ISSN 2277-2685
IJESR/Jan-Mar. 2026/ Vol-16/Issue-1/315-324
G Shivamani *et. al.*, /International Journal of Engineering & Science Research

Full Length Research Article

Hybrid Genetic Algorithm-Based QoS-Aware Task Scheduling In Cloud Environments

Dr.M.Gokilavani[1],G Shivamani[2],K Shiva Prasadh[3],M Lokesh Kumar[4]
[1]Associate Professor ; Department Of Information Technology, Guru Nanak Institutions Technical Campus, Hyderabad, India.
[2,3,4]B.Tech Students; Department Of Information Technology, Guru Nanak Institutions Technical Campus, Hyderabad, India.
shivaprasad3142@gmail.com[3]

Accepted 22-03-2026

Abstract
Enhancing multiple Quality of Service (QoS) parameters simultaneously remains a significant challenge in cloud computing (CC), particularly when the delivered QoS fails to meet end-user expectations. This study introduces an improved Smart Message Passing Interface Approach (SMPIA) integrated with two optimization techniques: Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). The proposed hybrid models, namely GA-SMPIA and PSO-SMPIA, are designed for efficient task scheduling and resource allocation in cloud environments. The primary objective is to minimize makespan and total execution time while maximizing resource utilization. A key contribution of this research is the formulation of a novel multi-objective cost function that determines the maximum cost associated with each transaction flow—an aspect not sufficiently addressed in earlier studies. This function incorporates multiple parameters, including flow load, makespan load, virtual machine (VM) capacity, and execution speed.Additionally, telecommunication transactions are categorized based on flow types, and an allocation matrix is constructed to map transaction flows to appropriate VMs. This matrix guides the routing of transactions to optimize performance. The study also evaluates how makespan and execution time influence overall resource utilization. Experimental results demonstrate that PSO-SMPIA achieves superior performance in terms of average resource utilization compared to existing methods such as SMPIA, Fuzzy SMPIA (FSMPIA), Optimized SMPIA (O-SMPIA), and GA-SMPIA. However, FSMPIA and O-SMPIA show better results in minimizing makespan and execution time individually. Notably, GA-SMPIA provides a balanced improvement across all key metrics by effectively reducing execution time, minimizing makespan, and enhancing resource utilization.
Overall, the proposed approaches significantly improve QoS delivery in cloud computing environments, making them suitable for real-world applications requiring efficient and reliable resource management.

Keywords: *Cloud Computing, Quality of Service (QoS), Task Scheduling, Resource Allocation, SMPIA, Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Makespan, Execution Time, Resource Utilization*

INTRODUCTION

Cloud computing has emerged as a dominant paradigm for delivering computing services due to its flexibility, scalability, and cost-effectiveness. By providing on-demand access to shared computing resources—including servers, storage, networks, and applications—cloud computing enables users to deploy and execute applications without managing physical infrastructure. However, the growing number of cloud users and the increasing complexity of applications have introduced significant challenges in resource management, task scheduling, and Quality of Service (QoS) assurance. Efficient task scheduling is crucial in cloud environments, as suboptimal allocation of tasks to virtual machines (VMs) can result in increased execution time, higher makespan, underutilized resources, SLA violations, and reduced user satisfaction. Cloud service providers must simultaneously balance multiple, often conflicting, factors such as execution time, makespan, operational cost, resource utilization, VM capacity, and workload distribution. Traditional scheduling techniques, such as First Come First Serve (FCFS), Round Robin, and basic heuristics, are generally single-objective and fail to adapt effectively to dynamic workloads and heterogeneous cloud infrastructures. Consequently, inefficiencies such as load imbalance, processing delays, and elevated operational costs arise, ultimately degrading QoS.

To address these limitations, metaheuristic optimization techniques have gained significant attention for their ability to explore large solution spaces and provide near-optimal solutions efficiently. Among these, Genetic Algorithm (GA)

and Particle Swarm Optimization (PSO) are widely applied in cloud task scheduling and resource allocation. GA, inspired by natural selection, explores diverse scheduling solutions through operations such as selection, crossover, and mutation. PSO, modeled on the collective behavior of bird flocks or fish schools, efficiently guides candidate solutions toward global and local optima.

Despite their effectiveness, both GA and PSO have inherent limitations when applied independently, including slow convergence or premature stagnation. To overcome these challenges, this study proposes a hybrid optimization framework that integrates GA and PSO with a Smart Message Passing Interface Approach (SMPIA) to achieve cost-aware and QoS-aware task scheduling in cloud environments. The framework aims to minimize makespan and total execution time while maximizing resource utilization and ensuring SLA compliance.

A key contribution of this work is the introduction of a multi-objective cost function that calculates the maximum allowable cost for each transaction flow—a factor not adequately addressed in previous research. The cost function incorporates flow load, makespan load, VM capacity, and execution speed. Transaction flows, particularly in cloud-based telecommunication environments, are classified based on their characteristics, and a transaction flow allocation matrix is generated to optimally assign tasks to VMs.

The framework also incorporates a Priority-Based Fair Scheduling (PBFS) algorithm to ensure equitable resource distribution and prevent task starvation. Combined with the hybrid GA–PSO SMPIA framework, PBFS achieves balanced workload distribution and improved QoS even under varying loads. Additionally, dynamic job migration allows tasks to be reassigned if a selected VM becomes overloaded or unsuitable, ensuring reliability and fault tolerance.

In summary, the proposed framework provides an intelligent, hybrid metaheuristic solution for cloud task scheduling. By simultaneously optimizing multiple QoS parameters while considering cost, fairness, and security, the system enhances cloud performance and user satisfaction compared to traditional approaches.

Scope of the Project

This project focuses on developing a QoS-aware and cost-aware cloud task scheduling framework based on hybrid optimization techniques. The scope includes:

- Designing intelligent scheduling modules that integrate GA, PSO, and SMPIA for resource allocation.
- Classifying transaction flows based on workload characteristics and applying Priority-Based Fair Scheduling (PBFS).
- Computing maximum cost for each transaction flow while optimizing makespan, execution time, VM capacity, and resource utilization.
- Incorporating dynamic load monitoring and VM capacity analysis for adaptive scheduling decisions.
- Implementing job migration to handle VM overload and ensure uninterrupted task execution.
- Providing secure user authentication and controlled access to cloud resources.
- Generating transaction flow allocation matrices to map tasks to suitable VMs.
- Collecting performance metrics for evaluation and comparative analysis with existing scheduling approaches.
- Supporting simulation and prototype-level implementation using web-based technologies.

The project does **not** cover large-scale production deployment or integration with real-time cloud service providers.

Objectives

The objectives of this project are as follows:

1. Design an intelligent task scheduling and resource management framework for cloud computing that ensures high QoS and cost efficiency.
2. Minimize makespan and total execution time while maximizing VM resource utilization.
3. Integrate GA and PSO with SMPIA for effective multi-objective optimization.
4. Compute the maximum cost for each transaction flow to support cost-aware scheduling.
5. Classify transaction flows to enable priority-based allocation.
6. Implement PBFS to ensure fairness and prevent task starvation.
7. Evaluate dynamic VM capacity to avoid overload conditions.
8. Support job migration to enhance reliability and fault tolerance.
9. Improve SLA compliance under varying workloads.
10. Ensure secure access and controlled execution of cloud tasks.
11. Record performance metrics (execution time, makespan, resource utilization) for evaluation.
12. Conduct comparative analysis with existing techniques to validate improvements.

Overall, the framework aims to enhance cloud service performance and user satisfaction.

Problem Statement

The rapid growth of cloud computing has made efficient task scheduling and resource management increasingly challenging in dynamic and heterogeneous cloud environments. Traditional scheduling mechanisms often optimize a single parameter, leading to:

- Poor VM utilization, higher operational costs, and SLA violations.

- Lack of intelligent workload classification and flow-based task management.
- Inefficient handling of variable and high-volume workloads.
- Absence of priority-aware scheduling, leading to task starvation.
- Slow convergence or premature stagnation when GA or PSO is applied independently.
- Difficulty balancing QoS with cost efficiency, particularly in transaction-heavy domains like telecommunications.

These limitations highlight the need for a multi-objective, intelligent, cost-aware scheduling framework that integrates hybrid optimization techniques, fair scheduling, and adaptive resource allocation.

Existing System

Current cloud systems dynamically allocate resources based on predicted application demand. These systems typically focus on workload and performance forecasting at both VM and physical machine levels. PSO is widely used for resource optimization, simulating particle behaviors to find optimal allocations. Although the existing system improves resource utilization, it primarily emphasizes performance optimization and does not adequately address security, fairness, or multi-objective QoS considerations.

Literature Survey

SG-PBFS (2024): A Shortest Gap–Priority Based Fair Scheduling technique that minimizes waiting time, ensures fairness, and improves resource utilization. Limitations include computational overhead and scalability concerns.

Priority-Based Gap Scheduling (2024): Assigns priorities and fills resource gaps for efficient task allocation. Reduces waiting time and improves QoS but may face increased complexity in dynamic and large-scale environments.

MTD-DHJS (2023): Makespan-optimized scheduling with dynamic computational time prediction. Reduces makespan and improves throughput but depends on accurate time prediction.

Grey Wolf Optimizer (2023): Multi-objective task scheduling for cloud-fog environments optimizing execution time, energy, and resource utilization. Increased computational complexity is a limitation.

Whale Optimization (2023): Workflow scheduling to minimize execution time and enhance resource utilization, with potential scalability challenges.

Optimized SMPIA (2022): Addresses task starvation and improves resource utilization with adaptive priority management, though with additional computational overhead.

Proposed System

The proposed system introduces a multi-objective, QoS-aware, and cost-aware framework for cloud resource management. Key features include:

- Calculation of maximum allowable cost per transaction flow for financial control.
- Hybrid GA–PSO integration with SMPIA for efficient scheduling.
- PBFS for fairness among tasks while respecting priorities.
- Secure cross-domain authentication and key agreement using digital certificates.
- Smart certificate re-issuance to prevent insecure blockchain certificate reuse.

PROJECT DESCRIPTION

This project presents a cloud-based task scheduling system designed to manage user requests, file uploads, and resource allocation efficiently within a structured web application framework. The system workflow begins with user registration and login interfaces, allowing authorized clients to access cloud services after approval by the administrator. The admin dashboard provides functionality for monitoring server and virtual machine (VM) status, managing user requests, and approving new registrations.

REQUIREMENTS ENGINEERING

The proposed system is designed as a **QoS-aware and cost-aware cloud task scheduling framework** that efficiently manages resources in dynamic cloud environments. Functionally, it enables users to submit tasks, which are analyzed based on workload size, execution requirements, and transaction flow characteristics. Scheduling parameters such as execution time, makespan, virtual machine (VM) capacity, and cost are computed to optimize task allocation. To achieve efficient task scheduling, a **hybrid optimization approach** is employed, integrating **Genetic Algorithm (GA)**, **Particle Swarm Optimization (PSO)**, and **Smart Message Passing Interface Approach (SMPIA)**. Task execution is further controlled using **Priority-Based Fair Scheduling (PBFS)**, which ensures equitable distribution of computational resources among multiple users. The system continuously monitors server load and triggers **job migration** when a VM becomes overloaded, ensuring uninterrupted task execution. Administrators can monitor user activities and resource utilization, while the framework guarantees improved QoS, minimized execution time, and optimized resource utilization. The system is designed to be scalable, supporting multiple concurrent users and workloads while maintaining reliable performance under dynamic cloud conditions.

The **hardware requirements** for developing and deploying the system include an Intel Core i3 processor or higher, 4 GB of DDR4 RAM, a 15.6-

inch LED monitor, and a minimum of 100 GB hard disk space. Additional peripherals such as a keyboard, mouse, and webcam are required for features like facial recognition or QR authentication. These specifications ensure the system operates efficiently during development, testing, and simulation. On the **software side**, the front-end is developed using Java EE technologies (JSP, Servlets) with Maven for build management, while the back-end uses MySQL 5.5 or higher. The system runs on Windows 10 or 11 and is deployed on Apache Tomcat 9.0, with support for web browsers such as Google Chrome and Mozilla Firefox. Development and maintenance are facilitated using IDEs like Eclipse or IntelliJ IDEA, providing an integrated environment for coding, debugging, and testing.

Priority-Based Fair Scheduling, the **JSP Dashboard**, and the **MySQL Database**. PBFS manages and prioritizes user access requests to prevent starvation and ensure fair handling of tasks. Each access request is stored in the database with its submission time and initial priority, and higher-priority requests are processed first while lower-priority requests are gradually promoted to maintain fairness. The JSP Dashboard acts as a secure web interface, allowing users and administrators to view uploaded files, transaction metadata, and the real-time status of approvals and task execution. It enables task submission, file search, and monitoring of policy compliance. The MySQL database supports these operations by storing user credentials, registration details, task metadata, and non-critical system logs. By separating critical data from auxiliary information, the database ensures both performance and data security, allowing quick retrieval of records for scheduling decisions and dashboard visualization.

Overall, the proposed system integrates advanced hybrid optimization techniques with fair scheduling and adaptive resource allocation to provide an intelligent, secure, and high-performance cloud task scheduling framework.

DESIGN ENGINEERING AND SYSTEM ARCHITECTURE

Design engineering provides a structured representation of the system by modelling its architecture, behavior, components, and interactions using UML (Unified Modeling Language) diagrams. These design artifacts serve as blueprints for implementation, testing, and maintenance, enabling developers and stakeholders to visualize system structure, functionality, and dynamics.

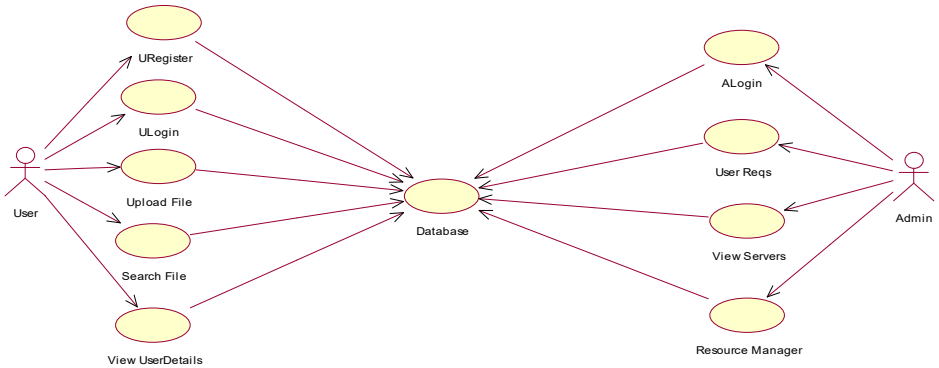
For the project titled **"Hybrid Genetic Algorithm-Based QoS-Aware Task Scheduling in Cloud Environments,"** design engineering emphasizes clear interaction flows between core modules—namely, *User* and *Admin*. It also incorporates secure task scheduling, resource management, and data handling to satisfy both functional and non-functional requirements.

The system is modeled using the following UML diagrams:

- **Use Case Diagrams:** Capture functional requirements, showing how actors initiate and interact with system services.
- **Class Diagrams:** Depict core classes, attributes, operations, and inter-class relationships.
- **Sequence Diagrams:** Describe dynamic interactions and message sequences between components during execution.
- **Activity Diagrams:** Represent workflows and control flows covering major system processes.
- **Component Diagrams:** Illustrate modular organization and component dependencies.
- **Deployment Diagrams:** Show hardware nodes and deployment of software components.

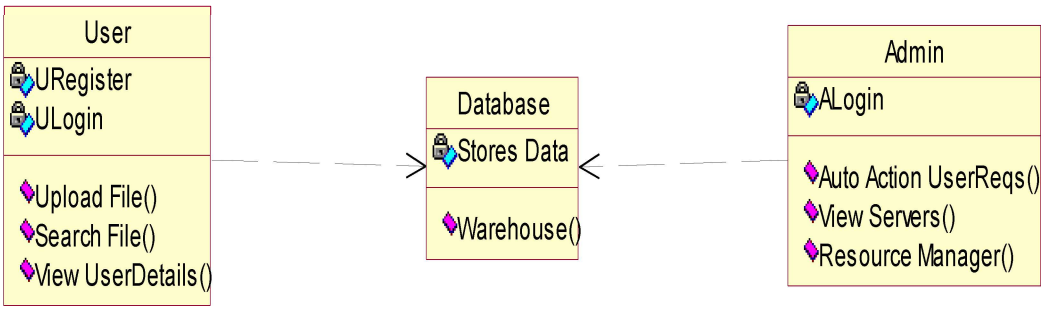
Collectively, these diagrams provide a comprehensive design representation that guides implementation, testing, optimization, and documentation throughout the software development life cycle.

Use Case Diagram



The use case diagram delineates the system's functional interactions involving two primary actors: **User** and **Admin**. Users perform operations such as registration, authentication, file upload, search, and retrieval. Administrators oversee approvals, resource monitoring, and system governance. Each use case includes proper validation and role-based access control to ensure secure execution. This high-level view clarifies how distinct actors engage with system features, facilitating effective requirement validation and system boundary definition.

Class Diagram

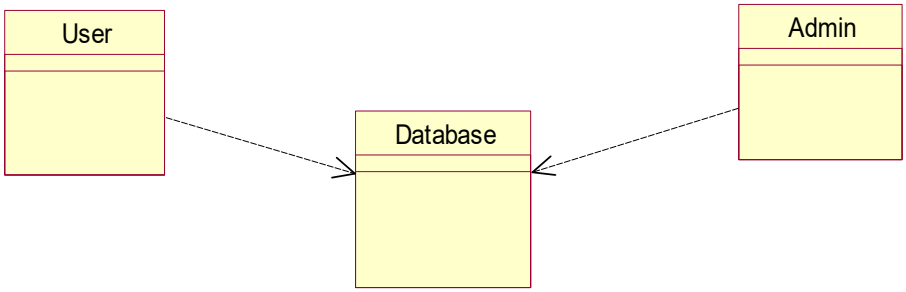


The class diagram defines the static structure of the system by specifying the principal classes, their attributes, methods, and relationships:

- **User:** Manages user identity, registration, and authentication.
- **Admin:** Handles administrative oversight including approval of users and system monitoring.
- **FileUpload:** Represents uploaded files, storing metadata like filename, path, and submission timestamp.
- **Request:** Models file access requests, tracking status and requester information.
- **Token/Permission:** Encapsulates secure access permissions issued post approval.
- **DatabaseConnection:** Facilitates persistent storage and retrieval operations.

Relationships between classes clarify associations such as file upload by users, request handling by administrators, and secure access via generated tokens. This static view illustrates system structure and data relationships critical to development.

Object Diagram

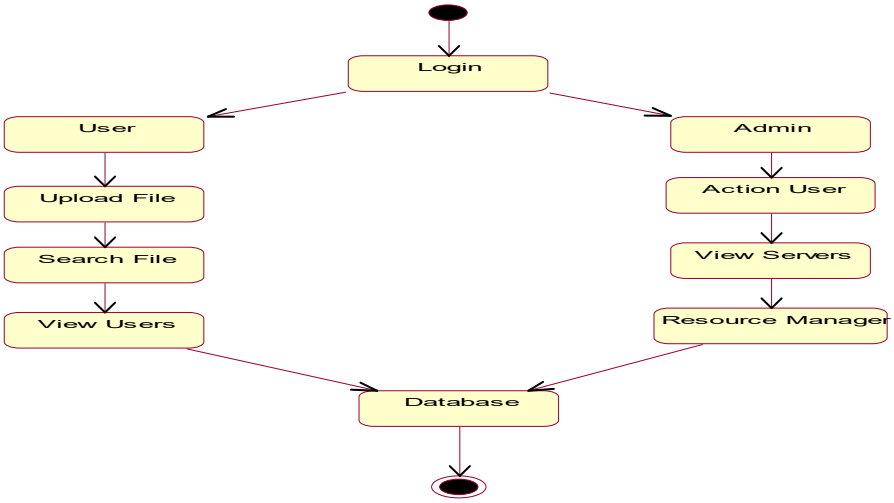


The object diagram provides a real-time snapshot of instance interactions at a defined execution point. Example instances include:

- **user1:User:** A logged-in user performing actions.
- **admin1:Admin:** An active administrator monitoring the system.
- **fileObj:FileUpload:** A specific file entity with properties like hash and status.
- **requestObj:Request:** A live file request instance linked to a file and user.
- **tokenObj:Token:** A permission object generated upon approval.
- **dbObj:DatabaseConnection:** Represents the database context storing current application state.

This diagram highlights concrete object links during operation, demonstrating how users, files, and administrative decisions interact during runtime.

State Chart Diagram



The state chart diagram depicts system state transitions throughout operational flow:

1. **Idle State:** System awaits user/admin interaction.
2. **Authentication:** Transition from idle to active state upon successful login.
3. **Processing:** Triggered when a user uploads a file or initiates a search.
4. **Retrieval:** Data is fetched from storage and presented.
5. **Admin Monitoring:** Admin enters states associated with server supervision and management.
6. **Logout:** System resets to idle on session termination.

Controlled state transitions ensure secure behavior, consistent processing, and role-specific system access.

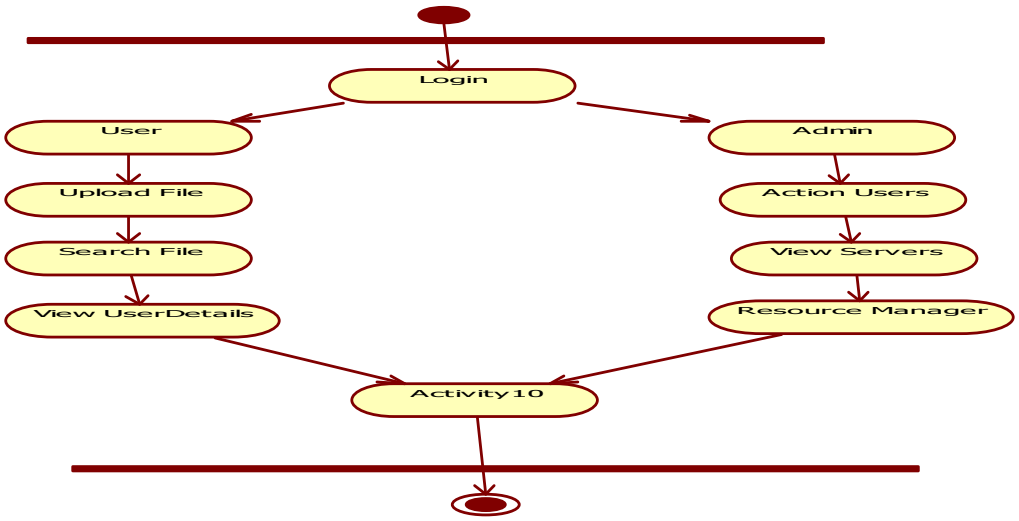
Sequence Diagram

This diagram specifies time-ordered exchanges among core components (User, Admin, Database). A typical flow includes:

- User registration inserted into database.
- Parallel login processes for users and admins.
- File upload and search operations invoking database operations.
- Admin monitoring via authenticated sessions and resource queries.

Sequence diagrams emphasize communication order and ensure that interactions maintain data integrity, authentication, and QoS adherence.

Collaboration Diagram



The collaboration diagram emphasizes structural relationships and message sequences between system modules. The user interacts with upload and search mechanisms, while the admin engages in resource control and monitoring workflows. Message exchanges articulate validated communication paths to support secure access and system coherence.

This flowchart-like depiction underscores sequential and conditional flows ensuring secure and efficient operation.

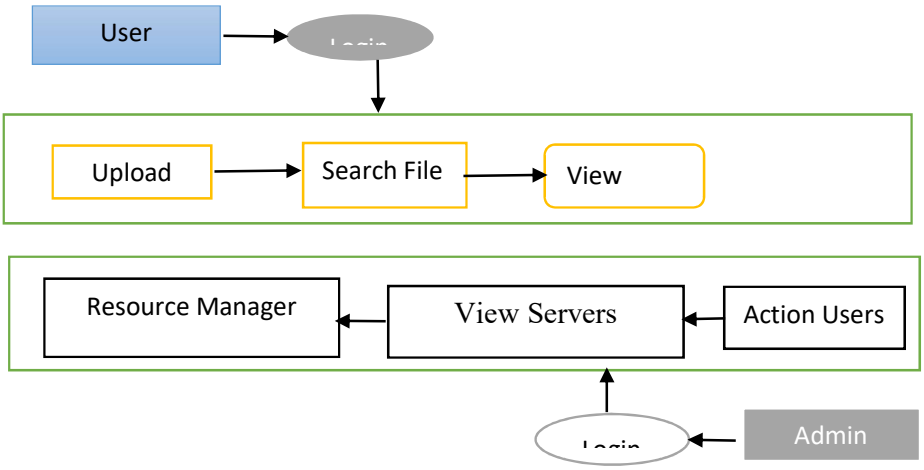
Component Diagram

Component diagrams represent modular elements and their interactions:

- **User Interface Components:** Registration, login, upload, search.
- **Admin Components:** Login, user approval, server view, and resource manager.
- **Database:** Central data repository.

Interactions between these modules illustrate how user and admin actions coordinate through shared services like authentication and data persistence.

Entity-Relationship (E-R) Diagram



The E-R model describes logical data relationships:

- **User** linked to authentication and file operations.
- **Admin** connected to oversight and resource controls.
- **File and Resource Entities** related to upload history and storage.

The database schema articulates entity relationships essential for maintaining referential integrity and secure access controls.

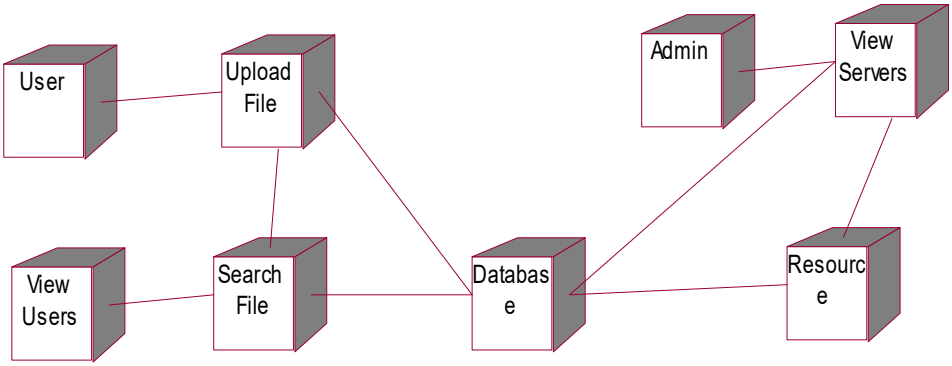
4.1.10 Data Flow Diagram (DFD)

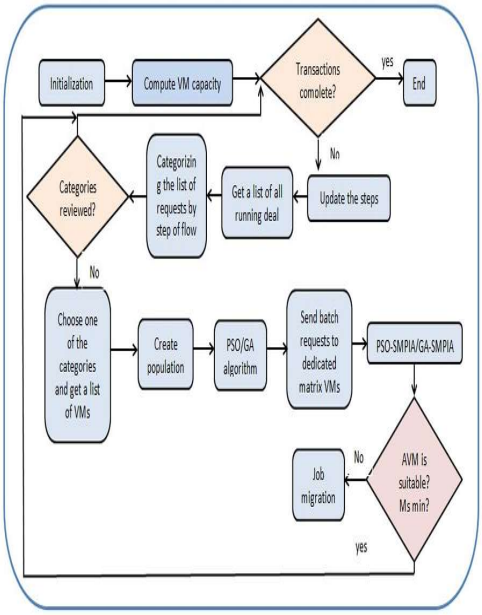
The DFD presents how data traverses through system components:

- **Level 0:** High-level overview of user, admin, system processes, and database interactions.
- **Level 1:** Detailed decomposition showing login, upload, search processes, and server monitoring pathways.

All data flows emphasize validation, storage, retrieval, and control through secure channels.

Deployment Diagram





The deployment view illustrates the physical distribution of system components:

- **Client Nodes:** User and admin access via web interfaces.
- **Application Server:** Hosts business logic, upload handlers, scheduling engine, and security modules.
- **Database Server:** Stores credentials, files, logs, and scheduling metadata.

This architecture ensures centralized processing with secure, scalable storage and controlled access.

System Architecture

The architecture implements a **Priority-Based Fair Scheduling approach** tailored for cloud environments. Key phases include:

- **Initialization:** Users and admins register and authenticate.
- **Request Intake:** User requests for uploads and searches are queued.
- **Priority Assignment:** Each request is weighted based on priority factors (role, request age, approval status).
- **Scheduling Logic:** The hybrid genetic algorithm evaluates and sequences requests for fairness and QoS optimization.
- **Execution & Monitoring:** Approved actions update database records; the Resource Manager tracks utilization.
- **Continuous Evaluation:** Pending requests remain in the scheduler queue until completion.

The overall architecture integrates secure access, efficient resource utilization, and fairness in processing while enabling scalable cloud operations.

DEVELOPMENT TOOLS

This chapter details the programming languages, frameworks, and development tools used in implementing the project. The primary development platform for this project is **Java**, with a focus on **J2EE** for building robust, enterprise-level applications. Java's versatility and platform independence make it suitable for cloud-based, database-driven systems.

Features of Java

The Java Framework

Java, developed by **James Gosling** at Sun Microsystems in 1995, is a high-level, object-oriented programming language that draws syntax from C and C++ but simplifies memory management and object handling. Java applications are typically compiled into **bytecode**, enabling execution on any device with a **Java Virtual Machine (JVM)**, regardless of hardware architecture.

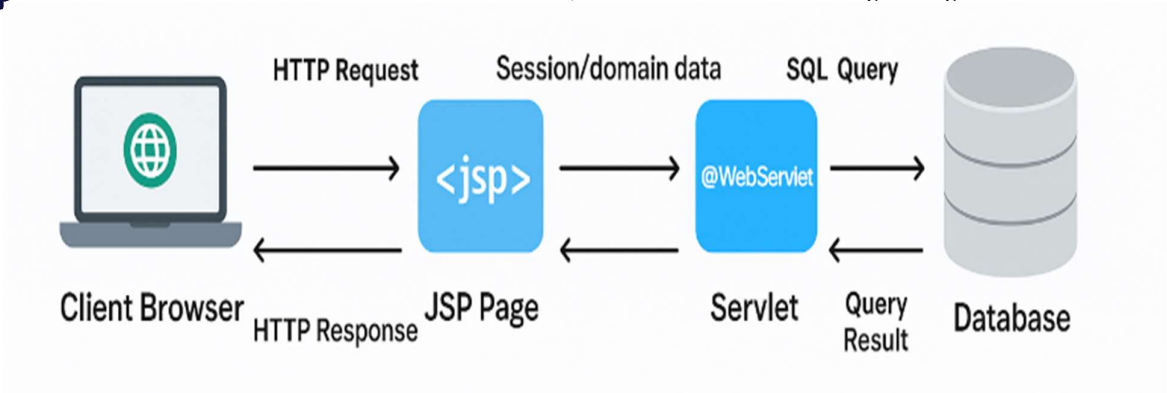
Key characteristics of Java include:

- **Platform Independence:** Write once, run anywhere.
- **Object-Oriented:** Supports modular design and reusable code.
- **Robustness and Security:** Automatic memory management, strong type checking, and secure execution environment.
- **Concurrency Support:** Built-in multithreading for efficient parallel execution.
- **Enterprise Development:** Enables server-side processing for online applications such as e-commerce platforms, forums, and forms processing.
- **Mobile and Embedded Systems:** Powers applications for smartphones, IoT devices, and consumer electronics.
- **Community Support:** Extensive resources, tutorials, and a global developer community ensure continuous innovation and support.

Object-Oriented Programming in Java

Java follows the core principles of **object-oriented programming (OOP):**

1. **Inheritance:** Enables creation of new classes by extending existing ones to reuse functionality and add new features.
2. **Encapsulation:** Combines data and methods into a single class, providing abstraction and controlled access.
3. **Polymorphism:** Supports multiple forms of a function through method overloading and overriding.
4. **Dynamic Binding:** Resolves object types at runtime, allowing flexible and adaptive behavior.



Software Testing

Software testing is a crucial phase in the software development lifecycle, aimed at detecting errors and ensuring that the system operates according to specified requirements. Testing involves executing software components with the intention of identifying faults, verifying functionality, and confirming that the system meets user expectations without failing under unacceptable conditions. Different types of testing address specific objectives and collectively ensure software reliability and quality.

Developing Testing Methodologies

Effective software testing begins with a comprehensive plan that evaluates both general functionality and specialized features across various platform configurations. The methodology involves strict adherence to quality control procedures, ensuring the application satisfies the requirements outlined in the system specification and is free from defects. Key considerations in developing a testing framework include test coverage, reproducibility, traceability, and validation against functional and non-functional requirements.

Unit Testing

Unit testing focuses on verifying the internal logic of individual software components. It involves designing test cases to validate that program inputs produce the correct outputs and that all decision branches and code paths function as intended. Unit tests are conducted after the completion of a component but before integration. This structural, knowledge-driven testing ensures that each business process or system configuration adheres to the documented specifications, with clearly defined inputs and expected results.

Functional Testing

Functional testing evaluates whether the software performs according to business, technical, and user requirements. It focuses on validating the following aspects:

System testing assesses the behavior of the fully integrated software system. It validates that the complete system meets requirements and produces predictable results under various configurations. Configuration-oriented integration tests are typical examples of system testing, emphasizing process flows and interconnections between system components.

Performance Testing

Performance testing measures whether the system meets time-related requirements, such as response time, transaction processing speed, and data retrieval efficiency. It ensures that the system can handle the expected load within predefined limits without performance degradation.

Future Enhancements

The proposed cloud task scheduling system can be further improved to enhance intelligence, scalability, and adaptability. Future work may include the integration of advanced machine learning models to enable predictive analysis of workload patterns, allowing dynamic and optimized task scheduling. Extending the hybrid scheduling framework to support heterogeneous virtual machines in real-time cloud environments can improve system responsiveness and resource efficiency. Auto-scaling mechanisms can be introduced to dynamically create or release virtual machines based on workload demands, thereby optimizing utilization and reducing operational costs. Additionally, the system can incorporate enhanced quality of service (QoS) parameters, such as energy consumption, network latency, and system throughput, to further improve overall performance. Adopting microservices and container-based deployment can increase modularity, scalability, and fault tolerance, while real-time monitoring dashboards can provide visualization of server load, task execution metrics, and system performance to support operational decision-making. Security can be strengthened through fine-grained access control and auditing mechanisms. Moreover, enabling distributed and multi-cloud architectures can improve availability, reliability, and fault tolerance. Intelligent fault detection and recovery mechanisms can reduce downtime and enhance system resilience. Finally, expanding support for large-scale workload simulations and integrating the system with real cloud service providers can help validate

performance under practical deployment scenarios and guide future optimization strategies.

Conclusion

This project demonstrates the development of an efficient cloud task scheduling system designed to optimize resource utilization and improve Quality of Service (QoS). The system effectively manages user requests, file uploads, and server allocation through structured scheduling mechanisms. By analyzing task size and server capacity, it selects the most suitable resources for execution while ensuring fair allocation through priority-based scheduling.

The system continuously monitors server load, preventing overload through job migration, and tracks user activity and system performance via administrative monitoring. Detailed execution records allow performance evaluation and analysis. The proposed approach reduces execution time, enhances overall system efficiency, and ensures reliable task execution even under high workload conditions.

Overall, the implementation demonstrates effective coordination among users, servers, and scheduling logic, providing a practical framework for cloud resource management. This system can serve as a foundation for more advanced cloud scheduling solutions in the future.

References

1. M. Mokhtari, P. Bayat, and H. Motameni, "Multi-objective task scheduling using smart MPI-based cloud resources," *Comput. Informat.*, vol. 40, no. 1, pp. 104–144, 2021.
2. T. Ma, Y. Chu, L. Zhao, and O. Ankhbayar, "Resource allocation and scheduling in cloud computing: Policy and algorithm," *IETE Tech. Rev.*, vol. 31, no. 1, pp. 4–16, Jan. 2014.
3. A. M. Manasrah and H. Ba Ali, "Workflow scheduling using hybrid GA-PSO algorithm in cloud computing," *Wireless Commun. Mobile Comput.*, vol. 2018, Art. no. 1934784.
4. A. K. Bardsiri and S. M. Hashemi, "QoS metrics for cloud computing services evaluation," *Int. J. Intell. Syst. Appl.*, vol. 6, no. 12, pp. 27–33, Nov. 2014.
5. M. Mokhtari, H. Motameni, and P. Bayat, "Solving task starvation and resource problems using optimized SMPIA in cloud," *Comput. Syst. Sci. Eng.*, vol. 42, no. 2, pp. 659–675, 2022.
6. M. Mokhtari and P. Bayat, "Improving completion time and execution time using FSMPIA: A case study," *Int. J. Nonlinear Anal. Appl.*, vol. 13, no. 1, pp. 3707–3721, 2022.
7. M. Farid et al., "A survey on QoS requirements based on particle swarm optimization scheduling techniques for workflow scheduling in cloud computing," *Symmetry*, vol. 12, no. 4, p. 551, Apr. 2020.
8. N. Kumar et al., "Achieving quality of service (QoS) using resource allocation and adaptive scheduling in cloud computing with grid support," *Comput. J.*, vol. 57, no. 2, pp. 281–290, Feb. 2014.
9. M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.
10. Y. Sun et al., "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization," *J. Syst. Softw.*, vol. 116, pp. 146–161, Jun. 2016.