. This query calculates an aggregate value over a dataset from a given year with some additional predicates (i.e., processing roughly 12 billion rows). Hence, performance differences in query run times can be directly attributed to improvements from vectorizing the algorithms. As query plans get more complicated, measurable benefits from utilizing instruction-level parallelism will also reduce in accordance with Amdahl's law, but are still impactful, showing a **reduction of 25% in query run times on average with Intel® AVX2**.¹⁰

We encourage readers of this paper to explore the possibilities with Intel® SSE for different use cases in the online [Intel® 64 and IA-32 Architectures Software Developer Manuals](#) and easy-to-use [Intel® Intrinsic Guide](#) tool.

Summary

In this paper, we presented our findings on running a large (100TB) data warehousing application using Microsoft® SQL Server 2016 on the Intel® Xeon® processor E7-8800 v4

product family and enterprise-class Intel® SSDs and RAID controllers. We demonstrated that it is possible to build and run a 100TB data warehouse on a symmetric multiprocessor (SMP) configuration using technology that is readily available today from Intel and Microsoft. Our system with four Intel® Xeon® processor E7-8800 v4 product family processors fuels the advanced business analytics capabilities of Microsoft® SQL Server 2016 to deliver stunning performance, processing 100TB worth of data to bring key business insights in a matter of minutes. SQL Server 2016, optimized for Intel® architecture, makes use of large system memory that can fit terabytes of data, eliminating costly I/O operations. Enterprise-level Intel® SSDs ensure fast data access, delivering performance as needed.

Through continuous innovation, Intel and Microsoft are committed to bringing exciting technology to customers and more power to their business analytics solutions. Stay tuned.

Acknowledgements

We would like to acknowledge:

Lindsey Allen for spearheading the project

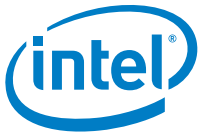
Jamie Reding for his TPC-H benchmark expertise

Sudin Bhat and **Lin Jiang** for providing insight on the columnstore engine

Christian Hescott for his guidance and improvements around performance experiments and results

Vic Stelter for his help with system setup and hardware expertise

Raechel Frick for her efforts in enabling the publishing of this paper within a short period of time



This paper has been updated to reflect the changes to our test platform and our most recent results using the latest-generation Intel® Xeon® processor E7-8890 v4. In the earlier version of the paper, we used the Intel® Xeon® processor E7-8890 v3 to build and run performance tests. We have retained sections of the earlier paper to guide users on how we set up our experiments and updated performance results obtained using the latest processors. For previous experiment results, please refer to Appendix B.

Disclaimer: The workload used in this paper is derived from portions of the TPC-H Benchmark* and, as such, is not comparable to published TPC-H Benchmark results.

1. <http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family.html>
2. Intel® technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at <http://www.intel.com>. For more details, please refer to the Legal Notice section in this paper.
3. <https://www-ssl.intel.com/content/www/us/en/solid-state-drives/data-center-family.html>
4. <http://www.intel.com/content/www/us/en/servers/raid/raid-controller-rs3sc008.html>
5. Microsoft claims based on SQL Server customer case studies. See <https://customers.microsoft.com> for more details.
6. Images used for illustration purposes only. They may not reflect actual components.
7. Since both the Intel® Xeon® processor E7-8890 v4 and the final Windows Server 2016 were not publicly available at the time of this test, we used the previous-generation Intel® Xeon® processor E7-8890 v3 and an earlier build of Windows Server 2016 for data generation.
8. 36% more performance average on 7 ad-hoc data warehousing queries, comparing the Intel® Xeon® processor E7-8890 v4 to the Intel® Xeon® Processor E7-8890 v3 on a 4-socket server with 6TB DDR4 memory, 100TB storage for data (64 Intel® SSD DC S3500 Series) running SQL Server 2016 RTM on Windows Server 2016.
9. SQL Server 2016 supports Intel® SSE4.2, Intel® AVX, and Intel® AVX2 instruction set extensions.
10. Up to 2.3x performance improvement and 25% average reduction in query run times on 7 ad-hoc data warehousing queries, comparing customized SQL Server 2016 RTM with Intel® AVX2 instruction set support to same engine without Intel® AVX2 support, on a 4-socket server with Intel® Xeon® processor E7-8890 v4 processors, 6TB DDR4 memory, 100TB storage for data (64 x Intel® SSD DC S3500 Series) running on Windows Server 2016. Reduction in query run time calculated as: $(1 - (\text{AVX2 run time} / \text{scalar run time})) \times 100$. Average reduction in run time is geometric mean of reduction in query run times for the 7 queries.

Copyright© 2017 Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, the Intel Inside logo, and Xeon are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more information, visit intel.com/performance.

Appendix A

Test Queries

In this section, we present the queries used in our tests.

Test Query 1

Test Query 1 creates a report on pricing on sales up to approximately the last 2.5 months of sales data. This query is derived from TPC-H Q1.

```

SELECT  L_RETURNFLAG,
        L_LINESTATUS,
        SUM(L_QUANTITY)                AS SUM_QTY,
        SUM(L_EXTENDEDPRI)            AS SUM_BASE_PRICE,
        SUM(L_EXTENDEDPRI*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
        SUM(L_EXTENDEDPRI*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
        AVG(L_QUANTITY)                AS AVG_QTY,
        AVG(L_EXTENDEDPRI)            AS AVG_PRICE,
        AVG(L_DISCOUNT)              AS AVG_DISC,
        COUNT_BIG(*)                  AS COUNT_ORDER
FROM    LINEITEM
WHERE   L_SHIPDATE <= dateadd(dd, -73, cast('1998-12-01' as date))
GROUP BY L_RETURNFLAG,
         L_LINESTATUS
ORDER BY L_RETURNFLAG,
         L_LINESTATUS

```

Test Query 2

Test Query 2 estimates the amount of revenue increase assuming that discounts within a range were not applied within a year. This query is derived from TPC-H Q6.

```

SELECT  SUM(L_EXTENDEDPRI*L_DISCOUNT) AS REVENUE
FROM    LINEITEM
WHERE   L_SHIPDATE >= '1993-01-01' AND
        L_SHIPDATE < dateadd(yy, 1, cast('1993-01-01' as date)) AND
        L_DISCOUNT BETWEEN 0.04 - 0.01 AND 0.04 + 0.01 AND
        L_QUANTITY < 25

```

Test Query 3

Test Query 3 lists the most important set of suppliers within a nation. This query is derived from TPC-H Q11.

```

SELECT  PS_PARTKEY,
        SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM    PARTSUPP,
        SUPPLIER,
        NATION
WHERE   PS_SUPPKEY = S_SUPPKEY AND
        S_NATIONKEY = N_NATIONKEY AND
        N_NAME = 'INDIA'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) >
    ( SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0000000010
      FROM PARTSUPP,
            SUPPLIER,
            NATION
      WHERE PS_SUPPKEY = S_SUPPKEY AND
            S_NATIONKEY = N_NATIONKEY AND
            N_NAME = 'INDIA'
    )
ORDER BY VALUE DESC

```

Test Query 4

Test Query 4 monitors the percentage of promotional revenue within a given month. This query is derived from TPC-H Q14.

```

SELECT 100.00 * SUM ( CASE WHEN P_TYPE LIKE 'PROMO%%'
                        THEN L_EXTENDEDPRI*(1-L_DISCOUNT)
                        ELSE 0
                      END) / SUM(L_EXTENDEDPRI*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM,
PART
WHERE L_PARTKEY = P_PARTKEY AND
      L_SHIPDATE >= '1997-05-01' AND
      L_SHIPDATE < dateadd(mm, 1, cast ('1997-05-01' as date))
    
```

Test Query 5

Test Query 5 finds the supplier that made the biggest contribution to overall revenue within a quarter. This query is derived from TPC-H Q15.

```

CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE)
AS
SELECT L_SUPPKEY,
       SUM(L_EXTENDEDPRI*(1-L_DISCOUNT))
FROM LINEITEM
WHERE L_SHIPDATE >= '1993-07-01' AND
      L_SHIPDATE < dateadd(mm, 3, cast ('1993-07-01' as date))
GROUP BY L_SUPPKEY
GO
SELECT S_SUPPKEY,
       S_NAME,
       S_ADDRESS,
       S_PHONE,
       TOTAL_REVENUE
FROM SUPPLIER,
REVENUE0
WHERE S_SUPPKEY = SUPPLIER_NO AND
      TOTAL_REVENUE = ( SELECT MAX(TOTAL_REVENUE)
                       FROM REVENUE0
                       )
ORDER BY S_SUPPKEY
DROP VIEW REVENUE0
    
```

Test Query 6

Test Query 6 displays the amount of profit on certain parts, grouped by supplier's nation for each year. This query is derived from TPC-H Q9.

```

SELECT NATION,
       O_YEAR,
       SUM(AMOUNT) AS SUM_PROFIT
FROM ( SELECT N_NAME AS NATION,
             datepart(yy, O_ORDERDATE) AS O_YEAR,
             L_EXTENDEDPRI*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
       FROM PART,
            SUPPLIER,
            LINEITEM,
            PARTSUPP,
            ORDERS,
            NATION
       WHERE S_SUPPKEY = L_SUPPKEY AND
            PS_SUPPKEY = L_SUPPKEY AND
            PS_PARTKEY = L_PARTKEY AND
            P_PARTKEY = L_PARTKEY AND
            O_ORDERKEY = L_ORDERKEY AND
            S_NATIONKEY = N_NATIONKEY AND
            P_NAME LIKE '%smoke%'
       ) AS PROFIT
GROUP BY NATION,
         O_YEAR
ORDER BY NATION,
         O_YEAR DESC

```

Test Query 7

Test Query 7 displays customers that caused high revenue loss due to returns. This query is derived from TPC-H Q10.

```

SELECT TOP 20
       C_CUSTKEY,
       C_NAME,
       SUM(L_EXTENDEDPRI*(1-L_DISCOUNT)) AS REVENUE,
       C_ACCTBAL,
       N_NAME,
       C_ADDRESS,
       C_PHONE,
       C_COMMENT
FROM CUSTOMER,
      ORDERS,
      LINEITEM,
      NATION
WHERE C_CUSTKEY = O_CUSTKEY AND
      L_ORDERKEY = O_ORDERKEY AND
      O_ORDERDATE >= '1993-05-01' AND
      O_ORDERDATE < dateadd(mm, 3, cast('1993-05-01' as date)) AND
      L_RETURNFLAG = 'R' AND
      C_NATIONKEY = N_NATIONKEY
GROUP BY C_CUSTKEY,
         C_NAME,
         C_ACCTBAL,
         C_PHONE,
         N_NAME,
         C_ADDRESS,
         C_COMMENT
ORDER BY REVENUE DESC

```

Test Query 8

Test query 8 finds lowest-cost supplier for a part in a region. This query is derived from TPC-H Query 2.

```

SELECT TOP 100
    S_ACCTBAL,
    S_NAME,
    N_NAME,
    P_PARTKEY,
    P_MFGR,
    S_ADDRESS,
    S_PHONE,
    S_COMMENT
FROM
    PART,
    SUPPLIER,
    PARTSUPP,
    NATION,
    REGION
WHERE
    P_PARTKEY = PS_PARTKEY AND
    S_SUPPKEY = PS_SUPPKEY AND
    P_SIZE = 50 AND
    P_TYPE LIKE '%%COPPER' AND
    S_NATIONKEY = N_NATIONKEY AND
    N_REGIONKEY = R_REGIONKEY AND
    R_NAME = 'AFRICA' AND
    PS_SUPPLYCOST = (
        SELECT MIN(PS_SUPPLYCOST)
        FROM
            PARTSUPP,
            SUPPLIER,
            NATION,
            REGION
        WHERE
            P_PARTKEY = PS_PARTKEY AND
            S_SUPPKEY = PS_SUPPKEY AND
            S_NATIONKEY = N_NATIONKEY AND
            N_REGIONKEY = R_REGIONKEY AND
            R_NAME = 'AFRICA'
    )
ORDER BY
    S_ACCTBAL DESC,
    N_NAME,
    S_NAME,
    P_PARTKEY
    
```

Test Query 9

Test query 9 displays top 10 unshipped orders with the highest value. This query is derived from TPC-H Query 3.

```

SELECT TOP 10
    L_ORDERKEY,
    SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
    O_ORDERDATE,
    O_SHIPPRIORITY
FROM
    CUSTOMER,
    ORDERS,
    LINEITEM
WHERE
    C_MKTSEGMENT = 'BUILDING' AND
    C_CUSTKEY = O_CUSTKEY AND
    L_ORDERKEY = O_ORDERKEY AND
    O_ORDERDATE < '1995-03-05' AND
    L_SHIPDATE > '1995-03-05'
GROUP BY
    L_ORDERKEY,
    O_ORDERDATE,
    O_SHIPPRIORITY
ORDER BY
    REVENUE DESC,
    O_ORDERDATE
    
```

Test Query 10

Test query 10 lists revenue volume from local suppliers. This query is derived from TPC-H Query 5.

```

SELECT      N_NAME,
            SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM        CUSTOMER,
            ORDERS,
            LINEITEM,
            SUPPLIER,
            NATION,
            REGION
WHERE       C_CUSTKEY      = O_CUSTKEY AND
            L_ORDERKEY     = O_ORDERKEY AND
            L_SUPPKEY      = S_SUPPKEY AND
            C_NATIONKEY    = S_NATIONKEY AND
            S_NATIONKEY    = N_NATIONKEY AND
            N_REGIONKEY    = R_REGIONKEY AND
            R_NAME         = 'MIDDLE EAST' AND
            O_ORDERDATE    >= '1993-01-01' AND
            O_ORDERDATE    < DATEADD(YY, 1, cast ('1993-01-01' as date))
GROUP BY   N_NAME
ORDER BY   REVENUE DESC

```

Test Query 11

Test query 11 checks if shipping mode impacts delivery dates of high-priority orders. This query is derived from TPC-H Query 12.

```

SELECT      L_SHIPMODE,
            SUM( CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR
                          O_ORDERPRIORITY = '2-HIGH'
                          THEN 1
                          ELSE 0
            END) AS HIGH_LINE_COUNT,
            SUM( CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND
                          O_ORDERPRIORITY <> '2-HIGH'
                          THEN 1
                          ELSE 0
            END) AS LOW_LINE_COUNT
FROM        ORDERS,
            LINEITEM
WHERE       O_ORDERKEY     = L_ORDERKEY AND
            L_SHIPMODE     IN ('FOB','MAIL') AND
            L_COMMITDATE   < L_RECEIPTDATE AND
            L_SHIPDATE     < L_COMMITDATE AND
            L_RECEIPTDATE  >= '1997-01-01' AND
            L_RECEIPTDATE  < dateadd(yy, 1, cast ('1997-01-01' as date))
GROUP BY   L_SHIPMODE
ORDER BY   L_SHIPMODE

```

Test Query 12

Test query 12 calculates average yearly revenue from small-shipment orders. This query is derived from TPC-H Query 17.

```

SELECT      SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
FROM        LINEITEM,
            PART
WHERE       P_PARTKEY      = L_PARTKEY AND
            P_BRAND        = 'Brand#44' AND
            P_CONTAINER    = 'JUMBO CASE' AND
            L_QUANTITY     < ( SELECT 0.2 * AVG(L_QUANTITY)
                              FROM LINEITEM
                              WHERE L_PARTKEY = P_PARTKEY

```

Appendix B

Building the Database

It took about 6 hours to load 10TB of data onto our system. This is a very impressive result, showing that SQL Server 2016 is capable of loading 100TB worth of data into a columnstore database in 60 hours at a rate of **1.6 TB/hour**.

Table 4 provides a summary of the amount of data processed for each step and how SQL Server 2016 performed for the whole build.

OPERATION	ELAPSED TIME (HH:MM)	APPROX. DATA SIZE (TB)	DATA PROCESSING RATE (TB/HR)
Bulk Insert into CCI	62:30	100	1.6
Build NCI	1:05	2	2
Create Statistics	4:40	45	9.5
Backup	2:00	45	22.5
Restore	3:30	45	13.3

Table 4. ETL and Database Management Statistics with Intel® Xeon® Processors E7-8890 v3

Performance Tests with the Intel® Xeon® Processor E7-8890 v3

In this section, we describe three performance tests we conducted with the test queries TQ1–7 with the Intel® Xeon® processor E7-8890 v3:

1- In-memory performance of SQL Server columnstore engine – We use this test to demonstrate the benefits of performance gains from utilizing data caching in memory, reducing I/O requirements. Note that some queries (even with 100TB data scale) can fit all data in memory. In cases where data is too big to fit in memory, we show that SQL Server can still run queries, utilizing I/O subsystems efficiently. For this test, we ran each query twice in isolation. Before the query execution, we ensured that there was no data caching (i.e., we issued DBCC DROPCLEANBUFFERS command). By doing so, we ensured that for the first execution, all data was read from disk. For the second run, data was cached in SQL Server columnstore object pool. Comparing the two runs, we can illustrate the benefits of in-memory execution.

2- Performance scaling with respect to data size – We use this test to compare how query execution time scales with increased amounts of data. To be able to test this, we built a reference database with the same schema, sized at 3TB. We then compared a second run from queries that run in memory. Note that on the same system, all queries running on a 3TB database will be in memory in a second run, whereas only 5 queries run in memory on the 100TB database.

3- Performance of queries when they are run concurrently – We use this test to loosely simulate a use case where multiple clients interact with the database simultaneously. In this test, we started all queries at the same time and measured the time to complete all requests. We then compared how long it took to execute queries sequentially.

In-Memory Performance of Columnstore Engine

Table 5 shows execution time for the test queries on 100TB, demonstrating the benefits of in-memory columnstore technology.

Apart from 2 queries (TQ6 and TQ7), all queries can fit in memory thanks to data compression. Caching data in memory provides **up to a 30x performance improvement and 2.5x on average** on subsequent operations on the same data. Note that in cases where the I/O subsystem is not as performant as our set up, benefits will further increase with larger system memory.

TEST QUERY ID	1ST RUN EXECUTION TIME (SEC)	2ND RUN EXECUTION TIME (SEC)	SPEED UP
TQ1	604	161	3.75
TQ2	160	5	30.05
TQ3	799	493	1.62
TQ4	175	105	1.66
TQ5	92	79	1.16
TQ6	6459	3762	1.72
TQ7	1029	914	1.13
Average Speed Up			2.54x

Table 5. Query Execution Time

Performance Scaling with Data Size

Our next experiment investigates the impact of data size on performance, showing that SQL Server performance scales with the increasing amount of data. To illustrate scaling efficiency, we compare performance of queries running on databases of different sizes, namely 3TB and 100TB. We ensure that data processed by each query scales linearly by database size (i.e., 33x) and that all queries run in memory (to make a fair comparison). We then calculate data-scaling efficiency as:

$$\text{Data-Scaling Efficiency} = \frac{\text{Execution Time on 3TB} \times 33}{\text{Execution Time on 100TB}} \times 100$$

In Table 6, we show the data-scaling efficiency of SQL Server 2016 on our platform. Average scaling efficiency of SQL Server 2016 running our test queries is 122%, which implies that in-memory execution is optimized for large data sets. We observed the lowest scaling in TQ3. Further investigation revealed that TQ3 has a long tail of single-threaded execution waiting for the client buffer to consume the very large result set returned by the query.

TEST QUERY ID	EXECUTION TIME ON 3TB (SEC)	EXECUTION TIME ON 100TB (SEC)	DATA-SCALING EFFICIENCY
TQ1	4.6	161.2	94%
TQ2	0.5	5.3	311%
TQ3	9.5	493.9	63%
TQ4	2.7	105.0	85%
TQ5	4.3	79.2	179%
Average Data Scaling Efficiency			122%

Table 6. Data-Scaling Efficiency

Performance of Running Queries Concurrently

Our final experiment is demonstrating a scenario where multiple users are running queries on the same system. In this test, we started all ad-hoc queries at the same time and executed them until completion, loosely simulating multiple client connections querying the data warehouse simultaneously.

Figure 7 shows a performance comparison of running all queries concurrently against running them sequentially. Running all ad-hoc queries concurrently takes 2:03 hours (i.e., longest-running query execution time), compared to 2:35 hours back to back.

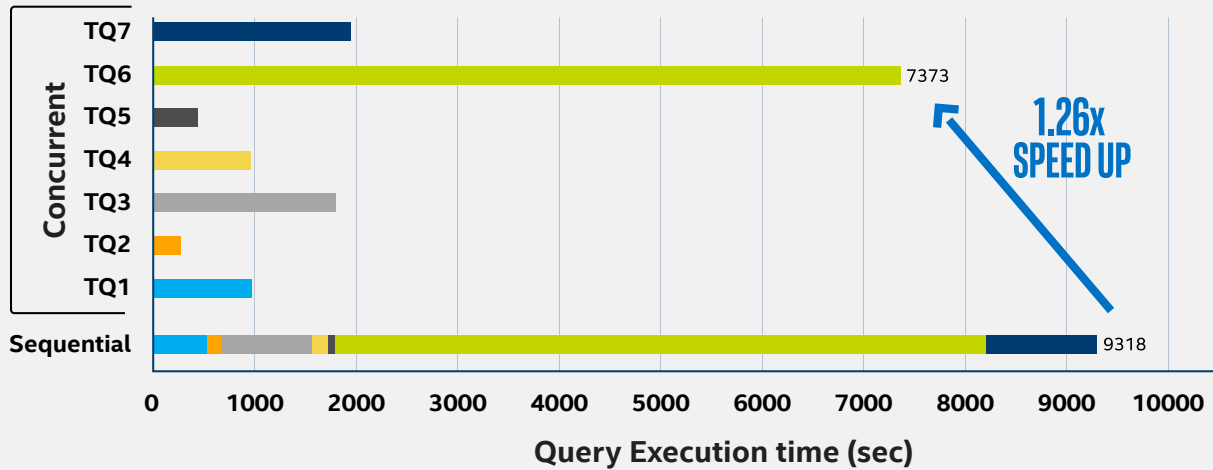


Figure 7. Concurrent vs. Sequential Query Execution Time

With 7 queries running simultaneously, we were putting our I/O subsystem to a test. We captured the workload characteristics using Perfmon. The workload provided 3:2 R/W request ratio, with 130KB reads and 64KB writes, on average. Our I/O subsystem provided **56K IOPS and 6ms latency on average—and 180K at peak with a max latency of 23ms**. Table 7 summarizes Perfmon characteristics of this workload.

PERFORMANCE CHARACTERISTICS OF CONCURRENT RUNS	
Avg. CPU Utilization	84%
Avg. Read Size	130KB
Avg. Write Size	64KB
Read/Write Ratio	3:2
Avg. Read Bandwidth	4.4GB/s
Avg. Write Bandwidth	1.3GB/s
Avg. I/O Bandwidth	5.6GB/s
Peak I/O Bandwidth	15.3GB/s
Avg. IOPS (Read + Write)	55,740
Peak IOPS (Read + Write)	179,484
Avg. I/O Latency	6ms
Peak I/O Latency	23ms

Table 7. Performance Characteristics of Concurrent Query Execution