School of Computing

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

NWC

Resource Allocation Algorithms: Detailed Analysis, Optimal Approaches, and Real-World Applications

Introduction

- Resource allocation is the process of distributing limited resources efficiently across competing demands in domains like cloud computing, networking, and supply chain management.
- Various algorithms, including Linear Programming, Game Theory, Heuristics, and AI-based methods, are used to optimize allocation while balancing cost, efficiency, and constraints.
- This project analyzes existing resource allocation algorithms, evaluates their efficiency, scalability, and real-world applications, and explores optimal hybrid approaches for improved performance.

Problem Selection and Description

- Existing resource allocation algorithms face challenges like high time complexity, inefficient allocation, and scalability issues, leading to suboptimal performance.
- These limitations impact cloud computing, job scheduling, and network bandwidth management, where efficient resource distribution is crucial.
- The goal is to redesign an improved algorithm that enhances efficiency, reduces computational overhead, and optimally allocates resources in real-time.

Existing Algorithm Analysis

- Greedy Algorithms: Allocate resources based on immediate best choices, used in Knapsack and Auction-based allocations.
- Divide & Conquer Methods: Break the problem into subproblems, used in Merge-based resource partitioning.
- Dynamic Programming (DP): Stores intermediate results to optimize allocation, used in Knapsack DP and Linear Programming Approximation.

Complexity Analysis

Algorithm	Recurrence Relation	Master's Theorem Analysis	Time Complexity	Space Complexity	Usage in Resource Allocation
Binary Search-Based Matching	$T(n) = T(n/2) + O(1)$	Case 2: $O(\log n)$ dominates	**$O(\log n)$**	**$O(\log n)$**	Fast lookup in **task scheduling, cloud resource matching**
Greedy Allocation (Auction, Knapsack Greedy)	No recurrence, sorts then assigns	-	**$O(n \log n)$** (sorting)	**$O(1)$ to $O(n)$**	**Real-time resource allocation, auctions**
Dynamic Programming (Knapsack, LP Approximation)	$T(n) = T(n-1) + O(n)$	Doesn't fit Master's Theorem	**$O(n^2)$ or $O(nW)$**	**$O(nW)$**	**Optimal allocation but high space usage**
Heuristic-Based Allocation	$T(n) = 2T(n/2) + O(n)$	Case 2: $O(n \log n)$ dominates	**$O(n \log n)$**	**$O(n)$**	**Cloud computing, load balancing, traffic control**

Steps Involved in Solving the Problem

- Preprocessing: Resources are sorted based on priority factors (e.g., demand, availability).
- Divide & Conquer Optimization: The dataset is split, and recursive allocation is performed with priority adjustments.
- Greedy Selection Refinement: The best resource is assigned based on real-time constraints.
- Dynamic Adjustment: The allocation is fine-tuned based on feedback, ensuring optimal resource utilization.
- Final Validation: The assigned resources are evaluated using efficiency metrics like execution time and utilization rate.

Mathematical Formulation of T(n)

- The recurrence is $T(n) = aT(n/b) + f(n)$, where a is subproblems, b is size reduction, and f(n) is merging cost.
- Example: If $a = 2$, $b = 2$, $f(n) = O(n)$, the relation is $T(n) = 2T(n/2) + O(n)$, defining problem breakdown.

Proposed Algorithm (Redesigned Approach)

- The redesigned algorithm optimizes allocation by combining Greedy, Divide & Conquer, and Heuristic Methods.
- Uses priority-based resource selection, reducing computation while maintaining near-optimal results.
- Enhances scalability and adaptability for real-time applications like cloud computing and task scheduling.

Master's Theorem Analysis

- Identify Recurrence Relation:
- Given $T(n) = 2T(n/2) + O(n)$, where $a = 2$, $b = 2$, and $f(n) = O(n)$.
- Apply Master's Theorem:
- Compute $\log_2(2) = 1$. Since $f(n) = O(n)$ matches $O(n^{\log_2(2)}) = O(n)$, we use case 2 of Master's Theorem.
- Determine Time Complexity:
- Since $f(n) = O(n) = O(n^1)$, the final complexity is $O(n \log n)$, making the algorithm efficient and scalable.

Conclusion and Future Work

- The proposed method improves speed, accuracy, and scalability over existing techniques, making it more efficient for resource allocation.
- Future work will focus on enhancing adaptability for large datasets and integrating AI-based optimizations for better performance.

Experimental Evaluation and Results

- The proposed algorithm achieves faster execution and lower complexity, proving its efficiency and scalability in large datasets.
- Used in cloud resource allocation to improve load balancing and security.
- Applied in blockchain data security for fast and secure transaction verification.

Guided by Dr.A.Prabhu Chakravarthy

Members Radhakrishna Chaitanya Manyam RA2311030010091

Poda Mounitha RA2311030010075