1. INTRODUCTION

1.1 Introduction to Project

WHAT IS CLOUD COMPUTING

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network (typically the Internet). The name comes from the common use of a cloud-shaped symbol as an abstraction for the complex infrastructure it contains in system diagrams. Cloud computing entrusts remote services with a user's data, software and computation. Cloud computing consists of hardware and software resources made available on the Internet as managed third-party services. These services typically provide access to advanced software applications and high-end networks of server computers.

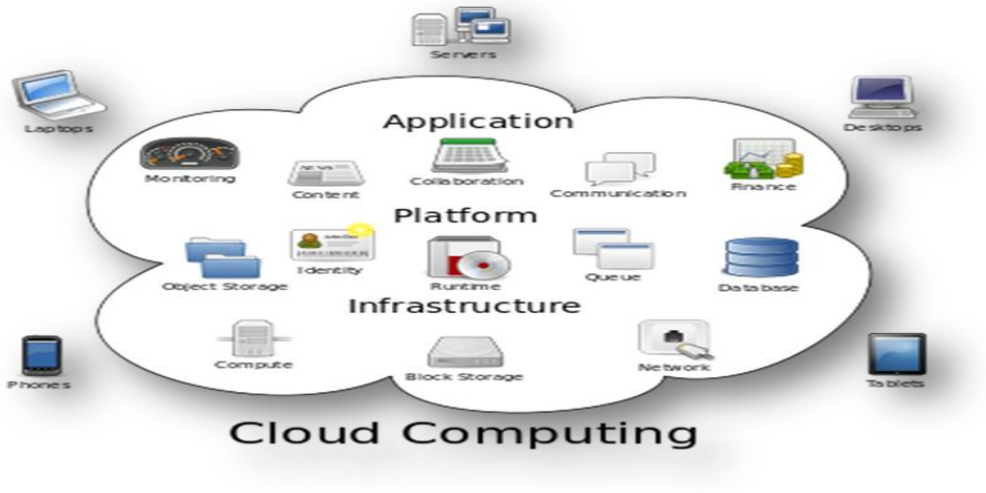


Fig.1.1 Structure of cloud computing

HOW CLOUD COMPUTING WORKS

The goal of cloud computing is to apply traditional supercomputing, or high-performance computing power, normally used by military and research facilities, to

perform tens of trillions of computations per second, in consumer-oriented applications such as financial portfolios, to deliver personalized information, to provide data storage or to power large, immersive computer games.

The cloud computing uses networks of large groups of servers typically running low-cost consumer PC technology with specialized connections to spread data-processing chores across them. This shared IT infrastructure contains large pools of systems that are linked together. Often, virtualization techniques are used to maximize the power of cloud computing.

CHARACTERISTICS AND SERVICES MODELS:

The salient characteristics of cloud computing based on the definitions provided by the National Institute of Standards and Terminology (NIST) are outlined below:

- **On-demand self-service**: A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider.
- **Broad network access**: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).
- **Resource pooling**: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location-independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or data center). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

- **Rapid elasticity**: Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.
- **Measured service**: Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).

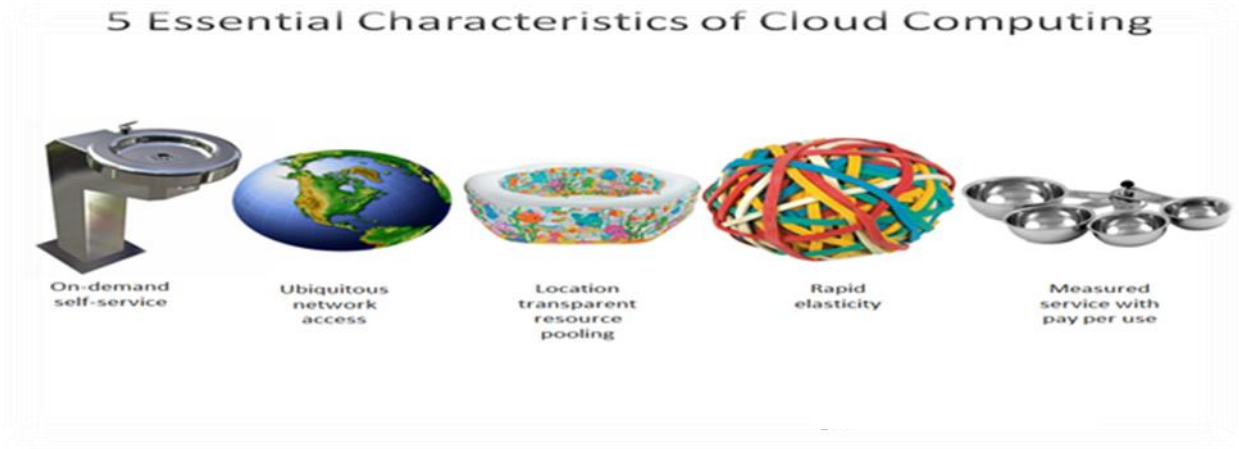


Fig.1.2 Characteristics of cloud computing

SERVICES MODELS**:**

Cloud Computing comprises three different service models, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The three service models or layer are completed by an end user layer that encapsulates the end user perspective on cloud services. The model is shown in figure below. If a cloud user accesses services on the infrastructure layer, for instance, she can run her own applications on the resources of a cloud infrastructure and remain responsible for the support, maintenance, and security of these applications herself. If she accesses a service on the application layer, these tasks are normally taken care of by the cloud service provider.

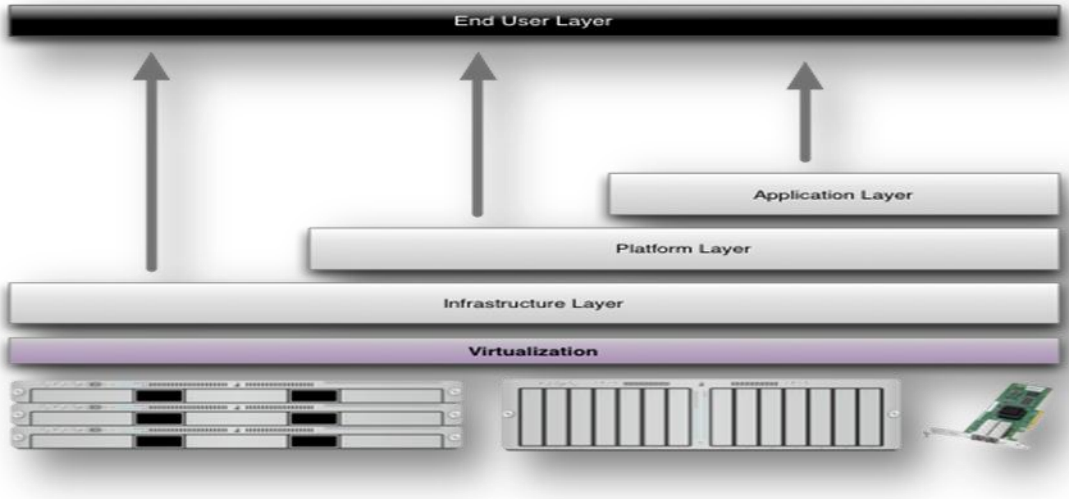


Fig.1.3 Structure of service models

BENEFITS OF CLOUD COMPUTING**:**

1. **Achieve economies of scale** – increase volume output or productivity with fewer people. Your cost per unit, project or product plummets.
2. **Reduce spending on technology infrastructure.** Maintain easy access to your information with minimal upfront spending. Pay as you go (weekly, quarterly or yearly), based on demand.
3. **Globalize your workforce on the cheap.** People worldwide can access the cloud, provided they have an Internet connection.
4. **Streamline processes.** Get more work done in less time with less people.
5. **Reduce capital costs.** There's no need to spend big money on hardware, software or licensing fees.
6. **Improve accessibility.** You have access anytime, anywhere, making your life so much easier!

7. **Monitor projects more effectively.** Stay within budget and ahead of completion cycle times.
8. **Less personnel training is needed.** It takes fewer people to do more work on a cloud, with a minimal learning curve on hardware and software issues.
9. **Minimize licensing new software.** Stretch and grow without the need to buy expensive software licenses or programs.
10. **Improve flexibility.** You can change direction without serious "people" or "financial" issues at stake.

ADVANTAGES:

1. **Price:**Pay for only the resources used.
2. **Security**: Cloud instances are isolated in the network from other instances for improved security.
3. **Performance:** Instances can be added instantly for improved performance. Clients have access to the total resources of the Cloud's core hardware.
4. **Scalability:** Auto-deploy cloud instances when needed.
5. **Uptime:** Uses multiple servers for maximum redundancies. In case of server failure, instances can be automatically created on another server.
6. **Control:** Able to login from any location. Server snapshot and a software library lets you deploy custom instances.
7. **Traffic:** Deals with spike in traffic with quick deployment of additional instances to handle the load.

2. LITERATURE SURVEY

1) "Elastras: An Elastic Transactional Data Store in the Cloud,"

AUTHORS: S. Das, D. Agrawal, and A.E. Abbadi

Over the last couple of years, "Cloud Computing" or "Elastic Computing" has emerged as a compelling and successful paradigm for internet scale computing. One of the major contributing factors to this success is the elasticity of resources. In spite of the elasticity provided by the infrastructure and the scalable design of the applications, the elephant (or the underlying database), which drives most of these web-based applications, is not very elastic and scalable, and hence limits scalability. In this paper, we propose ElasTraS which addresses this issue of scalability and elasticity of the data store in a cloud computing environment to leverage from the elastic nature of the underlying infrastructure, while providing scalable transactional data access. This paper aims at providing the design of a system in progress, highlighting the major design choices, analyzing the different guarantees provided by the system, and identifying several important challenges for the research community striving for computing in the cloud.

2. "Safety and Consistency in Policy-Based Authorization Systems,"

AUTHORS: A.J. Lee and M. Winslett

In trust negotiation and other distributed proving systems, networked entities cooperate to form proofs that are justified by collections of certified attributes. These attributes may be obtained through interactions with any number of external entities and are collected and validated over an extended period of time. Though these collections of credentials in some ways resemble partial system snapshots, these systems currently lack the notion of a consistent global state in which the satisfaction of authorization policies should be checked. In this paper, we argue that unlike the notions of consistency studied in other areas of distributed computing, the level of consistency required during policy evaluation is predicated solely upon the security requirements of the policy evaluator. As such, there is little incentive for entities to participate in complicated consistency preservation schemes like those used in distributed computing, distributed databases, and

distributed shared memory. We go on to show that the most intuitive notion of consistency fails to provide basic safety guarantees under certain circumstances and then propose several more refined notions of consistency which provide stronger safety guarantees.

3. "Automated Trust Negotiation Using Cryptographic Credentials,"

AUTHORS J. Li, N. Li, and W.H. Winsborough

In automated trust negotiation (ATN), two parties exchange digitally signed credentials that contain attribute information to establish trust and make access control decisions. Because the information in question is often sensitive, credentials are protected according to access control policies. In traditional ATN, credentials are transmitted either in their entirety or not at all. This approach can at times fail unnecessarily, either because a cyclic dependency makes neither negotiator willing to reveal her credential before her opponent, because the opponent must be authorized for all attributes packaged together in a credential to receive any of them, or because it is necessary to fully disclose exact attribute values, rather than merely proving they satisfy some predicate (such as being over 21 years of age). Recently, several cryptographic credential schemes and associated protocols have been developed to address these and other problems. However, they can be used only as fragments of an ATN process. This paper introduces a framework for ATN in which the diverse credential schemes and protocols can be combined, integrated, and used as needed. A policy language is introduced that enables negotiators to specify authorization requirements that must be met by an opponent to receive various amounts of information about certified attributes and the credentials that contain it. The language also supports the use of uncertified attributes, allowing them to be required as part of policy satisfaction, and to place their (automatic) disclosure under policy control.

4. "An Efficient System for Non- Transferable Anonymous Credentials with Optional Anonymity Revocation,"

AUTHORS; J. Camenisch and A. Lysyanskaya

A credential system is a system in which users can obtain credentials from organizations and demonstrate possession of these credentials. Such a system is anonymous

when transactions carried out by the same user cannot be linked. An anonymous credential system is of significant practical relevance because it is the best means of providing privacy for users. In this paper we propose a practical anonymous credential system that is based on the strong RSA assumption and the decisional Diffie-Hellman assumption modulo a safe prime product and is considerably superior to existing ones: (1) We give the first practical solution that allows a user to unlinkably demonstrate possession of a credential as many times as necessary without involving the issuing organization.

(2) To prevent misuse of anonymity, our scheme is the first to offer optional anonymity revocation for particular transactions. (3) Our scheme offers separability: all organizations can choose their cryptographic keys independently of each other. Moreover, we suggest more effective means of preventing users from sharing their credentials, by introducing all-or-nothing sharing: a user, who allows a friend to use one of her credentials once, gives him the ability to use all of her credentials, i.e., taking over her identity. This is implemented by a new primitive, called circular encryption, which is of independent interest, and can be realized from any semantically secure cryptosystem in the random oracle model.

5. "Recovery and Performance of Atomic Commit Processing in Distributed

Database Systems,"

AUTHORS; P.K. Chrysanthis, G. Samaras, and Y.J. Al-Houmaily**,**

A transaction is traditionally defined so as to provide the properties of atomicity, consistency, integrity, and durability (ACID) for any operation it performs. In order to ensure the atomicity of distributed transactions, an atomic commit protocol needs to be followed by all sites participating in a transaction execution to agree on the final outcome, that is, commit or abort. A variety of commit protocols have been proposed that either enhance the performance of the classical two-phase commit protocol during normal processing or reduce the cost of recovery processing after a failure. In this chapter, we survey a number of two-phase commit variants and optimizations, including some recent

ones, providing an insight in the performance trade-off between normal and recovery processing. We also analyze the performance of a representative set of commit protocols both analytically as well as empirically using simulation.

2.1 EXISTING SYSTEM:

To provide scalability and elasticity, cloud services often make heavy use of replication to ensure consistent performance and availability. As a result, many cloud services rely on the notion of eventual consistency when propagating data throughout the system. This consistency model is a variant of weak consistency that allows data to be inconsistent among some replicas during the update process, but ensures that updates will eventually be propagated to all replicas.

DISADVANTAGES OF EXISTING SYSTEM:

- Consistency problems can arise as transactional database systems are deployed in cloud environments and use policy-based authorization systems to protect sensitive resources.
- The system may suffer from policy inconsistencies during policy updates.
- It is possible for external factors to cause user credential inconsistencies over the lifetime of a transaction.

2.2 PROPOSED SYSTEM:

- We propose a new decentralized access control scheme for secure data storage in clouds that supports anonymous authentication.
- In the proposed scheme, the cloud verifies the authenticity of the series without knowing the user's identity before storing data.

- Our scheme also has the added feature of access control in which only valid users are able to decrypt the stored information.
- The scheme prevents replay attacks and supports creation, modification, and reading data stored in the cloud.

ADVANTAGES OF PROPOSED SYSTEM:

- Distributed access control of data stored in cloud so that only authorized users with valid attributes can access them.

3. SYSTEM DESIGN

3.1 SYSTEM REQUIREMENTS:

HARDWARE REQUIREMENTS:

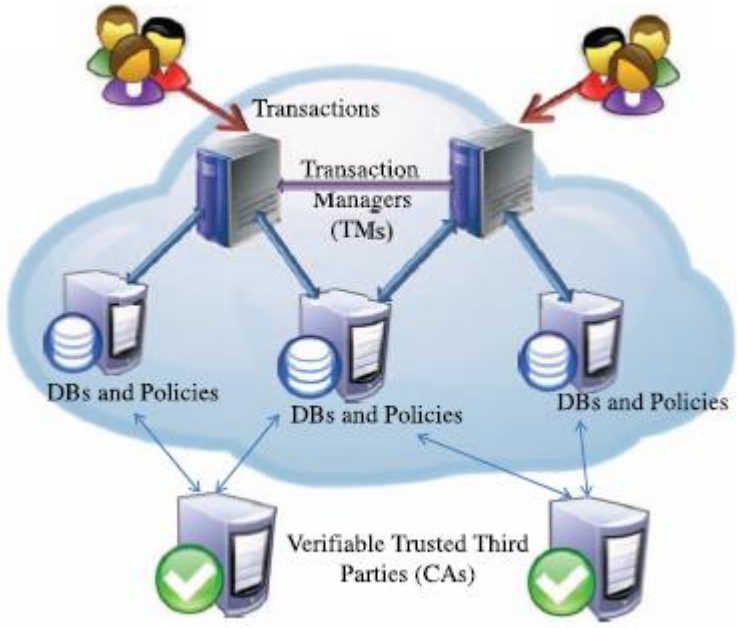
- System : Pentium IV 2.4 GHz.
- Hard Disk : 320 GB.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- Ram : 2 GB..

SOFTWARE REQUIREMENTS:

- Operating system : Windows XP/7.
- Coding Language : ASP.net, C#.net
- Tool : Visual Studio 2010

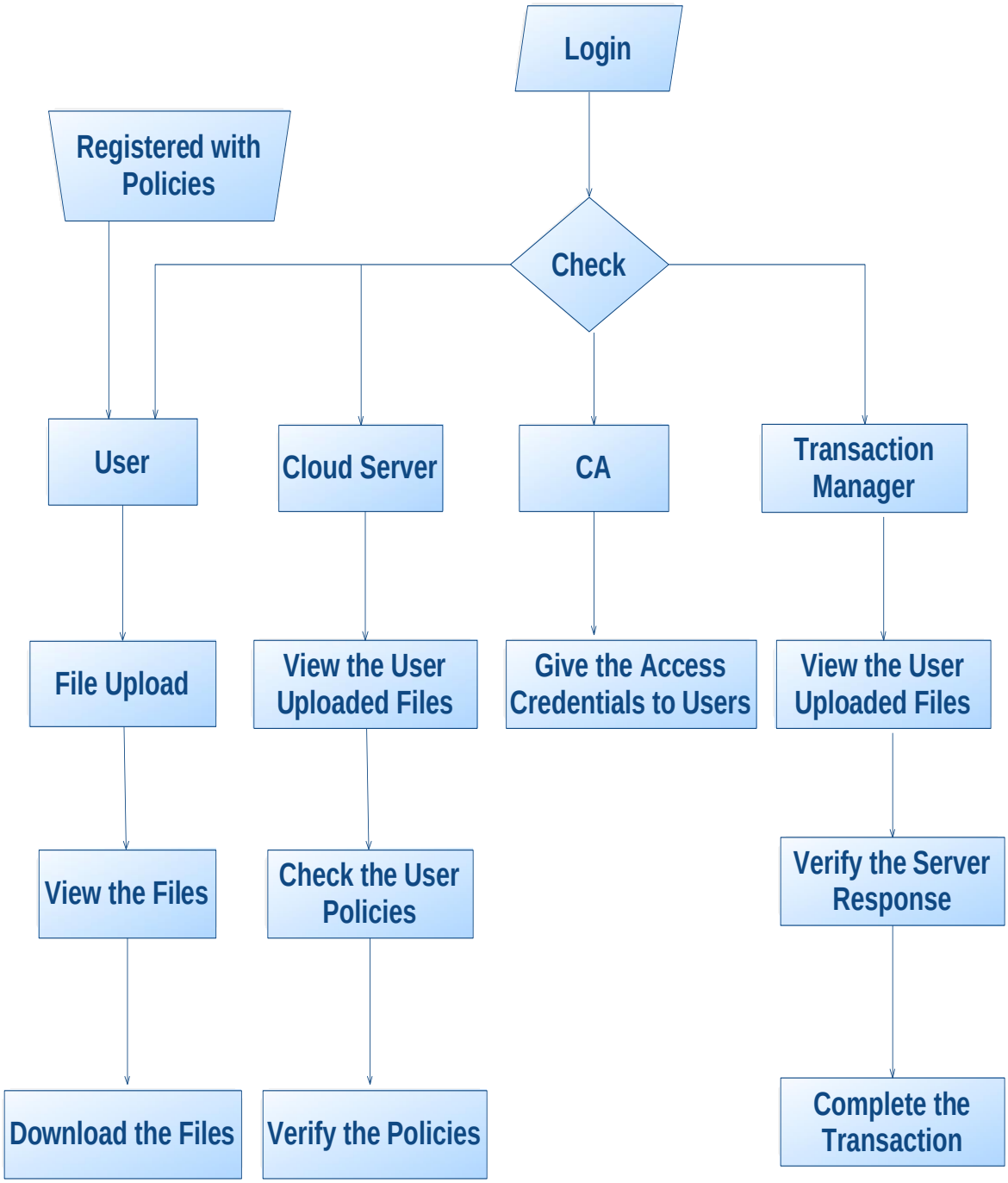
- Database : SQL SERVER 2008

SYSTEM ARCHITECTURE:



3.2 DATA FLOW DIAGRAM:

1. DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.


Login
Registered with Policies
Check
User
Cloud Server
CA
Transaction Manager
File Upload
View the User Uploaded Files
Give the Access Credentials to Users
View the User Uploaded Files
View the Files
Check the User Policies
Verify the Server Response
Download the Files
Verify the Policies
Complete the Transaction

UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

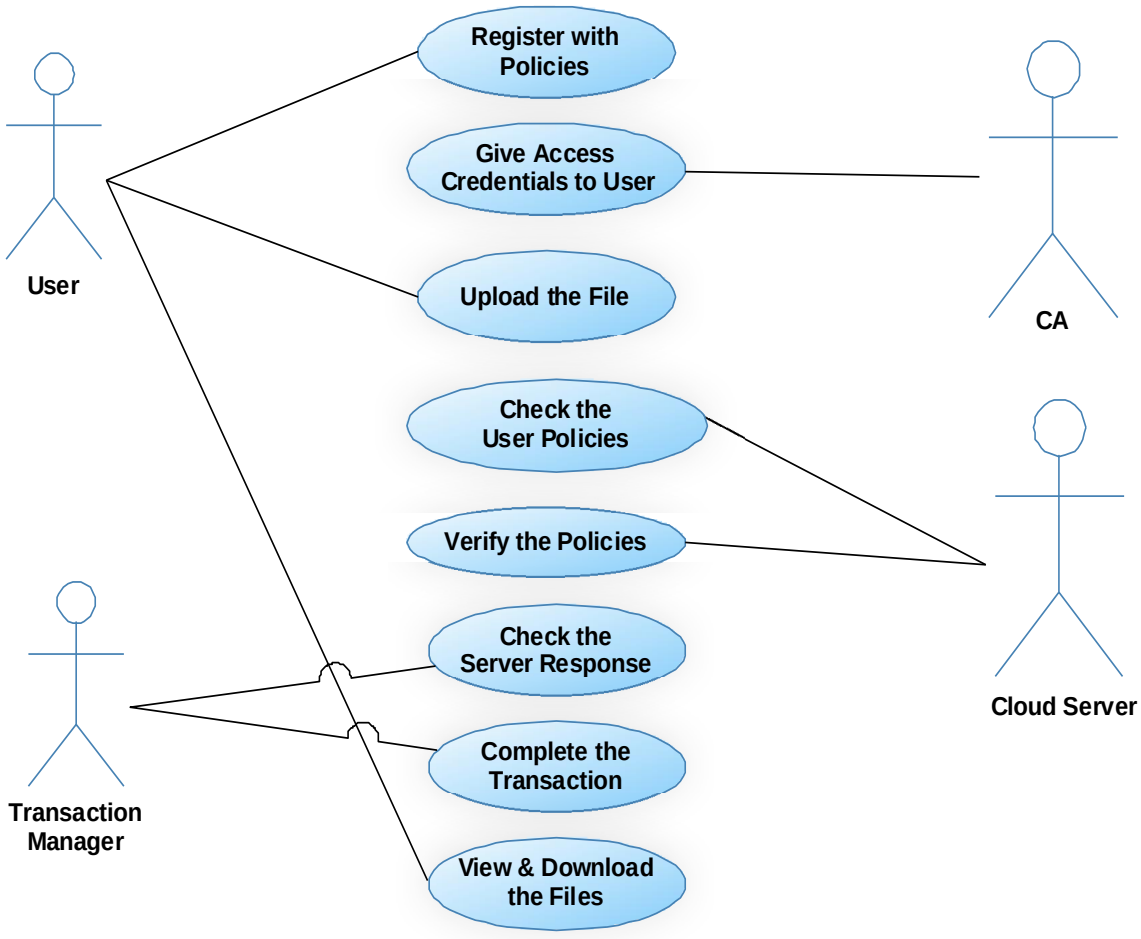
GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

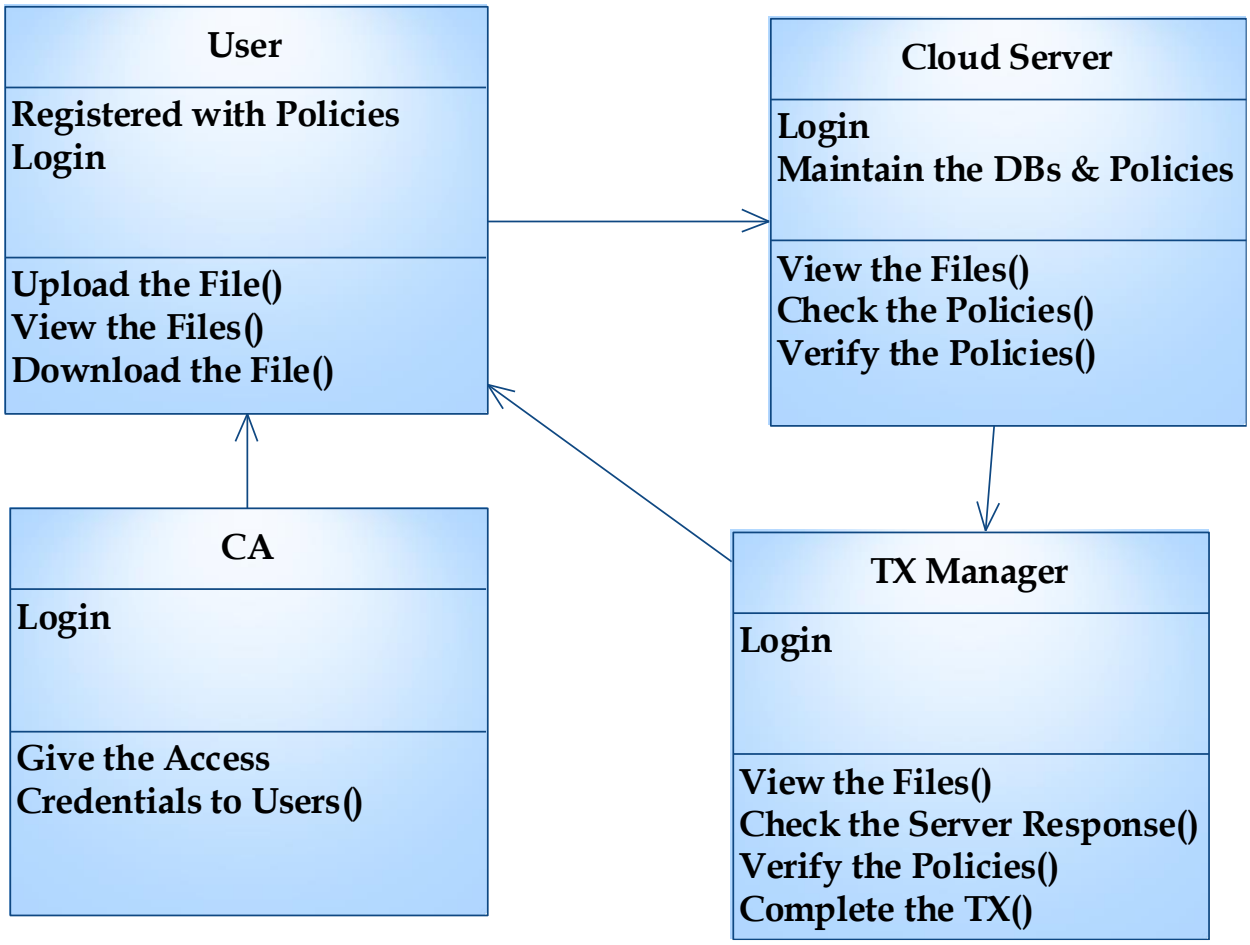
3.3 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



3.4 CLASS DIAGRAM:

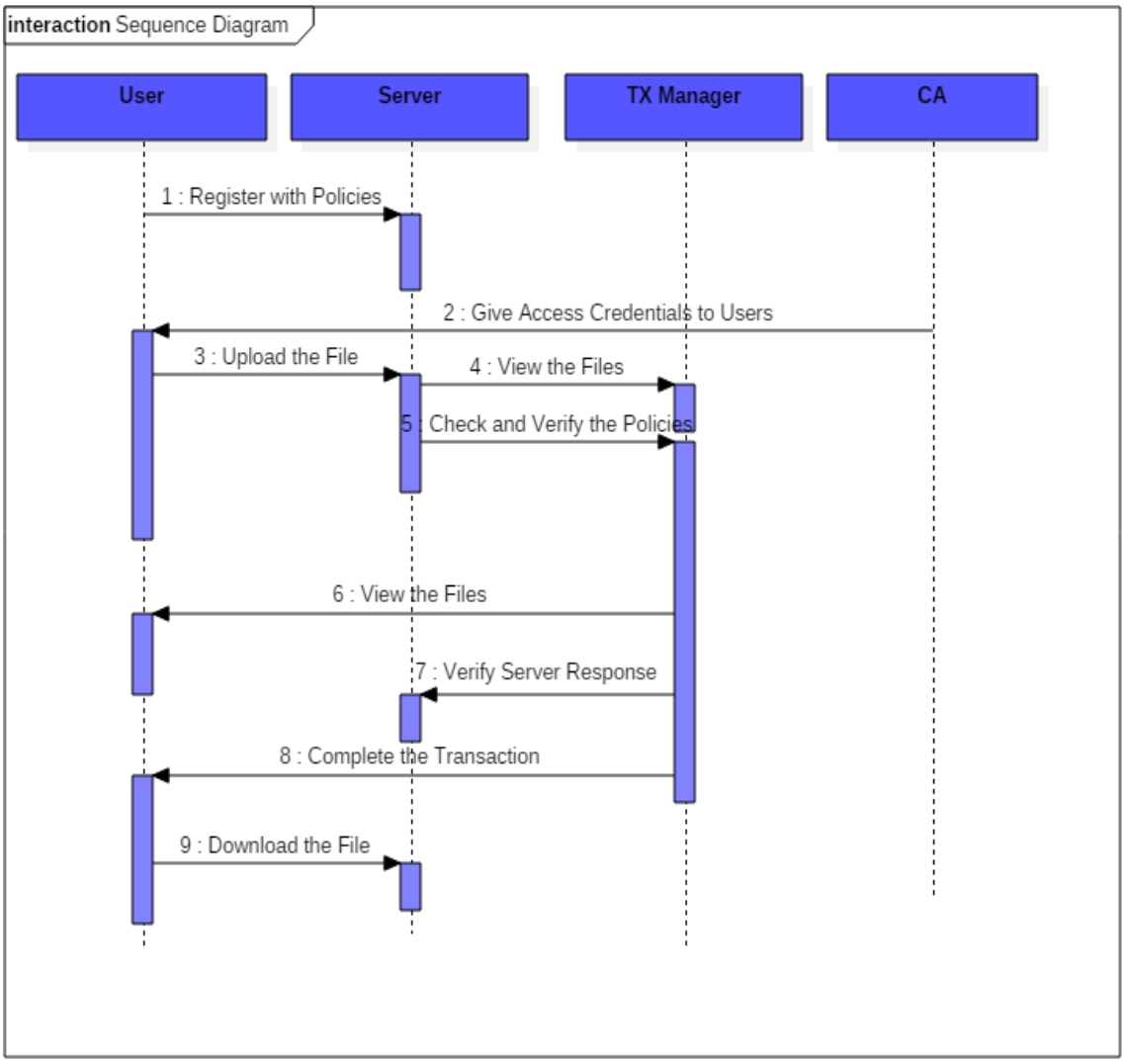
In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



3.5 SEQUENCE DIAGRAM:

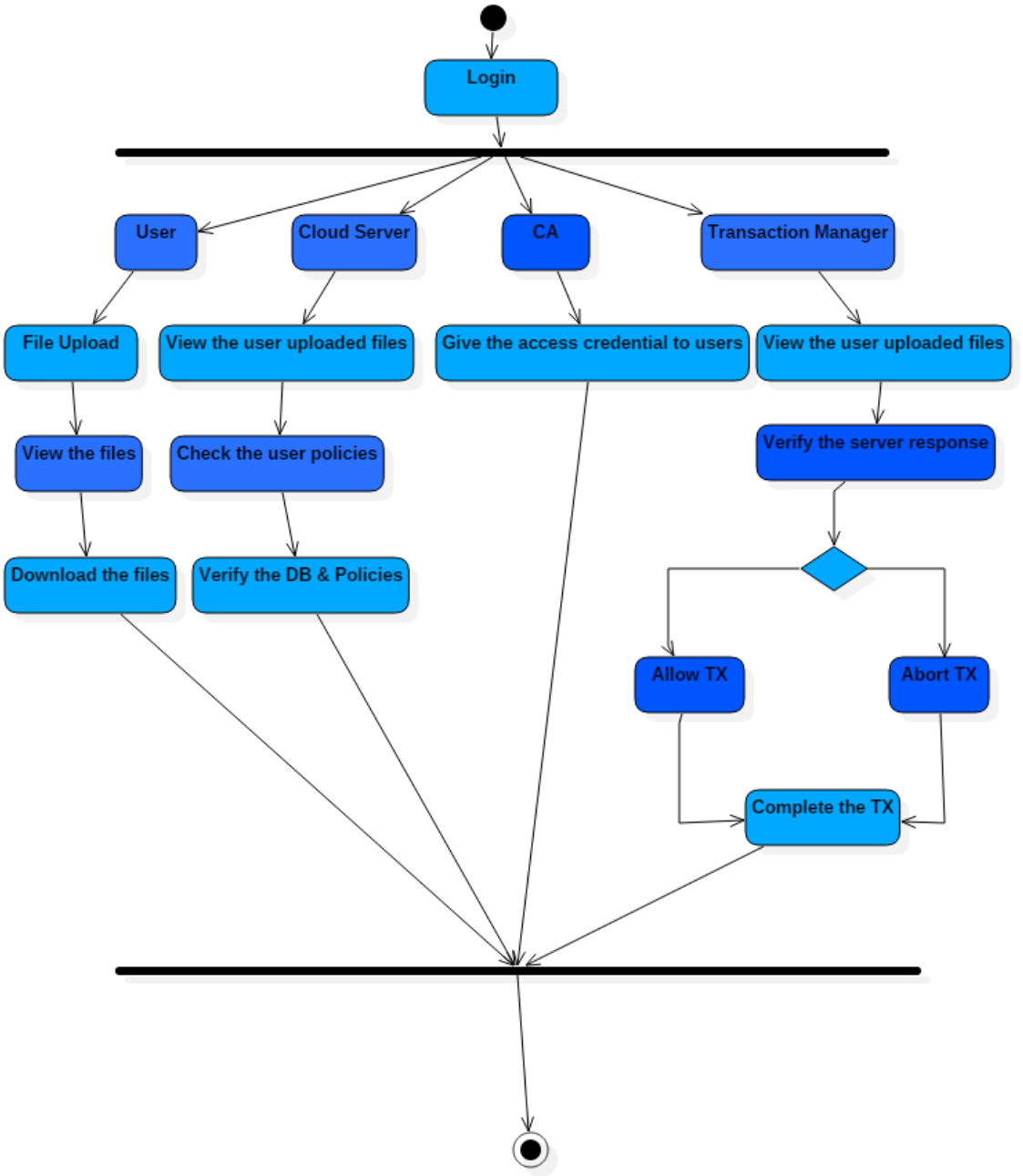
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a

construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



3.6 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.


Login
User
Cloud Server
CA
Transaction Manager
File Upload
View the user uploaded files
Give the access credential to users
View the user uploaded files
View the files
Check the user policies
Verify the server response
Download the files
Verify the DB & Policies
Allow TX
Abort TX
Complete the TX

4. IMPLEMENTATION

4.1 SOFTWARE ENVIRONMENT

The .NET Framework is a new computing platform that simplifies application development in the highly distributed environment of the Internet. The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that guarantees safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

The .NET Framework has two main components: the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and Remoting, while also enforcing strict type safety and other forms of code accuracy that ensure security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the

other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable Web Forms applications and XML Web services, both of which are discussed later in this topic.

Internet Explorer is an example of an unmanaged application that hosts the runtime (in the form of a MIME type extension). Using Internet Explorer to host the runtime enables you to embed managed components or Windows Forms controls in HTML documents. Hosting the runtime in this way makes managed mobile code (similar to Microsoft® ActiveX® controls) possible, but with significant improvements that only managed code can offer, such as semi-trusted execution and secure isolated file storage.

The following illustration shows the relationship of the common language runtime and the class library to your applications and to the overall system. The illustration also shows how managed code operates within a larger architecture.

FEATURES OF THE COMMON LANGUAGE RUNTIME

The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.

With regards to security, managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally featuring rich.

The runtime also enforces code robustness by implementing a strict type- and code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers

Generate managed code that conforms to the CTS. This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime eliminates many common software issues. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management resolves the two most common application errors, memory leaks and invalid memory references.

The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers.

Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.

While the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called just-in-time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to further increase performance.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Microsoft® SQL Server™ and Internet Information Services (IIS).

This infrastructure enables you to use managed code to write your business logic, while still enjoying the superior performance of the industry's best enterprise servers that support runtime hosting.

NET FRAMEWORK CLASS LIBRARY

The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new features of the .NET Framework. In addition, third-party components can integrate seamlessly with classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. Your collection classes will blend seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications.
- Scripted or hosted applications.
- Windows GUI applications (Windows Forms).
- ASP.NET applications.
- XML Web services.
- Windows services.

For example, the Windows Forms classes are a comprehensive set of reusable types that vastly simplify Windows GUI development. If you write an ASP.NET Web Form application, you can use the Web Forms classes.

CLIENT APPLICATION DEVELOPMENT

Client applications are the closest to a traditional style of application in Windows-based programming. These are the types of applications that display windows or forms on the desktop, enabling a user to perform a task.

Client applications include applications such as word processors and spreadsheets, as well as custom business applications such as data-entry tools, reporting tools, and so on. Client applications usually employ windows, menus, buttons, and other GUI elements, and they likely access local resources such as the file system and peripherals such as printers.

Another kind of client application is the traditional ActiveX control (now replaced by the managed Windows Forms control) deployed over the Internet as a Web page. This application is much like other client applications: it is executed natively, has access to local resources, and includes graphical elements.

In the past, developers created such applications using C/C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft® Visual Basic®. The .NET Framework incorporates aspects of these existing products into a single, consistent development environment that drastically simplifies the development of client applications.

The Windows Forms classes contained in the .NET Framework are designed to be used for GUI development. You can easily create command windows, buttons, menus, toolbars, and other screen elements with the flexibility necessary to accommodate shifting business needs.

For example, the .NET Framework provides simple properties to adjust visual attributes associated with forms. In some cases the underlying operating system does not support changing these attributes directly, and in these cases the .NET Framework automatically recreates the forms. This is one of many ways in which the .NET Framework integrates the developer interface, making coding simpler and more consistent.

Unlike ActiveX controls, Windows Forms controls have semi-trusted access to a user's computer. This means that binary or natively executing code can access some of the resources on the user's system (such as GUI elements and limited file access) without being able to access or compromise other resources. Because of code access security, many applications that once needed to be installed on a user's system can now be safely deployed through the Web. Your applications can implement the features of a local application while being deployed like a Web page.

ASP.NET

Server Application Development

Server-side applications in the managed world are implemented through runtime hosts. Unmanaged applications host the common language runtime, which allows your custom managed code to control the behavior of the server. This model provides you with

all the features of the common language runtime and class library while gaining the performance and scalability of the host server.

The following illustration shows a basic network schema with managed code running in different server environments. Servers such as IIS and SQL Server can perform standard operations while your application logic executes through the managed code.

SERVER-SIDE MANAGED CODE

ASP.NET is the hosting environment that enables developers to use the .NET Framework to target Web-based applications. However, ASP.NET is more than just a runtime host; it is a complete architecture for developing Web sites and Internet-distributed objects using managed code. Both Web Forms and XML Web services use IIS and ASP.NET as the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework.

XML Web services, an important evolution in Web-based technology, are distributed, server-side application components similar to common Web sites. However, unlike Web-based applications, XML Web services components have no UI and are not targeted for browsers such as Internet Explorer and Netscape Navigator. Instead, XML Web services consist of reusable software components designed to be consumed by other applications, such as traditional client applications, Web-based applications, or even other XML Web services. As a result, XML Web services technology is rapidly moving application development and deployment into the highly distributed environment of the Internet.

If you have used earlier versions of ASP technology, you will immediately notice the improvements that ASP.NET and Web Forms offers. For example, you can develop Web Forms pages in any language that supports the .NET Framework. In addition, your code no longer needs to share the same file with your HTTP text (although it can continue to do so if you prefer).

Web Forms pages execute in native machine language because, like any other managed application, they take full advantage of the runtime. In contrast, unmanaged ASP pages are always scripted and interpreted. ASP.NET pages are faster, more functional, and easier to develop than unmanaged ASP pages because they interact with the runtime like any managed application.

The .NET Framework also provides a collection of classes and tools to aid in development and consumption of XML Web services applications. XML Web services are built on standards such as SOAP (a remote procedure-call protocol), XML (an extensible data format), and WSDL (the Web Services Description Language). The .NET Framework is built on these standards to promote interoperability with non-Microsoft solutions.

For example, the Web Services Description Language tool included with the .NET Framework SDK can query an XML Web service published on the Web, parse its WSDL description, and produce C# or Visual Basic source code that your application can use to become a client of the XML Web service. The source code can create classes derived from classes in the class library that handle all the underlying communication using SOAP and XML parsing. Although you can use the class library to consume XML Web services directly, the Web Services Description Language tool and the other tools contained in the SDK facilitate your development efforts with the .NET Framework.

If you develop and publish your own XML Web service, the .NET Framework provides a set of classes that conform to all the underlying communication standards, such as SOAP, WSDL, and XML. Using those classes enables you to focus on the logic of your service, without concerning yourself with the communications infrastructure required by distributed software development.

Finally, like Web Forms pages in the managed environment, your XML Web service will run with the speed of native machine language using the scalable communication of IIS.

ACTIVE SERVER PAGES.NET

ASP.NET is a programming framework built on the common language runtime that can be used on a server to build powerful Web applications. ASP.NET offers several important advantages over previous Web development models:

- **Enhanced Performance.** ASP.NET is compiled common language runtime code running on the server. Unlike its interpreted predecessors, ASP.NET can take advantage of early binding, just-in-time compilation, native optimization, and caching services right out of the box. This amounts to dramatically better performance before you ever write a line of code.
- **World-Class Tool Support.** The ASP.NET framework is complemented by a rich toolbox and designer in the Visual Studio integrated development environment. WYSIWYG editing, drag-and-drop server controls, and automatic deployment are just a few of the features this powerful tool provides.
- **Power and Flexibility.** Because ASP.NET is based on the common language runtime, the power and flexibility of that entire platform is available to Web application developers. The .NET Framework class library, Messaging, and Data Access solutions are all seamlessly accessible from the Web. ASP.NET is also language-independent, so you can choose the language that best applies to your application or partition your application across many languages. Further, common language runtime interoperability guarantees that your existing investment in COM-based development is preserved when migrating to ASP.NET.
- **Simplicity.** ASP.NET makes it easy to perform common tasks, from simple form submission and client authentication to deployment and site configuration. For example, the ASP.NET page framework allows you to build user interfaces that cleanly separate application logic from presentation code and to handle events in a simple, Visual Basic - like forms processing model. Additionally, the common language runtime simplifies development, with managed code services such as automatic reference counting and garbage collection.

- **Manageability.** ASP.NET employs a text-based, hierarchical configuration system, which simplifies applying settings to your server environment and Web applications. Because configuration information is stored as plain text, new settings may be applied without the aid of local administration tools. This "zero local administration" philosophy extends to deploying ASP.NET Framework applications as well. An ASP.NET Framework application is deployed to a server simply by copying the necessary files to the server. No server restart is required, even to deploy or replace running compiled code.
- **Scalability and Availability.** ASP.NET has been designed with scalability in mind, with features specifically tailored to improve performance in clustered and multiprocessor environments. Further, processes are closely monitored and managed by the ASP.NET runtime, so that if one misbehaves (leaks, deadlocks), a new process can be created in its place, which helps keep your application constantly available to handle requests.
- **Customizability and Extensibility.** ASP.NET delivers a well-factored architecture that allows developers to "plug-in" their code at the appropriate level. In fact, it is possible to extend or replace any subcomponent of the ASP.NET runtime with your own custom-written component. Implementing custom authentication or state services has never been easier.
- **Security.** With built in Windows authentication and per-application configuration, you can be assured that your applications are secure.

LANGUAGE SUPPORT

The Microsoft .NET Platform currently offers built-in support for three languages: C#, Visual Basic, and JScript.

WHAT IS ASP.NET WEB FORMS?

The ASP.NET Web Forms page framework is a scalable common language runtime programming model that can be used on the server to dynamically generate Web pages.

Intended as a logical evolution of ASP (ASP.NET provides syntax compatibility with existing pages), the ASP.NET Web Forms framework has been specifically designed to address a number of key deficiencies in the previous model. In particular, it provides:

- The ability to create and use reusable UI controls that can encapsulate common functionality and thus reduce the amount of code that a page developer has to write.
- The ability for developers to cleanly structure their page logic in an orderly fashion (not "spaghetti code").
- The ability for development tools to provide strong WYSIWYG design support for pages (existing ASP code is opaque to tools).

ASP.NET Web Forms pages are text files with an .aspx file name extension. They can be deployed throughout an IIS virtual root directory tree. When a browser client requests .aspx resources, the ASP.NET runtime parses and compiles the target file into a .NET Framework class. This class can then be used to dynamically process incoming requests. (Note that the .aspx file is compiled only the first time it is accessed; the compiled type instance is then reused across multiple requests).

An ASP.NET page can be created simply by taking an existing HTML file and changing its file name extension to .aspx (no modification of code is required). For example, the following sample demonstrates a simple HTML page that collects a user's name and category preference and then performs a form postback to the originating page when a button is clicked:

ASP.NET provides syntax compatibility with existing ASP pages. This includes support for <% %> code render blocks that can be intermixed with HTML content within an .aspx file. These code blocks execute in a top-down manner at page render time.

CODE-BEHIND WEB FORMS

ASP.NET supports two methods of authoring dynamic pages. The first is the method shown in the preceding samples, where the page code is physically declared within the originating .aspx file. An alternative approach--known as the code-behind method--

enables the page code to be more cleanly separated from the HTML content into an entirely separate file.

INTRODUCTION TO ASP.NET SERVER CONTROLS

In addition to (or instead of) using <% %> code blocks to program dynamic content, ASP.NET page developers can use ASP.NET server controls to program Web pages. Server controls are declared within an .aspx file using custom tags or intrinsic HTML tags that contain a **runat="server"** attributes value. Intrinsic HTML tags are handled by one of the controls in the **System.Web.UI.HtmlControls** namespace. Any tag that doesn't explicitly map to one of the controls is assigned the type of **System.Web.UI.HtmlControls.HtmlGenericControl**.

Server controls automatically maintain any client-entered values between round trips to the server. This control state is not stored on the server (it is instead stored within an **<input type="hidden">** form field that is round-tripped between requests). Note also that no client-side script is required.

In addition to supporting standard HTML input controls, ASP.NET enables developers to utilize richer custom controls on their pages. For example, the following sample demonstrates how the **<asp:adrotator>** control can be used to dynamically display rotating ads on a page.

1. ASP.NET Web Forms provide an easy and powerful way to build dynamic Web UI.
2. ASP.NET Web Forms pages can target any browser client (there are no script library or cookie requirements).
3. ASP.NET Web Forms pages provide syntax compatibility with existing ASP pages.
4. ASP.NET server controls provide an easy way to encapsulate common functionality.
5. ASP.NET ships with 45 built-in server controls. Developers can also use controls built by third parties.
6. ASP.NET server controls can automatically project both uplevel and downlevel HTML.

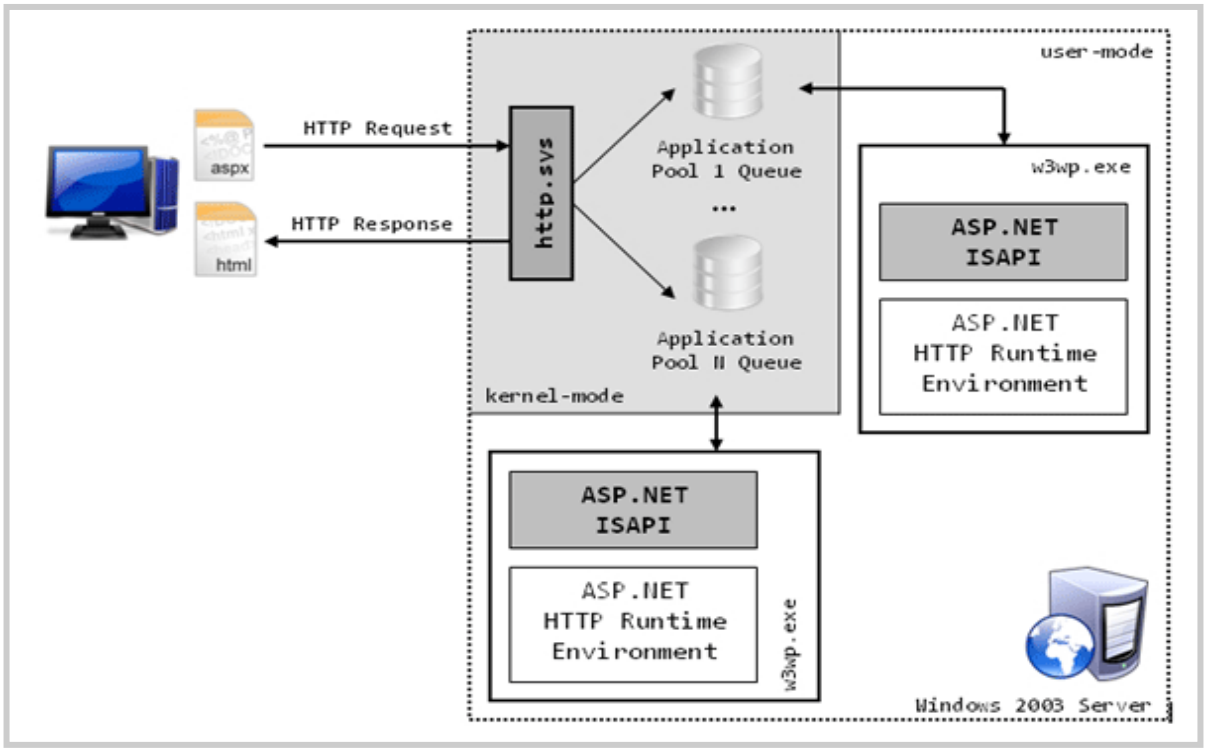
7. ASP.NET templates provide an easy way to customize the look and feel of list server controls.
8. ASP.NET validation controls provide an easy way to do declarative client or server data validation.

IIS 6.0 Process Model

The IIS 6 process model is the default model on machines running Windows 2003 Server operating system. It introduces several changes and improvements over the IIS 5 process model. One of the biggest changes is the concept of application pools. On IIS 5.X all web applications, that is, all AppDomains, were hosted by the ASP.NET worker process. To achieve a finer granularity over security boundaries and personalization, the IIS 6 process model allows applications to run inside different copies of a new worker process, w3wp.exe. Each application pool can contain multiple AppDomains and is hosted in a single copy of the worker process. In other words, the shift is from a single process hosting all applications to multiple processes hosting each an application pool. This model is also called the worker process isolation mode.

Another big change from the previous model is the way IIS listens for incoming requests. With the IIS 5 model, it was the IIS process, inetinfo.exe, who was listening on a specific TCP port for HTTP requests. In the IIS 6 architecture, incoming requests are handled and queued at kernel level instead of user mode via a kernel driver called http.sys; this approach has several advantages over the old model and is called kernel-level request queuing.

The IIS 6.0 process model



The above figure illustrates the principal components taking part in the request processing when using the II 6 model. Once a request arrives the kernel level device driver http.sys routes it to the right application pool queue. Each queue belongs to a specific application pool, and thus to a specific copy of the worker process, which next receives the request from the queue. This approach highly reduces the overhead introduced by named pipes used in IIS 5 model since no inter process communication is taking place, but the requests are headed to the worker process directly from the kernel level driver. This has many advantages concerning reliability, too. Since running in kernel mode, the request dispatching isn't influenced by crashes and malfunctions happing at user level, that is, in the worker processes. Thereby, even if a worker process crashes, the system is still capable of accepting incoming requests and eventually restarts the crashed process.

It’s the worker process who is in charge of loading the ASP.NET ISAPI extension, which, in turn, loads the CRL and delegates all the work to the HTTP Runtime.

The w3wp.exe worker process, differently from the aspnet_wp.exe process used in IIS 5 model, isn’t ASP.NET specific, and is used to handle any kind of requests. The specific worker process then decides which ISAPI modules to load according to the type of resources it needs to serve.

A detail not underlined in Figure 3 for simplicity reasons is that incoming requests are forwarded from the application pool queue to the right worker process via a module loaded in IIS 6 called Web Administration Service (WAS). This module is responsible for reading worker process – web application bindings from the IIS metabase and forwarding the request to the right worker process.

IIS Request Processing

This topic describes how a client request is processed on an IIS server

Request Processing by Type of Request

The following table describes what happens once an IIS process receives a request to be executed.

Request	Action
HTML Page	IIS returns the page in HTML format.
A file whose extension is mapped to a particular ISAPI extension, such as Asp.dll	IIS loads the appropriate DLL file and presents the request through the **Extension_Control_Block** data structure. For example, the .asp extension is mapped to Asp.dll, so that all requests for files with an .asp extension will be directed to Asp.dll. The .stm and .shtm extensions are mapped to the Ssinc.dll.

ISAPI extension	IIS loads the ISAPI DLL (if it is not already running) and the request is sent to the extension through the EXTENSION_CONTROL_BLOCK structure data structure.
CGI application	IIS creates a new process. IIS will then provide the query string and other parameters that are included with the request through the environment and standard input (STDIN) handle for the process.

ISAPI filters are always loaded as long as the Web service is running and a request server.

Before an IIS process receives a request to execute, some preliminary processing occurs that is described in the following steps:

1. A request arrives at HTTP.sys.
2. HTTP.sys determines if the request is valid. If the request is not valid, it sends a code for an invalid request back to the client.
3. If the request is valid, HTTP.sys checks to see if the request is for static content (HTML) because static content can be served immediately.
4. If the request is for dynamic content, HTTP.sys checks to see if the response is located in its kernel-mode cache.
5. If the response is in the cache, HTTP.sys returns the response immediately.
6. If the response is not cached, HTTP.sys determines the correct request queue, and places the request in that queue.
7. If the queue has no worker processes assigned to it, HTTP.sys signals the WWW service to start one.

8. The worker process pulls the request from the queue and processes the request, evaluating the URL to determine the type of request (ASP, ISAPI, or CGI).
9. The worker process sends the response back to HTTP.sys.
10. HTTP.sys sends the response back to the client and logs the request, if configured to do

Preliminary Request Processing on IIS 6.0 in IIS 5.0 Isolation Mode and Earlier Versions of IIS

The request processing of IIS 6.0 running in IIS 5.0 isolation mode is nearly identical to the request processing in IIS 5.1, IIS 5.0, and IIS 4.0. Before an IIS process receives a request to execute, some preliminary processing occurs that is described in the following steps:

1. A request arrives. If the requested application is running in-process, then Inetinfo.exe takes the request. If not, then DLLHost.exe takes the request.
2. Inetinfo.exe or DLLHost.exe determines if the request is valid. If the request is not valid, it sends a code for an invalid request back to the client.
3. If the request is valid, Inetinfo.exe or DLLHost.exe checks to see if the response is located in the IIS cache.
4. If the response is in the cache, it is returned immediately.
5. If the response is not cached, Inetinfo.exe or DLLHost.exe processes the request, evaluating the URL to determine if the request is for static content (HTML), or dynamic content (ASP, ASP.NET or ISAPI).
6. The response is sent back to the client and the request is logged, if IIS is configured to do so.

C#.NET

ADO.NET OVERVIEW

ADO.NET is an evolution of the ADO data access model that directly addresses user requirements for developing scalable applications. It was designed specifically for the web with scalability, statelessness, and XML in mind.

ADO.NET uses some ADO objects, such as the **Connection** and **Command** objects, and also introduces new objects. Key new ADO.NET objects include the **DataSet**, **DataReader**, and **DataAdapter**.

The important distinction between this evolved stage of ADO.NET and previous data architectures is that there exists an object -- the **DataSet** -- that is separate and distinct from any data stores. Because of that, the **DataSet** functions as a standalone entity. You can think of the DataSet as an always disconnected recordset that knows nothing about the source or destination of the data it contains. Inside a **DataSet**, much like in a database, there are tables, columns, relationships, constraints, views, and so forth.

A **DataAdapter** is the object that connects to the database to fill the **DataSet**. Then, it connects back to the database to update the data there, based on operations performed while the **DataSet** held the data. In the past, data processing has been primarily connection-based. Now, in an effort to make multi-tiered apps more efficient, data processing is turning to a message-based approach that revolves around chunks of information. At the center of this approach is the **DataAdapter**, which provides a bridge to retrieve and save data between a **DataSet** and its source data store. It accomplishes this by means of requests to the appropriate SQL commands made against the data store.

The XML-based **DataSet** object provides a consistent programming model that works with all models of data storage: flat, relational, and hierarchical. It does this by having no 'knowledge' of the source of its data, and by representing the data that it holds as collections and data types. No matter what the source of the data within the **DataSet**

is, it is manipulated through the same set of standard APIs exposed through the **DataSet** and its subordinate objects.

While the **DataSet** has no knowledge of the source of its data, the managed provider has detailed and specific information. The role of the managed provider is to connect, fill, and persist the **DataSet** to and from data stores. The OLE DB and SQL Server .NET Data Providers (System.Data.OleDb and System.Data.SqlClient) that are part of the .Net Framework provide four basic objects: the **Command**, **Connection**, **DataReader** and **DataAdapter**. In the remaining sections of this document, we'll walk through each part of the **DataSet** and the OLE DB/SQL Server .NET Data Providers explaining what they are, and how to program against them.

The following sections will introduce you to some objects that have evolved, and some that are new. These objects are:

- **Connections**. For connection to and managing transactions against a database.
- **Commands**. For issuing SQL commands against a database.
- **DataReaders**. For reading a forward-only stream of data records from a SQL Server data source.
- **DataSets**. For storing, Remoting and programming against flat data, XML data and relational data.
- **DataAdapters**. For pushing data into a **DataSet**, and reconciling data against a database.

When dealing with connections to a database, there are two different options: SQL Server .NET Data Provider (System.Data.SqlClient) and OLE DB .NET Data Provider (System.Data.OleDb). In these samples we will use the SQL Server .NET Data Provider. These are written to talk directly to Microsoft SQL Server. The OLE DB .NET Data Provider is used to talk to any OLE DB provider (as it uses OLE DB underneath).

Connections:

Connections are used to 'talk to' databases, and are represented by provider-specific classes such as **SqlConnection**. Commands travel over connections and

resultsets are returned in the form of streams which can be read by a **DataReader** object, or pushed into a **DataSet** object.

Commands:

Commands contain the information that is submitted to a database, and are represented by provider-specific classes such as **SqlCommand**. A command can be a stored procedure call, an UPDATE statement, or a statement that returns results. You can also use input and output parameters, and return values as part of your command syntax. The example below shows how to issue an INSERT statement against the **Northwind** database.

DataReaders:

The **DataReader** object is somewhat synonymous with a read-only/forward-only cursor over data. The **DataReader** API supports flat as well as hierarchical data. A **DataReader** object is returned after executing a command against a database. The format of the returned **DataReader** object is different from a recordset. For example, you might use the **DataReader** to show the results of a search list in a web page.

DATASETS AND DATAADAPTERS:

DataSets

The **DataSet** object is similar to the ADO **Recordset** object, but more powerful, and with one other important distinction: the **DataSet** is always disconnected. The **DataSet** object represents a cache of data, with database-like structures such as tables, columns, relationships, and constraints. However, though a **DataSet** can and does behave much like a database, it is important to remember that **DataSet** objects do not interact directly with databases, or other source data. This allows the developer to work with a programming model that is always consistent, regardless of where the source data resides. Data coming from a database, an XML file, from code, or user input can all be placed into **DataSet** objects. Then, as changes are made to the **DataSet** they can be tracked and verified before updating the source data. The **GetChanges** method of the **DataSet** object actually creates a second **DatSet** that contains only the changes to the

data. This **DataSet** is then used by a **DataAdapter** (or other objects) to update the original data source.

The **DataSet** has many XML characteristics, including the ability to produce and consume XML data and XML schemas. XML schemas can be used to describe schemas interchanged via WebServices. In fact, a **DataSet** with a schema can actually be compiled for type safety and statement completion.

DATAADAPTERS (OLEDB/SQL)

The **DataAdapter** object works as a bridge between the **DataSet** and the source data. Using the provider-specific **SqlDataAdapter** (along with its associated **SqlCommand** and **SqlConnection**) can increase overall performance when working with a Microsoft SQL Server databases. For other OLE DB-supported databases, you would use the **OleDbDataAdapter** object and its associated **OleDbCommand** and **OleDbConnection** objects.

The **DataAdapter** object uses commands to update the data source after changes have been made to the **DataSet**. Using the **Fill** method of the **DataAdapter** calls the SELECT command; using the **Update** method calls the INSERT, UPDATE or DELETE command for each changed row. You can explicitly set these commands in order to control the statements used at runtime to resolve changes, including the use of stored procedures. For ad-hoc scenarios, a **CommandBuilder** object can generate these at run-time based upon a select statement. However, this run-time generation requires an extra round-trip to the server in order to gather required metadata, so explicitly providing the INSERT, UPDATE, and DELETE commands at design time will result in better run-time performance.

1. ADO.NET is the next evolution of ADO for the .Net Framework.
2. ADO.NET was created with n-Tier, statelessness and XML in the forefront. Two new objects, the **DataSet** and **DataAdapter**, are provided for these scenarios.
3. ADO.NET can be used to get data from a stream, or to store data in a cache for updates.
4. There is a lot more information about ADO.NET in the documentation.

5. Remember, you can execute a command directly against the database in order to do inserts, updates, and deletes. You don't need to first put data into a **DataSet** in order to insert, update, or delete it.
6. Also, you can use a **DataSet** to bind to the data, move through the data, and navigate data relationships

SQL SERVER

A database management, or DBMS, gives the user access to their data and helps them transform the data into information. Such database management systems include dBase, paradox, IMS, SQL Server and SQL Server. These systems allow users to create, update and extract information from their database.

A database is a structured collection of data. Data refers to the characteristics of people, things and events. SQL Server stores each data item in its own fields. In SQL Server, the fields relating to a particular person, thing or event are bundled together to form a single complete unit of data, called a record (it can also be referred to as raw or an occurrence). Each record is made up of a number of fields. No two fields in a record can have the same field name.

During an SQL Server Database design project, the analysis of your business needs identifies all the fields or attributes of interest. If your business needs change over time, you define any additional fields or change the definition of existing fields.

SQL SERVER TABLES

SQL Server stores records relating to each other in a table. Different tables are created for the various groups of information. Related tables are grouped together to form a database.

PRIMARY KEY

Every table in SQL Server has a field or a combination of fields that uniquely identifies each record in the table. The Unique identifier is called the Primary Key, or simply the Key. The primary key provides the means to distinguish one record from all

other in a table. It allows the user and the database system to identify, locate and refer to one particular record in the database.

RELATIONAL DATABASE

Sometimes all the information of interest to a business operation can be stored in one table. SQL Server makes it very easy to link the data in multiple tables. Matching an employee to the department in which they work is one example. This is what makes SQL Server a relational database management system, or RDBMS.

It stores data in two or more tables and enables you to define relationships between the table and enables you to define relationships between the tables.

FOREIGN KEY

When a field is one table matches the primary key of another field is referred to as a foreign key. A foreign key is a field or a group of fields in one table whose values match those of the primary key of another table.

REFERENTIAL INTEGRITY

Not only does SQL Server allow you to link multiple tables, it also maintains consistency between them. Ensuring that the data among related tables is correctly matched is referred to as maintaining referential integrity.

DATA ABSTRACTION

A major purpose of a database system is to provide users with an abstract view of the data. This system hides certain details of how the data is stored and maintained. Data abstraction is divided into three levels.

Physical level: This is the lowest level of abstraction at which one describes how the data are actually stored.

Conceptual Level: At this level of database abstraction all the attributed and what data are actually stored is described and entries and relationship among them.

View level: This is the highest level of abstraction at which one describes only part of the database.

ADVANTAGES OF RDBMS

- Redundancy can be avoided
- Inconsistency can be eliminated
- Data can be Shared
- Standards can be enforced
- Security restrictions ca be applied
- Integrity can be maintained

DISADVANTAGES OF DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing of developing the software, the hardware has to be upgraded to allow for the extensive programs and the workspace required for their execution and storage. While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in case of failure the data can be recovered.

FEATURES OF SQL SERVER (RDBMS)

SQL SERVER is one of the leading database management systems (DBMS) because it is the only Database that meets the uncompromising requirements of today's most demanding information systems. From complex decision support systems (DSS) to the most rigorous online transaction processing (OLTP) application, even application that require simultaneous DSS and OLTP access to the same critical data, SQL Server leads the industry in both performance and capability

SQL SERVER is a truly portable, distributed, and open DBMS that delivers unmatched performance, continuous operation and support for every database.

SQL SERVER RDBMS is high performance fault tolerant DBMS which is specially designed for online transactions processing and for handling large database application.

SQL SERVER with transactions processing option offers two features which contribute to very high level of transaction processing throughput, which are

- The row level lock manager

ENTERPRISE WIDE DATA SHARING

The unrivaled portability and connectivity of the SQL SERVER DBMS enables all the systems in the organization to be linked into a singular, integrated computing resource.

PORTABILITY

SQL SERVER is fully portable to more than 80 distinct hardware and operating systems platforms, including UNIX, MSDOS, OS/2, Macintosh and dozens of proprietary platforms. This portability gives complete freedom to choose the database sever platform that meets the system requirements.

OPEN SYSTEMS

SQL SERVER offers a leading implementation of industry –standard SQL. SQL Server's open architecture integrates SQL SERVER and non –SQL SERVER DBMS with industries most comprehensive collection of tools, application, and third party software products SQL Server's Open architecture provides transparent access to data from other relational database and even non-relational database.

DISTRIBUTED DATA SHARING

SQL Server's networking and distributed database capabilities to access data stored on remote server with the same ease as if the information was stored on a single local computer. A single SQL statement can access data at multiple sites. You can store data where system requirements such as performance, security or availability dictate.

UNMATCHED PERFORMANCE

The most advanced architecture in the industry allows the SQL SERVER DBMS to deliver unmatched performance.

SOPHISTICATED CONCURRENCY CONTROL

Real World applications demand access to critical data. With most database Systems application becomes "contention bound" – which performance is limited not by

the CPU power or by disk I/O, but user waiting on one another for data access . SQL Server employs full, unrestricted row-level locking and contention free queries to minimize and in many cases entirely eliminates contention wait times.

NO I/O BOTTLENECKS

SQL Server's fast commit groups commit and deferred write technologies dramatically reduce disk I/O bottlenecks. While some database write whole data block to disk at commit time, SQL Server commits transactions with at most sequential log file on disk at commit time, On high throughput systems, one sequential writes typically group commit multiple transactions. Data read by the transaction remains as shared memory so that other transactions may access that data without reading it again from disk.

MODULES:

1. Server Module.
2. Cloud User Module.
3. Transaction Manager.
4. Certificate Authorities.

4.2. MODULE DESCRIPTIONS:

Server Model

In this Module, We design a cloud infrastructure consisting of a set of servers, where each server is responsible for hosting a subset of all data items belonging to a specific application domain.

Cloud User Module

* In this Module, Users interact with the system by submitting queries or update requests encapsulated in ACID transactions.
* Since transactions are executed over time, the state information of the credentials and the policies enforced by different servers are subject to changes

at any time instance, therefore it becomes important to introduce precise definitions for the different consistency levels that could be achieved within a transaction's lifetime. These consistency models strengthen the trusted transaction definition by defining the environment in which policy versions are consistent relative to the rest of the system. Before we do that, we define a transaction's view in terms of the different proofs of authorization evaluated during the lifetime of a particular transaction.

Transaction Manager

* A transaction is submitted to a Transaction Manager(TM) that coordinates its execution. Multiple TMs could be invoked as the system workload increases for load balancing, but each transaction is handled by only one TM.
* A common characteristic of most of our proposed approaches to achieve trusted transactions is the need for policy consistency validation at the end of a transaction.

Certificate Authorities

* We use the set of all credentials, which are issued by the Certificate Authorities (CAs) within the system. We assume that each CA offers an online method that allows any server to check the current status of credentials that it has issued.
* In this module, we provide a Safe transaction. A safe transaction is a transaction that is both trusted (i.e., satisfies the correctness properties of proofs of authorization) and database correct (i.e., satisfies the data integrity constraints).
* In this module, also develop Two Phase Validation system. As the name implies, 2PV operates in two phases: collection and validation. During collection, the TM first sends a Prepare-to-Validate message to each participant server. In response to this message, each participant 1) evaluates the proofs for each query of the transaction using the latest policies it has available and 2) sends a reply back to the TM containing the truth value (TRUE/FALSE) of those proofs along with the version number and policy identifier for each policy used.

4.3: SAMPLE CODE

Registration Page:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.SqlClient;
using System.Configuration;


public partial class Register : System.Web.UI.Page
{
   SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["BalancingCon"].Connection
String);
   protected void Page_Load(object sender, EventArgs e)
   {


   }
   protected void Button2_Click(object sender, EventArgs e)
   {
      clear();
      Label16.Text = "";
   }
   private void clear()
   {
      TextBox1.Text = "";
```

```
    TextBox2.Text = "";
    TextBox3.Text = "";
    TextBox4.Text = "";
    TextBox5.Text = "";
    TextBox6.Text = "";
    CheckBox1.Checked = false;
    CheckBox2.Checked = false;
    CheckBox3.Checked = false;
    CheckBox4.Checked = false;
    CheckBox5.Checked = false;
    CheckBox6.Checked = false;
    CheckBox7.Checked = false;
    CheckBox8.Checked = false;
    CheckBox9.Checked = false;
}
bool txtt, docc, pdff, jpgg, pngg, giff, mp33, mp44, avii;
protected void Button1_Click(object sender, EventArgs e)
{
    if (CheckBox1.Checked == true)
    {
        txtt = true;
    }
    if (CheckBox2.Checked == true)
    {
        docc = true;
    }
    if (CheckBox3.Checked == true)
    {
```

```
    pdff = true;
}
if (CheckBox4.Checked == true)
{
    jpgg = true;
}
if (CheckBox5.Checked == true)
{
    pngg = true;
}
if (CheckBox6.Checked == true)
{
    giff = true;
}
if (CheckBox7.Checked == true)
{
    mp33 = true;
}
if (CheckBox8.Checked == true)
{
    mp44 = true;
}
if (CheckBox9.Checked == true)
{
    avii = true;
}
string gender = RadioButtonList1.SelectedItem.ToString();
string acces = "NO";
```

```
        con.Open();

        SqlCommand cmd = new SqlCommand("insert into Register values('" +
TextBox1.Text + "','" + TextBox2.Text

        + "','" + TextBox3.Text + "','" + TextBox4.Text + "','" + gender + "','" +
TextBox5.Text + "','" + TextBox6.Text

        +
"','"+txtt+"','"+docc+"','"+pdff+"','"+jpgg+"','"+pngg+"','"+giff+"','"+mp33+"','"+mp44+
"','"+avii+"','"+acces+"')",con);

        cmd.ExecuteNonQuery();

        con.Close();

        clear();

        Label16.Text = "Registered Successfully...";

    }

}
```

LogIn Page:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Configuration;

using System.Data.SqlClient;


public partial class Login : System.Web.UI.Page

{

    SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["BalancingCon"].Connection
String);
```

```
protected void Page_Load(object sender, EventArgs e)
{

}
protected void Button1_Click(object sender, EventArgs e)
{
    con.Open();
    SqlCommand cmd = new SqlCommand("select Username from Register where Username = '"+TextBox1.Text+"'",con);
    SqlCommand cmd1 = new SqlCommand("select Password from Register where Username = '" + TextBox1.Text + "'", con);
    SqlCommand cmd2 = new SqlCommand("select LoginAccess from Register where Username = '" + TextBox1.Text + "'", con);
    string uname = (string)cmd.ExecuteScalar();
    string pwd = (string)cmd1.ExecuteScalar();
    string logacc = (string)cmd2.ExecuteScalar();
    con.Close();

    if (TextBox1.Text == uname && TextBox2.Text == pwd)
    {
        if (logacc == "YES")
        {
            Session["username"] = TextBox1.Text;
            Response.Redirect("fileupload.aspx");
        }
        else
        {
            Label6.Text = "You Are not a Authorised User.";
        }
```

```
        }
        else
        {
            Label6.Text = "InCorrect Details...";
        }
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        TextBox1.Text = "";
        TextBox2.Text = "";
    }
}
```

5. TESTING

5.1 INTRODUCTION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

5.2 TYPES OF TESTS

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

FUNCTIONAL TEST

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

SYSTEM TEST

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested.

Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

UNIT TESTING:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

TEST STRATEGY AND APPROACH

Field testing will be performed manually and functional tests will be written in detail.

TEST OBJECTIVES

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

INTEGRATION TESTING

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

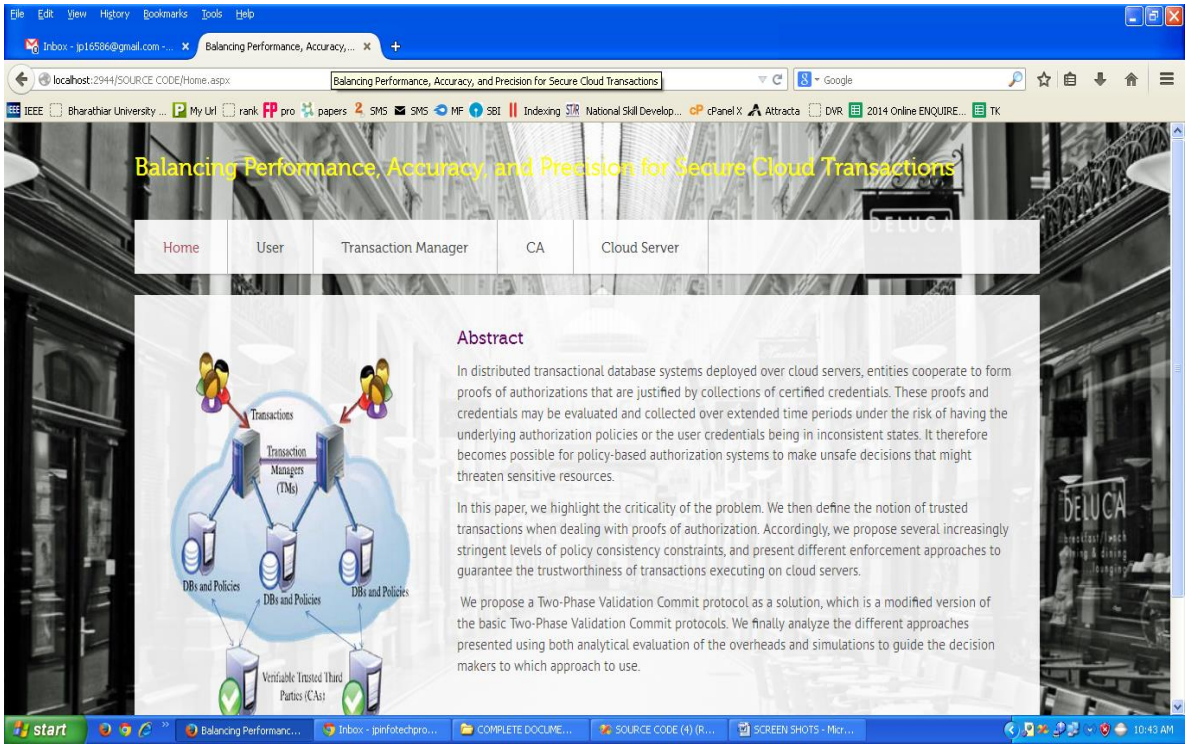
ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

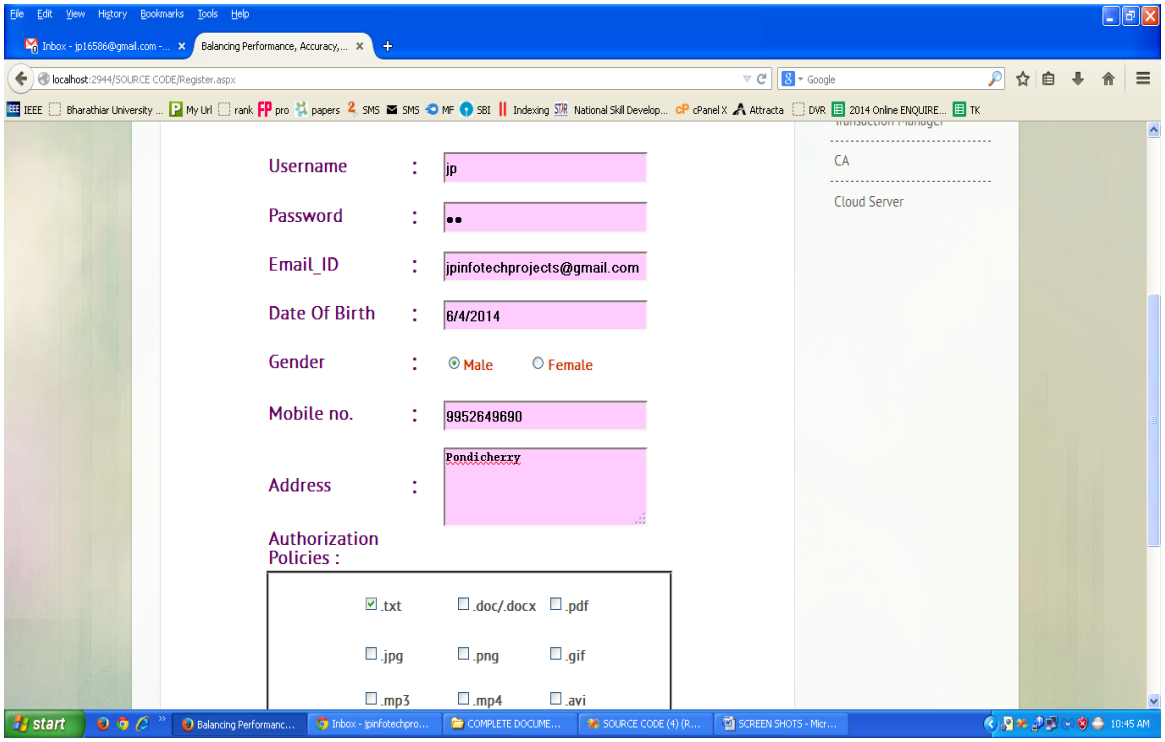
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6. RESULTS

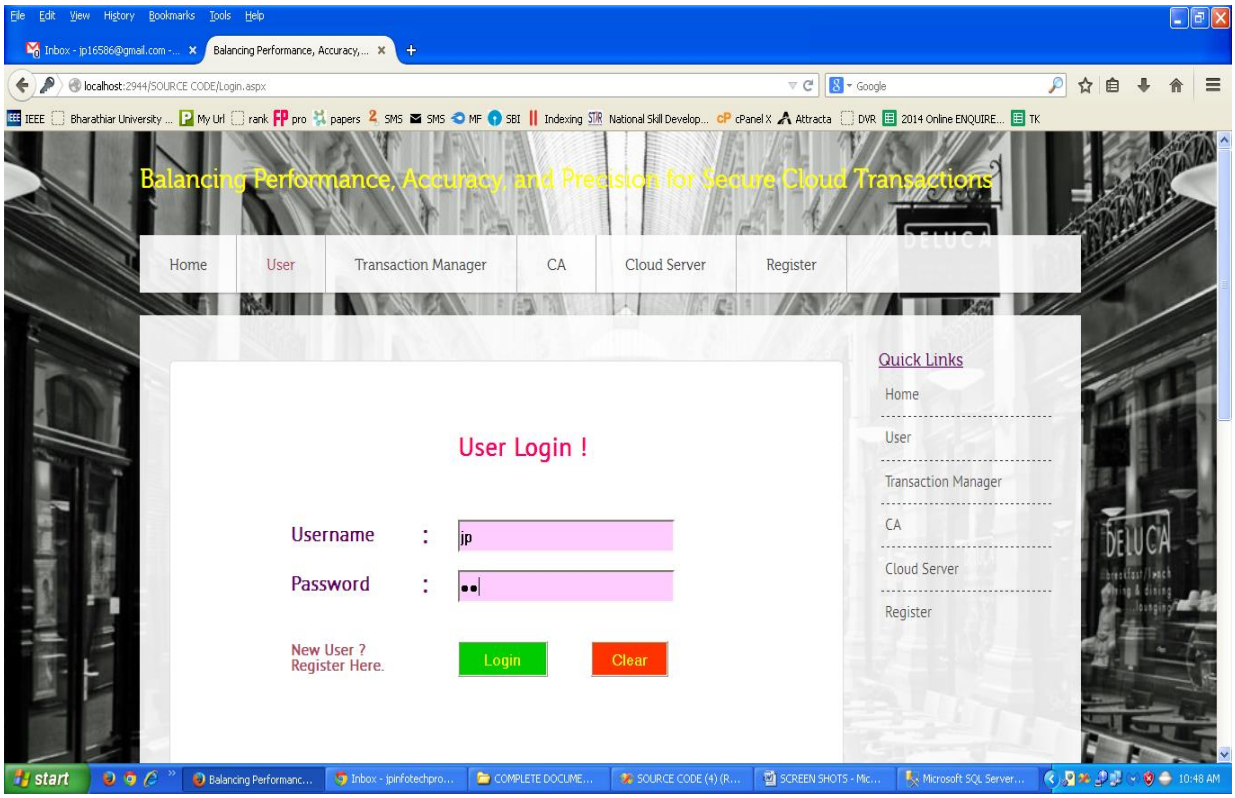
Home Page:



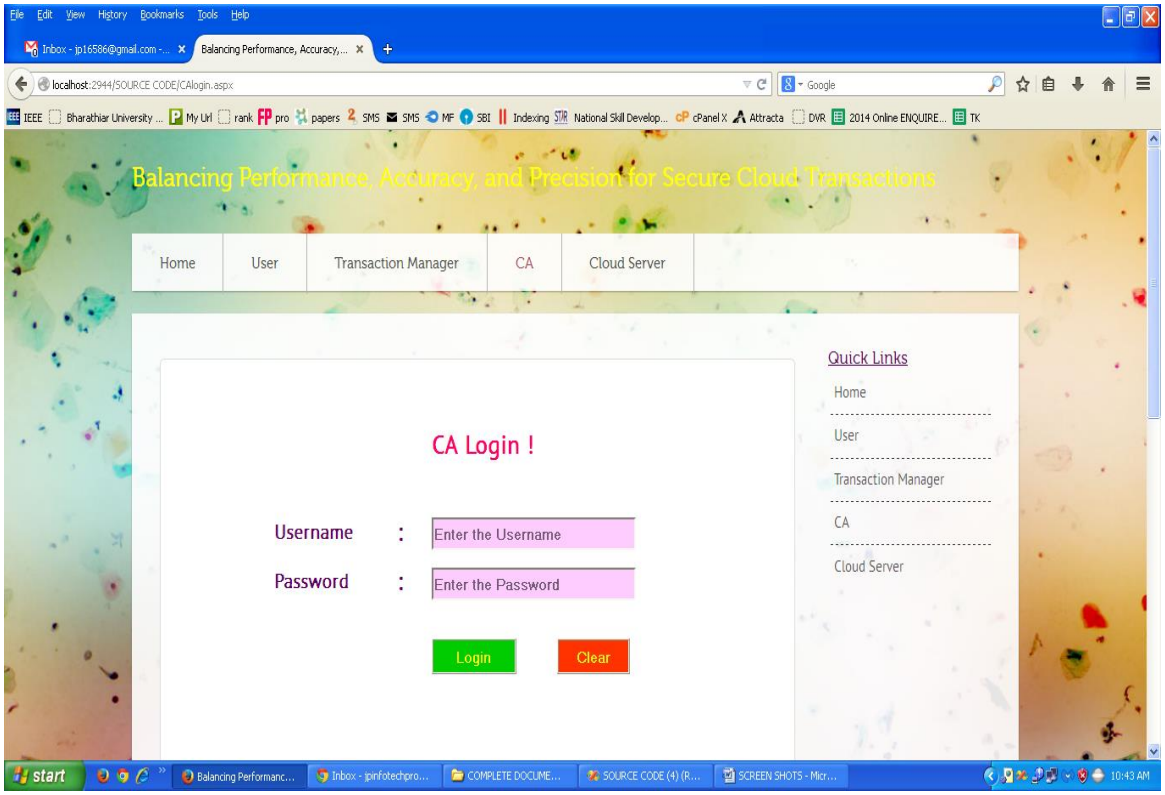
User Registration



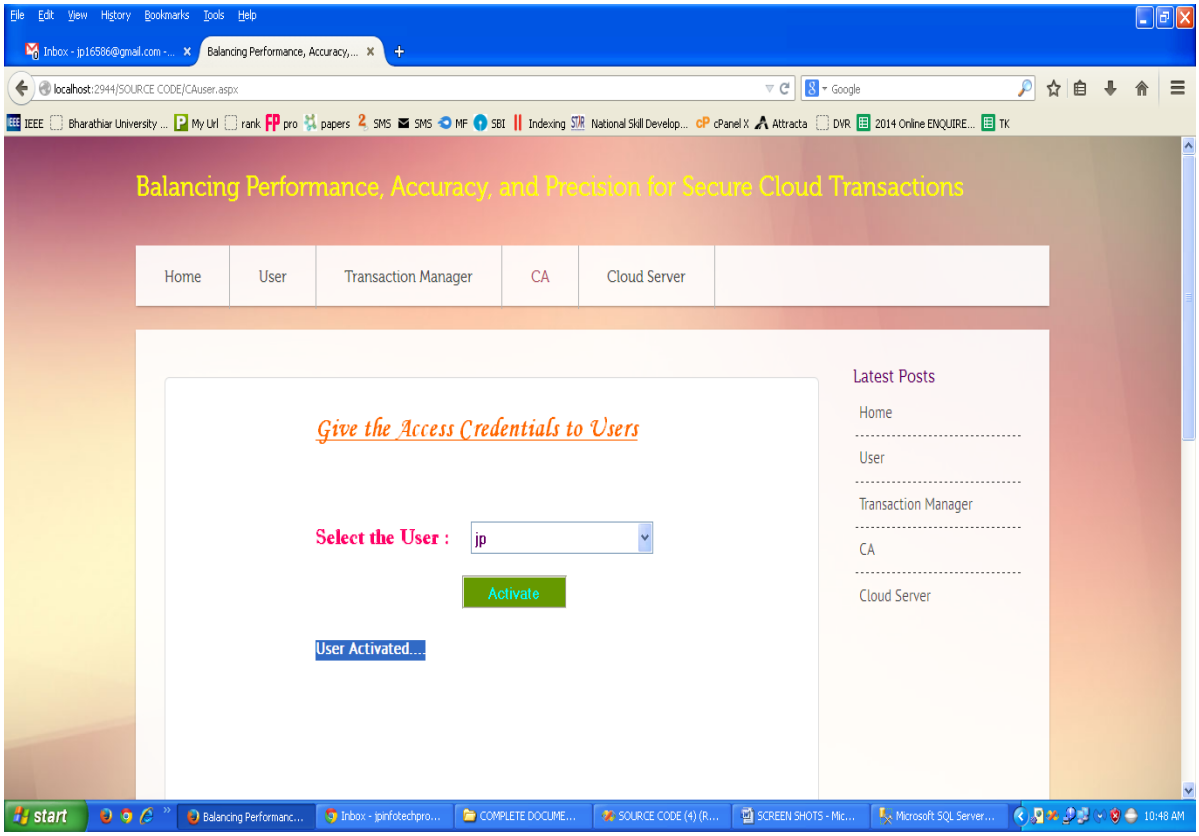
User LogIn



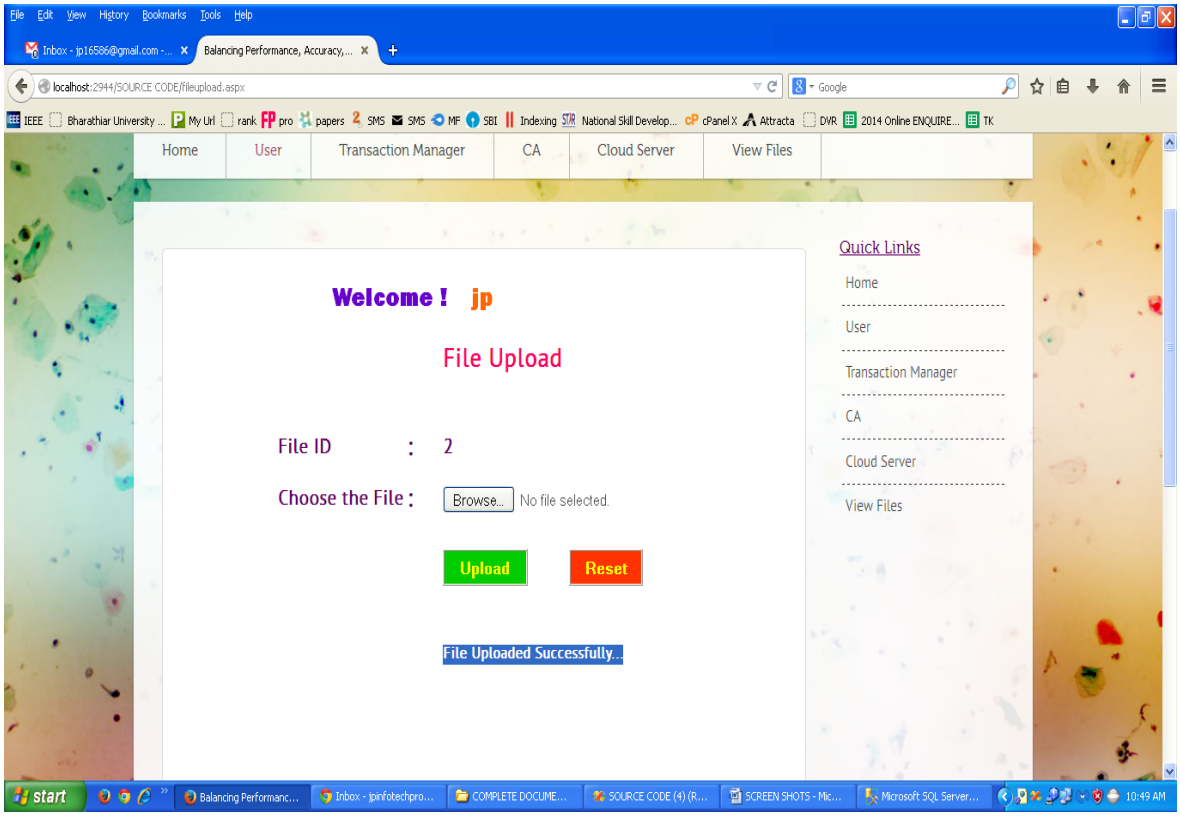
CA Login



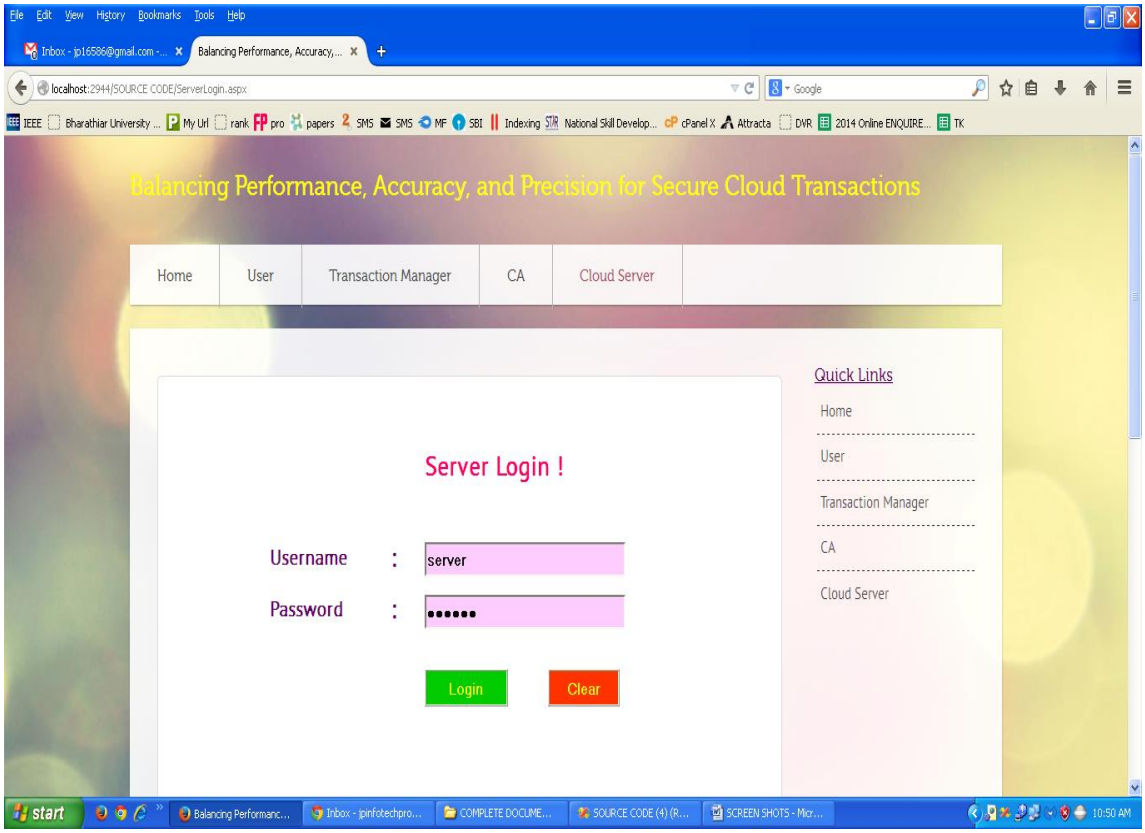
Access Credential



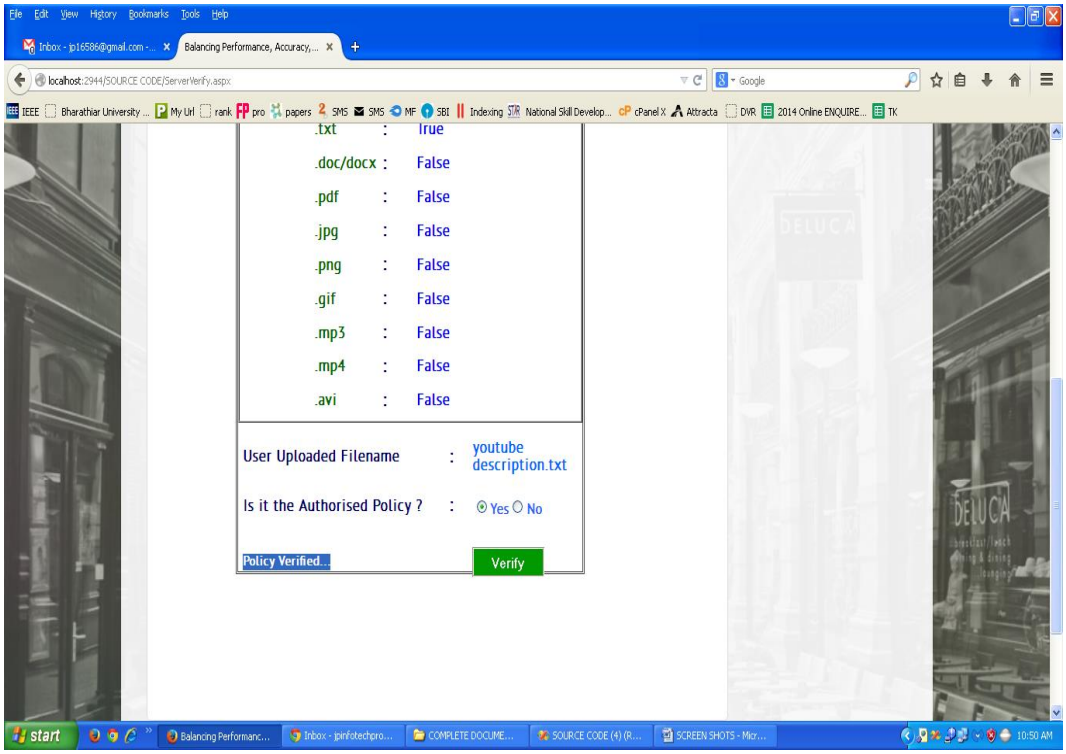
User Upload the File



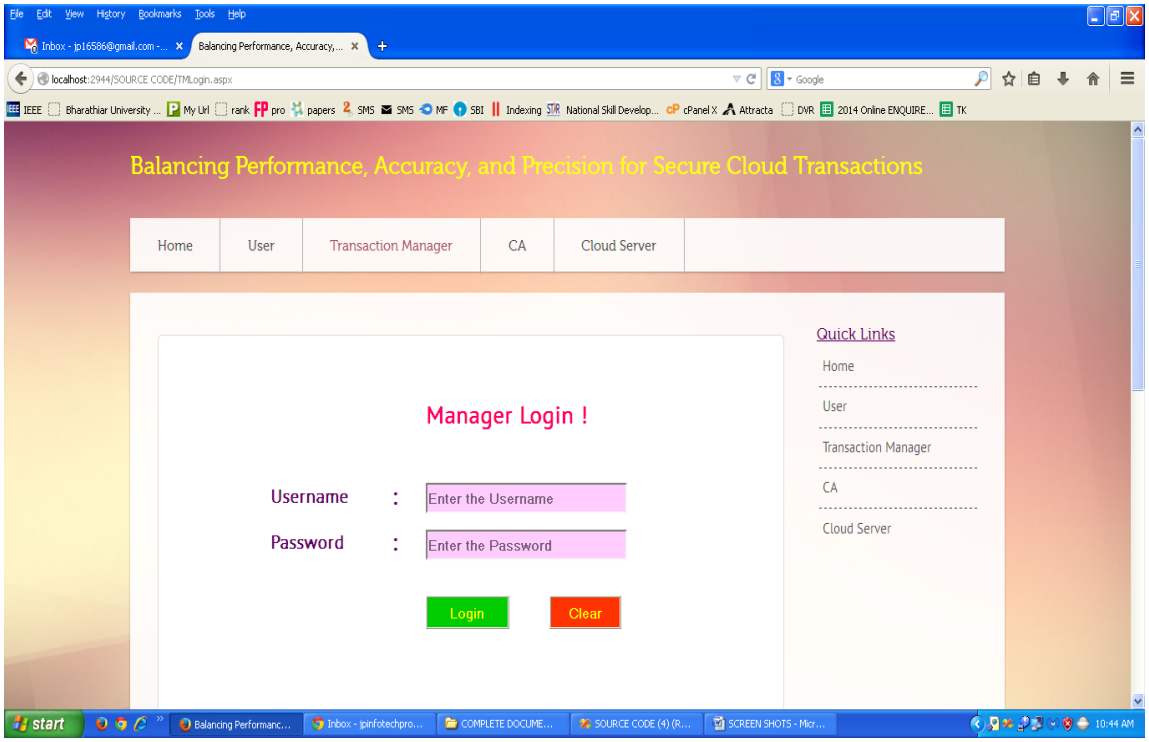
Server Login



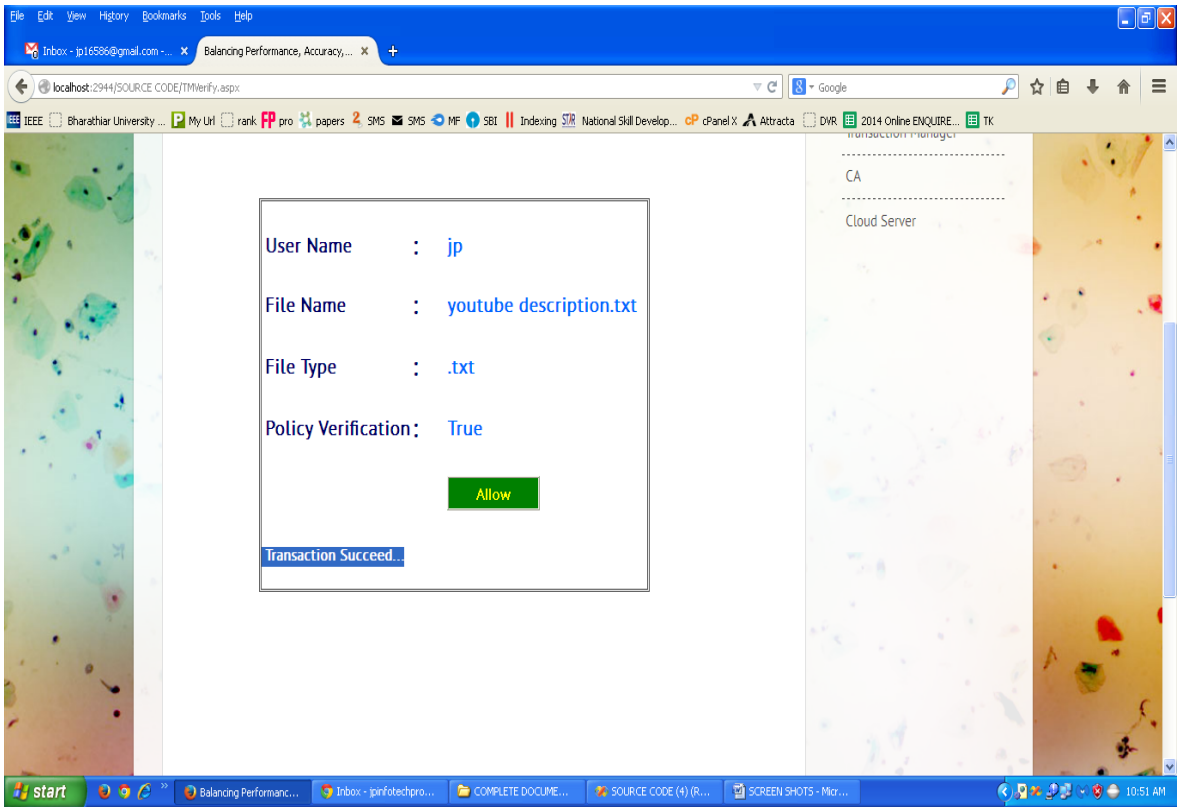
Policy Verify



Manager Login



Allow User Acess



7. CONCLUSION

Despite the popularity of cloud services and their wide adoption by enterprises and governments, cloud providers still lack services that guarantee both data and access control policy consistency across multiple data centers. In this paper, we identified several consistency problems that can arise during cloud-hosted transaction processing using weak consistency models, particularly if policy-based authorization systems are used to enforce access controls. To this end, we developed a variety of lightweight proof enforcement and consistency models—i.e., Deferred, Punctual, Incremental, and Continuous proofs, with view or global consistency—that can enforce increasingly strong protections with minimal runtime overheads. We used simulated workloads to experimentally evaluate implementations of our proposed consistency models relative to three core metrics: transaction processing performance, accuracy (i.e., global versus view consistency and regency of policies used), and precision (level of agreement among transaction participants). We found that high performance comes at a cost: Deferred and Punctual proofs had minimal overheads, but failed to detect certain types of consistency problems. On the other hand, high-accuracy models (i.e., Incremental and Continuous) required higher code complexity to implement correctly, and had only moderate performance when compared to the lower accuracy schemes. To better explore the differences between these approaches, we also carried out a tradeoff analysis of our schemes to illustrate how application-centric requirements influence the applicability of the eight protocol variants explored in this project.

8. REFERENCES

[1] M. Armbrust et al., “Above the Clouds: A Berkeley View of Cloud Computing,” technical report, Univ. of California, Feb. 2009.

[2] S. Das, D. Agrawal, and A.E. Abbadi, “Elastras: An Elastic Transactional Data Store in the Cloud,” Proc. Conf. Hot Topics in Cloud Computing (USENIX HotCloud ’09), 2009.

[3] D.J. Abadi, “Data Management in the Cloud: Limitations and Opportunities,” IEEE Data Eng. Bull., vol. 32, no. 1, pp. 3-12, Mar. 2009.

[4] A.J. Lee and M. Winslett, “Safety and Consistency in Policy-Based Authorization Systems,” Proc. 13th ACM Conf. Computer and Comm. Security (CCS ’06), 2006.

[5] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - Ocsp,” RFC 2560, http://tools.ietf.org/html/rfc5280, June 1999.

[6] E. Rissanen, “Extensible Access Control Markup Language (Xacml) Version 3.0,” http://docs.oasis-open.org/xacml/3.0/ xacml-3.0-core-spec-os-en.html, Jan. 2013.

[7] D. Cooper et al., “Internet x.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280, http://tools.ietf.org/html/rfc5280, May 2008.

[8] J. Li, N. Li, and W.H. Winsborough, “Automated Trust Negotiation Using Cryptographic Credentials,” Proc. 12th ACM Conf. Computer and Comm. Security (CCS ’05), Nov. 2005.

[9] L. Bauer et al., “Distributed Proving in Access-Control Systems,” Proc. IEEE Symp. Security and Privacy, May 2005.

[10] J. Li and N. Li, “OACerts: Oblivious Attribute Based Certificates,” IEEE Trans. Dependable and Secure Computing, vol. 3, no. 4, pp. 340-352, Oct.-Dec. 2006.

[11] J. Camenisch and A. Lysyanskaya, “An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation,” Proc. Int’l Conf. Theory and Application of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT ’01), 2001.

[12] P.K. Chrysanthis, G. Samaras, and Y.J. Al-Houmaily, “Recovery and Performance of Atomic Commit Processing in Distributed Database Systems,” Recovery Mechanisms in Database Systems, Prentice Hall PTR, 1998.

[13] M.K. Iskander, D.W. Wilkinson, A.J. Lee, and P.K. Chrysanthis, “Enforcing Policy and Data Consistency of Cloud Transactions,” Proc. IEEE Second Int’l Workshop Security and Privacy in Cloud Computing (ICDCS-SPCCICDCS-SPCC), 2011.

[14] G. DeCandia et al., “Dynamo: Amazons Highly Available Key-Value Store,” Proc. 21st ACM SIGOPS Symp. Operating Systems Principles (SOSP ’07), 2007.

[15] F. Chang et al., “Bigtable: A Distributed Storage System for Structured Data,” Proc. Seventh USENIX Symp. Operating System Design and Implementation (OSDI ’06), 2006.

[16] A. Lakshman and P. Malik, “Cassandra- A Decentralized Structured Storage System,” ACM SIGOPS Operating Systems Rev., vol. 44, pp. 35-40, Apr. 2010.

[17] B.F. Cooper et al., “PNUTS: Yahoo!’s Hosted Data Serving Platform,” Proc. VLDB Endowment, vol. 1, pp. 1277-1288, Aug. 2008.

[18] W. Vogels, “Eventually Consistent,” Comm. ACM, vol. 52, pp. 40-44, Jan. 2009.

[19] H. Guo, P.-A. Larson, R. Ramakrishnan, and J. Goldstein, “Relaxed Currency and Consistency: How to Say “Good Enough” in SQL,” Proc. ACM Int’l Conf. Management of Data (SIGMOD ’04), 2004.

[20] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, “Consistency Rationing in the Cloud: Pay Only When It Matters,” Proc. VLDB Endowment, vol. 2, pp. 253-264, Aug. 2009.