LAUTECH Journal of Computing and Informatics (LAUJCI) – ISSN: 2714-4194
Volume 3 Issue 1, June 2023 – www.laujci.lautech.edu.ng

RESOURCE ALLOCATION IN CLOUD COMPUTING USING A GENERALIZED KNAPSACK ALGORITHM

Dayo Reuben Aremu
Department of Computer Science,
University of Ilorin, Ilorin, Nigeria
draremu2006@gmail.com

Abiodun K. Moses
Department of Computer Science,
University of Ilorin, Ilorin, Nigeria
abbeykmos@yahoo.com

Oluwasogo S.A
Department of Computer Science,
University of Ilorin, Ilorin, Nigeria
samueloluwasogo@yahoo.com

ABSTRACT

Efficient allocation of resources in order to achieve optimal performance and cost-effectiveness is a critical challenge in cloud computing. This paper presents the Generalized Knapsack Algorithm (GKA) in order to address the resource allocation problem in cloud environments. The GKA aims to maximize the utilization of computing resources, memory, and bandwidth while considering various constraints. The paper presents a comprehensive analysis of the GKA's performance using both simulated experiments and real-world cloud datasets. Results demonstrate that the GKA outperforms existing resource allocation methods in terms of efficiency and scalability. The proposed approach provides a promising solution for enhancing resource allocation strategies in cloud computing, enabling better resource utilization and improved service delivery for cloud users. The study contributes to the advancement of cloud computing optimization and has practical implications for cloud service providers and users, fostering more effective resource management in cloud environments.

Keywords: Cloud-Computing, Knapsack-Problem, resource-allocation, and Generalized-Knapsack-Algorithm

1. INTRODUCTION

Cloud computing has emerged as a new computing paradigm for providing reliable, customized and Quality of Service (QoS) guaranteed and dynamic computing environments for scalable sharing of heterogeneous computing resources across large geographical distances and administrative domains. According to (Ousterhout, 1982), Resource Allocation (RA) in cloud computing is the process of assigning available resources to the needed cloud applications over the internet. This requires the type and amount of resources needed by each application in order to complete a user job. One of the key challenges in cloud computing is to present resource allocation design that is both useful and reliable to scientific users. This of course motivates further investigations into the optimization of cloud resource allocation strategy which provides a means of implementing quality of service (QoS) policies on the cloud. The strategy which should be based around the concept of options; that is when a task is submitted to the cloud, there should be many options available to allocate resources to the task of the user. The aim of this paper therefore is to develop a generalized Knapsack Model for resource allocation in Cloud computing environment. The specific objectives were to: formulate a generalized Knapsack based problem design for resource allocation in the cloud; implement the model designed; and evaluate the implemented model. The rest of the paper is organized as follows: Section 2 presented the related work by discussing the various types of knapsack problems and existing strategies/techniques for allocation of resources. Section 3 discussed the methodology by presenting the knapsack problem formulation for allocating cloud resources and solution design approach for implementing the knapsack model. Section 4 presents the implementation of the knapsack model. While section 5 presented the evaluation of the implemented model by comparing the theoretical and the computational results of the knapsack model, and section 6 concluded the paper.

2. RELATED WORKS

The Knapsack Problem (KP) derives its name from the thief's problem of choosing which of a variety of treasures to steal in such a way that the value of his spoils is maximized and that he doesn't exceed the weight carrying capacity of his knapsack. This section presents different types of Knapsack problems and discussed from literature, a number of strategies /techniques used for resource allocation on the grid and other related distributed systems by classifying them into the following categories: (i). First-come-first-served; (ii). Backfilling; (iii) Makespan Minimizing Heuristics; (iv).Gang Scheduling; (v). Matchmaking; and (vi). The Economic Grid and Market-based Resource Allocations.

2.1 TYPES OF KNAPSACK PROBLEMS

There are different types of Knapsack problems. Each of these different types is discussed as follows:

0-1 Knapsack Problem

The most common type of KP is the 0-1 Knapsack Problem. This problem is referred to as a 0-1 knapsack problem, as opposed to a fractional knapsack problem, because each treasure is indivisible. This type of KP restricts the number of copies of each kind of item that is taken to be zero (0) or one (1). The 0-1 Knapsack Problem is the problem of choosing a subset of the n items such that the corresponding profit sum is maximized without having the weight sum to exceed the capacity W. The Linear Programming Problem for the0-1 Knapsack Problem or the knapsack problem in its most basic form is:

$$\begin{aligned}
&Maximize \quad \sum_{j=1}^{n} P_j x_j && (1)\\
&Subject\, to \quad \sum_{j=1}^{n} w_j x_j \leq W\\
&\qquad x_j \in \{0,1\} \quad \forall j \in \{1, \; . \; . \; , n\}
\end{aligned}$$

Bounded Knapsack Problem

Bounded knapsack problem restricts the number of copies of each kind of item to be taken to a minimum integer value (say C_i). One common variant is that each item can be chosen multiple times. The bounded knapsack problem specifies, for each item *j*, an upper bound u_j (which may be a positive integer, or infinity) on the number of times item *j* can be selected:

$$\begin{aligned}
&Maximize \quad \sum_{j=1}^{n} P_j x_j\\
&Subject\, to \quad \sum_{j=1}^{n} w_j x_j \leq W && (2)\\
&\qquad u_j \leq x_j \leq 0 \; x_j \; \text{int}\, eger \;\; for\; all\;\; j
\end{aligned}$$

Unbounded Knapsack Problem

The Unbounded knapsack problem (sometimes called the **integer knapsack problem**) does not put restriction on the number of copies of each kind of item to be selected.

$$\begin{aligned} &Maximize \quad \sum_{j=1}^{n} P_j x_j \\ &Subject\, to \quad \sum_{j=1}^{n} w_j x_j \leq W \qquad (3) \\ &\qquad x_j \geq 0 \; x_j \; \text{int}\, eger \; for\; all \; j \end{aligned}$$

Multiple Choice Knapsack Problems

Multiple-choice knapsack problem is another variant of a knapsack problem in which the items are subdivided into classes and exactly one item must be taken from these classes.

$$\begin{aligned} &Maximize \quad \sum_{i=1}^{k} \sum_{j \in N_i}^{n} P_{i\,j} x_{i\,j} \\ &Subject\, to \quad \sum_{i=1}^{k} \sum_{i \in N} w_j x_{i\,j} \leq W \qquad (4) \\ &\qquad \sum_{j \in N_i}^{m} x_{ij} = 1 \quad for\; all\; 1 \leq i \leq k \\ &\qquad x_{ij} \in \{0, 1\} \quad for\; all\; 1 \leq i \leq k \; and\; all \;\; j \in N_i \end{aligned}$$

Subset Sum Problem

If for each item the profits and weights are identical, we get the subset sum problem (often the corresponding decision problem is given instead):

$$\begin{aligned} &Maximize \quad \sum_{j=1}^{n} P_j x_j \\ &Subject\, to \quad \sum_{j=1}^{n} P_j x_j \leq W \qquad (5) \\ &\qquad x_j \in \{0, 1\} \end{aligned}$$

Multiple Knapsack Problems

If we have *n* items and *m* knapsacks with capacities, we get the multiple knapsack problems:

$$\begin{aligned} &Maximize \quad \sum_{i=1}^{m} \sum_{j=1}^{n} P_{ij} x_{ij} \\ &Subject\, to \quad \sum_{j=1}^{n} w_j x_{i\,j} \leq W_i \quad for\; 1 \leq i \leq m \qquad (6) \\ &\qquad \sum_{i=1}^{m} x_{ij} \leq 1 \; \text{int}\, eger \; for\; all\; 1 \leq \; j \leq n \\ &\qquad x_{ij} \in \{0, 1\} \quad for\; all\; 1 \leq \; j \leq n \; and\; all\; 1 \leq \; j \leq m \end{aligned}$$

2.2 STRATEGIES/TECHNIQUES FOR RESOURCE ALLOCATION

In the literature, a number of strategies /techniques for resource allocation on the grid and other related distributed systems have been well discussed. Notable among these strategies/techniques are as discussed below:

i. **First-come-first-served**

The First-Come-First-Served (FCFS) resource allocation model is a simple and straightforward technique where resources are allocated to users or tasks in the order they arrive. In the context of cloud computing, this model can be applied to allocate resources such as virtual machines, storage, or network bandwidth to cloud users based on their request arrival time. However, FCFS may not always be the most efficient approach, especially in scenarios with varying resource demands and priorities.

ii. **Backfilling**

Backfilling is an advanced resource allocation model used in cloud computing and parallel computing environments to enhance resource utilization and reduce job waiting times. It is commonly applied in systems where jobs are submitted to a scheduling queue, and resources are allocated to jobs based on their requirements and availability. Backfilling allows small jobs to be scheduled ahead of larger ones, even if they arrive later, as long as they can execute without delaying the execution of the larger jobs. (Eielson, D. and Weil, A., 1998). If the resources for the job at the head of the queue are expected to become available at time t, then the EASY backfilling strategy allows a job with expected length l, where $l < t$, to be executed immediately, rather than waiting, assuming its required resources are available. This effectively fills the idle bubbles in the schedule and results in a shorter overall completion time.

iii. **Makespan Minimizing Heuristics**

Makespan-minimizing heuristics are resource allocation models used in cloud computing to optimize the overall completion time of a set of tasks or jobs. The goal is to minimize the makespan, which represents the time required to complete all tasks from the start to the end. These heuristics are often applied to scheduling problems to efficiently allocate resources in cloud environments and achieve better performance.

The makespan of a set of tasks is the time taken between the start of the first task and the completion of the last task. According to (Ibarra, O. H., and Kim, C. E., 1977), Finding the minimal makespan is non deterministic polynomial-time hard (NP-hard). As such, research in operations and heterogeneous computing has resulted in a number of heuristic approaches to minimizing the makespan (Braun, et. al., (2001).

The following are some common makespan-minimizing heuristics for resource allocation in cloud computing:

1. **Earliest Finish Time (EFT):** The EFT heuristic prioritizes tasks based on their estimated finish time. It schedules tasks with the earliest expected finish time first, aiming to reduce the makespan by completing shorter tasks early, which can lead to resource availability for subsequent tasks (Graham, et. al, 1979)

2. **Shortest Processing Time (SPT):** The SPT heuristic prioritizes tasks based on their processing time. It schedules shorter tasks first to reduce the makespan and free up resources for longer tasks later (Smith, 1956).

3. **Longest Processing Time (LPT):** The LPT heuristic prioritizes tasks based on their processing time but in the opposite order of SPT. It schedules longer tasks first, aiming to minimize the makespan by allocating more resources to complete the longer tasks early (Coffman, E. G., Garey, M. R., & Johnson, D. S., 1978).

4. **Critical Path Method (CPM):** CPM is used to identify the critical path in a project, which represents the longest sequence of tasks that determine the project's duration. Allocating resources optimally to tasks on the critical path helps reduce the overall project makespan.(Kelley, J., & Walker, R.,1959).

5. **Genetic Algorithms (GA):** Genetic algorithms are optimization techniques inspired by the process of natural selection. In cloud resource allocation, GA can be used to find near-optimal solutions for minimizing makespan by iteratively evolving a population of potential resource allocations (Goldberg, 1989).

6. **Ant Colony Optimization (ACO):** ACO is a metaheuristic inspired by the foraging behavior of ants. In cloud resource allocation, ACO can be applied to find efficient resource allocations that minimize the makespan

7. .**Min-Min:** The min-min heuristic assigns the shortest task to the resource that will execute it the fastest (Ibarra, O. H., and Kim, C. E., 1977), and (Braun, T. D. etal., 2001). The motivation behind min-min is that it seeks to cause the smallest change possible to the resources during the assignment of each task.

8. **Max-Min:** The max-min heuristic is similar to min-min, but instead prefers to run the longest tasks early. In the situation where there are many short tasks and few long tasks, then the min-min heuristic results in low efficiency and sub-optimal overall completion time. Because min-min postpones the long tasks, they are left running on a small proportion of the processors, leaving the remaining processors to be idle. The overall completion time is therefore dramatically increased for such sets of tasks.

9. Many other heuristics exist for the minimization of the makespan (Braun, T. D. etal., 2001). Some of these include the suffrage techniques (Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., and Freund., R. F., (Nov.,1999), (Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F., 2000), synthetic annealing, and others.

iv. Gang Scheduling

Gang scheduling is a resource allocation model used in cloud computing and distributed systems to improve the performance and efficiency of parallel applications. It involves scheduling multiple related tasks or processes as a group, called a "gang," and ensuring that they are executed simultaneously on different processors or nodes. This approach is particularly beneficial for parallel applications that require strong synchronization and communication among tasks
Ousterhout, J. K., (1982) made use of Gang scheduling strategy to improve interactivity in shared parallel computer systems by using time slicing. Time slicing is a technique commonly used in personal computers to provide the illusion of multitasking on a single processor system. Because each task is limited to its given time slice, gang scheduling has the added benefit of limiting the negative effects of bad scheduling decisions.

v. Matchmaking

The matchmaking resource allocation model in cloud computing is a technique used to efficiently match cloud service requests from users with available cloud resources. It involves finding suitable resource providers that meet the specific requirements of the users' requests. This approach aims to optimize resource allocation and enhance user satisfaction by ensuring that users get the resources they need while providers can efficiently utilize their resources.

According to (Raman, R., Livny, M., and Solomon, M., 1998), (Tuecke,S., 2001), and (Frey,J., et.al., 2001), Matchmaking is the resource allocation framework employed by Condorbatch system website." http://www.cs.wisc.edu/condor/.128.

In this strategy, each consumer (job) and provider (resource) provides a classified advertisement (ClassAd), which contains the entity's description and characteristics. In each ClassAd, the entities provide Requirements and Rank expressions: Requirements is a boolean expression used to specify constraints (e.g. that a job has for potential resources, or vice versa); and Rank is real number used to describe the preferences (e.g. of a resource for jobs with certain characteristics, or vice versa.), The Rank and Requirements are functions of the metadata contained in the job and resource ClassAds. During the matchmaking algorithm, the Rank and Requirements expressions are evaluated for each task-resource combination; the match which has a satisfied Requirements expression (Requirements==TRUE), and having the largest Rank is selected.

Matchmaking is a practical solution to the problem of specifying allocation preferences. However, matchmaking itself does not result in any resource allocation optimizations; rather, its value is that it provides a framework within which such optimization heuristics can be implemented. Further, the Rank mechanism is somewhat not flexible; as it does not easily allow for access to the state of the scheduler itself (e.g. the implementation of a round-robin placement algorithm is non-trivial).

vi. The Economic Grid and Market-based Resource Allocations

Buyya, R., (2002) presented Economic-based Distributed Resource Management and Scheduling for Grid Computing as an alternative and notable variation on the computational grid. The market-based allocation strategies of Buyya were first developed for the Nimrod-G resource broker and are now available in the tools provided by the Grid bus project (Buyya, R., and Venugopal, S., 2004). In this system, the grid is implemented with economical considerations, :resources have associated costs, and users pay to use them. Buyya showed how to apply various market-based pricing models to the grid, including flat pricing, auction style pricing, timing models (peak/off-peak), supply/demand models, and others. The Nimrod-G resource broker uses scheduling algorithms which minimize the total cost subject to timing constraints, or minimize the total time subject to budget constraints. The cost optimizing algorithm sorts the available resources by their cost, and assigns tasks to the cheapest resources first while ensuring their deadline is not exceeded. The time optimizing algorithm sorts the resources by the estimated time until they become available, and assigns tasks to the resource with the soonest availability while ensuring their budget is not exceeded.

The Economic-based Distributed Resource Management and Scheduling for Cloud Computing is Market-based resource allocation models involve creating a virtual marketplace where cloud providers and users can interact and negotiate resource allocations. Users can place bids for the desired resources, and providers can accept or reject these bids based on their pricing strategies and resource availability. This approach introduces a competitive environment that optimizes resource allocation based on supply and demand.

The Economic Grid and Market-based resource allocation models in cloud computing are inspired by economic principles and market mechanisms. These models aim to allocate cloud resources efficiently by introducing pricing, bidding, and negotiation strategies similar to those found in real-world economic markets. The idea is to create a dynamic environment where cloud providers and users can interact, negotiate resource prices, and make resource allocation decisions based on supply and demand

The Economic Grid is a resource allocation model that treats cloud resources as commodities traded in an economic marketplace. Cloud providers offer resources at varying prices based on demand and availability, and users can dynamically select resources based on their budget and requirements. Providers adjust resource prices based on utilization and demand, incentivizing users to balance their resource usage with the available supply.

(Sherwani, J., Ali, N., Lotia, N. Hayat, Z. and Buyya, R., 2004) and (Yeo, C. S., and Buyya, R., (Mar., 2007) incorporate a concept of utility into the scheduling decisions. In this system, the utility of a task is related to the budget allocated to the job, the deadline of the task, and any service-level agreements (SLA's) the task might have with resources. Libra schedules by assigning each task to a fraction of a processor in such a way that the task is likely to meet its budget and deadline constraints. Though these techniques are not directly applicable to the non-economic grid, they represent a significant contribution to computational grid research.

3. KNAPSACK PROBLE FORMULATION
`

The following is a general outline of how the Generalized Knapsack Algorithm can be applied to cloud resource allocation:

i. **Problem Setup**: Define the available resources and the resource demands of different tasks or applications. For example, you might have a set of Virtual Machines (VMs) with specific CPU, RAM, and storage capacities, and you also have a set of tasks/applications with their resource requirements.
ii. **Objective Function**: Formulate an objective function that you want to optimize. In cloud computing, this could be maximizing resource utilization, minimizing costs, or optimizing performance metrics like response time or throughput.
iii. **Constraints**: Set constraints that must be satisfied during resource allocation. For instance, you may have a budget constraint, where the total cost of resource allocation should not exceed a certain limit, or a resource capacity constraint, where the allocated resources for each VM should not exceed its capacity.

iv. **Apply the Generalized Knapsack Algorithm**: The Generalized Knapsack Algorithm is a variant of the classic 0/1 Knapsack Problem, which allows for fractional resource allocation. This means that instead of deciding whether to include an item (task or application) completely or exclude it, you can allocate fractions of the item's resource demands.
v. **Optimization**: Use the Generalized Knapsack Algorithm to find the best combination of resource allocations that maximizes the objective function while satisfying all the constraints.
vi. **Resource Allocation**: Once the algorithm has been applied, it will provide you with the optimal resource allocation strategy. You can then allocate the resources to the respective tasks or VMs accordingly.

3.1 THE GENERALIZE KNAPSACK PROBLEM FORMULATION

Given *n set of* tasks to be done in the Cloud and *m Cloud* resources, with P_{rj} as the profit associated with assigning task *j* to resources *r*, w_{rj} as the weight of assigning task *j* to resources *r*, and c_{rj} as the capacity of resource *r*, assign each task *j* to exactly one resource *r*, not exceeding resource capacities. Defining $x_{rj} = 1$ if task *j* is assigned to resource *r* = 0 otherwise, the *Generalize Knapsack Model* can be formulated as:

$$Maximize \quad z = \sum_{r=1}^{m}\sum_{j=1}^{n} p_{rj}x_{rj} \qquad (1)$$

Subject to

$$\sum_{j=1}^{n} w_{rj}x_{rj} \le c_r \; r \in M = \{1, . \; . \; . \; m\}\{respectknapsackcapacities\} \qquad (2)$$

$$\sum_{j=1}^{n} x_{rj} = 1 \quad j \in N = \{1, . \; . \; . \; n\} \quad \{all\, tasks\ assigned\} \qquad (3)$$

$$x_{ij} \in \{0,1\} \quad for\ all\ i \in M\ and\ j \in N$$

where

$$x_{ij} = \begin{cases} 1 & if\ task\ j\ is\ allocated\, resource\, i \\ 0 & otherwise \end{cases}$$

3.2 SOLUTION DESIGN APPROACH

This section presents design for implementing the presented Knapsack Model by adopting Dynamic Programming solution approach as follows:

i. **Problem Setup**: Given n items, each with a value (v[i]) and weight (w[i]), and a knapsack capacity (C).
ii. **Dynamic Programming Table**: Create a 2D array, dp, with dimensions (n+1) x (C+1). The table will be used to store the maximum value that can be obtained for different sub problems.
iii. **Initialization**: Initialize the first row and first column of the dp table with zeros, as they represent the cases of having no items or zero knapsack capacity (i.e., the value of the knapsack is zero).
iv. **Dynamic Programming Algorithm**:
    a. Iterate over each item (i) from 1 to n.
    b. For each item i, iterate over each possible capacity (j) from 1 to C.
    c. For each (i, j) pair in the dp table, calculate the maximum value that can be achieved considering two options: a. Exclude the current item: dp[i][j] = dp[i-1][j] b. Include the current item (either partially or fully):
        dp[i][j] = max(dp[i][j], dp[i-1][j-w[i]] + v[i])

The recurrence relation is based on choosing the maximum value between the case of excluding the current item (value obtained from the row above) and the case of including the current item (value obtained from the same row but with the remaining capacity after including the current item).

v. **Backtracking (Optional)**: If needed, you can use backtracking to find the items that were included in the optimal solution. Starting from the bottom-right corner of the dp table, follow the decisions that led to the maximum value and add the corresponding items to the knapsack.

vi. **Final Result**: The maximum value that can be obtained from the Generalized Knapsack Problem is stored in dp[n][C].

The dynamic programming approach ensures that each sub problem is solved only once, and the results are stored for future reference, preventing redundant computations. It provides an efficient solution to the Generalized Knapsack Problem, even for relatively large instances.

Keep in mind that the dynamic programming approach assumes that the values and weights are non-negative integers. If there are fractional weights or values, additional steps are required to handle them properly.

Step 1: Decompose the problem into smaller problems.

The researcher constructed an array $Z[0 \quad . \quad . \quad . \quad m, 0 \quad . \quad . \quad . \quad n]$, for $0 \le r \le m,\ 0 \le j \le n$, the entry $Z[r, j]$ will store d maximum (combined) total profit of items {1,2,3…,j} of combined size at most C_r. If we compute all the entries of this array, then the array entry $Z[M, N]$ will contain the maximum total profit of items that can be assigned into the knapsack/resources, that is, the sub-solution to our problem.

Step 2: Recursively define the value of an optimal solution in terms of solutions to smaller problems.

$$\begin{aligned} &Initial setting: \quad set\ Z[r,0] = 0,\ for\ 0 \le r \le m, \qquad no\ item \\ &\qquad\qquad Z[r, j] = -\infty, \quad for\ r < 0 \qquad illegalcase \end{aligned}$$

Recursive Step: Used

$$\begin{aligned} &Z[r, j] = \max(Z[r, j-1], P[r, j] + Z[r-1, j-1]) \quad for\ 1 \le j \le n, 0 \le r \le m\ if\ and\ if \\ &w[i][j] \quad < C[r]\ and\ C[r] \quad = \quad C[r] - W[r]; \end{aligned}$$

Step 3: Bottom-up computing $Z[r, j]$ using iteration. **Bottom:**

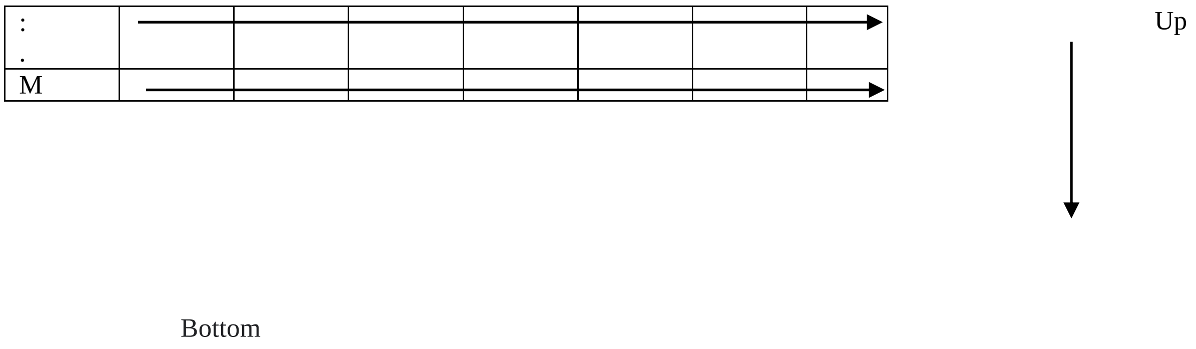
$Z[r,0] = 0$ for all

$0 \le r \le m$ Computing of table using :

$Z[r, j] = \max(Z[r, j-1], P[r, j] + Z[r-1, j-1])$ row by row

Table 3.1 Tabular Representation of Dynamic Programming Concept

Z[r,j]	J = 0	1	2	3	…	…	N
r = 0	0	0	0	0	…	…	0
1	⟶	⟶	⟶	⟶	⟶	⟶	⟶
2	⟶	⟶	⟶	⟶	⟶	⟶	⟶



4. IMPLEMENTATION

This section presents the implementation of the Dynamic programming approach to solving the generalized Knapsack Scheduling algorithm using Java programming language for the implementation.

4.1 THEORETICAL EXPERIMENT

The algorithm stated above was experimented with the example given below.

Example1: Given, the value of n (set of tasks) is 4; m (set of Cloud resources) is 5

Set of Cloud resources) =
C_r (Capacity of the Cloud resources) =

R	1	2	3	4	5
C_r	50	40	45	30	60

W_{rj} (weight of task j if assigned to resource r) P_{rj} (profit of task j if assigned to resource r),

W_{rj}	j = 1	2	3	n = 4
r = 1	5	6	7	8
2	3	12	4	5
3	6	7	9	10
4	4	8	6	2
m = 5	2	11	14	13

W_{rj}	j = 1	2	3	n = 4
r = 1	20	15	60	70
2	50	40	30	20
3	10	15	20	25
4	15	40	55	50
m = 5	60	65	10	5

Compute the optimal solution

Applying the Dynamic programming approach to the problem stated in the example above the researcher constructed an array $Z[0...0....4],\ 0 \le r \le m \; and \; 0 \le j \le n$ such that the maximum of the profits of tasks {1, 2, 3, 4}and capacity C_r

Our interest is to find

$$Z[r,j] = \max\left(Z[r,j-1], P[r,j] + Z[r-1,j-1]\right) \quad for\, 1 \le j \le n,\, 0 \le r \le m\ if\ and\ if$$
$$w[i][j] \quad < C[r]\, and\, C[r] \quad = \quad C[r] - W[r];$$

That is we must be sure that the W[i][j] is not greater than capacityC[r]

Case when: r= 0, j = 1 , Z[0,j] = 0 for all j member of N

r = 1, j = 1, Z[1,1] = max { 0, 20 + 0} > 0? = 20

r = 1, j = 2, Z[1,2] = max { 0, 15 + 0} > 20? = 20

r = 1, j = 3, Z[1,3] = max {0, 60 + 0} > 20? = 60

r = 1, j = 4, Z[1,4] = max {0, 70 + 0} > 60? = 70

r = 2, j = 1, Z[2,1] = max {0, 50 + 0} > 0? = 50

r = 2, j = 2, Z[2,2] = max {0, 40 + 20} > 50? = 60

r = 2, j = 3, Z[2,3] = max {0, 30 + 20} > 60? = 60

r = 2, j = 3, Z[2,4] = max {0, 20 + 60} > 50? = 80

If we continue in that manner,

Also we obtain the corresponding values

we obtain a table as shown below for Z[r,j] for X_{rj} as shown in the table below;

Z[r,j]	**j = 0**	**1**	**2**	**3**	**4**
r = 0	**0**	**0**	**0**	**0**	**0**
1	**0**	**20**	**20**	**60**	**70**
2	**0**	**50**	**60**	**60**	**80**
3	**0**	**10**	**65**	**80**	**85**
4	**0**	**45**	**50**	**120**	**130**
5	**0**	**60**	**110**	**110**	**125**

The valueX_{rj} = 1 if the task j is assign to the resource r.

X_{rj}	**j = 0**	**1**	**2**	**3**	**4**
r = 0	**0**	**0**	**0**	**0**	**0**
1	**0**	**1**	**0**	**1**	**1**
2	**0**	**1**	**1**	**0**	**1**
3	**0**	**1**	**1**	**1**	**1**
4	**0**	**1**	**1**	**1**	**1**
5	**0**	**1**	**1**	**0**	**1**

The optimal solution for Z[r,j] is obviously 130.

4.2 COMPUTATIONAL EXPERIMENT

The result of the computational experiment is as shown below:

Read in the N set of tasks to be done: 4

Read in the m Grid of knapsacks or resources to store items: 5

The capacity of knapsack: 60 50 55 40 45

The weight of assigning task to knapsack:

The cost of assigning task to knapsack: The result table for the maximum profit computed is:

20	15	60	70
50	40	30	20
10	15	20	25
15	40	55	50
60	65	10	5

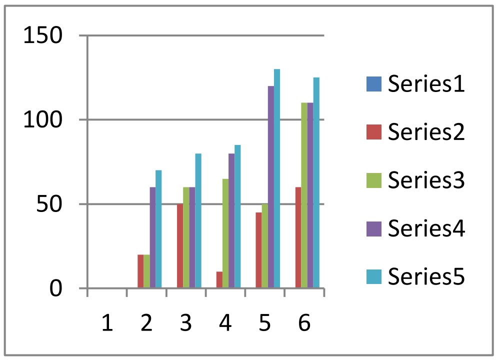
0	0	0	0	0
0	20	20	60	70
0	50	60	60	80
0	10	65	80	85
0	45	50	120	130
0	60	110	110	125

The result table for the X(r,j) is:

0	0	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
0	1	1	1	1
0	1	1	0	1

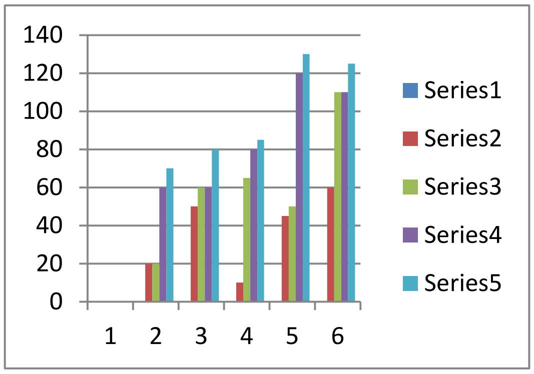
The maximum total computed Profit of 6 items that can be assigned to the 5 knapsack is: 225

Optimal Solution is at Position: Z[4,4]



4.2: Graphical representation of the output for theoretical experiment

4.1:



Graphical representation of the output for Computational experiment

4.3 DISCUSSION OF RESULT

Graph 4.1 and Graph 4.2 show the results of the theoretical computation and the experimental computational respectively. The results show that the presented Knapsack Model compared favourably with the theoretical model. Thus the Knapsack Model is appropriate for allocating Cloud Computing resources.

5. SUMMARY AND CONCLUSION

5.1 SUMMARY

The paper presents a novel approach, the Generalized Knapsack Algorithm (GKA), to address the resource allocation challenge in cloud computing. Efficient resource allocation is crucial to optimize cloud performance and cost-effectiveness. The GKA aims to maximize computing resources, memory, and bandwidth utilization while considering various constraints. The algorithm's performance is evaluated by comparing the theoretical and the computational results of the implemented model using diverse test cases. The results show that the GKA outperforms existing resource allocation methods in efficiency and scalability. This approach offers a promising solution for enhancing resource allocation strategies in cloud environments, improving service delivery for cloud users. The paper contributes to cloud computing optimization research, with practical implications for cloud service providers and users seeking effective resource management solutions.

5.2 CONCLUSION

In conclusion, this paper has introduced the Generalized Knapsack Algorithm (GKA) as a novel and efficient approach for resource allocation in cloud computing. Through simulated experiments and diverse test cases, the GKA demonstrated superior performance, surpassing existing methods in terms of resource utilization and scalability. Its potential to enhance cloud service delivery and cost-effectiveness makes it a valuable contribution to the field. While acknowledging certain limitations, the GKA's success warrants further exploration and refinement. Looking ahead, future research could investigate its applicability in multi-cloud and edge computing environments or explore hybrid approaches to improve its adaptability. As cloud computing continues to evolve, the GKA offers a promising solution for meeting the resource allocation challenges of the future.

.

6. REFERENCES

Feitelson, D. & Weil, A., (1998): “Utilization and predictability in scheduling the ibmsp2 with Backfilling,” in Parallel Processing Symposium,.

Ibarra, O. H., & Kim, C. E., (1977): “Heuristic algorithms for scheduling independent tasks on non identical processors,” J. ACM, vol. 24, no. 2, pp. 280–289.

Braun, T. D. Siegel, H. J., Beck, N., B¨ol¨oni, L. L., Maheswaran, M. Reuther, A. I., Robertson, M. D., Theys, J. P. Yao, B. Hensgen, D. and Freund, R. F., (2001): “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,” J. Parallel Distrib. Comput. vol. 61, no. 6, pp. 810–837.

Maheswaran,M., Ali,S. Siegel,H. J., Hensgen,D., and Freund,R. F., (Nov. 1999):Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,Journal of Parallel and Distributed Computing, vol. 59, pp. 671–680.

Casanova,H., Legrand,A., Zagorodnov,D., and Berman,F., (2000): Heuristics forscheduling parameter sweep applications in grid environments,” in Proceedingsof the 9th Heterogeneous Computing Systems Workshop (HCW 2000), (Cancun,Mexico), IEEE CS Press, Los Alamitos, CA, USA.

Ousterhout, J. K., (Oct., 1982): Scheduling techniques for concurrent systems, in Proceedings of The 3rd International Conference on Distributed Computer Systems, pp. 22–30.

Raman,R., Livny,M. and Solomon,M., (July, 1998): Matchmaking: distributed resource Management for high throughput computing,” in The Seventh International Symposium on High Performance Distributed Computing, pp. 140–146.

.

“Condor batch system website.” http://www.cs.wisc.edu/condor/.128

Frey,J., Tannenbaum,T., Livny,M., Foster,I., and Tuecke,S., (Aug. 2001): Condor-g: A Computation management agent for multi-institutional grids, in Proceedings of theTenth International Symposium on High Performance Distributed Computing,IEEE Press,.

Buyya, R.., (2002): Economic-based Distributed Resource Management and Scheduling for Grid Computing.PhD thesis, Monash University, Melbourne, Australia,.

Buyya,R., Abramson,D., and Giddy,J., (2000): Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," in The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region.

Buyya, R., and Venugopal,S., (Apr., 2004): The gridbus toolkit for service oriented grid and utility computing: an overview and status report, in 1st IEEE International Workshop on Grid Economics and Business Models, pp. 19–66.

Sherwani, J., Ali, N., Lotia, N. Hayat, Z. and Buyya, R., (May, 2004): Libra: A computational economy-based job scheduling system for clusters, Software: Practice and Experience, vol. 34, pp. 573–590.

Yeo, C. S., and Buyya, R., (Mar., 2007): Integrated risk analysis for a commercial computing service," in Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007).

Dyer, M. E., Kayal, N., and Walker, J., (1984): A branch and bound algorithm for solving the multiple choice knapsack problem, Journal of Computational and Applied Mathematics, II. 231 – 249.

Elghoneimy, E., Bouhali, O., & Alnuweiri, H., (2012). Resource allocation andScheduling in cloud computing. *International Conference on Networking and Communications (ICNC)*. Pp. 309-314.

(Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., and Freund., R. F., (Nov.,1999)

(Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F., 2000),

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). "Optimization and approximation in deterministic sequencing and scheduling: A survey." Annals of Discrete Mathematics, 5, 287-326.

Smith, W. E. (1956). "Various optimizers for single-stage production." Naval Research Logistics Quarterly, 3(1-2), 59-66.

Coffman, E. G., Garey, M. R., & Johnson, D. S. (1978). "Approximation algorithms for bin-packing: An updated survey." In Algorithms and Complexity (pp. 46-93). Springer, Berlin, Heidelberg.

Kelley, J., & Walker, R. (1959). "Critical path planning and scheduling: Mathematical basis." Operations Research, 7(4), 414-425.

Goldberg, D. E. (1989). "Genetic Algorithms in Search, Optimization and Machine Learning." Addison-Wesley Professional.

Braun, T. D. Siegel, H. J., Beck, N., B¨ol¨oni, L. L., Maheswaran, M. Reuther, A. I., Robertson, M. D., Theys, J. P. Yao, B. Hensgen, D. and Freund, R. F., (2001)..

(Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., and Freund., R. F., (Nov.,1999), (Casanova, H., Legrand, A., Zagorodnov, D. and Berman, F., 2000),