Best Practices for Decision Support Systems with Microsoft SQL Server 2012 using Dell EqualLogic PS Series Storage Arrays

A Dell EqualLogic Reference Architecture

Dell Storage Engineering
July 2013

Revisions

Date	Description
July 2013	Initial release

Table of contents

Revisions 2
Acknowledgements 5
Feedback 5
Executive summary 5
1 Introduction 6
1.1 Objective 6
1.1.1 Audience 6
1.2 Terminology 7
2 Dell EqualLogic PS6110 series product overview 9
3 Nature of data warehouse workloads and storage 10
4 Test configuration 11
4.1 Physical system configuration- SQL Servers 11
4.2 High-level system design 12
5 Baseline I/O profiling using IOMeter 13
6 DSS performance studies using TPC-H Benchmark Factory 15
6.1 SQL Server startup parameters test studies 15
6.2 Table partitioning studies 18
6.2.1 Identifying candidates for table partitioning 18
6.2.2 Number of partitions and volumes 19
6.3 Columnstore index studies 25
6.3.1 Understanding columnstore index 25
6.3.2 Columnstore index benefits 27
6.3.3 Limitations of using columnstore index 27
6.3.4 Columnstore index performance comparisons - TPC-H 28
6.4 SAN scaling 36
6.4.1 Performance comparaison - IOMeter versus TPC-H 36
6.4.2 Scaling only the arrays with constant user load – TPC-H 37
6.4.3 Scaling the arrays and user load- TPC-H 40
7 Best practices recommendations 44
7.1 Storage 44
7.2 Network Infrastructure 44

7.3 SQL Server Best Practices 45
7.3.1 SQL Server startup parameters 45
7.3.2 Database volume creation 45
7.3.3 Files and file groups 45
7.3.4 Data file growth 45
7.3.5 Transaction log file growth 46
7.3.6 Tempdb file growth 46
7.3.7 Columnstore Index 48
8 Conclusion 50
A Configuration details 51
B Columnstore index 53
Additional resources 54

Acknowledgements

This best practice white paper was produced by the following members of the Dell Storage team:

Engineering: Lakshmi Devi Subramanian

Technical Marketing: Magi Kapoor

Editing: Camille Daily

Additional contributors: Ananda Sankaran, Darren Miller, Mike Kosacek, and Rob Young

Feedback

We encourage readers of this publication to provide feedback on the quality and usefulness of this information by sending an email to SISfeedback@Dell.com.

SISfeedback@Dell.com

Executive summary

Data warehouse (DW) or decision support system (DSS) applications present organizations with unique opportunities for leveraging their data to support, grow and expand their business, and facilitate strategic planning. Companies are capturing, storing, and analyzing large amounts of data every day. As data is continuing to grow with increasing complexity, it is becoming more and more challenging for organizations to balance cost, capacity, and performance of these warehousing systems. Therefore, it is essential to configure a balanced end-to-end system to enable consistent performance without any bottlenecks during DW loading, query and maintenance processing. Simply adding more hardware resources at growing data is not only costly, but also highly inefficient. Properly designing and sizing the DW infrastructure for performance and capacity, and regularly monitoring resource utilization, can prevent bottlenecks and deliver a cost-effective solution.

This paper describes sizing and best practices for deploying a data warehouse on Microsoft® SQL Server® 2012 using Dell™ EqualLogic™ storage arrays and demonstrates that:

- EqualLogic PS Series arrays are capable of sustaining high levels of I/O performance for SQL Server 2012 DSS workloads.
- Columnstore Index can significantly improve the data warehouse query execution times.
- Adding EqualLogic PS Series arrays can scale capacity as well as I/O performance.

1 Introduction

DSS solutions require scalable storage platforms that offer high levels of performance and capacity. This paper presents sizing guidelines and best practices for deploying DSS solutions based on the results of SQL Server 2012 I/O performance tests conducted using Dell™ PowerEdge™ servers, Dell™ Force 10™ switches and EqualLogic storage. The EqualLogic PS Series array builds on a unique peer-storage architecture that is designed to provide the ability to distribute the load across multiple arrays and provide a SAN solution that scales as organizations grow. This pay as you grow model, allows customers to add arrays as their business demands increase the need for more storage or I/O capacity. Deploying the storage platform using validated best practices and recommended settings for various system components (storage, server, switch, operating system, and application) ensures optimal performance.

1.1 Objective

This paper identifies best practices and sizing guidelines for deploying SQL Server DSS applications with EqualLogic storage arrays. It also illustrates that the virtualized EqualLogic PS Series storage arrays scale-out proportionally providing a flexible and modular infrastructure for today's increasingly complex datacenters.

The following two major sections were analyzed during the test studies for this paper.

- **Baseline tests:** I/O profiling tests using IOMeter were executed to establish baseline I/O performance characteristics of the test storage configuration when running DSS-like I/O patterns before deploying databases.
- **Database tests:** Performance tests were executed by simulating SQL DSS queries using TPC-H benchmark from the Benchmark Factory® for Databases tool.

The test objectives determined:

- Baseline performance that could be achieved from a PS6110X on RAID 50 using IOMeter.
- DSS application performance studies:
  - SQL Server tuning parameters.
  - Impact of table partition on query performance.
  - Impact of columnstore Index on query performance.
  - Scalability of the storage arrays I/O performance with a DSS application simulation as storage arrays were added and while ensuring that the overall configuration was balanced with no resource bottlenecks on the server.

1.1.1 Audience

This white paper is primarily targeted to solution architects, database administrators, database managers and storage administrators, who are interested in using Dell EqualLogic storage to design, properly size and deploy Microsoft SQL Server 2012 running on the Windows Server® 2012 platform. It is assumed that the reader has an operational knowledge of the Microsoft SQL Server configuration and management of EqualLogic SANs and iSCSI SAN network design.

1.2 Terminology

The following terms are used throughout this document.

DSS I/O pattern: DSS workloads typically take a long time to complete and usually require processing large amounts of data that tend to utilize a major portion of the database server memory and processor time. The data I/O pattern is predominantly sequential and usually consists of large blocks, typically ranging from 64 KB to 512 KB in size. The key metric in measuring performance of DSS workloads is throughput (MB/sec).

Group: One or more EqualLogic PS Series arrays connected to an IP network that work together to provide SAN resources to the host servers.

Hypervisor: A hardware virtualization technique that enables running multiple guest operating systems on a single host system at the same time. The guest operating system shares the hardware of the host computer, such that each OS appears to have its own processor, memory and other hardware resources.

Partition (SQL Server): The table and index data gets divided into units that can be spread across more than one filegroup. Partitioning makes large tables or indexes more manageable, as it enables managing and accessing subsets of data quickly and efficiently, while maintaining the integrity of a data collection. By using partitioning, an operation such as loading data from an OLTP to an OLAP system is much faster. Maintenance operations performed on subsets of data are also more efficient because these operations target only the data that is required, instead of the whole table.

Perfmon: Perfmon.exe is a process associated with the Microsoft® Windows® Operating System. Performance Monitor, or Perfmon, gathers performance statistics on a regular interval, and saves those stats in a file. The database administrator picks the sample time, file format, and counter statistics to monitor.

Pool: Allocates storage space into partitions comprising one or more members. When a group is created, there is one storage pool, called default and this pool cannot be deleted, but can be renamed. Members and volumes are assigned to the default pool unless a different pool is specified. Space can be allocated to users and applications by creating volumes, which are seen on the network as iSCSI targets. If the group has multiple members, the group space can be divided into different pools and then assign members.

Primary data File (SQL Server): Contains the startup information for the database and points to the other files in the database. User data and objects can be stored in this file or in secondary data files. Every database has one primary data file. The recommended file name extension for primary data files is .mdf.

Primary Filegroup (SQL Server): The primary file and all system tables are allocated to the primary filegroup.

Range left partition function (SQL Server): The boundary value that specifies the upper bound of its partition. All values in partition 1 must to be less than or equal to the upper boundary of partition 1 and all values in partition 2 must be greater than partition 1's upper boundary.

Range right partition function (SQL Server): Here, each boundary value specifies the lowest value of its partition. All values in partition 1 must be less than the lower boundary of partition 2 and all values in partition 2 must be greater than or equal to partition 2's lower boundary.

SAN Headquarters (SAN HQ): Monitors one or more PS Series groups. The tool is a client/server application that runs on a Microsoft Windows system and uses simple network management protocol (SNMP) to query the groups. Much like a flight data recorder on an aircraft, SAN HQ collects data over time and stores it on the server for later retrieval and analysis. Client systems connect to the server to format and display the data in the SAN HQ GUI.

Secondary data file (SQL Server): Optional, user-defined files that store user data. Secondary files can be used to spread data across multiple disks by putting each file on a different disk drive. Additionally, if a database exceeds the maximum size for a single Windows file, the secondary data files can be used so the database can continue to grow. The recommended file name extension for secondary data files is .ndf.

TPC-H Scale: In TPC-H, the database size is defined with reference to scale factor (i.e., scale factor = 300 is approximately 300 GB).This size does not include the database index size and the database size will be bigger than this estimated size after it has been built.

Transaction log file (SQL Server): Holds the log information used to recover the database. There must be at least one log file for each database. The recommended file name extension for transaction logs is .ldf.

User-defined Filegroup (SQL Server): Specifically created by the user when the database is created or later modified. It can be created to group data files together for administrative, data allocation, and placement purposes.

Virtual Machine: A guest operating system implemented on a software representation of hardware resources (processor, memory, storage, network, etc.) running on top of a hypervisor in a virtualized server environment.

2 Dell EqualLogic PS6110 series product overview

The Dell EqualLogic PS6110 array is intelligent storage that is designed to provide simple management and seamless expansion. EqualLogic delivers comprehensive end-to-end solutions to store and manage data so that organizations can efficiently move the right data, to the right place, at the right time and for the right cost. For virtualization environments in medium to large enterprises, the PS6110 Series is designed to meet the current needs and be ready to grow as the demands of the virtual era increase.

The EqualLogic PS6110 Series, the second generation 10 GbE PS Series iSCSI array, features greater flexibility, throughput, capacity, density and performance than prior PS Series 10 GbE arrays. The PS6110 provides a sequential performance of up to 133% higher than the PS6100 Series for bandwidth-intensive applications such as streaming video and data warehousing. Performance may vary depending on the workload and drive type. This performance variance was observed in Dell tests (run in January 2012) that compared the PS6110 to the PS6100 arrays at RAID 50 with a 100% sequential read workload and 256KB I/O size. The PS6110 is a 10 GbE iSCSI SAN array that allows organizations to leverage their existing 10 GbE infrastructure with SPF+ or lower-cost 10GBASE-T.

The EqualLogic PS6110X storage arrays include a 2.5" SAS HDDs in 2U form factor. It is optimized for critical data center applications with up to 21.6 TB of capacity. Visit dell.com for feature and benefit details.

3 Nature of data warehouse workloads and storage

Different types of database applications have varying needs. Understanding the models for the most common database application workloads can be useful in predicting possible application behavior. The most common database application workload models are online transaction processing (OLTP) and data warehouse (DW). This paper focuses on DW workloads.

For information on OLTP models, refer "Best Practices and Sizing Guidelines for Transaction Processing Applications with Microsoft SQL Server 2012 using EqualLogic PS Series Storage" at http://en.community.dell.com/dell-groups/dtcmedia/m/mediagallery/20321740/download.aspx

DW applications are typically designed to support complex analytical query activities using very large data sets. The queries executed on a DSS database typically take a long time to complete and usually require processing large amounts of data. A DSS query may fetch millions of records from the database for processing. To support these queries the server reads large amounts of data from the storage devices. A DW profile contains the following pattern:

- Reads and writes tend to be sequential in nature and are generally the result of table or index scans and bulk insert operations.
- The read I/O to storage consists of large I/O blocks, ranging from approximately 64 KB to 512 KB in size.

The large I/O requests require high I/O throughput rates from storage to the database server to provide optimal performance. In addition to the significant I/O throughput required, the DW queries also require substantial processing resources (CPU and RAM). Therefore, the database server must be provided with sufficient processing and memory resources to handle the raw query results and return useable data.

The large I/O patterns and processing necessary in DW queries warrant careful system design to ensure that the performance requirements are met at each component in the system. These components include database and operating system settings, server resources, SAN design and switch settings, storage Multipath I/O (MPIO) software, storage resources, and storage design.

4 Test configuration

The SQL Server test system used to conduct testing for this paper is shown in Figure 1 and Figure 2.

4.1 Physical system configuration- SQL Servers

The physical connectivity of the two servers that hosted the SQL Server databases used for testing is shown in Figure 1. Other infrastructure components used in the test setup are shown in the high-level diagram in Figure 2.

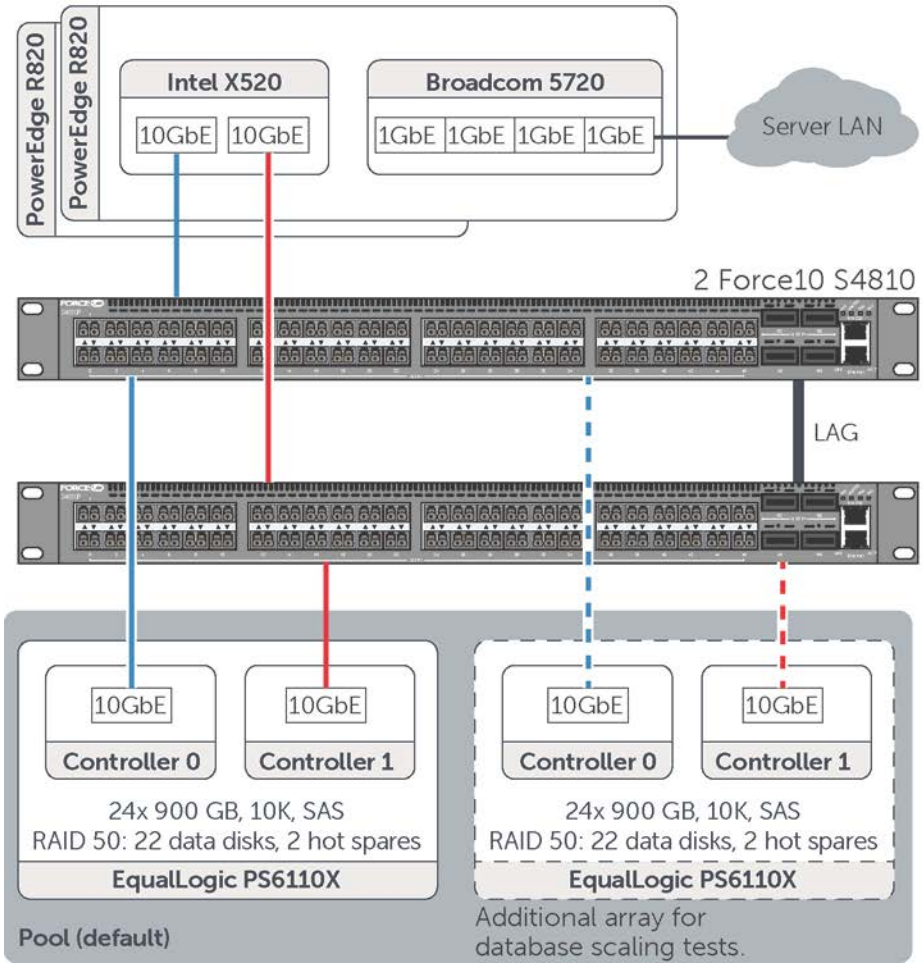

Figure 1 SQL Server LAN and iSCSI SAN connectivity

4.2 High-level system design

A high-level overview of the test configuration is shown in Figure 2.

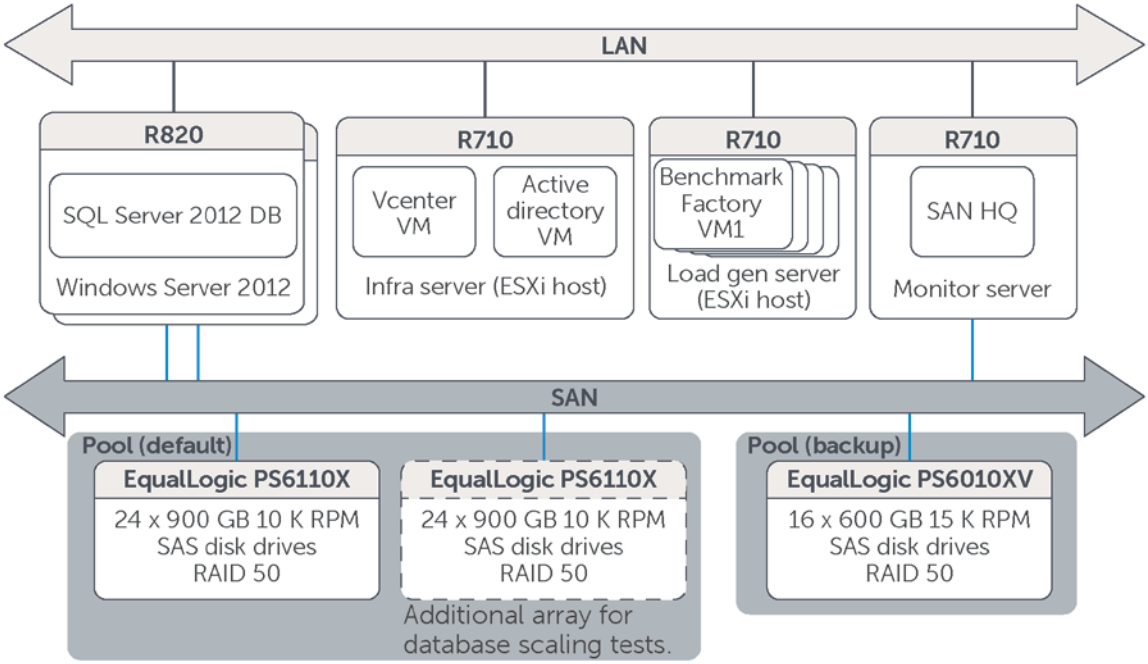

Figure 2 High-level overview of the test configuration

Key design details of the test system configuration shown in Figure 2 include:

- Two R820 Dell PowerEdge servers were used to host two SQL Server instances. Each of these servers had SQL Server 2012 Enterprise Edition installed on Windows Server 2012 Enterprise Edition.
- Both of the servers had 128 GB RAM each, and the SQL server instances were configured to use 117.7 GB of the memory by using the maximum server memory setting in SQL Server Management Studio.
- The infra server illustrated in Figure 2 had VMware ESXi 5 installed on it and hosted virtual machines for vCenter and active directory.
- The load gen server in Figure 2 had VMware ESXi 5 installed on it and hosted four Windows 2008 R2 workload simulation virtual machines with each running an instance of Benchmark Factory.
- The monitor server was a PowerEdge R710 running Windows 2008 R2 Enterprise edition. It hosted SAN HQ.
- The SAN switches consisted of two Dell Force10 S4810 switches configured with a LAG (2 x 40 GbE) interconnect. Redundant connection paths were created from each array controller to each switch in the stack.
- Two EqualLogic PS6110X arrays, each with 24 x 900 GB 10K RPM SAS disk drives in a RAID 50 configuration hosted SQL Server database volumes. For single array tests (sections 5 to 6.3), the two members were placed in separate pools and for the two arrays scalability tests (sections 6.4.3 and 6.4.4) the two members were placed in the same pool.

5 Baseline I/O profiling using IOMeter

A series of I/O profiling tests were executed using IOMeter, to establish the baseline I/O performance characteristics of the test storage configuration before deploying any databases. Since DSS workloads are mostly sequential with larger I/O block sizes, the baseline throughput numbers were established by running three types of large block I/O workload. They were:

100% Sequential reads of large I/O block sizes: This test determined the read I/O throughput of the array with a sequential I/O workload. This provided a baseline for workloads such as DSS query processing and report generation operations.

100% Sequential writes of large I/O block sizes: This test determined the write I/O throughput of the array with a sequential I/O workload. This provided a baseline for workloads such as bulk, or batch updates to table data and backup.

100% Random reads of large I/O block sizes: DSS database environments exhibit an I/O pattern of random read I/O with large blocks due to multiple users submitting large running queries to the system. Even though each user's query generates table scans involving sequential I/O, a collection of queries from multiple simultaneous users make the resulting I/O more parallel and random. This test provided a baseline for a random read workload.

In these tests, throughput was measured while evaluating the EqualLogic PS6110X array running I/O patterns with large block sizes of 100% sequential (reads and writes) and 100% random (reads) on a single PS6110X array. The volume size and the number of volumes used for the IOMeter test were chosen to roughly match the array capacity utilization when the actual database (300 scale) would be deployed. The IOMeter test file occupied the entire volume (number of sectors set to 0) and the volumes were exposed as NTFS volumes with a drive letter. The configuration parameters for the tests are shown in Table 1.

Table 1 Test parameters: I/O workload studies using IOMeter

Configuration Parameters	
EqualLogic SAN	One PS6110X (2.5", 24 10 K SAS drives,900 GB)
Volume configuration	Ten volumes, 300 GB each
RAID type	RAID 50
I/O mix	**I/O block size**
100% Sequential Reads	64K,128K,256K,512K,1MB (Queue depth-128 , worker per volume-1, number of sectors-0)
100% Sequential Writes	64K,128K,256K,512K,1MB (Queue depth-128 , worker per volume-1, number of sectors-0)
100% Random Reads	64K,128K,256K,512K,1MB (Queue depth-128 , worker per volume-1, number of sectors-0)

The results collected from the tests are illustrated in Figure 3 and Figure 4. Figure 3 shows sequential I/O throughout reported by IOMeter for the tested I/O block sizes. The theoretical throughput for a 10 GbE

interface is 1280 MB/sec. From the tests conducted, the maximum average read throughput achieved was 1042 MB/sec for a 128K block size and maximum average write throughput achieved was 680 MB/sec for 256K block size.

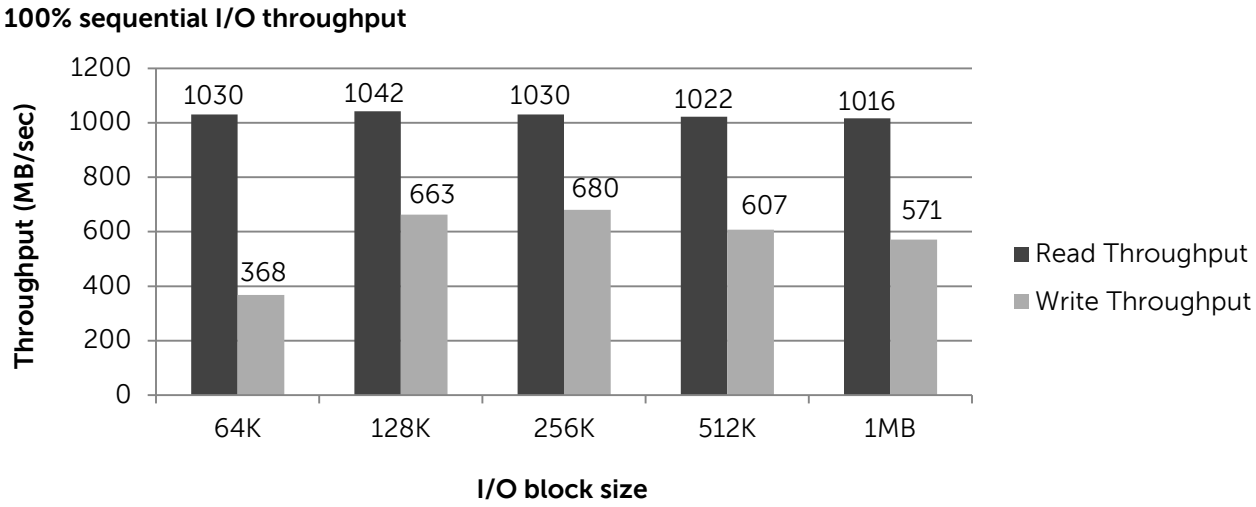

Figure 3 IOMeter sequential I/O throughput

Figure 4 shows the maximum random read throughput obtained for different I/O block sizes. The read throughput increased as the block sizes increased from 64K to 1MB.

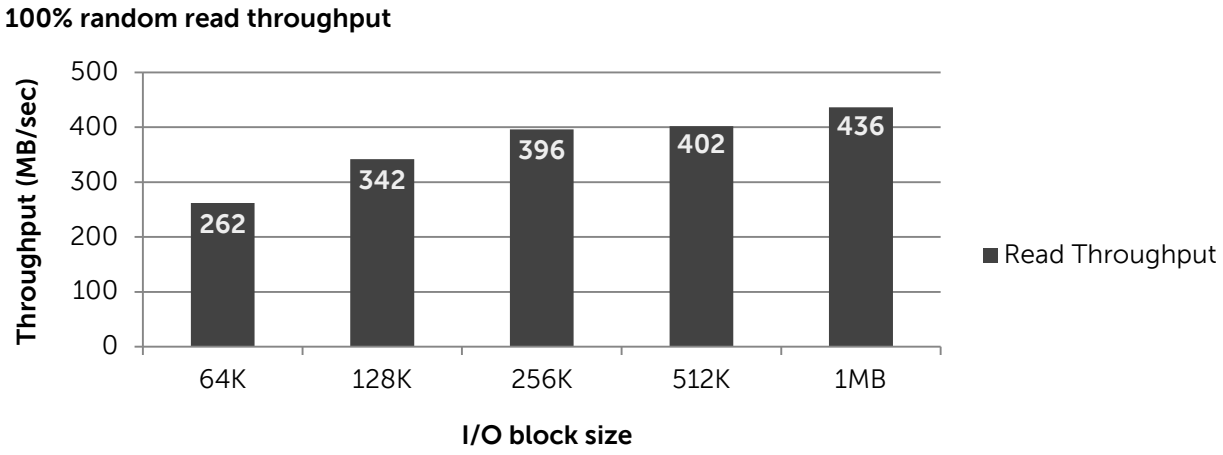

Figure 4 IOMeter random read throughput

Note: By default, the *MaxTransferLength* (maximum data size of an I/O request) in the Windows iSCSI initiator is 256K. I/Os larger than 256K are split into 256K chunks. Because the *MaxTransferLength* setting was left at the default for the tests in section 5, the block sizes 512K and 1MB were broken into 256K blocks. For details, refer to the section titled, "Microsoft Software Initiator Kernel Mode Driver" in the *iSCSI initiator Users Guide for Windows 7 and Windows Server 2008 R2* at http://www.microsoft.com/en-us/download/details.aspx?id=6408

6 DSS performance studies using TPC-H Benchmark Factory

A series of DSS application simulations were conducted using Benchmark Factory as the simulation tool running a TPC-H like workload. TPC-H is an industry standard benchmark for DSS. DSS user queries were run from Benchmark Factory against the database to understand the I/O behavior at the storage arrays when the SQL Server database executed those queries.

Before beginning the test runs, the database was populated and backed up to a backup array in the SAN (refer to Figure 2). The backed up database was restored as necessary before beginning subsequent test runs. This allowed each test run to start from exactly the same state.

6.1 SQL Server startup parameters test studies

Microsoft recommends using the below startup parameters and settings for the SQL Server Fast Track Data warehouse configuration (reference http://msdn.microsoft.com/en-us/library/hh918452.aspx). These parameters were set in the following tests to evaluate the performance benefits.

- **-E:** This parameter increases the number of contiguous extents in each file that are allocated to a database table as it grows. This option is beneficial because it improves sequential access. Refer to http://support.microsoft.com/kb/329526 for details.
- **-T1117:** It is recommended to pre-allocate data file space rather than depending on auto grow. However, if auto growth is enabled, then this trace flag ensures the even growth of all files in a file group when auto growth is enabled. For the tests performed, the data file space has been pre-allocated.
- Enable option **Lock Pages in Memory**. For more information, refer to http://support.microsoft.com/kb/918483
- **SQL Server Maximum Memory:** For SQL Server 2012, Fast Track 4.0 guidelines suggest allocating no more than 92% of total server RAM to SQL Server. If additional applications will share the server, the amount of RAM left available to the operating system should be adjusted accordingly.

For this test, the TPC-H database was loaded from Benchmark Factory by placing the two largest tables, *LineItem* and *Order*, in separate data files and then placing these files on separate volumes. All other tables were placed in a separate file belonging to a separate volume. This data layout pinpointed the table that produced the largest throughput while running TPC-H user queries. Identifying the table that constituted for the largest percentage of the total read throughput was crucial for the subsequent table partition tests. The configuration parameters used for this test are shown in Table 2.

Table 2 Database file and volume layout test parameters

Configuration Parameters	
EqualLogic SAN	One PS6110X (2.5", 24 10 K SAS drives,900 GB)
RAID type	RAID 50
SQL DB volume Configuration	
SQL DB Files- Volume size	• Primary Data File (.mdf) – 100GB • LineItem Table DataFile1 (.ndf) – 600 GB • Order Table (.ndf)-300 GB • All other tables (.ndf)–200 GB • Log File (.ldf)-100 GB • Tempdb-600GB
Data Warehouse Workload Parameters	
Database load & Workload generation	TPC-H from Benchmark Factory
Database size	300 scale (~620 GB including data and indexes)
Number for users/streams	6 (As per TPC-H standard, the minimum required streams/users to be run for a 300 scale Database is 6)
Queries	22 TPC-H queries per user
SQL Server Memory (Max memory allocation) in GB	117.7 GB (92% of (128 GB RAM)
SQL Server parameters	-E, -T1117 and lock pages in memory
CPU	4* Intel® Xeon® Processor E5-4620 @2.20 GHz,8 Cores per socket

In order to test these parameters, a baseline test was first performed using just the default setting and system performance was measured. The same test was repeated after setting the startup parameters and lock pages in memory settings to measure the performance. The results are shown in Figure 5.

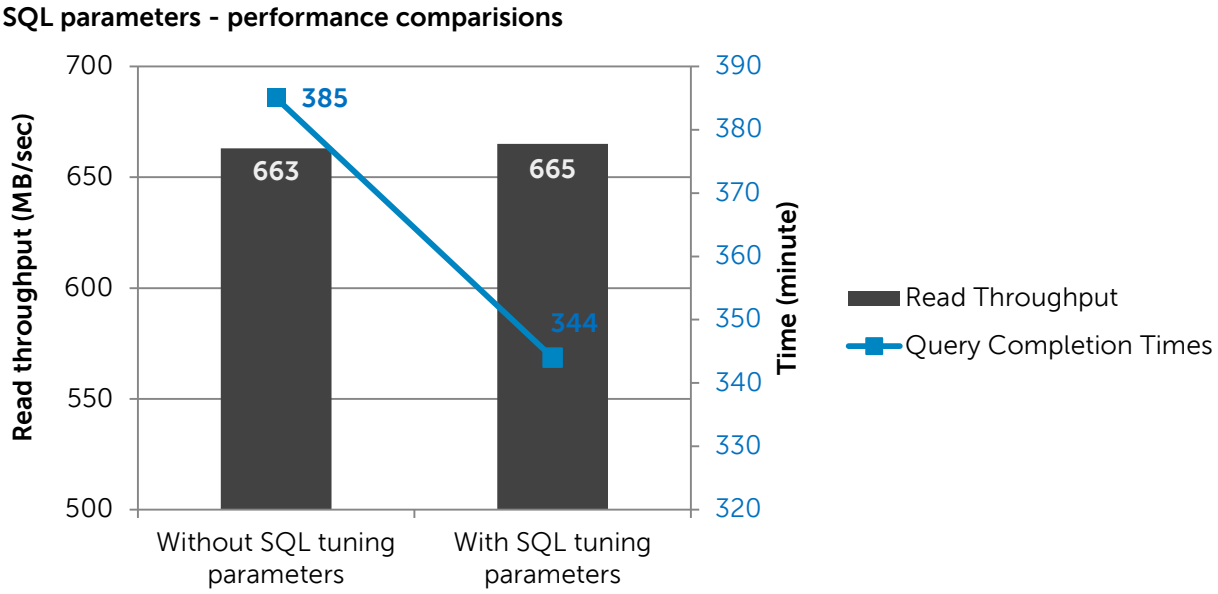

Figure 5 SQL Parameter studies

The query completion time is the duration between the first query submission and completion of the last query. From the tests conducted with the SQL server parameters set, the total execution time of the queries decreased by 41 minutes (10% decrease) compared to the execution without setting the SQL Server startup parameters. However, the throughput remained the same in both the cases. The decrease in query execution times can be attributed to the startup option setting "-E" that improved the sequential access by increasing the number of contiguous extents in each file. Scans benefit from this contiguity because the data is less fragmented allowing for fewer file switches and faster access. The "maximum SQL Server memory" setting and "lock pages memory" policy setting also contributed to the execution performance improvement. These settings prevented the system from paging memory to the disk and making the pages remain in the memory which improved data reads.

One of the key performance metrics in DW is the query execution times. Setting these SQL Server parameters improved DW user query execution times in the tests performed in this section. The remaining database tests presented in this paper incorporated these settings as well in order to take advantage of the performance benefits.

6.2 Table partitioning studies

This section evaluates the impact of SQL Server table partitioning on DSS performance in a data warehouse environment. Partitioning breaks up database objects by allowing subsets of data to reside in separate file groups. This is beneficial to data warehouse databases in several ways. Some of the table partition benefits listed by Microsoft are:

- **Data pruning:** If aged data needs to be archived with partitioned tables, the process of removing it takes less time since there is less data.
- **Improved load speed:** By loading into smaller partitioned tables the incremental load process can be significantly more efficient.
- **Maintenance:** Activities including loading data, backing up and restoring tables can run in parallel to achieve dramatic increases in performance once the data warehouse staging application has been built to support partitioning.
- **Query speed:** A table partition may or may not yield query performance improvement depending on the database schema and query characteristics. The query performance might be similar for partitioned and non-partitioned fact tables. When the partitioned database is properly designed, the relational engine in a query plan will only include the partition(s) necessary to resolve that query. The resulting query will perform well against the partitioned table, similar to a properly indexed, combined table with a clustered index on the partitioning key.

6.2.1 Identifying candidates for table partitioning

Before performing the table partition, the best candidates for partition need to be identified. The following data layout was implemented on the database to identify the candidates.

- *LineItem* and *Order* tables were the two largest tables in TPC-H database.
- The first largest table (LineItem) was placed in its own filegroup (LineItem_Filegroup) and on a separate volume.
- The second largest table (Order) was placed on its own filegroup (Order_Filegroup) and on separate volume.
- The rest of the tables were placed in a separate filegroup (Others_Filegroup) and placed on a separate volume.

TPC-H queries from benchmark factory were run against the above data layout to identify the table that constituted the highest percentage of the total throughput (reference tests in Section 6.1). The LineItem table comprised about 70% of the total read throughput achieved while running queries which is why this table was chosen as a partition candidate.

For the table partition tests in this section, the table was partitioned based on the right range partitioning scheme using the l_shipdate date-time column. This column was chosen because most of the TPC-H queries accessing the LineItem table used l_shipdate in their *where* clause.

6.2.2 Number of partitions and volumes

This test evaluated the performance gains of using more volumes by having more table partitions in each volume. The l_shipdate column in the LineItem table had years ranging from 1992 to 1998. Two table partition test studies were performed.

- Four partitions - Placing the data for every two years in a partition.
- Eight partitions- Placing the data for every year in a partition.

For the test studies with four partitions, the LineItem table was divided into four filegroups and each filegroup had its own data file placed in a separate volume. For the test studies with eight partitions, the LineItem table was divided into eight filegroups and each filegroup had its own data file placed in a separate volume. Refer to Table 3 for volume layout details and Figure 6 for a visual representation of the volume layout.

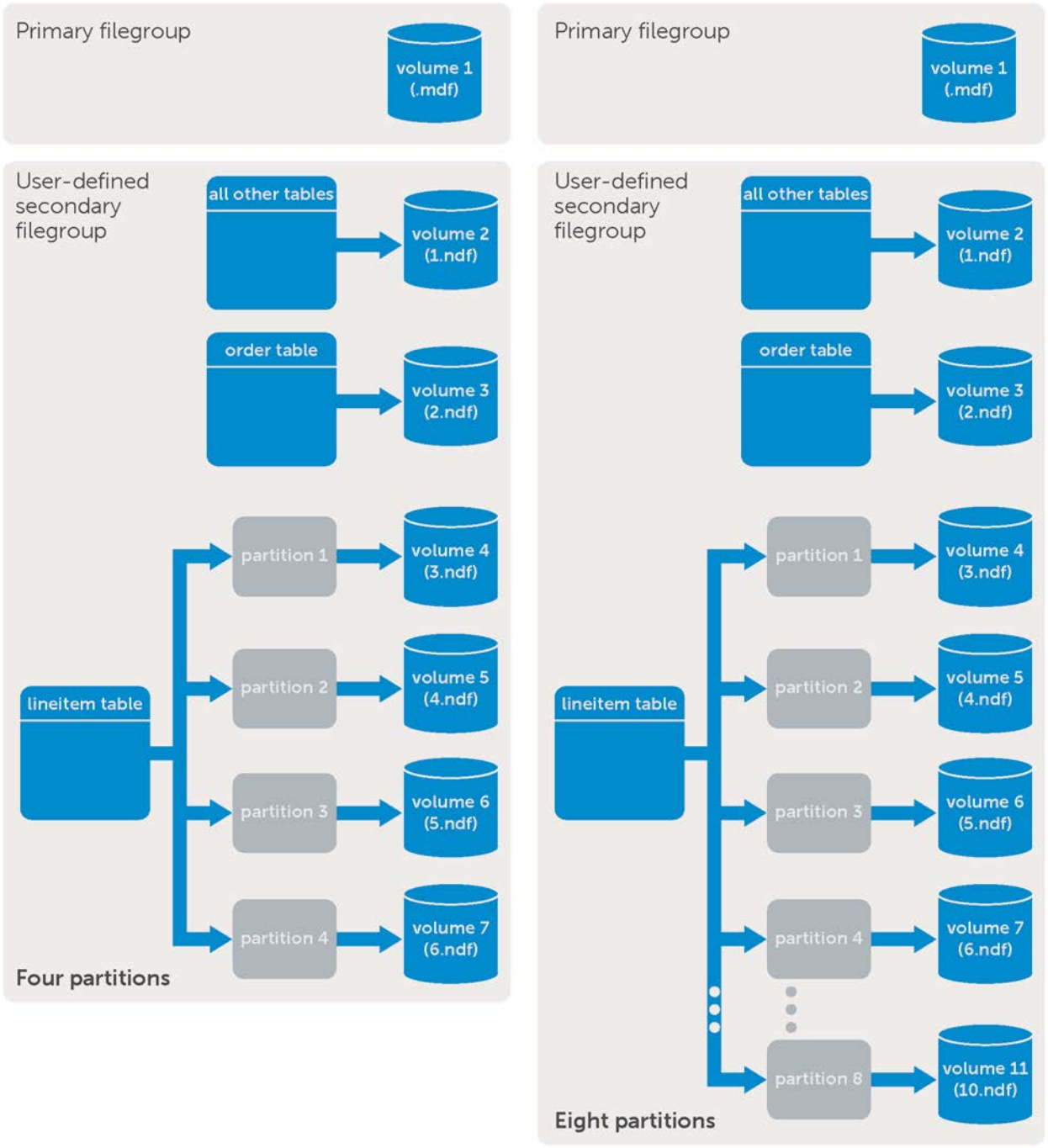

Figure 6 Four and eight table partition data file layout

Table 3 Table partition test parameters

Configuration parameters	
EqualLogic SAN	One PS6110X (2.5", 24 10 K SAS drives,900 GB)
RAID type	RAID 50
Table partition pests- volume layout	
Four partitions (LineItem table, partitioned into four partitions)	• Primary data file (.mdf) – 100GB • LineItem table partition1 & non-clustered indexes (.ndf) – 1 TB • LineItem table partition2 (.ndf)-200 GB • LineItem table partition3 (.ndf)-200 GB • LineItem table partition4 (.ndf)-200 GB • Order table & non-clustered indexes (.ndf)-300 GB • All other tables & non-clustered indexes (.ndf)–200 GB • Log file (.ldf)-100 GB • Tempdb-600 GB
Eight partitions (LineItem table, partitioned into eight partitions)	• Primary data file (.mdf) – 100GB • LineItem table partition1 & non-clustered indexes (.ndf) – 1 TB • LineItem table partition2 (.ndf)-200 GB • LineItem table partition3 (.ndf)-200 GB • LineItem table partition4 (.ndf)-200 GB • LineItem table partition5 (.ndf)-200 GB • LineItem table partition6 (.ndf)-200 GB • LineItem table partition7 (.ndf)-200 GB • LineItem table partition8 (.ndf)-200 GB • Order table & non-clustered indexes (.ndf)-300 GB • All other tables & non-clustered indexes (.ndf)–200 GB • Log file (.ldf)-100 GB • Tempdb-600 GB
Data warehouse workload parameters	
Database load & workload generation	TPC-H from Benchmark Factory
Database size	300 scale (~620 GB including data and indexes)
Number for users/streams	6 (As per TPC-H standard, the minimum required streams/users to be run for a 300 scale Database is 6)
Queries	22 TPC-H queries per user
SQL Server memory (Max memory allocation) in GB	117.7 GB
SQL Server parameters	-E, -T1117 and lock pages in memory
CPU	4* Intel® Xeon® Processor E5-4620 @2.20 GHz,8 Cores per socket

Figure 7 shows the test results that were collected while running six users (132 queries) on four partitions and then on eight partitions. The average read throughput remained almost the same since the single array was running at its maximum achievable throughput. However, the query execution time improved during the eight partition test by 11% due to the increased number of volumes. This means more reader threads get issued by SQL Server and fewer volume queue depths seen at the storage.

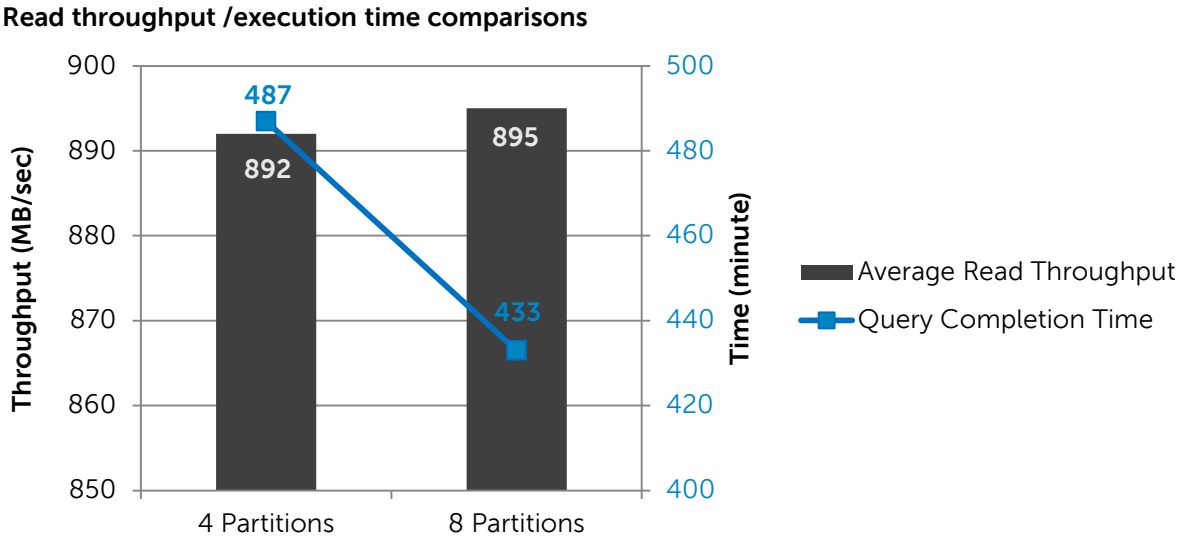

Figure 7 Four and eight table partition performance comparisons

Table partitioning speeds up the load process and provides easy maintainability, but is not beneficial when it comes to query execution speed in the case of data warehouse databases. In the tests conducted for this paper, either the four or eight table partition tests did not yield a query performance improvement over a non-partitioned table when more users ran the same queries. There was throughput improvement (892 MB/sec) with the table partition (refer to Figure 7) compared to the non-partitioned table (665 MB/sec, refer to Figure 5), but at the cost of execution time. In SQL Server there are multiple threads accessing the same table partition with more users, more threads access the same partition. This makes the threads wait and I/O access is more randomized at the logical volume/physical disk level, causing increased query execution times. In spite of table partition's reduced query performance; the eight partition layout was used for the scalability tests (section 6.4) due to the benefits offered by table partitions.

Comparing the four and eight partition case in Figure 7, the eight partitions offered better execution times. To understand the cause for improved query execution time in the eight partition case, the queries were examined individually by running a single user power test (one user with 22 queries run in the same order). The power test was run on both the four and eight partitions. Table 4 shows the average response times of these 22 queries.

Table 4 One user power test query response time (seconds) comparison

Queries	LineItem Table - Four Partitions	LineItem Table- Eight Partitions
Pricing summary report query (Q1)	**335.678**	**355.918**
Minimum cost supplier query (Q2)	11.893	1.158
Shipping priority query (Q3)	**291.178**	**292.457**
Order priority checking query (Q4)	348.959	408.225
Local supplier volume query (Q5)	363.942	313.058
Forecasting revenue change query (Q6)	**204.53**	**78.346**
Volume shipping query (Q7)	**333.83**	**239.192**
National market share query (Q8)	442.987	433.611
Product type profit measure query (Q9)	572.652	544.943
Returned item reporting query (Q10)	348.855	302.482
Important stock identification query (Q11)	192.573	187.735
Shipping modes and order priority query (Q12)	**444.854**	**384.98**
Customer distribution query (Q13)	107.131	146.968
Promotion effect query (Q14)	**997.375**	**104.46**
Top supplier query (Q15)	**202.146**	**118.025**
Parts-supplier relationship query (Q16)	39.37	35.557
Small-quantity-order revenue query (Q17)	3.93	3.665
Large volume customer query (Q18)	330.447	312.85
Discounted revenue query (Q19)	46.229	48.17
Potential part promotion query (Q20)	**246.154**	**148.975**
Suppliers who kept orders waiting query (Q21)	1081.497	957.511
Global sales opportunity query (Q22)	63.022	58.281

The bold queries (Q1, Q3, Q6, Q7, Q12, Q14, Q15 and Q20) use the partition column l_shipdate in their *where* clause. The response times for the queries in blue (in the four partition column) improved with the

eight partition test. This was because there were fewer requests queued on each LineItem volume due to data being spanned across more volumes in the eight partition case. This can be seen from Figure 8.

Figure 8 shows the average queue depth (or number of outstanding I/Os) of the LineItem volumes (which contained the partitions) from SANHQ. The number of outstanding I/Os on the eight partition test was 41% less compared to the four partition test. This proved that the request were getting served quicker and provided a lower execution time.

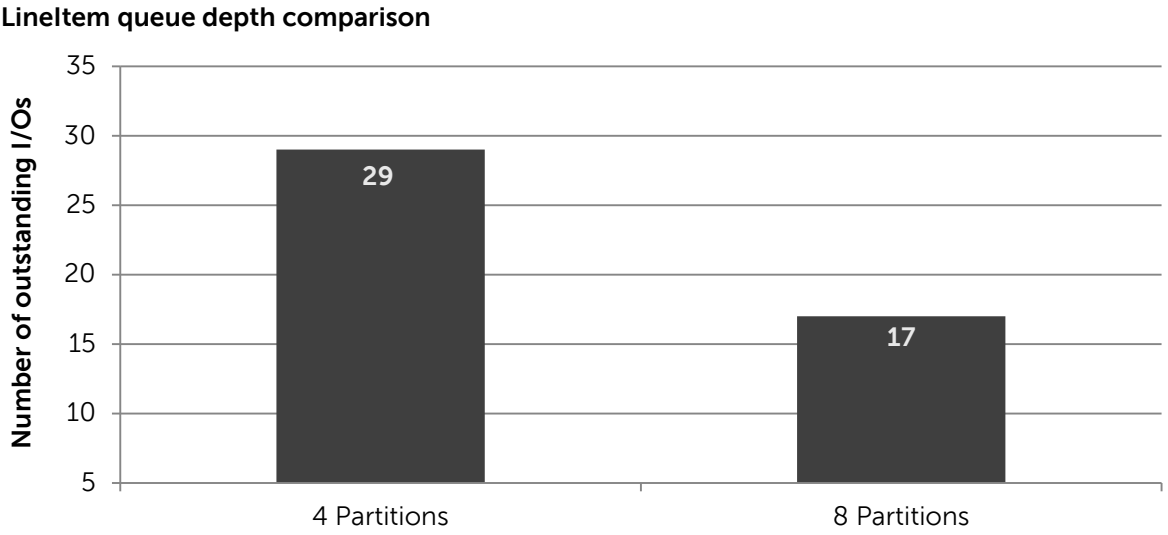

Figure 8 Average queue depths – LineItem volumes

The eight partition test showed improved SQL Server query efficiency compared to the four partition test case. However, the read throughput remained the same since the single array was running at its maximum achievable throughput. With more volumes, the number of outstanding I/Os in each volume was less and provided faster query execution times. The query execution times and I/O throughput depends on the partitioning scheme chosen, which in turn is heavily dependent on the data and workload characteristics. Typically for DSS database applications, partitioning offers these benefits when implemented on larger tables using a scheme based on the application and query behavior.

Table partitioning speeds up the load process and provides easy maintainability, but is not beneficial when it comes to query execution speed in the case of data warehouse databases. It must be noted that as per Microsoft (Refer to "Query Speed" in section 6.2), table partitioning would not improve any query execution speeds for data warehouse relational databases. It would yield similar performance for partitioned and non-partitioned fact tables.

6.3 Columnstore index studies

Columnstore index is a new feature in SQL Server 2012 that can be used to speed up the processing time on common data warehousing queries. This section evaluates the benefits of columnstore indexing on a TPC-H DW database, by implementing columnstore indices on the largest tables and then comparing its performance against the database without columnstore indexing.

6.3.1 Understanding columnstore index

Columnstore index groups and stores data for each column and then joins all the columns to complete the index. These differ from traditional indexes which group and store data for each row and then join the rows to complete the index.

For some types of queries, the SQL Server query processor can take advantage of the columnstore layout to significantly improve query execution times. SQL Server columnstore index technology is especially appropriate for typical data warehousing data sets. Columnstore indexes can transform the data warehousing experience for users by enabling faster performance for common data warehousing queries such as filtering, aggregating, grouping, and star-joining queries. Read the MSDN Library article "Columnstore Indexes" at http://msdn.microsoft.com/en-us/library/gg492088.aspx

Figure 9 Row store versus columnstore indexes

The steps involved in creating a columnstore index are detailed in Microsoft's IEEE paper "Columnar storage in SQL Server 2012", that can be found at http://sites.computer.org/debull/A12mar/apollo.pdf. The key steps are:

1. Each column is converted to a set of rows called column segments. (The columnstore index is divided into units of transfer called segments. A segment is stored as a Large Object (LOB), and can consist of multiple pages).
2. The rows are divided into row groups (Each row-group has about one million rows).
3. Each row group is encoded and compressed independently to form one compressed column segment for each column included in the index.
4. Each column segment is stored as a separate Binary Large Object (blob) and may span across multiple pages.
5. A segment directory (Contains additional metadata about each segment such as number of rows, size, how data is encoded, and min and max values) keeps track of segments location so all segments containing a column can be easily located..
6. Dictionaries (A storage element that is used as a means to efficiently encode large data types) provide mapping to segments in a columnstore index. Columnstore index dictionaries make it possible to pull only segments that are needed when a plan is created and a query executed. Dictionary compression is used for (large) string columns and the resulting dictionaries are stored in separate blobs. The dictionary compression technique can yield very good compression for repeated values, but yields bad results if the values are all distinct.

This index storage technique offers several benefits (listed in section 6.3.2). It must be noted that the column segments and dictionaries are not stored in the page-oriented buffer pool but in a separate memory pool designed for handling large objects. The objects are stored adjacently and not scattered across separate pages in the memory pool. This improves column scanning as there are no page breaks involved.

The SQL Server query processor can take advantage of the columnstore layout to significantly improve query execution times. SQL Server columnstore index technology is especially appropriate for typical data warehousing data sets. Columnstore indexes can transform the data warehousing experience for users by enabling faster performance for common data warehousing queries such as filtering, aggregating, grouping, and star-join queries.

6.3.2 Columnstore index benefits

Columnstore Index offers query performance and space saving benefits for SQL Server data warehouse database applications.

- **Space Savings:** The columnstore index is compressed when created. On creation, it uses the VertiPaq™ compression algorithm, which compresses the data more than either *page* or *row* compression. These indexes are much smaller than b-tree indexes.
- **Query Performance: The optimizations used to speed up the query processing here are**:
  - **Index compression:** Columnstore Indexes are much smaller which allows faster index scans.
  - **Optimized memory structures:** Columnstore Index uses its own memory pool and does not use fixed size pages from the buffer pool provided sufficient memory is available for SQL Server.
  - **Parallelism:** The batch mode feature enables for parallel executions.

There can be both row store index and column store index on the same table. The query optimizer will decide when to use the column store index and when to use other types of indexes.

6.3.3 Limitations of using columnstore index

Some key limitations of columnstore index listed by Microsoft are stated below. For all other detailed limitations refer to the MSDN Library article titled, "Basics: Columnstore Index Restrictions and Limitations" at http://msdn.microsoft.com/en-us/library/gg492088.aspx#Restrictions

- Creating a columnstore index makes the table read-only.
- Columnstore index might not be the ideal choice for selective queries, which touch only one (or a few) rows or queries that lookup a single row or a small range of rows. Ex: OLTP workloads
- Cannot be clustered. Only non-clustered columnstore indexes are available.
- Cannot act as a primary key or a foreign key.
- A columnstore index on a partitioned table must be partition-aligned.
- A column store cannot be combined with
  - *page* or *row* compression
  - Replication
  - Change tracking
  - Change data capture
  - File stream

6.3.4 Columnstore index performance comparisons - TPC-H

The tests performed in this section evaluate the performance gains in a data warehouse database application when using columnstore indexes on the largest tables. Three tests were performed.

- Baseline tests, with no columnstore index on the eight partitions file layout (refer to Figure 6Figure 6).
- Implementing columnstore index on the first largest partitioned table (LineItem).
- Implementing Columnstore index on the largest partitioned table (LineItem) and the second largest non-partitioned table (Order). This test was done to demonstrate the columnstore index usage on both kinds of tables (partitioned and non-partitioned tables).

Note: When creating columnstore indexes on a partitioned table, changes to the table partitioning syntax is not required. A columnstore index on a partitioned table must be partition-aligned with the base table. Therefore a non-clustered columnstore index can only be created on a partitioned table if the partitioning column is one of the columns in the columnstore index.

Table 5 Columnstore index studies test parameters

Configuration parameters	
EqualLogic SAN	One PS6110X (2.5", 24 10 K SAS drives,900 GB)
RAID type	RAID 50
Columnstore index tests- volume layout	
Baseline (no columnstore index & changed SQL Max memory to 115 GB) LineItem table • Eight partitions	• Primary data File (.mdf) – 100GB • LineItem table partition1 & non-clustered indexes (.ndf) – 1 TB • LineItem table partition2 (.ndf)-200 GB • LineItem table partition3 (.ndf)-200 GB • LineItem table partition4 (.ndf)-200 GB • LineItem table partition5 (.ndf)-200 GB • LineItem table partition6 (.ndf)-200 GB • LineItem table partition7 (.ndf)-200 GB • LineItem table partition8 (.ndf)-200 GB • Order table & non-clustered indexes (.ndf)-300 GB • All other tables & non-clustered indexes (.ndf)–200 GB • Log file (.ldf)-100 GB • Tempdb-600 GB

Configuration parameters	
Columnstore index on LineItem table LineItem table • Eight partitions • Columnstore index aligned with the table partition scheme	• Primary data file (.mdf) – 100GB • LineItem table partition1 & non-clustered indexes (.ndf) – 1 TB • LineItem table partition2 (.ndf)-200 GB • LineItem table partition3 (.ndf)-200 GB • LineItem table partition4 (.ndf)-200 GB • LineItem table partition5 (.ndf)-200 GB • LineItem table partition6 (.ndf)-200 GB • LineItem table partition7 (.ndf)-200 GB • LineItem table partition8 (.ndf)-200 GB • Order table & non-clustered indexes (.ndf)-300 GB • All other tables & non-clustered indexes (.ndf)–200 GB • Log file (.ldf)-100 GB • Tempdb-600 GB
Columnstore index on LineItem & Order tables LineItem table • Eight partitions • Columnstore index aligned with the table partition scheme Order table • Columnstore index	• Primary data file (.mdf) – 100GB • LineItem table partition1 & non-clustered indexes (.ndf) – 1 TB • LineItem table partition2 (.ndf)-200 GB • LineItem table partition3 (.ndf)-200 GB • LineItem table partition4 (.ndf)-200 GB • LineItem table partition5 (.ndf)-200 GB • LineItem table partition6 (.ndf)-200 GB • LineItem table partition7 (.ndf)-200 GB • LineItem table partition8 (.ndf)-200 GB • Order table & non-clustered indexes (.ndf)-300 GB • All other tables & non-clustered indexes (.ndf)–200 GB • Log file (.ldf)-100 GB • Tempdb-600 GB
Data warehouse workload parameters	
Database load & workload generation	TPC-H from Benchmark Factory
Database size	300 scale (~620 GB including data and indexes)
Number for users/streams	6 (As per TPC-H standard, the minimum required streams/users to be run for a 300 scale database is 6)
Queries	22 TPC-H queries per user
SQL Server memory (Max memory allocation) in GB	115 GB (refer Figure 10 & note below)
SQL Server parameters	-E, -T1117 and lock pages in memory
CPU	4* Intel® Xeon® Processor E5-4620 @2.20 GHz,8 cores per socket

The Columnstore Index objects, such as the column segments and dictionaries, do not use the page-oriented buffer pool. Instead, they use their own memory pool which is created automatically by SQL Server memory management and designed for handling large objects. Columnstore Index query operations consume more memory. This can be seen from Figure 10, where six users were run from TPC-H on the database with no columnstore index and with columnstore index with 92% of RAM (117.7 GB) allocated to SQL Server by max memory setting. For the database with columnstore index, the test completed with error 701 "*There is insufficient system memory in resource pool 'default' to run this query*". This can be seen from Figure 10 presented with arrows, where the available memory reaches close to 0 MByte at times. These values were collected using Perfmon during the test duration.

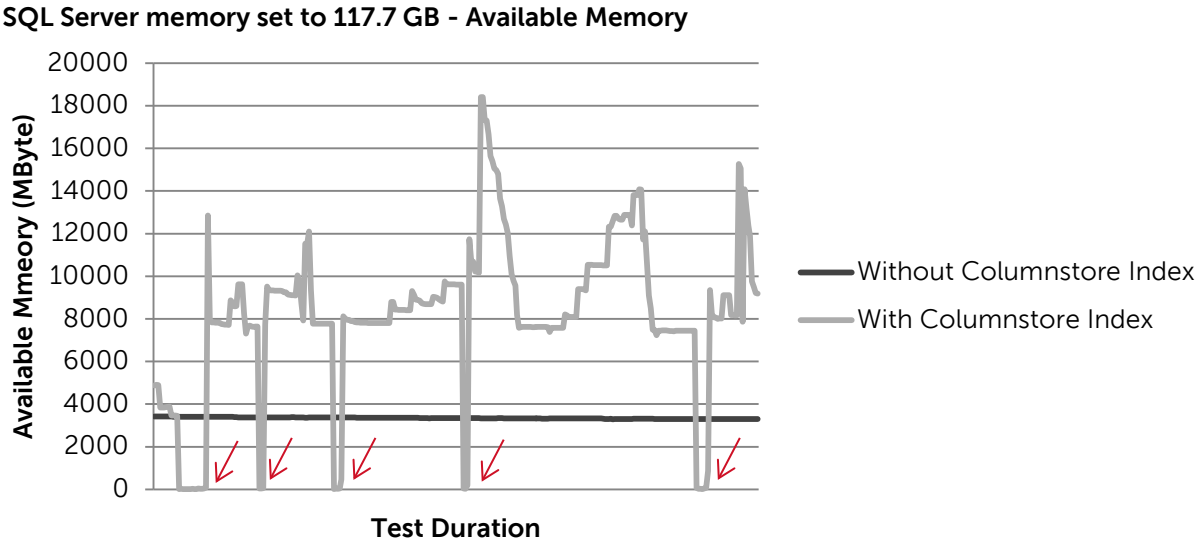

Figure 10 Available Memory comparison – with and without Columnstore Index at 117.7GB SQL Server memory

Hence the SQL Server max memory setting was reduced to 115 GB from 117.7 GB for the columnstore index studies. The MaxDOP (Maximum Degree of Parallelism) and Resource Governor settings were left at defaults. A separate baseline test with no columnstore index (refer to Figure 11) was performed with this changed memory setting at the SQL Server to compare with the columnstore index tests.

Note: To avoid memory issues like error 701, adjusting the MaxDOP (Max Degree of Parallelism) and Resource Governor setting or providing sufficient RAM on the servers running SQL Server databases would help. Refer to http://support.microsoft.com/kb/2834062 and http://social.technet.microsoft.com/wiki/contents/articles/3540.sql-server-columnstore-index-faq.aspx for more details. For the tests conducted in this paper, the MaxDOP and Resource Governor settings were at defaults.

Figure 11 shows the average read throughput and query execution time comparisons for the three tests performed (explained in Table 5). While implementing columnstore index on the largest partitioned table, the query execution time reduced by 58% and the average read throughput reduced by 41% compared to the baseline test without columnstore indexing. The slight increase in the baseline read throughput (906

MB/sec) compared to the read throughput (895 MB/sec) in Figure 7 is due to the change in the memory setting to 115 GB.

While implementing the columnstore index on the two largest tables, the query execution time further reduced by 67% compared to the baseline without columnstore indexing and by 21% compared to having the columnstore index on just the largest table. This shows that implementing columnstore index on large tables is beneficial. While columnstore index can be built on very small tables, the performance advantage is less noticeable when the table is small.

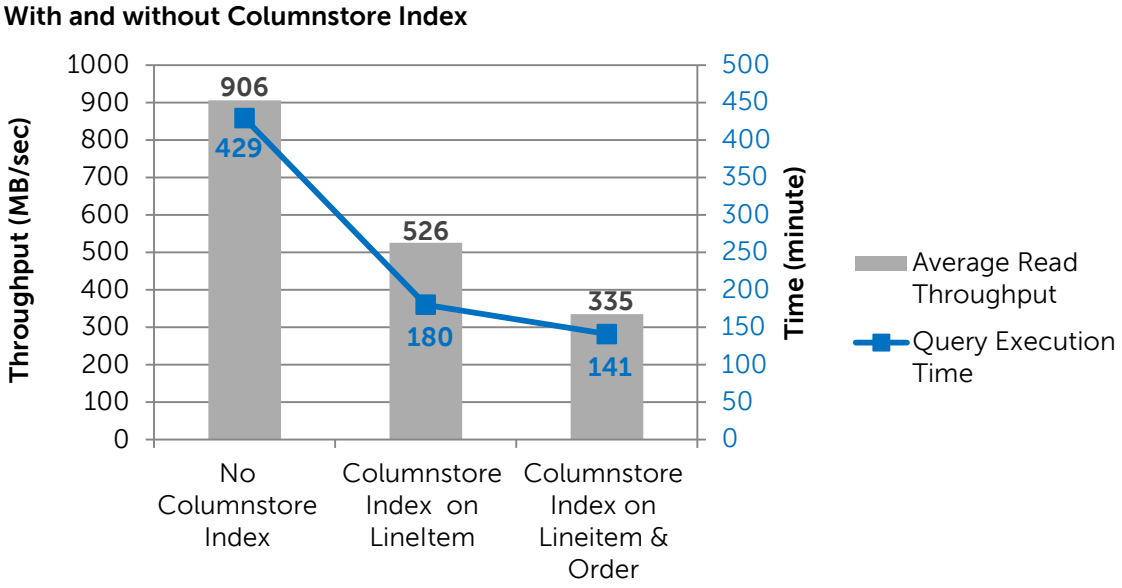

Figure 11 Throughput and execution time comparison with and without columnstore index

The decrease in throughput in the columnstore index is due to the change in the I/O block size and read/write percentage (refer to the SANHQ results shown in Figure 14). This change in block sizes and read/write percentage happened while using columnstore index because only the needed columns were brought into the memory and were heavily compressed. The data can often be compressed more effectively when it is stored in columns rather than in rows. Typically, there is more redundancy within a column than within a row allowing greater compression.

When data is compressed more, the effective data that is fetched from the storage is less. A larger fraction of the data can reside in a given size of memory. This reduces the read throughput significantly, allowing for faster query response times. Retaining more of the working set data in memory speeds up response times for subsequent queries that access the same data.

Columnstore index efficiency enables heavier workloads to run. EqualLogic storage has more room to achieve throughput at around 895 MB/sec (table partition studies, Figure 7). However, the higher workload test (users greater than six) was not performed since the scope of this test was limited to evaluating the performance benefits of using columnstore index and not saturating the single array.

Figure 12 shows the increase in percentage CPU utilization when using the columnstore index measured using Perfmon at the SQL Server. Even though the CPU utilization percentage increased, it remained well below 80%. The increase in CPU utilization is attributed to the data compression and other memory management operations involved during the columnstore index query processing.

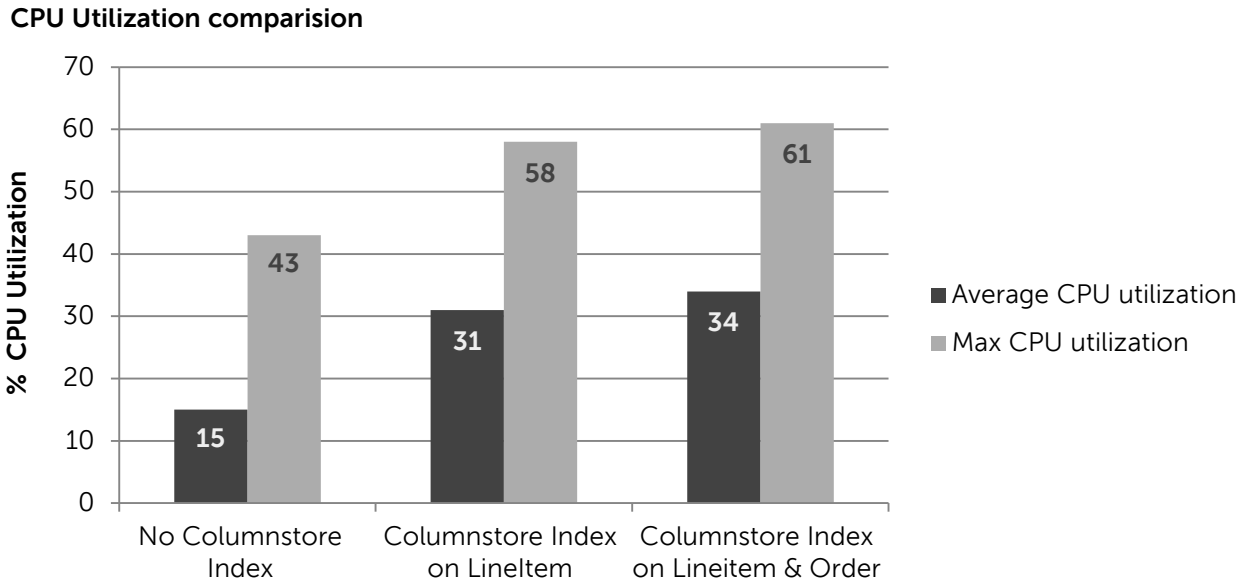

Figure 12 CPU utilization comparisons – With and without Columnstore Index

Note: To reduce CPU time, a new set of query operators called batch mode processing, which processes a batch of rows all at once, occurs when using columnstore index. Standard query processing in SQL Server is based on a one row at a time iterator model. The query optimizer decides to use batch-mode or row-mode operators. In SQL Server 2012, only a subset of the query operators is supported in batch mode. They are scan, filter, project, hash (inner) join and (local) hash aggregation.

The sample SAN HQ results for these tests can be seen in Figure 13 and Figure 14. Figure 13 shows the pool view in SAN HQ for the baseline test without columnstore index and Figure 14 shows the columnstore index on the LineItem table while running six users from Benchmark Factory.

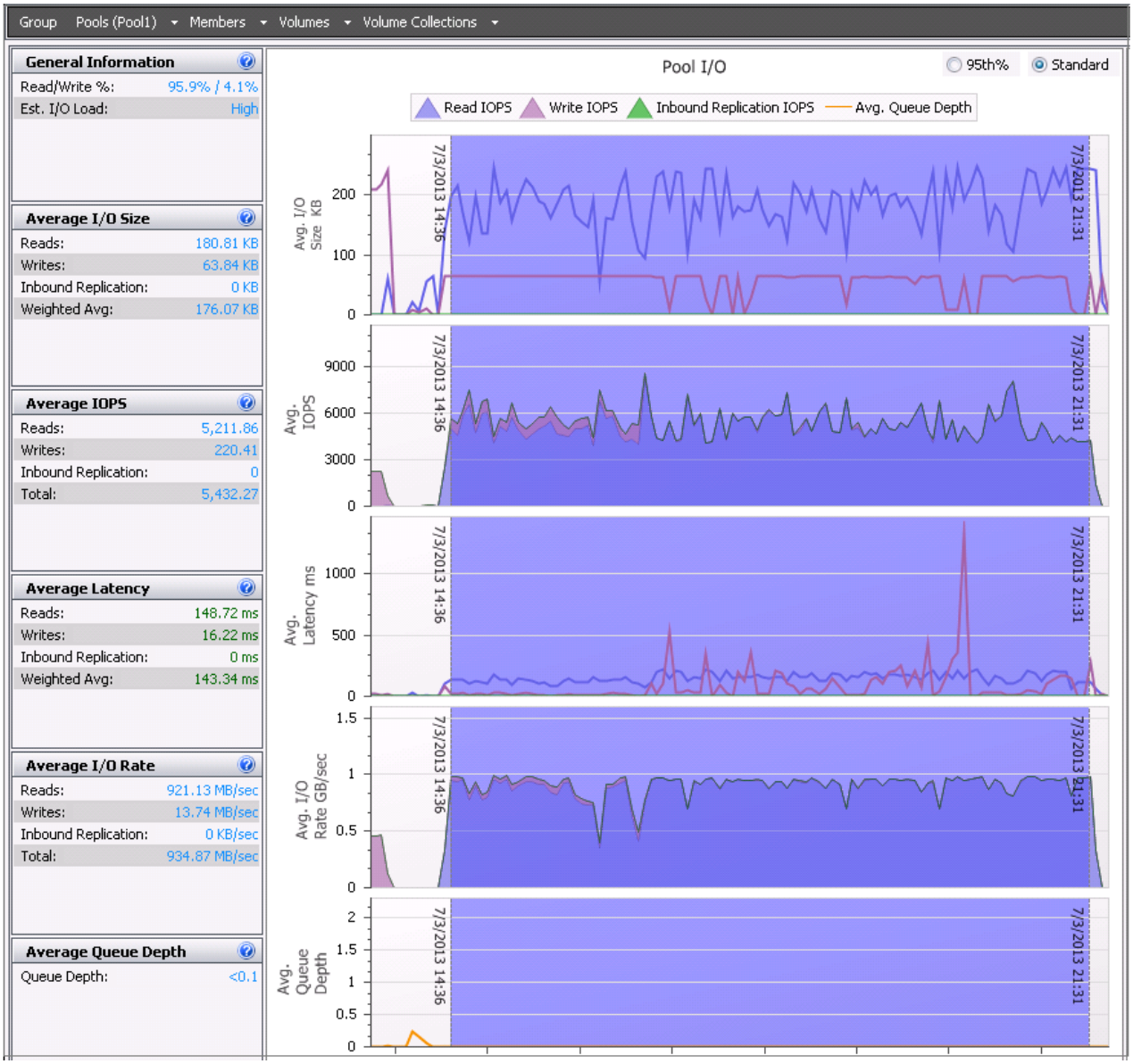

Figure 13 Baseline without columnstore index (6 users) – SAN HQ pool view

Figure 14 Columnstore index on the LineItem table (6 users) – SAN HQ pool view

Table 6 summarizes the results from the above SANHQ graphs.

Table 6 SAN HQ results summary

Performance metrics	No columnstore index	Columnstore index on LineItem table
Read/Write %	95.9/4.1	87.1/12.1
Read IO size (KB)	180.81	95.18
Write IO size (KB)	63.84	63.84
Read latency (ms)	148.72	66.03
Write latency (ms)	16.22	2.57
Read IO rate (MB/sec)	921.13	540.55
Write IO rate (MB/sec)	13.74	53.25

To check the query execution time improvement in the columnstore index tests shown in Figure 11, the queries were examined individually by running a single user power test (one user with 22 queries run in the same order). The power test was run on the baseline tests (without columnstore index) and columnstore index implemented on the largest partitioned LineItem table. Table 7 shows the comparison of the average query response times taken by each of the 22 queries.

Table 7 One user power test query response time comparison

Queries	No. columnstore index	Columnstore index on LineItem table
Pricing summary report query (Q1)	396.356	38.739
Minimum cost supplier query (Q2)	11.718	6.916
Shipping priority query (Q3)	270.43	152.184
Order priority checking query (Q4)	375.791	102.903
Local supplier volume query (Q5)	310.635	84.571
Forecasting revenue change query (Q6)	239.414	9.767
Volume shipping query (Q7)	421.015	93.029
National market share query (Q8)	542.994	107.376
Product type profit measure query (Q9)	312.575	892.794
Returned item reporting query (Q10)	187.646	74.622
Important stock identification query (Q11)	391.593	172.567
Shipping modes and order priority query (Q12)	150.458	60.468
Customer distribution query (Q13)	521.473	99.774
Promotion effect query (Q14)	114.398	26.164
Top supplier query (Q15)	104.548	18.488
Parts-supplier relationship query (Q16)	34.437	22.467
Small-Quantity-Order Revenue Query (Q17)	3.861	3.622
Large Volume Customer Query (Q18)	280.608	173.477
Discounted revenue query (Q19)	44.126	27.588
Potential part promotion query (Q20)	148.875	77.19
Suppliers who kept orders waiting query (Q21)	949.149	385.989
Global sales opportunity query (Q22)	60.426	28.669
Average Query response time (second)	**266**	**120**

From the tests in this section, it is evident that using columnstore index on large tables improved the query response times of almost all the TPC-H queries. For the one user power test conducted in this section, the average query response times lowered by 54% when using columnstore index compared to the power test

without columnstore index. Using columnstore index along with table partition would help make use of the table partition benefits (such as DW loading and maintenance) as well as the columnstore index benefits of improved query execution times.

6.4 SAN scaling

The SAN scaling test measured the throughput scalability as the number of EqualLogic PS Series storage arrays increased within a group. This section covers,

- Performance comparisons between IOMeter and TPC-H test results on a single array.
- Scaling the arrays (1 server and 2 arrays).
- Scaling both the arrays and the hosts (2 servers and 2 arrays).

6.4.1 Performance comparaison - IOMeter versus TPC-H

This section compares the maximum throughput achievable from a single array using IOMeter (section 5) and table partition tests (section 6.2.2) prior to performing the scalability studies. In the TPC-H application test, performance data was gathered from six users with 132 queries. The read/write ratio for the actual application testing was about 96/4, the read I/O block size was around 170 K and the number of volumes was 13 (the read/write ratio and the I/O block size are from SAN HQ). These results can be compared to the 100% sequential reads with 128 K block size IOMeter test on 10 volumes (refer to section 5, Figure 3). The results collected from the studies with one array are graphed in Figure 15.

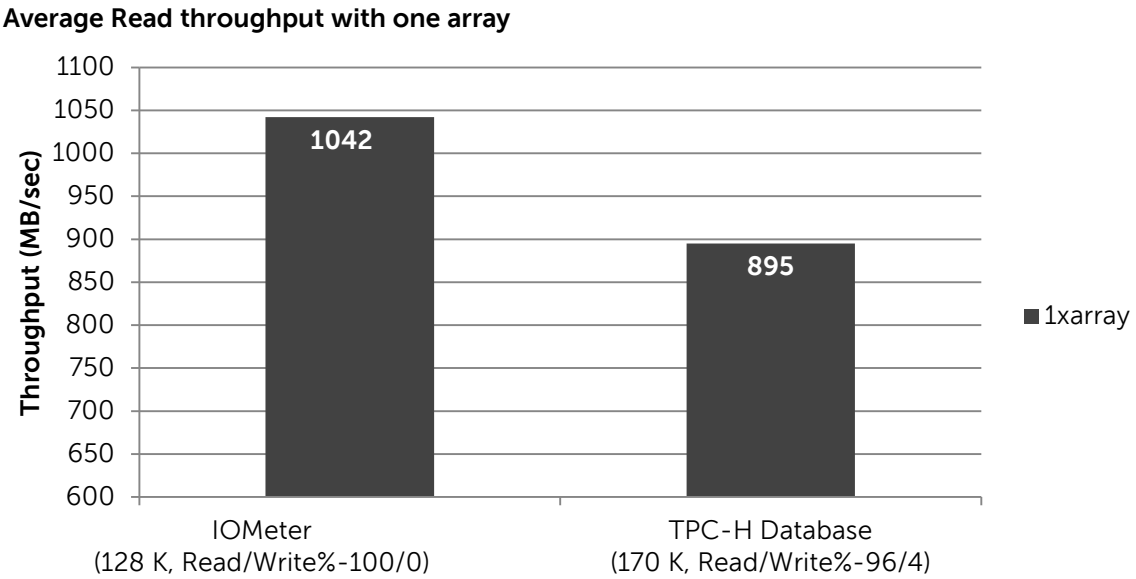

Figure 15 Storage I/O throughput comparison on 1 array – IOMeter and application simulation

Using a more realistic TPC-H Benchmark Factory test, the results presented in Figure 15 show that a single array could achieve an average throughput of 895 MB/sec. A difference in throughput of around 14% was observed between the baseline-IOMeter test simulating a DSS workload and the actual database test on

one array. This was because there was no application cache or I/O writes involved with the IOMeter tool. The Benchmark Factory TPC-H database test had SQL Server buffer cache and a small percentage (4%) of I/O writes (Tempdb and Log files) involved causing the difference in throughput. In addition, the workloads were not 100% sequential when the actual TPC-H database was accessed by multiple users.

6.4.2 Scaling only the arrays with constant user load – TPC-H

For the scalability tests, the eight partition configuration (shown in the table partition studies in section 6.2.2) was used because it provided a better execution time. Six user queries were run against a single database deployed in single array (Test Configuration #1A, Figure 16) and two arrays (Test Configuration #1B, Figure 16). The volume configuration shown in Figure 16 and the configuration parameters shown in Table 8 were used for this test.

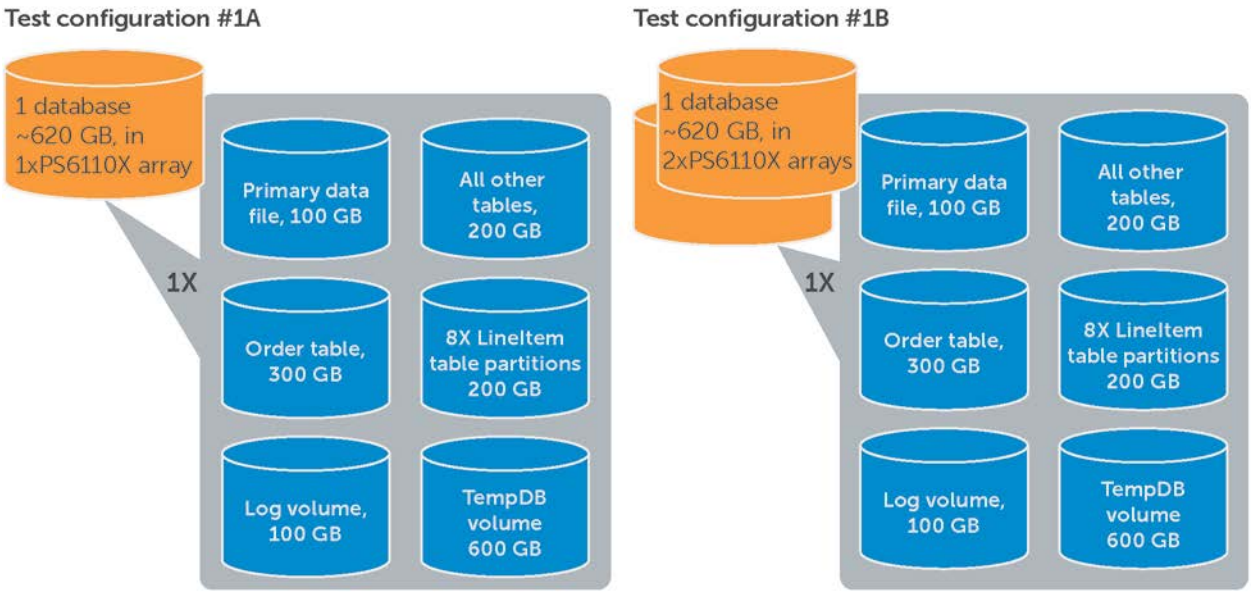

Figure 16 Volume layout for the SAN scaling studies

Table 8 Test parameters: SAN scaling

Configuration Parameters	
EqualLogic SAN	Two PS6110X (2.5", 24 10 K SAS drives,900 GB)
RAID type	RAID 50
Volume Configurations	
Test Configuration #1A (Six users, one database, one array)	• One PS6110X (in one EqualLogic storage pool) • 6 users (132 queries) from Benchmark Factory • Database volumes: Refer to Figure 16
Test Configuration #1B	• Two PS6110X (in one EqualLogic storage pool)

(Six users, one database, two arrays)	• 6 users (132 queries) from Benchmark Factory • Database volumes: Refer to Figure 16
Data Warehouse Workload Parameters	
Database load & Workload generation	TPC-H from Benchmark Factory
Database size	300 scale (~620 GB including data and indexes)
Number for users/streams	6 (As per TPC-H standard, the minimum required streams/users to be run for a 300 scale Database is 6)
Queries	22 TPC-H queries per user
SQL Server Memory (Max memory allocation) in GB	117.7 GB
SQL Server parameters	-E, -T1117 and lock pages in memory
CPU	4* Intel® Xeon® Processor E5-4620 @2.20 GHz,8 Cores per socket

This test checked the performance improvement while adding a second array to the existing pool. The database was spread across both arrays while six users with 132 queries were constantly run on one array. The test was repeated after scaling to two arrays to gather the performance differences. Figure 17 shows the read throughput and query completion time improvements after the second array was added.

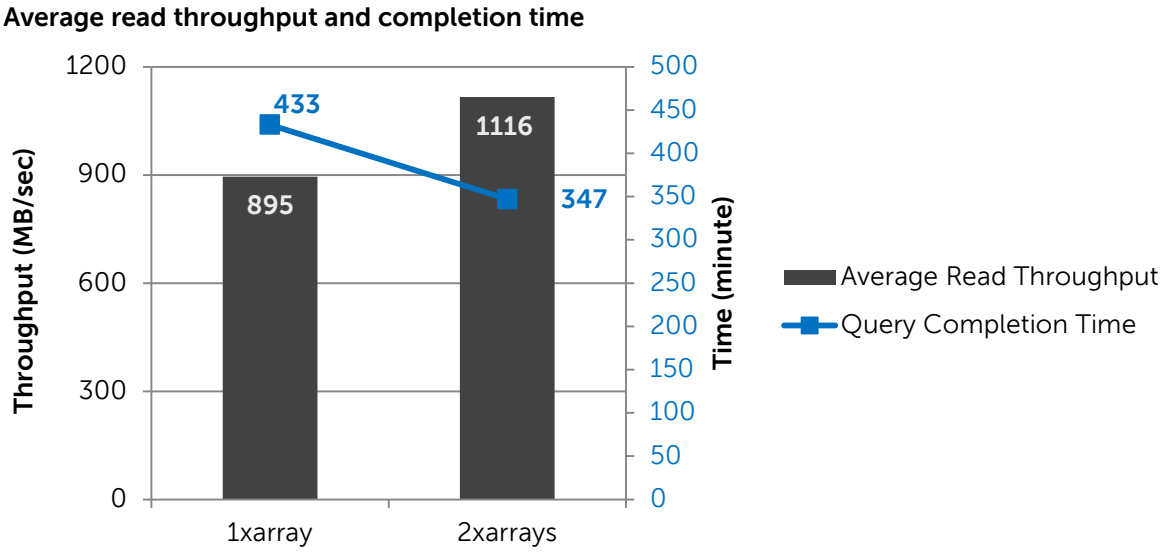

Figure 17 Read throughput, query completion time- performance comparison

The storage throughput increased by 24% for the constant user load by scaling the array. This improvement was due to the increase in storage processing resources such as number of disk drives,

storage processing controllers (including on-board cache), and the network interfaces. The increased storage throughput that became available as the number of arrays increased allowed faster data fetching by the database. For a constant user load, the queries were also completed faster resulting in decreased completion times when the array was scaled. With two arrays, the completion time was 20% less than with a single array.

Proportional scaling in the array throughput was not observed because of the memory limitation at the server. Figure 18 shows the available memory (MBytes) collected using Perfmon during the test duration in both single array and scaled arrays scenarios with constant load from a single server. The average available MBytes reduced by 16% when the second array was added and this limited the linear throughput scalability. Here adding more RAM would help to achieve linear scalability.

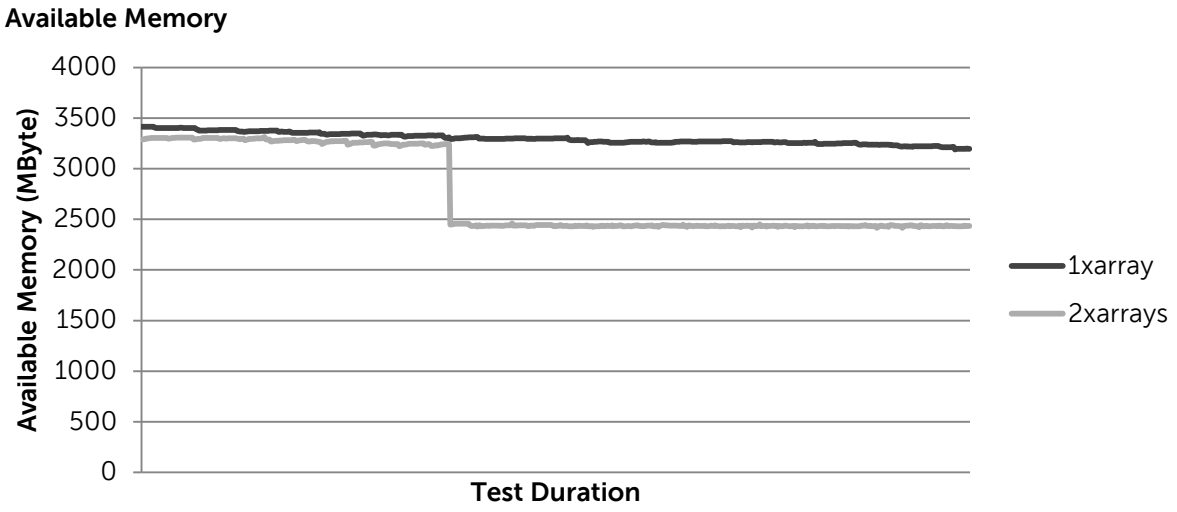

Figure 18 Available memory comparison

Since DW user queries consumed substantial CPU processing, to check the CPU utilization at the hosts during the array scaling tests, the average and maximum CPU utilization percentages were measured using Perfmon. As shown Figure 19, the average and the maximum CPU utilizations observed in this test were well below 80%. This was a result of the higher processor power of the Dell PowerEdge R820 servers (four sockets with eight cores in each). An increase of 33% in the average CPU utilization was observed when the array was scaled from one to two for a constant workload from a single host.

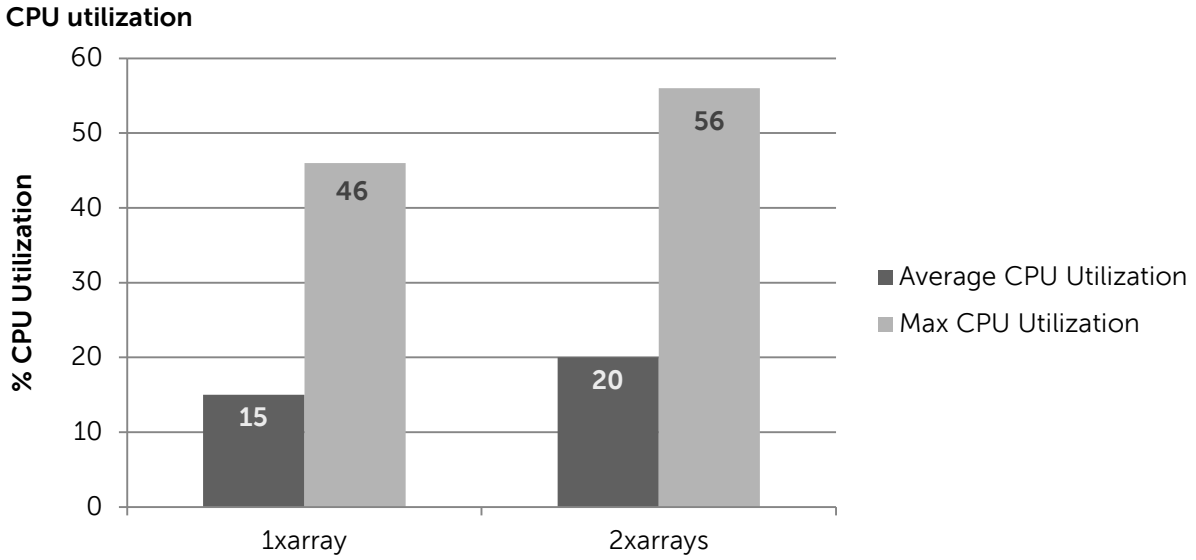

Figure 19 CPU utilization comparison

6.4.3 Scaling the arrays and user load- TPC-H

In this test, both the arrays and the workload were scaled. As arrays were scaled to two, the number of databases and users were also doubled to push more load on the two arrays. Two databases were deployed with 12 users running 264 queries. The volume configuration used for the two array scaling test is shown in Test Configuration #2 of Figure 20 and the configuration parameters shown in Table 9.

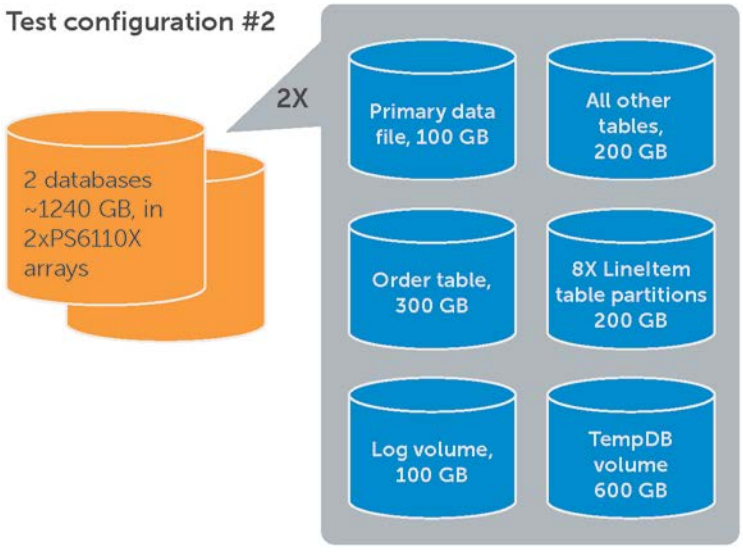

Figure 20 Volume layout for the SAN scaling studies

Table 9 Test parameters: SAN scaling

Configuration Parameters	
EqualLogic SAN	Two PS6110X (2.5", 24 10 K SAS drives,900 GB)
RAID type	RAID 50
Volume Configurations	
Test Configuration #2 (Two databases, two arrays)	• Two PS6110X (in one EqualLogic storage pool) • 12 users (264 queries) from Benchmark Factory • Database volumes: Refer to Figure 20
Data Warehouse Workload Parameters	
Database load & Workload generation	TPC-H from Benchmark Factory
Database size	300 scale (~620 GB including data and indexes)
Number for users/streams	6 (As per TPC-H standard, the minimum required streams/users to be run for a 300 scale Database is 6)
Queries	22 TPC-H queries per user
SQL Server Memory (Max memory allocation) in GB	117.7 GB
SQL Server parameters	-E, -T1117 and lock pages in memory
CPU	4* Intel® Xeon® Processor E5-4620 @2.20 GHz,8 Cores per socket

The scalability test with two arrays in a single storage pool utilizing the volume layout described Figure 20 was performed to get the maximum read throughput by scaling both the arrays and the hosts. When the array was scaled to two, the users were also doubled by scaling the workload. This increased the load that was sent to the arrays to get the maximum read throughput achievable. Figure 21 shows the average read throughput and query completion time comparison between one array and two array scaling.

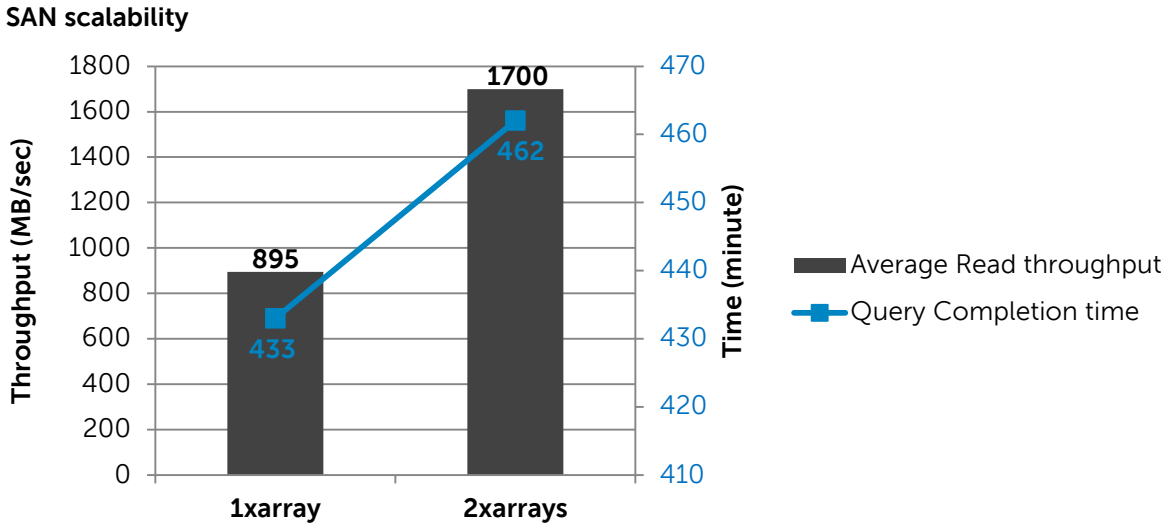

Figure 21 SAN Scalability

Figure 21 Illustrates that the average read throughput scaled proportionally with the addition of more arrays. The query execution times increased by 6% in the two array scaling tests due to the increase in user load. The average and maximum CPU utilization percentages were measured using Perfmon. As shown in Figure 22, the average and the maximum CPU utilizations observed in the scalability test were well below 80%. In the two array scaling tests, the average and maximum CPU utilizations were similar to the one array scaling tests. This was a result of the second Dell PowerEdge R820 server which provided the additional CPU processing power.

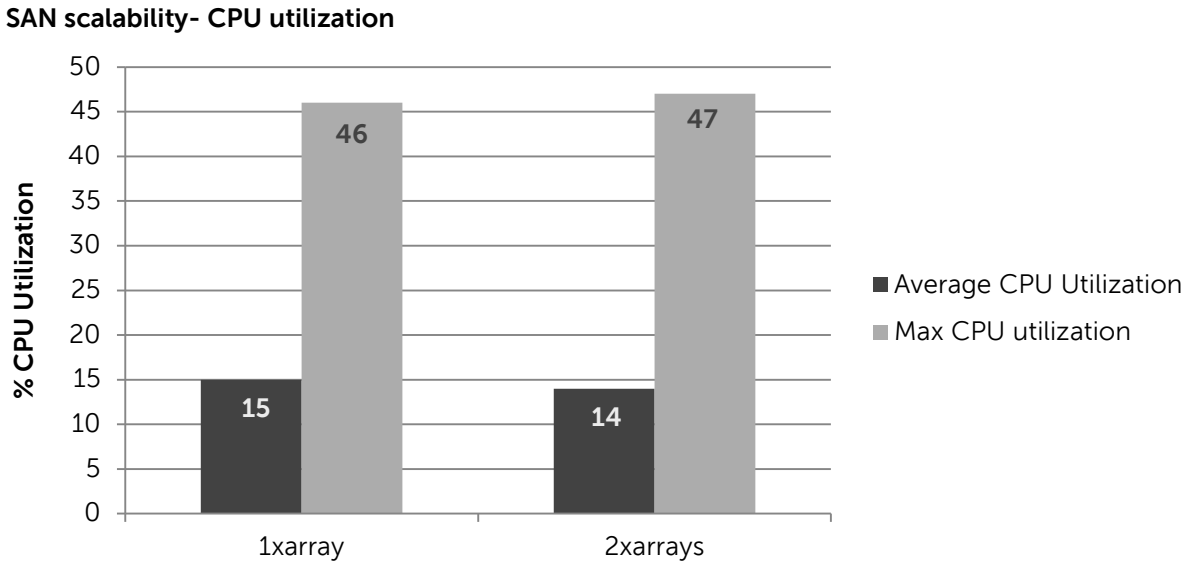

Figure 22 SAN scalability – CPU utilization

The ability to easily scale a data warehouse is crucial for businesses that depend on database applications. From the above scalability tests, the usable capacity and the I/O throughput of the arrays scaled linearly, and without bottlenecks at the servers. The EqualLogic peer storage scale-out architecture increases the available storage resources to accomplish linear scaling. The storage processing resources scaled are the number of disk drives, the number storage processing controllers (including on-board cache), and the number of storage network interfaces.

7 Best practices recommendations

This section summarizes the Dell best practices used in the test setup and also the best practices derived by running the tests results presented sections 5-6.

7.1 Storage

Use the following best practices when configuring Dell EqualLogic storage arrays for a data warehouse solution.

- It is an important to physically separate data warehouse workloads on a separate storage system that is optimized for data warehouse query I/O activity. This will prevent DSS workloads from interfering with the operation of other workloads.
- Because DW applications are typically bandwidth intensive, 10 GbE iSCSI is recommended.
- The storage arrays must be configured to support capacity and performance requirements by considering the hard drive types, RAID policy and array controller capabilities.
- For this testing PS6110X with SAS 10K drives were chosen because they offer a good balance for capacity and throughput needs.
- For data warehouse workloads, RAID 50 is recommended because it offers a good balance between capacity and performance.

7.2 Network Infrastructure

The following list provides network infrastructure design best practices.

- Since data warehouse workloads tend to be sequential in nature and have larger block sizes, the key metric in measuring performance would be throughput. Using 10GbE connectivity between the server and storage would yield a higher throughput performance.
- Design separate network infrastructures to isolate the LAN traffic from the SAN traffic (iSCSI).
- Design redundant SAN component architectures. This includes the NICs on the servers and switches for the storage network (including server blade chassis switches and external switches).
- Make sure that the server NIC ports and storage array NIC ports are connected so that any single component failure in the SAN will not disable access to the storage array volumes.
- Enable flow control on both the server NICs and switch ports connected to the server and storage ports.
- Enable jumbo frames on the server SAN ports and switch ports.
- On iSCSI SAN switches, spanning tree should be disabled on switch ports connecting to end devices like server and storage ports. The Portfast setting should be enabled in the switch configuration for these ports.

7.3 SQL Server Best Practices

The following SQL Server best practices were implemented for the tests documented in this paper.

7.3.1 SQL Server startup parameters

The following startup options are recommended to be added to the SQL Server Startup options:

- **-E:** This parameter increases the number of contiguous extends in each file that are allocated to a database table as it grows. This option is beneficial because it improves sequential access.
- **-T1117:** This trace flag ensures the even growth of all files in a file group when auto growth is enabled.
- **SQL Server Maximum Memory:** It is recommended to allocate no more than 92% of total server RAM to SQL Server. If additional applications will share the server, the amount of RAM left available to the operating system should be adjusted accordingly. For this reference architecture, the maximum server memory was set at 117.76 GB out of 128 GB RAM.
- **Lock Pages in memory:** Grant the windows lock pages in memory privilege to the SQL Server service account. This should be done to prevent the Windows operating system from paging out the buffer pool memory of the SQL Server process by locking memory that is allocated for the buffer pool in physical memory

7.3.2 Database volume creation

- Use a basic disk storage type for all EqualLogic volumes.
- Use the default disk alignment provided by Windows 2008 or greater.
- Use an NTFS file system with a 64 KB allocation unit for SQL database and log partitions.

7.3.3 Files and file groups

A database file is a physical allocation of space and can be designated as primary (.mdf), secondary (.ndf), or log (.ldf). Database objects can be grouped in file groups for allocation, performance, and administration purposes. User defined and primary filegroups are the two types of file groups and either of them can be the default filegroup. The primary file is assigned to the primary filegroup. Secondary files can be assigned to user filegroups or the primary filegroup. Log files are never a part of a file group. Log space is managed separately from data space. Microsoft's recommendations are:

- If the primary filegroup is set as default, the size or the auto grow setting needs to be carefully planned to avoid running out of space.
- Microsoft recommends that with larger database deployments that are easily administrated and for performance reasons, to define a user-defined filegroup as the default. In addition, create all secondary database files in user-defined filegroups so that user objects do not compete with system objects for space in the primary filegroup.

7.3.4 Data file growth

If sufficient space is not initially assigned to a data warehouse database, the database could grow continuously while loading and performance would be affected. Performance is improved if the initial file

size and the percent growth are set to a reasonable size to avoid the frequent activation of the auto grow feature. Microsoft's recommendations are:

- Leave the auto grow feature on at database creation time to avoid running out of space and also to let SQL Server automatically increase allocated resources when necessary without DBA intervention provided there is physical disk space available.
- Set the original size of the database to a reasonable size to avoid the premature activation of the auto grow feature.
- Set the auto grow increment to a reasonable size to avoid the frequent activation of the auto grow feature.

7.3.5 Transaction log file growth

The transaction log is a serial record of all modifications and their executions that occurred in the database. SQL Server uses the transaction log for each database to recover transactions. The log file size depends on the recovery models and the frequency of the log backups. The most preferred recovery model is *full* to minimize downtime and data loss. Data warehouse workloads are primarily read intensive when running user queries and typically generate minimal transaction log activity during read activity. However, the log activity would be significant during the data warehouse load process. Microsoft's recommendations for the transaction log are:

- Place the log and the data files into separate volumes.
- Set the original size of the transaction log to a reasonable size to avoid constant activation of the auto grow feature, which creates new virtual files and stops logging activity as space is added.
- Set the auto grow percent to a reasonable but small enough size to avoid frequent activation of the auto grow feature and to prevent stopping the log activity for too long a duration
- Use manual shrinking rather than automatic shrinking.

7.3.6 Tempdb file growth

The tempdb database is a global resource that holds the temporary tables and stored procedures for all users connected to the system. The tempdb database is recreated every time the SQL Server starts so that the system starts with a clean copy of the database. The tempdb database can be I/O intensive in data warehouse databases. Determining the appropriate size for tempdb in a production environment depends on many factors, including the workload and SQL Server features that are used. To ensure the tempdb is sized and working properly:

- Monitor the tempdb file periodically to check its growth and performance. Perfmon, SANHQ can be used to monitor tempdb volumes.
- When a new database is created, start with allocating 10% of the total database size for all the instances. Adjust the file size as the database grows.
- Pre-allocate space for the tempdb files by setting the tempdb file size to a value large enough to accommodate a typical workload in the environment. If the space is not pre-allocated, set the auto growth increment based on Microsoft's recommendation (refer to Table 11). For the tests conducted in this paper, the tempdb file size was pre-allocated to the largest table size

(LineItem-~300 GB LineItem) and volume size (600 GB), to avoid any issues during query processing. The tempdb I/O profile observed for the TPC-H data warehouse database tests conducted in his paper is shown in Table 10.

- A tempdb I/O can be sequential or random. However, running multiple users or workloads can make the tempdb I/O be more random even when the data warehouse workload is sequential.

Table 10 Tempdb IO profile

Tempdb performance metrics	Values
Read/Write block size	64 K/64 K
Read/Write %	50/50

The above profile is specific to the nature of the TPC-H workload used in the tests; other user databases can generate an I/O that is very different. Periodic monitoring of the existing set up would help identify the nature of tempdb. The Microsoft recommendations below were followed for the tests conducted in this paper.

- Set the recovery model of tempdb to simple. This model automatically reclaims log space to keep space requirements small.
- Pre-allocate space for all tempdb files by setting the file size to a value large enough to accommodate the typical workload in the environment. This prevents tempdb from expanding too frequently, which can affect performance. The tempdb database should be set to auto grow to increase disk space for unplanned exceptions. Data warehouse workloads consume tempdb for their queries. Pre-allocating space for all tempdb files manually would be beneficial.
- Set the file growth increment to a reasonable size to avoid the tempdb database files from growing too frequently by a small value. Microsoft recommends the following general guidelines for setting the file growth increment for tempdb files. For more details, refer to "Optimizing tempdb performance" located at http://msdn.microsoft.com/en-us/library/ms175527%28v=sql.105%29.aspx

Table 11 Tempdb file growth recommendations

Tempdb file size	File growth increment
0 to 100 MB	10 MB
100 to 200 MB	20 MB
200 MB or more	10%

7.3.7 Columnstore Index

The columnstore index study in section 6.3 proved that by using columnstore index on the two largest tables, the data warehouse query execution times improved significantly. Below are the resulting best practices and few things to remember stated by Microsoft when using columnstore index. For details refer to http://msdn.microsoft.com/en-us/library/gg492088.aspx.

- **Updating data in a columnstore index:** Tables that have a columnstore index cannot be updated. There are three ways to work around this problem.
  - To update a table with a columnstore index, drop the columnstore index, perform any required `INSERT`, `DELETE`, `UPDATE`, or `MERGE` operations, and then rebuild the columnstore index.
  - Partition the table and switch partitions. For a bulk insert, insert data into a staging table, build a columnstore index on the staging table, and then switch the staging table into an empty partition.
  - For other updates, switch a partition out of the main table into a staging table, disable or drop the columnstore index on the staging table, perform the update operations, rebuild or re-create the columnstore index on the staging table, and then switch the staging table back into the main table.
  - Place static data into a main table with a columnstore index. Put new data and recent data likely to change into a separate table with the same schema that does not have a columnstore index. Apply updates to the table with the most recent data.
- **Choosing columns for columnstore index:** Some of the performance benefit of a columnstore index is derived from the compression techniques that reduce the number of data pages that must be read and manipulated to process the query. Compression works best on character or numeric columns that have large amounts of duplicated values. For example, dimension tables might have columns for postal codes, cities, and sales regions. If many postal codes are located in each city, and if many cities are located in each sales region, then the sales region column would be the most compressed, the city column would have somewhat less compression, and the postal code would have the least compression. Although all columns are good candidates for a columnstore index, adding the sales region code column to the columnstore index will achieve the greatest benefit from columnstore compression, and the postal code will achieve the least.
- **Columnstore index on partitioned table:** Columnstore indexes are designed to support queries in very large data warehouse scenarios where partitioning is common. When creating columnstore indexes on a partitioned table, they must be partition-aligned with the base table. Therefore a non-clustered columnstore index can only be created on a partitioned table if the partitioning column is one of the columns in the columnstore index.
- **Memory:** Column store processing is optimized for in-memory processing. SQL Server implements mechanisms that enable data (and most data structures) to spill to disk when insufficient memory is available. If severe memory restrictions are present, processing uses the row store. There may be instances in which the columnstore index is chosen as an access method but memory is insufficient to build the needed data structures. The effective memory requirement for any query depends on the specific query. Building a columnstore index requires approximately 8 megabytes times the number of columns in the index, times the DOP (degree

of parallelism). Generally the memory requirements increase as the proportion of columns that are strings increases. Therefore, decreasing the DOP can reduce the memory requirements for building the columnstore index. Creating a columnstore index is a parallel operation, subject to the limitations on the number of CPUs available and any restrictions set on MAXDOP setting.

- **CPU:** Creating a column store index might take slightly longer than creating clustered row store index on the same set of data, as extra CPU cycles are required for compression. From the tests in section 6.3.4, there was an increase in the percentage CPU utilization while running TPC-H queries on tables with columnstore index. To avoid any CPU bottlenecks while using columnstore index, it is necessary to set the needed CPU cores at the server running SQL Server database. The new batch mode processing enhancement in the Query Optimizer, optimized for multicore CPUs and increased memory throughput of modern hardware architecture would be beneficial in reducing the CPU time.
- **Does not support SEEK:** If the query is expected to return a small fraction of the rows, then the optimizer is unlikely to select the columnstore index. If the table hint `FORCESEEK` is used, the optimizer will not consider the columnstore index.
- **Cannot be combined with the following features**:
  - Page and row compression, and vardecimal storage format (a columnstore index is already compressed in a different format.)
  - Replication
  - Change tracking
  - Change data capture
  - Filestream

8 Conclusion

In today's IT world, as data is growing very rapidly, organizations are being stressed to optimize for storage capacity as well as for application performance. Data warehouse applications help organizations achieve better and faster data analytics to improve decision making processes while staying competitive. However, before a data warehouse application is deployed, specific design decisions and principles should be taken into consideration to meet capacity and performance requirements. It is also very important to be able to scale these data warehouse environments linearly as the organizations grow along with the data.

Dell tests validated that EqualLogic PS Series arrays, along with Microsoft SQL Server 2012 provides users with a flexible, high-performing, and robust data warehouse solution. The lab validated tests explained in this paper prove that EqualLogic 10 GbE iSCSI PS6110X storage arrays provide high levels of I/O throughput warranted by data warehouse applications. Taking the performance level a step further, the scalability of EqualLogic arrays was also tested and the results show that they scale linearly in both performance and capacity as an organization grows.

The tests presented in this paper were:

- The SQL Server tuning parameter tests performed in section 6.1 proved that the Microsoft recommended configurations improve the query execution times for data warehouse workloads.
- SQL Server table partitioning can be implemented to speed up the data warehouse bulk load operations, backups/restores and maintenance activities. However, they may not be beneficial to speed up the query performance. The partitioning scheme and structure need to be selected based on the nature of the queries and in line with the business requirements. Evaluate the table partition benefits on a test environment before implementing it into production.
- Columnstore Index significantly improved the Data warehouse query execution times as seen in section 6.3.
- EqualLogic PS arrays provide the high throughput that DSS applications demand. In addition, with the scalability of array resources, the system also scales proportionally in both performance and capacity as the business grows.

Dell EqualLogic PS Series storage arrays provide a flexible, scalable, high-performing, and robust foundation for a data warehouse implementation based on Microsoft SQL Server 2012.

A Configuration details

This section contains an overview of the hardware configurations used throughout the testing described in this document.

Table 12 Test configuration hardware components

Test Configuration	Hardware Components
SQL Server® 1	One PowerEdge R820 server running Windows Server 2012, hosting 1 SQL Server database instance: BIOS Version: 1.4.0 4 x 8 Core Intel® Xeon® E5-4620 Processors 2.20 GHz 128 GB RAM, 4 x 146GB 15K SAS internal disk drives Broadcom 5720 1GbE quad-port NIC (LAN on motherboard), firmware version 7.4.8 Intel(R) Ethernet 10G 2P X520 Firmware level 13.5.2
SQL Server® 2	One PowerEdge R820 server running Windows Server 2012, hosting 1 SQL Server database instance: BIOS Version: 1.4.0 4 x 8 Core Intel® Xeon® E5-4620 Processors 2.20 GHz 128 GB RAM, 4 x 146GB 15K SAS internal disk drives Broadcom 5720 1GbE quad-port NIC (LAN on motherboard), firmware version 7.4.8 Intel(R) Ethernet 10G 2P X520 Firmware level 13.5.2
INFRA SERVER	One (1) Dell PowerEdge R710 Server running VMware ESXi 5, hosting two (2) Windows Server 2008 R2 virtual machines for Active Directory and vCenter: BIOS Version: 6.3.0 Quad Core Intel® Xeon® X5570 Processor 2.67 GHz 48 GB RAM,2 x 146GB 15K SAS internal disk drives Broadcom 5709c 1GbE quad-port NIC (LAN on motherboard) – firmware version 7.4.8
LOAD GEN SERVER	One (1) Dell PowerEdge R710 Server running VMware ESXi 5, hosting 1 Windows Server 2008 R2 virtual machine for Quest Benchmark Factory: BIOS Version: 6.3.0 Quad Core Intel® Xeon® X5650 Processor 2.67 GHz 48 GB RAM, 2 x 146GB 15K SAS internal disk drives Broadcom 5709c 1GbE quad-port NIC (LAN on motherboard) – firmware version 7.4.8
MONITOR SERVER	One (1) Dell PowerEdge R710 Server with Windows Server 2008 R2 for SANHQ: BIOS Version: 6.3.0 Intel Xeon X5650 Processor 2.67 GHz 48 GB RAM,2 x 146GB 15K SAS internal disk drives Broadcom 5709c 1GbE quad-port NIC (LAN on motherboard) – firmware version

	7.4.8
Network	2 x Force10 S4810 10Gb Ethernet Switch, Firmware: 8.3.12.1
Storage	2 x EqualLogic PS6110X: 24 x 900GB 10K RPM SAS disk drives as RAID 50, with two hot spare disks Dual 10GbE controllers running firmware version 6.0.2

This section contains an overview of the software configurations used throughout the testing described in this document.

Table 13 Test configuration software components

Test Configuration	Software Components
Operating systems	• Microsoft Windows Server 2012 Enterprise Edition • EqualLogic Host Integration Toolkit(HIT) version 4.5.0 installed
Applications	SQL Server 2012 SP1 Enterprise Edition
Monitoring Tools	EqualLogic SAN Headquarters version 2.5 Windows Perfmon
Simulation Tools	Benchmark Factory for Databases version 6.9

B Columnstore index

T-SQL query to create and align the columnstore Index along the table partition scheme is as follows.

```
USE [300_TPCH_DB]
GO
SET ANSI_PADDING ON
GO
CREATE NONCLUSTERED COLUMNSTORE INDEX [NonClusteredColumnStoreIndex-20130620-
123327] ON [dbo].[H_Lineitem]
(
   [l_orderkey],
   [l_partkey],
   [l_suppkey],
   [l_linenumber],
   [l_quantity],
   [l_extendedprice],
   [l_discount],
   [l_tax],
   [l_returnflag],
   [l_linestatus],
   [l_shipdate],
   [l_commitdate],
   [l_receiptdate],
   [l_shipinstruct],
   [l_shipmode],
   [l_comment]
) WITH (DROP_EXISTING = OFF) ON [LineItem_8_PS]([l_shipdate]);

GO
```

Additional resources

Support.dell.com is focused on meeting your needs with proven services and support.

DellTechCenter.com is an IT Community where you can connect with Dell Customers and Dell employees for the purpose of sharing knowledge, best practices, and information about Dell products and your installations.

Referenced or recommended Dell publications:

- Dell EqualLogic Configuration Guide:
http://en.community.dell.com/dell-groups/dtcmedia/m/mediagallery/19852516/download.aspx

Referenced or recommended Microsoft publications:

- Partitioned Table and Index Strategies Using SQL Server 2008
http://msdn.microsoft.com/en-us/library/dd578580.aspx
- Using Partitions in a Microsoft SQL Server 2000 Data Warehouse
http://msdn.microsoft.com/library/aa902650%28SQL.80%29.aspx
- Fast Track Data Warehouse Reference Guide for SQL Server 2012
http://msdn.microsoft.com/en-us/library/hh918452.aspx
- Column Store Indexes
http://msdn.microsoft.com/en-us/library/gg492088.aspx
- SQL Server Columnstore Index FAQ
http://social.technet.microsoft.com/wiki/contents/articles/3540.sql-server-columnstore-index-faq.aspx

For EqualLogic best practices white papers, reference architectures, and sizing guidelines for enterprise applications and SANs, refer to Storage Infrastructure and Solutions Team Publications at:

- http://dell.to/sM4hJT