

**Fast Track Data Warehouse Reference Guide for SQL Server 2012**

SQL Server Technical Article

**Writers:** Eric Kraemer, Mike Bassett, Eric Lemoine, Dave Withers

**Technical Reviewers:** Claude Lorenson, Susan Price, Ralph Kemperdick, Henk van der Valk, Alexi Khalyako, Oliver Chiu

**Published:** March 2012

**Applies to:** SQL Server 2012

**Summary:** This paper defines a reference configuration model (known as Fast Track Data Warehouse) using an resource balanced approach to implementing a symmetric multiprocessor (SMP)-based SQL Server database system architecture with proven performance and scalability for data warehouse workloads. The goal of a Fast Track Data Warehouse reference architecture is to achieve an efficient resource balance between SQL Server data processing capability and realized component hardware throughput.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.

Contents

[FTDW Change History 6](#_Toc320098899)

[Introduction 6](#_Toc320098900)

[Audience 6](#_Toc320098901)

[Fast Track Data Warehouse 6](#_Toc320098902)

[Fast Track 7](#_Toc320098903)

[Value Proposition 7](#_Toc320098904)

[Methodology 7](#_Toc320098905)

[Holistic Component Architecture 7](#_Toc320098906)

[Workload Optimized Approach 8](#_Toc320098907)

[Validated SQL Server Fast Track Reference Configurations 9](#_Toc320098908)

[Summary 9](#_Toc320098909)

[FTDW Workload 9](#_Toc320098910)

[Data Warehouse Workload Patterns 9](#_Toc320098911)

[Workload Evaluation 10](#_Toc320098912)

[Qualitative Data Warehouse Workload Attributes 11](#_Toc320098913)

[Choosing a FTDW Reference Configuration 12](#_Toc320098914)

[Option 1: Basic Evaluation 13](#_Toc320098915)

[Step 1: Evaluate the Customer Use Case 13](#_Toc320098916)

[Step 2: Choose a Published FTDW Reference Architecture 14](#_Toc320098917)

[Option 2: Full Evaluation 14](#_Toc320098918)

[Process Overview 14](#_Toc320098919)

[Step 1: Evaluate the Customer Use Case 15](#_Toc320098920)

[Step 2: Establish Evaluation Metrics 15](#_Toc320098921)

[Step 3: Choose a Fast Track Data Warehouse Reference Architecture 16](#_Toc320098922)

[Option 3: User-Defined Reference Architectures 16](#_Toc320098923)

[Step 1: Define Workload 16](#_Toc320098924)

[Step 2: Establish Component Architecture Benchmarks 16](#_Toc320098925)

[Choosing an FTRA Summary 17](#_Toc320098926)

[FTDW Standard Configuration 18](#_Toc320098927)

[Hardware Component Architecture 18](#_Toc320098928)

[Component Requirements and Configuration 18](#_Toc320098929)

[Application Configuration 20](#_Toc320098930)

[Windows Server 2008 R2 20](#_Toc320098931)

[SQL Server 2012 Enterprise 20](#_Toc320098932)

[Storage System 22](#_Toc320098933)

[SQL Server Best Practices for FTDW 26](#_Toc320098934)

[Data Architecture 26](#_Toc320098935)

[Table Structure 26](#_Toc320098936)

[Table Partitioning 27](#_Toc320098937)

[Indexing 28](#_Toc320098938)

[xVelocity In-Memory Columnstore Indexes 28](#_Toc320098939)

[Database Statistics 30](#_Toc320098940)

[Compression 30](#_Toc320098941)

[Managing Data Fragmentation 31](#_Toc320098942)

[File System Fragmentation 31](#_Toc320098943)

[Multiple Filegroups 33](#_Toc320098944)

[Loading Data 33](#_Toc320098945)

[Incremental Loads 34](#_Toc320098946)

[Data Migration 35](#_Toc320098947)

[Benchmarking and Validation 37](#_Toc320098948)

[Performing Baseline FTDW Validation 38](#_Toc320098949)

[Baseline Testing with SQLIO 39](#_Toc320098950)

[Performing Fast Track Database Benchmark 41](#_Toc320098951)

[Calculating MCR 42](#_Toc320098952)

[Calculating BCR 43](#_Toc320098953)

[Published FTDW Reference Architectures 46](#_Toc320098954)

[Conclusion 46](#_Toc320098955)

[Appendix 47](#_Toc320098956)

[FTDW System Sizing Tool 47](#_Toc320098957)

[Validating a User-Defined FTRA 47](#_Toc320098958)

[Synthetic I/O Testing 47](#_Toc320098959)

[Generating Test Files with SQLIO 47](#_Toc320098960)

[Workload Testing 50](#_Toc320098961)

[Measuring the MCR for Your Server (Optional) 50](#_Toc320098962)

[Measuring the BCR for Your Workload 51](#_Toc320098963)

[Factors Affecting Query Consumption Rate 54](#_Toc320098964)

# FTDW Change History

The following table provides a list of notable changes or updates for versioned releases of the Fast Track Data Warehouse Reference Guide.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Version | Note | Location |
| New for SQL Server 2012 | 4.0 | Links to other SQL Server Best Practices documents | [Important](#_SQL_Server_Best) |
| New for SQL Server 2012 | 4.0 | Benchmarking and validation | [Caution](#_Benchmarking_and_Validation) |
| New for SQL Server 2012 | 4.0 | Memory requirements | [RAM](#_Server_Memory) |
| New for SQL Server 2012 | 4.0 | xVelocity memory-optimized columnstore indexes | [Columnstore indexes](#_Column_Store_Index) |
| New for SQL Server 2012 | 4.0 | Solid state storage | [Solid state](#_Solid_State_Storage) |
| New for SQL Server 2012 | 4.0 | Validation and columnstore indexes | [Validation](#_Fast_Track_Validation) |
| New for SQL Server 2012 | 4.0 | Validation of baseline I/O | [SQLIO](#_Testing_with_SQLIO) |

**Table 1:** Change history

# Introduction

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

## Audience

The target audience for this document consists of IT planners, architects, DBAs, and business intelligence (BI) users with an interest in choosing standard, proven system architectures for FTDW-conforming SQL Server workloads.

# Fast Track Data Warehouse

The SQL Server Fast Track Data Warehouse initiative provides a basic methodology and concrete examples for the deployment of balanced hardware and database configuration for a data warehousing workload. For more information, see the [FTDW Workload](#_FTDW_Workload) section of this document.

Balance is a measure of key system components of a SQL Server installation; storage, server, storage network, database, and operating system. Each of these components is tuned to optimal configuration. The goal is to achieve an efficient out-of-the-box balance between SQL Server data processing capability and hardware component resources. Ideally, your configuration will include minimum system hardware to satisfy storage and performance requirements for a data warehousing workload.

## Fast Track

The SQL Server Fast Track brand identifies a component hardware configuration that conforms to the principles of the FTDW reference architecture (FTRA). Each FTRA is defined by a workload and a core set of configuration, validation, and database best practice guidelines. The following are key principles of the Fast Track program:

* Workload-specific benchmarks. System design and configuration are based on real concurrent query workloads.
* Detailed and validated hardware component specifications.
* Component architecture balance between database capability and key hardware resources.

## Value Proposition

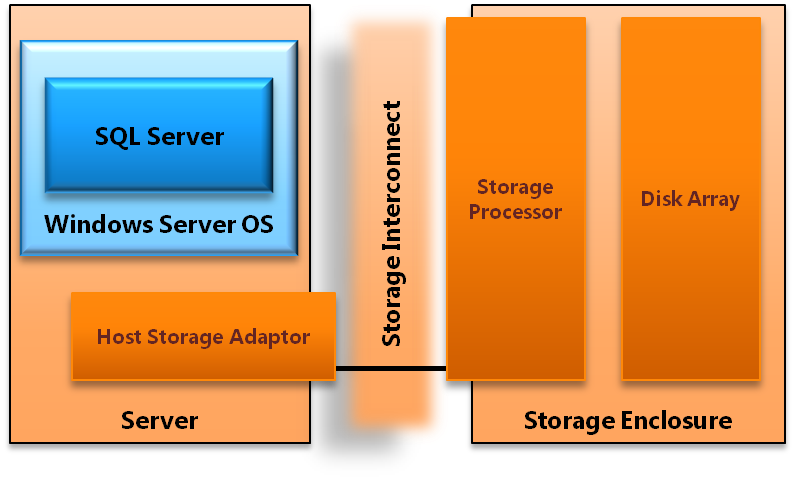
The following principles create the foundation of the FTDW value proposition:

* **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.
* **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

## Holistic Component Architecture

SQL Server FTDW reference architectures provide a practical framework for balancing the complex relationships between key components of 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 Approach

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 small request, lookup-based online transaction processing (OLTP) workloads and scan-intensive, large-request, analytical data warehousing. OLTP use cases are heavily indexed to support low latency retrieval of small numbers of rows from data sets that often have little historical data volume. 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. With Fast Track data warehousing workloads an emphasis is placed on achieving consistently high I/O scan rates (measured in MB/s) rather than the more traditional focus on operations per second (measured in IOPS).

The challenge of very different workloads is addressed by clearly defining the attributes of customer workloads. SQL Server Fast Track workloads comprise a qualitative list of attributes that uniquely define a common database application use case. In addition, each workload is represented by quantitative measures including standard benchmark queries. Workload-specific benchmarking is used to validate database configuration, best practices, and component hardware recommendations.

## Validated SQL Server Fast Track Reference Configurations

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

## Summary

The SQL Server FTDW specification described in this reference guide is workload-centric and component balanced. This 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

## 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 simply add drives; it is not uncommon to see hundreds of disks supporting a relatively small data warehouse in an attempt to overcome the I/O performance limitations of mapping a seek-based I/O infrastructure to a scan based workload. This is frequently seen in large shared storage area network (SAN) environments that are traditionally seek optimized. Many storage I/O reference patterns and techniques that encourage random I/O access, introducing disk latency and reducing the overall storage subsystem throughput for a data warehouse workload that is scan intensive.

Fast Track Data Warehouse is a different way of optimizing for data warehouse workloads. By aligning database files and configuration with efficient disk scan (rather than seek) access, performance achieved from individual disks can be many factors higher. The resulting per-disk performance increase reduces the number of disks needed to generate sufficient I/O throughput to satisfy the ability of SQL Server to process data for a given workload. Furthermore, you can avoid some index-based optimization techniques used to improve disk seek.

## Workload Evaluation

When analyzing workloads for FTDW based systems it is important to consider fit against practices and system configurations outlined in this document. Data warehouse requirements can vary by customer and certain requirements, such as database replication, may not be appropriate for all FTDW designed systems. Key, initial criteria for this type of workload evaluation are outlined here.

**Scan-Intensive**

Queries in a data warehouse workload frequently scan a large number of rows. For this reason, 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 and a correlated increase in 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 carefully managed. Workloads that commonly introduce such volatility may not be well aligned to FTDW. Where volatility does occur, we recommend periodic maintenance to minimize fragmentation.

**Index-Light**

Adding nonclustered indexes generally adds performance to lookups of one or few records. If nonclustered indexes are applied to tables where large numbers of rows are to be retrieved, the resulting increase in random disk seek operations can degrade overall system performance. Maintaining indexes can also add significant data management overhead, which may create risk for service-level agreement (SLA) and the ability to meet database load windows.

In contrast, sequential scan rates can be many factors higher (10 times or more) than random access rates. A system that minimizes the use of random seek, inducing secondary indexes, typically sees much higher average sustained I/O rates. This means more efficient use of storage I/O resources and more predictable performance for large scan-type queries.

FTDW methodology prescribes database optimization techniques that align with the characteristics of the targeted workload. Clustered index and range partitioning are examples of data structures that support efficient scan-based disk I/O, and we recommend them as the primary tools for data architecture based optimization for FTDW environments.

**Partition-Aligned**

A common trait of FTDW workloads is the ability to take advantage of SQL Server partitioning. 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 and significantly reduce the size of table scans without sacrificing fragmentation or disk I/O throughput.

**Additional 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:
  + The type of fragmentation of primary concern can be measured in terms of [fragment size](#_Extent_Fragmentation_1). A fragment represents contiguous allocations of 8K database pages.
  + Expanding the server by adding storage requires that all performance-sensitive tables be repopulated in a manner consistent with guidelines provided in this document.
  + Implementing volatile data structures, such as tables with regular row-level update activity, may require frequent maintenance (such as defragmentation or index rebuilds) to reduce fragmentation.
  + Loading of cluster index tables with batches of cluster key IDs that overlap existing ranges is a frequent source of fragmentation. This should be carefully monitored and managed in accordance with the best practices provided in this reference guide.
* Data warehousing can mean many things to different audiences. Care should be taken to evaluate customer requirements against FTDW workload attributes.

## Qualitative Data Warehouse Workload Attributes

You can define the FTDW workload 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 or operational data store (ODS) workload.

|  |  |  |
| --- | --- | --- |
| Attribute | Workload affinity:  Data warehouse | OLTP/ODS |
| Use Case Description | * 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) | * 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 | * 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 | * 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 | * 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) | * Minimal use of heap table structures * Clustered index table structures that support detailed record lookups (1 to few rows per request) * Smaller fact tables (for example, less than100 GB) * Relatively small data sizes (for example, a few terabytes) |
| Database optimization | * Minimal use of secondary indexes (described earlier as index-light) * Partitioning is common | * Heavy utilization of secondary index optimization |

**Table 2:** Data warehouse workload attributes

# 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 final approach requires detailed workload profiling and system benchmarking in advance of purchase or deployment. It requires a high degree of technical knowledge in the areas of enterprise server and storage configuration as well as SQL Server database optimization.

## Option 1: Basic Evaluation

In this scenario, the customer has already targeted an FTDW reference configuration or has alternative methods to determine server and CPU requirements. If you use this option, you do not need to perform a full platform evaluation (that is, a proof of concept).

## 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. The first step of choosing a configuration is to identify these characteristics; start by examining key areas of your customer’s requirements and usage patterns.

#### 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 because conflicts may indicate that 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 providing 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](#_FTDW_Workload) section of this document.
2. Evaluate FTDW best practices. Practices that relate 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 a fully informed decision can be made when a validated FTDW reference architecture is chosen. 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 here; primary customer use cases that directly conflict with any of these attributes should be carefully evaluated because they may render the methodology invalid for the use case.

#### 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 reading tens of thousands to millions (or more) 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 Architecture

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, you do not perform a full platform evaluation in an FTDW basic evaluation. Instead, you select a conforming FTDW configuration that matches your customer’s estimated 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. This approach assumes a high degree of expertise in database system architecture and data warehouse deployment. Fast Track partners and Microsoft technical sales resources are typically involved in this process.

### Process Overview

The following process flow summarizes the FTDW Full Evaluation selection process:

1. Evaluate Fast Track workload attributes against the target usage scenario.
2. Identify server and/or bandwidth requirements for the customer use case. A published FTDW reference configuration must be chosen before you begin an evaluation.
3. Identify a query that is representative of customer workload requirement.
4. Calculate the Benchmark Consumption Rate (BCR) of SQL Server for the query.
5. Calculate the Required User Data Capacity (UDC).
6. Compare BCR and UDC ratings against published Maximum CPU Consumption Rate (MCR) and Capacity ratings for conforming Fast Track reference architectures.

The following describes individual points of the Full Evaluation process flow in detail.

### Step 1: Evaluate the Customer Use Case

#### Workload Evaluation

This process is the same as for Option 1: Basic Evaluation.

#### Select FTDW Evaluation Hardware

Before you begin a full system evaluation, you must choose and deploy a published FTDW reference configuration for testing. 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 run on existing hardware.
* Ad-hoc analysis. The [FTDW Sizing Tool](http://download.microsoft.com/download/D/F/A/DFAAD98F-0F1B-4F8B-988F-22C3F94B08E0/Fast%20Track%20Core%20Calculator%20v1.2.xlsx) provides a basic approach to calculating FTDW system requirements based on basic assumptions about the targeted database workload. This spreadsheet tool is available for download from <http://download.microsoft.com/download/D/F/A/DFAAD98F-0F1B-4F8B-988F-22C3F94B08E0/Fast%20Track%20Core%20Calculator%20v1.2.xlsx>.

### 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)

For more information about calculating these metrics, see the [Benchmarking and Validation](#_Benchmarking_and_Validation) section of this document.

#### MCR

This metric measures the maximum SQL Server data processing rate for a standard query and data set for a specific server and CPU combination. This is provided as a per-core rate, and it is measured as a query-based scan from memory cache. MCR is the initial starting point for Fast Track system design. It represents an estimated maximum required I/O bandwidth for the server, CPU, and workload. MCR is useful as an initial design guide because it requires only minimal local storage and database schema to estimate potential throughput for a given CPU. *It is important to reinforce that MCR is used as a starting point for system design – it is not a measure of system performance.*

#### BCR

BCR is measured by a set of queries that are considered definitive of the FTDW workload. BCR is calculated in terms of total read bandwidth from disk and cache, rather than cache only as with the MCR calculation. BCR can enable tailoring of the infrastructure for a given customer use case by measuring against a set of queries that matches customer workload patterns. Or, in the case of partner validated FTRA, a set of benchmark queries are used that ensure systems are designed for high-stress workloads. In summary, BCR is a real measure of data processing using multiple queries under concurrent workload against significant data volumes.

#### User Data Capacity

This is the anticipated database capacity for the SQL Server database. Fast Track user data capacity accounts for post-load database compression and represents an estimate for the amount of uncompressed user data files or streams that can be loaded to the Fast Track system. The standard compression ratio used for FTDW is 3.5:1.

Note that any storage expansion beyond initial deployment potentially requires data migration that would effectively stripe existing data across the new database file locations. For this reason it is important to take expected database growth and system life expectancy into account when choosing an appropriate reference architecture.

### Step 3: Choose a Fast Track Data Warehouse Reference Architecture

After it is calculated, BCR can be compared against published MCR and capacity ratings provided by Fast Track partners for each published FTRA. For more information about our partners, see [Fast Track Data Warehousing](http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx) (http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx).

You can use the BCR metric as a common reference point for evaluating results from the test/evaluation system against published configurations. Starting with the BCR data, your customer can choose the Fast Track option that best aligns with the test results.

## Option 3: User-Defined Reference Architectures

This approach leverages the FTDW methodology to tailor a system for a specific workload or set of hardware. This approach requires a thorough understanding of both SQL Server and the hardware components that it runs on. The following steps outline the general approach for developing a user-defined reference architecture that conforms to FTDW principles.

### 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 FTRAs, 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 a basic framework for developing a reference architecture for a predefined workload:

1. Establish the Maximum CPU Core Consumption Rate (MCR) for the chosen server and CPU. Use the method outlined in the [Benchmarking and Validation](#_Benchmarking_and_Validation) section of this document to calculate MCR. 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. Use the MCR value to estimate storage and storage network requirements and create an initial system design.
3. Procure a test system based on the initial system design. Ideally this will be the full configuration specified.
4. Establish a Benchmark Consumption Rate (BCR). Based on workload evaluation, identify a query or in the ideal case a set of representative queries. Follow the practices described in the [Measuring BCR for Your Workload](#_Measuring_the_BCR) section of this document.
5. Adjust the system design based on the results.
6. Establish the final server and storage configuration.

#### 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 [Validating a User-Defined FTRA](#_Validating_a_Fast) section of this document. To validate your system, follow these steps:

1. Evaluate component throughput against the established performance requirements. This ensures that real system throughput matches expectations.
2. Validate system throughput by rebuilding to final configuration and executing final benchmarks. As a general rule, the final BCR should achieve 80 percent or better of the system MCR.

## Choosing an FTRA Summary

The following table summarizes the three FTRA selection options.

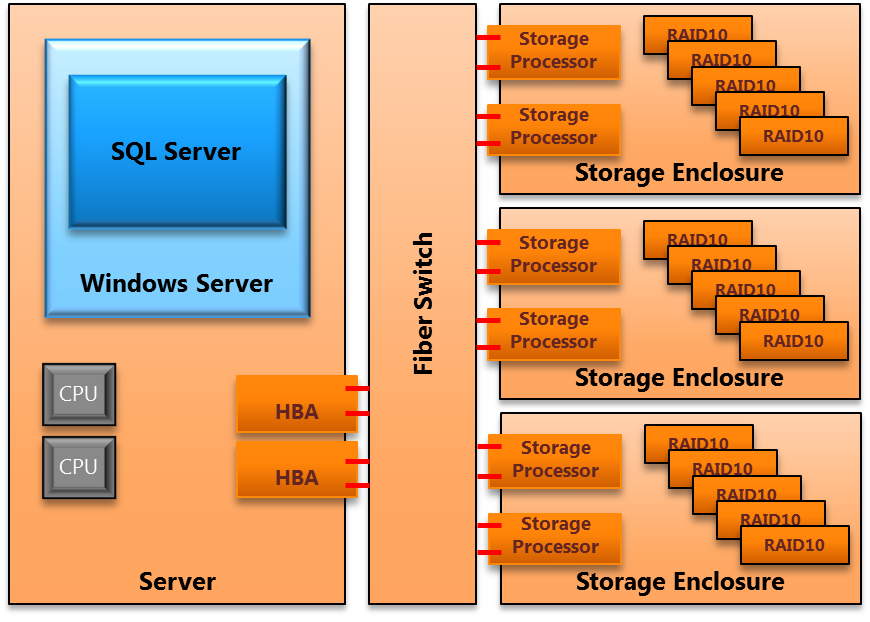
|  |  |  |
| --- | --- | --- |
| Option | Pros | Cons |
| Basic evaluation | * Very fast system set-up and procurement (days to weeks) * Minimized cost of design and evaluation * Lower infrastructure skill requirements | * Possibility of over-specified storage or under-specified CPU |
| Full evaluation | * Predefined reference architecture tailored to expected workload * Potential for cost-saving on hardware * Increased confidence in solution | * Evaluation takes effort and time (weeks to months) * Requires detailed understanding of target workload |
| User-defined reference architecture | * Potential to reuse existing hardware * Potential to incorporate latest hardware * System highly tailored for your use-case | * Process takes several months * Requires significant infrastructure expertise * Requires significant SQL Server expertise |

**Table 3**: Comparison of different evaluation options

# FTDW Standard Configuration

## Hardware Component Architecture

Current FTDW reference architectures are based on dedicated storage configurations. Currently published options include switched SAN, direct attached SAN, direct attached SAS, SAS-RBOD, and iSCSI. Disk I/O throughput is achieved through the use of independent, dedicated storage enclosures and processors. Additional details and configurations are published by each Fast Track vendor. Figure 2 illustrates the component-level building blocks that comprise a FTDW reference architecture based on SAN storage.



**Figure 2:** Example storage configuration for a 2-socket, 12-core server

### Component Requirements and Configuration

#### Server Memory

***Total RAM:*** RAM allocation for FTRAs are based on benchmark results with the goal of balancing maximum logical throughput (total pages read from disk and buffer over time) with CPU utilization. Table 4 lists recommended memory allocations for SQL Server 2012 reference architectures. The maximum memory values provided are not hard limits, but represent average values for successfully validated systems.

|  |  |  |
| --- | --- | --- |
| Server size | Minimum memory | Maximum memory |
| 1 socket | 64 GB | 128 GB |
| 2 socket | 128 GB | 256 GB |
| 4 socket | 256 GB | 512 GB |
| 8 socket | 512 GB | 768 GB |

**Table 4:** Recommended memory allocations for SQL Server 2012

The following considerations are also important to bear in mind when you evaluate 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 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.
* **xVelocity memory-optimized columnstore index**: Workloads that heavily favor columnstore index query plans run more efficiently with memory pools at the higher end of the ranges listed in Table 4.

#### 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 and PCIe bus capabilities. This recommendation is a general guideline and is consistent with testing performed during FTDW reference configuration development.

If zoning is used, only ports in use for Fast Track should exist in the zone(s). Detailed FC network topology and configuration is documented in the Technical Configuration Guide provided by each Fast Track partner and specific to each published FTRA.

***Multipath I/O (MPIO):*** MPIO should be configured. Each volume hosted on dedicated storage arrays should have at least one Active path.

Round-robin with subset is the default policy used for Fast Track configurations but is rarely used for partner reference architectures, because more optimal configurations are identified by FTDW partner engineering teams. Partner-specific DSMs and/or documents often prescribe different settings and should be reviewed prior to configuration.

#### Storage

***Local Disk:*** A 2-disk RAID1 array is the minimum allocation for Windows Server and SQL Server installation. Sufficient disk space should be allocated for virtual RAM and paging requirements. In general, the greater of 250 GB or 1.5 times system RAM should be available in free disk space. Remaining disk configuration depends on the use case and customer preference.

***Logical File System:*** Mounting LUNs to mount-point folder paths in Windows, rather than drive letters, is preferred due to the number of volumes in many Fast Track systems.

It can also be useful to understand which Windows operating system drive assignment represents which LUN (volume), RAID disk group, and Windows Server mount point in the storage enclosures. You can adopt a naming scheme for the mount points and volumes when mounting LUNs to Windows folders. For more information about device naming schemes, see the Fast Track partner technical configuration guidance.

You can use vendor-specific tools to achieve the recommended volume naming scheme. If an appropriate tool does not exist, you can make one disk available to Windows at a time from the storage arrays while assigning drive names to ensure the correct physical-to-logical topology.

***Physical File System:*** For more information, including detailed instructions, see the [Application Configuration](#_Application_Configuration) section of this document.

***Storage Enclosure Configuration:*** All enclosure settings remain at their defaults unless otherwise documented by Fast Track partner technical documentation. FTDW specifications for file system configuration require 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 R2

Unless otherwise noted, default settings should be used for the Windows Server 2008 R2 Enterprise operating system. Ensure that the latest service pack and all critical updates are applied. The Multipath I/O feature is required for many reference architectures. For more information about detailed MPIO configuration, see the Fast Track partner’s technical configuration guide for the given reference architecture. Confirm that Windows Server 2008 R2 is installed as an Application Server role to ensure proper .NET framework installation and defaults.

### SQL Server 2012 Enterprise

#### Startup Options

**-E** must 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. For more information about this option, see [Microsoft Knowledge Base Article 329526](http://support.microsoft.com/kb/329526) (http://support.microsoft.com/kb/329526). It is important to ensure that the **-E** option has taken effect at database startup. The option is case-sensitive and format-sensitive. White space before or after the option can prevent initialization.

**-T1117** should also be added to the start-up options. This trace flag ensures even growth of all files in a file group in the case that autogrow is enabled. The standard FTDW recommendation for database file growth is to preallocate rather than autogrow (with the exception of **tempdb**). For more information, see the [Storage Configuration Details](#_Storage_Configuration_Details) section of this document.

Enable option **Lock Pages in Memory**. For more information, see [How to: Enable the Lock Pages in Memory Option](http://go.microsoft.com/fwlink/?LinkId=141863) (http://go.microsoft.com/fwlink/?LinkId=141863).

**-T834** should be evaluated on a case-by-case basis. This trace flag can improve throughput rates for many data warehousing workloads. This flag enables large page allocations in memory for the SQL Server buffer pool. For more information about this and other trace flags, see [Microsoft Knowledge Base Article 920093](http://support.microsoft.com/kb/920093) (http://support.microsoft.com/kb/920093).

**Note**: At this time SQL Server 2012 does not support the use of **–T834** if columnstore indexes are in use on the database. If you plan to use columnstore indexes, do not use this trace flag.

#### SQL Maximum Memory

For SQL Server 2012 no more than 92 percent of total server RAM should be allocated to SQL Server. If additional applications are to share the server, the amount of RAM left available to the operating system should be adjusted accordingly. This setting is controlled by **the max server memory** option. For more information about memory settings for validated reference architectures, see the FTDW partner documentation.

#### Resource Governor

Data warehousing workloads typically include complex queries that operate on large volumes of data. These queries can consume large amounts of memory, and they may spill to disk if memory is constrained. This behavior has specific implications in terms of resource management. You can use the Resource Governor technology in SQL Server 2012 to manage resource usage.

In default settings for SQL Server, Resource Governor is 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.

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. Although 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 concurrent query workloads. 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).

Vendor specific guidance and practices for Fast Track solutions should also be reviewed. In particular, larger 4-socket and 8-socket Fast Track solutions may rely on specific Resource Governor settings to achieve optimal performance.

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.

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).

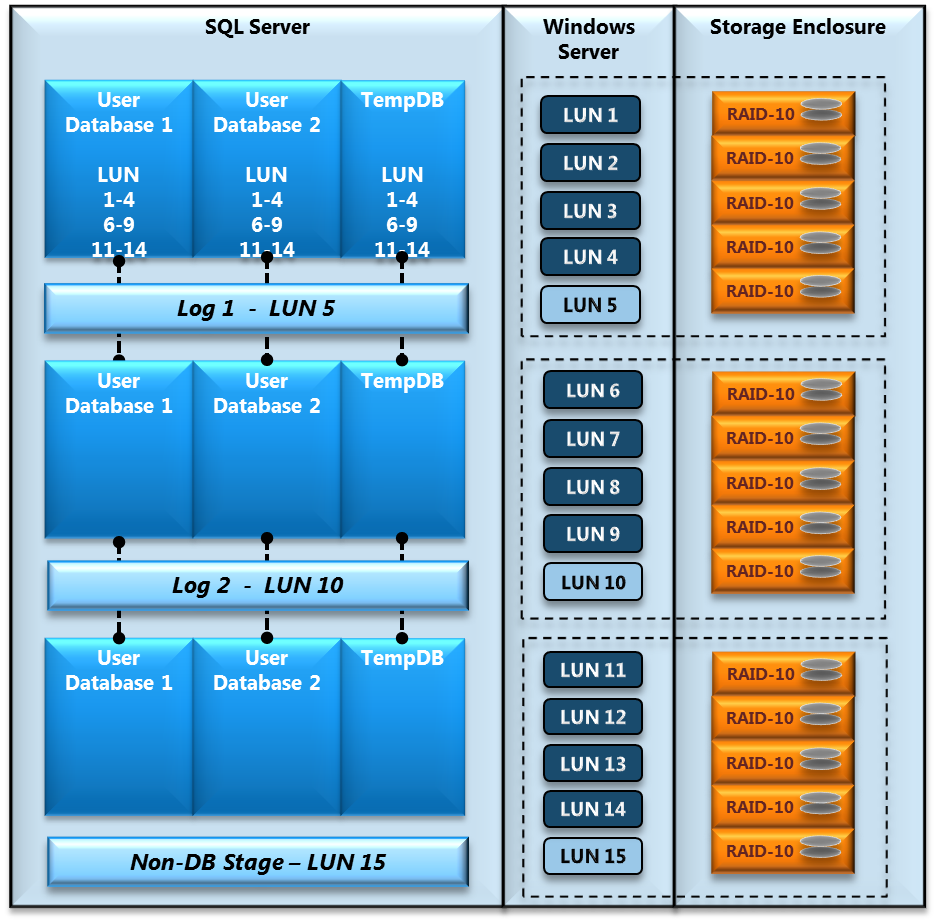
### Storage System

Managing fragmentation is crucial to system performance over time for FTDW reference architectures that place primary database storage on hard disk drives (HDD). 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. This should be considered a reference case as specific topology varies greatly by Fast Track partner. The typical database stack contains the following elements:

* Physical disk array:4 spindle RAID 1+0 is the standard approach depicted in Figure 3. RAID 5 and RAID 6 have also been used in some partner reference architectures for SQL Server 2008 R2 and SQL Server 2012.
* Operating system volume assignment (LUN)
* Databases: User, System Temp, System Log



**Figure 3**: Example comprehensive storage architecture for a FTDW system based on three storage enclosures with one LUN (volume) per disk group

#### Storage Configuration Details

For *each* storage enclosure, do the following.

1. Create disk groups of four disks each, using RAID 1+0 (RAID 10). The exact number of disk groups per storage enclosure can vary by vendor. For more information, see the vendor-specific documentation. In general, the number is (2) RAID10 and (1) RAID1 disk group for large form factor (LFF) enclosures and (5) RAID10 disk groups for small form factor (SFF) enclosures.  
   The total volumes used as filegroup locations for primary data should not exceed 32. If the total number of storage system LUN exceeds this threshold, larger disk groups can be used to reduce LUN count while maintaining similar I/O throughput. For example, use an 8 disk RAID 10 disk group with 1 LUN instead of a 4 disk RAID 10 disk group with 1 LUN. There is some reduction in throughput and efficiency with larger disk groups. This varies by storage technology.
2. Dedicate all but one disk group to primary user data (PRI). Primary user data locations are synonymous with SQL Server database filegroup locations.  
   All FTRAs call for either one or two LUNs per PRI disk group. Refer to vendor-specific guidance for your chosen reference architecture. These LUNs are used to store the SQL Server database files (.mdf and .ndf files).
3. Ensure that primary storage processor assignment for each disk volume allocated to primary data within a storage enclosure is evenly balanced. For example, a storage enclosure with four disk volumes allocated for primary data will have two volumes assigned to storage processor “A” and two assigned to storage processor “B”.
4. Create one LUN on the remaining disk group to host the database transaction logs. For some larger Fast Track configurations log allocations are limited to only the first few storage enclosures in the system. In this case the additional disk groups are used for nondatabase staging or left unpopulated to reduce cost.

For each database, do the following:

1. Create at least one filegroup containing one data file per PRI 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 include all PRI LUNs as locations for each filegroup.
2. When you create the files for each filegroup, preallocate them to their largest anticipated size, with a size large enough to hold the anticipated objects.
3. Disable the autogrow option 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](#_Managing_Data_Fragmentation) section of this document.

For **tempdb**, do the following:

1. Preallocatespace, and then add a single data file per LUN. Be sure to make all files the same size.
2. Assign temp log files onto one of the LUNs dedicated to log files.
3. Enable autogrow; in general the use of a large growth increment is appropriate for data warehouse workloads. A value equivalent to 10 percent of the initial file size is a reasonable starting point.
4. Follow standard SQL Server best practices for database and **tempdb** sizing considerations. Greater space allocation 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.

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.
2. Enable the autogrow option for log files.
3. Ensure that log capacity aligns to the requirements provided in Table 5. Some variation is acceptable depending on specific system design characteristics.

|  |  |  |
| --- | --- | --- |
| System RAM (GB) | FT rated capacity (terabytes) | Recommended minimum log allocation  Mirrored GB free space |
| <= 96 | <=10 | 300 GB X 1 volume |
| <= 128 | >10  <=40 | 300 GB x 2 volume  or  600 GB x 1 volume |

**Table 5:** Log allocation recommendations

Refer to existing best practices for SQL Server transaction log allocation and management.

#### Solid State Storage

FTDW reference architectures that utilize solid state storage for primary (PRI) data have many advantages, including simplified management, lower operational costs, and predictable maintenance.

***Simplified management:*** Solid state storage does not require fragmentation management. The SQL Server startup option **–E** should still be used, but no further optimization or management of page allocation is required. This simplification makes long-term management of FTDW environments significantly easier. In addition, larger disk groups and lower volume/LUN counts can be used with no negative performance implications. This change simplifies filegroup creation and maintenance.

***I/O resiliency:*** Solid state storage has minimal performance degradation under high concurrency or page fragmentation. In addition, mixed random read (seek) workload does not negatively impact large request (scan) I/O patterns.

***Predictable maintenance:*** Many solid-state storage options provide software based write-life monitoring with lower frequency of difficult-to-predict physical failures.

***Lower operational costs:*** While more expensive at list-price, solid-state storage offers a more efficient balance of I/O throughput to capacity per unit. Effective FTDW workload I/O rates for 300 GB 10k SAS HDD average 50 MBs. Enterprise MLC SSD delivers between 150 and 200 MBs at 600-GB capacity. Additionally, solid-state storage draws significantly less power, generates less heat, and often supports higher density solutions.

#### Solid-State Storage Configuration

The following adjustments can be made to standard FTDW storage configuration guidelines if solid state storage is used for PRI volumes.

* If mirroring is required, RAID1+0 or RAID5 can be used. RAID5 provides the best capacity at no performance penalty for FTDW workloads on solid state.
* LUN and volume count can be reduced to as low as one PRI volume per storage unit. It is helpful in some cases to have the PRI volume count be a multiple of the CPU core count. The minimum PRI volume count is two.
* The transaction log can be placed on solid state as well, but FTDW workloads are not typically log-bound. Cost can be reduced by placing the log on traditional HDD. The same is true of local storage for Windows Server and SQL Server installation.
* Recommendations for page fragmentation management and cluster index parallel loading can be ignored, because logical database fragmentation does not impact solid stage I/O performance.

# SQL Server Best Practices for FTDW

Practices for Fast Track workloads are validated and documented in two cases. The first occurs if a Fast Track practice differs substantively from established SQL Server best practices. The second case occurs in scenarios where existing practices are missing or not easily accessible. The practices provided here are not intended to be comprehensive, because there is a rich set of existing documentation for SQL Server database deployment. Existing SQL Server technical documentation and best practices should be referred to on numerous topics related to a FTDW deployment.

**Important**: There are several links to documentation written for SQL Server 2008 R2 in this guide. We believe that the majority of this guidance is still valuable for SQL Server 2012. You should look for updated versions of these documents as they become available. Future releases of this reference guide will update links at they become available.

## 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.

Choosing a table type comes down to how the data in the table will be accessed the majority of the time. The following information 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 inherently 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 partitioning is applied).

Scanning heap tables reaches maximum throughput at 32 files, so use of heaps for large fact tables on systems with high LUN (more than 32) or core (more than 16) counts may require the use of Resource Governor, DOP constraints, or changes to the standard Fast Track database file allocation.

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 clustered index should also be the clustered index key.  
  **Note**: Choosing a clustered index key that is not the date partition column for a clustered index table might be advantageous in some cases. However, this is likely to lead to fragmentation unless *complete* partitions are loaded, because new data that overlaps existing clustered index key ranges creates page splits.
* Queries against the table normally do granular or range constrained lookups, not full table or large multi-range scans.

### Table Partitioning

Table partitioning can be an important tool for managing fragmentation in FTDW databases. For example, partitioning can be used to update or delete large blocks of range based user data from a table without addressing other parts of the table. In contrast, deleting row by row from a cluster index can induce significant extent fragmentation. A common scenario is to rebuild newer partitions after they age and the frequency of DML operations for the range of data decreases. The partition is now stable relative to DML operations and has minimal extent fragmentation.

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 are scanned. (Partitions that support Analysis Services ROLAP mode may be better structured as clustered indexes.)

For more information about table partitioning, see the white paper [Partitioned Table and Index Strategies Using SQL Server 2008](http://msdn.microsoft.com/en-us/library/dd578580(v=SQL.100).aspx) (http://msdn.microsoft.com/en-us/library/dd578580(v=SQL.100).aspx).

### Indexing

Consider the following guidelines for FTDW index creation:

* Use a clustered index for date ranges or common restrictions.
* Use columnstore index whenever possible. The following section discusses best practices for working with columnstore indexes in FTDW environments.
* Reserve nonclustered indexing for situations where granular lookup is required and table partitioning does not provide sufficient performance. If possible use a columnstore index as an alternative to nonclustered index.
* Nonclustered, covering indexes may provide value for some data warehouse workloads. These should be evaluated on a case-by-case basis and compared against the columnstore index.

### xVelocity In-Memory Columnstore Indexes

SQL Server 2012 introduces a new data warehouse query acceleration feature based on columnar technology: columnstore indexes. These new indexes, combined with enhanced query processing features, improve data warehouse query performance for a broad range of analytical queries.

xVelocity memory-optimized columnstore indexes are “pure” columnstores (not hybrid) because they store all data for included columns on separate pages. Columnstore indexes improve I/O scan performance and buffer hit rates, and they are well aligned with FTDW design methodology.

#### Best Practices

Columnstore index objects reside alongside tables and are created in a similar fashion to nonclustered indexes. These facts do imply that incremental storage capacity is necessary. It is not necessary to create columnstore indexes in separate filegroups unless frequent changes to the table targeted by the index are expected. Maintaining columnstore indexes in separate filegroups can help you manage page fragmentation over time in highly volatile environments.

#### Creating Columnstore Indexes for Normalized Data Models

Normal data models (that is, 3NF) often trigger joins between two or more large (fact) tables. These types of joins are not currently ideal for columnstore index processing, and they may display performance regressions relative to non-columnstore-index query plans. The following approaches can help you avoid this issue with normal data models:

* Use query-level hints to block columnstore index processing from being used.
* Use OPTION(IGNORE\_NONCLUSTERED\_COLUMNSTORE\_INDEX)
* Rewrite queries. For more information, see the resources listed in the [Columnstore Index General Best Practices](#_CSI_General_Best) section of this document.
* Try omitting common join keys from one table involved in the SQL join(s) that display performance regressions from non-columnstore-index query plans. Omitting the join key from the columnstore index on one table may result in the columnstore index not being used for queries that join on the omitted column. This approach may be useful in environments where query level options cannot be applied. Be aware that omitting a column from the columnstore index is not guaranteed to result in a better query plan and may impact other queries for which the columnstore index would provide performance benefits. If you choose to use this option, selecting a column from the smaller of the tables involved can reduce the performance impact on other queries. Note that declared (DDL) primary keys must be included in the columnstore index, which may limit available join columns. Even if you omit a primary key column from the columnstore index definition, all primary key columns are added automatically to the columnstore index when it is created.

While normal data models are not perfectly optimized for columnstore indexes in the current release, it is important to note that FTDW benchmarking is based on a modified version of TPC-H, which is a normalized model. Significant gains were still measured for concurrent workloads that mixed both columnstore index and non-columnstore-index query plans, including FTDW rated throughput that nearly doubled overall workload performance in some cases.

#### Creating Columnstore Indexes for Dimensional Data Models

Follow standard [columnstore index best practices](#_CSI_General_Best) for dimensional models such as star schemas. This can be considered a best-case scenario for columnstore index processing.

#### Memory Management for Columnstore Indexes

FTRA validated for SQL Server 2012 generally have more total system RAM than similar configurations for SQL Server 2008 R2. The primary reason for this is that columnstore index enhanced workloads run more efficiently with larger memory pools. Resource Governor should always be used to set the maximum amount of memory per session for FTDW environments in which you plan to take advantage of columnstore indexes. Validated FTRAs document Resource Governor settings used to achieve FT-rated performance, and these values can be considered a starting point for customer workloads. Ideally the setting will be evaluated and tuned specifically for a customer workload after system installation.

The following SQL command configures SQL Server Resource Governor to these recommendations. In this case the maximum amount of memory per session is set to 19 percent.

ALTER RESOURCE GOVERNOR RECONFIGURE;

ALTER WORKLOAD GROUP [default] WITH(request\_max\_memory\_grant\_percent=19);

***xVelocity memory-optimized Columnstore Index General Best Practices***

FTDW reference guidance covers only practices unique to Fast Track. For more information about columnstore indexes, see the [SQL Server 2012 CSI Tuning Guide](http://social.technet.microsoft.com/wiki/contents/articles/sql-server-columnstore-performance-tuning.aspx) (http://social.technet.microsoft.com/wiki/contents/articles/sql-server-columnstore-performance-tuning.aspx) and the [SQL Server 2012 CSI FAQ](http://social.technet.microsoft.com/wiki/contents/articles/sql-server-columnstore-index-faq.aspx) (http://social.technet.microsoft.com/wiki/contents/articles/sql-server-columnstore-index-faq.aspx).

### Database Statistics

Your decision of 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, see [Statistics for SQL Server 2008](http://msdn.microsoft.com/en-us/library/dd535534.aspx) (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 (sync or async) options for statistics (the system default in SQL Server). Use of this technique minimizes the need to run statistics manually.
* 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 (multi-column) 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.

### Compression

FTDW configurations are designed with page compression enabled. We recommend that you use page compression on all fact tables. Compression of small dimension tables (that is, those with fewer than a million rows) is optional. With larger dimension tables it is often beneficial to use page compression. In either case, compression of dimension tables should be evaluated on a use case basis. Row compression is an additional option that provides reasonable compression rates for certain types of data.

SQL Server page compression shrinks data in tables, indexes, and partitions. This reduces the amount of physical space required to store user tables, which enables more data to fit into the SQL Server buffer pool (memory). One benefit of this is in a reduction of the number of I/O requests serviced from physical storage.

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 by servicing I/O requests from memory buffer. Usually, page compression has a compression ratio (original size/compressed size) of between 2 and 7:1, with 3:1 being a typical conservative estimate. Your results will vary depending on the characteristics of 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 an FTDW 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 preallocation of files to their expected maximum 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 or tables 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.

The following query provides key information for evaluating logical fragmentation for a FTDW table. The highest priority metric is Average Fragment Size. This value provides an integer that represents the average number of SQL Server pages that are clustered in contiguous extents.

SELECT  db\_name(ps.database\_id) as database\_name

              ,object\_name(ps.object\_id) as table\_name

              ,ps.index\_id

              ,i.name

              ,cast (ps.avg\_fragmentation\_in\_percent as int) as [Logical Fragmentation]

              ,cast (ps.avg\_page\_space\_used\_in\_percent as int) as [Avg Page Space Used]

              ,cast (ps.avg\_fragment\_size\_in\_pages as int) as [Avg Fragment Size In Pages]

              ,ps.fragment\_count as [Fragment Count]

              ,ps.page\_count

              ,(ps.page\_count \* 8)/1024/1024 as [Size in GB]

FROM sys.dm\_db\_index\_physical\_stats (DB\_ID() *--NULL = All Databases*

                                                              , OBJECT\_ID('$(TABLENAME)')

                                                              , 1

                                                              , NULL

                                                              , 'SAMPLED') AS ps     *--DETAILED, SAMPLED, NULL = LIMITED*

INNER JOIN sys.indexes AS i

              on (ps.object\_id = i.object\_id AND ps.index\_id = i.index\_id)

WHERE ps.database\_id = db\_id()

    and ps.index\_level = 0;

The following table provides a general guideline for interpreting Average Fragment Size values.

|  |  |  |
| --- | --- | --- |
| Average fragment size | Status | Action |
| >400 | Ideal | This is an ideal value and can be difficult to maintain for some data structures. |
| 300-399 | Green | The table will provide good I/O performance and does not require maintenance for logical fragmentation. |
| 150-299 | Yellow | Logical fragmentation is most likely impacting I/O efficiency. Maintenance is recommended to improve fragment count. |
| 10-149 | Red | Severe logical fragmentation. Large I/O requests against this structure will result in significant disk head movement and reduce overall system I/O efficiency. |
| <10 | Red | Average Fragment Size values this low typically mean that the SQL Server startup option **–E** has not been set or is not being recognized at startup. |

**Table 6:** Average Fragment Size values

Finally, it is important to note that Average Fragment Size results should not be evaluated for tables or partitions that are smaller than 500 MB. Small data structures simply do not have enough pages in total to achieve highly efficient fragment counts. Furthermore, these smaller data structures typically represent relatively small data requests and have limited impact on overall system I/O efficiency. Best results are often seen by managing only the largest, most frequently accessed tables in a data warehouse environment.

#### 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 because doing so can negate the benefits of large allocations. An index rebuild or the use of INSERT…SELECT to insert data 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. A MAXDOP value of 1 is ideal but can lead to very slow load rates. It is possible to set MAXDOP values as high as 8 in some circumstances. For more information, see the [Loading Data](#_Loading_Data_1) section of this document.

### Multiple Filegroups

Separate filegroups can be created to minimize logical fragmentation for 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 cases in which incremental data that mostly overlaps the existing clustered index key range is frequently loaded.
* 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. Additionally, very large fact tables can also be placed in separate filegroups.

## 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).

Another useful resource is the Fast Track 3.0 Data Load Best Practices guide. This Microsoft PowerPoint presentation can be found at the [SQL Server Fast Track DW Portal](http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx) (http://msdn.microsoft.com/en-us/library/dd425070.aspx). While initially based on SQL Server 2008 R2, this document remains applicable to SQL Server 2012.

### 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

Create a staging table that matches the structure (including partitioning) of the destination table:

* Perform a serial or multithreaded bulk insert into the empty clustered index staging table using moderate, nonzero batch size values to avoid spilling sorts to **tempdb**. Best performance will be achieved with some level of parallelism. The goal of this step is performance; therefore the page splits and logical fragmentation induced by parallel and/or concurrent inserts are not a concern.
* Insert from the staging table into the destination clustered index table using a single INSERT…SELECT statement with a MAXDOP value of 1. MAXDOP 1 ensures minimal [extent fragmentation](#_Extent_Fragmentation_1) but often sacrifices performance. MAXDOP settings of up to 8 can be used to increase load performance but will show increasing extent fragmentation as parallelism increases. The effective balance in this trade-off is best evaluated on a case-by-case basis.

##### Option 3

This option requires the use of two filegroups and two or more tables. The approach requires a partitioned cluster index table and is best suited for tables that see high levels of logical fragmentation in the most current partitions with little to no change activity to older partitions. The overall goal is to place volatile partitions in a dedicated filegroup and age or “roll” those partitions to the static filegroup after they stop receiving new records or changes to existing records:

* Create two filegroups, following FTDW guidance. One will be dedicated to volatile partitions and the other to static partitions. A *volatile partition* is one in which more than 10 percent of rows will change over time. A *static partition* is one that is not volatile.
* Build the primary cluster index partitioned table in the static filegroup.
* Build a table consistent with one of the following two general approaches:
  + A single heap table with a constraint that mirrors the partition scheme of the primary table. This constraint should represent the volatile range of the primary data set and may span one or more partition ranges of the primary table scheme. This is most useful if initial load performance is the key decision criteria because loads to a heap are generally more efficient than loads to a cluster index.
  + A single cluster index table with a partition scheme that is consistent with the primary table partition. This allows direct inserts with low degree of parallelism (DOP) to the primary table as volatile partitions age. After they are aged via insert to the primary table, partitions are dropped and new ranges added.
* Build a view that unions both tables together. This presents the combination of the two tables as a single object from the user perspective.
* After the volatile data ranges become static from a change-data perspective, use an appropriate aging process such as partition switch:
  + If a heap table with constraint is used; move data by partition range to the static filegroup via insert to staging table. Use CREATE INDEX and partition switching to move the data into the primary table. For more information about this type of operation for FTDW configurations, see the [Data Migration](#_Data_Migration) section of this document.
  + If a partitioned clustered index is used; Use a DOP that is less than or equal to 8. Next, INSERT restricted by partition range directly into the primary table. You may need to set the DOP as low as 1 to avoid fragmentation depending on overall system concurrency.

### 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 earlier for incremental load processing.

#### Clustered Index Load Process

Several 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 of identical structure. Use moderate, nonzero batch size to keep sorts in memory. 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

Use partition switch operations in a multi-step process that generally provides the best results for large load operations. This approach adds more complexity to the overall process and is designed to demonstrate an approach that is optimal for raw load performance. The primary goal of this approach is to enable parallel write activity at all phases of the insert to cluster index operation without introducing logical fragmentation. This is achieved by staging the table across multiple filegroups prior to inserting the data into the final destination table.

1. Identify the partition scheme for the final, destination cluster index table.
2. Create a stage filegroup.
3. Create an uncompressed, nonpartitioned heap “base” staging table in the stage filegroup.
4. Bulk insert data using WITH TABLOCK to the base staging table. Multiple, parallel bulk-copy operations are the most efficient approach if multiple source files are an option. The number of parallel load operations to achieve maximum throughput is dependent on server resources (CPU and memory) and the data being loaded.
5. Identify the number of primary filegroups to be supported. This number should be a multiple of the total number of partitions in the destination table. The number also represents the total number of INSERT and CREATE INDEX operations to be executed concurrently in later steps. As an example, for a table with 24 partitions and a server with eight cores, a database with eight primary filegroups would be indicated. This configuration allows execution of eight parallel inserts in the next steps, one for each of the eight primary filegroups and CPU core. Each filegroup, in this case, would contain three partition ranges worth of data.
6. Create the number of primary filegroups as determined earlier.
7. Create one staging heap table in each primary filegroup for each partition range, with no compression. Create a constraint on the staging table that matches the corresponding partition range from the destination table. Using the example given earlier, there would be three staging tables per primary filegroup created in this step.
8. Create the destination, partitioned cluster index table with page compression. This table should be partitioned across all primary filegroups. Partitions should align with the heap staging table constraint ranges.
9. Execute one INSERT or SELECT statement from the base staging table to the staging filegroup tables for each primary filegroup. This should be done in parallel. Be sure that the predicate for the INSERT or SELECT statement matches the corresponding partition ranges. Never run more than one INSERT or SELECT statement per filegroup concurrently.
10. Execute one CREATE CLUSTERED INDEX command with page compression per filegroup for the newly populated staging tables. This can be done in parallel but never with DOP higher than 8. Never run more than one create index per filegroup concurrently. Be sure to use the SORT\_IN\_TEMPDB option whenever performing a CREATE INDEX operation to avoid fragmenting the primary filegroups. The optimal number of concurrent create index operations will depend on server size, memory, and the data itself. In general, strive for high CPU utilization across all cores without oversubscribing (85-90 percent total utilization).
11. Execute serial partition switch operations from the staging tables to the destination table. This can also be done at the completion of each staging CREATE INDEX operation.

# Benchmarking and Validation

This section provides a basic description of the processes used to design and qualify SQL Server FTDW reference architectures. The goal of providing this information is to support user-defined or custom reference architectures based on FTDW methodology. For benchmarking, troubleshooting, or verification of published and prevalidated partner reference architectures, contact the publishing partner (H-P, Dell, EMC, IBM, Cisco, and others).

The process for FTDW validation can be divided into the two categories described here.

#### Baseline Hardware Validation

The goal of hardware validation is to establish real, rather than rated, performance metrics for the key hardware components of the Fast Track reference architecture. This process determines actual baseline performance characteristics of key hardware components in the database stack.

#### Fast Track Database Validation

Establishing SQL Server performance characteristics, based on a FTDW workload, allows for comparison against the performance assumptions provided by the baseline hardware evaluation process. In general, database workload throughput metrics should reflect at least 80 percent of baseline rates for Fast Track validated reference architectures. The performance metrics calculated in this process are the basis for published FTDW performance values and are based on concurrent SQL query workloads executed through the Fast Track Reference Point benchmarking tool.

Reference Point is a Microsoft software tool distributed to Fast Track hardware partners and is the sole infrastructure through which an official Fast Track reference architecture can be validated and approved by Microsoft. The tool instantiates a reference database schema and drives multiple concurrent query workloads designed to identify bottlenecks and establish key system performance measures.

#### Fast Track Validation with xVelocity Memory-optimized Columnstore Indexes

SQL Server 2012 implements columnstore index technology as a nonclustered indexing option for pre-existing tables. Individual queries may or may not use columnstore index optimization plans depending on query structure. This means that the mix of traditional row and new columnar query plans for a FTDW environment at any given time cannot be predicted.

For these reasons, FTDW for SQL Server 2012 system design and validation is based on non-columnstore-index benchmarks. FTDW systems are designed to run effectively in the case that no columnar optimization is achieved for any given period of time. Significant performance gains are often achieved when columnstore index query plans are active and this performance can be viewed as incremental to the basic system design.

Fast Track for SQL Server 2012 partner-validated reference architectures do publish an additional logical throughput rating for columnstore-index-enhanced benchmarks, and these numbers can be used to approximate the positive impact to query performance customers can expect under concurrent query workload. These numbers are based on the same FTDW benchmarks and schema used for all system validations.

## Performing Baseline FTDW Validation

Baseline validation 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. The goal is to ensure that hardware and operating system configuration are accurate and deliver expected results based on design and development benchmarks.

Windows Server Performance and Reliability Monitor (also known as 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/sqlcat/b/whitepapers/archive/2007/11/21/predeployment-i-o-best-practices.aspx).

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

### Baseline 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 -o120 -b512 d:\iobw.tst –t1

In this case R indicates a read test, 30 is the test duration in seconds, 120 is the number of outstanding requests issued, 512 is the block size in kilobytes of requests made, d:\iobw.tst is the location of the test file, and 1 is the number of threads.

In order to test aggregate bandwidth scenarios, multiple SQLIO tests must be issued in parallel. A single instance of SQLIO should be used for each primary data mount point (disk volume). Parallelization of the SQLIO instances can be achieved using Windows PowerShell or other scripting methods. For FTDW partner-validated reference architectures, baseline I/O validation scripts may be available from the partner.

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.

#### Step 1 - Validate I/O Bandwidth

The first step in validating a FTDW configuration is to determine the maximum aggregate throughput that can be realized between the storage I/O network and the server. 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 at least two read threads per file. Be sure to calculate aggregate outstanding reads. For example, 2 read threads with 50 outstanding requests would account for 100 total outstanding requests to the target LUN.
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 Multi-Path Fibre Channel paths.

#### Step 2 - Validate LUN/Volume Bandwidth

This test is similar to the previous test. 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 volume, for example, 25 GB per volume. Similar parameters should be used for SQLIO as described in step 1.

Large block (512 KB) sequential reads should be issued against the test files on each volume. 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 volume individually and then test the two simultaneously. Disk group throughput varies by storage vendor and configuration but comparison can always be made to single HDD read rates. A 4-disk RAID1+0 disk group, for example, could achieve a peak read rate of nearly four times the single HDD read rate for this type of basic read pattern. RAID 1 or 1+0 performance can vary by storage product as some vendors technology allows for “mirrored read”, which enables I/Os to be serviced from both sides of the mirrored pair when contiguous requests are received.

#### Step 3 - Validate Aggregate Bandwidth

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

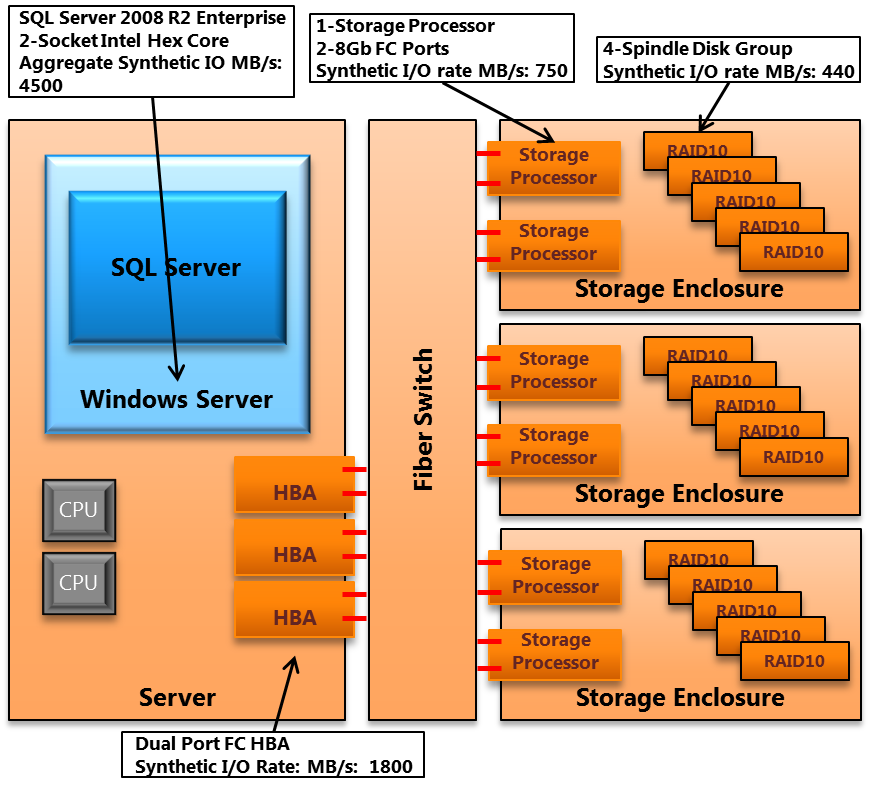
The results of this test illustrate the maximum aggregate throughput achievable when reading data from the physical disks.

Data is read from the large data file, as in the previous test, on each volume simultaneously.

Aggregate performance from disk should be in the region of 80 percent to 90 percent of the aggregate storage I/O bandwidth, for balanced FTDW systems.

#### Component Ratings

The following diagram illustrates synthetic benchmark results that are consistent with values seen in similar Fast Track reference architectures.



**Figure 4:** Example of synthetic benchmark realized bandwidth for a 2-socket, 12-core server with 3 8Gbps dual-port HBA cards, with 12 4-disk RAID1+0 primary data volume

#### Summary

Baseline hardware 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.

## Performing Fast Track Database Benchmark

This phase of FTRA evaluation measures SQL Server performance for the FTDW workload in terms of two core metrics. The first, Maximum CPU Consumption Rate (MCR), is a measure of peak I/O processing throughput. The second, Benchmark CPU Consumption Rate (BCR) is a measure of actual I/O processing throughput for a query or a query-based workload.

#### What Is MCR?

The MCR calculation provides a per-core I/O throughput value in MB or GB per second. This value is measured by executing a predefined, read-only, nonoptimized query, from buffer cache and measuring the time to execute against the amount of data in MB or GB. Because MCR is run from cache it represents the peak nonoptimized scan rate achievable through SQL Server for the system being evaluated. For this reason MCR provides a baseline peak rate for initial design purposes. It is not intended to indicate average or expected results for a real workload. Validated FTDW architectures will have aggregate baseline I/O throughput results that are at least 100 percent of the server-rated MCR. Another way to explain this is that MCR represents the best possible SQL Server processing rate for a reasonable, worst-case workload.

MCR can also be used as a frame of reference when comparing other published and validated FTDW reference architectures for SQL Server 2012.

In summary:

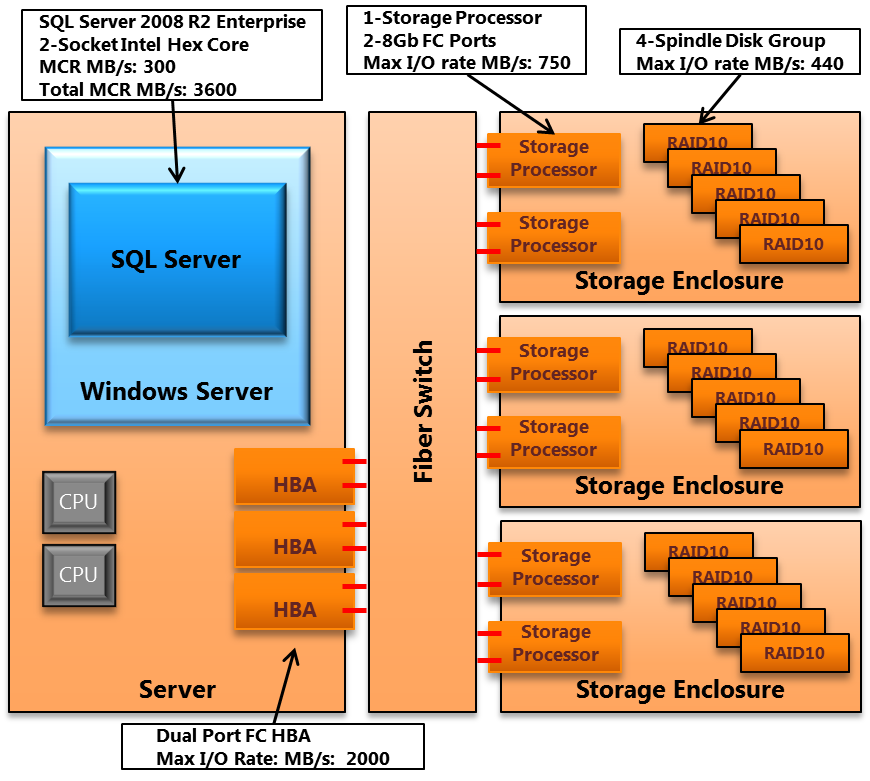
* MCR is not definitive of actual results for a customer workload.
* MCR provides a maximum data processing rate baseline for SQL Server and a single query associated with the Fast Track workload.
* MCR is specific to a CPU and server. In general, rates for a given CPU do not vary greatly by server and motherboard architecture but final MCR should be determined by actual testing.
* MCR throughput rating can be used as a comparative value against existing, published FTDW reference architectures. This can assist with hardware selection prior to component and application testing.

### Calculating MCR

A baseline CPU consumption rate for the SQL Server application is established by running a standard SQL query defined for the FTDW program. This query is designed to be a relatively simple representation of a typical query for the workload type (in this case data warehouse) and is run from buffer cache. The resulting value is specific to the CPU and the server the query is being executed against. Use the following method to calculate MCR:

1. Create a reference dataset based on the TPC-H lineitem table or similar data set. The table should be of a size that it can be entirely cached in the SQL Server buffer pool yet still maintain a minimum one-second execution time for the query provided here.
2. For FTDW the following query is used: *SELECT sum([integer field]) FROM [table] WHERE [restrict to appropriate data volume] GROUP BY [col].*
3. The environment should:
   * Ensure that Resource Governor settings are at default values.
   * Ensure that the query is executing from the buffer cache. Executing the query once should put the pages into the buffer, and subsequent executions should read fully from buffer. Validate that there are no physical reads in the query statistics output.
   * Set STATISTICS IO and STATISTICS TIME to ON to output results.
4. Run the query multiple times, at MAXDOP = 4.
5. Record the number of logical reads and CPU time from the statistics output for each query execution.
6. Calculate the MCR in MB/s using the formula:  
    ( [Logical reads] / [CPU time in seconds] ) \* 8KB / 1024
7. A consistent range of values (+/- 5%) should appear over a minimum of five query executions. Significant outliers (+/- 20% or more) may indicate configuration issues. The average of at least 5 calculated results is the FTDW MCR.

Based on the MCR calcuation, a component architecture throughput diagram can be constructed. For the purposes of system MCR evaluation, component throughput is based on vendor-rated bandwidth. This diagram can be useful for system design, selection, and bottleneck analysis. Figure 5 illustrates an example of this.



**Figure 5**: Example of Maximum CPU Consumption Rate (MCR) and rated component bandwidth for a 2-socket, 12-core server based on Intel Westmere CPUs

For more information about measuring MCR, see [Workload Testing](#_Workload_Testing) in the appendix.

### Calculating BCR

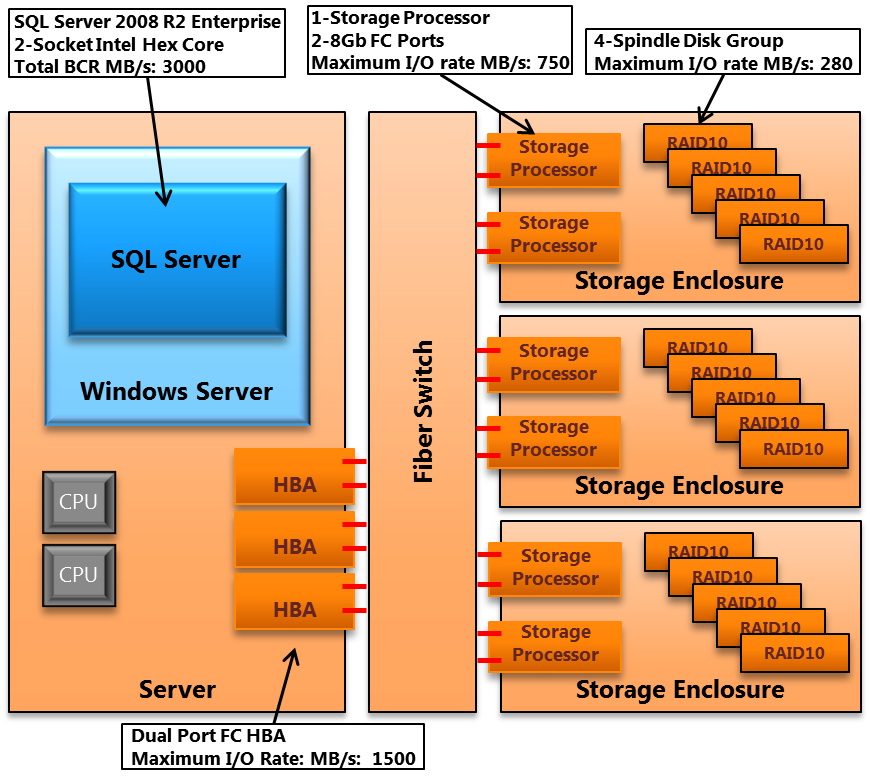
A benchmark CPU consumption rate for the SQL Server application is established by running a baseline set of SQL queries, under an appropriate level of concurrency, that are specific to your data warehouse workload. The number of queries and level of concurrency used depends entirely on the expected use case. The query workload should be serviced from disk, not from the SQL Server buffer pool as with MCR. The resulting value is specific to the CPU, the server, and the workload it is being executed against. The appendix entry for [Workload Testing](#_Workload_Testing) provides a more detailed example of creating a BCR workload benchmark.

Use the following method to calculate BCR:

1. Create a reference dataset that contains at least one table. The table should be of significant enough 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. The basic query form for FTDW is the following: *SELECT sum([integer field]) FROM [table] WHERE [restrict to appropriate data volume] GROUP BY [col].* This can be used as a starting point for query workload design if customer queries are not readily available. TPC-H is another commonly used query benchmark that can be used as a reference query set.
3. For a FTDW customer benchmark it is always ideal to choose a queries that are representative of the target workload. Queries should be scheduled across multiple concurrent sessions that are representative of peak historical or projected activity for the customer environment. The following criteria can be considered in query selection:
   * 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.
   * The query should not cause writes of data to **tempdb** unless this characteristic is a critical part of the target workload.
   * The query should return minimal rows. The SET ROWCOUNT option can be used to manage this. A ROWCOUNT value greater than 100 should be used (105 is standard for Fast Track benchmarking). Alternatively aggregation can be used to reduce records returned from large unrestricted scans.
4. The environment should:
   * Ensure that 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 output results.
5. Run the query or workload multiple times, starting at MAXDOP 8. 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:  
      ( [Logical reads] / [CPU time in seconds] ) \* 8KB / 1024
   * This gives you a range for BCR. If multiple queries are used, use a weighted average to determine BCR.

#### BCR Results

Figure 6 illustrates SQL Server workload based benchmark results that are consistent with values seen in similar Fast Track Data Warehouse reference architectures.



**Figure 6:** Example of synthetic benchmark realized bandwidth for a 2-socket, 12-core server with 3 8Gbps dual-port HBA cards, with 12 4-disk RAID1+0 primary data LUN

#### Interpreting BCR

If your BCR for the average query is much lower than the standard MCR benchmarked for the FTRA, 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, or increasing the size of the disks – these steps can help reduce the cost of the storage infrastructure to a balanced level. Alternatively you might think about using a higher socket count server or higher-performance CPUs that can take advantage of the surplus of storage I/O throughput. In either case the goal is to balance database processing capability with storage I/O throughput.

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

# Published FTDW Reference Architectures

Detailed hardware reference architecture specifications are available from each participating Fast Track Data Warehouse partner. For more information, including links to each partner, see [Fast Track Data Warehousing](http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx) (http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx).

FTDW capacity evaluated by the estimated the amount of noncompressed user data files that can be loaded to the database. This is called User Data Capacity (UDC). This calculation assumes that page compression is enabled for all tables and that data volumes will be mirrored. An average compression factor of 3.5:1 is used. In addition an allocation of up to 30 percent of noncompressed capacity is allocated to **tempdb** before calculating UDC. Note that for larger configurations with more total capacity this ratio is reduced to as low as 20 percent.

For more information about **tempdb** sizing, see [Capacity Planning for tempdb](http://msdn.microsoft.com/en-us/library/ms345368.aspx) (http://msdn.microsoft.com/en-us/library/ms345368.aspx).

# 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:**

[SQL Server Website](http://www.microsoft.com/sqlserver/)

[SQL Server Fast Track Website](http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx)

[SQL Server TechCenter](http://technet.microsoft.com/en-us/sqlserver/)

[SQL Server Online Resources](http://msdn.microsoft.com/en-us/sqlserver/)

[Top10 Best Practices for Building Large Scale Relational Data Warehouses](http://go.microsoft.com/fwlink/?LinkId=141862) (SQLCAT team)

[How to: Enable the Lock Pages in Memory Option (Windows)](http://go.microsoft.com/fwlink/?LinkId=141863)

[Tuning options for SQL Server 2005 and SQL Server 2008 for high performance workloads](http://go.microsoft.com/fwlink/?LinkId=141864)

[How to: Configure SQL Server to Use Soft-NUMA](http://go.microsoft.com/fwlink/?LinkId=141865)

[Database File Initialization](http://go.microsoft.com/fwlink/?LinkId=141866)

[How to: View or Change the Recovery Model of a Database (SQL Server Management Studio)](http://go.microsoft.com/fwlink/?LinkId=141867)

[Monitoring Memory Usage](http://go.microsoft.com/fwlink/?LinkId=141868)

[Troubleshooting Storage Area Network (SAN) Issues](http://go.microsoft.com/fwlink/?LinkId=141870)

[Installing and Configuration MPIO](http://technet.microsoft.com/en-us/library/ee619752(v=WS.10).aspx)

[SQL Server 2000 I/O Basics White Paper](http://go.microsoft.com/fwlink/?LinkId=141876)

[Data Compression: Strategy, Capacity Planning and Best Practices](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](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20Fast%20Track%20Data%20Warehouse%20Reference%20Guide).

# Appendix

## FTDW System Sizing Tool

The FTDW System Sizing tool is a spreadsheet calculator that assists you with the process of calculating a customer workload requirement in terms of FTDW system throughput. You can use this tool in the absence of a test platform or as a starting point when evaluating customer requirements. The tool can be found at [Fast Track Data Warehousing](http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx) (http://www.microsoft.com/sqlserver/en/us/solutions-technologies/data-warehousing/fast-track.aspx). Additionally, some partner vendors have built their own conforming Fast Track sizing tools. These can be found at the partner websites.

## Validating a User-Defined FTRA

### Synthetic I/O Testing

SQLIO is a tool available for download from Microsoft that enables you to test the I/O subsystem independently of SQL Server.

### Generating Test Files with SQLIO

When you run SQLIO, it creates an appropriate test file, if such a file 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 the following:

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 create files in parallel, create multiple parameter files and then run multiple SQLIO sessions concurrently.

#### Validate Storage Bandwidth (from Cache)

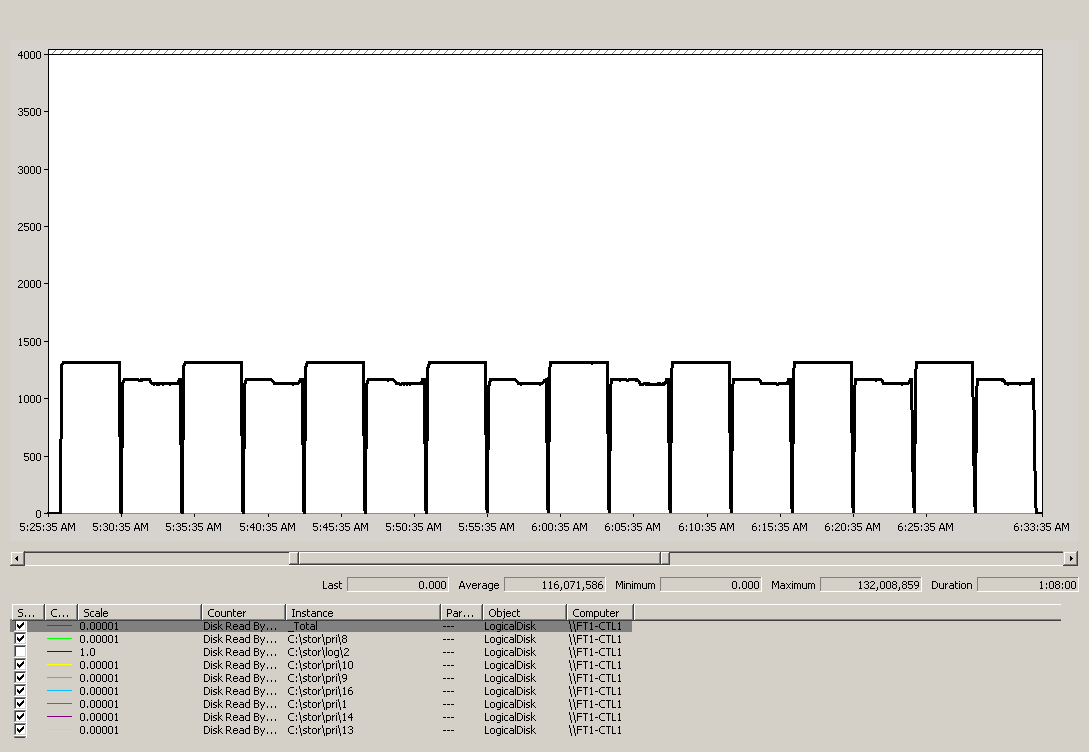
Using a small test file with a read duration of several minutes ensures that the file is completely resident in the array cache. Figure 7 shows the **Logical Disk > Read Bytes / sec** counter against the disks in an example Fast Track system at various outstanding request numbers and block size. Tests 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 four requests per file. Each individual disk should contribute to the total bandwidth.



**Figure 7:** Logical Disk > Read Bytes / sec counter

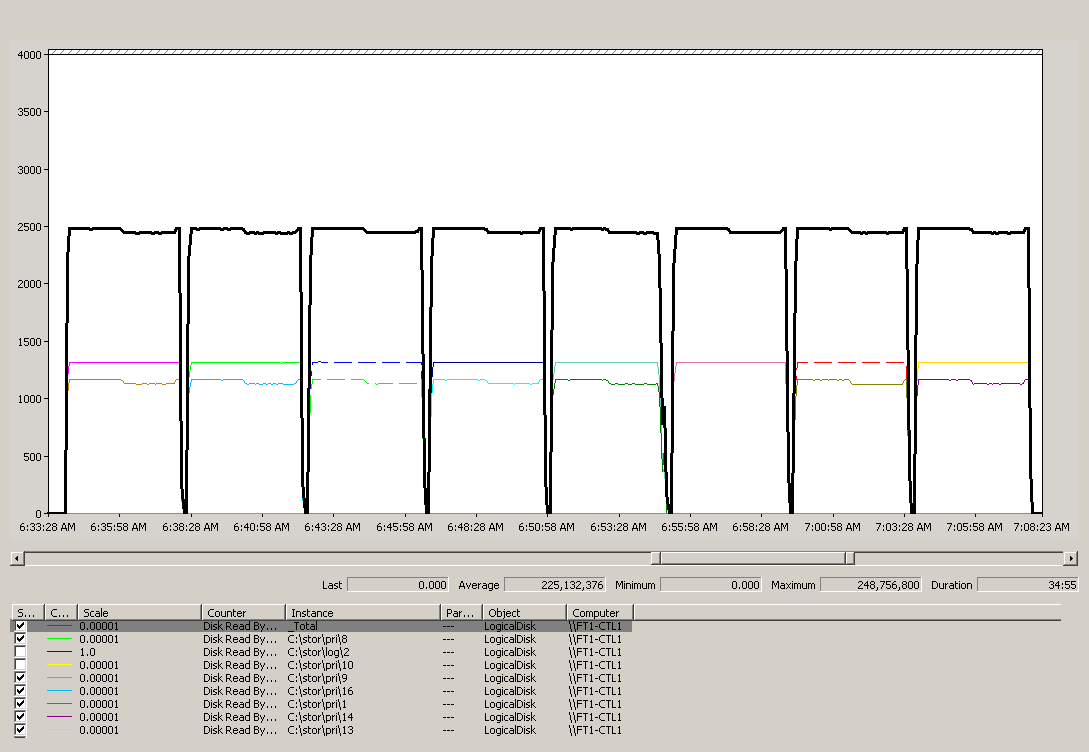
#### Validate LUN/VolumeBandwidth (from Disk)

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 volume, one at a time. You may see that some of the LUNs appear to be slightly faster than others. This is not unusual but differences greater than 15 percent should be examined.



**Figure 8:** Validating LUN\Volume and RAID pair bandwidth

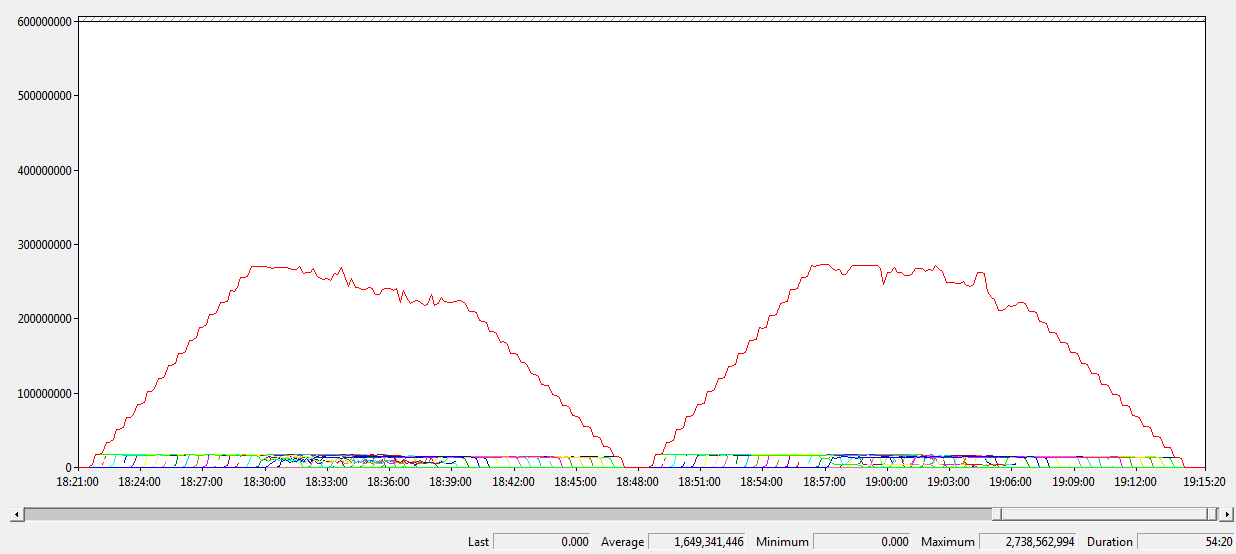
Run simultaneous tests against one or more volume that share the same disk group. The following picture shows the output of tests against 8 disk groups.



**Figure 9:** Testing LUNs that share disk groups

#### Validate Aggregate Bandwidth (from Disk)

The following test demonstrates the effect of stepping up the I/O throughput, adding in an additional volume 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 based on the TPC-H lineitem table or similar data set. The table should be of a size that it can be entirely cached in the SQL Server buffer pool yet still maintain a minimum 2 second execution time for the query provided here.
2. For FTDW the following query is used: *SELECT sum([integer field]) FROM [table] WHERE [restrict to appropriate data volume] GROUP BY [col].*
3. The environment should:
   * Ensure that Resource Governor is set to default values.
   * Ensure that the query is executing from the buffer cache. Executing the query once should put the pages into the buffer and subsequent executions should read fully from buffer. Validate that there are no physical reads in the query statistics output.
   * Set STATISTICS IO and STATISTICS TIME to ON to output results.
4. Run the query multiple times, at MAXDOP = 4.
   * Record the number of logical reads and CPU time from the statistics output for each query execution.
   * Calculate the MCR in MB/s using the formula:  
      ( [Logical reads] / [CPU time in seconds] ) \* 8KB / 1024
   * A consistent range of values (+/- 5%) should appear over a minimum of five query executions. Significant outliers (+/- 20% or more) may indicate configuration issues. The average of at least 5 calculated results is the FTDW MCR.

### 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, an 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 FTDW system, with 16 data LUNs.

CREATE DATABASE FT\_Demo ON PRIMARY Filegroup FT\_Demo

( NAME = N 'FT\_Demo\_.mdf' , FILENAME = N'C:\FT\PRI\SE1-SP1-DG1-v1' , SIZE = 100MB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v1.ndf' , FILENAME = N'C:\FT\PRI\SE1-SP1-DG1-v1' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v2.ndf' , FILENAME = N'C:\FT\PRI\SE1-SP1-DG2-v2' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v3.ndf' , FILENAME = N'C:\FT\PRI\SE1-SP2-DG3-v3' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v4.ndf' , FILENAME = N'C:\FT\PRI\SE1-SP2-DG4-v4' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v6.ndf' , FILENAME = N'C:\FT\PRI\SE2-SP1-DG6-v6' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v7.ndf' , FILENAME = N'C:\FT\PRI\SE2-SP1-DG7-v7' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v8.ndf' , FILENAME = N'C:\FT\PRI\SE2-SP2-DG8-v8' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v9.ndf' , FILENAME = N'C:\FT\PRI\SE2-SP2-DG9-v9' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v11.ndf' , FILENAME = N'C:\FT\PRI\SE3-SP1-DG11-v11' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v12.ndf' , FILENAME = N'C:\FT\PRI\SE3-SP1-DG12-v12' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v13.ndf' , FILENAME = N'C:\FT\PRI\SE3-SP2-DG13-v13' , SIZE = 417GB , FILEGROWTH = 0 ),

( NAME = N 'FT\_Demo\_v14.ndf' , FILENAME = N'C:\FT\PRI\SE3-SP2-DG14-v14' , SIZE = 417GB , FILEGROWTH = 0 ),

LOG ON

( NAME = N 'FT\_LOG\_v5.ldf' , FILENAME = N 'C:\FT\LOG\SE1-SP2-DG5-v5' , SIZE = 100GB , MAXSIZE = 500GB , FILEGROWTH = 50 )

GO

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Configure recommended settings\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALTER DATABASE FT\_Demo SET AUTO\_CREATE\_STATISTICS ON

GO

ALTER DATABASE FT\_Demo SET AUTO\_UPDATE\_STATISTICS ON

GO

ALTER DATABASE FT\_Demo SET AUTO\_UPDATE\_STATISTICS\_ASYNC ON

GO

ALTER DATABASE FT\_Demo SET RECOVERY SIMPLE

GO

sp\_configure 'show advanced options', 1

go

reconfigure with override

go

/\*\*\*\*\*\*\*\*Make sure all tables go on our filegroup and not the Primary filegroup\*\*\*\*/

ALTER DATABASE FT\_Demo

MODIFY FILEGROUP FT\_Demo

DEFAULT

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\_linenumber 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 FT\_Demo

GO

CREATE CLUSTERED INDEX cidx\_lineitem

ON lineitem(l\_shipdate ASC)

WITH( SORT\_IN\_TEMPDB = ON

, DATA\_COMPRESSION = PAGE

)

ON FT\_Demo

GO

#### Loading Data for BCR Measurement

As described earlier 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

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. Note that this example is not indicative of performance that will be achieved on any given system. BCR numbers are unique to the system, schema size, data types, query structure, and statistics to name a few of many variables.

|  |  |
| --- | --- |
| Query complexity | Per-core BCR  (*Page Compressed* ) at MAXDOP 4 |
| Simple | 201 MB/s |
| Average | 83 MB/s |
| Complex | 56 MB/s |

**Table 7:** Examples of benchmarks

**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 4)

**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 4)

**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 4)

### 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.