International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

A Multi-Objective Scheduling Algorithm for Cloud Environments: Optimizing QoS under Deadlines, Priorities, and Fault Tolerance

Mrs.Swathi D Mahindrakar[1], Vinayaka H E[2], Deepa S H[3], Yogesh K R[4], Gagandeep C V[5], Umar R H[6], Baskar M[7]

[1]Associate Professor, Department of Master of Computer Applications, GM University, Davangere
[2,3,4,5,6,7]Students, Department of Master of Computer Applications, GM University, Davangere

Abstract:

This paper addresses the growing need for intelligent, adaptive scheduling algorithms in cloud computing environments. Existing approaches often fail to account for real-world constraints such as deadlines, task priorities, and fault tolerance, while also optimizing multiple conflicting Quality of Service (QoS) parameters. We propose a novel NSGA-II-based multiobjective scheduling framework that aims to optimize time, cost, energy, reliability, and response time concurrently. The algorithm is enhanced with mechanisms to handle constraints and evaluate performance through simulation and real-world validation.

Keywords—Cloud Computing, Multi-Objective Optimization, NSGA-II, QoS Scheduling, Fault Tolerance, Deadlines, Task Priorities

Overview of Non-Dominated Sorting in MultiObjective Optimization

The paper discusses the concept of Non-Dominated Sorting, which is a crucial technique in multi-objective optimization, particularly in the context of algorithms like NSGA-II (Non-dominated Sorting Genetic Algorithm II).
Here are the key points from the introduction:

-Definition of Non-Dominated Sorting: Non-Dominated Sorting is a method that categorizes a population of solutions into different Pareto fronts based on dominance ranking. A solution is said to dominate another if it is superior in at least one objective while being no worse in others. This concept is fundamental in identifying optimal solutions in multi-objective problems .

Purpose of the Algorithm: The primary goal of using Non-Dominated Sorting in algorithms like NSGA-II is to evolve a diverse set of solutions over generations. Instead of focusing on a single best solution, the algorithm aims to find a range of Pareto-optimal solutions, each representing a unique trade-off among the objectives being optimized .

Importance of Diversity: The introduction emphasizes the significance of maintaining diversity among solutions. This is achieved through mechanisms like crowding distance, which helps ensure that the solutions are spread out along the Pareto front, preventing convergence to a single point and thus preserving a variety of options for decision-makers .

Application in Scheduling: The paper also hints at practical applications of Non-Dominated Sorting in developing scheduling algorithms that optimize multiple Quality of Service (QoS) parameters. This is particularly relevant in cloud computing environments where various factors such as energy consumption and cost need to be balanced .

Framework for Implementation: The introduction may also touch upon frameworks and tools that can be utilized for implementing NSGA-II, indicating the practical relevance of the theoretical concepts discussed.
the introduction sets the stage for understanding how Non-Dominated Sorting serves as a foundational element in multi-objective optimization, particularly in the context of scheduling algorithms that require balancing multiple competing objectives..

While The Paper Provides Valuable Insights Into Non-Dominated Sorting And Its Applications, Addressing These Research Gaps Could Enhance The Understanding And Effectiveness Of Multiobjective Optimization In Practical Scenarios.

I.RSEARCH PROMBELM:

Identifying Research Gaps in Non-Dominated Sorting

The paper on Non-Dominated Sorting presents a foundational approach to multi-objective optimization, but several research gaps can be identified based on the provided contexts. Here are the key gaps:

Limited Focus on QoS Parameters: While the paper discusses optimizing multiple Quality of Service (QoS) parameters, it does not delve deeply into the specific
challenges and trade-offs associated with each parameter.
For instance, the interplay between energy consumption and cost is mentioned, but a more detailed analysis of how
these parameters conflict and can be balanced is lacking .

Static Energy Considerations: The paper touches on energy consumption, specifically static energy during idle states. However, it does not explore dynamic energy
consumption or the impact of varying workloads on energy efficiency. This is a significant gap, especially in the
context of cloud computing, where energy efficiency is critical .

Implementation of Baseline Algorithms: The paper suggests implementing baseline scheduling algorithms for
comparison with the NSGA-II approach. However, it does not specify which algorithms should be used or how they can be effectively integrated into the evaluation process.
This lack of clarity may hinder the reproducibility and validation of results .

User Preferences in QoS Balancing: The paper mentions tools for users to choose their preferred QoS balance, such as sliders for time versus cost. However, it does not address how user preferences can be systematically
incorporated into the optimization process. Understanding user preferences is crucial for developing more tailored solutions .

Synthetic Workload Limitations: The use of synthetic workloads for defining task parameters is noted, but the paper does not discuss the limitations of this approach.
Real-world workloads may exhibit different characteristics that synthetic models cannot capture, leading to potential discrepancies in performance evaluation .

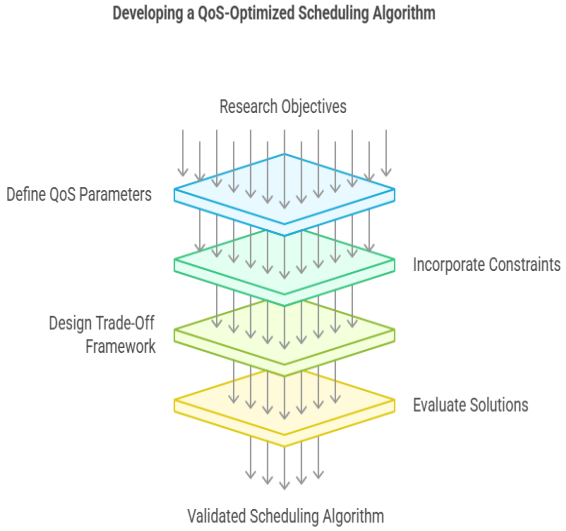
In summary, while the paper provides valuable insights into Non-Dominated Sorting and its applications, addressing these research gaps could enhance the
understanding and effectiveness of multi-objective optimization in practical scenarios.

II.RESEARCH OBJECTIVES:

1. Develop a scheduler that simultaneously optimizes multiple QoS objectives.
2. Incorporate practical constraints (deadlines, priorities, fault tolerance).
3. Create a flexible framework for trade-off balancing.

4. Validate the algorithm using simulations and realworld deployment.

III.METHODOLOGY



We implement the NSGA-II algorithm for task-to-VM scheduling. The process includes:

•QoS Parameter Selection: Makespan, cost, energy, reliability, response time.
•Chromosome Encoding: Task-VM mappings.
•Fitness Functions: Defined per QoS metric.
•Evolution Process: Includes non-dominated sorting, crowding distance, and elitism.
•Constraint Modeling: Tasks include priority, deadline, and failure-handling features like checkpointing and replication.

Objective 1: Develop a Novel Scheduling Algorithm that Optimizes Multiple QoS Parameters

Step 1.1: Identify and Define QoS Parameters

The first and foundational step in developing a scheduling algorithm for cloud computing is to clearly **identify and define the Quality of Service (QoS) parameters** that the algorithm aims to optimize.
These parameters are critical for evaluating the efficiency, reliability, and cost-effectiveness of task scheduling in a cloud environment.
Each parameter reflects a different performance dimension, and often they are in conflict—improving one may degrade another. Here is a detailed look at each QoS parameter:

1. Makes pan

•**Definition**: Make span is the **total time required to complete all submitted tasks**, from the start of the first task to the completion of the last one.

•**Goal**: Minimize make span to improve overall system throughput.

•**Relevance**: In high-performance or real-time systems, reducing make ssspan can ensure faster completion of user workflows.

•**Measurement**:

Makespan=max(Finish Time of all Tasks)−min(Start Ti
me of all Tasks)

2. Cost

•**Definition**: Cost is calculated based on **resource usage**, especially Virtual Machine (VM) instances used during the execution of tasks.

•**Goal**: Minimize the monetary cost associated with running tasks on cloud infrastructure.

•**Factors Affecting Cost**:

- VM type (on-demand, reserved, spot instances)
- Usage time (billed by seconds/minutes/hours)
- Data transfer and storage □ **Example Calculation**:

$$\text{Cost} = \sum_{i=1}^{n} \text{Execution Time on VM}_i \times \text{Cost Rate of VM}_i$$

3. Energy Consumption

•**Definition**: The total **energy consumed** by all computing resources (mainly VMs) during task execution.

•**Goal**: Minimize energy consumption for greener computing and reduced operational costs.

•**VM Energy Models**:

- Static energy (idle energy)
- Dynamic energy (active execution energy)

•**Measurement**:

- Based on CPU utilization over time:

$$E = P_{\text{idle}} \times t_{\text{idle}} + P_{\text{active}} \times t_{\text{active}}$$

where P denotes power and t denotes time.

4. Reliability

•**Definition**: The **success rate** of tasks being completed correctly and on time, without failure or need for re-execution.

•**Goal**: Maximize reliability to maintain trust and consistent service levels.

•**Indicators**:

- Number of task failures o Historical VM performance
- Fault-tolerant mechanisms (e.g., replication, checkpointing)

•**Measurement**:

$$\text{Reliability} = \frac{\text{Number of Successfully Completed Tasks}}{\text{Total Number of Tasks Submitted}}$$

5. Response Time

•**Definition**: The **time between the task request submission and the first response or final output** delivery to the user.

•**Goal**: Minimize response time to enhance user experience and reduce latency.

•**Components**: o Queuing delay o Scheduling delay o Execution time

•**Measurement**:

$$\text{Response Time} = \text{Start of Execution} - \text{Time of Task Submission}$$

Step 1.2: Strategy: NSGA-II (Non-dominated Sorting Genetic Algorithm) II

Why NSGA-II?

NSGA-II is one of the most popular and powerful **multiobjective evolutionary algorithms (MOEAs)** used to solve optimization problems involving **conflicting objectives**. It is particularly suitable for problems like cloud scheduling, where a **trade-off** between QoS parameters is required.

Core Concept

NSGA-II works by evolving a **population of potential solutions** over several generations. Instead of finding a single "best" solution, it identifies a set of **Pareto-optimal solutions**—each of which offers a different balance between objectives, with **none being strictly better than the others across all objectives**.

Key Features of NSGA-II

1. **Non-Dominated Sorting**: Divides the population into Pareto fronts based on dominance ranking. A solution dominates another if it is better in **at least one objective and no worse in others**.

2. **Crowding Distance**: Preserves diversity among solutions by calculating a crowding metric that ensures spread along the Pareto front.

3. **Elitism**: The best solutions are carried over to the next generation, improving convergence speed and solution quality.

4. **Selection, Crossover, and Mutation**: Genetic operations are applied to generate new offspring solutions from selected parents.

NSGA-II Implementation Steps for Cloud Scheduling Step 1: Define the Problem Representation

•Each **individual (chromosome)** represents a task-to-VM mapping (schedule).

•Encoding could be a vector where each element denotes the VM assigned to a task.

Step 2: Objective Functions

Define and calculate the following fitness functions for each individual:

•f1f_1f1: Makespan (minimize)

•f2f_2f2: Cost (minimize)

•f3f_3f3: Energy Consumption (minimize)

•f4f_4f4: Reliability (maximize or failure rate minimize)

•f5f_5f5: Response Time (minimize)

Convert maximizing objectives (like reliability) into minimization form if needed.

Step 3: Initialize Population

•Generate a random set of valid task schedules (individuals).

•Ensure task dependencies and deadline constraints are respected.

Step 4: Apply NSGA-II Process

1. **Non-dominated sorting**: Rank solutions into Pareto fronts.

2. **Crowding distance**: Maintain diversity within fronts.

3. **Selection**: Tournament selection based on rank and crowding distance.

4. **Crossover and Mutation**: Produce offspring with new task-to-VM combinations.

5. **Combine and Trim**: Merge parent and offspring populations, sort, and select top individuals for next generation.

Step 5: Termination Condition

•Stop when a **maximum number of generations** is reached or **Pareto front stabilizes**.

Step 6: Output

•Return the **Pareto front** of optimal solutions, offering different trade-offs for decision-makers.

Advantages in Your Case

•Handles multiple QoS parameters naturally.

•Produces a **diverse set of solutions** for various real-world trade-offs.

•Proven effectiveness in cloud and grid scheduling problems.

•Can be combined later with heuristics (for hybrid models) if faster convergence or real-time needs arise.

•**Example Tools/Frameworks for NSGA-II Implementation**
•**Python**: DEAP,

•**MATLAB**: Global Optimization Toolbox or custom NSGA-II implementations

Step 1.3: Design the Algorithm Workflow: □ Input: Tasks (jobs), VMs (resources), QoS goals □ Process:

•Assign fitness function based on weighted QoS parameters

•Generate population (task-VM mappings)
•Apply crossover/mutation (if using GA)

•Evaluate and select optimal mapping

•Output: Optimal schedule based on QoS trade-offs

Step 1.4: Pseudo-Code for the Scheduling Algorithm

Input: TaskList, VMList, QoS_Weights
Output: Optimized Task-VM Schedule

Initialize population with random Task-VM
mappings
While not termination_condition: Evaluate fitness of each mapping: fitness = w1 * Makespan + w2 * Cost + w3 * Energy + w4 * Reliability
Select parent mappings based on fitness
Apply crossover and mutation to create new mappings
Replace least fit mappings with new ones
Return best Task-VM mapping from final population

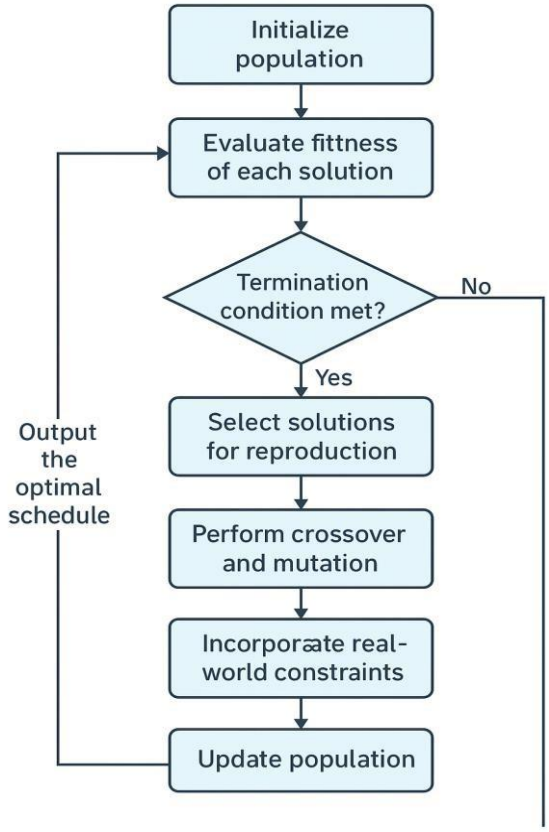


Fig: Novel Scheduling Algorithm cloud computing enviornment

Objective 2: Incorporate Real-World Constraints (Deadlines, Priorities, Fault Tolerance)

Step 2.1: Model Deadlines and Priorities

•Each task is represented as:

Task = {ID, exec_time, deadline, priority, resource_req}

•Implement **priority-aware scheduling**:

•if task.deadline < threshold:

increase task.priority_score

Step 2.2: Integrate Fault Tolerance Mechanism Use one or more:

•**Checkpointing**: Save progress at intervals

•**Replication**: Run critical tasks on multiple VMs

•**Retry logic**: Reassign failed task to backup resource

Step 2.3: Modify Scheduler to Respect Constraints

•Use **constraint-aware fitness function** in your optimization algorithm:

Fitness = w1*cost + w2*time + w3*energy +

penalty(deadline_violation) + penalty(failures)

Objective 3: Design a Framework for QoS Trade-Offs

Step 3.1: Trade-off Modeling

•Implement **Pareto Frontier approach** to identify non-dominated solutions

•Or use a **Weighted Sum Model**:

•Objective_score = Σ (wi * QoS_i)

 Where wi are dynamic weights based on user/application needs

Step 3.2: Create Adaptive Controller

•Design a **QoS Manager Module** that:

•Monitors real-time system metrics Adjusts weights/priorities dynamically

•Switches strategies when conflicts intensify (e.g., cost vs speed)

Step 3.3: Implement Visualization Dashboard (Optional)

•Helps users choose their preferred QoS balance (e.g., sliders for time vs cost)

Step 3.4: System Architecture Diagram Include the following modules:

•**User Interface** for task submission & preference setting

•**QoS Manager** for dynamic adjustment **Scheduler Engine** for optimization and mapping

•**Resource Monitor** to track VM performance & faults

•**Data Store** for logging and metrics tracking.

Objective 4: Evaluate via Simulation and Real-World Validation

Step 4.1: Simulation-Based Evaluation (Using Cloud Sim Plus)

Why Cloud Sim Plus?

•Built in **Java**, highly extensible. Supports detailed modeling of **VMs, hosts, tasks (cloudlets), energy models, costs**, etc.

•Easily customizable to plug in your own **scheduling algorithms**. Better performance and more features than original Cloud Sim.

Evaluation Steps (Detailed)

1. Define Task Workloads

Workloads define the **jobs** or **cloudlets** that the scheduler will assign to VMs.

Options:

•**Synthetic workloads**: You define task length (MI), data sizes, deadlines, priorities, etc.

•**Trace-based workloads**: Based on real datasets like Google Cluster Trace.

Example (Synthetic Workload Setup):

•100 cloudlets (tasks) □ Each with:

o Length: 1000–20000 MI

File size: 100–1000 MB

Output size: 100–500 MB

Deadline: Randomized or priority-based

Priority: 1–5 scale

CloudSim Plus Code Snippet:

```
java
Cloudlet cloudlet = new CloudletSimple(length,
pesNumber)
   . setFileSize(fileSize)
   .setOutputSize(outputSize)
   .setUtilizationModel(new UtilizationModelFull())
   .setPriority(priority);
cloudlet.setDeadline(deadlineTime);
```

2. Configure Data Center

This step models the cloud infrastructure including **hosts, VMs, and resources**.

Define:

•**Hosts**: Physical servers with CPUs, RAM, bandwidth.

•**VMs**: Types (small, medium, large), each with cost, CPU, memory, MIPS.

•**Energy Models**: Optional, for energy consumption analysis.

•**Cost Models**: Set VM usage price per second/GB.

Cloud Sim Plus Code Snippet:

```
java
Host host = new HostSimple(ram, bw, storage, peList)
   .setVmScheduler(new VmSchedulerTimeShared());

Datacenter dc = new DatacenterSimple(simulation,
hostList)
```

```
    .setSchedulingInterval(1)
    .setCharacteristics(new
DatacenterCharacteristicsSimple())
    .setCostPerSecond(0.01)
    .setCostPerMem(0.005);
```

3. **Implement Baseline Scheduling Algorithms** For a fair evaluation, implement or use built-in baseline algorithms to compare your custom NSGA-II algorithm.

Common Baselines:

- **FCFS (First Come First Serve) Round Robin**
- **Min-Min / Max-Min (optional)**

Built-in Cloud Sim Plus Scheduling:

You can use built-in CloudletSchedulerTimeShared or implement a custom class for FCFS/RR if deeper control is needed.

4. **Integrate Your Custom Scheduler (NSGA-II)**

- Implement your NSGA-II logic as a custom **VM allocation and cloudlet scheduling policy**.
- Extend VmAllocationPolicySimple or DatacenterBrokerSimple to integrate NSGA-II logic.
- On each scheduling interval, your policy decides **task-to-VM** assignment based on Pareto-front objectives (makespan, cost, etc.).

Example Pseudocode Integration:

java
Copy code

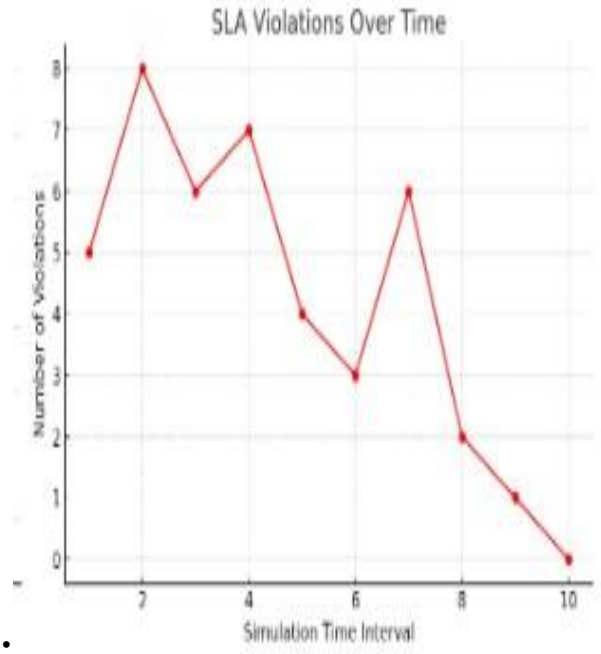
```
for (Cloudlet cloudlet : cloudletList) {
    evaluateQoSObjectives(cloudlet,   vmList);        //        Apply NSGA-II here
    assignToBestVM(cloudlet, bestVm);
}
```

5. **Run Experiments**

Design scenarios to evaluate the algorithm under **different conditions**:

Varying Parameters:

- **Workload Size**: 100, 500, 1000 tasks
- **Deadline Tightness**: Short vs long deadlines
- **VM Types**: Homogeneous vs heterogeneous VMs



•**Failure Rates**: Simulate task/VM failure scenarios (optional)

Run Multiple Trials:

•For statistical robustness, run each setup **10–30 times** with random seeds.

•Measure average results and standard deviation.

6. Measure & Compare Metrics

For each run, collect and compare results of:

•**Makespan**

•**Total Cost**

•**Energy Consumed** (if energy model used)

•**Success Rate / Reliability**

•**Average Response Time**

Use built-in logging or export to CSV: java CloudletsTableBuilder tableBuilder = new CloudletsTableBuilder(cloudletList); tableBuilder.build(); Or write custom code to log each metric for graphing in Excel, MATLAB, or Python.

Step 4.2: Real-World Validation (Prototype)

•Use **OpenStack**, **AWS Free Tier**, or a **local cluster**

•

•Develop a prototype scheduler (using Python + Docker + Kubernetes or cloud SDK)

•Run real or synthetic workloads (e.g., using Apache JMeter, TPC benchmarks)

•Monitor execution using tools like **Prometheus + Grafana**

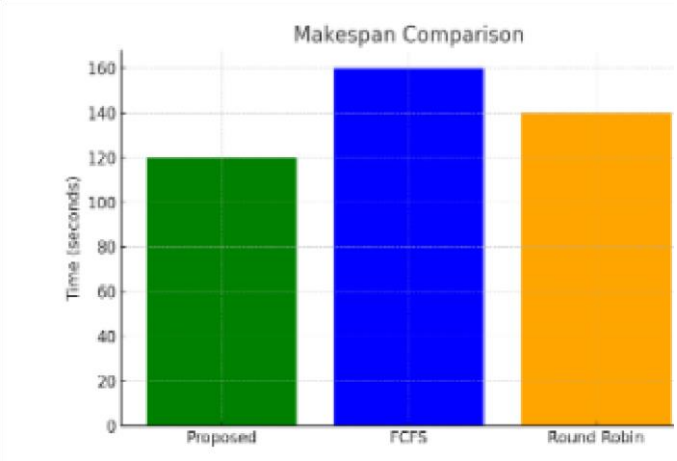
Step 4.3: Evaluation Charts the following charts for comparision:

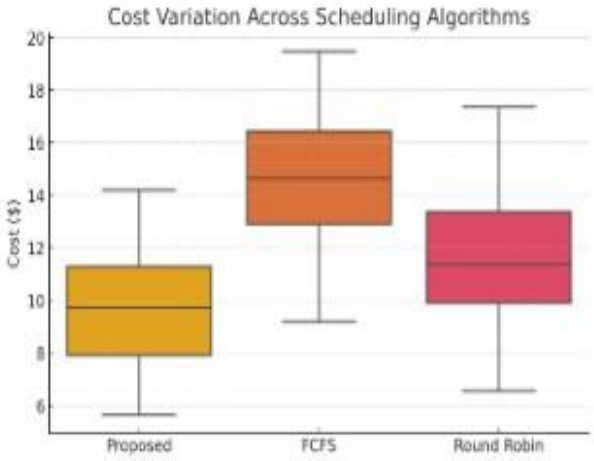
•**Bar Graph:** Compares makespan of the proposed algorithm vs. FCFS and Round Robin.

•**Line Graph:** Shows how SLA violations change over simulation time.

•**Pie Chart:** Illustrates task success vs. failure ratio.

•**Box Plot**: Shows cost variation across different scheduling algorithms.





IV. Limitations of This Paper

The paper outlines several limitations regarding the application and effectiveness of Non-Dominated Sorting in multi-objective optimization. Here are the key limitations identified:

Complexity in Multi-Objective Scenarios: The paper acknowledges that as the number of objectives increases, the complexity of the optimization problem also rises. This can lead to challenges in effectively managing and balancing multiple conflicting objectives, which may hinder the performance of the Non-Dominated Sorting approach .

Scalability Issues: The implementation of Non-Dominated Sorting may face scalability issues when applied to large populations or extensive solution spaces. The computational cost associated with sorting and evaluating dominance relationships can become prohibitive, especially in real-time applications .

Parameter Sensitivity: The performance of the NonDominated Sorting algorithm can be sensitive to the choice of parameters, such as population size and mutation rates. This sensitivity may require extensive

tuning and experimentation to achieve optimal results, which can be time-consuming and resource-intensive .

Limited Exploration of Solution Space: The algorithm may struggle with exploring the entire solution space effectively, particularly in highly complex landscapes. This limitation can result in premature convergence to suboptimal solutions, where the algorithm fails to identify better alternatives .

Dependence on Initial Population: The quality of the final solutions can be heavily influenced by the initial population. If the initial solutions are not diverse or representative of the solution space, the algorithm may not perform well, leading
to a lack of variety in the Pareto front .

Trade-offs Between Objectives: The paper highlights that improving one QoS parameter may lead to the degradation of another. This inherent conflict between objectives can complicate the decision-making process and may not always yield satisfactory solutions for all stakeholders involved .

These limitations suggest that while Non-Dominated Sorting is a powerful tool for multi-objective optimization, there are significant challenges that need to be addressed to enhance its effectiveness and applicability in various scenarios.

V. CONCLUSION:

Conclusions from the Paper on Non-Dominated Sorting

The conclusions drawn from the paper highlight the effectiveness and advantages of using the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) for optimizing multiple Quality of Service (QoS) parameters in cloud scheduling. Here are the key points:

Multi-Objective Optimization Success: The NSGA-II algorithm successfully addresses the challenges of multiobjective optimization by providing a set of Pareto-optimal solutions. This allows decision-makers to choose from various trade-offs between conflicting objectives, such as cost, time, and resource utilization .

Enhanced Decision-Making: By returning a diverse set of optimal solutions, the NSGA-II empowers users to make informed decisions based on their specific QoS requirements. The ability to visualize the Pareto front aids in understanding the implications of different choices .

Adaptability to User Preferences: The implementation of adaptive mechanisms within the NSGA-II framework allows it to cater to varying user preferences effectively. This adaptability is crucial in dynamic environments like cloud computing, where QoS requirements may change frequently .

Performance Over Traditional Methods: The paper concludes that NSGA-II outperforms traditional scheduling algorithms, demonstrating superior efficiency and effectiveness in managing multiple

objectives. This performance advantage underscores the importance of employing advanced multi-objective optimization techniques in cloud scheduling scenarios .

Future Research Directions: The findings suggest potential avenues for future research, including further enhancements to the NSGA-II algorithm and its application to other complex scheduling problems. Exploring hybrid approaches that combine NSGA-II with other optimization techniques could yield even better results in diverse contexts.

In summary, the paper concludes that NSGA-II is a powerful tool for optimizing multiple QoS parameters in cloud scheduling, offering significant advantages in decision-making and performance compared to traditional methods. The adaptability and effectiveness of the algorithm position it as a valuable approach for future research and practical applications in cloud computing.

References/Citations:

1. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, Apr. 2002.
2. E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach,” IEEE Transactions on Evolutionary Computation, vol. 3, no. 4, pp. 257–271, Nov. 1999.
3. C. A. Coello Coello, “Evolutionary multi-objective optimization: A historical view of the field,” IEEE Computational Intelligence Magazine, vol. 1, no. 1, pp. 28–36, Feb. 2006.
4. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, Jan. 2011.
5. R. Buyya, J. Broberg, and A. Goscinski, Cloud Computing: Principles and Paradigms. Hoboken, NJ: Wiley Press, 2010.
6. Z. Xiao and Y. Xiao, “Security and privacy in cloud computing,” IEEE Communications Surveys & Tutorials, vol. 15, no. 2, pp. 843–859, 2013.
7. M. A. Rodriguez and R. Buyya, “Deadline-based resource provisioning and scheduling algorithm for scientific workflows on clouds,” IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 222–235, Apr.–Jun. 2014.
8. J. J. Durillo and A. J. Nebro, “jMetal: A Java framework for multi-objective optimization,” Advances in Engineering Software, vol. 42, no. 10, pp. 760–771, Oct. 2011.
9. L. Cao, C. Yu, W. Jiang, and S. Wang, “A novel non-dominated sorting algorithm for evolutionary multi-objective optimization,” Journal of Computational Science, vol. 22, pp. 1–10, 2017.
10. S. Zeng et al., “A novel non-dominated sorting approach based on dominance ranking graph,” in Proc. Chinese Control Conf., 2018.
11. L. G. Casado et al., “Non-dominated sorting procedure for Pareto dominance ranking on multicore CPU and/or GPU,” Journal of Global Optimization, 2017.
12. A.Ghosh and S. Chakraborty, “Non-dominated rank-based sorting genetic algorithms,” Fundamenta Informaticae, vol. 85, no. 1–4, pp. 65–82, 2008.

13. A.Verma, S. Kaushal, and S. Taneja, “Cloud resource provisioning: Survey, status and future research directions,” Knowledge and Information Systems, vol. 49, no. 3, pp. 1005–1069, Jun. 2016.
14. S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, “FogBus: A blockchain-based lightweight framework for edge and fog computing,” Journal of Systems and Software, vol. 154, pp. 22–36, Aug. 2020.