



**Analysis and Optimization of
Data Storage
using Enhanced Object Models
in the .NET Framework**

ASHISH TANDON

**Submitted in partial fulfilment of the requirements of
Napier University for the degree of
Master of Science with Advanced Software Engineering**

**Napier University
School of Computing
September 2007**

Authorship Declaration

I, **Ashish Tandon**, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed;
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;
3. I have acknowledged all main sources of help;
4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;
5. I have read and understand the penalties associated with Academic Misconduct.
6. I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date:

Matriculation no:

Data Protection declaration

Under the 1998 Data Protection Act we cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name against *one* of the options below to state your preference.

	The University may make this dissertation, with indicative grade, available to others.
	The University may make this dissertation available to others, but the grade may not be disclosed.
	The University may not make this dissertation available to others.

Signature

(you must sign and date this page)

Date

Abstract

The purpose of this thesis is to benchmark the database to examine and analyze the performance using the Microsoft COM+ the most commonly used component framework heavily used for developing component based applications. The prototype application based on Microsoft Visual C#.NET language used to benchmark the database performance on Microsoft .NET Framework environment 2.0 and 3.0 using the different sizes of data range from low (100 Rows) to high volume (10000 Rows) of data with five or ten number of users connections. There are different type of application used like COM+, Non-COM+ and .NET based application to show their performance on the different volume of data with specified numbers of user on the .NET Framework 2.0 and 3.0.

The result has been analyzed and collected using the performance counter variables of an operating system and used Microsoft .NET class libraries which help in collecting system's level performance information as well. This can be beneficial to developers, stakeholders and management to decide the right technology to be used in conjunction with a database. The results and experiments conducted in this project results in the substantial gain in the performance, scalability and availability of component based application using the Microsoft COM+ features like object pooling, application pooling, role-based, transactions isolation and constructor enabled.

The outcome of this project is that Microsoft COM+ component based application provides optimized database performance results using the SQL Server. There is a performance gain of at least 10% in the COM+ based application as compared to the Non COM+ based application. COM+ services features come at the performance penalty. It has been noticed that there is a performance difference between the COM+ based application and the application based on role based security, constructor enable and transaction isolation of around 15%, 20% and 35% respectively. The COM+ based application provides performance gain of around 15% and 45% on the low and medium volume of data on a .NET Framework 2.0 in comparison to 3.0. There is a significant gain in the COM+ Server based application on .NET Framework 3.0 of around 10% using high volume of data. This depicts that high volume of data application works better with Framework 3.0 as compared to 2.0 on SQL Server.

The application performance type results represents that COM+ component based application provides better performance results over Non-COM+ and .NET based application. The difference between the performance of COM+ application based on low and medium volume of data was around 20% and 30%. .NET based application performs better on the high volume of data results in performance gain of around 10%.

Similarly more over the same results provided on the test conducted on the MS Access. Where COM+ based application running under .NET Framework 2.0 performs better result other than the Non-COM+ and .NET based application on a low and medium volume of data and .NET Framework 3.0 based COM+ application performs better results on high volume of data.

Contents

Authorship Declaration.....	2
Abstract.....	3
Contents	4
List of Figures.....	8
List of Charts	9
List of Tables.....	10
Acknowledgements.....	11
1 Introduction	12
1.1 Project Overview	12
1.2 Background.....	13
1.3 Aims and Objectives	13
1.4 Thesis Structure.....	14
2 Theory.....	15
2.1 Introduction.....	15
2.2 COM	15
2.3 Microsoft® Transaction Server (MTS).....	15
2.4 COM+	16
2.4.1 Object Pooling	16
2.4.2 Just In Time Compiler (JIT).....	16
2.5 .NET Framework	17
2.5.1 .NET Framework 1.1	17
2.5.2 .NET Framework 2.0	18
2.5.3 .NET Framework 3.0	18
2.6 SQL Server 2005.....	18
2.7 C# .NET 2005	18
2.8 Conclusion	20
3 Literature Review	21
3.1 Introduction.....	21
3.2 Importance of Data Storage	21
3.3 Microsoft COM+	22
3.4 COM+ Services.....	23
3.4.1 JITA	23
3.4.2 Transactions	26
3.4.3 Object Pooling	26
3.4.4 Transaction Scenario.....	29

3.5	Conclusion	29
4	Design.....	30
4.1	Introduction.....	30
4.2	Requirement and Analysis	30
4.3	Interface design.....	31
4.4	Analysis of Development environment.....	33
4.5	Analysis of Database.....	33
4.6	Project and Classes Implementation	33
4.6.1	Client Application	33
4.6.2	COM Access	33
4.6.3	COM SQL.....	34
4.6.4	NoOMAccess.....	34
4.6.5	NoCOM SQL.....	34
4.7	Evaluation design.....	34
	Experiment 1:.....	34
	Experiment 2:.....	34
	Experiment 3:.....	34
	Experiment 4:.....	34
	Experiment 5:.....	35
	Experiment 6:.....	35
	Experiment 7:.....	35
	Experiment 8:.....	35
	Experiment 9:.....	35
	Experiment 10:.....	35
	Experiment 11:.....	35
	Experiment 12:.....	35
	Experiment 13:.....	35
	Experiment 14:.....	35
	Experiment 15:.....	36
	Experiment 16:.....	36
	Experiment 17:.....	36
	Experiment 18:.....	36
	Experiment 19:.....	36
	Experiment 20:.....	36
	Experiment 21:.....	36
	Experiment 22:.....	36
	Experiment 23:.....	36

Experiment 24:.....	36
Experiment 25:.....	36
Experiment 26:.....	37
Experiment 27:.....	37
Experiment 28:.....	37
Experiment 29:.....	37
Experiment 30:.....	37
Experiment 31:.....	37
4.8 Conclusion	37
5 Implementation	38
5.1 Introduction.....	38
5.2 Configuration Information	38
5.3 Pooled Component Implementation.....	39
5.4 Non Pooled Component Implementation.....	39
5.5 JITA.....	40
5.6 Matrix Implementation	40
5.7 Calculation of Median and Standard Deviation	41
5.8 Data Grid Implementation	42
5.9 Dynamic Query	42
5.10 Testing Implementation.....	43
5.11 Conclusion	43
6 Evaluation.....	45
6.1 Introduction.....	45
6.2 Methodology	45
6.3 SQL Server Experiments Results.....	46
6.4 MS Access Experiments Results.....	65
6.5 Conclusion	74
7 Conclusion	75
7.1 Introduction.....	75
7.2 Conclusion	75
7.3 Critical Analysis.....	76
7.4 Future Work	77
8 References.....	78
Appendix 1	82
A. PrototypeApplication.cs.....	82
B. Assembly info [PrototypeApplication.cs]	96
C. COMAccess.cs	97

D. Assembly info [COMAccess.cs]	99
E. COMSQL.cs	100
F. Assembly info[COMSQL.cs].....	101
G. NoCOMAccess.cs	102
H. Assembly info [NoCOMAccess.cs]	103
I. NoCOMSQL.cs	104
J. Assembly info [NoCOMSQL.css].....	105
K. AppConfig.....	106
Appendix 2	107
A. GANTT Chart	107

List of Figures

Figure 1.1 Microsoft COM+ Evolution	12
Figure 2.1 Microsoft .NET Framework (Microsoft, 2007).	17
Figure 2.2 Microsoft C# Project life cycle (Visual C# Developer Center, 2007).	19
Figure 3.1 Microsoft COM+ TPC Performance Result	22
Figure 3.2 Microsoft COM+ Non JITA Performance Result	23
Figure 3.3 Microsoft COM+ JITA Performance Result	24
Figure 3.4 Microsoft COM+ and Enterprise Services Performance Result	25
Figure 3.5 Microsoft COM+ and ES Typical Method Performance Result	25
Figure 3.6 Microsoft COM+ and ES Typical Method [No Transaction Performance Result].	26
Figure 3.7 The life cycle of a component using JITA and object pooling (Löwy, 2001).	27
Figure 3.8 Pooled and Non Pooled Component Performance	28
Figure 4.1 Prototype application	32
Figure 4.2 Prototype application Database and COM+ option	32
Figure 4.3 Prototype application Data volume and Users option	32
Figure 5.1 Show Data in the DataGrid option.....	42
Figure 5.2 Show Data Volume and User option.....	43

List of Charts

Chart 6.3-1 COM+ Application Performance [No Object Pooling and JIT]	46
Chart 6.3-2 COM+ Application Performance [Object Pooling and JIT]	47
Chart 6.3-3 COM+ Application Performance	48
Chart 6.3-4 COM+ v/s Non COM+ Component Performance	49
Chart 6.3-5 COM+ Role Based Security Component Performance	50
Chart 6.3-6 COM+ Transaction Component Performance	51
Chart 6.3-7 COM+ Component features and their Performance	52
Chart 6.3-8 Non-COM+ component performance on SQL Server (5 Users)	53
Chart 6.3-9 Non-COM+ component performance on SQL Server (10 Users).....	54
Chart 6.3-10 COM+ component performance on SQL Server (5 Users).....	55
Chart 6.3-11 COM+ component performance on SQL Server (10 Users).....	56
Chart 6.3-12 .NET component performance on SQL Server (5 Users)	57
Chart 6.3-13 .NET component performance on SQL Server (10 Users)	58
Chart 6.3-14 Application performance on SQL Server (5 Users and 100 Rows)	59
Chart 6.3-15 Application performance on SQL Server (10 Users and 100 Rows)	60
Chart 6.3-16 Application performance on SQL Server (5 Users and 1000 Rows)	61
Chart 6.3-17 Application performance on SQL Server (10 Users and 1000 Rows)	62
Chart 6.3-18 Application performance on SQL Server (5 Users and 10000 Rows)	63
Chart 6.3-19 Application performance on SQL Server (10 Users and 10000 Rows)	64
Chart 6.4-1 Non-COM+ component performance on MS Access (5 Users)	65
Chart 6.4-2 Non-COM+ component performance on MS Access (10 Users)	66
Chart 6.4-3 COM+ component performance on MS Access (5 Users).....	66
Chart 6.4-4 COM+ component performance on MS Access (10 Users).....	67
Chart 6.4-5 .NET Application performance on MS Access (5 Users)	68
Chart 6.4-6 .NET Application performance on MS Access (10 Users)	68
Chart 6.4-7 Application performance on MS Access (5 Users and 100 Rows)	69
Chart 6.4-8 Application performance on MS Access (10 Users and 100 Rows)	70
Chart 6.4-9 Application performance on MS Access (5 Users and 100 Rows)	71
Chart 6.4-10 Application performance on MS Access (10 Users and 1000 Rows)	71
Chart 6.4-11 Application performance on MS Access (5 Users and 10000 Rows)	72
Chart 6.4-12 Application performance on MS Access (10 Users and 10000 Rows)	73

List of Tables

Table 1 Pooled Object Performance.....	28
Table 6.2.1 Experiment Matrix	45
Table 6.3.1 COM+ Application Performance Data [No Pooling and JIT].....	46
Table 6.3.2 COM+ Application Performance Data [Pooling and JIT].....	47
Table 6.3.3 COM+ v/s Non COM+ Component Data Performance	48
Table 6.3.4 COM+ v/s Non COM+ Component Data Performance	49
Table 6.3.5 COM+ Role Based Component Performance Data.....	50
Table 6.3.6 COM+ Transaction Component Performance Data	51
Table 6.3.7 COM+ Component features Data Performance	52
Table 6.3.8 Non-COM+ component Data performance on SQL Server (5 Users)	53
Table 6.3.9 Non-COM+ component Data performance on SQL Server (10 Users)	54
Table 6.3.10 COM+ component Data performance on SQL Server (5 Users).....	55
Table 6.3.11 COM+ component Data performance on SQL Server (10 Users).....	56
Table 6.3.12 .NET component data performance on SQL Server (5 Users)	57
Table 6.3.13.NET component data performance on SQL Server (10 Users)	58
Table 6.3.14 Application performance data on SQL Server (5 Users and 100 Rows)	59
Table 6.3.15 Application performance data on SQL Server (10 Users and 100 Rows)	60
Table 6.3.16 Application performance data on SQL Server (5 Users and 1000 Rows)	61
Table 6.3.17 Application performance data on SQL Server (10 Users and 1000 Rows)	62
Table 6.3.18 Application performance data on SQL Server (5 Users and 10000 Rows)	63
Table 6.3.19 Application performance data on SQL Server (10 Users and 10000 Rows)	64
Table 6.4.1 Non-COM+ component performance data on MS Access (5 Users)	65
Table 6.4.2 Non-COM+ component performance data on MS Access (10 Users)	66
Table 6.4.3 COM+ component performance data on MS Access (5 Users)	67
Table 6.4.4 COM+ component performance data on MS Access (10 Users)	67
Table 6.4.5 .NET Application performance data on MS Access (5 Users)	68
Table 6.4.6 .NET Application performance data on MS Access (10 Users)	69
Table 6.4.7 Application performance data of MS Access (5 Users and 100 Rows).....	69
Table 6.4.8 Application performance data of MS Access (10 Users and 100 Rows).....	70
Table 6.4.9 Application performance data of MS Access (5 Users and 1000 Rows).....	71
Table 6.4.10 Application performance data of MS Access (10 Users and 1000 Rows).....	72
Table 6.4.11 Application performance data of MS Access (5 Users and 10000 Rows).....	72
Table 6.4.12 Application performance data of MS Access (10 Users and 10000 Rows).....	73

Acknowledgements

William Buchanan

First and foremost, thanks to Professor Bill Buchanan, Napier University for giving me the opportunity to participate in this project under his valuable guidance. He also helped make this project better than what I had written, for which I will forever be in his debt.

Additional thanks to Lecturer Alistair Lawson and Dr. Emma Hart for their constant support and guidance during the academic year study.

Last but certainly not least I must thank my wife Ashima for providing unconditional support and encouragement throughout the project.

Ashish Tandon
1-Sep-07

1 Introduction

1.1 Project Overview

There are problems associated in the earlier Microsoft Windows development environments with the development and deployment of applications. The new framework platform which has been launched by Microsoft attempts to solve this problem. In the Microsoft development IDE versions, Microsoft Visual Studio 6.0 and earlier, it was difficult and require lot of time to write a code to write a Java or C++ class, and to derive, or to use it directly in the Visual Basic code. Microsoft solved this problem by creating the Component Object Model (COM) which allows compiled components to communicate with each other, over a binary language.

Unfortunately COM has its own defects and there is no way in which COM technologies which allows the components to be managed and discovered during the runtime. The .NET Framework solves this problem using the concept know as reflection, or also solves the error handling issues which came across while making an API call, the API might raise an error, or might return an error code. If the error code is being returned, the calling component must have the knowledge of the known errors. (Bayer, 2001)The .NET Framework solves this problem as it raises exceptions for the all errors. It also provides the low - level features which were difficult to write and requires more time to code using the earlier development versions like COM+ Object Pooling; Role based security; access to SMTP, HTTP and FTP. This can be possible now for the Visual Studio .NET developers. (Figure 1.1)

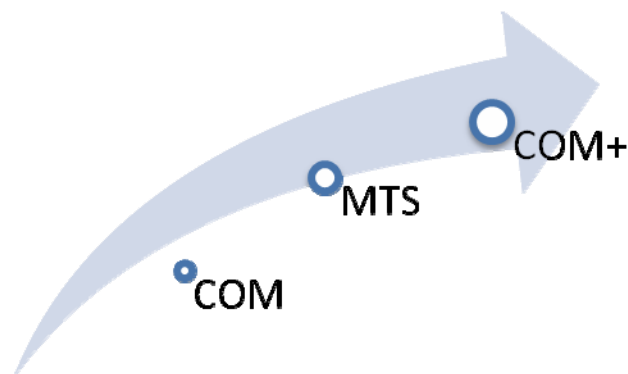


Figure 1.1 Microsoft COM+ Evolution

MTS is the extended version of COM and it is an important feature of the Microsoft Windows NT® operating system that simplifies the development and deployment of server centric applications built using Microsoft Component Object Model (COM) technologies. The thesis uses COM+ features, in conjunction with a data source and .NET Framework. Database performance is benchmarked on low and high volumes

of data in database using variation of COM+ application components and COM+ settings under the Microsoft .NET Framework environment.

1.2 Background

We can say that Microsoft's initiative in middleware started with its introduction of MTS (Microsoft Transaction Server). MTS provides us many useful features for developing and deploying multi-tier enterprise applications. However, the name MTS is somewhat confusing, because it does a lot more than transaction handling. That is why when Microsoft released Windows 2000, it bundled COM+ with it, included within it all the useful features of COM and MTS. And unlike MTS, COM+ is not an optional feature under Windows (Global Architect, 2007).

Recently, there's been much said about Enterprise JavaBeans (EJB) and Microsoft's COM+ technologies. Some assert that EJB is new, and is therefore not ready for prime-time. Others question the historical scalability of Windows, and are uneasy about using Windows 2000 in their mission-critical deployments. However, those real, successful E-Commerce systems are being developed today's to both EJB and COM+. Despite the lack of support for certain features in each platform, today's development teams have learned to cope with some of the limitations of their chosen platform, such as lack of persistent components in COM+, or lack of queued components in EJB. It is very rare that an architectural decision will be made solely on the basis of features, as the two architectures are very, very similar. Rather, the overwhelming business forces at play are much greater factors.

The great feature of Microsoft technology is they always seem to undercut the competition when it comes to price. There is a remarkably low cost per transaction in Windows 2000, and this stems from the volume pricing Microsoft employs. Furthermore, the COM+ subsystem ships with Windows 2000, whereas EJB-based application servers are sold separately from the underlying platform. When you couple low-cost Intel hardware with a Microsoft-based middleware solution, the cost per transaction is remarkably low (Roman, 2007).

1.3 Aims and Objectives

The aim of this thesis is to optimize and analyse the performance of a COM+ based application, under the .NET framework environment. This is achieved using the following objectives:

- Improve database performance through enhanced COM+ techniques such as object pooling, and the ability to adjust the transactional isolation level for database operations.
 - Conduct a critical review of appropriate literature and benchmarking techniques performed on different data sources.
 - Conduct experiments and evaluation on COM+ settings and how this reflects in database and application performance.
 - Analyze and optimize Microsoft COM+ based application scalability and availability.
-

1.4 Thesis Structure

- Chapter 1 **Introduction.** This chapter outlines the background, scope and objective, along with the thesis structure of the work performed.
- Chapter 2 **Theory.** This chapter outlines the underlying theory of commonly used technology and terminology which are required to understand the context of the project.
- Chapter 3 **Literature Review.** This chapter outlines the literature review and research conducted in the areas of Microsoft COM+ and the performance of the database used with COM+ services.
- Chapter 4 **Design.** This chapter outlines the application information which includes the design, architecture, prototyping and experiments performed using the Microsoft COM+ core services.
- Chapter 5 **Implementation.** This chapter outlines the details behind the implementation of experiments conducted and explained in this chapter.
- Chapter 6 **Evaluation.** The results from the experiments obtained are analyzed and evaluated considering the parameters.
- Chapter 7 **Conclusion.** This chapter summarises the work performed for this project, presents the findings and suggest further work required in this field.
-

2 Theory

2.1 Introduction

This chapter outlines and describe the theory required to understand the project. This includes the technologies used in the presentation, business and data tier layer. The theory behind the COM, MTS and COM+ technologies will help in understanding the features exposed by these middleware technologies and also help in understanding the application language and their features in conjunction with the database used. We also looked into the detailed features provided by different Microsoft .NET Framework and the application compilation process cycle used for the language.

2.2 COM

COM (Component Object Model) is the Microsoft technology which enables software applications and components to communicate with each other. It is used to build re-usable software components, which can be linked together to build the applications, and take advantage of Microsoft Operating System services. COM was initially used in the Microsoft Office products and allows dynamic linking of the Microsoft Word documents to the Excel spreadsheets and allows users to build scripts using COM automation (Microsoft, COM: Component Object Model Technologies, 2007).

COM is designed primarily for Microsoft Visual Basic® and C++ developers. COM is a distributed, platform independent and object oriented system for creating binary software component that can interact with other components. COM runs on wide variety of operation systems and COM family includes technologies like COM+, Distributed COM (DCOM) and ActiveX® controls (MS, 2007).

2.3 Microsoft® Transaction Server (MTS)

MTS is a Microsoft component-based transaction processing system for building and deploying high performance, scalable and robust enterprise level, production quality database applications, which can be deployed and administered using the rich graphical tool. It is ideal for developing e-commerce and business intranet and internet application, and works with any application development tool capable of producing an ActiveX DLL. This includes application development tools like Microsoft Visual C++, Visual Basic and Visual J++ IDE. By providing a true component - oriented run time environment, MTS 1.0 changed the way developers built server centric applications and has eliminated the infrastructure code (Technet, 2007).

MTS version 2.0 is an important feature of the Microsoft Windows NT® operating system and simplifies the development and deployment of server - centric applications built using COM technologies. It also extends the environment by integrating the following technologies:

- Microsoft Internet Information Server 4.0 (IIS).
- Transactional connectivity to Oracle and DB2 databases.
- Integration with Microsoft Message Queue Server 1.0 (MSMQ).
- Connectivity through Microsoft SNA Server 4.0.
- COM Transaction Integrator (Corporation, 1998).

2.4 COM+

COM+ is the evolution of MTS and COM and COM+ is the name of the COM-based services and technologies first released in Windows 2000 (Microsoft, COM: Component Object Model Technologies, 2007). COM+ provides new features which extend applications written using MTS and COM - based technologies. Developers can now handle the management tasks which were difficult to program using COM, such as thread security and allocation (Bayer, 2001).

COM+ is designed primarily for Microsoft Visual Basic and Visual C++ developers. COM+ version 1.0 ships with Microsoft Windows 2000 and COM+ version 1.5 ships with Microsoft Windows 2003 operating system and Microsoft Windows XP (MS, 2007). It is being widely used to develop high - level mission critical, enterprise level distributed applications on the Microsoft operating systems (MS, 2007). COM+ 1.5 has functional features for distributed application which helps in increased performance and scalability (McKeown, 2003).

2.4.1 Object Pooling

Object pooling is an automatic service provided by COM that enables the developer to configure a component to have instances of it kept active in a pool, and it is available any client that request the component. Using the object pooling significant performance and scaling benefits can be achieved by reusing objects (MSDN, 2007). Bayer (2001) defines that:

Pooling is an object pool is a collection of pre-instantiated objects and use object pooling when your object needs to acquire expensive database resources such as database connections (Bayer, 2001).

2.4.2 Just In Time Compiler (JIT)

When the Microsoft .NET application code is compiled, the compiler generates code written in the Microsoft Intermediate Language (MSIL). The JIT compiler is responsible to convert the MSIL instructions into the native machine code that a CPU understands, and it also responsible for performing the verification process that the class loader performs. The concept that not all of an application's code is always executed by JIT, this improves the performance and the scalability of the .NET application (Bayer, 2001). Troelsen(2007) defines:

The entity that compiles CIL code into meaningful CPU instructions is termed a just-in-time (JIT) compiler, which sometimes goes by the friendly name of Jitter. The .NET runtime environment leverages a JIT compiler for

each CPU targeting the runtime, each optimized for the underlying platform (Troelsen, 2007).

2.5 .NET Framework

The Microsoft .NET framework provides many services that simplify application deployment and development. The Common language Runtime (CLR) is able to provide the services that all applications run on the top of the same execution engine Figure 2.1. It also consists of collection of Framework Class library (FCL). These libraries are used to create different types of applications like windows, web, mobile and distributed applications. (Microsoft, .NET Framework Conceptual Overview, 2007). Liberty & MacDonald (2006) defines that:

The .NET Framework sits on top of any flavour of the Windows operation system. The most important components of the Framework are the Common language Runtime (CLR) and the Framework Class Library (FCL), which provides an enormous number of predefined types or classes for developers to use in program (Liberty & MacDonald, 2006).

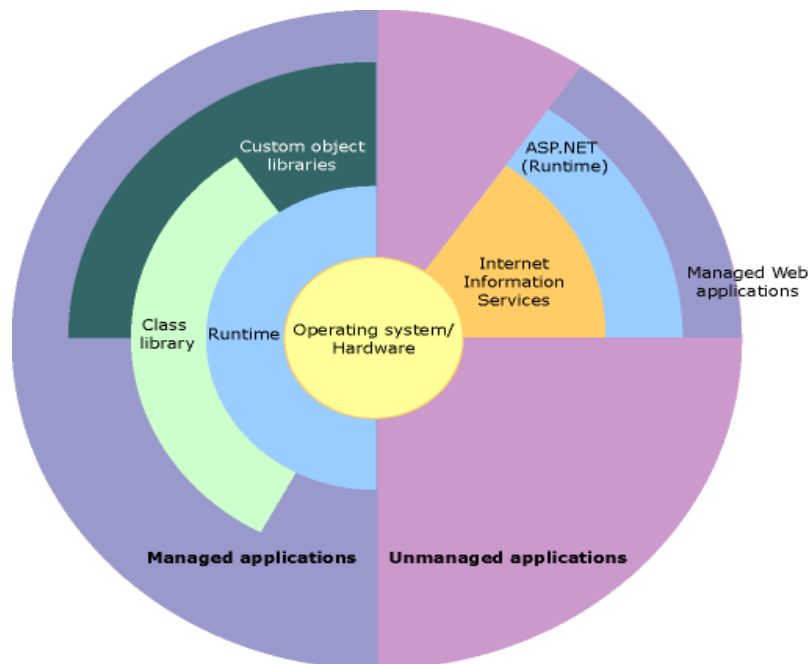


Figure 2.1 Microsoft .NET Framework (Microsoft, 2007).

There are several releases of the .NET Framework. These are outlined next:

2.5.1 .NET Framework 1.1

.NET Framework 1.1 is the first major Microsoft Framework upgrade and release. It includes the following main features:

- Built in ASP.NET controls for mobile application.
- Built in support for ODBC and oracle database.
- Integration of IPv6 protocol and numerous API changes.

2.5.2 .NET Framework 2.0

The .NET Framework 2.0 was released with the launch of Visual Studio .NET 2005. The following are the features included since 1.1 Framework.

- Providing full 64 - bit application support.
- ASP.NET personalization features.
- Release .NET Micro Framework (Compact Framework).

2.5.3 .NET Framework 3.0

.NET Framework 3.0 formerly named WinFX is the vital component of the Windows vista operating systems, such that:

With the release of the Vista operating system (OS), Microsoft officially shipped the third version of the .NET base class libraries. Within this release, developers are provided with several new technologies represented by a set of new .NET assemblies (Troelsen, 2007).

2.6 SQL Server 2005

Databases are the building blocks for the distributed and enterprise - level intranet and internet applications. Microsoft SQL Server is the Relational Database Management System (RDBMS) and analysis platform for large scale e-commerce, data warehousing and online transaction processing (OLTP) applications. The Database engine provides the controlled access and rapid transaction processing, and it is the core service for storing, processing, and securing data. Microsoft SQL Server 2005 is focused on making it easier to deploy, create and manager enterprise level database systems and applications, while increasing scalability, performance, reliability, security, availability and programmability (Whalen, Gracia, Patel, Misner, & Isakov, 2007), and it is highlighted by:

Some things are, however, worth waiting for, and SQL Server 2005 falls squarely in that camp. The number and importance of new or rewritten Features is almost staggering (Vieira, 2007).

Microsoft SQL Server 2005 includes key enhancements to manageability, availability, scalability and security to enterprise data management. New technologies have brought significant increase in developer productivity which includes new and expanded development tool which are integrated with the application framework, XML and Web services (Vieira, 2007).

2.7 C# .NET 2005

Figure 2.2, Microsoft C# is a type safe object - oriented language which enables developers to build a wide range of robust and secure applications run on the .NET framework. Developers can use C# for various kinds of applications such as client, Web and distributed applications (Troelsen, 2007).

Microsoft Visual C# 2005 provides advanced features, including:

- Advanced IDE for code development.
 - Convenient user interface for development.
-

- Integrated debugger and many more tools to provide rapid application development based on version 2.0 of the C# language.
- Provides features like re-factoring, debugging code, code snippets, database explorer and powerful navigation and searching.
- Support for all three coding models: inline, code - behind and mixed inline and code behind.
- Ability to import and export user preferences (Liberty & MacDonald, 2006).

C# language is highly expressive and easy to learn for the developers from Java, C++ background and designed to take advantage of the Common Language Runtime (CLR) that .NET program all rely upon (Kingsley & Kingsley-, 2007).

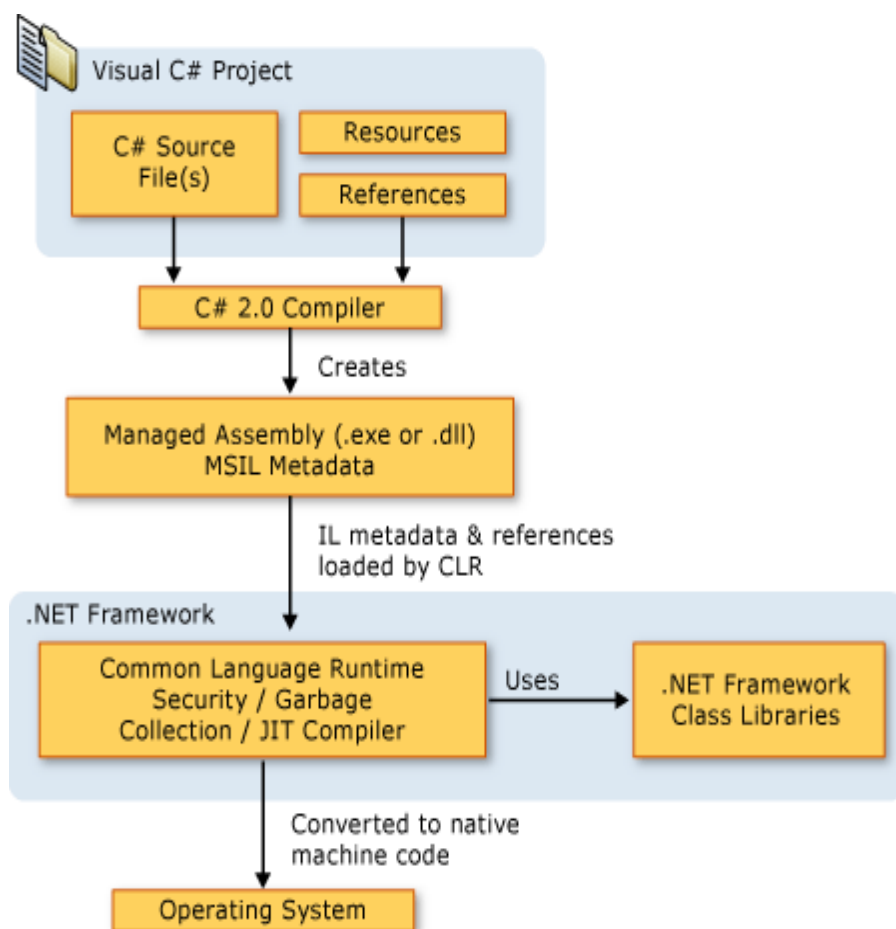


Figure 2.2 Microsoft C# Project life cycle (Visual C# Developer Center, 2007).

2.8 Conclusion

This chapter gives the outline and brief description about the relevant topic which are required to be understood for the thesis. This chapter outline how COM+ have emerged from COM and MTS and what are the new features are available in COM+ and what are the new features available in the database. There is also a brief introduction about the process compilation for C# .NET application. All of the mentioned Microsoft technologies COM, COM+, MTS, SQL Server and C#.NET contributing towards the achievement of the objectives. Different .NET Framework are backward compatible, but with new features must be assessed for usefulness.

3 Literature Review

3.1 Introduction

This section of the research paper emphasizes on the COM+ features and the research/test conducted using the COM+ settings on the data storage in the industry. Test result and data has been used and evaluated for the analysis.

The following section describes the importance of data storage and how the COM+ features like Object Pooling and Just in time compiler provides significant performance gain over the non pooled components.

3.2 Importance of Data Storage

The explosion of data on the web has taken new dimensions as data storage has been tremendously increased in the past 25 years from merely a Kilobyte (KB) data to gigantic Petabyte (PB) of data stored data centre servers. It has become a challenging task for Database researchers and developers who are building the interoperable business and enterprise components, heterogeneous query processor, clustered database and database extensions to provide consistent data access to diverse data sources. All this diverse information can be accessed using data access API's, abstraction and common data exchange formats (Deshpande & Blakeley, 2000).

Database management system are required to store, retrieve and manipulate large amounts of data in an efficient and reliable manner for the industry growing at the rate of 35% per year and generated the revenue of more than 7 billion in 1994. In order to efficiently manage the data storage it must have the specialized high level language to read data from the database, data structures to physically store the data and provide the reliability and integrity when database is accessed concurrently by many users (Yannakakis, 1995).

Microsoft data access API known as OLEDB builds on Microsoft's Component Object Model (COM) having the universal Access strategy which can access data to both database and non database irrespective of the location or format. However, most of the data of the mission critical application is stored in the multiple storage location for the purpose of performance and functionality purpose. Although most database companies follows Universal Storage strategy which provides user to store data of different type such as video, text, audio and pictures inside the database. By providing the integration of wide variety of data sources on a central location efficient and reliable applications can be developed (Blakeley, 1997).

3.3 Microsoft COM+

One component framework heavily used for developing component-based software systems is Microsoft's COM+ (Martin Pinzger, 2003). COM+ covers two main areas which are fundamental programming architecture for building software application components as defined in the COM specification and a group of component services using the COM+ runtime environment. Microsoft developed the Microsoft Transaction Server (MTS), the first Windows – based implementation of a runtime environment to provide component services (Eddon, 1999). and MTS is used to develop and deploy scalable, high performance and reliable distributed application. Which can be achieved by combining the technology of Component based environment (Limprecht, 1997). COM+ is a much more powerful runtime environment than anything else that is ever been deployed on a PC platform (Platt, 2000).

The Figure 3.1 below shows the performance result of the Microsoft COM+ Technology by the TPC. The TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry (TPC, 2007).


Rank	Company	System	tpmC	Price/tpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted	Cluster
1		HP ProLiant ML350G5	102,454	.73 US \$	12/31/07	Oracle Database 11g Standard Edition One	Microsoft Windows Standard x64 Edt. SP1 R2	Microsoft COM+	09/12/07	N
2		HP ProLiant ML350G5	100,926	.74 US \$	06/08/07	Oracle Database 10g Standard Edition One	Oracle Enterprise Linux	Microsoft COM+	06/08/07	N
3		PowerEdge 2900/1/2.33GHz/2x4M	69,564	.91 US \$	03/09/07	Microsoft SQL Server 2005 Standard Ed.	Microsoft Windows 2003 Server Std Edt SP1	Microsoft COM+	03/09/07	N
4		HP ProLiant ML350G5	82,774	.94 US \$	03/27/07	Microsoft SQL Server 2005 x64 Enterprise Edt. SP1	Microsoft Windows 2003 x64 Server Std. Ed.	Microsoft COM+	03/27/07	N
5		PowerEdge 2900/3.0GHz/4M	65,833	.98 US \$	06/26/06	Microsoft SQL Server 2005 Standard Ed.	Microsoft Windows 2003 Server Std Edt SP1	Microsoft COM+	06/30/06	N
6		PowerEdge 2800/1/2.8GHz/2+2M	38,622	.99 US \$	11/08/05	Microsoft SQL Server 2005 x64 Std. Ed.	Microsoft Windows 2003 x64 Server Std. Ed.	Microsoft COM+	09/26/05	N
7		PowerEdge 2800/1/3.6GHz/2M	28,244	1.29 US \$	02/09/06	Microsoft SQL Server 2005 Workgroup Ed.	Microsoft Windows Server 2003 Standard Edition	Microsoft COM+	02/09/06	N
8		PowerEdge 2900/1/2.66GHz/2x4M	126,371	1.33 US \$	06/08/07	Microsoft SQL Server 2005 x64 Enterprise Edt SP2	Microsoft Windows Server 2003 Enterprise Edition SP1	Microsoft COM+	06/08/07	N
9		PowerEdge 2800/1/3.4GHz/2M	28,122	1.40 US \$	04/30/05	Microsoft SQL Server 2000 Workgroup Ed.	Microsoft Windows Server 2003 Server	Microsoft COM+	02/24/05	N
10		PowerEdge 2850/1/3.4GHz/1M	26,410	1.53 US \$	12/10/04	Microsoft SQL Server 2000 Standard Ed.	Microsoft Windows Server 2003 Server	Microsoft COM+	12/10/04	N

Figure 3.1 Microsoft COM+ TPC Performance Result

3.4 COM+ Services

To measure the performance of Enterprise Services compared to COM+, components in the following languages:

- Visual C++ .NET and ATL COM+
- Visual Basic 6 COM+
- C# and .NET Framework 1.1 Enterprise Services
- Visual Basic .NET and .NET Framework 1.1 Enterprise Services

Each component contains two public methods:

- The trivial SUM () method adds two numbers together to simulate a lightweight operation that performs no disk or database access operations.
- The Trivial method Sale () typical method is transacted and calls the private method InsertSale () that inserts a record into a table and completes the transaction before returning. This method illustrates the performance characteristics of a "real-world" method doing typical business application work.

3.4.1 JITA

The Figure 3.2 below shows the number of calls per second achieved by repeatedly creating an object, calling its trivial method, and releasing it.

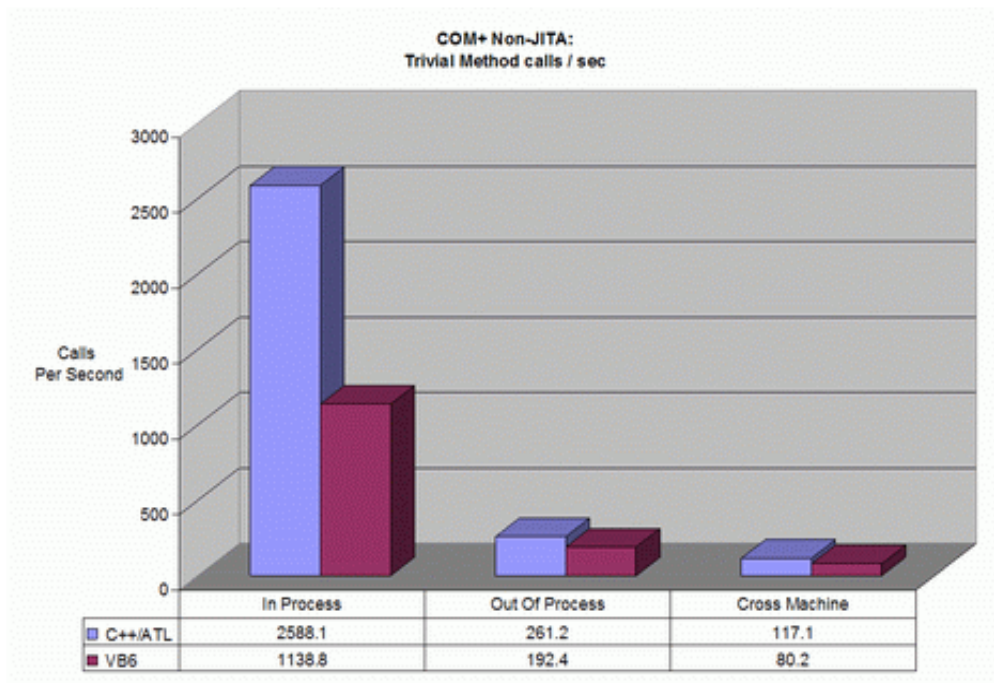


Figure 3.2 Microsoft COM+ Non JITA Performance Result

The Figure 3.3 illustrates what happens if we run a modified test that takes advantage of JIT-activation by creating a single object, and repeatedly calling a trivial method that adds two numbers together and then releases the object at the end of the test loop.

These results show a significant performance improvement in the number of calls per second when using JIT-activation. Using JIT-activation and Visual Basic 6 produces results that are almost 33 times faster than C++ without using JIT-activation (approximately 8600 Visual Basic 6 JIT-activated calls per second compared to approximately 261 Visual C++ non-JIT-activated calls per second).

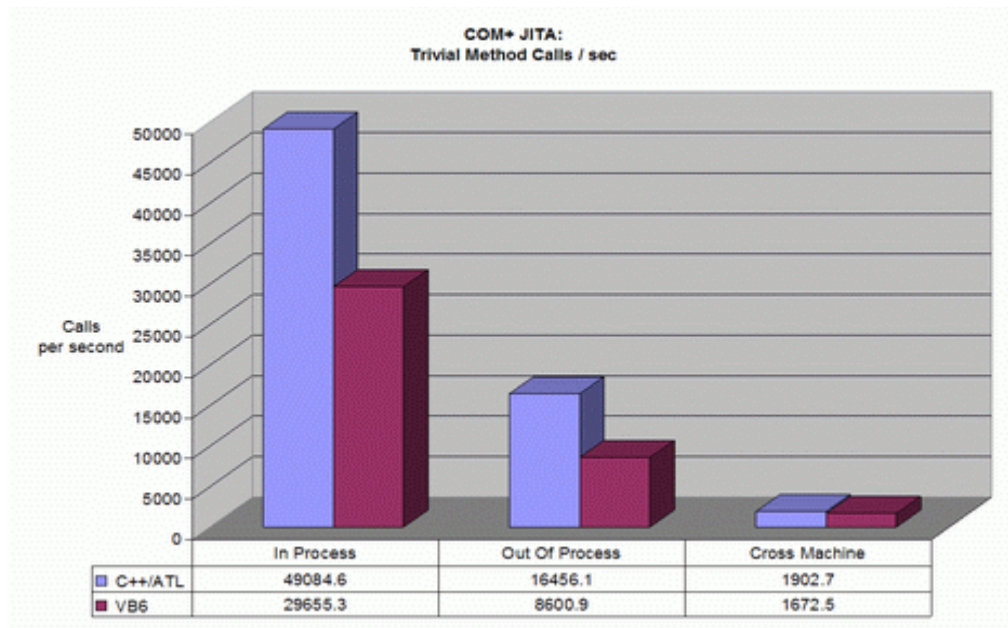


Figure 3.3 Microsoft COM+ JITA Performance Result

The Figure 3.4 below shows the performance of Enterprise Services using C# and Visual Basic .Net that calls the trivial method. Most of the cost of activating and releasing the object is gone, but the cost of delivering the call is still there, due to operations such as marshaling the buffers and converting to a call stack. Even with this very simple method, Enterprise Services is very close to the performance of Visual Basic 6 when going cross process. When calling across machines, all the languages perform very closely to each other.

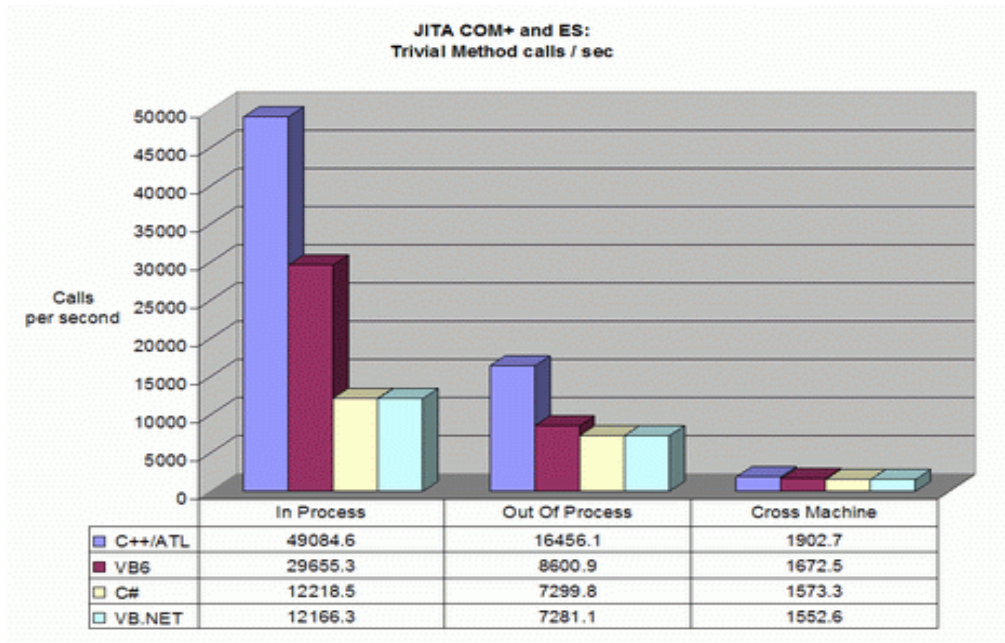


Figure 3.4 Microsoft COM+ and Enterprise Services Performance Result

The Figure 3.5 shows the relative performance of the same application written in four different languages that repeatedly calls a typical method to open a database connection and execute a simple SQL statement while inside a distributed transaction

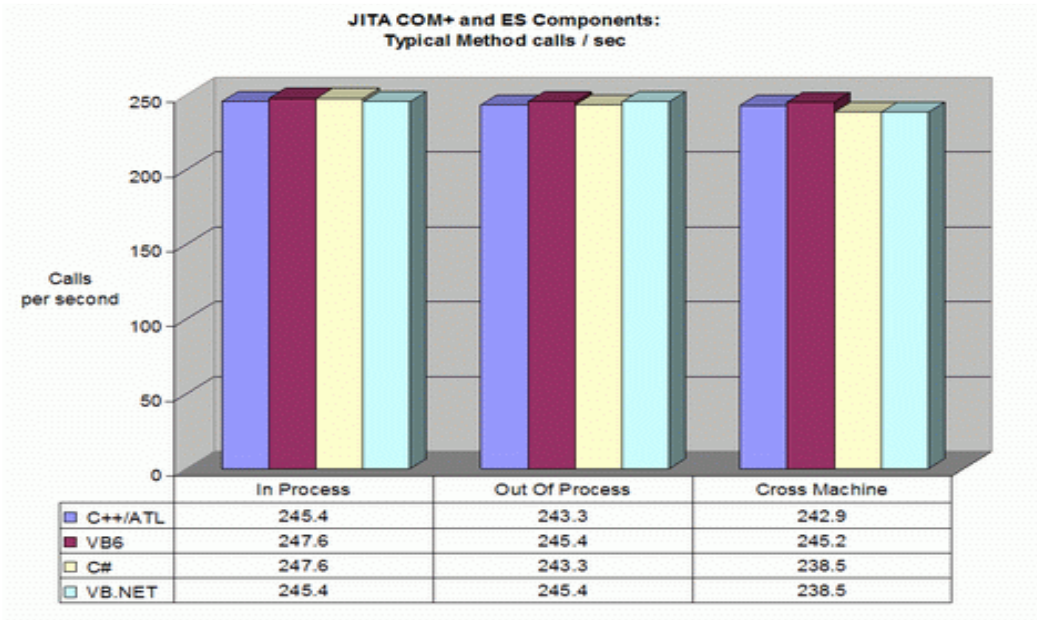


Figure 3.5 Microsoft COM+ and ES Typical Method Performance Result

The preceding results show that, within the experimental error, all languages give equivalent results when doing significant work inside the method. COM+ native applications written using C++ and Visual Basic 6 using ADO perform at the same speed as C# or Visual Basic .NET applications using Enterprise Services. Note that it matters very little from a performance perspective if you are running cross-process or cross-machine.

The object oriented language community has changed and there are now some good OO languages like C# .NET which is having excellent implementations and development environments (Gray, 2004).

3.4.2 Transactions

After turning off the “require transaction” setting in COM+ for each components gives the answer to "How much impact does COM+ distributed transactions have on the performance of these components?", The Figure 3.6 shows the results.

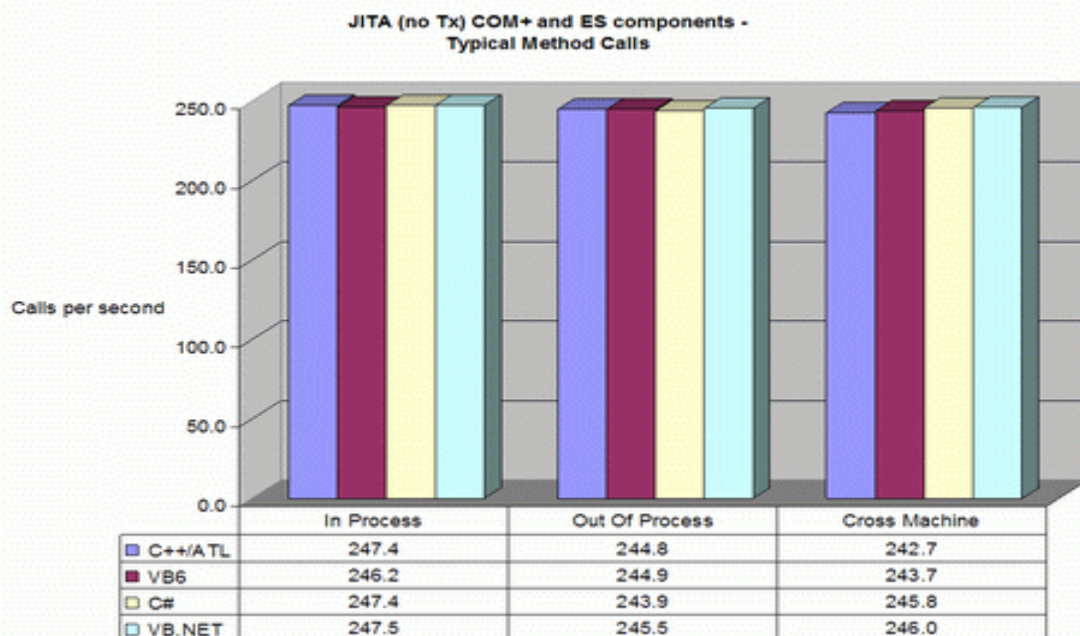


Figure 3.6 Microsoft COM+ and ES Typical Method [No Transaction Performance Result]

As can be seen from the chart above, the performance of the components without COM+ transaction support is practically identical to the performance of the components with transactions turned on. This clearly illustrates that the impact of COM+ transactions is negligible in these tests (Turne, Burek, & Driver, 2004).

3.4.3 Object Pooling

Figure 3.7, COM+ activates and deactivates objects to achieve efficient memory usage. In this discussion, COM+ pools components that use the thread-neutral apartment (TNA) model. Although Visual Basic does not currently support this model, pooling can allow COM+ to use memory more efficiently by avoiding the overhead of repetitive resource allocation. By avoiding the resource allocation is a

key concept for designing scalable systems and applies to all resources not just in the memory. The object pooling provides the significant performance improvement when objects are dealing with the database connection resources. Object pooling bypass the process of repeatedly connecting to a database.

COM+ and ADO 2.5/.NET classes can be used to efficiently manage the database connections through a process known as session pooling.

In Visual Studio 2005, Enterprise Services will be enhanced to eliminate one of the activation round trips, yielding a 20-30% improvement in performance (compared to the .NET Framework 1.1) when using the "activate/single call/release" pattern. However, user should avoid this pattern if at all possible (Turne, Burek, & Driver, 2004).

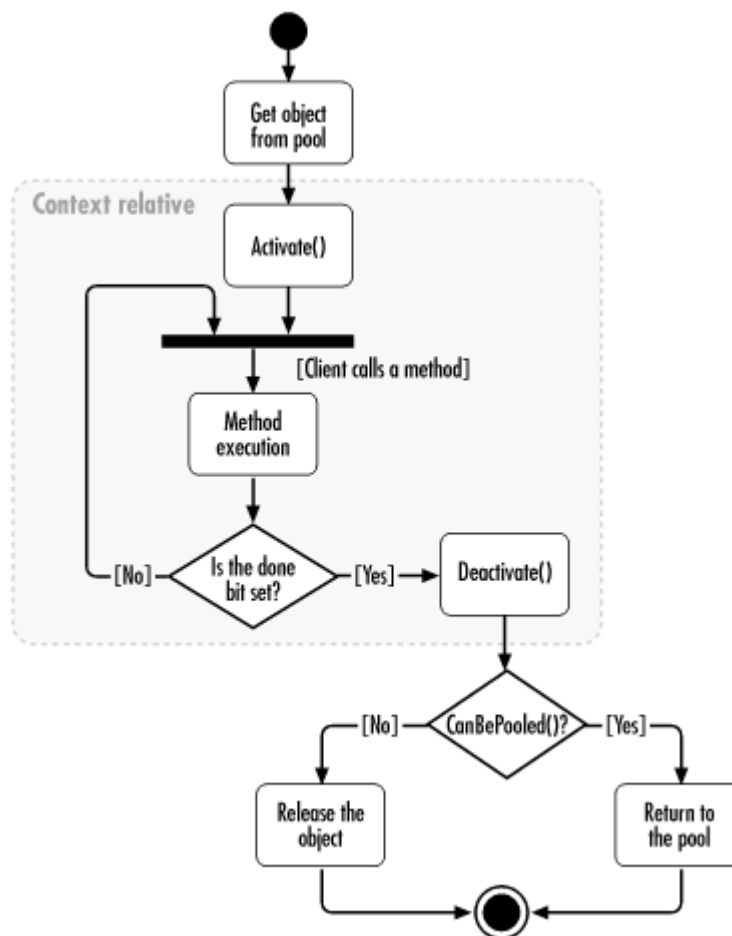


Figure 3.7 The life cycle of a component using JITA and object pooling (Löwy, 2001)

```
//Enabling COM+ Object Pooling Feature
[System.EnterpriseServices.ObjectPooling
(true, 10, 100, CreationTimeout = 5000)
]

//Enabling COM+ JIT Feature
[System.EnterpriseServices.JustInTimeActivation(true)]

//Enabling COM+ Transaction Option
[System.EnterpriseServices.Transaction
(TransactionOption.NotSupported)
]
```

The result in Table 1 shows Pooled Object results which were based on the COM+ with Object pooling and JIT. The Non Pooled Objects results were based on COM+ services without the Object pooling and JIT. When we run this on my machine, we got the following output:

Results	Ticks
Pooled Objects	404234
Non Pooled Objects	595959

Table 1 Pooled Object Performance

The results may differ somewhat depending on system configuration. The Figure 3.8 shows the performance gain for using the pooled component and COM+ object pooling can provide significant benefits (Bayer, 2001).

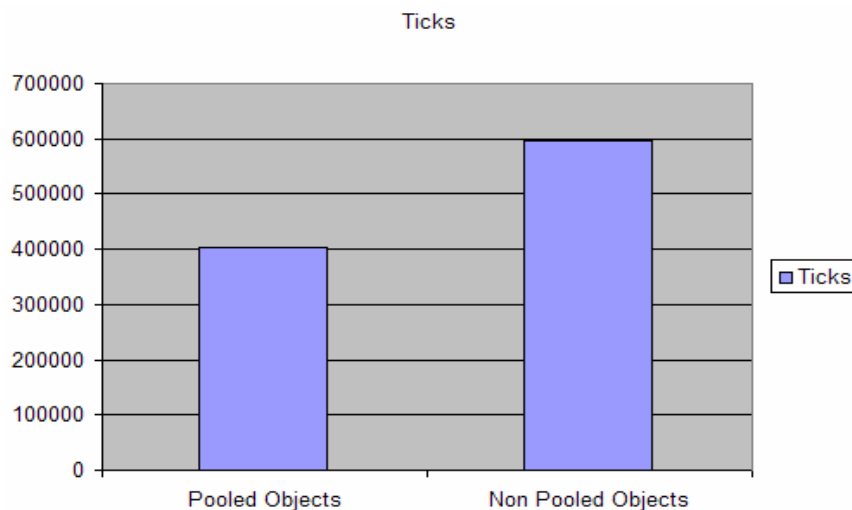


Figure 3.8 Pooled and Non Pooled Component Performance

3.4.4 Transaction Scenario

In a relational database system, all modifications occur as a result of an INSERT, DELETE, or UPDATE statement. The accounting transactions on a database have implemented using different database tables. The basic rule of accounting is that everything should balance; so for every debit, there must be a corresponding credit, and vice versa. For example, if user a want to pay a £50 bill to XYZ Corp, system debit Cash for 100 and credit Accounts Payable for 100. These two T ledgers are represented by database tables (Brill, 2000).

3.5 Conclusion

The results above illustrate how important JIT-activation and the "create/repeat call/release" calling pattern are as an aid to ensuring that your components perform as well as possible. By holding and reusing references to pooled and JIT-activated components, user can minimize component activations and disposals and achieve high levels of performance. In order to optimize the performance of COM+ components, it is important to minimize the number of cross-process or cross-machine calls made between caller and component. (Turne, Burek, & Driver, 2004). A good way to achieve this is to design COM+ components with methods that perform as much work as possible in a single call, even if that means designing components that deviate from architectural purity. It has been noticed from the research and the literature studied COM+ component which uses the Object Pooling and JITA features providing the significant performance gain over the non-COM+ based components.

4 Design

4.1 Introduction

This chapter defines how the prototype application was designed which are based on the earlier research and literature review. There are various test and experiments are mentioned in the literature review aided in designing the prototype application. The previous research on the COM+ Object Pooling and JITA also facilitates to plan the experiments on the new features of the technology.

The research also helped to design the interface which covers basic foundation for all the experiments based on data storage and enhance object model using the .NET framework. All the experiments are documented and used for the purpose of evaluating the application prototype.

This chapter outlines two main areas. The first consists of design, requirement and analysis of the underlying prototype application to conduct the experiments using the enhance object model. The second area covers a brief discussion about the evaluation design methodology which helps in identifying the information required for conducting the experiments at an early stage and also eases the evaluation process.

4.2 Requirement and Analysis

The requirement is the first phase for designing prototype application. The main objective of the prototype application is to benchmark different databases to analyse and optimize their performance using the enhance object model i.e. Microsoft COM+. The final prototype has to provide the answer to the following requirements:

- **Object Pooling:** The COM+ component must communicate with the database layer and provides the result of the query passed by the presentation layer. This can be evaluated by enabling the Object Pooling feature on / off. All these results must be documented, analyzed and evaluated against the time taken for the request.
 - **JITA:** The COM+ component must communicate with the database layer and provides the result of the query passed by the presentation layer. This can be evaluated by enabling the JIT compilation features on / off. All these results should be documented, analyzed and evaluated against the time taken for the request.
 - **Databases:** At the database layer, different databases must be used to distinguish the performance of COM+ component. This would provide the information on how COM+ component behave on the specific databases. Data storage performance can be benchmarked as per the numbers of user who are requesting the data. This will provide the information about the impact of users on the performance of databases in conjunction with the Microsoft COM+ settings. The database should have the different volumes of data. Analysis and evaluation can be performed on different data volume. This enables to analysis the COM+ component performance on the data.
-

- **Framework:** The experiments must be performed on the different Microsoft .NET Framework to analyse the performance of the COM+ components behaviour under the .NET environment.
- **User Preference:** The prototype presentation layer must give the following choices to the user for conducting the experiments:
 - User should have the option to choose the database for getting the results on the presentation layer.
 - User should have the flexibility to show data on the presentation screen.
 - User should have the option of COM+ functionality to choose from i.e. COM+ using object pooling and JITA and COM+ not using the Object pooling and JITA.
 - User should have the option of choosing a database of different volume sizes and should have the option to benchmark the selected database with different number of users load.

4.3 Interface design

In earlier research work and literature review there was an option to conduct the experiment using the specific features. The idea is to integrate most of the COM+ features and provide common interface to benchmark and analyse the database performance.

The interface Figure 4.1 used for the prototype application have all the options which are required to perform experiments on the databases using the COM+ settings under the controlled .NET environment. The prototype application is having a user friendly interface, which gives the ease of option to choose from database selection, technology and data volume selection in an easy manner.

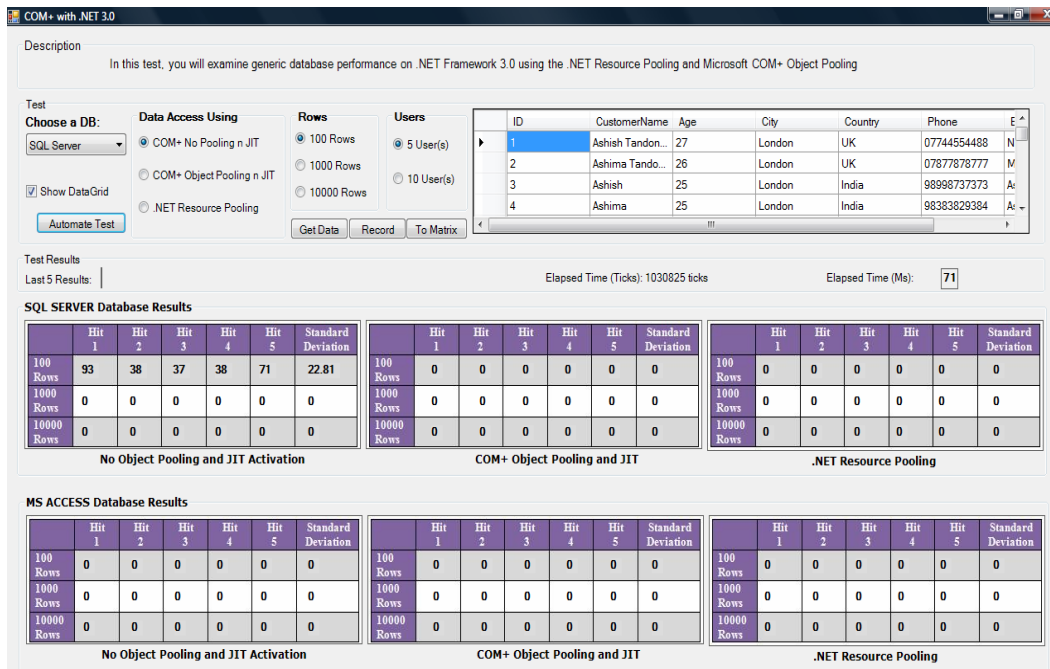


Figure 4.1 Prototype application

The information provided in the Figure4.2 allows the user to choose the database and the COM+ application type i.e. with or without object pooling and JITA option.

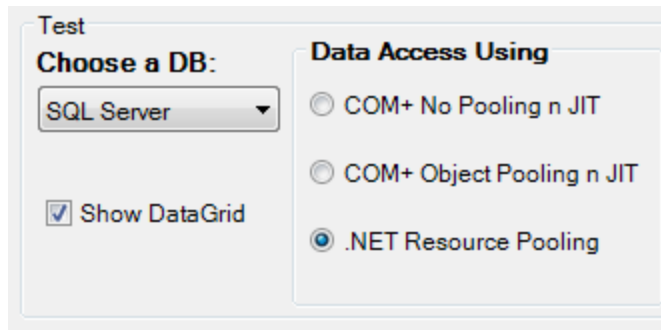


Figure 4.2 Prototype application Database and COM+ option

The information provided in the Figure 4.3 allows the user to choose the data which ranges from low to high volume and allows the user to choose the user load.

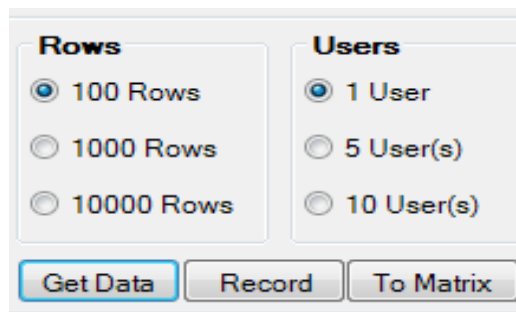


Figure 4.3 Prototype application Data volume and Users option

4.4 Analysis of Development environment

The main requirement is to benchmark the database performance using the Microsoft COM+ object model in the .NET Framework. It requires the language that gives the better Integrated Development Environment (IDE) and provides readily available components or framework classes that could be easily used to develop the prototype application. The integration between the business layer and the database layer should take less time and provides the flexibility to change the code easily.

Microsoft Visual Studio .NET provides the rich experience of GUI and a vast collection of the framework class library. It provides features of creating GUI with the relevant components, automated code snippets, better integration with the Microsoft database applications, less time spent on coding and deploying the components.

Microsoft Visual C# .NET was used in the prototype application. The other .NET languages run under the same .NET environment. There are few minor differences in functionality between the two languages as all .NET languages are interoperable. The decision is mostly driven by personal preference to write the code in C# as the syntax is more widely adopted.

4.5 Analysis of Database

The database and its integration with the language play an important role in achieving the overall performance. Microsoft SQL Server 2005 and Microsoft Access 2003 are used to conduct experiments. As SQL Server 2005 is the Relational Database Management System (RDBMS) which provide diverse features for transaction processing and provides the controlled access for storing and processing the data.

The integration of Microsoft Visual Studio and SQL Server and Microsoft Access provides the rich collection of framework class library which gives the power to create, manage and developed database applications in less time.

4.6 Project and Classes Implementation

There are three main projects used in the overall prototype application architecture that consist of four component application and one client application which is consuming these components to get the desired results. The following are the project categorization and the classes used in these projects.

4.6.1 Client Application

PrototypeApplication.cs: This class file is used for developing C#.NET based application who calls the other project for getting the results for the experiments conducted.

4.6.2 COM Access

COMAccess.cs: This class file is used for developing non-COM+ application with COM+ pooling and JIT for Access database.

4.6.3 COM SQL

COMSQL.cs: This class file is used for developing non-COM+ application with COM+ pooling and JIT for SQL database.

4.6.4 NoOMAccess

NoCOMAccess.cs: This class file is used for developing non-COM+ application without COM+ pooling and JIT for Access database.

4.6.5 NoCOM SQL

NoCOMSQL.cs: This class file is used for developing non-COM+ application without COM+ pooling and JIT for SQL database.

4.7 Evaluation design

Evaluation of the prototype application is the primary reason for analysing the COM+ behaviour under the .NET Framework. Therefore it is really important to judge the performance result of the COM+ components on different databases. The following experiments are used to evaluate the performance of COM+ component on different databases on the different COM+ parameters i.e. object pooling, JITA, transaction support, constructor used.

Experiment 1: COM+ Application Performance with No Object Pooling and JITA: The fundamental idea behind this experiment was to analyze the performance of COM+ Server and COM+ Library applications without using the COM+ 1.5 Object Pooling and JITA activation features running under the .NET Framework. This experiment was aimed to achieve the objective which identify that which COM+ application provides better performance and results against the database retrieval query.

Experiment 2: COM+ Application Performance with Object Pooling and JITA: The fundamental idea behind this experiment was to analyze the performance of COM+ Server and COM+ Library applications using the COM+ 1.5 Object Pooling and JITA activation features running under the .NET Framework. This experiment was aimed to achieve the objective which identify that which COM+ application provides better performance and results against the database retrieval query.

Experiment 3: The Comparative Performance of COM+ application: The main idea for this experiment was to compare the performance difference between the COM+ Server application and COM+ Library application. This experiment was aimed to achieve the objective to comparatively show the difference between the components using the COM+ features and COM+ component not using the COM+ features.

Experiment 4: COM+ and Non COM+ Component Performance: This experiment was performed to test the performance of COM+ component developed on .NET Framework using the .NET System.EnterpriseServices and the performance of the .NET class library component. The aim of this experiment was to identify the performance gap between the COM+ and Non COM+ components under the .NET Framework.

Experiment 5: COM+ Role Based Component Performance: This experiment was the step forward towards the COM+ features implementation and experimentation. This experiment performed to test COM+ role base component and was aimed to achieve the objective that does COM+ role based security features having a performance cost or not.

Experiment 6: COM+ Transaction Based Component Performance: This experiment was another implementation of COM+ features implementation and experimentation. This experiment performed on the COM+ component with the transaction required new mode. It was aimed to achieve the objective that does COM+ transaction required mode was having a performance cost or not.

Experiment 7: COM+ component features and their performance: This experiment was aimed to compare the performance of the COM+ component with JIT, component with role based and transaction required mode.

Experiment 8: Non-COM+ component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 9: Non-COM+ component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 10: COM+ component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 11: COM+ component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 12: .NET based application component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 13: .NET based application component performance on SQL Server: This experiment was aimed to compare the performance of Non-COM+ component using the SQL Server 2005 data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 14: Application performance on SQL Server (5 Users + 100 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the low volume (100 Rows) of SQL Server data and 5 users.

Experiment 15: Application performance on SQL Server (10 Users + 100 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the low volume (100 Rows) of SQL Server data and 10 users.

Experiment 16: Application performance on SQL Server (5 Users + 1000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the medium volume (1000 Rows) of SQL Server data and 5 users.

Experiment 17: Application performance on SQL Server (10 Users + 1000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the medium volume (1000 Rows) of SQL Server data and 10 users.

Experiment 18: Application performance on SQL Server (5 Users + 10000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the high volume (10000 Rows) of SQL Server data and 5 users.

Experiment 19: Application performance on SQL Server (10 Users + 10000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the high volume (10000 Rows) of SQL Server data and 10 users.

Experiment 20: Non-COM+ component performance on MS Access: This experiment was aimed to compare the performance of Non-COM+ component using the MS Access data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 21: Non-COM+ component performance on MS Access: This experiment was aimed to compare the performance of Non-COM+ component using the MS Access data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 22: COM+ component performance on MS Access: This experiment was aimed to compare the performance of COM+ component using the MS Access data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 23: COM+ component performance on MS Access: This experiment was aimed to compare the performance of COM+ component using the MS Access data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 24: .NET based application component performance on MS Access: This experiment was aimed to compare the performance of .NET based component using the MS Access data of different volume with 5 users on Microsoft Framework 2.0 and 3.0.

Experiment 25: .NET based application component performance on MS Access: This experiment was aimed to compare the performance of .NET based component using

the MS Access data of different volume with 10 users on Microsoft Framework 2.0 and 3.0.

Experiment 26: Application performance on MS Access (5 Users + 100 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the low volume (100 Rows) of MS Access data and 5 users.

Experiment 27: Application performance on MS Access (10 Users + 100 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the low volume (100 Rows) of MS Access data and 10 users.

Experiment 28: Application performance on MS Access (5 Users + 1000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the medium volume (1000 Rows) of MS Access data and 5 users.

Experiment 29: Application performance on MS Access (10 Users + 1000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the medium volume (1000 Rows) of MS Access data and 5 users.

Experiment 30: Application performance on MS Access (10 Users + 10000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the high volume (10000 Rows) of MS Access data and 5 users.

Experiment 31: Application performance on MS Access (10 Users + 10000 Rows): This experiment was aimed to compare the application type performance on Microsoft .NET Framework 2.0 and 3.0 using the high volume (10000 Rows) of MS Access data and 5 users.

4.8 Conclusion

This chapter outlines the design framework used by the prototype application and how the COM+ components are designed and developed, and how COM+ features has been effectively used to benchmark the database. The experiment structure mentioned the brief details about the nature of experiment which uses the two major databases, SQL Server and MS Access. COM+ features like pooling, JIT, role base security and constructor object have been designed and will be tested on the different .NET Framework environment.

The experiments designed for evaluating the prototype have been developed to test on the different .NET Framework, database and user load. They will test the performance of COM+, Non-COM+ and .NET based application on the different volume of data ranges from low (100 Rows) to high volume (10000 Rows) and also with the user connection under the Microsoft .NET Framework 2.0 and 3.0 environment.

5 Implementation

5.1 Introduction

The focal point of this chapter will be on the implementation of the prototype application. Each section defines and explains how it was programmed. The prototype application has been developed and implemented using the Microsoft C#.NET. Moreover we will explain about the following COM+ features implementation in this chapter.

- Object pooling
- JITA
- Role Based Security
- Transaction isolation level

We will also explain about the how configuration information is handled, dynamic attributes for pooled and non pooled components, dynamic SQL query, mathematic function used to calculate median and standard deviation, activity matrix implementation and explain about the testing methodology used for calculation the time between the call request and reply received.

5.2 Configuration Information

The following code explains how the prototype application uses the two database connection information. We can customize how the common language runtime locates and loads assembly files by adding application configuration files (app.config files) to Visual C# .NET project and we have stored the database connection information in the configuration file.

The information stored in the app.config is a XML file and we have added one node named as 'add' for each database and provides the information in their attributes which includes name, provider Name and connection String. This information is required for the COM+ to establish and perform the relevant database operation on the database.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="SQL Server"
      providerName="System.Data.SqlClient"
      connectionString="Data Source=.\SQLEXPRESS;Initial
      Catalog=OfficeMart;Integrated Security=True;"/>
    <add name="MS Access" providerName="System.Data.OleDb"
      connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
      Source=|DataDirectory|\Databases\MyData.mdb;Persist
      Security Info=True" />
  </connectionStrings>
</configuration>
```

When we build the project, the development environment automatically creates a copy of app.config file, changes its file name so that it has the same file name as your executable (MSDN2, 2007).

5.3 Pooled Component Implementation

The following code explains how we can make a pooled COM+ component. The class uses the System.EnterpriseServices namespace which is used to derive an object from a service component and then give that object a transaction attribute that would specify how it uses transactions (TV, 2007). Using the EnterpriseServices.ObjectPooling we can make the component pooled by providing the parameters like Min Pool, Max Pool and the timeout value of a pooled component. We can also turn on or off the JITA activation and provide the transaction option to supported or not supported.

```
namespace ObjectPoolServer
{
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.SqlClient;

    [System.EnterpriseServices.ObjectPooling
     (true, 10, 100, CreationTimeout = 5000)
    ]
    [System.EnterpriseServices.JustInTimeActivation(true)]
    [System.EnterpriseServices.Transaction
     (TransactionOption.NotSupported)
    ]

    public class PooledObject : ServicedComponent
    {
        private System.Data.SqlClient.SqlConnection _cnn;
        private System.Data.SqlClient.SqlCommand _cmd;

        public PooledObject()
    }
}
```

5.4 Non Pooled Component Implementation

The following code explains how to turn off the COM+ Object Pooling feature to false which builds the component without the object pooling feature. We developed the non - pooled and non JITA enabled COM+ component by setting their parameter value to false. This component will not use the COM+ object pooling and JITA features and will be treated as the dynamic link library.

```
namespace ObjectPoolLibrary
{
    //Pooling without object pooling and JIT
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.SqlClient;
    using System.Reflection;

    [System.EnterpriseServices.ObjectPooling(false)]
    [System.EnterpriseServices.JustInTimeActivation(false)]
    [System.EnterpriseServices.Transaction
    (TransactionOption.NotSupported)]
    ]

    public class PooledObject : ServicedComponent
    {
        private System.Data.SqlClient.SqlConnection _cnn;
        private System.Data.SqlClient.SqlCommand _cmd;

        public PooledObject()
    }
}
```

5.5 JITA

Microsoft COM+ Component can be build with JITA enabled or disabled. The following code explains how to turn the COM+ JITA feature on or off by setting the JustInTimeActivation attribute to true or false.

```
[System.EnterpriseServices.JustInTimeActivation(false)]
```

5.6 Matrix Implementation

The Matrix implementation has been developed by placing the various label objects on the interface and interface has the functionality to record the last 5 performance result. Once the performance is recorded user has to click on the Matrix button to transfer the result in their respective tables and their respective cell values.

```
act1.Text = sArr[0]; act2.Text = sArr[1]; act3.Text = sArr[2];
act4.Text = sArr[3]; act5.Text = sArr[4];sd5r2.Text =
SDInitiate().ToString();
```

5.7 Calculation of Median and Standard Deviation

This is been the good idea to calculate the median and the standard deviation of the results collected using the different experiment (MSDN2, 2007). The following code is used to calculate the variance which accepts the last 5 recorded performance result and firstly it calculates the average and uses the .NET framework class library method which returns a specified number raise to the specified power (Easy Calculation, 2007).

```
public static double GetVariance(double[] data)
{
    int len = data.Length;
    // Get average
    double avg = Average(data);

    double sum = 0;
    for (int i = 0; i < data.Length; i++)
        sum += Math.Pow((data[i] - avg), 2);
    return sum / len;
}

public static double GetStdev(double[] data)
{
    return Math.Sqrt(GetVariance(data));
}

private static double Average(double[] data)
{
    double DataTotal = 0;
    try
    {
        for (int i = 0; i < data.Length; i++)
        {
            DataTotal += data[i];
        }
        //return SafeDivide(DataTotal, data.Length);
    }
    catch (Exception e)
    {
        MessageBox.Show("Error 111: There was an error
in processing the request" + e.Message, "Error
Calculating Average", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    return SafeDivide(DataTotal, data.Length);
}
```

5.8 Data Grid Implementation

The Data Grid implementation gives the flexibility to the user to view the data returned from the backend databases. User can check and uncheck this option by checking the checkbox 'Show DataGrid' on the interface as mentioned in the Figure 5.1. Also the following code binds the data returned from the database with the Data Grid control lying on the application interface only if the check box control value is true.

```
if (cbShowDBGrid.Checked==true)
    displayDataGridView.DataSource = myDataSet.Tables[0];
```

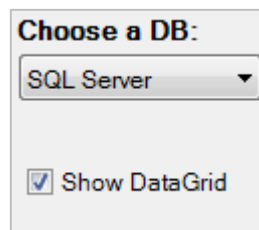


Figure 5.1 Show Data in the DataGrid option

5.9 Dynamic Query

The Dynamic Query enables the user to experiment and benchmark the database on different volume of data. The following code explains how the low, medium and high volume of data is retrieved from the backend databases. In the database there is only one table which consist of more than 10000 records and a few cities are having are having the least and maximum records.

The query is dynamically created on the basis of the user's input and the same application logic is user for the user load. See Figure 5.2 which allows the user to choose the volume of data range from 100 Rows to 10000 Rows which is termed as low volume to high volume of data. User can also choose the number of user connection to be used while performing the database operation.

```
//Set the value and query for Rows selected
if (rbLowVolumeData.Checked == true)
{
    //The following query returns 100 Row(s)
    myQuery = "SELECT * FROM Customers where
              city='London'";

    //The query considered under the low volume data
    sVolume = "low";
}

else if (rbAverageVolumeData.Checked == true)
{
    //The following query returns 1000 Row(s)
    myQuery = "SELECT * FROM Customers where
              city='Livingston'";
```

```

//The query considered under the medium volume data
sVolume = "medium";
}

else if (rbHighVolumeData.Checked == true)
{
//The following query returns 10000 Row(s)
myQuery = "SELECT * FROM Customers where
          city='New Delhi'";

//The query considered under the high volume data
sVolume = "high";
}

```

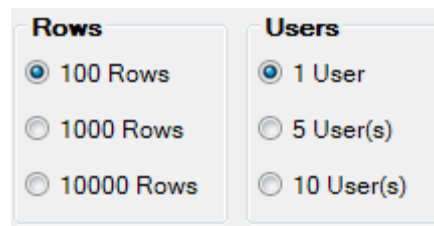


Figure 5.2 Show Data Volume and User option

5.10 Testing Implementation

The following code explains how the time taken between the request made and received is calculated. The .NET function “DateTime.Ticks” have been used to implement this functionality. .NET DateTime.Ticks “The value of this property represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight, January 1, 0001”. As measurements are purely comparative, (MSDN, .NET Framework Developer Center, 2007). no need to have an accurate timing functions, as .NET Data Time functions have the sufficient and suitable grain.

```

Stopwatch myWatch = new Stopwatch();
myWatch.Start();
myWatch.Stop();

elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): " +
myWatch.ElapsedTicks.ToString() + " ticks";

millisecondsTextLabel.Text = "Elapsed Time (Ms):";
lblms.Text = myWatch.ElapsedMilliseconds.ToString();

```

5.11 Conclusion

This chapter of the thesis include all the features that were discussed during the prototype design. The decision to choose the Microsoft Visual Studio .NET environment and C# .NET as language to implement the prototype features, due to the nature of implementation required and the rapid application tool to develop the application of this type. The programming required for this prototype application was not difficult in comparison to enterprise level application. All the application method have the structured exception handling to catch any exception comes during the running instance.

The ability to generate forms quickly, performing mathematical calculation, displaying the data in a right and a proper manner for the ease to understand and analyze and also the integration with other type of application type projects makes the right choice of choosing the environment and language. This saves lot of time which can be used on the testing and integration of the system. All the unit testing and integration with other project has been performed to make sure that system should provide the consistent and reliable data information of the experiments conducted. The simplicity of the system makes gathering the results of experiments from different sources easier. The final prototype possesses all of the requirements and features needed to facilitate a successful evaluation of the COM+ services on the data storage.

6 Evaluation

6.1 Introduction

In this chapter of the thesis, we will attempt to evaluate the Prototype Application used to analyse and optimize the database performance using the enhance object model in .NET Framework and also attempt to determine whether it fulfils the requirement of this project as mentioned at the beginning of this project. To evaluate the success of this prototype application a series of experiments were designed to test each experiment with the relevant set of parameters. These experiments tested the COM+ server and library application performance and how database performance was reflected using the COM+ features on the .NET Framework environment 2.0 and 3.0. Other experiments conducted on the application type like; COM+ component, non-COM+ component and .NET based application performance on the .NET Framework 2.0 and 3.0.

Different types of experiments were chosen to contribute in the overall evaluation process. They represented different application performance, different databases used, range of data volume and range of user load.

6.2 Methodology

The experiments were carried out using the following matrix. The Row value contains the value in average milliseconds for the last 5 database performance results. The column value having the details of the number of hit ranges from 1 to 5. Every hit time was placed on the mentioned hit column against their respective row value. The value of the median calculated from the last 5 response time and the standard deviation value is calculated using the median.

The same matrix was used to calculate the median and standard deviation values for the pooled, non pooled component and .NET connection pooling for both the SQL Server and MS Access databases.

	Hit 1	Hit 2	Hit 3	Hit 4	Hit 5	Median	SD
Rows 100							
Rows 1000							
Rows 10000							

Table 6.2.1 Experiment Matrix

6.3 SQL Server Experiments Results

Following are the series of experiments conducted using the developed prototype application.

The chart 6.3.1 shows that COM+ library application has taken more time for the request as compared to the server application using the COM+ object model under the .NET environment. As per the table 6.3.1, it has been notice that on the concurrent connections of ten users, server application results are twice faster than the library application and there is an in the increase up to 210 % in time for the library application. The results are based on the trivial method performing read operation on the database. The chart represents that COM+ Server application gives significant performance gain over the COM+ library application with no pooling and JIT features.

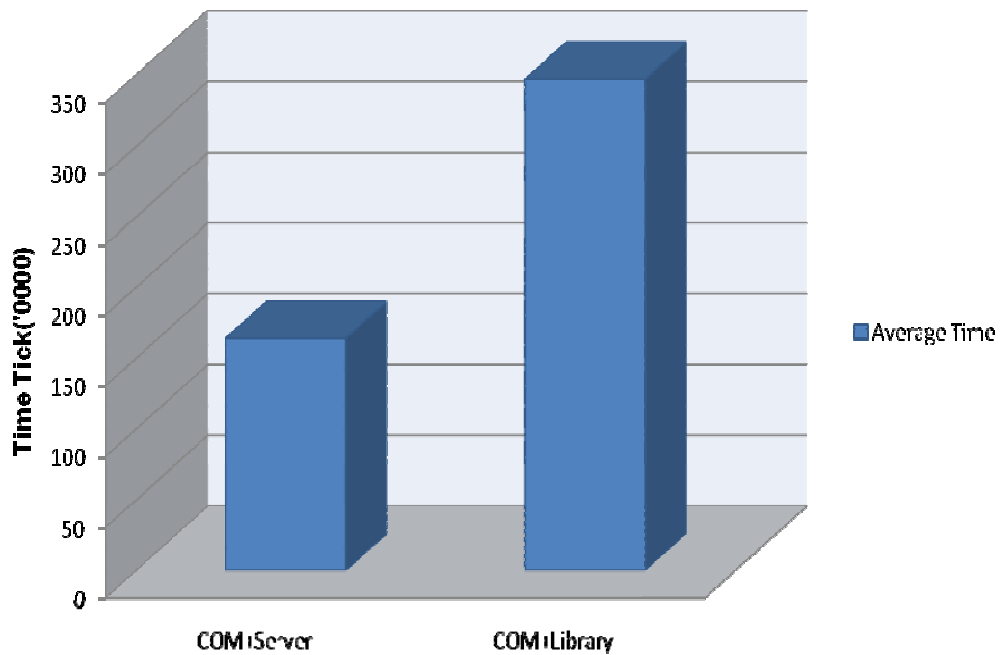


Chart 6.3-1 COM+ Application Performance [No Object Pooling and JIT]

	COM+ Server	COM+ Library	No of Users
Average	164	34	10
% Up	100	211	10

Table 6.3.1COM+ Application Performance Data [No Pooling and JIT]

The chart 6.3.2 shows pooling and JIT enabled results of server and library applications using ten concurrent users. The results represents that there is slight performance gain using the server application over library application. As per the table 6.3.2, it has been notice that on the concurrent user connections, server application results are 10 % faster than the library application based on the trivial method performing read operation on database. Thus COM+ server application provides performance gain over the library application even when the object pooling and JIT is enabled.

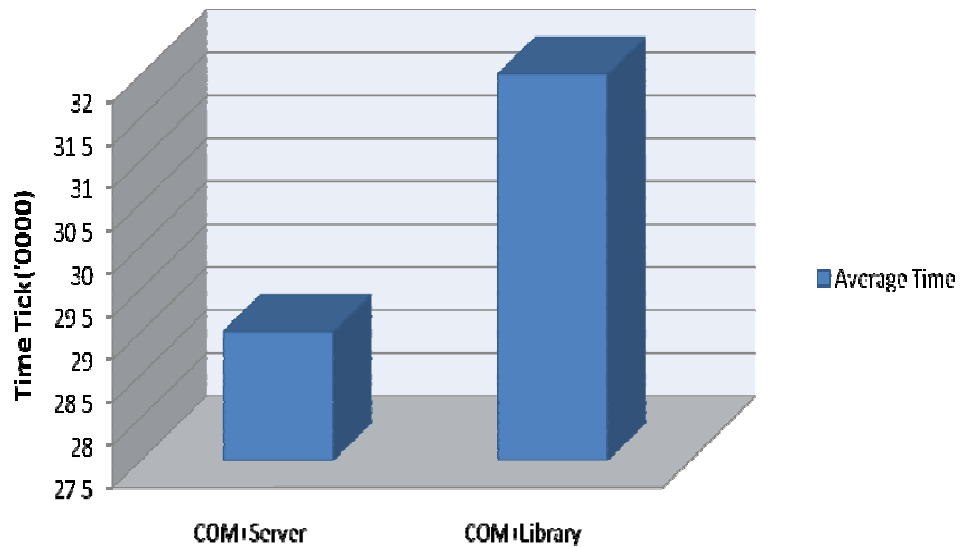


Chart 6.3-2 COM+ Application Performance [Object Pooling and JIT]

	COM+ Server	COM+ Library	No of Users
Average	29	32	10
% Up	100	110	10

Table 6.3.2 COM+ Application Performance Data [Pooling and JIT]

The chart 6.3.3 shows the comparative summary of the COM+ server and library application type results which uses the trivial method which established the connection with the database to perform the read operation. The results are the Avg. time taken by the trivial method using the ten user connections. The results represents that there is a significant performance gain with server application having no pooling and JIT and slightly better performance gain even with pooling and JIT. In both the cases out - process application has better performance results over the in - process application.

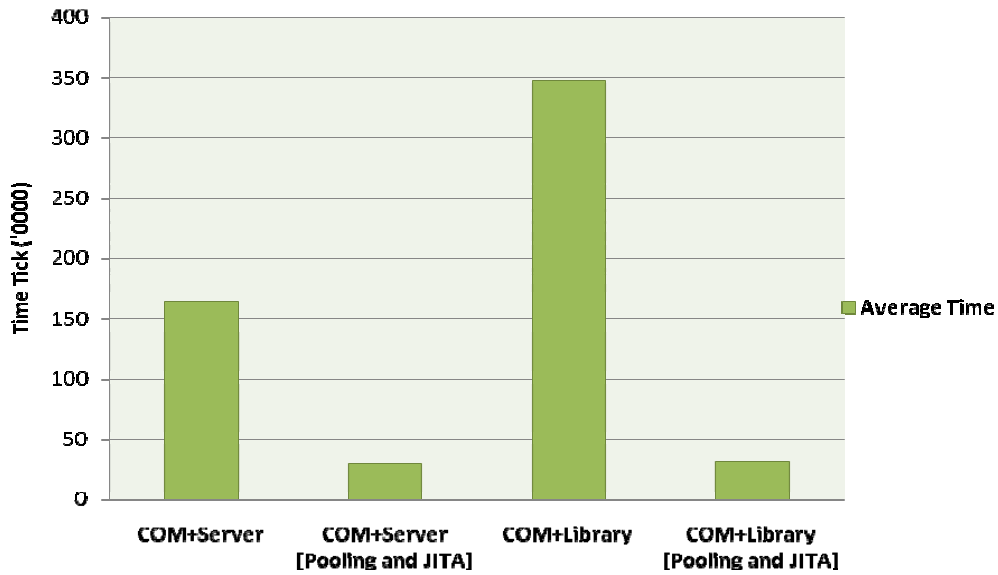


Chart 6.3-3 COM+ Application Performance

	COM+Server	COM+Server [Pooling and JITA]	COM+Library	COM+Library [Pooling and JITA]
Average Time	164	30	348	32

Table 6.3.3 COM+ v/s Non COM+ Component Data Performance

The chart 6.3.4 shows the performance results of COM+ and non COM+ component application. The experiment is performed using the pooling and JIT features of COM+ and trivial method performing read operation on the database. The results represents that COM+ component provides better performance results over non COM+ component and they are twice faster than the other. As per the table 6.3.4, it has been notice that COM component provides better result over non COM component, it can be seen that there is more than 225% increase in time taken for non COM component. Thus creating a COM component using the pooling and JI features for performing the database operation is a better choice over a non COM component.

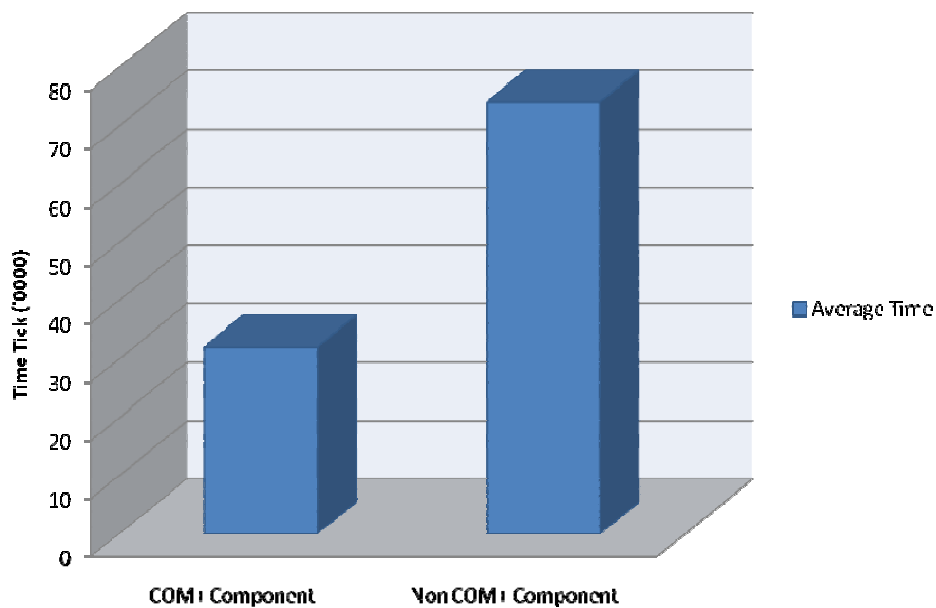


Chart 6.3-4 COM+ v/s Non COM+ Component Performance

	COM+ Component	Non COM+ Component
Average Time	32	74
% Up	100	232

Table 6.3.4 COM+ v/s Non COM+ Component Data Performance

The chart 6.3.5 shows the results of component with pooling, JIT and component with pooling, JIT and role based security. The results represents that role based security features comes at the performance cost, which shows that non role based component provides better results over the role based component with JIT and pooling. As per the table 6.3.5, it has been notice that on the concurrent user connections, component with role based security features was taking 20% more time than other component with pooling and JIT enabled.

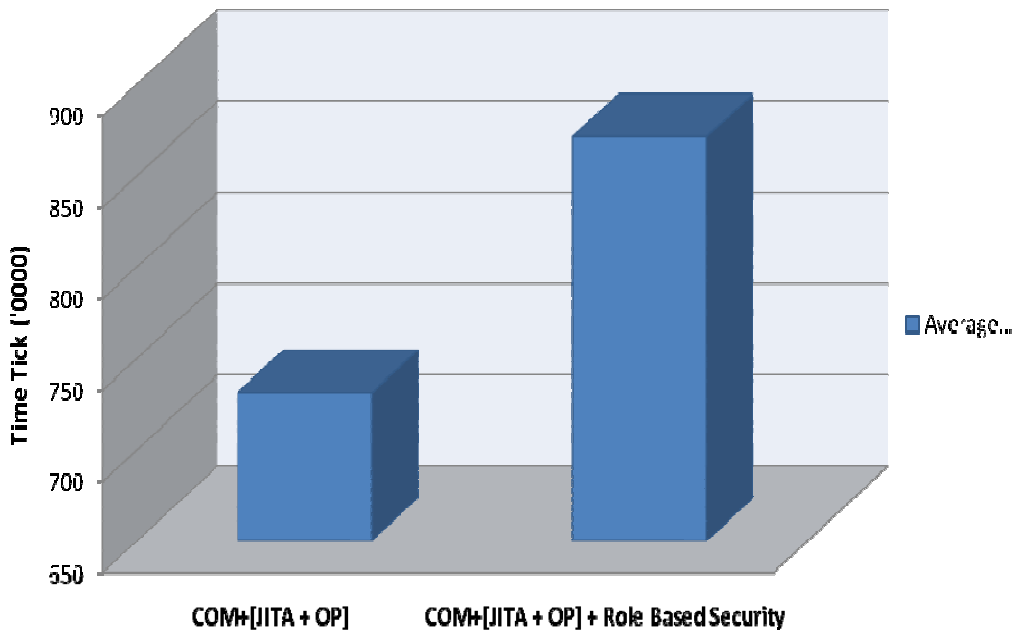


Chart 6.3-5 COM+ Role Based Security Component Performance

	COM+[JITA + OP]	COM+[JITA + OP] + Role Based Security
Average Time	631	761
% Up	100	121

Table 6.3.5 COM+ Role Based Component Performance Data

The chart 6.3.6 shows the results of component with pooling, JIT and other component with pooling, JIT and transaction required new property. The results represents that component with transaction required new property performs slightly slower than the component with the role based property. As per the table 6.3.6, it has been notice that on the concurrent user connections, component with transaction required new property was taking around 35% more time over the component with pooling and JIT enabled.

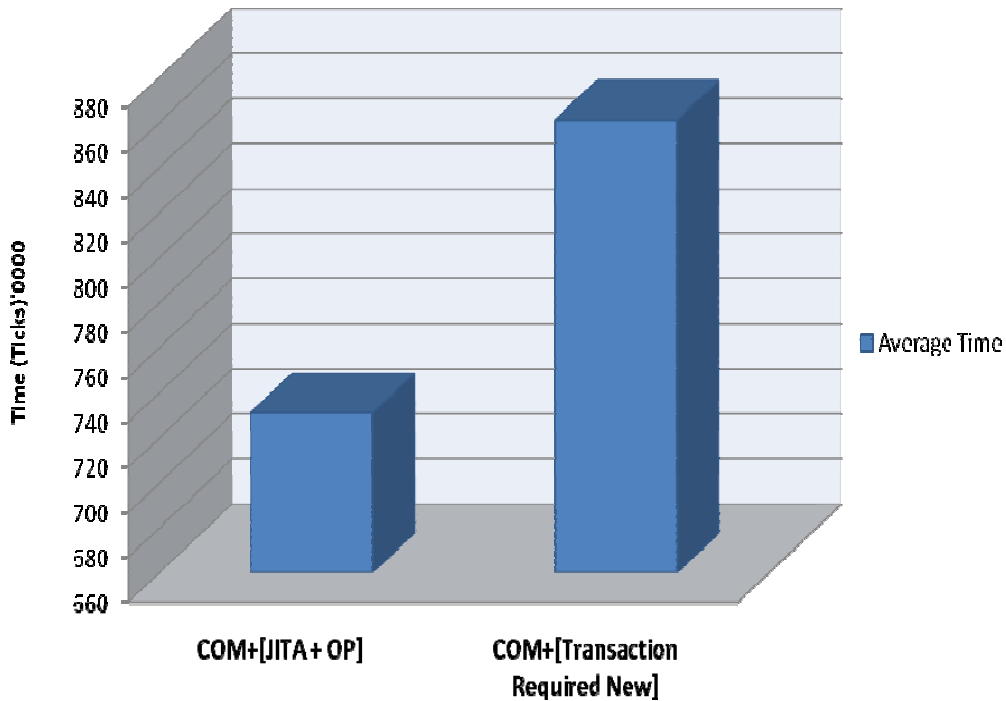


Chart 6.3-6 COM+ Transaction Component Performance

	COM+[JITA + OP]	COM+[Transaction Required New]
Average Time	631	861
% Up	100	137

Table 6.3.6 COM+ Transaction Component Performance Data

The chart 6.3.7 shows the comparative results of the COM+ property component which are based on the trivial read method on the database. As per the table 6.3.7, it has been notice that by making use of the COM+ features into a component slighter performance cost comes into play. The difference between the COM+ server application with pooling, JIT and COM+ transaction property component is around 30%. This makes the component capable of providing COM+ features

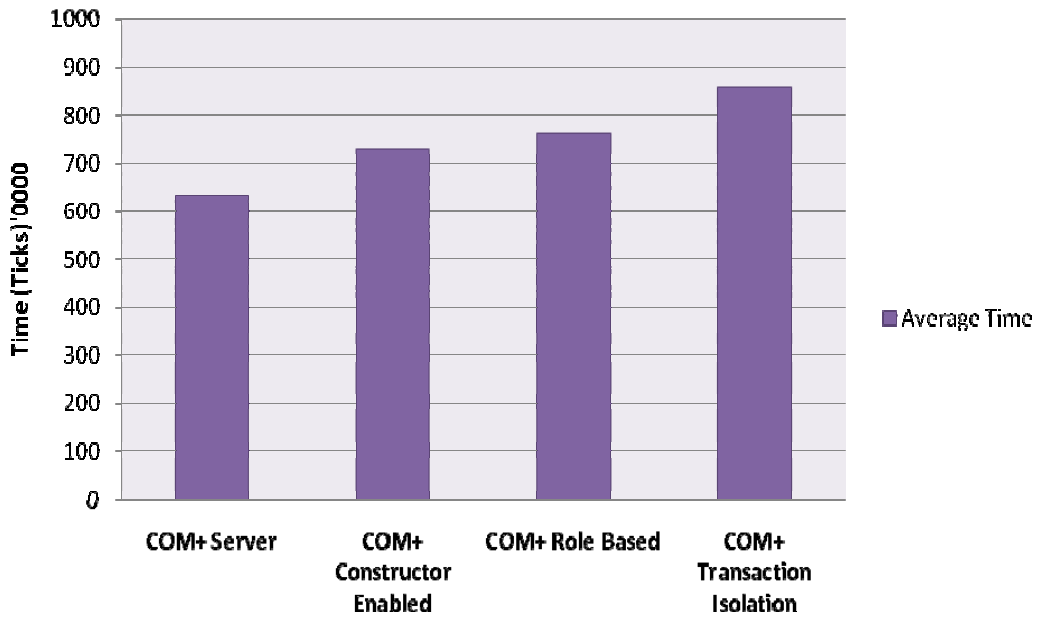


Chart 6.3-7 COM+ Component features and their Performance

	COM+ Server	COM+ Constructor Enabled	COM+ Role Based	COM+ Transaction Isolation
Avg. Time(Ticks)'0000	631	731	761	861

Table 6.3.7 COM+ Component features Data Performance

The chart 6.3.8 shows the results of non - COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and minor performance gain on high volume of data under 3.0 Framework. As per the table 6.3.8, it has been noticed that medium volume of data took around 43% of more time on 3.0 to perform the operation as compared to time taken by 2.0.

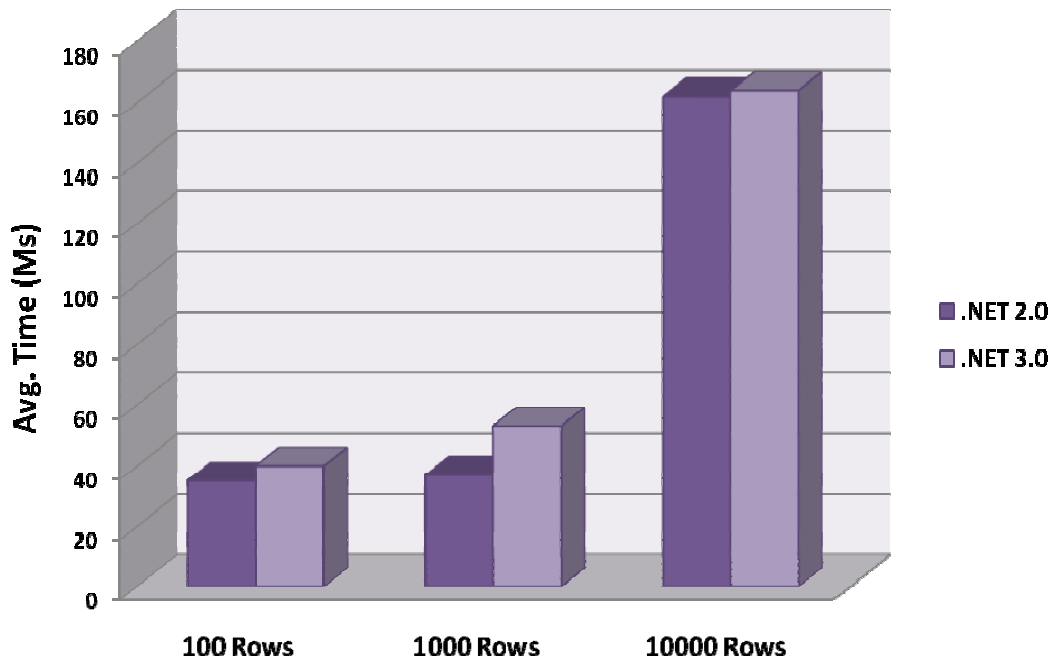


Chart 6.3-8 Non-COM+ component performance on SQL Server (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	35	37	162
.NET 3.0	40	53	160
% Difference	14.29	43.24	1.23

Table 6.3.8 Non-COM+ component Data performance on SQL Server (5 Users)

The chart 6.3.9 shows the results of non - COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the ten database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and minor performance gain on high volume of data under 3.0 Framework. As per the table 6.3.6, it has been noticed that high volume of data provides better performance results with the .NET 3.0 Framework, which reduced the time taken to around 14%.

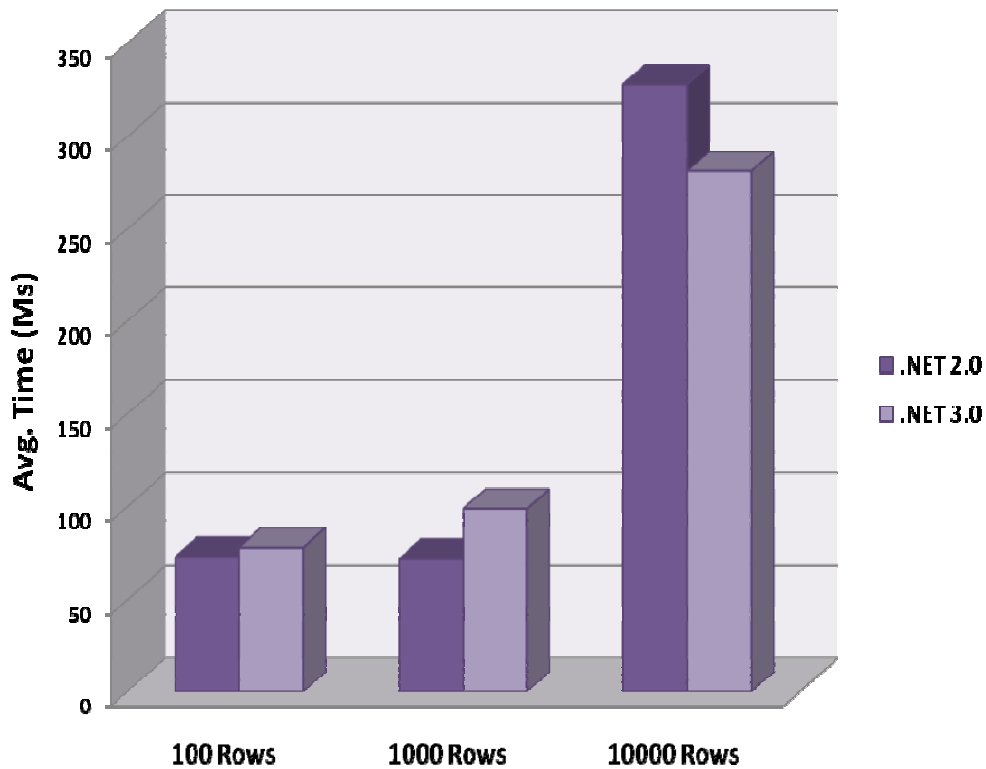


Chart 6.3-9 Non-COM+ component performance on SQL Server (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	73	72	328
.NET 3.0	78	99	281
% Difference	6.85	37.50	-14.33

Table 6.3.9 Non-COM+ component Data performance on SQL Server (10 Users)

The chart 6.3.10 shows the results of COM+ component with pooling and JIT tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using five database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and minor performance gain on high volume of data under 3.0 Framework. As per the table 6.3.10, it has been noticed that high volume of data provides better performance results with the .NET 3.0 Framework, which provides performance gain of 7%.

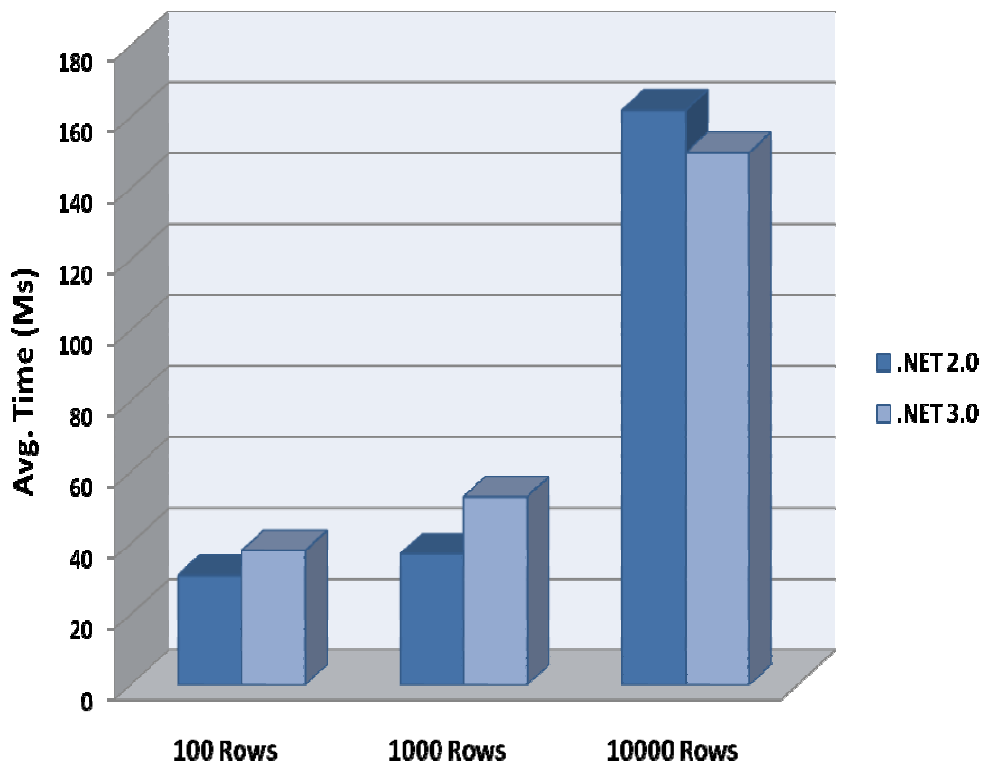


Chart 6.3-10 COM+ component performance on SQL Server (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	31	37	162
.NET 3.0	38	53	150
% Difference	22.58	43.24	-7.41

Table 6.3.10 COM+ component Data performance on SQL Server (5 Users)

The chart 6.3.11 shows the results of COM+ component with pooling and JIT tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using ten database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and minor performance gain on high volume of data under 3.0 Framework. As per the table 6.3.11, it has been noticed that high volume of data provides better performance results with the .NET 3.0 Framework, which provides performance gain of 12%.

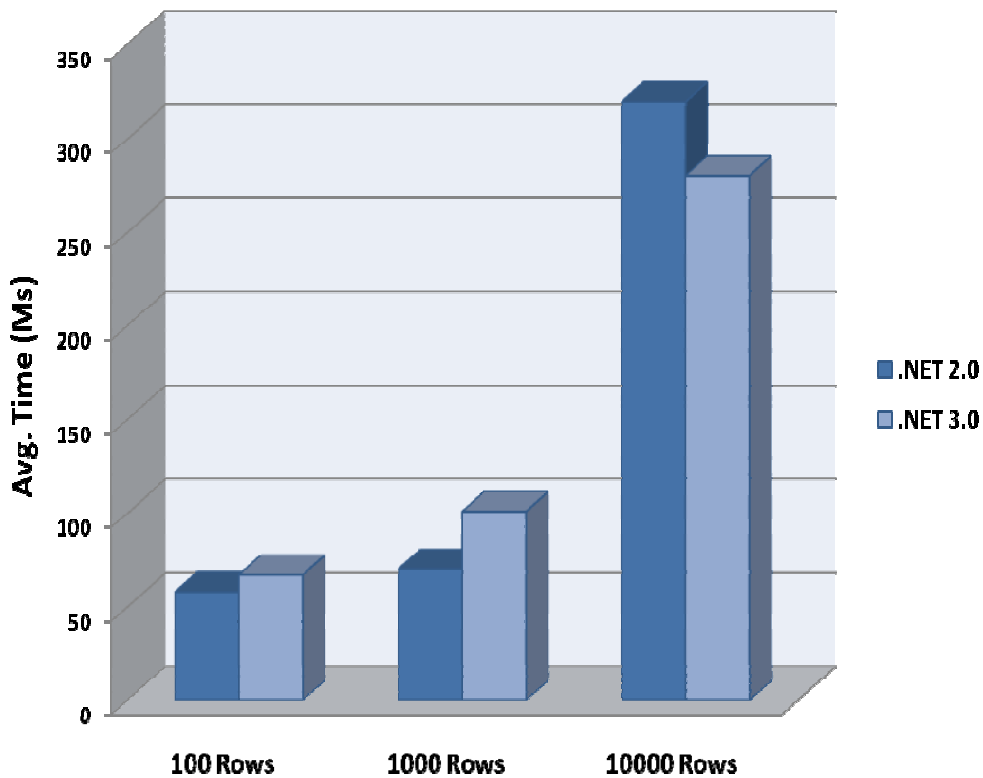


Chart 6.3-11 COM+ component performance on SQL Server (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	58	70	320
.NET 3.0	67	101	280
% Difference	15.52	44.29	-12.50

Table 6.3.11 COM+ component Data performance on SQL Server (10 Users)

The chart 6.3.12 shows the results of Microsoft .NET connection pooling component tested on the two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and significant performance gain on high volume of data under 3.0 Framework. As per the table 6.3.12, it has been noticed that high volume of data provides better performance results with the .NET 3.0 Framework, which provides performance gain of around 13%.

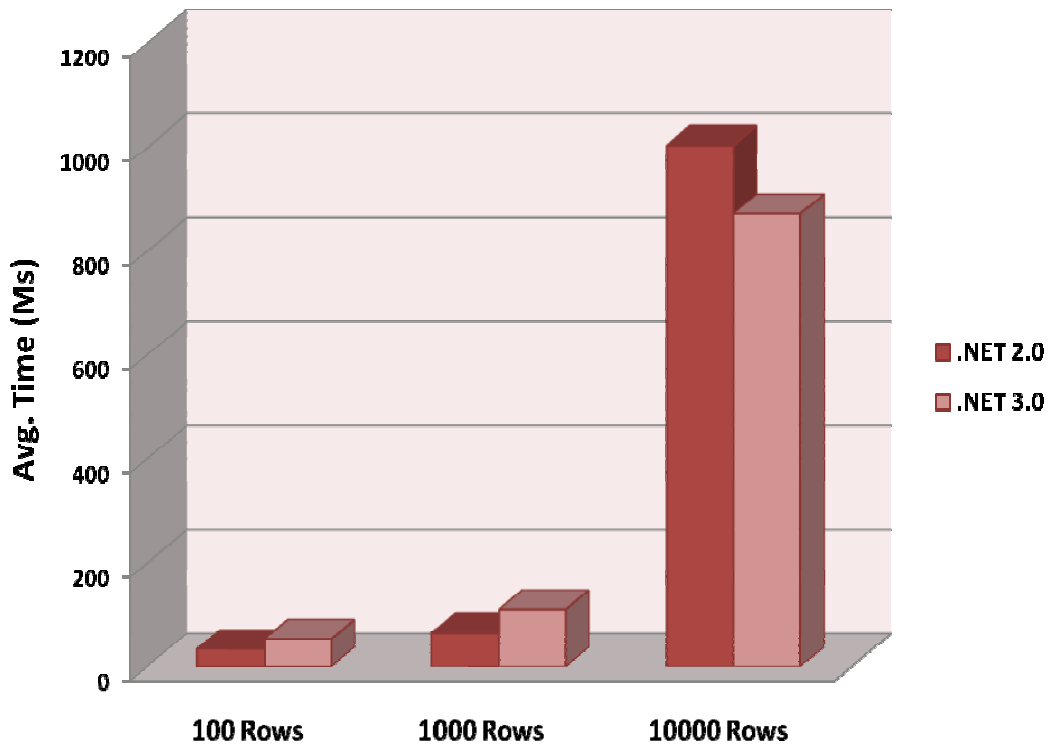


Chart 6.3-12 .NET component performance on SQL Server (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	34	66	1005
.NET 3.0	54	110	873
% Difference	58.82	66.67	-13.13

Table 6.3.12 .NET component data performance on SQL Server (5 Users)

The chart 6.3.13 shows the results of COM+ component with pooling and JIT tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using ten database users. The results represents that Framework 2.0 provides better results over 3.0 on low and medium volume of data and significant performance gain on high volume of data under 3.0 Framework. As per the table 6.3.13, it has been noticed that high volume of data provides better performance results with the .NET 3.0 Framework, which provides performance gain of around 15%.

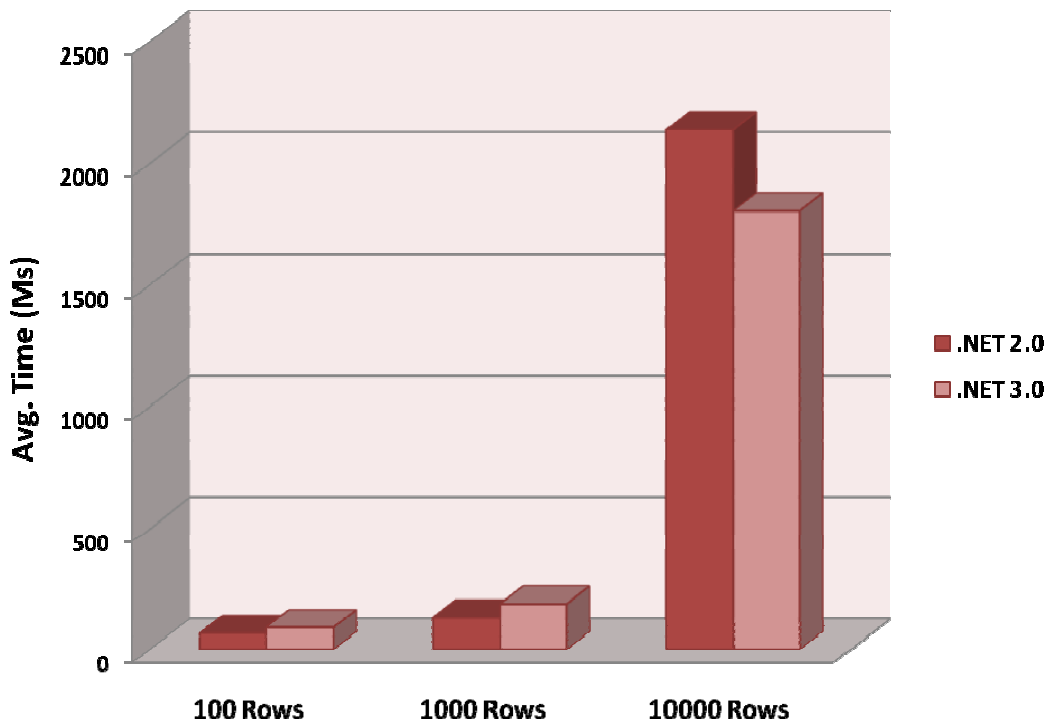


Chart 6.3-13 .NET component performance on SQL Server (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	67	132	2140
.NET 3.0	91	188	1804
% Difference	35.82	42.42	-15.70

Table 6.3.13.NET component data performance on SQL Server (10 Users)

The chart 6.3.14 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of low volume (100 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application.

As per the table 6.3.14, it has been noticed that COM+ based application took 38ms for the same database read operation as compared to 54ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application was around 59%.

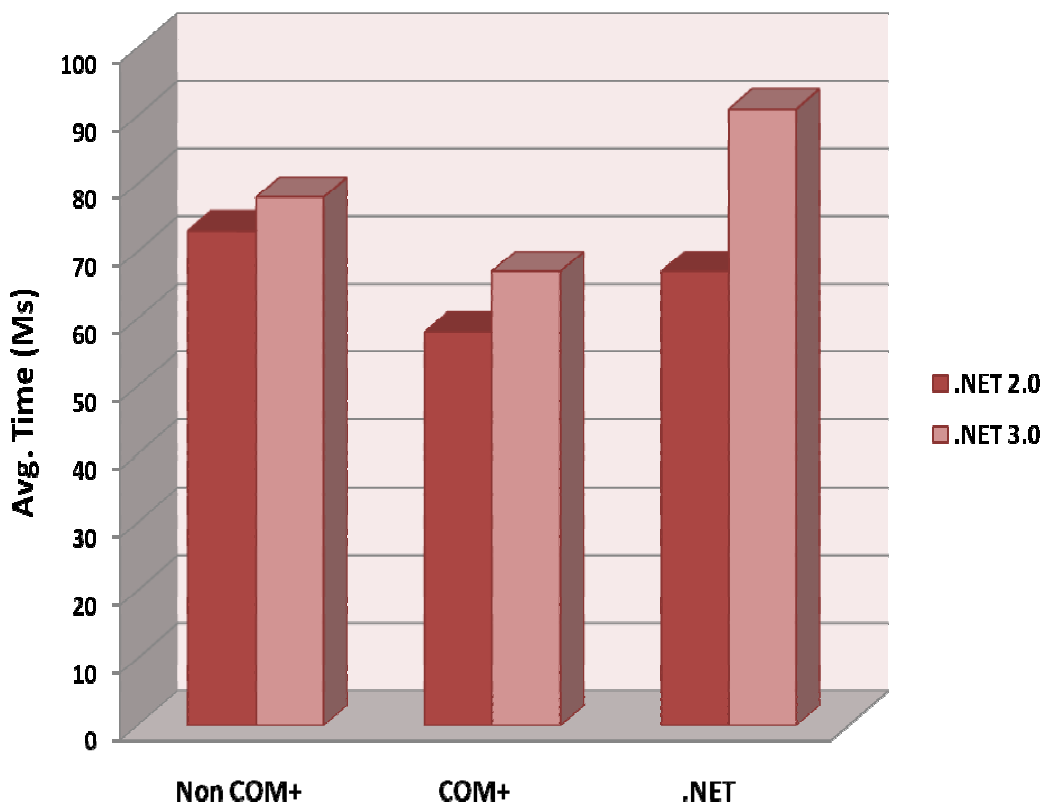


Chart 6.3-14 Application performance on SQL Server (5 Users and 100 Rows)

	Non COM+	COM+	.NET
.NET 2.0	35	31	34
.NET 3.0	40	38	54
% Difference	14.29	22.58	58.82

Table 6.3.14 Application performance data on SQL Server (5 Users and 100 Rows)

The chart 6.3.15 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of low volume (100 Rows). The results were based on the trivial database read method performed using the ten database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application.

As per the table 6.3.15, it has been noticed that COM+ based application took 67ms for the same database read operation as compared to 91ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application was around 35%.

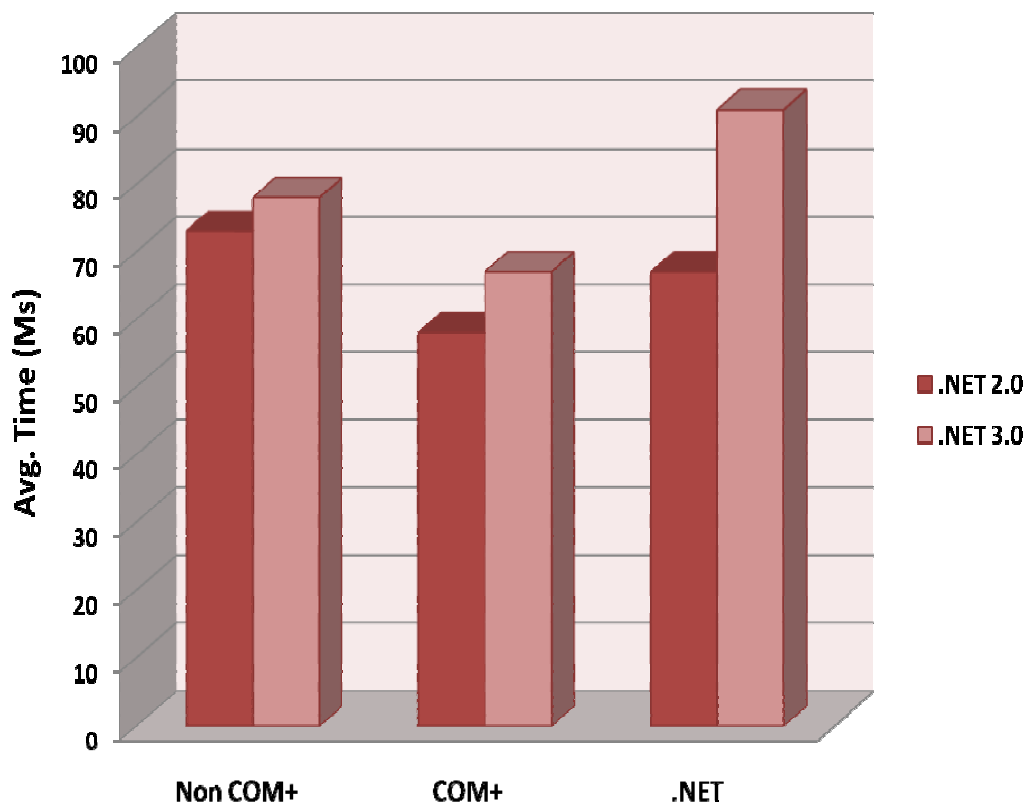


Chart 6.3-15 Application performance on SQL Server (10 Users and 100 Rows)

	Non COM+	COM+	.NET
.NET 2.0	73	58	67
.NET 3.0	78	67	91
% Difference	6.85	15.52	35.82

Table 6.3.15 Application performance data on SQL Server (10 Users and 100 Rows)

The chart 6.3.16 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of medium volume (1000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application.

As per the table 6.3.16, it has been noticed that COM+ based application took 53ms for the same database read operation as compared to 110ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application was around 66%. Moreover Non-COM+ and COM+ based application showed the same performance.

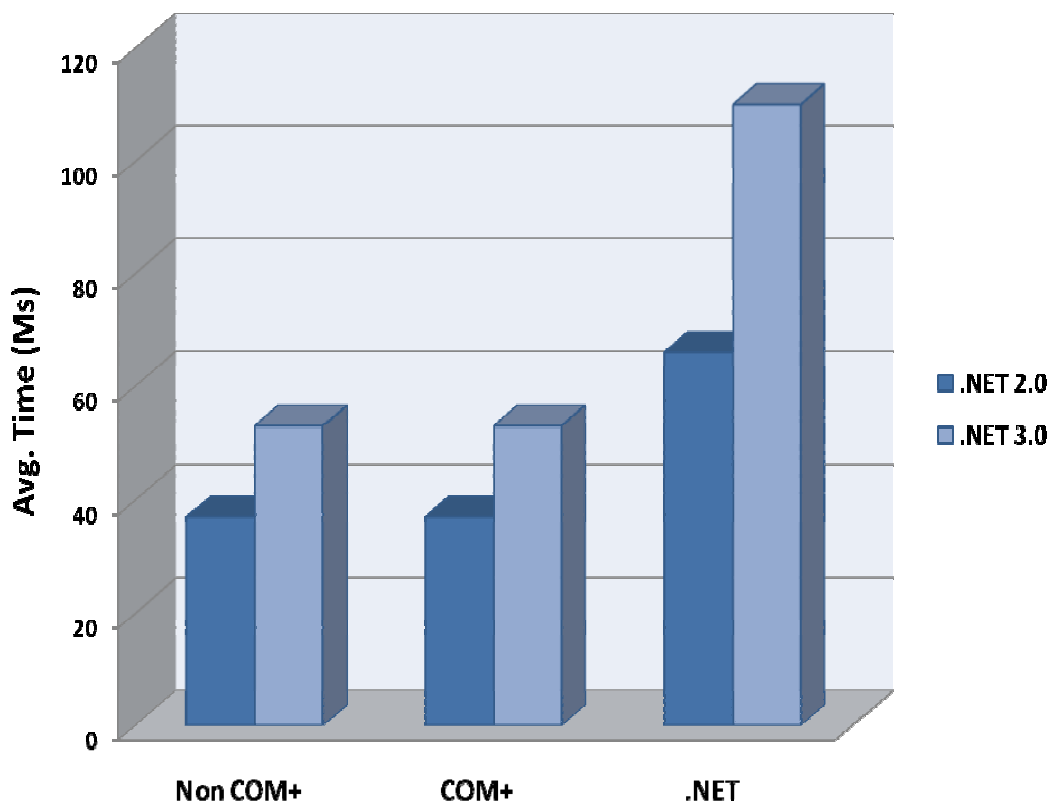


Chart 6.3-16 Application performance on SQL Server (5 Users and 1000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	37	37	66
.NET 3.0	53	53	110
% Difference	43.24	43.24	66.67

Table 6.3.16 Application performance data on SQL Server (5 Users and 1000 Rows)

The chart 6.3.17 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of medium volume (1000 Rows). The results were based on the trivial database read method performed using the ten database users. The results represents that COM+ and Non COM+ component based application provides better performance results over .NET based application.

As per the table 6.3.17, it has been noticed that COM+ based application took 101ms for the same database read operation as compared to 188ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application was around 42%. Moreover Non-COM+ provides better results on .Framework 3.0.

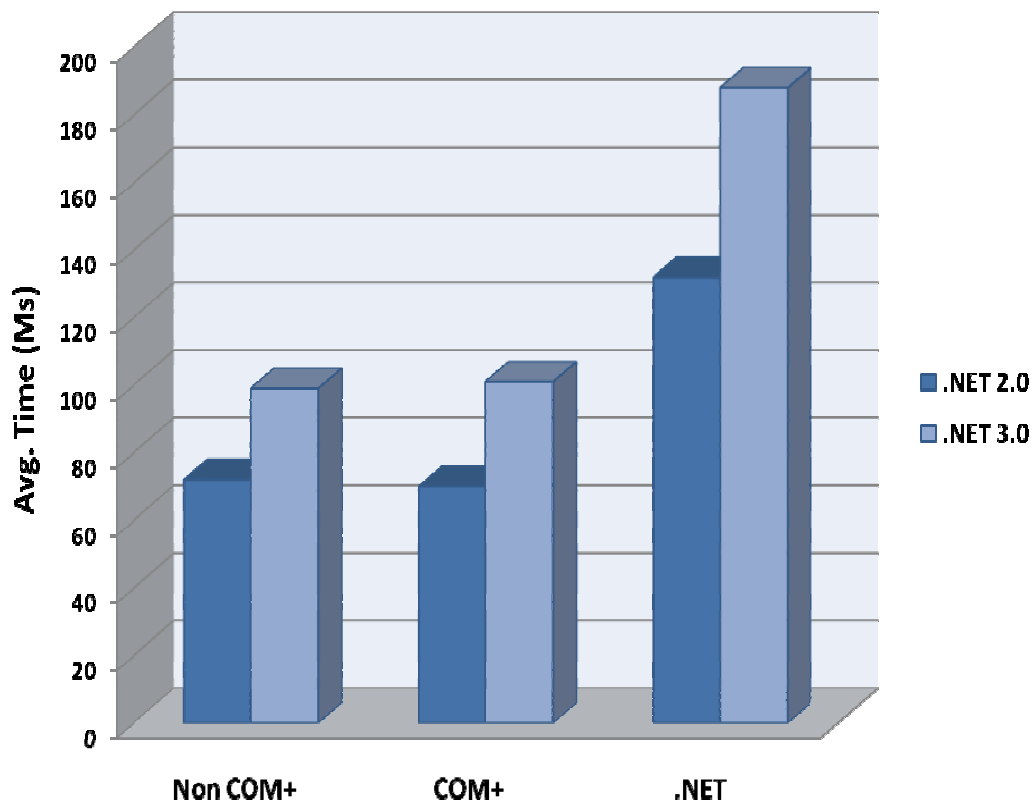


Chart 6.3-17 Application performance on SQL Server (10 Users and 1000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	72	70	132
.NET 3.0	99	101	188
% Difference	37.50	44.29	42.42

Table 6.3.17 Application performance data on SQL Server (10 Users and 1000 Rows)

The chart 6.3.18 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of high volume (10000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application.

As per the table 6.3.18, it has been noticed that there was a significant performance gain of around 7% and 13% using COM+ and .NET based application on Framework 3.0. Therefore results showed that Framework 3.0 was optimized for the high volume of database operations.

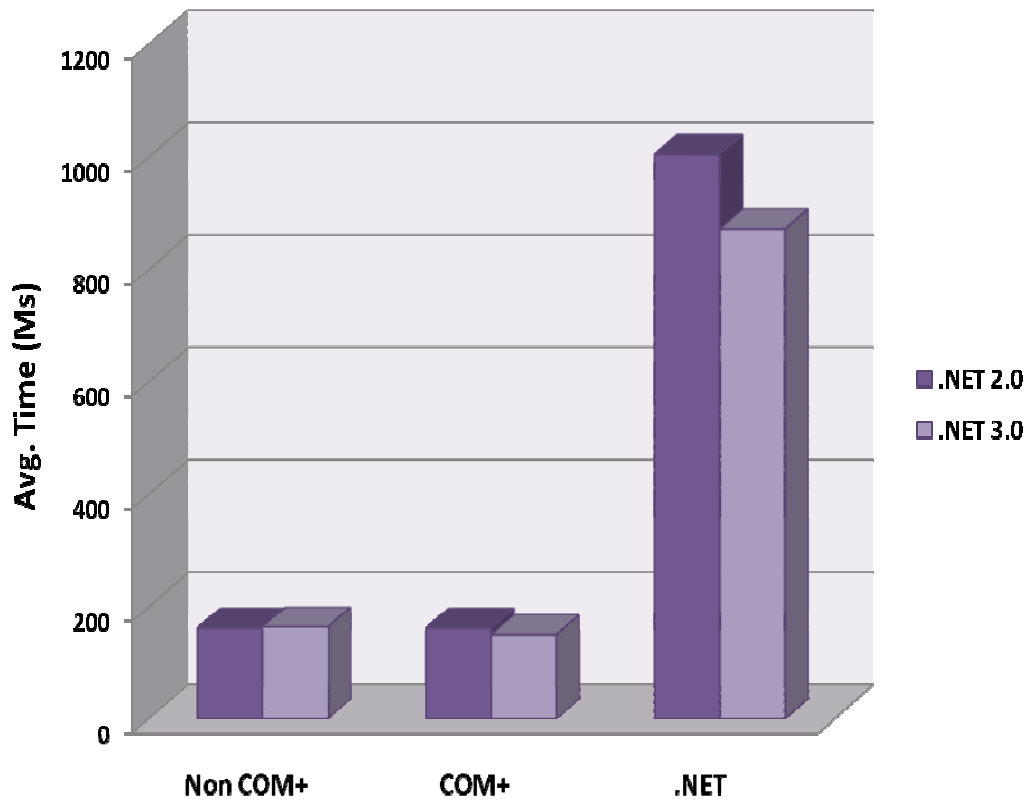


Chart 6.3-18 Application performance on SQL Server (5 Users and 10000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	162	162	1005
.NET 3.0	164	150	873
% Difference	1.23	-7.41	-13.13

Table 6.3.18 Application performance data on SQL Server (5 Users and 10000 Rows)

The chart 6.3.19 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of high volume (10000 Rows). The results were based on the trivial database read method performed using the ten database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application.

As per the table 6.3.19, it has been noticed that there was a significant performance gain of around 14% and 12% using Non COM+ and COM+ based application on Framework 3.0. However the .NET based application were taking more time on high volume of data compared with COM+ based application.

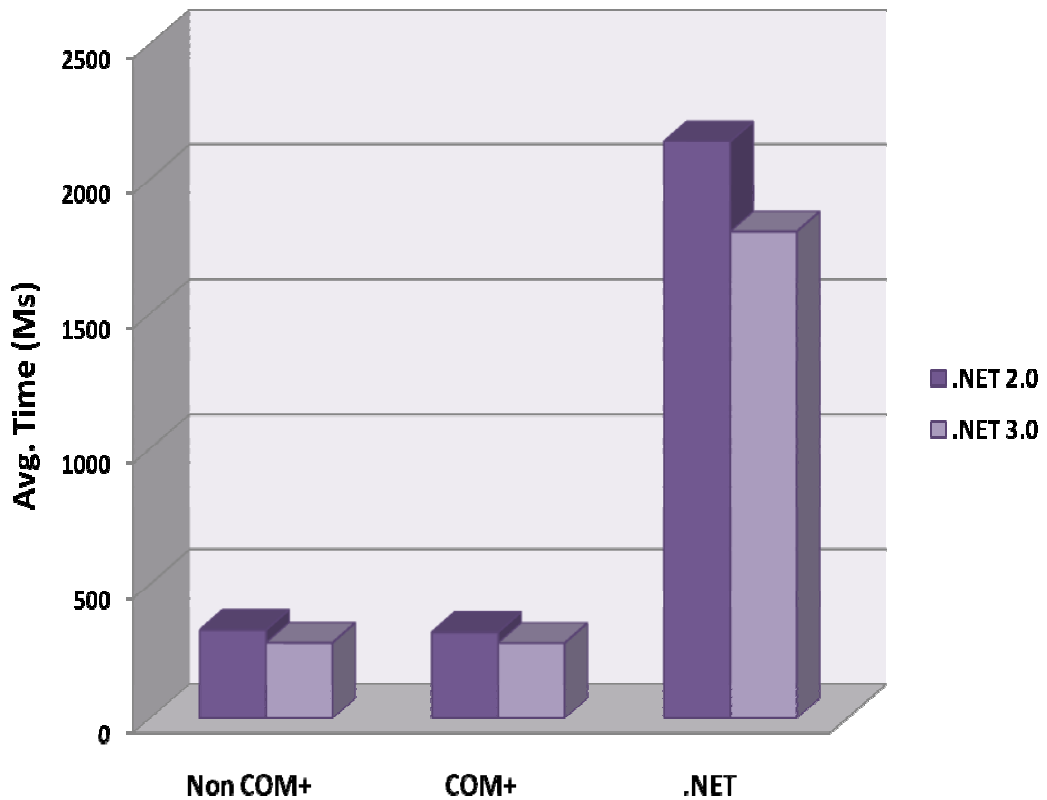


Chart 6.3-19 Application performance on SQL Server (10 Users and 10000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	328	320	2140
.NET 3.0	281	280	1804
% Difference	-14.33	-12.50	-15.70

Table 6.3.19 Application performance data on SQL Server (10 Users and 10000 Rows)

6.4 MS Access Experiments Results

Following are the series of experiments conducted using the developed prototype application on Microsoft Access database.

In the charts through 6.4.1 to 6.4.6, we have performed the test on MS Access database. The results represents that MS Access database performance tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five or ten database users.

In the charts through 6.4.7 to 6.4.8, we have performed the test on MS Access database using the different application types. The results represents that MS Access database performance tested on two different Microsoft .NET Framework (version 2.0 and 3.0).

The chart 6.4.1 shows the results of non - COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five users. The results represents that Framework 2.0 provides better results over 3.0 on the entire three data volume category. As per the table 6.4.1, it has been noticed that the maximum performance difference between the two frameworks were around 280%.

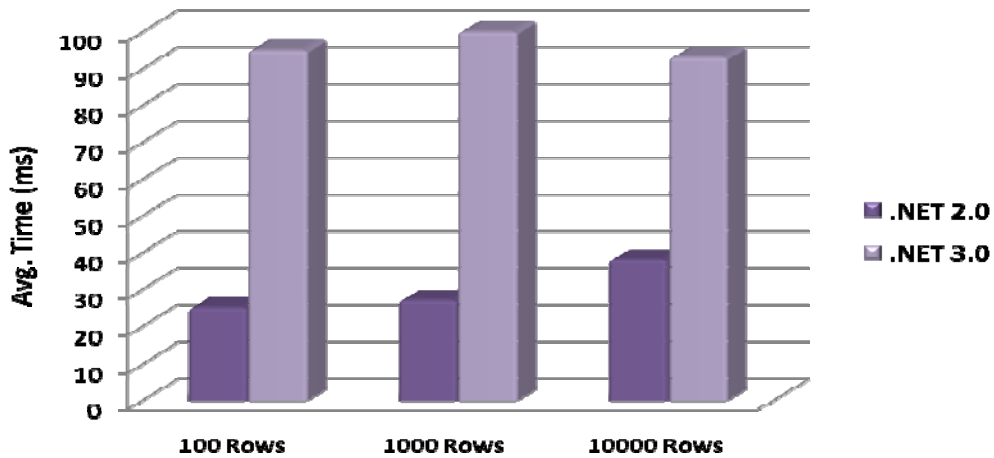


Chart 6.4-1 Non-COM+ component performance on MS Access (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	25	27	38
.NET 3.0	95	100	93
%Difference	280	270	144

Table 6.4.1 Non-COM+ component performance data on MS Access (5 Users)

The chart 6.4.2 shows the results of non - COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the ten users. The results represents that Framework 2.0 provides better results over 3.0 on the entire three data volume category. As per the table 6.4.1, it has been noticed that the maximum performance difference between the two frameworks were around 250%.

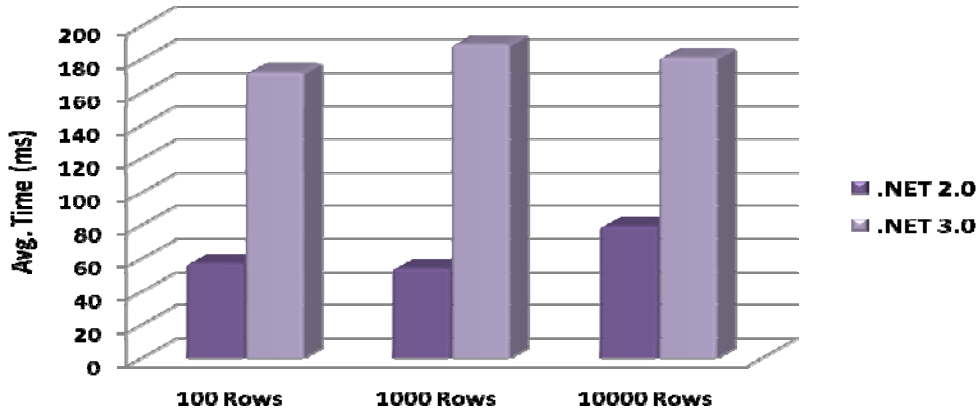


Chart 6.4-2 Non-COM+ component performance on MS Access (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	57	54	79
.NET 3.0	172	189	181
%Difference	201	250	129

Table 6.4.2 Non-COM+ component performance data on MS Access (10 Users)

The chart 6.4.3 shows the results of COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five users. The results represents that Framework 2.0 provides better results over 3.0 on the entire three data volume category. As per the table 6.4.3, it has been noticed that the maximum performance difference between the two frameworks were around 116%.

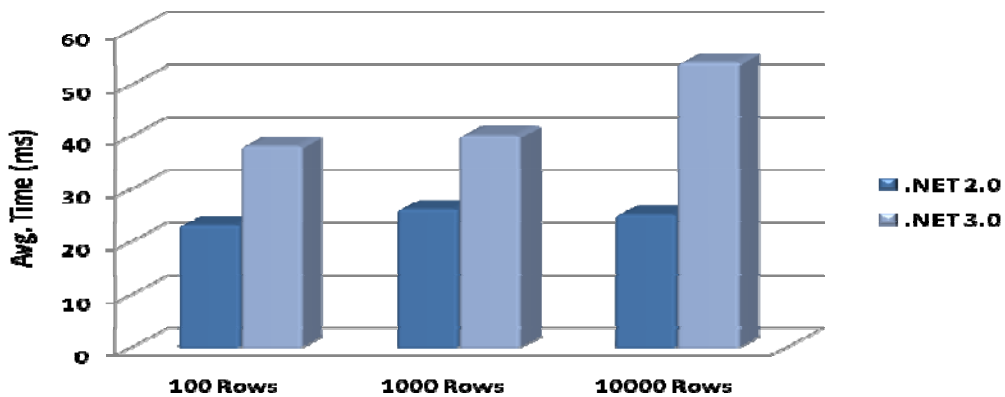


Chart 6.4-3 COM+ component performance on MS Access (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	23	26	25
.NET 3.0	38	40	54
%Difference	65	53	116

Table 6.4.3 COM+ component performance data on MS Access (5 Users)

The chart 6.4.4 shows the results of COM+ component tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the ten users. The results represents that Framework 2.0 provides better results over 3.0 on the entire three data volume category. As per the table 6.4.4, it has been noticed that the maximum performance difference between the two frameworks were around 235%

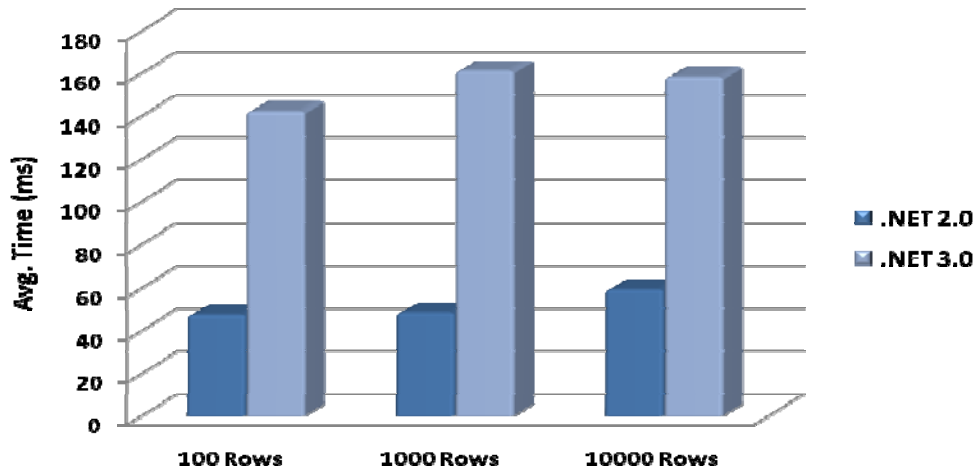


Chart 6.4-4 COM+ component performance on MS Access (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	47	48	59
.NET 3.0	142	161	158
%Difference	202	235	167

Table 6.4.4 COM+ component performance data on MS Access (10 Users)

The chart 6.4.5 shows the results of .NET based application tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five users. The results represents that Framework 2.0 provides better results over 3.0 on the low volume of data and almost similar on medium volume of data. As per the table 6.4.5, it has been noticed that the Framework 3.0 better results on high volume of data.

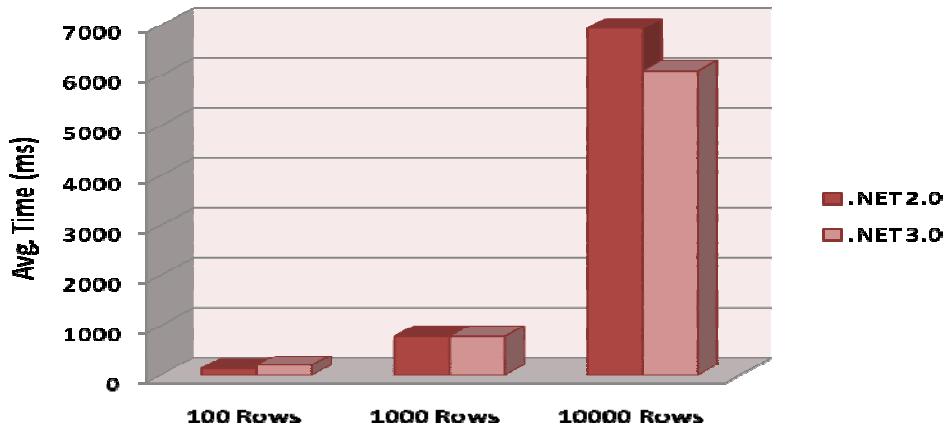


Chart 6.4-5 .NET Application performance on MS Access (5 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	132	792	6940
.NET 3.0	204	792	6099
%Difference	54.55	0.00	-12.12

Table 6.4.5 .NET Application performance data on MS Access (5 Users)

The chart 6.4.6 provides the nearly the same results performed using the ten users. The results represents that Framework 2.0 provides better results over 3.0 on the low volume of data and almost similar on medium volume of data. As per the table 6.4.5, it has been noticed that the Framework 3.0 better results on high volume of data.

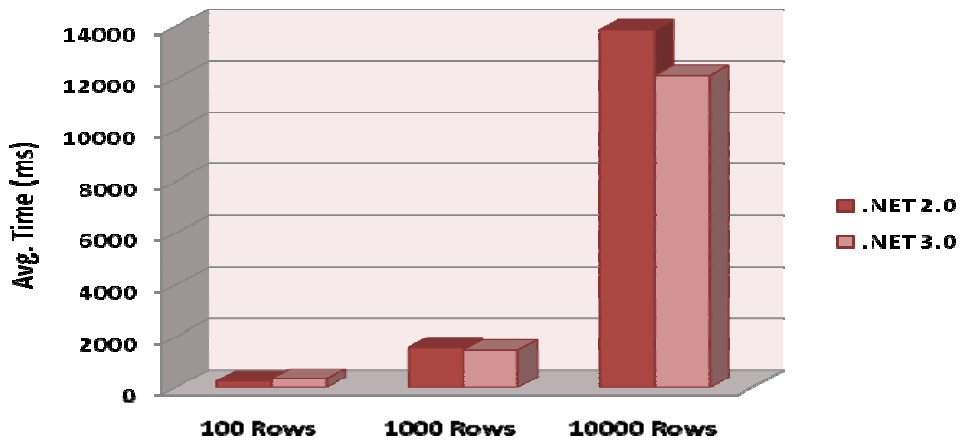


Chart 6.4-6 .NET Application performance on MS Access (10 Users)

	100 Rows	1000 Rows	10000 Rows
.NET 2.0	259	1575	13865
.NET 3.0	353	1481	12132
%Difference	36.29	-5.97	-12.50

Table 6.4.6 .NET Application performance data on MS Access (10 Users)

The chart 6.4.7 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of low volume (100 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application on MS Access database. As per the table 6.4.7, it has been noticed that COM+ based application took 38ms for the same database read operation as compared to 204ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application is around 280% on using non COM+ application.

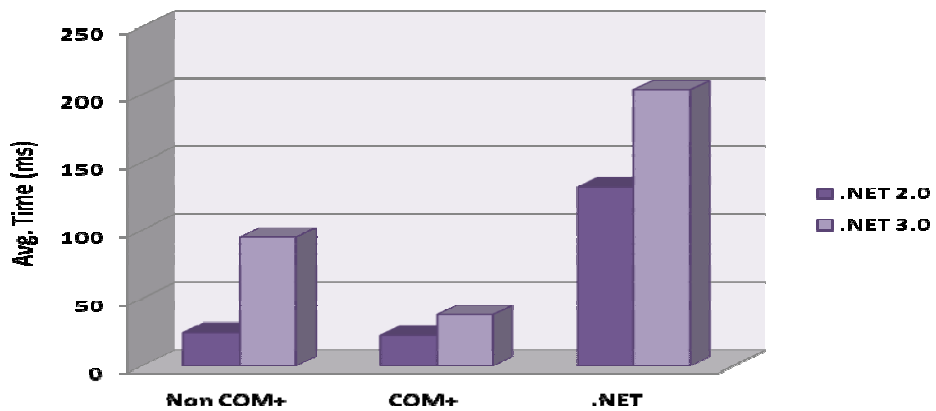


Chart 6.4-7 Application performance on MS Access (5 Users and 100 Rows)

	Non COM+	COM+	.NET
.NET 2.0	25	23	132
.NET 3.0	95	38	204
%Difference	280.00	65.22	54.55

Table 6.4.7 Application performance data of MS Access (5 Users and 100 Rows)

The chart 6.4.8 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of low volume (100 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application on MS Access database. As per the table 6.4.8, it has been noticed that COM+ based application took 38ms for the same database read operation as compared to 204ms taken by .NET application on Framework 3.0 and the performance difference between 2.0 and 3.0 Framework for the NET application is around 280% on using non COM+ application.

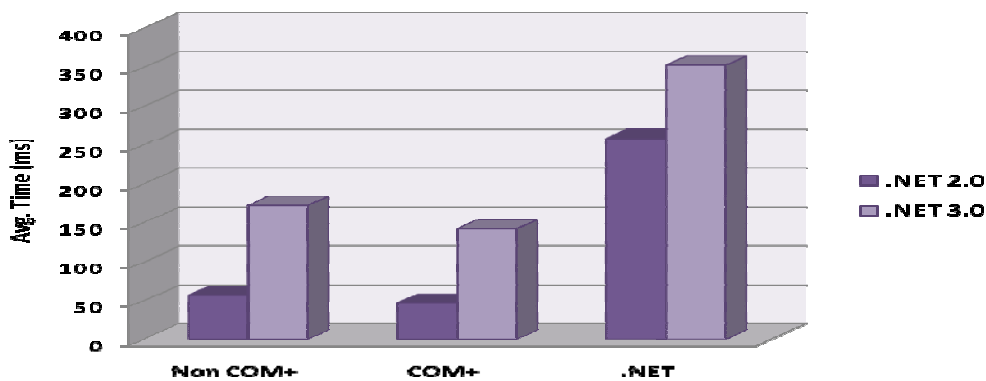


Chart 6.4-8 Application performance on MS Access (10 Users and 100 Rows)

	Non COM+	COM+	.NET
.NET 2.0	57	47	259
.NET 3.0	172	142	353
%Difference	201.75	202.13	36.29

Table 6.4.8 performance

Application data of MS

Access (10 Users and 100 Rows)

The chart 6.4.9 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of medium volume (1000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application on MS Access database. As per the table 6.4.9, it has been noticed that there is not much performance difference between .NET application performance running on 2.0 and 3.0 Framework.

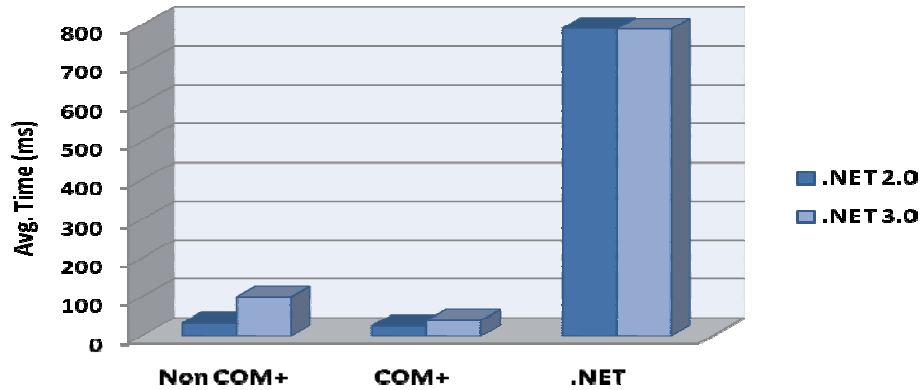


Chart 6.4-9 Application performance on MS Access (5 Users and 100 Rows)

	Non COM+	COM+	.NET
.NET 2.0	33	26	792
.NET 3.0	100	40	790
%Difference	203	53	-0.25

Table 6.4.9 Application performance data of MS Access (5 Users and 1000 Rows)

The chart 6.4.10 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of medium volume (1000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ and .NET based application on MS Access database. As per the table 6.4.10, it has been noticed that there is slight performance difference between .NET framework 2.0 and 3.0.

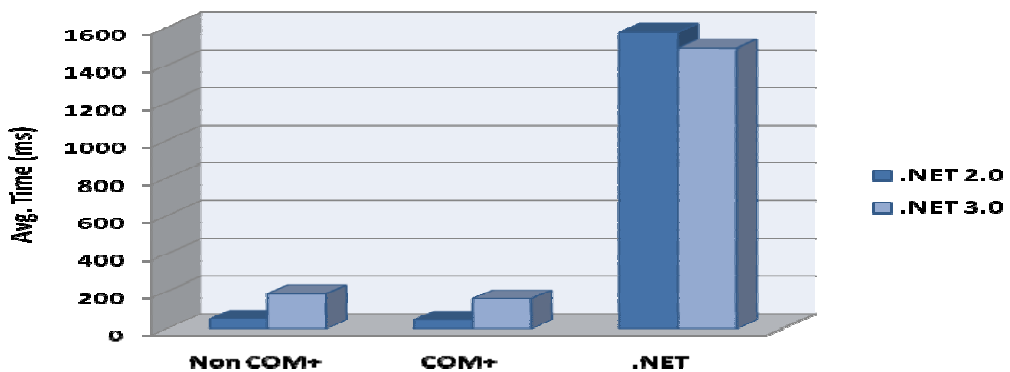


Chart 6.4-10 Application performance on MS Access (10 Users and 1000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	54	48	1575
.NET 3.0	189	161	1491
%Difference	250	235	-5

Table 6.4.10 Application performance data of MS Access (10 Users and 1000 Rows)

The chart 6.4.11 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with five database users for each database hit of high volume (10000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ only on Framework 2.0. As per the table 6.4.11, it has been noticed that there is slight performance gain for .NET based application running on framework 3.0 accessing high volume of data.

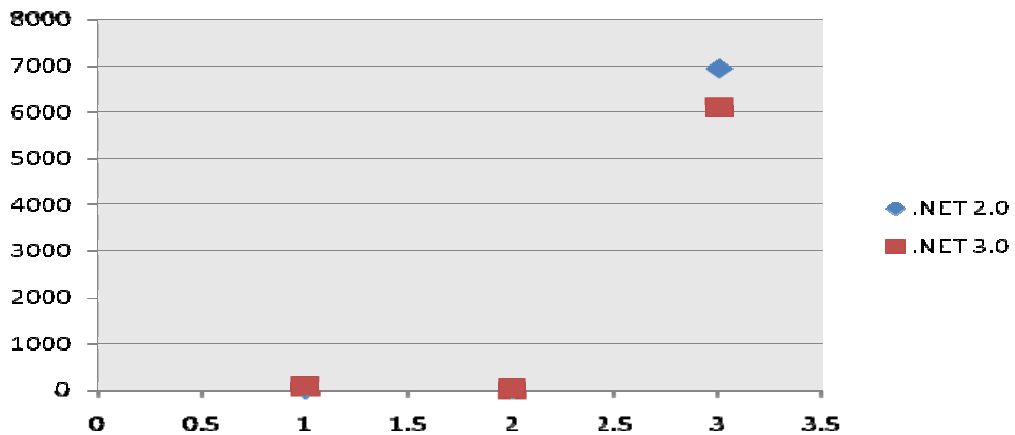


Chart 6.4-11 Application performance on MS Access (5 Users and 10000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	38	25	6940
.NET 3.0	93	54	6099
%Difference	144.74	116.00	-12.12

Table 6.4.11 Application performance data of MS Access (5 Users and 10000 Rows)

The chart 6.4.12 shows the results of the different application type performance which were tested on the two different Microsoft .NET Framework (version 2.0 and 3.0) along with ten database users for each database hit of high volume (10000 Rows). The results were based on the trivial database read method performed using the five database users. The results represents that COM+ component based application provides better performance results over non-COM+ only on Framework 2.0. As per the table 6.4.12, it has been noticed that there was a performance gain of around 12% for .NET based application running on framework 3.0 accessing high volume of data.

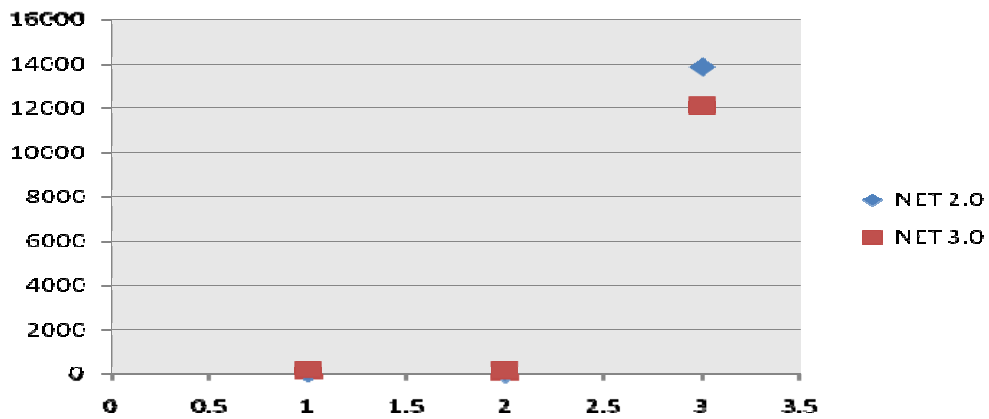


Chart 6.4-12 Application performance on MS Access (10 Users and 10000 Rows)

	Non COM+	COM+	.NET
.NET 2.0	79	59	13865
.NET 3.0	181	158	12132
%Difference	129.11	167.80	-12.50

Table 6.4.12 Application performance data of MS Access (10 Users and 10000 Rows)

6.5 Conclusion

This chapter was aimed to evaluate the experiments conducted on SQL Server and MS Access database using the COM+ services. SQL Server database was being benchmarked with five or ten database users on .NET Framework 2.0 and 3.0 and also on different volume of data. Similarly MS Access also being benched marked with five or ten users on different .NET Framework 2.0 and 3.0 with different volume of data. Both the SQL Server and Access databases were also benchmarked using the different application types i.e. COM+, non-COM+ and .NET based application and on different user and data volume load.

The SQL Server tested on the different .NET Framework, which concludes in the results that .NET Framework 2.0 provides better results on the low and medium volume of data although Framework 3.0 provides better performance results on the high volume (10000 Rows) of data around 10% of performance gain over Framework 2.0. COM+ based application provides the better performance results for the low and medium volume of data nearly the same performance response for the non COM+ based application. The .NET based application provides the performance gain when high volume database operation is performed.

The MS Access database performance tested on two different Microsoft .NET Framework (version 2.0 and 3.0). The results were based on the trivial database read method performed using the five or ten database users. It is clearly visible from the results that Framework 2.0 provides better performance over 3.0 on low and medium volume of data and Framework 3.0 provides major performance gain on high volume of data.

Moreover it has been noticed from the experiments conducted on MS Access that the results provides the same performance which we have seen in the past experiments on SQL Server except the significant performance gain on .NET application running on Framework 3.0 when accessing the high volume data (10000 Rows) and COM+ application have provided better performance results over the non-COM+ and .NET based application.

7 Conclusion

7.1 Introduction

This thesis aimed at analyzing and optimizing the database performance using enhanced object model in the .NET Framework. This is carried out by implementing a prototype application which integrated the COM+ pooled component, COM+ non pooled component and .NET connection pooling projects to measure the performance. The Microsoft SQL Server and MS Access databases were used to benchmark and analyze the database performance under the different experimental conditions.

This chapter provides the critical analysis of the whole project that includes both design and implementation cycle. Suggestions on future work that could be carried out further are also included. In addition, it also highlights other areas of technology in which technology in which next version of application or experiment could be performed.

7.2 Conclusion

This thesis was aimed to show how COM+ services can be used to analyze and optimize the database performance. The experiments were based on developed prototype windows based application having the trivial database read method for both the databases, option to increase or decrease the user load and volume of data using different application types. One of the initial experiments showed the impact of using the COM+ server and COM+ library application, and their performance based on both the databases. The basic task like component initialization, role bases, constructor initialization, component task and other we can use COM+ server application to save time in writing the code, programmers efforts and provides better results over the COM+ library application.

COM+ can provides the supply of powerful services that can help to create quickly sophisticated and stable application. The main drawback of using COM+ services is performance cost. We saw that we can use part of COM+ services like Object Pooling, JITA, application partitioning and role based security and COM+ library application with acceptable cost. COM+ server that provides most of the interesting services has bad influence over performance mainly due to its usage of DCOM. In the experiments conducted in this thesis it has been noticed that COM+ based application provides better results as compared to the .NET based application when retrieving the large volume of data from the database.

The initial experiments showed that COM+ library application took more time to process the request as compared to the server application under the .NET environment and on a ten database user connection server application with no pooling and JIT performed twice faster than the library application. The server application provides better results and performance gain of more than 10% even when the object pooling and JIT is enabled. The server application was the better choice over library application which performed the trivial database read operation. COM+ component using the pooling and JIT provide better results and were twice

faster than the non-COM+ component. As features comes at the cost, on the concurrent user connections component with the role base security feature was taking 20% more time than component not using role based security. Similarly COM+ component with the transaction isolation and constructor enabled property was taking around 35% and 20% more time over the component with pooling and JIT.

The performance of non-COM+ component on SQL Server low and medium volume of data with five users showed that framework 2.0 provides better results over 3.0 and on the high volume of data with ten users framework 3.0 provides the performance gain of around 14% over 2.0 component. The performance of COM+ component on SQL Server low and medium volume of data with five users showed that framework 2.0 provides better results over 3.0 and there is also a slight performance gain on 3.0 over 2.0 for medium volume of data and on the high volume of data with ten users .NET Framework 3.0 provides the performance gain of around 12% over 2.0 component. This shows that Framework 3.0 is optimized for the high end application having high volume of data.

7.3 Critical Analysis

The objective of this thesis is to analyse and optimize the performance of database using the enhanced object model under the .NET Framework. Initially various experiments has been conducted on the COM+ application type which includes the COM+ Server application and COM+ Library application. During the initial design phase of the prototype application, the interface was simple with basic controls to initiate and process the request. However once the test or experiments has been incorporated, we altered the basic interface to include rich user options thus facilitating the user to test the database using different number of user connections and application types. While performing experiments, we faced problems in accessing MS Access database with more number of users. This resulted because we were using the .NET Framework SQL class libraries instead of using OLEDB connection which hindered the performance of the overall result. We have overcome this issue by using Microsoft OLEDB class libraries on different volumes of data. The successful experiments were conducted on different .NET Framework versions 2.0 and 3.0. However we have not tested the performance for the conducted experiments on early versions of .NET Framework which can be considered in the next version of this prototype application. Moreover the time taken and calculated for all the experiments performed was comes as an average of five consecutive hits as per user input instead of using two average hits data, this resulted in providing consistent median time for various experiments.

The challenge was to find the median and the variance of the conducted experiments which was performed manually in early phase of design. Consequently, we decided and implemented the whole calculations for median and variance in the application itself to avoid manual process of calculation. This saved not only the time but also the efforts required in the manual process and also provided the application performance results on a click of a button. Also the C# .NET code has been implemented with structured exception handling. The application shows customised error messages that contain relevant information regarding the exceptions. The messages are easy to understand by a normal user using the prototype application as well as helpful for the developer to quickly trace the root of exception.

It has been noticed from the experiments conducted that COM+ Services feature comes at a cost as COM+ component with role based security and transaction isolation comes with the performance penalty to the operating system. Also writing a .NET managed code provides the ease to the user to develop the application in a very short span of time as compared to COM+ application in the unmanaged environment. Although we can use .NET namespace for creating COM+ components but it takes time and resources. Therefore the trivial database read operation method has been tested on .NET based application and COM+ based application, which resulted in that COM+ based application provide better results over .NET based application. So the application scope and requirements must be analysed before choosing the application type and .NET Framework.

7.4 Future Work

As the prototype application was architect around COM+ services and developed using the C# .NET language. There will be a scope for the future work to be performed on Microsoft .NET Framework 3.5 to analysis and assess the performance of the database performance. The SQL Server new version can be benchmarked using the different load based testing and other databases can be used to benchmark the performance of the COM+ services which includes object pooling, JIT, transaction isolation property and application recycling.

The experiments could be taken to the other platforms for their cross platform performance and could be tested on the family of windows operating systems. The following are the category where future work can be performed.

- Using the new .NET Framework and comparative study on the performance of the previous versions.
- Conducting further experiments on other database like Oracle. MS access 2007, MY SQL and DB2.
- Implementation of COM+, non-COM+ and .NET based application on 64-bit machine architecture.
- Implementation of COM+, non-COM+ and .NET based application on COM+ components on different operating systems.

The above mentioned future area of work is wide in their operation but the features of the COM+ should be assessed for usefulness.

8 References

- Bayer, D. (2001). *C# COM+ Programming*. New York, NY: M&T Books.
- Blakeley, J. (1997). Universal Data Access with OLE DB. Proceedings of COMPCON 97 , 2.
- Brill, G. (2000). *Applying COM+*. SAMS.
- Corporation, M. (1998). Microsoft Developer Network. Retrieved 11 August, 2007, from Transactional Component Services: A Guide to Reviewing Microsoft Transaction Server 2.0: <http://msdn2.microsoft.com/en-us/library/ms810020.aspx>
- Deshpande, A., & Blakeley, J. A. (2000). Data Access. Proceedings of the 2000 ACM SIGMOD international conference on Management of data (p. 579). Texas: ACM Press, NY, USA.
- Eddon, G. (1999). COM+: the evolution of component services. *Computer* , 104-106.
- Global Architect. (2007.). Retrieved July 09, 2007, from <http://www.ciol.com/content/flavour/middleware/101032701.asp>
- Gray, J. (2004). The Revolution in Database Architecture. Extended abstract of talk at ACM SIGMOD 2004, Paris, France, June 2004 (p. 4). Association for Computing Machinery, Inc.
- Kingsley, A., & Kingsley-, K. (2007). *C# 2005 Programmer's Reference*. Indianapolis, IN 46256: Wiley Publishing, Inc.
- Liberty, J., & MacDonald, B. (2006). *Learnig C# 2005, Second Edition*. Sebastopol, CA: O' Reilly Media, Inc.
- Limprecht, R. (1997). Microsoft Transaction Server. Proceedings of COMPCON 97 , 14.
- Löwy, J. (2001). *COM and .NET Component Services*. USA: O'Reilly.
- Martin Pinzger, J. O. (2003). Analyzing and Understanding Architectural Characteristics of COM+ Components . 11th IEEE International Workshop on Program Comprehension (IWPC'03) , 54.
-

McKeown, M. (2003). .NET Enterprise Services and COM+ 1.5 Architecture. Enterprise Services Technical Articles .

Microsoft. (2007). .NET Framework Conceptual Overview . Retrieved August 1, 2007, from .NET Framework Developer Center : <http://msdn2.microsoft.com/en-us/library/zw4w595w.aspx>

Microsoft. (2007). Retrieved July 18, 2007, from COM: Component Object Model Technologies: <http://www.microsoft.com/com/default.msp>

MS. (2007). Retrieved July 17, 2007, from COM+ (Component Services): <http://msdn2.microsoft.com/en-us/library/ms685978.aspx>

MSDN. (2007). COM+ Object Pooling Concepts. Retrieved August 12, 2007, from COM+ (Component Services): <http://msdn2.microsoft.com/en-us/library/ms682784.aspx>

MSDN2 (2007). Visual C# Development Environment . Retrieved August 31, 2007, from Visual C# Developer Center : [http://msdn2.microsoft.com/en-us/library/ms184658\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms184658(vs.80).aspx)

Platt, D. S. (2000). COM+ and Windows 2000: Ten Tips and Tricks for Maximizing COM+ Performance. MSDN Magazine The Microsoft Journal for Developers .

Roman, E. (2007). Middleware. Retrieved Aug 11, 2007, from The Middleware company: <http://www.theserverside.com/tt/articles/article.tss?l=EJB-ComPlus>

Technet, M. (2007). Quick Tour of MS Transaction Server. Retrieved August 10, 2007, from Microsoft TechNet: <http://www.microsoft.com/technet/archive/transsrv/quicktr.msp?mfr=true>

Templeman, J., & Mueller, J. P. (2003). COM Programming with Microsoft® .NET. USA: Microsoft Press.

TPC. (2007). Retrieved September 20, 2007, from TPC: About the TPC: <http://www.tpc.org/information/about/abouttpc.asp>

Troelsen, A. (2007). Pro C# with .NET 3.0. USA: Apress.

Turne, R., Burek, L., & Driver, D. (2004). COM+ Technical Articles. Retrieved August 2, 2007, from .NET Enterprise Services Performance: <http://msdn2.microsoft.com/en-us/library/ms809840.aspx>

Vieira, R. (2007). Professional SQL Server™ 2005 Programming. Indianapolis, Indiana: Wiley Publishing, Inc.

Visual C# Developer Center . (2007). Retrieved August 7, 2007, from Introduction to the C# Language and the .NET Framework : [http://msdn2.microsoft.com/en-us/library/z1zx9t92\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/z1zx9t92(VS.80).aspx)

Whalen, E., Gracia, M., Patel, B., Misner, S., & Isakov, V. (2007). Microsoft SQL Server 2005 Administrator's Companion. Redmond, USA: Microsoft Press.

Yannakakis, M. (1995). Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95). (p. 224). IEEE Computer Society, DC, USA.

Appendix

APPENDIX SECTION FOLLOWS:

Appendix 1

Appendix 2

Appendix 1

A. PrototypeApplication.cs

```
// *****
// Name:                PrototypeApplication.cs
// Author:              Ashish Tandon
// Version:             Version 1.0.1.3
// Updated On:         08-Oct-07
// Created On:         15-May-07
/* Description: This class file performs the client application
  functionality consist of interface request and process.
  This also calls the other class library as per the
  information provided by the user on the interface
*/
// *****

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Windows.Forms;
using System.Data.Common;
using System.Configuration;
using System.Diagnostics;
using System.Collections;

// Namespace for COMSQLServer
using ObjectPoolServer;

// Namespace for NoCOMSQLServer
using ObjectPoolLibrary;

// Namespace for NoAccessCOM
using NoCOMAccess;

// Namespace for AccessCOM
using COMAccess;

namespace DBFactory
{
    public partial class factoryClassesForm : Form
    {
        //sDatabase contains the name of the database
        string sDatabase;

        //sType contains the application type i.e
        //1. COM+ No Pooling n JIT
        //2. COM+ Pooling n JIT
        //3. .NET Pooling
        string sType;

        //myQuery contains the query as per the data volume
        string myQuery;
    }
}
```

```
//sVolume contains low, medium and high value of data volume
string sVolume;

//iUser, Number of users selected to benchmark the database
int iUser;

//sArr, Array records the last 5 transaction time
string[] sArr = new string[5];
//Used for the transaction count
int Rcounter = 0;

public factoryClassesForm()
{
    InitializeComponent();
    providerComboBox.SelectedIndex = 0;
}

//Interface Record Button Event
private void cmdRecord_Click(object sender, EventArgs e)
{
    RecordLastResult();
}

//Interface ToMatrix Button Event
private void cmdToMatrix_Click(object sender, EventArgs e)
{
    SendToMatrix();
}

//Interface Automate Button Event
private void cmdAutomate_Click(object sender, EventArgs e)
{
    GetUserInput();

    //Automation Test Sequence for SQL SERVER Database
    if (providerComboBox.SelectedItem.ToString() == "SQL
Server")
    {
        //If application type is COM+ no Pooling n JIT
        if (sType == "NOCOM")
        {
            for (int i = 0; i < iUser; i++)
            {
                NoCOMSQL();
                RecordLastResult();
            }
            SendToMatrix();
        }
        //If application type is COM+ with Pooling n JIT
        else if (sType == "COM")
        {
            for (int i = 0; i < iUser; i++)
            {
                COMSQL();
                RecordLastResult();
            }
            SendToMatrix();
        }
    }
}
```

```

        //If application type is .NET Pooling
        else
        {
            for (int i = 0; i < iUser; i++)
            {
                NETPooling();
                RecordLastResult();
            }
            SendToMatrix();
        }
    }
}

//Interface GetData Button Event
private void getDataButton_Click(object sender, EventArgs e)
{
    try
    {
        GetUserInput();
        if (sDatabase == "MS Access")
        {
            if (rbCOMLibrary.Checked == true)
                //COM+ application no object pooling n JIT
                NonCOMAccess();
            else if (rbCOMServer.Checked == true)
                //COM+ application object pooling n JIT
                COMAccess();
        }
        else
        {
            if (rbCOMLibrary.Checked == true)
                //COM+ application no object pooling n JIT
                NoCOMSQL();
            else if (rbCOMServer.Checked == true)
                //COM+ application object pooling n JIT
                COMSQL();
            else if (rbNETPooling.Checked == true)
                //COM+ application no object pooling n JIT
                NETPooling();
        }
    }
    catch(Exception e1)
    {
        MessageBox.Show("Error 102: There was an error in
            processing the request" + e1.Message, "Error
            GetData Method", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

//GetUserInput()
//This method read the users input from interface and sets
//their respective values in the variables for further
access
private void GetUserInput()
{
    try
    {
        //Set the value for Database selected

```



```
}

//NonCOMAccess()
//This method calls the library which are not using the COM+
//features for their MS Access database read operation
private void NonCOMAccess()
{
    try
    {
        //Stopwatch for time recording
        Stopwatch myWatch = new Stopwatch();
        myWatch.Start();

        for (int i = 0; i < iUser; i++)
        {
            NoCOMAccess.NoCOMAccess oANonCom = new
                NoCOMAccess.NoCOMAccess();
            oANonCom.ExecuteQuery(myQuery);
        }
        //Stop the watch
        myWatch.Stop();

        //Convert time into TimeTicks
        elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): "
            + myWatch.ElapsedTicks.
                ToString() + " ticks";

        //Convert time into Milliseconds
        millisecondsTextLabel.Text = "Elapsed Time (Ms): ";
        lblms.Text = myWatch.ElapsedMilliseconds.ToString();
    }
    catch (Exception e1)
    {
        MessageBox.Show("Error 103: There was an error in
            processing the request" + e1.Message, "Error Non
            COM Application Access", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

//COMAccess()
//This method calls the library which are using the COM+
//features for their MS Access database read operation
private void COMAccess()
{
    try
    {
        Stopwatch myWatch = new Stopwatch();
        myWatch.Start();

        for (int i = 0; i < iUser; i++)
        {
            COMAccess.COMAccess oACom = new
                COMAccess.COMAccess();
            oACom.ExecuteQuery(myQuery);
        }
        myWatch.Stop();
    }
}
```

```
        elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): "
            + myWatch.ElapsedTicks.
                ToString() + " ticks";
        millisecondsTextLabel.Text = "Elapsed Time (Ms): ";
        lblms.Text = myWatch.ElapsedMilliseconds.ToString();
    }
    catch (Exception e)
    {
        MessageBox.Show("Error 104: There was an error in
            processing the request" + e.Message, "Error COM
            Application Access", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

//NonCOMSQL()
//This method calls the library which are not using the COM+
//features for their MS SQL Server database read operation
private void NoCOMSQL()
{
    try
    {
        Stopwatch myWatch = new Stopwatch();
        myWatch.Start();

        for (int i = 0; i < iUser; i++)
        {
            ObjectPoolLibrary.PooledObject po = new
                ObjectPoolLibrary.PooledObject();
            po.ExecuteLibQuery(myQuery);
        }
        myWatch.Stop();
        elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): "
            + myWatch.ElapsedTicks.
                ToString() + " ticks";
        millisecondsTextLabel.Text = "Elapsed Time (Ms): ";
        lblms.Text = myWatch.ElapsedMilliseconds.ToString();
    }
    catch (Exception e)
    {
        MessageBox.Show("Error 105: There was an error in
            processing the request" + e.Message, "Error Non
            COM Application SQL", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

//NonCOMSQL()
//This method calls the library which are using the COM+
//features for their MS SQL Server database read operation
private void COMSQL()
{
    try
    {
        Stopwatch myWatch = new Stopwatch();
        myWatch.Start();
        for (int i = 0; i < iUser; i++)
        {
```

```

        ObjectPoolServer.PooledObject pos = new
            ObjectPoolServer.PooledObject();
        pos.ExecuteServerQuery(myQuery);
    }
    myWatch.Stop();
    elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): "
        + myWatch.ElapsedTicks.
        ToString() + " ticks";
    millisecondsTextLabel.Text = "Elapsed Time (Ms): ";
    lblms.Text = myWatch.ElapsedMilliseconds.ToString();
}
catch (Exception e)
{
    MessageBox.Show("Error 106: There was an error in
        processing the request" + e.Message, "Error COM
        Application SQL", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
}

```

```

//NETPooling()
//This method calls the framework class library function
//and features for their database read operation
private void NETPooling()
{
    try
    {
        string myName = getConnectionString();
        DataSet myDataSet = new DataSet();

        //Reading configuration data from app.config
        ConnectionStringSettings myConnectionSettings =
            ConfigurationManager.
            ConnectionStrings[myName];
        DbProviderFactory myProvider =
            DbProviderFactories.GetFactory
            (myConnectionSettings.ProviderName);

        Stopwatch myWatch = new Stopwatch();
        myWatch.Start();
        for (int i = 0; i < iUser; i++)
        {
            DbConnection myConnection =
                myProvider.CreateConnection();
            myConnection.ConnectionString =
                myConnectionSettings.ConnectionString;

            myConnection.Open();

            DbDataAdapter myAdapter =
                myProvider.CreateDataAdapter();
            DbCommand myCommand =
                myProvider.CreateCommand();

            myCommand.Connection = myConnection;
            myCommand.CommandText = myQuery;

            myAdapter.SelectCommand = myCommand;
            myAdapter.Fill(myDataSet);

```



```

    }
    myWatch.Stop();
    elapsedTimeTextLabel.Text = "Elapsed Time (Ticks): "
        + myWatch.ElapsedTicks.
        ToString() + " ticks";
    millisecondsTextLabel.Text = "Elapsed Time (Ms):";
    lblms.Text = myWatch.ElapsedMilliseconds.ToString();
    if (cbShowDBGrid.Checked == true)
        displayDataGridView.DataSource =
myDataSet.Tables[0];
}

catch (Exception e)
{
    MessageBox.Show("Error 107: There was an error in
        processing the request" + e.Message, "Error
        Application .NET ", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
}

//getConnectionString()
//This method provide the information about which
//database is selected for database read operation
private string getConnectionString()
{
    string sCons = providerComboBox.SelectedItem.ToString();
    return sCons;
}

//RecordLastResult()
//This method records the value of the last 5 database read
operation
private void RecordLastResult()
{
    try
    {
        if (Rcounter == 5)
            cmdRecord.Enabled = false;
        if (Rcounter < 5)
        {
            cmdRecord.Enabled = true;
            sArr[Rcounter] = lblms.Text;
            lblLast5.Text += " -|- " +
            sArr[Rcounter].ToString() + " -|- ";
            Rcounter += 1;
        }
    }
    else
    {
        Rcounter = 0;
        MessageBox.Show("Kindly convert the Displayed
            Last 5 Results to the relevant
            Matrix by Clicking on the - To Matrix - button",
            "Click on To Matrix Button", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}
catch (Exception e)
{

```

```

        MessageBox.Show("Error 108: There was an error in
        processing the request" + e.Message, "Recording
        Last Transaction Time", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    }
}

//SendToMatrix()
//This method sends the recorded values to their
//respective matrix table as per the choice made by user
private void SendToMatrix()
{
    cmdRecord.Enabled = true;
    Rcounter = 0;

    try
    {
        if (providerComboBox.SelectedItem.ToString() == "SQL
            Server")
        {
            switch (sType)
            {
                case "NoCOM":
                {
                    if (sVolume == "low")
                    {
                        snch1.Text = sArr[0]; snch2.Text
                        = sArr[1]; snch3.Text = sArr[2];
                        snch4.Text = sArr[3]; snch5.Text
                        = sArr[4]; sdlr1.Text =
                        SDInitiate().ToString();
                    }
                    else if (sVolume == "medium")
                    {
                        snct1.Text = sArr[0]; snct2.Text
                        = sArr[1]; snct3.Text = sArr[2];
                        snct4.Text = sArr[3]; snct5.Text
                        = sArr[4]; sdlr2.Text =
                        SDInitiate().ToString();
                    }
                    else if (sVolume == "high")
                    {
                        snctt1.Text = sArr[0];
                        snctt2.Text = sArr[1];
                        snctt3.Text = sArr[2];
                        snctt4.Text = sArr[3];
                        snctt5.Text = sArr[4];
                        sdlr3.Text =
                        SDInitiate().ToString() + "
                        Avg: " +
                        InitiateAverage().ToString();
                    }
                }
            }
            break;
        case "COM":
        {
            //Table 2
            if (sVolume == "low")
            {

```

```

        sch1.Text = sArr[0]; sch2.Text =
        sArr[1]; sch3.Text = sArr[2];
        sch4.Text = sArr[3]; sch5.
        Text = sArr[4];sd2r1.Text =
        SDInitiate().ToString();
    }
    else if (sVolume == "medium")
    {
        sct1.Text = sArr[0]; sct2.Text =
        sArr[1]; sct3.Text = sArr[2];
        sct4.Text = sArr[3]; sct5.
        Text = sArr[4];sd2r2.Text =
        SDInitiate().ToString();
    }
    else if (sVolume == "high")
    {
        sc tt1.Text = sArr[0]; sc tt2.Text
        = sArr[1]; sc tt3.Text = sArr[2];
        sc tt4.Text = sArr[3];
        sc tt5.Text = sArr[4]; sd2r3.Text
        = SDInitiate().ToString();
    }
}
break;
case ".NET":
{
    //Table 3
    if (sVolume == "low")
    {
        sh1.Text = sArr[0]; sh2.Text =
        sArr[1]; sh3.Text = sArr[2];
        sh4.Text = sArr[3]; sh5.Text =
        sArr[4];
        sd3r1.Text =
        SDInitiate().ToString();
    }

    else if (sVolume == "medium")
    {
        st1.Text = sArr[0]; st2.Text =
        sArr[1]; st3.Text = sArr[2];
        st4.Text = sArr[3]; st5.Text =
        sArr[4];
        sd3r2.Text =
        SDInitiate().ToString();
    }

    else if (sVolume == "high")
    {
        stt1.Text = sArr[0]; stt2.Text =
        sArr[1]; stt3.Text = sArr[2];
        stt4.Text = sArr[3]; stt5.Text =
        sArr[4];
        sd3r3.Text =
        SDInitiate().ToString();
    }
}
break;
}
}
}

```

```
else
{
    //
    if (sType == "NoCOM" && sVolume == "low")
    {
        anch1.Text = sArr[0]; anch2.Text = sArr[1];
        anch3.Text = sArr[2]; anch4.Text = sArr[3];
        anch5.Text = sArr[4];
        sd4r1.Text = SDInitiate().ToString();
    }

    if (sType == "NoCOM" && sVolume == "medium")
    {
        anct1.Text = sArr[0]; anct2.Text = sArr[1];
        anct3.Text = sArr[2]; anct4.Text = sArr[3];
        anct5.Text = sArr[4];
        sd4r2.Text = SDInitiate().ToString();
    }

    if (sType == "NoCOM" && sVolume == "high")
    {
        anctt1.Text = sArr[0]; anctt2.Text =
        sArr[1]; anctt3.Text = sArr[2]; anctt4.Text
        = sArr[3]; anctt5.Text = sArr[4];
        sd4r3.Text = SDInitiate().ToString();
    }

    if (sType == "COM" && sVolume == "low")
    {
        ach1.Text = sArr[0]; ach2.Text = sArr[1];
        ach3.Text = sArr[2]; ach4.Text = sArr[3];
        ach5.Text = sArr[4];sd5r1.Text =
        SDInitiate().ToString();
    }

    if (sType == "COM" && sVolume == "medium")
    {
        act1.Text = sArr[0]; act2.Text = sArr[1];
        act3.Text = sArr[2]; act4.Text = sArr[3];
        act5.Text = sArr[4];sd5r2.Text =
        SDInitiate().ToString();
    }

    if (sType == "COM" && sVolume == "high")
    {
        actt1.Text = sArr[0]; actt2.Text = sArr[1];
        actt3.Text = sArr[2]; actt4.Text = sArr[3];
        actt5.Text = sArr[4];sd5r3.Text =
        SDInitiate().ToString();
    }

    //Table 3
    if (sType == ".NET" && sVolume == "low")
    {
        ah1.Text = sArr[0]; ah2.Text = sArr[1];
        ah3.Text = sArr[2]; ah4.Text = sArr[3];
        ah5.Text = sArr[4];sd6r1.Text =
        SDInitiate().ToString();
    }
}
```

```

        if (sType == ".NET" && sVolume == "medium")
        {
            at1.Text = sArr[0]; at2.Text = sArr[1];
            at3.Text = sArr[2]; at4.Text = sArr[3];
            at5.Text = sArr[4];sd6r2.Text =
            SDInitiate().ToString();
        }

        if (sType == ".NET" && sVolume == "high")
        {
            att1.Text = sArr[0]; att2.Text = sArr[1];
            att3.Text = sArr[2];att4.Text = sArr[3];
            att5.Text = sArr[4];sd6r3.Text
            =SDInitiate().ToString();
        }

        }//else
    }
    catch (Exception e)
    {
        MessageBox.Show("Error 109: There was an error in
        processing the request" + e.Message, "Error Send Data To
        Matrix", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    }
    lbllast5.Text = "";
}

//GetVariance()
//This method accepts the input as array of
//double type and calculate the variance
public static double GetVariance(double[] data)
{
    int len = data.Length;
    double avg = Average(data);
    double sum = 0;
    for (int i = 0; i < data.Length; i++)
        sum += Math.Pow((data[i] - avg), 2);
    return sum / len;
}

//GetVariance()
//This method accepts the input as array of
//double type and calculate the standard deviation
public static double GetStdev(double[] data)
{
    return Math.Sqrt(GetVariance(data));
}

//Average()
//This method accepts the input as array of
//double type and calculate the average
private static double Average(double[] data)
{
    double DataTotal = 0;
    try

```

```
        {
            for (int i = 0; i < data.Length; i++)
            {
                DataTotal += data[i];
            }
            //return SafeDivide(DataTotal, data.Length);
        }

        catch (Exception e)
        {
            MessageBox.Show("Error 111: There was an error
in processing the request" + e.Message, "Error
Calculating Average", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
        return SafeDivide(DataTotal, data.Length);
    }

    //SafeDivide()
    //This method accepts the two input parameter of
    //double values and performs the divide operation
    private static double SafeDivide(double value1, double
value2)
    {
        double ret = 0;
        try
        {
            if ((value1 == 0) || (value2 == 0)) { return ret; }
            ret = value1 / value2;
        }
        catch (Exception e)
        {
            MessageBox.Show("Error 112: There was an error in
processing the request" + e.Message, "Error Safe
Divide", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
        return ret;
    }

    //SDInitiate()
    //This method converts the string value type to
    //double type
    private double SDInitiate()
    {
        double sd=0;
        double[] dConvert= new double[5];

        try
        {
            dConvert[0] = double.Parse(sArr[0]);
            dConvert[1] = double.Parse(sArr[1]);
            dConvert[2] = double.Parse(sArr[2]);
            dConvert[3] = double.Parse(sArr[3]);
            dConvert[4] = double.Parse(sArr[4]);
            sd = GetStdev(dConvert);
            //return Math.Round(sd, 2);
        }
        catch (Exception e)
        {
```

```
        MessageBox.Show("Error 113: There was an error in
processing the request" + e.Message, "Error Standard
Deviation", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    return Math.Round(sd, 2);
}

//InitiateAverage()
//This method converts the string value type to
//double type and return the average
private double InitiateAverage()
{
    double dAvg=0;
    double[] dAvgArr = new double[5];
    try
    {
        dAvgArr[0] = double.Parse(sArr[0]);
        dAvgArr[1] = double.Parse(sArr[1]);
        dAvgArr[2] = double.Parse(sArr[2]);
        dAvgArr[3] = double.Parse(sArr[3]);
        dAvgArr[4] = double.Parse(sArr[4]);

        dAvg = Average(dAvgArr);
        //return dAvg;
    }

    catch (Exception e)
    {
        MessageBox.Show("Error 114: There was an error in
processing the request" + e.Message, "Error
Calculate Average", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    return dAvg;
}
}
}
```

B. Assembly info [PrototypeApplication.cs]

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
// following
// set of attributes. Change these attribute values to modify the
// information
// associated with an assembly.
[assembly: AssemblyTitle("DBFactory")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("DBFactory")]
[assembly: AssemblyCopyright("Copyright © 2005")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
// visible
// to COM componenets. If you need to access a type in this
// assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project
// is exposed to COM
[assembly: Guid("8d67ec3f-efc6-4906-85c3-5829547ef8ab")]
[assembly: AssemblyVersion("1.1.0.0")]
[assembly: AssemblyFileVersion("1.1.0.0")]
```

C. COMAccess.cs

```
// *****
// Name:                COMAccess.cs
// Author:              Ashish Tandon
// Version:             Version 1.0.1.3
// Updated On:          08-Oct-07
// Created On:          03-May-07
/* Description: This class file performs the COM+ applicatoin
  functionality with object pooling and JIT for the
  MS Access database.*/
// *****

namespace COMAccess
{
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.OleDb;
    using System.Data.SqlClient;
    using System.Reflection;

    //Enabling COM+ Object Pooling Feature
    [System.EnterpriseServices.ObjectPooling
     (true, 10, 100, CreationTimeout = 5000)
    ]

    //Enabling COM+ JIT Feature
    [System.EnterpriseServices.JustInTimeActivation(true)]

    //Enabling COM+ Transaction option
    [System.EnterpriseServices.Transaction
     (TransactionOption.NotSupported)
    ]

    public class COMAccess : ServicedComponent
    {
        private System.Data.OleDb.OleDbConnection _cnn;
        private System.Data.OleDb.OleDbCommand _cmd;

        public COMAccess()
        {
            _cnn = new
            OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
            Source=C:\\Temp\\Test\\v1.0\\TestEnvironment\\DBFactory\\Datab
            ases\\MyData.mdb;Persist Security Info=True");
            _cmd = new OleDbCommand();
            _cmd.CommandType = System.Data.CommandType.Text;
            _cmd.Connection = _cnn;
            _cnn.Open();
        }
        [AutoComplete]
    }
}
```

```
public void ExecuteQuery(string sQuery)
{
    _cmd.CommandText = sQuery;
    _cmd.ExecuteNonQuery();
}
}
```

D. Assembly info [COMAccess.cs]

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the
// following
// set of attributes. Change these attribute values to modify the
// information
// associated with an assembly.
[assembly: AssemblyTitle("COMAccess_Server")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Napier University")]
[assembly: AssemblyProduct("COMAccess_Server")]
[assembly: AssemblyCopyright("Copyright © Napier University 2007")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not
// visible to COM components. If you need to access a type in this
// assembly from COM, set the ComVisible attribute to true on that
// type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project
// is exposed to COM
[assembly: Guid("cdb58b3d-5c3b-409a-a12f-86029595e1b8")]

[assembly: AssemblyVersion("1.1.1.0")]
[assembly: AssemblyFileVersion("1.1.1.0")]
```

E. COMSQL.cs

```
// *****
// Name: COMSQL.cs
// Author: Ashish Tandon
// Version: Version 1.0.1.3
// Updated On: 08-Oct-07
// Created On: 03-May-07
/* Description: This class file performs the COM+ applicatoin
functionality with object pooling and JIT for the
MS SQL SERVER Database.*/
// *****
namespace ObjectPoolServer
{
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.SqlClient;
    //Enabling COM+ Object Pooling Feature
    [System.EnterpriseServices.ObjectPooling
        (true, 10, 100, CreationTimeout = 5000)
    ]

    //Enabling COM+ JIT Feature
    [System.EnterpriseServices.JustInTimeActivation(true)]
    //Enabling COM+ Transaction Option
    [System.EnterpriseServices.Transaction
        (TransactionOption.NotSupported)
    ]
    public class PooledObject : ServicedComponent
    {
        private System.Data.SqlClient.SqlConnection _cnn;
        private System.Data.SqlClient.SqlCommand _cmd;

        public PooledObject()
        {
            _cnn = new SqlConnection("Integrated
Security=SSPI;Persist Security Info=False;Initial
Catalog=OfficeMart;Data Source=.\SQLEXPRESS;");
            _cmd = new SqlCommand();
            _cmd.CommandType = System.Data.CommandType.Text;
            _cmd.Connection = _cnn;
            _cnn.Open();
        }
        [AutoComplete]
        public void ExecuteServerQuery(string sSQLQuery)
        {
            _cmd.CommandText = sSQLQuery;
            _cmd.ExecuteNonQuery();
        }
    }
}
```

F. Assembly info[COMSQL.cs]

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.EnterpriseServices;
using System.Runtime.InteropServices;

//
// General Information about an assembly is controlled through the
// following
// set of attributes. Change these attribute values to modify the
// information
// associated with an assembly.
//
[assembly: AssemblyTitle( "" )]
[assembly: AssemblyDescription( "" )]
[assembly: AssemblyConfiguration( "" )]
[assembly: AssemblyCompany( "" )]
[assembly: AssemblyProduct( "" )]
[assembly: AssemblyCopyright( "" )]
[assembly: AssemblyTrademark( "" )]
[assembly: AssemblyCulture( "" )]
[assembly: AssemblyVersion( "1.0.*" )]
[assembly: AssemblyDelaySign( false )]
[assembly: AssemblyKeyName( "" )]
[assembly: ComVisibleAttribute( true )]
```

G. NoCOMAccess.cs

```
// *****
// Name:                NoCOMAccess.cs
// Author:              Ashish Tandon
// Version:             Version 1.0.1.3
// Updated On:          08-Oct-07
// Created On:          03-May-07
/* Description: This class file performs the COM+ applicatoin
functionality without object pooling and JIT for the
MS Access database.*/
// *****

namespace NoCOMAccess
{
    //Pooling without object pooling and JIT
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.OleDb;
    using System.Data.SqlClient;
    using System.Reflection;

    [System.EnterpriseServices.ObjectPooling(false)]
    [System.EnterpriseServices.JustInTimeActivation(false)]
    [System.EnterpriseServices.Transaction
    (TransactionOption.NotSupported)]
    ]
    public class NoCOMAccess : ServicedComponent
    {
        private System.Data.OleDb.OleDbConnection _cnn;
        private System.Data.OleDb.OleDbCommand _cmd;

        public NoCOMAccess()
        {
            _cnn = new
            OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
            Source=C:\\Temp\\Test\\v1.0\\TestEnvironment\\DBFactory\\Datab
            ases\\MyData.mdb;Persist Security Info=True");
            _cmd = new OleDbCommand();
            _cmd.CommandType = System.Data.CommandType.Text;
            _cmd.Connection = _cnn;
            _cnn.Open();
        }
        [AutoComplete]
        public void ExecuteQuery(string sQuery)
        {
            _cmd.CommandText = sQuery;
            _cmd.ExecuteNonQuery();
        }
    }
}
}
```

H. Assembly info [NoCOMAccess.cs]

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.EnterpriseServices;
using System.Runtime.InteropServices;

//
// General Information about an assembly is controlled through the
// following
// set of attributes. Change these attribute values to modify the
// information
// associated with an assembly.
//
[assembly: AssemblyTitle( "" )]
[assembly: AssemblyDescription( "" )]
[assembly: AssemblyConfiguration( "" )]
[assembly: AssemblyCompany( "" )]
[assembly: AssemblyProduct( "" )]
[assembly: AssemblyCopyright( "" )]
[assembly: AssemblyTrademark( "" )]
[assembly: AssemblyCulture( "" )]
[assembly: ApplicationActivation(ActivationOption.Library)]
//

[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyDelaySign(false)]
[assembly: AssemblyKeyName( "" )]
[assembly: ComVisibleAttribute(true)]
```

I. NoCOMSQL.cs

```
// *****
// Name:           NoCOMSQL.cs
// Author:          Ashish Tandon
// Version:         Version 1.0.1.3
// Updated On:      08-Oct-07
// Created On:      03-May-07
/* Description: This class file performs the COM+ applicatoin
  functionality without object pooling and JIT for the
  MS SQL SERVER database.*/
// *****

namespace ObjectPoolLibrary
{
    //Pooling without object pooling and JIT
    using System;
    using System.Xml;
    using System.EnterpriseServices;
    using System.Data;
    using System.Data.OleDb;
    using System.Data.SqlClient;
    using System.Reflection;

    [System.EnterpriseServices.ObjectPooling(false)]
    [System.EnterpriseServices.JustInTimeActivation(false)]
    [System.EnterpriseServices.Transaction
    (TransactionOption.NotSupported)]
    ]

    public class PooledObject : ServicedComponent
    {
        private System.Data.SqlClient.SqlConnection _cnn;
        private System.Data.SqlClient.SqlCommand _cmd;
        public PooledObject()
        {
            _cnn = new SqlConnection("Integrated
            Security=SSPI;Persist Security Info=False;Initial
            Catalog=OfficeMart;Data Source=.\SQLEXPRESS;");
            _cmd = new SqlCommand();
            _cmd.CommandType = System.Data.CommandType.Text;
            _cmd.Connection = _cnn;
            _cnn.Open();
        }
        [AutoComplete]
        public void ExecuteLibQuery(string sQuery)
        {
            _cmd.CommandText = sQuery;
            _cmd.ExecuteNonQuery();
        }
    }
}
}
```

J. Assembly info [NoCOMSQL.css]

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.EnterpriseServices;
using System.Runtime.InteropServices;

//
// General Information about an assembly is controlled through the
// following
// set of attributes. Change these attribute values to modify the
// information
// associated with an assembly.
//
[assembly: AssemblyTitle("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: ApplicationActivation(ActivationOption.Library)]
//

[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyDelaySign(false)]
[assembly: AssemblyKeyName("")]
[assembly: ComVisibleAttribute(true)]
```

K. AppConfig

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is where we store all the connection string information
      If we want to connect to a different data source, we can
reference
      this file to get the appropriate information, but do not need
to
      change any of the existing code.
-->
<configuration>
  <connectionStrings>
    <add name="SQL Server "
providerName="System.Data.SqlClient "
    connectionString="Data Source=.\SQLEXPRESS;Initial
Catalog=OfficeMart;Integrated Security=True;" />
    <add name="MS Access " providerName="System.Data.OleDb"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Temp\Test\v1.0\TestEnvironment\DBFactory\Databases\MyData.
mdb;Persist Security Info=True" />
  </connectionStrings>
</configuration>
```
