



Fast Track Data Warehouse 2.0 Architecture

SQL Server Technical Article

Writers: Dave Salch, Eric Kraemer, Umair Waheed, Paul Dyke

Technical Reviewers: Jose Blakeley, Stuart Ozer, Eric Hanson, Mark Theissen, Mike Ruthruff

Published: November 2009

Updated: 14 October 2009

Applies to: SQL Server 2008 Enterprise

Summary: This paper defines a reference configuration model (known as Fast Track Data Warehouse) using a CPU core-balanced approach to implementing a symmetric multiprocessor (SMP)-based SQL Server data warehouse with proven performance and scalability expectations on data warehouse workloads. The goal of a Fast Track Data Warehouse reference configuration is to achieve a cost-effective balance between SQL Server data processing capability and realized component hardware throughput.

Copyright

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Excel, PowerShell, SQL Server, Windows, and Windows Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

- Introduction 4
 - Audience 4
- Fast Track Data Warehouse 4
- Methodology 5
- FTDW Workload Evaluation 7
- Choosing a FTDW Reference Configuration 10
 - Option 1: Basic Evaluation..... 10
 - Option 2: Full Evaluation 11
 - Option 3: User-Defined Reference Architectures 13
- FTDW Configuration 14
 - Hardware Configuration..... 14
 - Application Configuration..... 16
- SQL Server Best Practices for FTDW 19
 - Data Architecture..... 19
 - Indexing..... 21
 - Managing Data Fragmentation 22
 - Loading Data 23
- Benchmarking and Validation..... 25
 - Component Evaluation 26
 - System Component Validation 27
- Fast Track Data Warehouse Reference Configurations 31
- Conclusion 32
- Appendix..... 34
- FTDW CPU Core Calculator 34
- Integration Services Parallel Bulk Load Examples 35
- Validating a Fast Track Reference Architecture 37
 - Synthetic I/O Testing 37
 - Workload Testing 40

Introduction

This document defines the component architecture and methodology for the SQL Server Fast Track Data Warehouse program (FTDW). The result of this approach is the validation of a minimal Microsoft® SQL Server® data management software configuration, including software and hardware, required to achieve and maintain a baseline level of “out of box” scalable performance when deploying an SQL Server database for many data warehousing workloads.

Audience

The target audience for this document consists of IT planners, architects, DBAs, CIOs, CTOs, and business intelligence (BI) users with an interest in options for their BI applications and in the factors that affect those options.

Fast Track Data Warehouse

The SQL Server Fast Track Data Warehouse initiative provides a basic methodology and concrete examples for the deployment of balanced data warehouse architectures for predefined customer workloads.

Balance is measured across the key components of a SQL Server installation; storage, server, application settings, and configuration settings for each component are evaluated. The goal is to achieve an efficient out-of-the-box balance between SQL Server data processing capability and aggregate hardware I/O throughput. Ideally, minimum storage is purchased to satisfy customer storage requirements and provide sufficient disk I/O for SQL Server to achieve a benchmarked maximum data processing rate.

Fast Track

The Fast Track designation signifies a component hardware configuration that conforms to the principles of the FTDW reference architecture. The reference architecture is defined by a workload and a core set of configuration, validation, and database best practice guidelines. The following are key principles of a Fast Track reference architecture:

- Detailed and validated hardware component specifications
- Validated methodology for database and hardware component evaluation
- Component architecture balance between database capability and hardware bandwidth

Value Proposition

The following principles create the foundation of the FTDW value proposition:

- **Predetermined balance across key system components.** This minimizes the risk of overspending for CPU or storage resources that will never be realized at the application level. This translates to a lower total cost of ownership (TCO) for customers in deploying data mart and data warehouse solutions.
- **Predictable out-of-the-box performance.** Fast Track configurations are built to capacity that already matches the capabilities of the SQL Server application for a selected server and workload.

- **Workload-centric.** Rather than being a one-size-fits-all approach to database configuration, the FTDW approach is aligned specifically with a data warehouse use case.

Methodology

Data Warehouse Workload Patterns

Typically questions asked of data warehouses require access to large volumes of data. Data warehouses need to support a broad range of queries from a wide-ranging audience (for example, finance, marketing, operations, and research teams).

In order to overcome the limitations of traditional data warehouse systems, organizations have resorted to using traditional RDBMS optimization techniques such as building indexes, preaggregating data, and limiting access to lower levels of data. The maintenance overheads associated with these approaches can often overwhelm even generous batch windows. As a data warehouse becomes more mature and the audience grows, supporting these use-case specific optimizations becomes even more challenging, particularly in the case of late-arriving data or data corrections.

A common solution to this challenge is to throw hardware at the issue, particularly on the storage side; it is not uncommon to see hundreds of disks supporting small data warehouses, to overcome the I/O performance limitations of current storage architectures, particularly those using shared SAN. Traditional optimization techniques can encourage random I/O access patterns, introducing disk latency and reducing the overall storage subsystem throughput.

Fast Track Data Warehouse is a different way of thinking about optimizing an RDBMS for data warehouse workloads. If the sequential layout data is designed to better support scanning operations, the performance achieved from individual disks is much higher, resulting in an increase of aggregate I/O throughput. The performance improvement reduces the number of disks to support a given workload. Furthermore, there is a significant decrease in the need for optimization techniques such as indexes and preaggregations. Cost savings on storage can be passed onto the CPU side of the equation, enabling faster data processing. This lowers the not only the cost of the storage infrastructure, but also the cost of maintenance. Queries are supported from granular data, removing the need to rebuild indexes and aggregations.

When considering workloads for Fast Track Data Warehouse based systems, the data warehouse architect should consider not only whether the current data warehouse design fits the FTDW workload, but whether the data warehouse could benefit if the FTDW best practices outlined in this document are adopted.

Holistic Component Architecture

Fast Track reference architectures provide a practical framework for balancing the complex relationships between key components of a database system architecture. Referred to generically as a “stack”, the component architecture is illustrated in Figure 1.

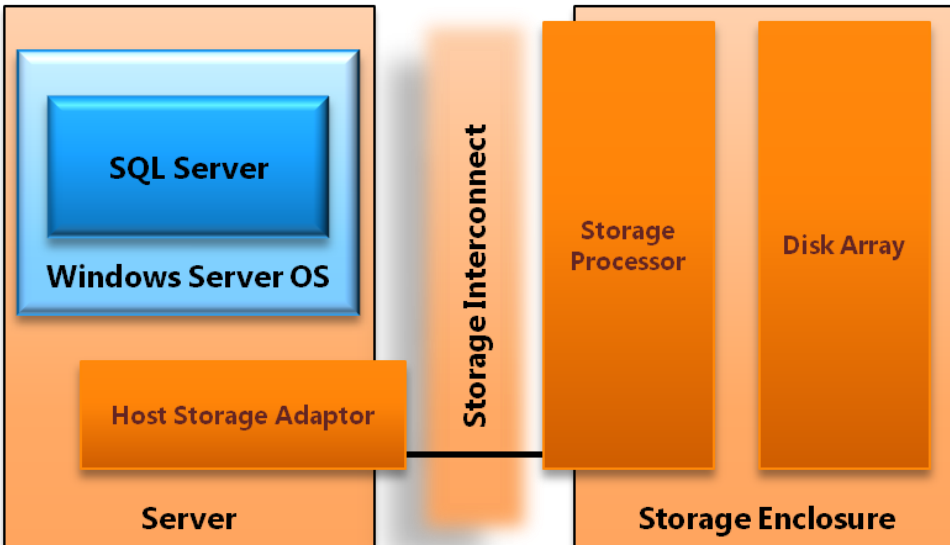


Figure 1: Example Fast Track database component architecture

Each component of the stack is a link in a chain of operations necessary to process data in SQL Server. Evaluating the stack as an integrated system enables benchmarking that establishes real bandwidth for each component. This ensures that individual components provide sufficient throughput to match the capabilities of the SQL Server application for the prescribed stack.

Workload Optimized

Different database application workloads can require very different component architectures to achieve optimal resource balance. A classic example of this can be found in the contrast between discrete lookup based OLTP workloads and scan-intensive analytical data warehousing. OLTP use cases are heavily indexed to support low latency retrieval of small numbers of rows from relatively large data sets. These types of database operations induce significant disk head movement and generate classic random I/O scan patterns. Analytical use cases, such as data warehousing, can involve much larger data requests and benefit greatly from the increased total throughput potential of sequential disk scans.

For these contrasting use cases, the implications for a balanced component stack are significant. Average, per-disk random I/O scan rates for modern SAS disk drives can be a factor of 10 times slower when compared to sequential scan rates for the same hardware. This translates directly to the amount of disks required to provide SQL Server enough disk I/O to fully utilize CPU resources for a given server.

The challenge of very different workloads is addressed by clearly defining the attributes of customer workloads associated with the FTDW program. FTDW workloads comprise a qualitative list of attributes that uniquely define a common database application use case. In addition, each workload is represented by standard benchmark queries. Workload-centric benchmarking is used to validate database configuration, best practices, and component hardware recommendations.

Validated FTDW Reference Configurations

All published Fast Track reference configurations are validated as conforming to the set of principles and guidelines provided in this document. Examples of this process can be found in later sections of this document.

Summary

FTDW specifications are workload-centric and component balanced. The approach acknowledges that one-size-fits-all provisioning can be inefficient and costly for many database use cases. Increasingly complex business requirements coupled with rapidly scaling data volumes demand a more realistic approach. By presenting a combination of prescriptive reference architectures, benchmarking of hardware and software components, and clearly targeted workloads, this document provides a practical approach to achieving balanced component architectures.

FTDW Workload Evaluation

FTDW configurations are positioned specifically for data warehousing scenarios. The data warehousing workload has the following core principles relating to SQL Server database operations; when applying FTDW principles or reference configurations, it is important to evaluate the affinity of the target workload to these high-level principles.

Scan-Intensive

Queries in a data warehouse workload typically scan a large number of rows after filters are applied, so disk scan performance becomes an increasing priority in contrast to transactional workloads that stress disk seek time. The FTDW reference architecture optimizes hardware and database software components with disk scan performance as the key priority. This results in more efficient sequential disk reads, increasing overall disk I/O throughput per drive.

Nonvolatile

After data is written, it is rarely changed. DML operations, such as SQL update, that move pages associated with the same database table out of contiguous alignment should be minimized. Workloads that introduce such volatility may not be well aligned to FTDW. Where volatility does occur, periodic maintenance to minimize fragmentation is essential over time.

Index-Light

In FTDW-based systems, minimize the use of secondary indexes. Adding nonclustered indexes generally adds performance to discrete lookups of one or few records. However, if nonclustered indexes are applied to tables in which large numbers of rows are to be retrieved, additional fragmentation and increased random I/O disk scan patterns can degrade overall system performance.

Maintaining indexes introduces a significant data management overhead, which can generate challenges in meeting service-level agreements (SLAs) and load windows. The net performance benefit to an environment tuned for large analytical scan patterns can become negative.

Sequential scan rates can be many factors higher (10 times or more) than random access rates. In addition, indexes are of reduced value when large data scans are required by most queries. For these reasons, this document prescribes other performance optimizations, such as clustered index or table partitioning.

Partition-Aligned

Partitioning can simplify data lifecycle management and assist in minimizing fragmentation over time. In addition, query patterns for large scans can take advantage of range partition qualification.

Key Considerations

The following additional considerations should be taken into account during the evaluation of a database workload:

- The implementation and management of an index-light database optimization strategy is a fundamental requirement for FTDW workloads.
- It is assumed that minimal data fragmentation will be maintained within the data warehouse. This implies the following:
 - Expanding the server by adding storage requires that all performance-sensitive tables be rewritten in a manner consistent with guidelines provided in this document.
 - Implementing highly volatile data structures, such as tables with regular row-level update activity, may also require frequent maintenance (such as defragmentation or index rebuilds) to reduce fragmentation.
- Data warehousing is can mean many things to different audiences. Care should be taken to evaluate customer requirements against FTDW workload attributes.

Descriptive Data Warehouse Workload Attributes

The FTDW workload can be defined through the properties of the following subject areas related to database operations:

- User requirements and access pattern
- Data model
- Data architecture
- Database optimization

The following table summarizes data warehouse workload attributes; contrast is provided through comparison to an OLTP workload.

Attribute**Workload Affinity****Data Warehouse****OLTP**

Use Case Description	<ul style="list-style-type: none">• Read-mostly (90%-10%)• Updates generally limited to data quality requirements• High-volume bulk inserts• Medium to low overall query concurrency; peak concurrent query request ranging from 10-30.• Concurrent query throughput characterized by analysis and reporting needs• Large range scans and/or aggregations• Complex queries (filter, join, group-by, aggregation)	<ul style="list-style-type: none">• Balanced read-update ratio (60%-40%)• Concurrent query throughput characterized by operational needs• Fine-grained inserts and updates• High transaction throughput (for example, 10s K/sec)• Medium-to-high overall user concurrency. Peak concurrent query request ranging from 50-100 or more• Usually very short transactions (for example, discrete minimal row lookups)
Data Model	<ul style="list-style-type: none">• Highly normalized centralized data warehouse model• Denormalization in support of reporting requirements often serviced from BI applications such as SQL Server Analysis Services• Dimensional data structures hosted on the database with relatively low concurrency, high volume analytical requests• Large range scans are common• Ad-hoc analytical use cases	<ul style="list-style-type: none">• Highly normalized operational data model• Frequent denormalization for decision support; high concurrency, low latency discrete lookups• Historical retention of data is limited• Denormalized data models extracted from other source systems in support of operational event decision making
Data Architecture	<ul style="list-style-type: none">• Significant use of heap table structures• Large partitioned tables with clustered indexes supporting range restricted scans• Very large fact tables (for example, hundreds of gigabytes to multiple terabytes)• Very large data sizes (for example, hundreds of terabytes to a petabyte)	<ul style="list-style-type: none">• Minimal use of heap table structures• Clustered index table structures support detailed record lookups (1 to few rows per request).• Smaller fact tables (for example, less than 100 GB)• Relatively small data sizes (for example, few terabytes)
Database Optimization	<ul style="list-style-type: none">• Minimal use of secondary indexes (described earlier as index-light)• Partitioning is common	<ul style="list-style-type: none">• Heavy utilization of secondary index optimization

Choosing a FTDW Reference Configuration

There are three general approaches to using the FTDW methodology described within this document. The first two are specific to the use of published, conforming Fast Track reference architectures for data warehousing. These approaches enable the selection of predesigned systems published as part of the FTDW program. The third approach treats core Fast Track methodology as a guideline for the creation of a user-defined data warehousing system. This implies detailed workload profiling and system benchmarking, and it requires a high degree of technical knowledge.

Option 1: Basic Evaluation

The basic evaluation process describes a scenario in which a full platform evaluation (proof of concept) will not be performed. Instead, the customer has already targeted an FTDW reference configuration or has alternative methods to determine server and CPU requirements.

Step 1: Evaluate the Customer Use Case

Fast Track Data Warehouse reference configurations are not one-size-fits-all configurations of software and hardware. Rather, they are configured for the characteristics of a data warehousing workload.

Workload

FTDW workload definitions provide two key points for use case evaluation. The first is a set of core principles that define key elements of the workload as it relates to SQL Server performance. These principles should be measured carefully against a given use case as conflicts may indicate a target workload is not appropriate for an FTDW reference architecture.

The second component to a workload is a general description of the targeted use case. This provides a useful high-level description of the use case in addition to provide a reasonable starting point for evaluating workload fit.

Workload Evaluation

The following list outlines a basic process for customer workload evaluation. This is a qualitative assessment and should be considered a guideline:

1. Define the targeted workload requirements. Compare and contrast to FTDW workload attributes. For more information, see the [FTDW Workload Evaluation](#) section of this document.
2. Evaluate FTDW best practices. Practices relating to database management and data architecture and system optimization should be evaluated against the target use case and operational environment.

Making a Decision

The goal of this workload assessment is to ensure that you make a fully informed decision when you choose a published FTDW reference configuration. In reality most data warehousing scenarios represent a mixture of conforming and conflicting attributes relative to the FTDW

workload. High priority workload attributes with a strong affinity for Fast Track reference configurations are listed below; primary customer use cases that directly conflict with any of these attributes should be carefully evaluated.

Workload

The following workload attributes are high priority:

- Critical workloads feature scan-intensive data access patterns (that is, those that can benefit from sequential data placement). In general, individual query requests involve tens of thousands to millions (or higher) of rows.
- High data capacity, low concurrency relative to common OLTP workloads.
- Low data volatility. Frequent update/delete DML activity should be limited to a small percentage of the overall data warehouse footprint.

Database Management

This includes database administration, data architecture (data model and table structure), and data integration practices:

- Index-light, partitioned data architecture.
- Careful management of database fragmentation, through suitable loading and ETL strategies and periodic maintenance.
- Predictable data growth requirements. FTDW systems are prebuilt to fully balanced capacity. Storage expansion requires data migration.

Step 2: Choose a Published FTDW Reference Configuration

A customer may have a server in mind when performing a simple evaluation based on budget or experience. Alternatively, the customer may already have a good idea of workload capacity or an existing system on which to base analysis of bandwidth requirements. In any case, a full platform evaluation is not performed in an FTDW simple evaluation. Instead, a conforming FTDW configuration is selected that matches the estimated customer requirements.

Option 2: Full Evaluation

Fast Track-conforming reference architectures provide hardware component configurations paired with defined customer workloads. The following methodology allows for a streamlined approach to choosing a database component architecture that ensures better out-of-the-box balance among use case requirements, performance, and scalability.

Process Overview

The following process flow summarizes the FTDW selection process:

- Evaluate Fast Track workload attributes against the target usage scenario.
- Identify server and/or bandwidth requirements for the customer use case. A published FTDW reference configuration must be chosen to begin an evaluation.
- Identify a query that is representative of customer workload requirement.
- Calculate the Benchmark Consumption Rate (BCR) of SQL Server for the query.

- Calculate the Required User Data Capacity (UDC).
- Compare BCR and UDC ratings against published Maximum CPU Consumption Rate (MCR) and Capacity ratings for conforming Fast Track reference architectures.

The following three-step process describes individual points of the process flow in additional detail.

Step 1: Evaluate the Customer Use Case

Workload Evaluation

This process is the same as for Option 1.

Select FTDW Evaluation Hardware

Before you begin a full system evaluation, you must choose and deploy a published FTDW reference configuration. You can choose among several methods to identify an appropriate reference configuration. The following approaches are common:

- Budget. The customer chooses to buy the highest-capacity system and/or highest-performance system for the available budget.
- Performance. The customer chooses to buy the highest-performing system available.
- In-house analysis. The decision is based on workload analysis the customer has executed on existing hardware.
- Ad-hoc analysis. The [FTDW CPU Core Calculator](#) provides a basic approach to calculating CPU requirements based on basic assumptions about the targeted database workload.

Step 2: Establish Evaluation Metrics

The following three metrics are important to a full FTDW evaluation, and they comprise the key decision criteria for hardware evaluation:

- Maximum CPU Core Consumption Rate (MCR)
- Benchmark Consumption Rate (BCR)
- Required User Data Capacity (UDC)

MCR

This metric measures the maximum SQL Server data processing rate for a standardized query against a standardized data set for a specific server and CPU combination. This is provided as a per-core rate measured from memory cache (that is, assuming no I/O bottlenecks) for all published and conforming FTDW reference configurations. MCR is the basis from which the published FTDW reference configurations are designed.

BCR

BCR is measured by a query or set of queries that have been identified as definitive of a targeted customer workload. BCR is also calculated in terms of actual bandwidth from disk, rather than from cache as with the MCR calculation. The relative comparison of MCR to BCR

values provides a common reference point when evaluating different reference architectures for unique customer use cases. The determination of BCR can enable tailoring of the infrastructure for a given customer use case.

UDC

This is simply the customer-required user data capacity for the SQL Server database. It is important that growth rates are taken into account when determining UDC. Fast Track databases cannot simply be expanded by adding new storage and new LUNs. Any storage expansion beyond initial deployment requires data migration to maintain even distribution of data across disk volumes allocated to database files.

As MCR is provided with each FTDW reference configuration for evaluation, and UDC is a customer-supplied metric. The focus of this step is to establish the BCR metric. The following process describes the standard approach used to measure BCR.

Calculating BCR

Detailed definition of the BCR calculation process can be found in the “**Error! Reference source not found.**” section of this document. To perform this calculation, follow these steps:

1. Perform basic system validation to ensure component hardware configuration.
2. Identify a single query (or a small group of queries) that is representative of customer workload. This query will be used to establish BCR.
3. Execute and record the BCR benchmark.

Step 3: Choose a Fast Track Data Warehouse Reference Configuration

After it is calculated, BCR is compared against published MCR and capacity ratings provided with each published FTDW reference configuration. This creates a common reference point for evaluating conforming FTDW reference configurations.

Option 3: User-Defined Reference Architectures

The FTDW methodology presented is specific to the data warehousing workload. For this reason it is very important to interpret all recommendations and practices within that context. The following steps provide a general outline for developing user-defined reference architectures for data warehousing.

Step 1: Define Workload

Understanding the target database use case is central to FTDW configurations, and this applies equally to any custom application of the guidance provided within this document. Guidance for Fast Track RAs, specifically on the topic of workloads, can be used as a reference model for incorporating workload evaluation into component architecture design.

Step 2: Establish Component Architecture Benchmarks

The following framework provides an outlined approach to developing a reference architecture for a predefined workload. For more information about this process, see the **Error! Reference source not found.** section of this document.

1. Establish the Maximum CPU Core Consumption Rate (MCR).
 - a. Use the method outlined in the **Error! Reference source not found.** section to calculate the standardized MCR.
 - b. You can also use published MCR ratings for FTDW configurations. In general, CPUs of the same family have similar CPU core consumption rates for the SQL Server database.
2. Establish a Benchmark Consumption Rate (BCR).
 - a. Based on workload evaluation, identify a query for CPU consumption rate testing.
 - b. Use the results of the query to establish a BCR.
3. Determine basic disk I/O and CPU requirements relative to the MCR and the BCR.
4. Establish the server and storage configuration required to meet the BCR.

Step 3: System Validation

The goal of system benchmarking should be configuration and throughput validation of the hardware component configuration identified in Step 2. For more information about this process, see the **Error! Reference source not found.** section of this document.

1. Evaluate component throughput against the established target BCR. This ensures that real system throughput matches expectations.
2. Validate system throughput by re-establishing and re-evaluating the BCR.

FTDW Configuration

Hardware Configuration

Component Architecture

Fast Track Data Warehousing reference architectures are based on a dedicated SAN storage configuration. For more information, including examples of conforming, detailed system specifications, see the **Error! Reference source not found.** section of this document. Figure 2 illustrates the component level building blocks that comprise a Fast Track Data Warehousing reference architecture.

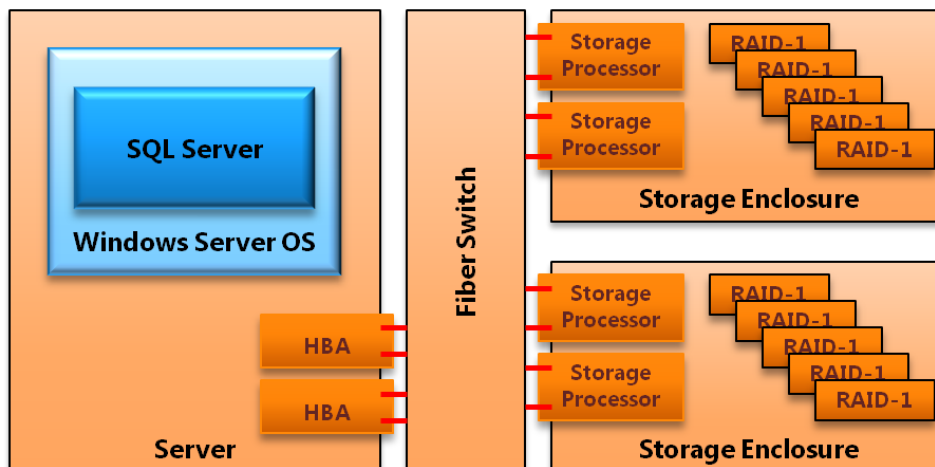


Figure 2: Example storage configuration for a 2-socket, 8-core server

Component Requirements and Configuration

Server

Memory: 4GB of RAM per core is the minimum recommendation. More memory may be necessary for certain workloads. The following considerations are important when evaluating system memory requirements:

- **Query from cache:** Workloads that service a large percentage of queries from cache may see an overall benefit from increased RAM allocations as the workload grows.
- **Hash Joins and Sorts:** Queries that rely on large-scale hash joins or perform large-scale sorting operations will benefit from large amounts of physical memory. With smaller memory, these operations will spill to disk and heavily utilize **tempdb**, which introduces a random I/O pattern across the data drives on the server.
- **Loads:** Bulk inserts can also introduce sorting operations that utilize **tempdb** if they cannot be processed in available memory.

Local Disk: A 2-disk RAID 1 array is the minimum allocation for operating system and SQL Server application installation. Remaining disk configuration depends on the use case and customer preference.

Fibre Channel SAN

HBA – SAN: All HBA and SAN network components vary to some degree by make and model. In addition, storage enclosure throughput can be sensitive to SAN configuration. This recommendation is a general guideline and is consistent with testing performed during FTDW reference configuration development.

All FC ports should be connected to the FC switch within the same FC zone. Only ports in use for Fast Track should exist in the zone. If the FC switch is dedicated to the Fast Track server, all zoning should be removed. Affinity between HBA port and SAN port is not prescribed by the Fast Track RA. Instead, aggregate bandwidth is assumed across a shared zone – hence a FC switch is required. HBA queue depth should be set appropriate to the anticipated workload.

Multipath I/O (MPIO): MPIO should be configured. Each disk volume / LUN from the SAN should appear as a separate disk in Windows. Each disk hosted on the SAN arrays should have multiple MPIO paths defined. To determine the number of paths, use the following calculation:

$$[\text{Number of HBA ports on server}] * [\text{Number of connected FC ports on host array}]$$

Round-robin should be used where the Microsoft Device-Specific Module (DSM) is configured. Vendor-specific DSMs may require different settings.

Storage

Logical File System: Mounting LUNs to windows folders (mount points) rather than drive letters is preferred due to the number of drives added in a Fast Track system.

It can be useful to understand which Windows® operating system drive assignment represents which LUN (volume), RAID array, and physical disk in the storage enclosures. You can adopt a naming scheme for the mount points and volumes when mounting LUNs to Windows folders. The following example illustrates volume mappings for a single-storage enclosure.

Volume Label	Mount Point	Array	Virtual Disk	LUN (volume)
01DATA01V01	01	01	DATA01	01
01DATA01V02	02	01	DATA01	02
01DATA02V01	03	01	DATA02	01
01DATA02V02	04	01	DATA02	02
02DATA03V01	05	02	DATA03	01
02DATA03V02	06	02	DATA03	02
02DATA04V01	07	02	DATA04	01
02DATA04V02	08	02	DATA04	02

You can use vendor-specific tools to achieve mapping, or you can simply make one disk available to Windows at a time, from the storage arrays while assigning drive names.

Physical File System: For more information, including detailed instructions, see the [Application Configuration](#) section within the [Integrated Storage System](#) section.

Storage Enclosure Configuration: All enclosure settings remain at their defaults. FTDW specifications for file system configuration require SAN storage enclosures that allow specific configuration of RAID groupings and LUN assignments. This should be taken into account for any FTDW reference configuration hardware substitutions or custom hardware evaluations.

Application Configuration

Windows Server 2008

Default settings should be used for the Windows Server® 2008 operating system. The Multipath I/O feature is required.

SQL Server 2008

Startup Options

-E should be added to the start-up options. This increases the number of contiguous extents in each file that are allocated to a database table as it grows. This improves sequential disk access. Microsoft Knowledge Base Article [329526](#) describes the **-E** option in more detail.

-T1117 should be added to the start-up options. This trace flag ensures even growth of all files in a file group.

Resource Governor

Data warehousing workloads typically include a good proportion of complex queries operating on large volumes of data. These queries often consume large amounts of memory, and they can spill to disk where memory is constrained. This has specific implications in terms of

resource management. You can use the Resource Governor technology in SQL Server 2008 to manage resource usage. Maximum memory settings are of particular importance in this context.

In default settings for SQL Server, Resource Governor provides a maximum of 25 percent of SQL Server memory resources to each session. This means that, at worst, three queries heavy enough to consume at least 25 percent of available memory will block any other memory-intensive query. In this state, any additional queries that require a large memory grant to run will queue until resources become available.

If more you need concurrency for memory intensive queries, you can use Resource Governor to reduce the maximum memory consumed per query. However, as a result, concurrent queries that would otherwise consume large amounts of memory utilize **tempdb** instead, introducing more random I/O, which can reduce overall throughput. While it can be beneficial for many data warehouse workloads to limit the amount of system resources available to an individual session, this is best measured through analysis of concurrency benchmarks and workload requirements. For more information about how to use Resource Governor, see [Managing SQL Server Workloads with Resource Governor](http://msdn.microsoft.com/en-us/library/bb933866.aspx) (<http://msdn.microsoft.com/en-us/library/bb933866.aspx>) in SQL Server Books Online.

In summary, there is a trade-off between lowering constraints that offer higher performance for individual queries and more stringent constraints that guarantee the number of queries that can run concurrently. Setting MAXDOP at a value other than the maximum will also have an impact – in effect, it further segments the resources available, and it constrains I/O throughput to an individual query.

For more information about best practices and common scenarios for Resource Governor, see the white paper [Using the Resource Governor](http://msdn.microsoft.com/en-us/library/ee151608.aspx) (<http://msdn.microsoft.com/en-us/library/ee151608.aspx>).

Integrated Storage System

Managing fragmentation is crucial to system performance over time for Fast Track Data Warehouse reference architectures. For this reason, a detailed storage and file system configuration is specified.

Storage System Components

Figure 3 provides a view that combines three primary layers of storage configuration for the integrated database stack. It shows one-half of a full 2-socket, 8-core Fast Track reference architecture. The database stack contains the following elements:

- Physical disk array
 - Primary data allocation
 - Log allocation
- Operating system volume assignment (LUN)
- Database application file system
 - User databases
 - Database Temp space
 - Transaction log allocation

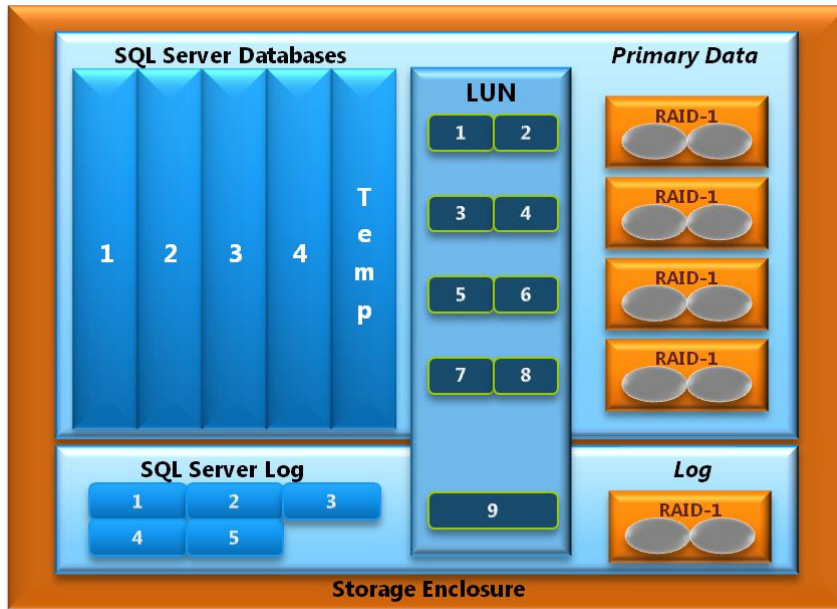


Figure 3: Example comprehensive storage architecture for a single storage enclosure

Storage Configuration Details

For each storage enclosure, do the following.

1. Create five RAID groups of two disks each, using RAID 1.
2. On each of the first four RAID groups, create the primary data space for database files. Create two equally sized LUNs per RAID group. These LUNs will be used to store the SQL Server database files (.mdf and .ndf files). Spread ownership of the disk volumes across available storage processors.
3. Create one LUN on the remaining RAID group to host the database transaction logs.

For each database, do the following:

1. Create at least one filegroup containing one data file per LUN. Be sure to make all the files the same size. If you plan to use multiple filegroups within a single database to segregate objects (for example, a staging database to support loading), be sure to create one data file per LUN for *every* filegroup.
2. When creating the files for each filegroup, preallocate them to their full extent, with a size large enough to hold the anticipated objects.
3. Disable Autogrow for data files, and manually grow all data files when the current size limit is being approached.
4. For more information about recommendations for user databases and filegroups, see the [Managing Data Fragmentation](#) section of this document.

For **tempdb**, do the following:

1. Preallocate **tempdb**, and add a single **tempdb** data file per LUN. Be sure to make all files the same size.
2. Move temp log files onto one of the LUNs dedicated to LOG files.

3. Enable Autogrow for all **tempdb** data files; use a large growth increment, such as a number equivalent to 10 percent of the initial file size. Multiple file growths cause unnecessary overhead. You should set global trace flag 1117 to enable even growth of files across the filegroup.

For the transaction log, do the following:

1. Create a single transaction log file per database on one of the LUNs assigned to the transaction log space. Spread log files for different databases across available LUNs or use multiple log files for log growth as required. Enable Autogrow for log files.
2. The log size must be at least twice the size of largest supported DML operation (using uncompressed data volumes).

tempdb Allocation

Follow standard SQL Server best practices for database and **tempdb** sizing considerations. Greater **tempdb** space may be required during the migration phase or during the initial data load of the warehouse. For more information, see [Capacity Planning for tempdb](http://msdn.microsoft.com/en-us/library/ms345368.aspx) (<http://msdn.microsoft.com/en-us/library/ms345368.aspx>) in SQL Server Books Online.

SQL Server Best Practices for FTDW

Data Architecture

Table Structure

The type of table that is used to store data in the database has a significant effect on the performance of sequential access. It is very important to design the physical schema with this in mind to allow the query plans to induce sequential I/O as much as possible.

There is a choice to be made with regard to the type of table that is used to store your data. The decision comes down to how the data in the table will be accessed the majority of the time. The following decision tree can be used to help determine which type of table should be considered based on the details of the data being stored.

Heap Tables

Heap tables provide clean sequential I/O for table scans and generally lower overhead with regards to table fragmentation. They do not allow for optimized (direct-access) range based scans as found with a clustered index table. In a range scan situation, a heap table scans the entire table (or appropriate range partition, if applicable).

Scanning heap tables reaches maximum throughput at 32 files, so use of heaps for large fact tables on system greater than 16 cores is not recommended, unless other benefits of heap tables outweigh the scan limitations.

It is best to use heap tables where:

- The majority of high priority queries against the table reference contain predicates that reference a variety of disparate columns or have no column predicates.
- Queries normally perform large scans as opposed to range-restricted scans, such as tables used exclusively to populate Analysis Services cubes. (In such cases, the heap table should be partitioned with the same granularity as the Analysis Services cube being populated.)
- Query workload requirements are met without the incremental overhead of index management or load performance is of paramount importance – heap tables are faster to load.

Clustered Index Tables

In the data warehouse environment, a clustered index is most effective when the key is a range-qualified column (such as date) that is frequently used in restrictions for the relevant query workload. In this situation, the index can be used to substantially restrict and optimize the data to be scanned.

It is best to use clustered index tables if:

- There are range-qualified columns in the table that are used in query restrictions for the majority of the high-priority query workload scenarios against the table:
 - For FTDW configurations, the partitioned date column of a CI should also be the clustered index key.
 - Choosing a clustered index key that is not the date partition column for a CI table might be advantageous in some cases. However, this is likely to lead to fragmentation unless *complete* partitions are loaded at a time, because new data that overlaps existing clustered index key ranges creates page splits.
- Queries against the table normally do granular lookups, not larger range scans.

Table Partitioning

Table partitioning provides an important benefit with regard to data management and minimizing fragmentation. Because table partitioning groups the table contents into discrete blocks of data, it enables you to manage these blocks as a whole. For example, using this technique to delete data from your database in blocks can help reduce fragmentation in your database. In contrast, deleting row by row induces fragmentation and may not even be feasible for large tables.

In addition, large tables that are used primarily for populating SQL Server Analysis Services cubes can be created as partitioned heap tables, with the table partitioning aligned with the cube's partitioning. When accessed, only the relevant partitions of the large table will be scanned. (Partitions that support Analysis Services ROLAP mode may be better structured as clustered indexes.)

For more information about table partitioning, click the following link:

<http://download.microsoft.com/download/D/B/D/DBDE7972-1EB9-470A-BA18-58849DB3EB3B/PartTableAndIndexStrat.docx>

Indexing

We recommend the best practices for implementing indexing strategies that work well for sequential data access:

- Use a clustered index for date ranges or common restrictions; otherwise use nonindexed heap tables.
- Put nonclustered indexes on tables where query restriction or granular lookup is required and table partitioning does not provide sufficient performance.
- Limit the use of nonclustered indexes to situations where the number of rows searched for in the average query is very low compared to overall table size.

Database Statistics

When to run statistics and how often to update them is not dependent on any single factor. The available maintenance window and overall lack of system performance are typically the two main reasons where database statistics issues are addressed.

For more information about SQL Server statistics, click the following link:

<http://msdn.microsoft.com/en-us/library/dd535534.aspx>

Best Practices

We recommend the following best practices for database statistics:

- Use the AUTO CREATE and AUTO UPDATE of statistics (the system default in SQL Server). Use of this technique will minimize the need to run statistics manually. There is an initial delay on the first query that uses the table, because the sampling will take place at that time. This is only an issue for the first query; subsequent queries utilize the existing statistics until the number of table modifications (rows added) surpass specific thresholds, at which point statistics are updated again.
- If you must gather statistics manually, statistics ideally should be gathered on all columns in a table. If it is not possible to run statistics for all columns, you should at least gather statistics on all columns that are used in a WHERE or HAVING clause and on join keys. Index creation builds statistics on the index key, so you don't have to do that explicitly.
- Composite (multicolumn) statistics are critical for many join scenarios. Fact-dimension joins that involve composite join keys may induce suboptimal nested loop optimization plans in the absence of composite statistics. Auto-statistics will not create, refresh, or replace composite statistics.
- Statistics that involve an increasing key value (such as a date on a fact table) should be updated manually after each incremental load operation. In all other cases, statistics can

be updated less frequently. If you determine that the `AUTO_UPDATE_STATISTICS` option is not sufficient for you, run statistics on a scheduled basis.

- With larger data sets, if the gathering of statistics is time-consuming and problematic, it you can run the statistics in parallel up to the number of cores available on the configuration.

Compression

The Fast Track data warehouse configurations are designed with page compression enabled. It is highly recommended that you use page compression on all fact tables. Compression of small dimension tables (that is, those with less than a million rows) is not recommended. Normally, you should compress larger dimensions using page compression; you may need to experiment to determine what setting is right for your dimensions.

SQL Server 2008 compression shrinks data in tables, indexes, and partitions, which reduces the amount of physical space required to store data. It also increases query performance by allowing more data to fit into system memory and reducing the numbers of I/O operations required.

The amount of actual compression that can be realized varies relative to the data that is being stored and the frequency of duplicate data fields within the data. If your data is highly random, the benefits of compression are very limited. Even under the best conditions, the use of compression increases demand on the CPU to compress and decompress the data, but it also reduces physical disk space requirements and under most circumstances improves query response time via throughput reduced number of I/Os. Usually, page compression has a compression ratio (original size/compressed size) of between 2 and 7, with 2.5 being typical. Your results may vary depending on your data.

Managing Data Fragmentation

Fragmentation can happen at several levels, all of which must be controlled to preserve sequential I/O. A key goal of a Fast Track Data Warehouse is to keep your data as sequentially ordered as possible while limiting underlying fragmentation. If fragmentation is allowed to occur, overall system performance suffers.

Periodic defragmentation is necessary, but the following guidelines can help you minimize the number of time-consuming defragmentation processes.

File System Fragmentation

Disk blocks per database file should be kept contiguous on the physical platter within the NTFS file system. Fragmentation at this level can be prevented by preallocating all files to their expected size upon creation.

NTFS file system defragmentation tools should be avoided. These tools are designed to work at the operating system level and are not aware of internal SQL Server data file structures.

Extent Fragmentation

Within SQL Server, all of the pages within a file, regardless of table association, can become interleaved down to the extent size (2M) or page level (8K). This commonly occurs due to concurrent DML operations, excessive row-level updates, or excessive row-level deletes.

Fully rewriting the table(s) in question is the only way to ensure optimal page allocation within a file. There are no alternative methods to resolving this type of database fragmentation. For this reason, it is important to follow guidelines for SQL Server configuration and best practices for loading data and managing DML.

Index Fragmentation

An index can be in different physical (page) and logical (index) order.

Do not use the ALTER INDEX REORGANIZE command to resolve this type of fragmentation as this can negate the benefits of large allocations. An index rebuild or an INSERT-SELECT into a new copy of the index (which avoids a resort) can resolve this issue. Any ALTER INDEX REBUILD process should specify SORT_IN_TEMPDB=TRUE to avoid fragmentation of the destination filegroup. MAXDOP 1 should also be used to maintain index page order and improve subsequent scan speeds.

Filegroup Creation

Separate filegroups can be created to handle volatile data use cases such as:

- Tables or indexes that are frequently dropped and re-created (leaving gaps in the storage layout that are refilled by other objects)
- Indexes for which there is no choice but to support as highly fragmented because of page splits, such as incremental data is frequently loaded that mostly overlaps existing clustered index key range
- Smaller tables (such as dimension tables) that are loaded in relatively small increments, which can be placed in a volatile filegroup to prevent those rows from interleaving with large transaction or fact tables
- Staging databases from which data is inserted into the final destination table

Other tables can be placed in a nonvolatile filegroup.

Very large fact tables can also be placed in separate file groups.

Loading Data

The Fast Track component architecture is balanced for higher average scan rates seen with sequential disk access. To maintain these scan rates, care must be taken to ensure contiguous layout of data within the SQL Server file system.

This section is divided into the following two high-level approaches, incremental load and data migration. This guidance is specific, but not exclusive, to Fast Track data warehousing.

For more information about SQL Server bulk load, see [The Data Loading Performance Guide](http://msdn.microsoft.com/en-us/library/dd425070.aspx) (<http://msdn.microsoft.com/en-us/library/dd425070.aspx>).

Incremental Loads

This section covers the common day-to-day load scenarios of a data warehouse environment. This section includes load scenarios with one or more of the following attributes:

- Small size relative to available system memory
- Load sort operations fit within available memory
- Small size relative to the total rows in the target load object

The following guidelines should be considered when you are loading heap and clustered index tables.

Heap Table Load Process

Bulk inserts for heap tables can be implemented as serial or parallel process. Use the following tips:

- To execute the movement of data into the destination heap table, use BULK INSERT with the TABLOCK option. If the final permanent table is partitioned, use the BATCHSIZE option, because loading to a partitioned table causes a sort to **tempdb** to occur.
- To improve load time performance when you are importing large data sets, run multiple bulk insert operations simultaneously to utilize parallelism in the bulk process.

Clustered Index Load Process

Two general approaches exist to loading clustered index tables with minimal table fragmentation:

- **Option 1:** Use BULK INSERT to load data directly into the destination table. For best performance, the full set of data being loaded should fit into an in-memory sort. All data loaded should be handled by a single commit operation by using a BATCHSIZE value of 0. This setting prevents data in multiple batches from interleaving and generating page splits. If you use this option, the load must occur single-threaded.
- **Option 2:** Perform a serial or multithreaded bulk insert to an empty clustered index staging table using moderate, nonzero batch size values to avoid spilling sorts to **tempdb**. Next, insert data into the destination clustered index table using a single INSERT...SELECT statement with a MAXDOP value of 1. This MAXDOP hint setting ensures that data pages are placed contiguously within the SQL Server data file. This option is useful if you need more parallelism in the loading process, because multiple load streams into the staging table can be supported at once.

Data Migration

This covers large one-time or infrequent load scenarios in a data warehouse environment. These situations can occur during platform migration or while test data is loaded for system benchmarking. This topic includes load scenarios with one or more of the following attributes:

- Load operations that exceed available system memory
- High-concurrency, high-volume load operations that create pressure on available memory

Heap Table Load Process

Follow the guidance provided for incremental load processing.

Clustered Index Load Process

Two general approaches exist to loading clustered index tables with minimal table fragmentation:

- **Option 1:** Use BULK INSERT to load data directly into a clustered index target table. Sort operations and full commit size should fit in memory for best performance. Care must be taken to ensure that separate batches of data being loaded do not have index key ranges that overlap.
- **Option 2:** Perform a serial or multithreaded bulk insert to an empty clustered index staging table. Next, insert data into an empty clustered index table using a single INSERT...SELECT statement with a MAXDOP value of 1.
- **Option 3:** Use multithreaded bulk inserts to a partition conforming heap staging table, using moderate nonzero batch size values to keep sorts in memory. Next, use serial or parallel INSERT...SELECT statements spanning each partition range to insert data into the clustered index table.
- **Option 4:** Partition switch operations can be used: A set of staging heap tables corresponding to each partition are created and loaded (in parallel) in a volatile filegroup destination database (this prevents the temporary heap tables from fragmenting the destination filegroup). Then create clustered indexes on the nonvolatile destination filegroup for each heap (configuring parallelism as needed). After an index is built, use partition switch operations to instantly populate the destination table. Be sure to use the SORT_IN_TEMPDB option whenever performing a CREATE INDEX operation to avoid fragmenting the destination filegroup.

Parallelism can be introduced in the staging phase of the CI load. However, parallelism in the final insert to the destination CI table introduces some level of fragmentation. There is therefore a trade-off between load performance and scan and query performance, which needs to be considered.

Examples

For more information about multithreaded bulk inserts, see [SSIS Parallel Bulk Load Examples](#) in the appendix.

For more information about data loading and optimizations, see [The SQL Server 2008 Data Loading Performance Guide](http://msdn.microsoft.com/en-us/library/dd425070.aspx) (<http://msdn.microsoft.com/en-us/library/dd425070.aspx>).

Benchmarking and Validation

You can use the metrics described in this section to evaluate resource balance for the SQL Server application and component architecture. These metrics are categorized by workload definition and component evaluation (both hardware and application).

Component Evaluation

The primary goal of Fast Track component evaluation is to determine how much hardware component bandwidth to provision the overall component stack so that CPU capacity is fully realized.

Building Out a New Fast Track Data Warehouse Reference Architecture

Evaluation of the database component architecture starts with a standardized SQL Server data processing rate for a specific server and CPU referred to as Maximum CPU Core Consumption Rate (MCR). MCR is a measure of how much (page-compressed) data can be consumed by a single CPU core, from a table loaded residing in memory, running a standard query. MCR for the FTDW reference configurations in this document was found to be **200 MB per second per core** of page-compressed data (a test dataset exhibited a 2:1 compression ratio).

The MCR calculation provides a starting point for component hardware selection, prior to platform testing and validation. You can use it as a proxy for similar CPU or server configurations. For example, an AMD Opteron two-socket, 8-core server from the reference architecture list provided in this document can be expected to manage a maximum throughput of 1600 MB per second of page-compressed data. Similar to the stated Miles Per Gallon rating on a new car, actual consumption rate will vary to some degree with a real workload.

This number can be used to build out the storage and interconnect infrastructure. Based on server-specific MCR, a rated component architecture diagram can be constructed. At this stage of evaluation, all components show only vendor-rated bandwidth. Figure 4 illustrates an example of this.

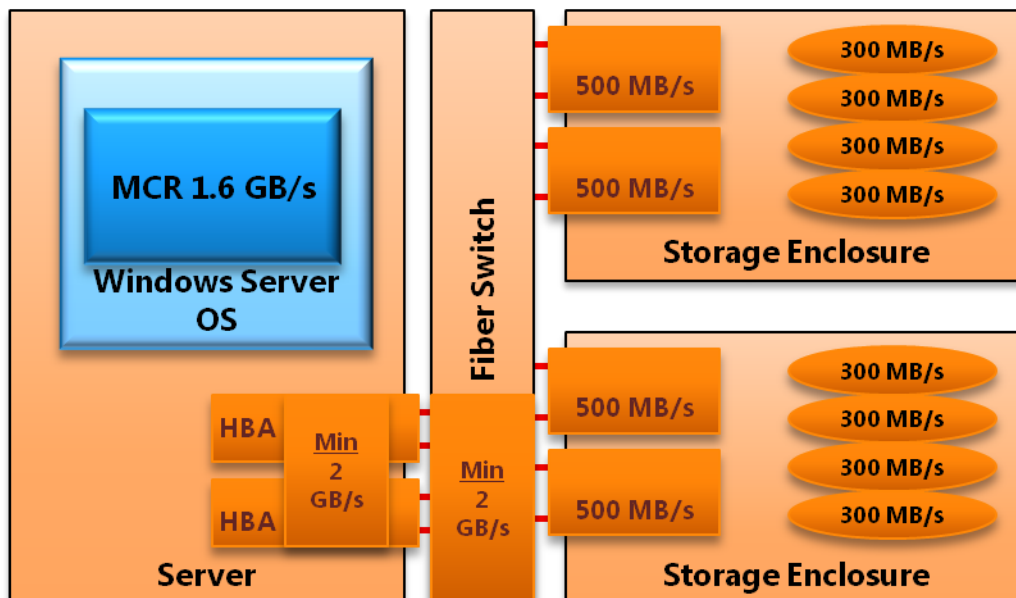


Figure 4: Example of *Maximum CPU Consumption Rate (MCR)* and rated component bandwidth for a 2 socket, 8 core server, with 4x 8Gbps FC HBA ports

MCR is not definitive of actual results for a particular workload. It is used as a general guide for hardware selection prior to component and application testing.

For more information about measuring MCR, see “Workload Testing” in the appendix.

System Component Validation

After you acquire a FTDW-based system, validation of system configuration and performance of the server is an important first step. This section outlines some guidance for carrying out synthetic testing before you deploy SQL Server on a FTDW system and query testing after you deploy SQL Server.

Synthetic I/O Benchmarking

The goal of this phase of component evaluation is to establish real, rather than rated, bandwidth ratings for the key hardware components of the Fast Track reference architecture. This testing is done at the operating system level with a tool such as SQLIO. SQL Server application testing is not done in this phase, and all tests are synthetic, best-case scenarios. In addition, this methodology can provide validation that the database component stack is correctly configured.

Windows Server Performance and Reliability Monitor (Perfmon) can be used to track, record, and report on I/O performance. A tool such as SQLIO can be used to test I/O bandwidth. For more information about SQLIO, including instructions and download locations, see the SQLCAT white paper [Predeployment I/O Best Practices](http://sqlcat.com/whitepapers/archive/2007/11/21/predeployment-i-o-best-practices.aspx).
(<http://sqlcat.com/whitepapers/archive/2007/11/21/predeployment-i-o-best-practices.aspx>).

A block size of 512 KB should be used to simulate the largest I/Os that SQL Server can issue.

The following components and validation processes are used to generate baseline hardware benchmarks.

Step 1 - Validate Fibre Channel Bandwidth

The first step in validating a Fast Track configuration is to determine whether the aggregate throughput that the connectivity of the configuration can provide can be realized. This involves removing disk as a bottleneck and focusing on the nondisk components (that is, HBAs, switch infrastructure, and array controllers). Use the following steps to perform this task using SQLIO:

1. Generate a small data file on each LUN to be used for database files. These files should be sized such that all data files will fit into the read cache on the array controllers (for example, 50 MB per file).
2. Use SQLIO to issue sequential reads against the file simultaneously using large block I/O sizes (512K) and a single thread per file.
3. Start with a relatively low value for outstanding I/Os (-o) and repeat tests increasing this value until there is no further gain in aggregate throughput.

The goal of this test is to achieve aggregate throughput that is reasonable compared with the theoretical limits of the components in the path between the server and storage. This test validates the bandwidth between the server and the SAN storage processors – that is, the MPIO and Fibre Channel paths.

As a reference point, for a configuration with 2 dual-port 8-GB HBA cards connected to 2 SAN arrays (each with 2 ports), a bandwidth of 1.4 to 1.6 GB per second can be expected.

Step 2 - Validate LUN and RAID Pair Bandwidth

This test is similar to the previous tests. However, a larger file is used to remove possible benefits from array cache from controller cache. These test files should be large enough to simulate the target database file size per LUN, for example, 25 GB per LUN. Similar parameters should be used for SQLIO.

Large block (512 KB) sequential reads should be issued against the test files on each LUN. We recommend that you use a single thread per file with an outstanding request depth somewhere between 4 and 16 (start small and increase until maximum throughput is achieved). First test each LUN individually and then test the two simultaneously. An aggregate RAID pair throughput should demonstrate dual read capability from a single 2-disk RAID 1 pair by achieving at least 80 percent of twice the single LUN read rate.

Step 3 - Validate Aggregate Bandwidth

In this test, sequential reads should be run across all of the available data LUNs concurrently against the same files used in step 2. SQLIO should be run using a single thread per test file, with an I/O size of 512K and a number of outstanding I/Os as determined by the previous test.

The results of this test illustrate the maximum aggregate throughput achievable when reading data from the physical disks.

Data is read from the large data file, as in the previous test, on each LUN simultaneously. This implies two threads per RAID 1 disk pair for all data LUNs.

Aggregate performance from disk should be in the region of 80 percent to 90 percent of the aggregate Fibre Channel bandwidth, for balanced FTDW systems.

Component Ratings

The following diagram illustrates synthetic benchmark results that are generally consistent with results seen from similar conforming Fast Track reference architectures (for more information, see the “Fast Track Data Warehouse Reference Configurations” section of this document).

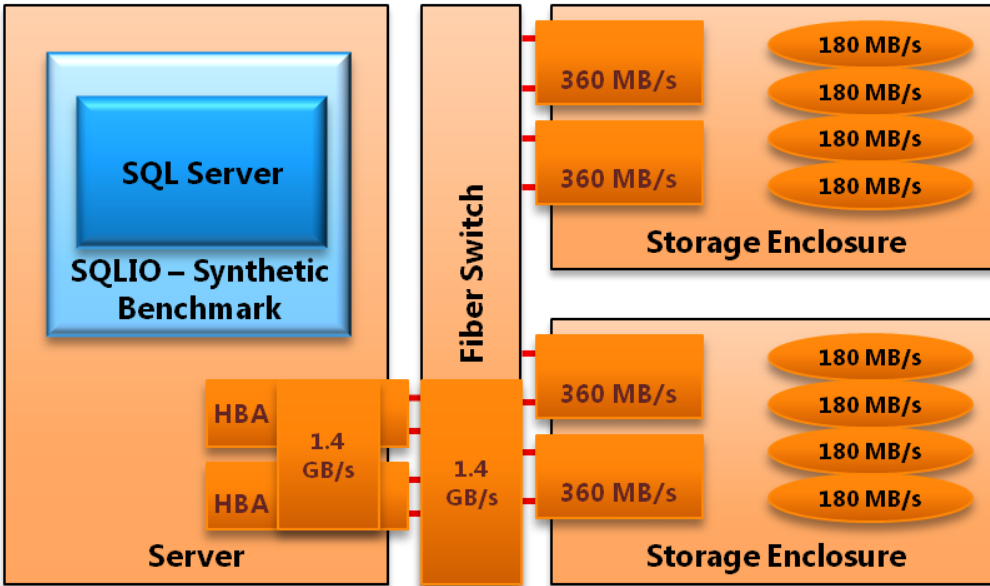


Figure 5: Example of synthetic benchmark realized bandwidth for a 2-socket, 8-core server with 2 8Gbps dual-port HBA cards, using 4 pairs of disks in RAID 1 per array for primary data space

Summary

Synthetic benchmarking validates the actual bandwidth capability for the key hardware components of the database stack. This is done with a series of best-case synthetic tests executed through a tool like SQLIO.

SQL Server Application Benchmarking

The final step in a Fast Track component evaluation brings the storage components of the database stack into the evaluation. This benchmark measures the ability of the SQL Server database to process data from disk. Referred to as Benchmark Consumption Rate (BCR), this metric is similar to MCR in concept. A key difference is that BCR is based on a standard query from the target workload. MCR is measured from cache and represents a maximum theoretical data processing rate, whereas BCR is a measure of the actual throughput for the target workload taking into account the entire stack of components.

BCR

A baseline CPU consumption rate for the SQL Server application is established by running a standard SQL query (or a set of queries) that are specific to your data warehouse workload. This query should be serviced from disk, not from the SQL Server buffer pool or entirely from the SAN array cache. The resulting value is specific to the CPU, the server, and the workload it is being executed against. Use the following method to calculate BCR:

1. Create a reference dataset that contains at least one table. The table should be of significant size that it is not entirely cached in either the SQL Server buffer pool cache or in the SAN array cache. In absence of customer data, a synthetic dataset can be used. It is important to attempt to approximate the expected characteristics of the data for the targeted use case.

2. Pick a simple query containing a simple aggregation and group by statement. The basic query form is as follows: *SELECT sum([integer field]) FROM [table] WHERE [restriction] GROUP BY [col]*. The chosen query (or queries) should:
 - Represent average target workload requirements. This may imply increasing or decreasing the complexity of the basic query form, adding joins and or discarding more or less data through projection and restriction.
 - Not cause writes of data to **tempdb**.
 - Return minimal rows.
3. The environment should:
 - Ensure Resource Governor settings are set at default.
 - Ensure caches are cleared before the query is run, using `DBCC dropcleanbuffers`.
 - Set STATISTICS IO and STATISTICS TIME to ON to report extended query metrics.
4. Run the query multiple times, starting at MAXDOP 1. Each time you run the query, increase the MAXDOP setting for the query, clearing caches between each run.
 - Record the number of logical reads and CPU time from the statistics output.
 - Calculate the BCR in MB/s using the formula:
$$([\text{Logical reads}] / [\text{CPU time}]) * 8\text{KB} / 1024$$
 - This gives you a range for BCR. If multiple queries are used, use a weighted average to determine BCR.

Component Ratings

Figure 6 illustrates SQL Server workload based benchmark results that are generally consistent with results seen from similar Fast Track Data Warehouse reference configurations. (For more information, see the [Fast Track Data Warehouse Reference Configurations](#) section of this document.)

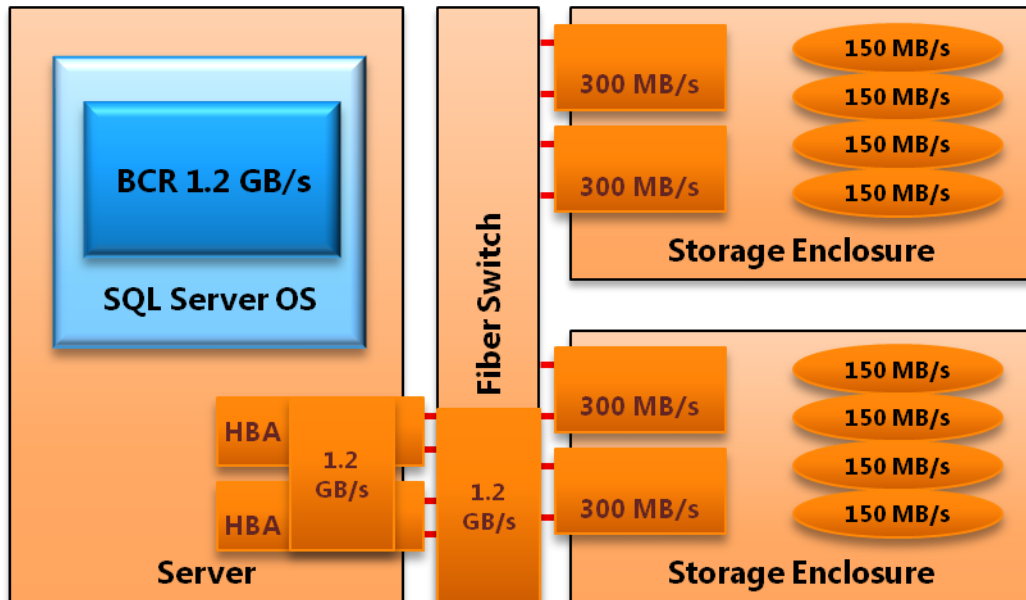


Figure 6: Example of application benchmark realized bandwidth for a 2-socket, 8-core server, using 4 pairs of disks in RAID 1, per array (that is, a total of 16 physical disks)

Interpreting BCR

If your BCR for the average query is much lower than the MCR (that is, 200 MB/s per core), you are likely to be CPU bound. In response, you might think about reducing the storage throughput, for example by reducing the number of arrays, introducing more disks per array, increasing the size of the disks – these will reduce the cost of the storage infrastructure to a balanced level. Alternatively you might think about using a higher number or higher performance CPUs that will increase your BCR. Doing so will result in a system balanced for that workload.

Correspondingly, if your BCR is higher than the MCR, you may need more I/O throughput to service your requirement in a balanced fashion.

Fast Track Data Warehouse Reference Configurations

The following hardware specifications are examples of component architectures that conform to the principles of Fast Track Data Warehouse reference architectures for data warehousing, outlined in this document.

The Maximum CPU consumption rate (MCR) for both AMD and Intel CPUs listed in these specifications is approximately **200 MB per second per core** of page compressed data.

Server	CPU	CPU Cores	SAN	Data Drive Count	Initial Capacity *	Max Capacity **
HP Proliant DL 385 G6	(2) AMD Opteron Istanbul six core 2.6 GHz	12	(3) HP MSA2312	(24) 300GB 15k SAS	6TB	12TB
HP Proliant DL 385 G6	(2) AMD Opteron Istanbul six core 2.6 GHz	12	(3) EMC AX4	(24) 300GB 15k FC	6TB	12TB
HP Proliant DL 585 G6	(4) AMD Opteron Istanbul six core 2.6 GHz	24	(6) HP MSA2312	(48) 300GB 15k SAS	12TB	24TB
HP Proliant DL 585 G6	(4) AMD Opteron Istanbul six core 2.6 GHz	24	(6) EMC AX4	(48) 300GB 15k FC	12TB	24TB
HP Proliant DL 785 G6	(8) AMD Opteron Istanbul six core 2.8 GHz	48	(12) HP MSA2312	(96) 300GB 15k SAS	24TB	48TB
HP Proliant DL 785 G6	(8) AMD Opteron Istanbul six core 2.8 GHz	48	(12) EMC AX4	(96) 300GB 15k FC	24TB	48TB
Dell PowerEdge R710	(2) Intel Xeon Nehalem quad core 2.66 GHz	8	(2) EMC AX4	(16) 300GB 15k FC	4TB	8TB
Dell Power Edge R900	(4) Intel Xeon Dunnington six core 2.67GHz	24	(6) EMC AX4	(48) 300GB 15k FC	12TB	24TB
IBM X3650 M2	(2) Intel Xeon Nehalem quad core 2.67 GHz	8	(2) IBM DS3400	(16) 200GB 15K FC	4TB	8TB
IBM X3850 M2	(4) Intel Xeon Dunnington six core 2.67 GHz	24	(6) IBM DS3400	(24) 300GB 15k FC	12TB	24TB
IBM X3950 M2	(8) Intel Xeon Dunnington four core 2.13 GHz	32	(8) IBM DS3400	(32) 300GB 15k SAS	16TB	32TB

* Core-balanced compressed capacity based on 300GB 15k SAS not including hot spares and log drives. Assumes 25% (of raw disk space) allocated for **tempdb**.

** Represents storage array fully populated with 300GB15k SAS and use of 2.5:1 compression ratio. This includes the addition of one storage expansion tray per enclosure.

Capacity figures do not include **tempdb** or log space allocations; they relate solely to space reserved for data files.

An assumption was made in calculating the capacity of the FTDW reference configurations resulting in approximately 288 GB of disk space allocated per SAN array. Requirements for **tempdb** vary significantly among implementations. For more information about **tempdb** sizing see [Capacity Planning for tempdb](http://msdn.microsoft.com/en-us/library/ms345368.aspx) in SQL Server Books Online (<http://msdn.microsoft.com/en-us/library/ms345368.aspx>).

Approximately 272 GB of available log space is available per SAN array.

Conclusion

SQL Server Fast Track Data Warehouse offers a template and tools for bringing a data warehouse from design to deployment. This document describes the methodology, configuration options, best practices, reference configurations, and benchmarking and validation techniques for Fast Track Data Warehouse.

For more information:

<http://www.microsoft.com/sqlserver/>: SQL Server Web site

<http://technet.microsoft.com/en-us/sqlserver/>: SQL Server TechCenter

<http://msdn.microsoft.com/en-us/sqlserver/>: SQL Server DevCenter

[SQL Server CAT Team 10 Best Practices for Building Large Scale Relational Data Warehouses](#)

[How to: Enable the Lock Pages in Memory Option](#)

[Tuning options for SQL Server 2005 that is running in high performance workloads](#)

[How to: Configure SQL Server to Use Soft-NUMA](#)

[Database File Initialization](#)

[How to: View or Change the Recovery Model of a Database \(SQL Server Management Studio\)](#)

[Monitoring Memory Usage](#)

[Scalability and VLDB Resources on Microsoft.com](#)

[824190 Troubleshooting Storage Area Network \(SAN\) Issues](#)

[325590 How to Use Diskpart.exe to Extend a Data Volume](#)

[Microsoft Storage Technologies – Multipath I/O](#)

[Support in Windows Server 2003: Improving Manageability and Performance in Hardware RAID and Storage Area Networks](#)

[INF: Support for Network Database Files](#)

[SQL Server 2000 I/O Basics White Paper](#)

<http://sqlcat.com/whitepapers/archive/2009/05/29/data-compression-strategy-capacity-planning-and-best-practices.aspx>

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback.](#)

Appendix

FTDW CPU Core Calculator

A spreadsheet calculator has been developed to assist with the calculation of the estimated number of CPU core necessary to support an estimated workload. This can be used in the absence of a test platform or as a starting point when evaluating workload requirements.

Follow these steps to enter into the spreadsheet to support its calculations:

1. Determine the MCR rate in megabytes per second for the server you intend to deploy. This can be published within an existing FTDW specification, or it can be benchmarked in accordance with methodology provided within this document. A proxy value can also be used, based on previously published values for similar server and CPU configurations (this will be a best-guess estimate).
2. Specify the estimated or actual number of active concurrent sessions that this configuration will be supporting.
3. Based on estimated user workload, determine the amount of data scanned (in megabytes) in the system's simple, average, and complex queries.
4. For each workload type, specify as a percentage to the total the amount of each workload type that the system will support.
5. Specify the desired target response time in seconds that this amount of data should be scanned (in megabytes) for each of the query types.

Example of Calculating CPU Core Requirement for a Given Workload

This example makes the following assumptions:

- This is a Fast Track configuration using one of the conforming Fast Track Data Warehouse configurations listed in this document. Note that all of these servers and CPU combinations have been determined to have a target CPU consumption rate of compressed data at 200 megabytes per second per core.
- The total number of users expected on the system is 3,000 users. The rule of thumb used to estimate the active concurrent user count is 1 percent of the anticipated total user count to estimate the concurrent user count, and 1 percent of the estimated concurrent user count to estimate the number of concurrent queries.
- The workload is based on simple/average/complex queries with a 70/20/10 respective percentage mix.
- Estimated query scan sizes for the workload above are:
 - Simple queries scan approximately 8,000 MB, and desired response time is 25 seconds (under load).
 - Average queries scan approximately 75,000 MB, and desired response time is 180 seconds (under load).

- Complex queries scan approximately 450,000 MB, and desired response time is 1200 seconds (under load).

Based on these assumptions, the CPU core calculator estimates the number of cores to support this workload to be approximately 32. Figure 7 shows the calculator.

Fast Track SMP RA for SQL Server 2008 CPU Core Calculator v2.4

Updated 10/09/2009 - uw

This spreadsheet can be used to estimate the number of cores required to support a user workload and workload mix.

Enter your factors into the green fields and the results will be calculated in the pink cells.

The spreadsheet uses a weighted average to determine the number of cores required based on your inputs.

User Variable Input

Anticipated total number of users expected on the system	3,000	users	Adjust for workload mix		Estimated % of workload		Estimated % data found in SQL Server cache		Estimated Query Data Scan Volume MB (Uncompressed)		Desired Query Response Time (seconds) (under load)		Estimated Disk Scan volume MB (Uncompressed)	
Estimated percent of actual query concurrency	1%	concurrency	Simple		70%	10%	8,000		25		7,200			
Fast Track DW CPU max core consumption rate (MCR) in MB/s of page compressed data per core	200	MB/s	Average		20%	0%	75,000		180		75,000			
Estimated compression ratio (default = 2.5:1)	2.5	:1	Complex		10%	0%	450,000		1,200		450,000			
Estimated drive serial throughput speed in compressed MB/s	100	MB/s			100%									
Number of data drives in single storage array	8	drives												
Usable capacity per drive	272	GB												
Space Reserved for TempDB	26%													

Calculations and Results

	% of core consumption rate achieved	Expected per CPU core consumption rate (MB/s)	Calculated Single Query Scan Volume in MB (compressed)	Calculated Target Concurrent Queries	Estimated Target Queries per Hour	Required IO Throughput in MB/s	Estimated Number of Cores Required	Estimated Single Query Run Time (seconds)
Simple	100%	200	2,880	21	3,024	2,419	12.10	0.5
Average	50%	100	30,000	6	120	1,000	10.00	9.4
Complex	25%	50	180,000	3	9	450	9.00	112.5
				30	3,153	3,869	32.00	

Arrays Required based on throughput	Single Array Throughput in MB/s	Throughput in MB/s for All Required Arrays
5	800	4,000

Suggested Fast Track RA Server Requirements

No of CPU cores	Number of arrays	Total Compressed Data Capacity (TB)	Max achievable IO Throughput in MB/s	Max achievable CPU consumption in MB/s	Required IO Throughput in MB/s
32	8	16	6,400	6,400	3,869

Figure 7: FTDW CPU Core Calculator

Integration Services Parallel Bulk Load Examples

Integration Services Scripts for Parallel Bulk Data Loading

The following stored procedures were created and invoked in SQL Server Integration Services to load data into a test schema.

The staging table was created as HEAP with 16 partitions. The following stored procedure shows the logic used to control how data was loaded into each partition. Note that the BATCHSIZE value was calculated by taking the row size and dividing it into the available memory per thread. TABLOCK was used to ensure minimal logging for the load.

Source files were broken out by partition and were bulk inserted by partition. The Integration Services package contained 16 tasks (one per partition) that were run in parallel on each NUMA node and called the stored procedure to populate a corresponding partition.

```
USE [stage]
GO
Create procedure [dbo].[spM_Load_DW_VEH_DRVR_PREM_DT]
@PartitionNumber int
as
declare @e int,@ExecSQL nvarchar(1000)

INSERT INTO [msdb].[dbo].[LoadSummary]
    ([JobName]
    ,[Partition]
    ,[BeginTime])
VALUES
    ('Load_DW_VEH_DRVR_PREM_DT',
    @PartitionNumber,
    GETDATE())

Set @execSQL='BULK INSERT STAGE.PWRD200102.DW_VEH_DRVR_PREM_DT_partition from
"D:\bcp\DW_VEH_DRVR_PREM_DT_'+CAST(@PartitionNumber as varchar(3))+'.txt" WITH (DATAFILETYPE = "char",
FIELDTERMINATOR = "|",
ROWTERMINATOR = "\n",
MAXERRORS=1000,
BATCHSIZE=100000,
TABLOCK);
'

exec @e=sp_executesql @execSQL

update msdb.dbo.LoadSummary
set Endtime=GETDATE(),
    ResultCode=@e
    where JobName='Load_DW_VEH_DRVR_PREM_DT'
    and Partition=@PartitionNumber
GO
```

Integration Services Scripts for Parallel INSERT...SELECT to Final Table

The following stored procedures were created and invoked in Integration Services to load the data from the staging table into the permanent data schema.

The following stored procedure shows logic to insert and select each partition from the StageDB HEAP into the final clustered index table. Note that trace flag 610 is turned on globally to perform minimal logging on the insert to the clustered table and that a MAXDOP value of 1 is used to minimize fragmentation.

```
USE [PM]
GO
create procedure [dbo].[spM_INSERT_DW_VEH_DRVR_PREM_DT]
@PartitionNumber int
as
declare @e int

INSERT INTO [msdb].[dbo].[LoadSummary]
    ([JobName]
    ,[Partition]
    ,[BeginTime])
VALUES
    ('Insert_DW_VEH_DRVR_PREM_DT',
    @PartitionNumber,
    GETDATE())
```

```
insert into STAGE.PWRD200102.DW_VEH_DRVR_PREM_DT select * from STAGE.PWRD200102.DW_VEH_DRVR_PREM_DT_partition
where $PARTITION.pf_DW_VEH_DRVR_PREM_DT(FK_POL_N) = @PartitionNumber option (maxdop 1);

set @e=@@error
update msdb.dbo.LoadSummary
set Endtime=GETDATE(),
    ResultCode=@e
where JobName='Insert_DW_VEH_DRVR_PREM_DT'
and Partition=@PartitionNumber
GO
```

The Integration Services package contained 16 tasks that were run in parallel on each NUMA node and called the stored procedure to populate a corresponding partition.

Validating a Fast Track Reference Architecture

Synthetic I/O Testing

SQLIO is a tool available for download from Microsoft that enables testing of the I/O subsystem independent of SQL Server. For more information about using SQLIO, including general predeployment advice, see the [SQL Server Pre-deployment I/O Best Practices](http://technet.microsoft.com/en-gb/library/cc966412.aspx) white paper (<http://technet.microsoft.com/en-gb/library/cc966412.aspx>).

Generating Test Files with SQLIO

Running SQLIO causes an appropriate file to be created, if it is not already present. To generate a file of a specific size, use the `-F` parameter. For example using a parameter file (param.txt) containing:

```
C:\stor\pri\1\iobw.tst 1 0x0 50
```

Running SQLIO with the `-F` parameter generates a 50 MB file on first execution:

```
Eq sqlio -kW -s60 -fsequential -o1 -b64 -LS -Fparam.txt
```

This process can take some time for large files. Create one file on each data disk on which you will host SQL Server data and **tempdb** files. This can be achieved by adding more lines to the parameter file, which will create the required files one by one. To achieve parallel file creation, create multiple parameter files and run multiple SQLIO sessions concurrently.

Testing with SQLIO

Use of SQLIO is described more completely in the best practices article. Read tests generally take the form:

```
sqlio -kR -fSequential -s30 -o4 -b512 d:\iobw.tst
```

where R indicates a read test, 30 is the test duration in seconds, 4 is the number of outstanding requests issued, 512 is the block size in kilobytes of requests made, and d:\iobw.tst is the location of the test file.

In order to test aggregate bandwidth scenarios, multiple SQLIO tests must be issued in parallel. This can be achieved using Windows PowerShell™ scripts or other scripting methods.

The predeployment best practices article also covers how to track your tests using Windows Server Performance and Reliability Monitor. Recording and storing the results of these tests will give you a baseline for future performance analysis and issue resolution.

Validate Fibre Channel Bandwidth (from cache)

Using a small test file and a duration of several minutes ensures that the file is completely resident in the array cache. If you follow the Fast Track recommendations, there will be 8 files per array. Figure 7 shows the Logical Disk > Read Bytes / sec counter against the disks in an example Fast Track system at various outstanding request numbers and block size. Tests should run for at least a few minutes to ensure consistent performance. The figure shows that optimal performance requires an outstanding request queue of at least 4 requests per file. Each individual disk should contribute to the total bandwidth.

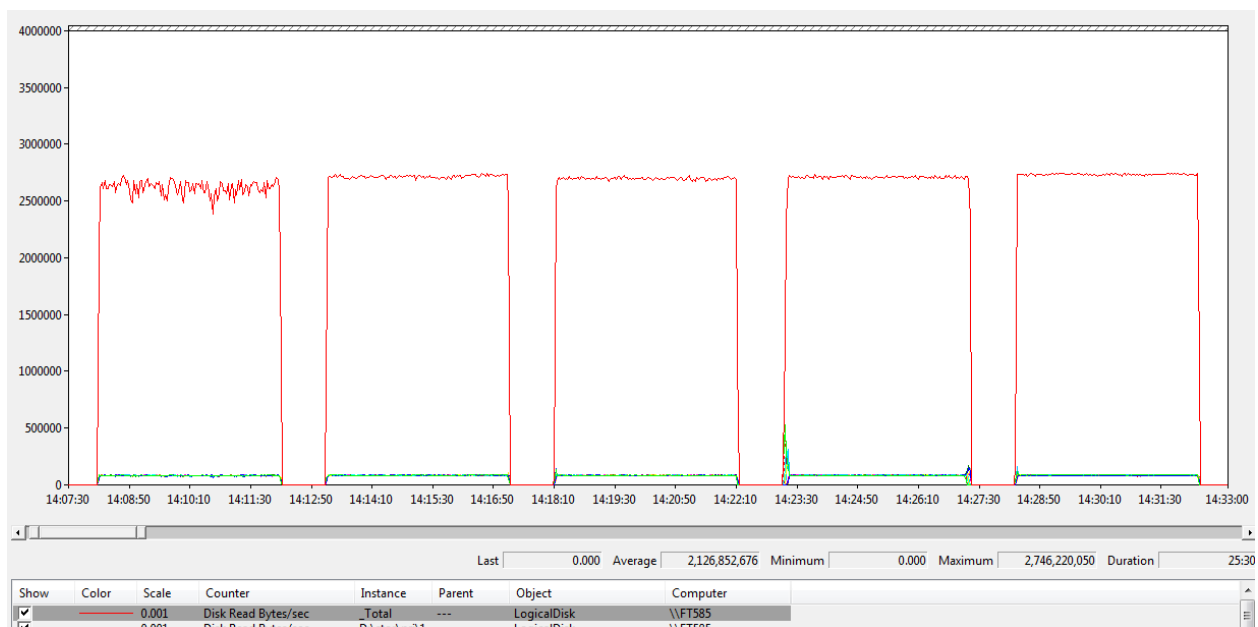


Figure 8: Logical Disk > Read Bytes / sec counter

As a reference point, for a configuration with 2 dual-port 8 GB HBA cards connected to 2 SAN arrays (each with 2 ports), a bandwidth of 1.4 to 1.6 GB per second can be expected. For a balanced FTDW configuration, the total aggregate bandwidth should scale roughly in line with the number of arrays and CPU cores.

Validate LUN and RAID Pair Bandwidth (from Disk)

These tests ensure all disk volumes presented by the disk arrays to Windows are capable of contributing to the overall aggregate bandwidth, by reading from each LUN, one at a time. Using the recommendation of 2 disks per RAID 1 disk pair, you will see that some of the LUNs appear to be slightly faster than others. This is expected due to differences in the position of the LUN on the disk platter. Each LUN should be able to service a bandwidth of between 120 to 150 MB per second.

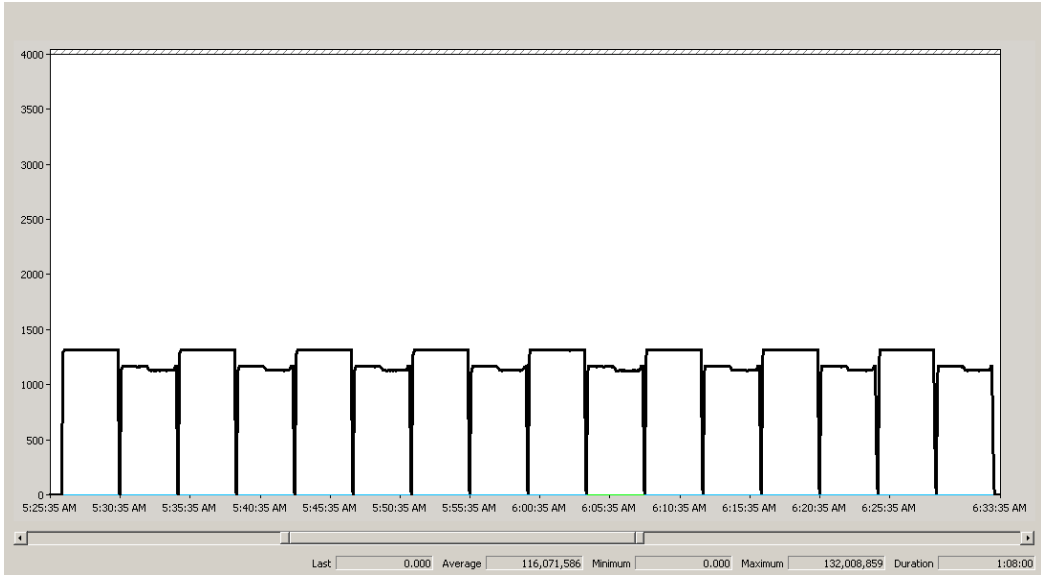


Figure 9: Validating LUN and RAID pair bandwidth

If you are able to map disks to individual disk volumes on the disk arrays, run simultaneous tests against both LUNs that share the same RAID 1 disk pair. Each disk pair should be able to service an aggregate bandwidth equal to the bandwidth available from each LUN. If you are unable to map disks presented to windows to individual arrays, you can show for each disk that all other disks are able to service their full bandwidth concurrently with that disk. The picture shows the output of tests against 8 RAID pairs.

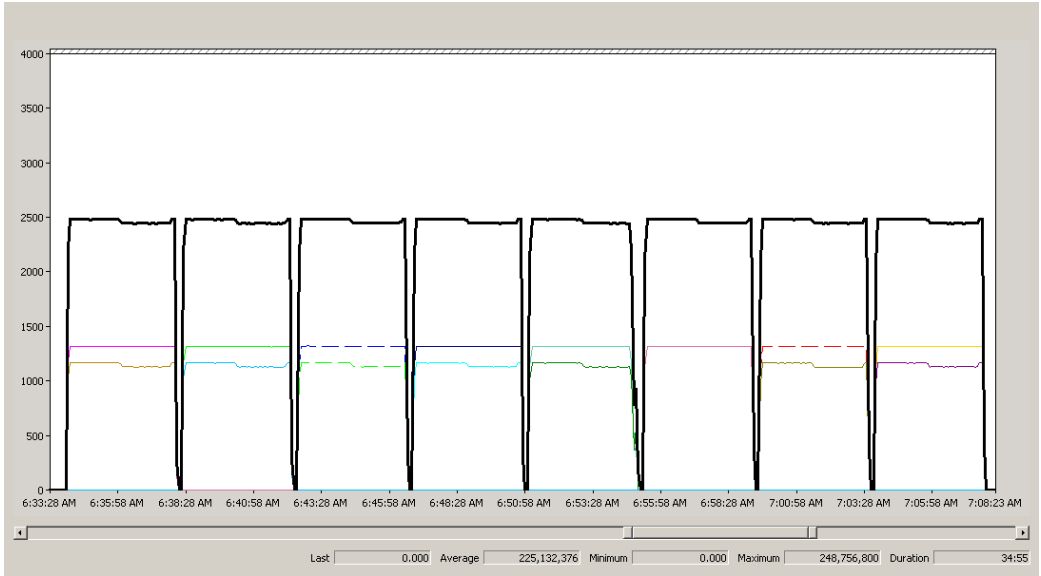


Figure 10: Testing LUNs that share RAID pairs

Validate Aggregate Bandwidth (from Disk)

The following test demonstrates the effect of stepping up the I/O throughput, adding in an additional LUN into the test at regular intervals. As each test runs for a set interval, you see a

step down. You should observe a similar pattern. Peak aggregate bandwidth from disk should approach 80 percent to 90 percent of the bandwidth demonstrated from cache in the first step. The graph shows the test at multiple block sizes 512K and 64K.

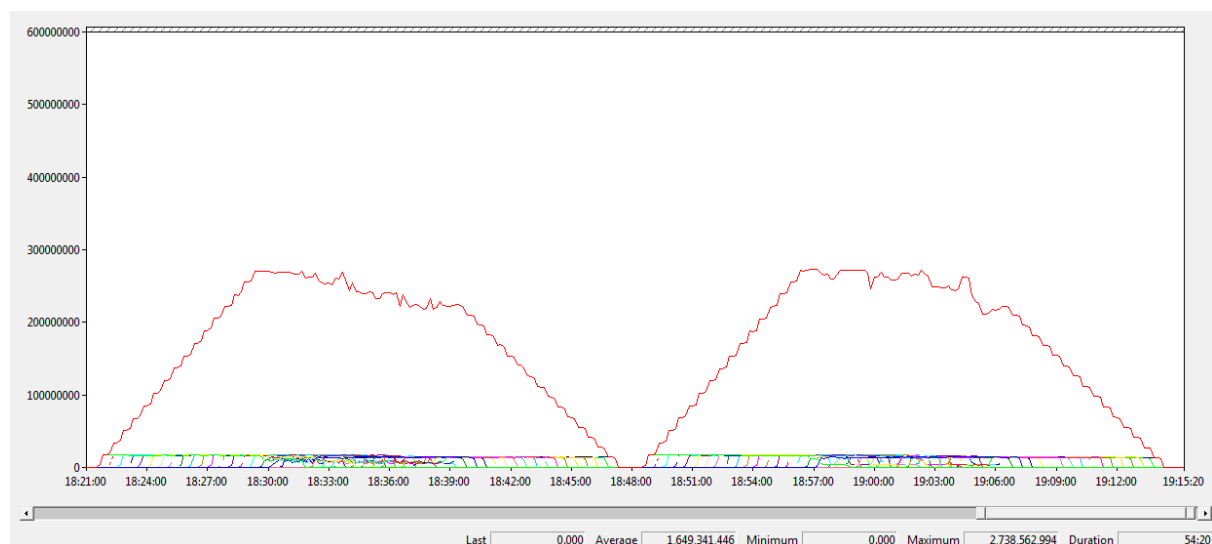


Figure 10: Aggregate bandwidth at multiple block sizes

Workload Testing

Measuring the MCR for Your Server (Optional)

The goal of MCR is to estimate the maximum throughput of a single CPU core, running SQL Server, in absence of I/O bottleneck issues. MCR is evaluated per-core. If you chose to calculate this for your own server, additional details describing the methodology for calculating MCR are provided here:

1. Create a reference dataset that contains at least one table with page compression on. The table should fit within available memory.
 - a. A standard synthetic dataset should be used. TPC-H data for the lineitem table was used to evaluate FTDW reference configurations.
2. Load the table into SQL Server buffer cache. For example, run a `SELECT COUNT(*)` from the table.
 - a. Check that there is no disk activity when queries are run against the table.
 - b. You may need to ensure that Resource Governor settings enable enough memory is available to the session to host the table.
3. Execute the test query.
 - a. Run `SET STATISTICS IO ON` for the session to enable session I/O statistics.
 - b. Run `SET STATISTICS TIME ON` for the session to enable session time statistics.
 - c. Execute a query containing a simple aggregation and a `GROUP BY` statement.
 - i. The following query is used to evaluate MCR for FTDW configurations:

```
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE FROM
LINEITEM WHERE L_DISCOUNT BETWEEN 0.04 - 0.01 AND 0.04
+ 0.01 AND L_QUANTITY < 25 OPTION (MAXDOP 1)
```

- ii. The query should utilize 100 percent CPU. It may be necessary to increase query complexity for systems with newer CPUs.
4. Ensure that the query is answered from cache prior to benchmarking by monitoring disk activity, or by using the output from session statistics to measure physical reads.
5. Record the number of logical reads (pages) and CPU duration from the statistics output of the MAXDOP 1 execution. Take an average of multiple readings.
6. Calculate the MCR in MB per second using the following formula:

$$([\text{Logical reads}]/[\text{CPU time}]) * 8\text{KB} / 1024$$

Measuring the BCR for Your Workload

BCR measurement is similar to MCR measurement, except that data is serviced from disk, not from cache. The query and dataset for BCR is representative of your target data warehousing workload.

One approach for BCR is to take a simple query, and average query, and a complex query from the workload. The complex queries should be those that put more demands on the CPU. The simple query should be analogous to the MCR and should do a similar amount of work, so that it is comparable to the MCR.

Creating the Database

Here is an example of a CREATE DATABASE statement for an 8-core Fast Track Data Warehouse system, with 16 data LUNs.

```
CREATE DATABASE [TPCH100GB_COMP] ON
PRIMARY
( NAME = N'TPCH100GB_COMP', FILENAME = N'C:\stor\pri\1\TPCH100GB_COMP.mdf' , SIZE =
50MB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
FILEGROUP TPCH100GB_COMP DEFAULT
( NAME = N'TPCH100GB_COMP_01a', FILENAME = N'C:\stor\pri\1\TPCH100GB_COMP_01a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_02a', FILENAME = N'C:\stor\pri\2\TPCH100GB_COMP_02a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_03a', FILENAME = N'C:\stor\pri\3\TPCH100GB_COMP_03a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_04a', FILENAME = N'C:\stor\pri\4\TPCH100GB_COMP_04a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_05a', FILENAME = N'C:\stor\pri\5\TPCH100GB_COMP_05a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_06a', FILENAME = N'C:\stor\pri\6\TPCH100GB_COMP_06a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_07a', FILENAME = N'C:\stor\pri\7\TPCH100GB_COMP_07a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_08a', FILENAME = N'C:\stor\pri\8\TPCH100GB_COMP_08a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_09a', FILENAME = N'C:\stor\pri\9\TPCH100GB_COMP_09a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_10a', FILENAME = N'C:\stor\pri\10\TPCH100GB_COMP_10a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
( NAME = N'TPCH100GB_COMP_11a', FILENAME = N'C:\stor\pri\11\TPCH100GB_COMP_11a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ) ,
```

```

( NAME = N'TPCH100GB_COMP_12a', FILENAME = N'C:\stor\pri\12\TPCH100GB_COMP_12a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ),
( NAME = N'TPCH100GB_COMP_13a', FILENAME = N'C:\stor\pri\13\TPCH100GB_COMP_13a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ),
( NAME = N'TPCH100GB_COMP_14a', FILENAME = N'C:\stor\pri\14\TPCH100GB_COMP_14a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ),
( NAME = N'TPCH100GB_COMP_15a', FILENAME = N'C:\stor\pri\15\TPCH100GB_COMP_15a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 ),
( NAME = N'TPCH100GB_COMP_16a', FILENAME = N'C:\stor\pri\16\TPCH100GB_COMP_16a.ndf' ,
SIZE = 10GB , MAXSIZE = UNLIMITED, FILEGROWTH = 0 )
LOG ON
( NAME = N'TPCH100GB_COMP_log', FILENAME = N'C:\stor\log\1\TPCH100GB_COMP_log.LDF' ,
SIZE = 20GB , MAXSIZE = 500GB , FILEGROWTH = 10%)
GO

```

```

ALTER DATABASE TPCH100GB_COMP SET RECOVERY SIMPLE
GO

```

Creating the Test Tables

Here is an example CREATE TABLE statement.

```

CREATE TABLE lineitem
( l_orderkey      bigint not null,
  l_partkey       integer not null,
  l_suppkey       integer not null,
  l_linenumbers   integer not null,
  l_quantity      float not null,
  l_extendedprice float not null,
  l_discount      float not null,
  l_tax           float not null,
  l_returnflag    char(1) not null,
  l_linestatus    char(1) not null,
  l_shipdate      datetime not null,
  l_commitdate    datetime not null,
  l_receiptdate   datetime not null,
  l_shipinstruct  char(25) not null,
  l_shipmode      char(10) not null,
  l_comment       varchar(132) not null
)
ON TPCH100GB_COMP
GO

```

```

CREATE CLUSTERED INDEX cidx_lineitem
ON lineitem(l_shipdate ASC)
WITH( SORT_IN_TEMPDB = ON
      , DATA_COMPRESSION = PAGE
)
ON TPCH100GB_COMP
GO

```

Loading Data for BCR Measurement

As described elsewhere in this document, Fast Track Data Warehouse systems are sensitive to the fragmentation of database files. Use one of the techniques this document describes to load data. During FTDW testing, the clustered index load method described as option 2 was used.

Using the TPC-H datagen tool, lineitem table data was generated to a size of 70 GB, using options -s100, generating the file in 8 parts, and using the -S and -C options.

Trace flag 610 was set during all load operations to use minimal logging where possible.

Using BULK INSERT, this data was inserted in parallel into a single clustered index staging table, using minimal logging; we chose a block size that would not overwhelm available memory and that would reduce spillage to disk. Disabling page locks and lock escalation on the staging table improved performance during this phase.

A final insert was performed into an identical target table, with MAXDOP 1 (using the TABLOCK hint) and avoiding a sort.

Running Queries for BCR Measurement

Run queries using the method outlined in the Benchmarking and Validation section.

Use the SQL Server Profiler tool to record relevant information for query benchmarks. SQL Server Profiler should be set up to record logical reads, CPU, duration, database name, schema name, the SQL statement, and the actual query plans. Alternatively the statistics session parameters `set statistics io on` and `set statistics time on` can be used.

Here are a few example queries (based on queries from the TPC-H benchmark) and the BCR achieved on reference systems.

Query Complexity	Per-core BCR (Page Compressed) at MAXDOP 1
Simple	201 MB/s
Average	83 MB/s
Complex	56 MB/s

Simple

```
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_discount between 0.04 - 0.01 and 0.04 + 0.01 and
    l_quantity < 25
option (maxdop 1)
```

Average

```
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
```

```

    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count_big(*) as count_order
from
    lineitem
where
    l_shipdate <= dateadd(dd, -90, '1998-12-01')
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus
option (maxdop 1)

```

Complex

```

select
    100.00 * sum(case
        when p_type like 'PROMO%'
        then l_extendedprice*(1-l_discount)
        else 0
        end) / sum(l_extendedprice * (1 - l_discount)) as
    promo_revenue
from
    lineitem,
    part
where
    l_partkey = p_partkey
    and l_shipdate >= '1995-09-01'
    and l_shipdate < dateadd(mm, 1, '1995-09-01')
option (maxdop 1)

```

Factors Affecting Query Consumption Rate

Not all queries will achieve the Maximum CPU Consumption Rate (MCR) or Benchmark Consumption Rate (BCR). There are many factors that can affect the consumption rate for a query. Queries simpler than the workload used to generate the consumption rate will have higher consumption rates, and more complex workloads will have lower consumption rates. Many factors that can affect this complexity and the consumption rate, for example:

- **Query complexity:** The more CPU intensive the query is, for example, in terms of calculations and the number of aggregations, the lower the consumption rate.

- **Sort complexity:** Sorts from explicit order by operations or group by operations will generate more CPU workload and decrease consumption rate. Additional writes to **tempdb** caused by such queries spilling to disk negatively affect consumption rate.
- **Query plan complexity:** The more complex a query plan, the more steps and operators, the lower the CPU consumption rate will be, because each unit of data is processed through a longer pipeline of operations.
- **Compression:** Compression will decrease the consumption rate of data in real terms, because consumption rate by definition is measured for queries that are CPU bound, and decompression consumes CPU cycles. However, the increased throughput benefits usually outweigh additional CPU overhead involved in compression, unless the workload is highly CPU intensive. When comparing consumption rates for compressed and uncompressed data, take the compression factor into account. Another way of looking at this is to think of the consumption rate in terms of rows per second.
- **Data utilization:** Discarding data during scans (for example, through query projection and selection) is quite an efficient process. Queries that use all the data in a table have lower consumption rates, because more data is processed per unit data throughput.