OPEN ACCESS

International
Journal
Of Modern Engineering Research (IJMER)

# Development of a Suitable Load Balancing Strategy In Case Of a Cloud Computing Architecture

Dayananda RB[1], Prof. Dr. G. Manoj Someswar[2]
[1](Associate Professor, Department of CSE, RRIT, Bangalore – 90, Karnataka, India)
[2](Professor & HOD, Department of CSE & IT, Nawab Shah Alam Khan College of Engineering & Technology, Affiliated to JNTU, Hyderabad, Malakpet, Hyderabad – 500024, India)

***Abstract:*** *Cloud computing is an attracting technology in the field of computer science. In Gartner's report, it says that the cloud will bring changes to the IT industry. The cloud is changing our life by providing users with new types of services. Users get service from a cloud without paying attention to the details. NIST gave a definition of cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. More and more people pay attention to cloud computing. Cloud computing is efficient and scalable but maintaining the stability of processing so many jobs in the cloud computing environment is a very complex problem with load balancing receiving much attention for researchers. Since the job arrival pattern is not predictable and the capacities of each node in the cloud differ, for load balancing problem, workload control is crucial to improve system performance and maintain stability. Load balancing schemes depending on whether the system dynamics are important can be either static or dynamic. Static schemes do not use the system information and are less complex while dynamic schemes will bring additional costs for the system but can change as the system status changes. A dynamic scheme is used here for its flexibility. The model has a main controller and balancers to gather and analyze the information. Thus, the dynamic control has little influence on the other working nodes. The system status then provides a basis for choosing the right load balancing strategy. The load balancing model given in this research article is aimed at the public cloud which has numerous nodes with distributed computing resources in many different geographic locations. Thus, this model divides the public cloud into several cloud partitions. When the environment is very large and complex, these divisions simplify the load balancing. The cloud has a main controller that chooses the suitable partitions for arriving jobs while the balancer for each cloud partition chooses the best load balancing strategy.*

***Keywords:*** *Load Balancing Strategy, User Interface Design, Requirements Traceability Matrix, Application Execution, Database Staging, Effective Network Performance*

## I. Introduction

The mysticism of cloud computing has worn off, leaving those required to implement cloud computing directives with that valley-of-despair feeling. When the hype is skimmed from cloud computing—private, public, or hybrid—what is left is a large, virtualized data center with IT control ranging from limited to non-existent. In private cloud deployments, IT maintains a modicum of control, but as with all architectural choices, that control is limited by the systems that comprise the cloud.

In a public cloud, not one stitch of cloud infrastructure is within the bounds of organizational control. Hybrid implementations, of course, suffer both of these limitations in different ways. But what cloud computing represents—the ability to shift loads rapidly across the Internet—is something large multi-national and even large intra-national organizations mastered long before the term "cloud" came along. While pundits like to refer to cloud computing as revolutionary, from the technologists' perspective, it is purely evolutionary. Cloud resources and cloud balancing extend the benefits of global application delivery to the smallest of organizations. In its most basic form, cloud balancing provides an organization with the ability to distribute application requests across any number of application deployments located in data centers and through cloud-computing providers.

Cloud balancing takes a broader view of application delivery and applies specified thresholds and service level agreements (SLAs) to every request. The use of cloud balancing can result in the majority of users being served by application deployments in the cloud providers' environments, even though the local application deployment or internal, private cloud might have more than enough capacity to serve that user. A variant of cloud balancing called cloud bursting, which sends excess traffic to cloud implementations, is also being implemented across the globe today. Cloud bursting delivers the benefits of cloud providers when usage is high, without the expense when organizational data centers—including internal cloud deployments—can handle the workload. In one vision of the future, the shifting of load is automated to enable organizations to configure clouds and cloud balancing and then turn their attention to other issues, trusting that the infrastructure will perform as designed.

## II. Existing System

Cloud computing is efficient and scalable but maintaining the stability of processing so many jobs in the cloud computing environment is a very complex problem with load balancing receiving much attention for researchers. Since the job arrival pattern is not predictable and the capacities of each node in the cloud differ, for load balancing problem, workload control is. crucial to improve system performance and maintain stability. Load balancing schemes depending on whether the system dynamics are important can be either static and dynamic. Static schemes do not use the system information and are less complex while dynamic schemes will bring additional costs for the system but can change as the system status changes. A dynamic scheme is used here for its flexibility.

**Disadvantages of Existing System**

Load balancing schemes depending on whether the system dynamics are important can be either static and dynamic. Static schemes do not use the system information and are less complex.

## III. Proposed System

Load balancing schemes depending on whether the system dynamics are important can be either static and dynamic. Static schemes do not use the system information and are less complex while dynamic schemes will bring additional costs for the system but can change as the system status changes. A dynamic scheme is used here for its flexibility. The model has a main controller and balancers to gather and analyze the information. Thus, the dynamic control has little influence on the other working nodes. The system status then provides a basis for choosing the right load balancing strategy.

The load balancing model given in this research article is aimed at the public cloud which has numerous nodes with distributed computing resources in many different geographic locations. Thus, this model divides the public cloud into several cloud partitions. When the environment is very large and complex, these divisions simplify the load balancing. The cloud has a main controller that chooses the suitable partitions for arriving jobs while the balancer for each cloud partition chooses the best load balancing strategy.

**Advantages of Proposed System**

Load balancing schemes depending on whether the system dynamics are important can be either static and dynamic . Static scheme does use the system information and are less complex.

## IV. The Study Of The System

To conduct studies and analyses of an operational and technological nature, and to promote the exchange and development of methods and tools for operational analysis as applied to defense problems.

**Logical design**

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system.

**Physical design**

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified / authenticated, how it is processed, and how it is displayed as output. In Physical design, following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing Requirements,

5. System control and backup or recovery.

Put another way, the physical portion of systems design can generally be broken down into three sub-tasks:
1. User Interface Design
2. Data Design
3. Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system. Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the systems design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase. Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

## V. Input & Output Representation

**Input Design**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:
- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

**Objectives**

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

**Output Design**

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

a. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
b. Select methods for presenting information.
c. Create document, report, or other formats that contain information produced by the system.

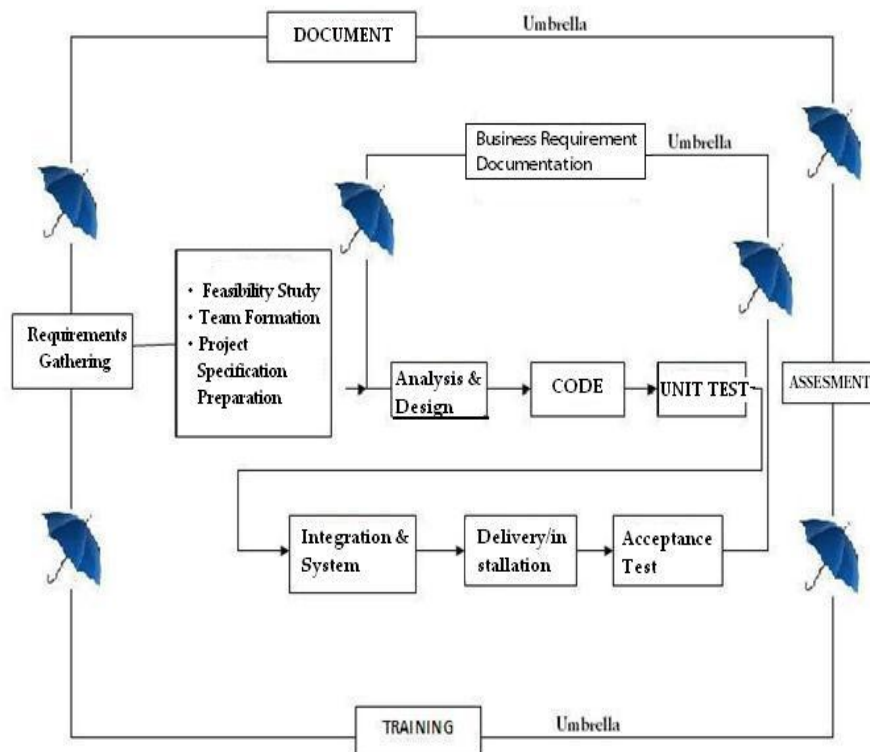
The output form of an information system should accomplish one or more of the following objectives.
- Convey information about past activities, current status or projections of the future.

- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

## VI. Process Model Used With Justification

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

**SDLC (Umbrella Model):**

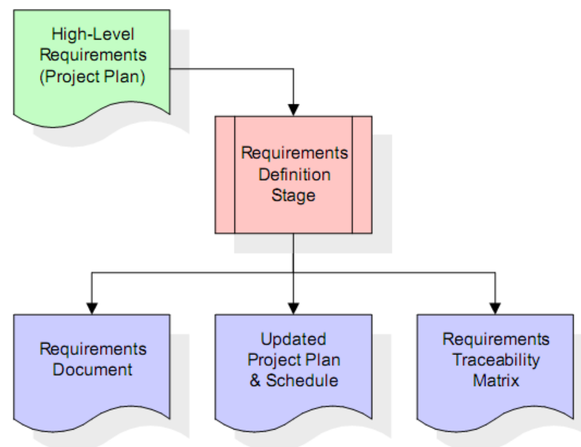



**Stages of SDLC:**

- Requirement Gathering and Analysis
- Designing
- Coding
- Testing
- Deployment

**Requirements Definition Stage and Analysis**

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.

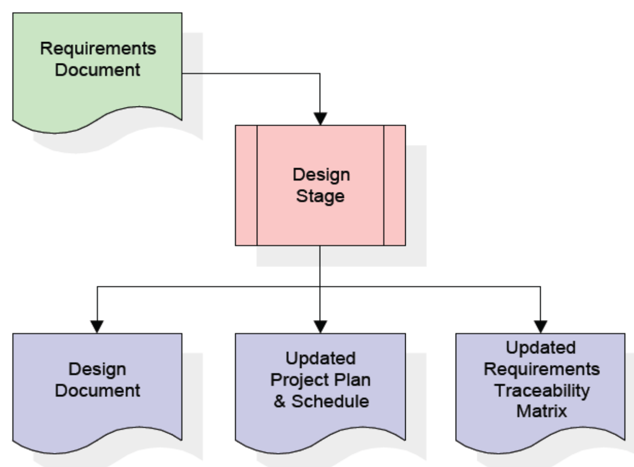



These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). the requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document. The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term *requirements traceability*. The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

**Design Stage**

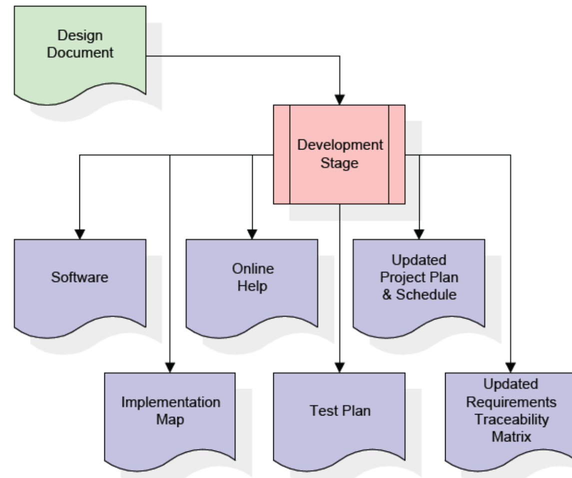
The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.




When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

**Development Stage**

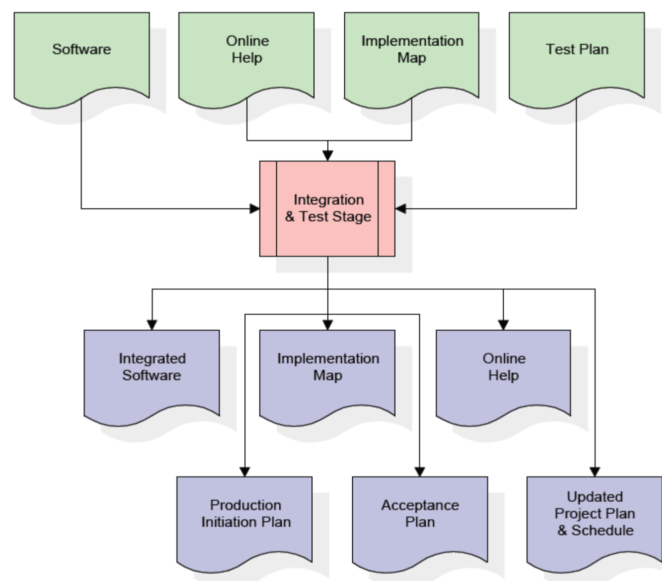
The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.




The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

**Integration & Test Stage**

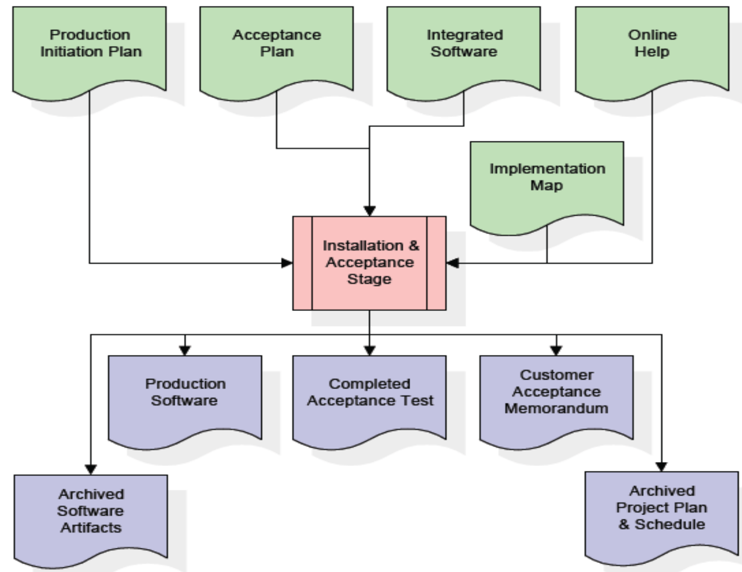
During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled intothe Production Initiation Plan.



The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

**Installation & Acceptance Stage**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer. After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

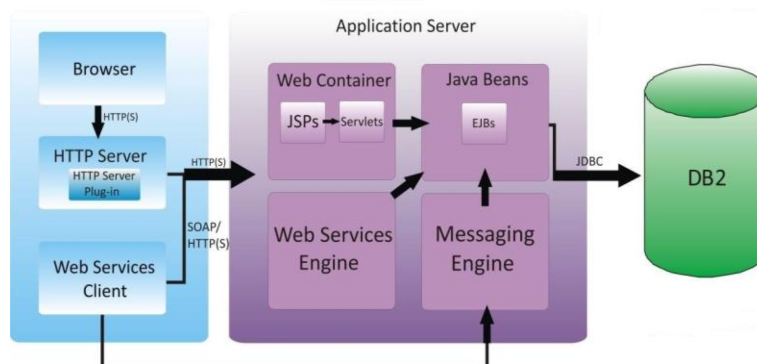



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

## VII. System Architecture

**Architecture Flow**

Below architecture diagram represents mainly flow of request from the users to database through servers. In this scenario overall system is designed in three tiers separately using three layers called presentation layer, business layer, data link layer. This project was developed using 3-tier architecture.



**3-Tier Architecture**

The three-tier software architecture (a three layer architecture) emerged in the 1990s to overcome the limitations of the two-tier architecture. The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging.

The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems

**Advantages of Three-Tier**

- Separates functionality from presentation.
- Clear separation - better understanding.
- Changes limited to well define components.
- Can be running on WWW.
- Effective network performance**.**

**System Design Introduction:**

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

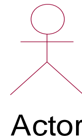
## VIII. UML Diagrams

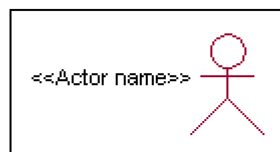
**1. Global Use Case Diagrams:**

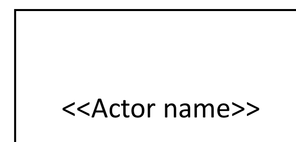
**Identification of actors:**

**Actor:** Actor represents the role a user plays with respect to the system. An actor interacts with, but has no control over the use cases.

**Graphical representation:**




An actor is someone or something that:

Interacts with or uses the system.

- Provides input to and receives information from the system.
- Is external to the system and has no control over the use cases.

Actors are discovered by examining:

- Who directly uses the system?
- Who is responsible for maintaining the system?
- External hardware used by the system.
- Other systems that need to interact with the system.

Questions to identify actors:

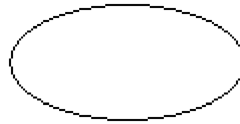
- Who is using the system? Or, who is affected by the system? Or, which groups need help from the system to perform a task?
- Who affects the system? Or, which user groups are needed by the system to perform its functions? These functions can be both main functions and secondary functions such as administration.
- Which external hardware or systems (if any) use the system to perform tasks?
- What problems does this application solve (that is, for whom)?
- And, finally, how do users use the system (use case)? What are they doing with the system?

The actors identified in this system are:

a. **System Administrator**
b. **Customer**
c. **Customer Care**

**Identification of use cases:**
**Usecase:** A use case can be described as a specific way of using the system from a user's (actor's) perspective.

## IX. Graphical Representation



A more detailed description might characterize a use case as:
- Pattern of behavior the system exhibits
- A sequence of related transactions performed by an actor and the system
- Delivering something of value to the actor

Use cases provide a means to:
- capture system requirements
- communicate with the end users and domain experts
- test the system

Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system.
Guide lines for identifying use cases:
- For each actor, find the tasks and functions that the actor should be able to perform or that the system needs the actor to perform. The use case should represent a course of events that leads to clear goal
- Name the use cases.
- Describe the use cases briefly by applying terms with which the user is familiar.

This makes the description less ambiguous
Questions to identify use cases:
- What are the tasks of each actor?
- Will any actor create, store, change, remove or read information in the system?
- What use case will store, change, remove or read this information?
- Will any actor need to inform the system about sudden external changes?
- Does any actor need to inform about certain occurrences in the system?
- What use cases will support and maintains the system?

**Flow of Events**
A flow of events is a sequence of transactions (or events) performed by the system. They typically contain very detailed information, written in terms of what the system should do, not how the system accomplishes the task. Flow of events are created as separate files or documents in your favorite text editor and then attached or linked to a use case using the Files tab of a model element.
A flow of events should include:
- When and how the use case starts and ends
- Use case/actor interactions
- Data needed by the use case
- Normal sequence of events for the use case
- Alternate or exceptional flows

**Construction of Use case diagrams:**
Use-case diagrams graphically depict system behavior (use cases). These diagrams present a high level view of how the system is used as viewed from an outsider's (actor's) perspective. A use-case diagram may depict all or some of the use cases of a system.
A use-case diagram can contain:
- actors ("things" outside the system)
- use cases (system boundaries identifying what the system should do)
- Interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations.

**Relationships in use cases:**

**1. Communication:**

The communication relationship of an actor in a use case is shown by connecting the actor symbol to the use case symbol with a solid path. The actor is said to communicate with the use case.

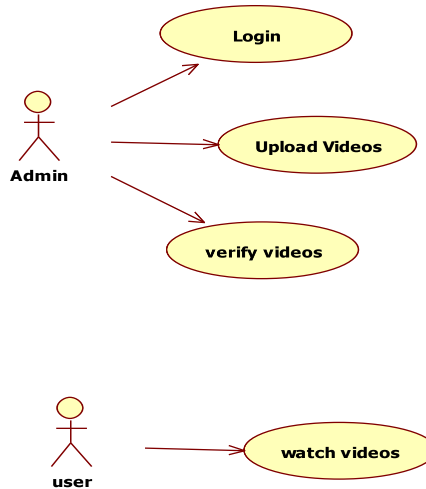
**2. Uses:**

A Uses relationship between the use cases is shown by generalization arrow from the use case.

**3. Extends:**

The extend relationship is used when we have one use case that is similar to another use case but does a bit more. In essence it is like subclass.

**Main Use Case Diagram in our system:**



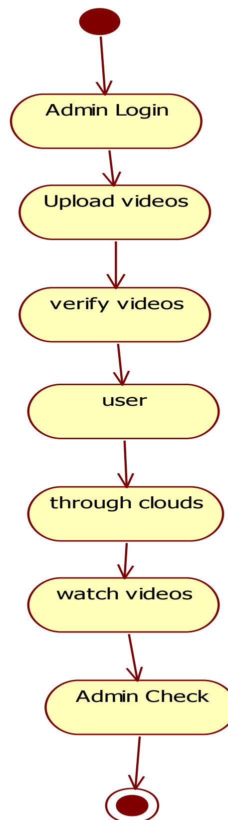

## X. Activity Diagram

Activity diagrams provide a way to model the workflow of a business process, code-specific information such as a class operation. The transitions are implicitly triggered by completion of the actions in the source activities. The main difference between activity diagrams and state charts is activity diagrams are activity centric, while state charts are state centric. An activity diagram is typically used for modeling the sequence of activities in a process, whereas a state chart is better suited to model the discrete stages of an object's lifetime. An activity represents the performance of task or duty in a workflow. It may also represent the execution of a statement in a procedure. You can share activities between state machines. However, transitions cannot be shared. An action is described as a "task" that takes place while inside a state or activity.
Actions on activities can occur at one of four times:

- **on entry** ---- The "task" must be performed when the object enters the state or activity.
- **on exit** ---- The "task" must be performed when the object exits the state or activity.
- **do** ---- The "task" must be performed while in the state or activity and must continue until exiting the state.
- **on event** ---- The "task" triggers an action only if a specific event is received.
- An **end state** represents a final or terminal state on an activity diagram or state chart diagram.
- A **start state** (also called an "initial state") explicitly shows the beginning of a workflow on an activity diagram.
- **Swim lanes** can represent organizational units or roles within a business model. They are very similar to an object. They are used to determine which unit is responsible for carrying out the specific activity. They show ownership or responsibility. Transitions cross swim lanes
- **Synchronizations** enable you to see a simultaneous workflow in an activity diagram Synchronizations visually define forks and joins representing parallel workflow.
- A **fork** construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. A corresponding join should ideally accompany every fork that appears on an activity diagram. A **join** consists of two of more flows of control that unite into a single flow of control.

All model elements (such as activities and states) that appear between a fork and join must complete before the flow of controls can unite into one.

- An **object flow** on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input).




## XI. Sequence Diagrams

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

**Object:**

An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance. The object icon is similar to a class icon except that the name is underlined:An object's concurrency is defined by the concurrency of its class.

**Message:**

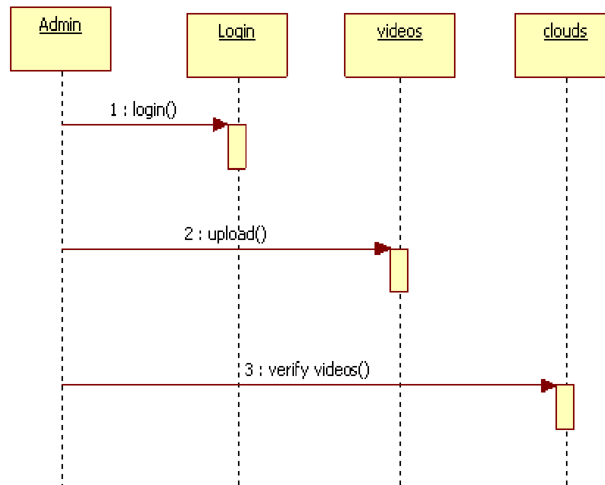
A message is the communication carried between two objects that trigger an event. A message carries information from the source focus of control to the destination focus of control. The synchronization of a message can be modified through the message specification. Synchronization means a message where the sending object pauses to wait for results.

**Link:**

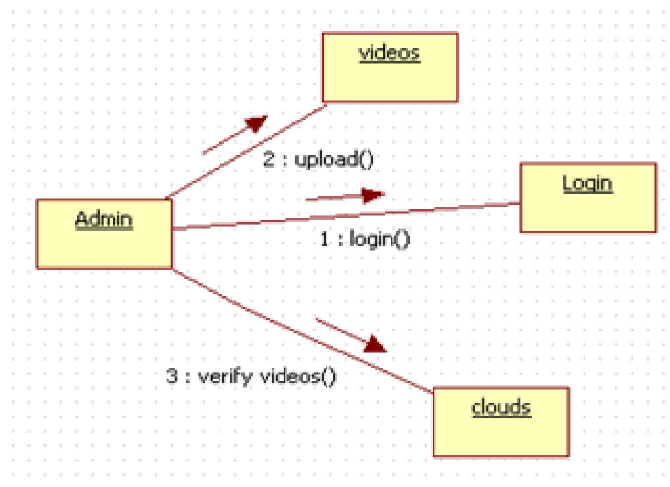
A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes. The existence of a relationship between two classes symbolizes a path of communication between instances of the classes: one object may send messages to another. The link is depicted

as a straight line between objects or objects and class instances in a collaboration diagram. If an object links to itself, use the loop version of the icon.

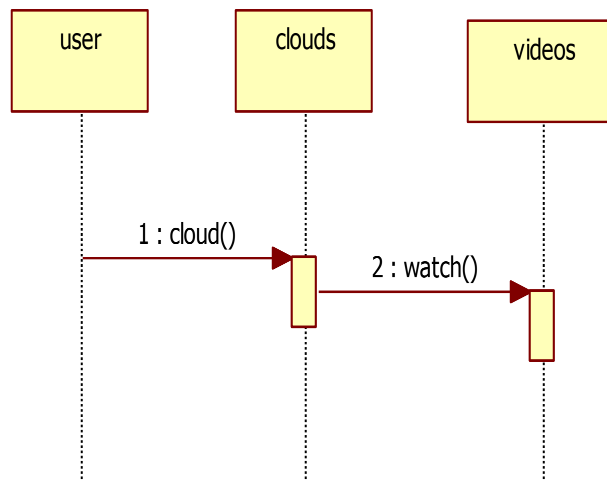
**Admin Sequence:**



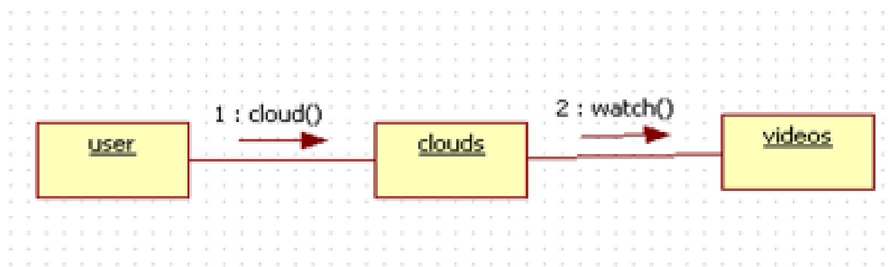

**Admin Collaboration:**




**User Sequence:**



**User Collaboration:**




## XII. Class Diagram

**Identification of analysis classes:**

A class is a set of objects that share a common structure and common behavior (the same attributes, operations, relationships and semantics). A class is an abstraction of real-world items.

There are 4 approaches for identifying classes:

a. Noun phrase approach:
b. Common class pattern approach.
c. Use case Driven Sequence or Collaboration approach.
d. Classes , Responsibilities and collaborators Approach

**1. Noun Phrase Approach:**

The guidelines for identifying the classes:

- Look for nouns and noun phrases in the use cases.
- Some classes are implicit or taken from general knowledge.
- All classes must make sense in the application domain; Avoid computer
- Implementation classes – defer them to the design stage.
- Carefully choose and define the class names After identifying the classes we have to eliminate the following types of classes:
- Adjective classes.

**2. Common class pattern approach:**

The following are the patterns for finding the candidate classes:

- Concept class.
- Events class.
- Organization class
- Peoples class
- Places class
- Tangible things and devices class.

**3. Use case driven approach:**

We have to draw the sequence diagram or collaboration diagram. If there is need for some classes to represent some functionality then add new classes which perform those functionalities.

## XIII. CRC Approach

**The process consists of the following steps:**

- Identify classes' responsibilities ( and identify the classes )
- Assign the responsibilities
- Identify the collaborators.

**Identification of responsibilities of each class:**

The questions that should be answered to identify the attributes and methods of a class respectively are:

a. What information about an object should we keep track of?
b. What services must a class provide?

**Identification of relationships among the classes:**
Three types of relationships among the objects are:
Association: How objects are associated?
Super-sub structure: How are objects organized into super classes and sub classes?
Aggregation: What is the composition of the complex classes?

**Association:**

The **questions** that will help us to identify the associations are:

a. Is the class capable of fulfilling the required task by itself?
b. If not, what does it need?
c. From what other classes can it acquire what it needs?

**Guidelines for identifying the tentative associations:**

- A dependency between two or more classes may be an association. Association often corresponds to a verb or prepositional phrase.
- A reference from one class to another is an association. Some associations are implicit or taken from general knowledge.

**Some common association patterns are:**
Location association like part of, next to, contained in…..
Communication association like talk to, order to ……

We have to eliminate the unnecessary association like implementation associations, ternary or n-ary associations and derived associations.

**Super-sub class relationships:**

Super-sub class hierarchy is a relationship between classes where one class is the parent class of another class (derived class).This is based on inheritance.
Guidelines for identifying the super-sub relationship, a generalization are:

1**. Top-down***:*

Look for noun phrases composed of various adjectives in a class name. Avoid excessive refinement. Specialize only when the sub classes have significant behavior.

2. **Bottom-up***:*
Look for classes with similar attributes or methods. Group them by moving the common attributes and methods to an abstract class. You may have to alter the definitions a bit.

3. **Reusability***:*
Move the attributes and methods as high as possible in the hierarchy.

4. **Multiple inheritances***:*
Avoid excessive use of multiple inheritances. One way of getting benefits of multiple inheritances is to inherit from the most appropriate class and add an object of another class as an attribute.

**Aggregation or a-part-of relationship:**

It represents the situation where a class consists of several component classes. A class that is composed of other classes doesn't behave like its parts. It behaves very difficultly. The major properties of this relationship are transitivity and anti symmetry.

The **questions** whose answers will determine the distinction between the part and whole relationships are:

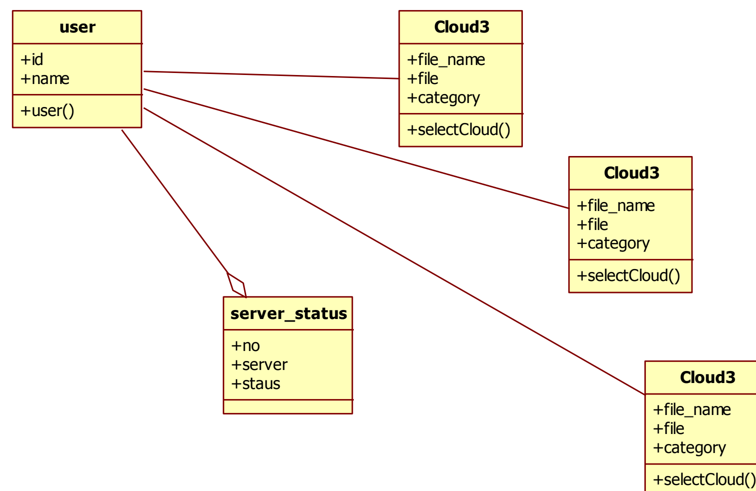
- Does the part class belong to the problem domain?
- Is the part class within the system's responsibilities?
- Does the part class capture more than a single value?( If not then simply include it as an attribute of the whole class)
- Does it provide a useful abstraction in dealing with the problem domain?

There are three types of aggregation relationships. They are:

**Assembly:**
It is constructed from its parts and an assembly-part situation physically exists.

**Container:**
A physical whole encompasses but is not constructed from physical parts.

**Collection member:**
A conceptual whole encompasses parts that may be physical or conceptual. The container and collection are represented by hollow diamonds but composition is represented by solid diamond.




## XIV. Conclusion

Till now we have discussed on basic concepts of Cloud Computing and Load balancing. In addition to that, the load balancing technique that is based on Swarm intelligence has been discussed. We have discussed how the mobile agents can balance the load of a cloud using the concept of Ant colony Optimization. The limitation of this technique is that it will be more efficient if we form cluster in our cloud. So, the research work can be proceeded to implement the total solution of load balancing in a complete cloud environment. Our objective for this paper is to develop an effective load balancing algorithm using Ant colony optimization technique to maximize or minimize different performance parameters like CPU load, Memory capacity, Delay or network load for the clouds of different sizes.

## REFERENCES

[1] R. Hunter, The why of cloud, http://www.gartner.com/ DisplayDocument?doc cd=226469&ref= g noreg, 2012.
[2] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, Cloud computing: Distributed internet computing for IT and scientific research, Internet Computing, vol.13, no.5, pp.10-13, Sept.-Oct. 2009.
[3] P. Mell and T. Grance, The NIST definition of cloud computing, http://csrc.nist.gov/ publications/nistpubs/800-145/SP800-145.pdf, 2012.
[4] Microsoft Academic Research, Cloud computing, http:// libra.msra.cn/Keyword/6051/cloud-computing?query= cloud%20computing, 2012.
[5] Google Trends, Cloud computing, http://www.google. com/trends/explore#q=cloud%20computing, 2012.
[6] N. G. Shivaratri, P. Krueger, and M. Singhal, Load distributing for locally distributed systems, Computer, vol. 25, no. 12, pp. 33-44, Dec. 1992.
[7] B. Adler, Load balancing in the cloud: Tools, tips and techniques, http://www.rightscale. com/info center/white-papers/Load-Balancing-in-the-Cloud.pdf, 2012.