

# Cost and Performance Optimization for Application Deployment & Scaling

## Table des matières

1.1 Context and Motivation.....	1
1.2 Formal Definition of the Application Deployment Problem.....	2
1.3 Computational Complexity.....	2
1.4 Core Intuition Behind the Proposed Approaches.....	3
2. Problem-Solving Approaches.....	3
2.1 Formal Model.....	3
2.2 Greedy Placement and Repair Mechanism.....	4
2.3 Hill Climbing Approach.....	5
2.4 Genetic Algorithm Approach.....	6
3. Evaluation.....	6
3.1 Experimental Setup.....	6
3.2 Performance Comparison.....	7
3.3 Scalability Study.....	8
3.4 Critical Analysis.....	9
4. Conclusion.....	10

## 1. Introduction

### 1.1 Context and Motivation

Cloud computing is widely used to deploy modern distributed applications composed of multiple interacting components. These applications must handle fluctuating workloads while ensuring performance, reliability, and cost efficiency.

Cloud providers offer a large variety of virtual machine (VM) types with different capacities and pricing models, making resource provisioning highly flexible. However, poor provisioning decisions can lead either to unnecessary costs or to system instability.

Beyond technical considerations, inefficient resource allocation also raises important economic and environmental concerns. Over-provisioning virtual machines results in higher operational costs, particularly due to CPU and memory pricing, and contributes to increased energy consumption and carbon footprint in data centers.

The motivation of this project is therefore to optimize application deployment and scaling decisions by minimizing infrastructure costs while guaranteeing stable application behavior under varying workloads.

## 1.2 Formal Definition of the Application Deployment Problem

The application deployment problem studied in this project considers an application composed of multiple components, each receiving a workload that must be processed reliably. To handle incoming requests, each component can be replicated multiple times.

These replicas must be deployed on a set of available virtual machine types, each characterized by limited CPU and memory capacities and an associated cost. A single virtual machine can host several replicas as long as its resource constraints are respected.

The objective is to determine the number of replicas, the number and types of virtual machines, and the placement of replicas such that the application remains stable while minimizing the total infrastructure cost.

## 1.3 Computational Complexity

The application deployment and scaling problem is computationally hard due to its combinatorial nature. It combines several well-known NP-hard problems, including resource allocation, bin packing, and cost optimization. In particular, placing replicas on virtual machines under CPU and memory constraints corresponds to a multi-dimensional bin packing problem, while selecting the number and types of virtual machines relates to knapsack and facility location problems.

Moreover, decisions regarding the number of replicas directly affect resource requirements and placement feasibility, creating strong interdependencies between variables. As the number of components and available VM types increases, the number of possible configurations grows exponentially. Consequently, exact optimization methods become impractical for realistic problem sizes, motivating the use of heuristic and metaheuristic approaches.

## 1.4 Core Intuition Behind the Proposed Approaches

The proposed approaches are based on the intuition that efficient solutions can be obtained by combining global exploration with local feasibility enforcement. Instead of attempting exact optimization, the problem is addressed through heuristic search strategies that iteratively construct and improve feasible solutions.

Two complementary methods are investigated. Hill Climbing focuses on local search by progressively improving a single solution, providing fast convergence but limited global exploration. In contrast, the Genetic Algorithm explores the solution space globally through population-based search, allowing it to escape local optima at the cost of higher computational effort. Both approaches rely on a greedy placement strategy and a repair mechanism to ensure feasibility with respect to stability and resource constraints, enabling a fair and effective comparison between local and global optimization techniques.

## 2. Problem-Solving Approaches

### 2.1 Formal Model

This section presents the formal mathematical model used to represent the application deployment and scaling problem.

#### 2.1.1 Decision Variables

Let :

- $r_i$  denote the number of replicas deployed for component  $i$ .
- $x_j$  denote the number of virtual machines (VMs) of type  $j$  provisioned.
- $y_{ij}$  denote the number of replicas of component  $i$  placed on VMs of type  $j$ .

#### 2.1.2 Objective Function

The objective is to minimize the total hourly cost of the infrastructure :

$$\min Z = \sum_{j \in J} x_j \cdot Cost_j$$

where  $Cost_j$  is the hourly cost of a VM of type  $j$ . This objective captures the economic aspect of the deployment while abstracting away operational details.

### 2.1.3 Constraints

The model is subject to the following constraints:

Stability constraints

Each component must be able to process its incoming workload:

$$\forall i \in I, r_i \cdot \mu_i \geq \lambda_i$$

CPU capacity constraints

$$\forall j \in J, \sum_{i \in I} y_{ij} \cdot ReqCPU_i \leq x_j \cdot CPU_j$$

Memory capacity constraints

$$\forall j \in J, \sum_{i \in I} y_{ij} \cdot ReqRAM_i \leq x_j \cdot RAM_j$$

Replica placement constraints

$$\forall i \in I, \sum_{j \in J} y_{ij} = r_i$$

VM availability constraints

$$\forall j \in J, 0 \leq x_j \leq Avail_j$$

All decision variables are non-negative integers.

## 2.2 Greedy Placement and Repair Mechanism

To ensure feasibility of candidate solutions, a greedy placement strategy combined with a repair mechanism is employed.

The greedy placement algorithm assigns replicas to VM types in increasing order of cost. For each VM type, replicas are placed as long as sufficient CPU and memory resources remain available. This strategy ensures a fast and deterministic construction of a feasible placement whenever possible.

The repair mechanism is used to correct infeasible solutions generated by the optimization algorithms. It enforces the minimum number of replicas required for stability, respects VM availability constraints, and incrementally adds low-cost VMs when placement is

impossible. If a solution cannot be repaired after a limited number of attempts, it is discarded.

This combination plays a critical role by guaranteeing feasibility while keeping the optimization process efficient.

## **2.3 Hill Climbing Approach**

### **2.3.1 Principle**

Hill Climbing is a local search algorithm that starts from an initial feasible solution and iteratively improves it by exploring its neighborhood. At each step, a small modification is applied to the current solution, such as adding or removing a replica or a VM.

If the modified solution yields a lower cost while remaining feasible, it replaces the current solution.

### **2.3.2 Algorithm Description**

The Hill Climbing algorithm follows these steps :

1. Generate an initial feasible solution.
2. Apply a local modification to the solution.
3. Repair the solution if necessary to ensure feasibility.
4. Accept the new solution if it improves the objective function.
5. Repeat until no further improvement is possible or a stopping criterion is met.

### **2.3.3 Strengths and Limitations**

Hill Climbing is simple to implement and converges quickly to a locally optimal solution. However, it explores only a limited portion of the solution space and may become trapped in local minima, preventing it from finding globally optimal solutions.

## 2.4 Genetic Algorithm Approach

### 2.4.1 Individual Representation

In the Genetic Algorithm, each individual represents a complete deployment configuration encoded by the vectors  $r$  and  $x$ . The placement matrix  $y$  is computed using the greedy placement algorithm.

### 2.4.2 Genetic Operators

The algorithm uses :

- Tournament selection to favor low-cost solutions.
- One-point crossover to combine parent solutions.
- Mutation to introduce small random variations in replicas and VM counts.

### 2.4.3 Feasibility Handling

After crossover and mutation, each individual is repaired using the repair mechanism. This guarantees that all individuals in the population satisfy stability and resource constraints before evaluation.

### 2.4.4 Parameter Tuning

A parameter sensitivity analysis was conducted to study the impact of population size, number of generations, and mutation rate on the performance of the Genetic Algorithm. Each configuration was executed multiple times to account for the stochastic nature of the algorithm.

The evaluation focuses on solution quality, stability, and convergence behavior.

The results of this analysis are presented and discussed in Section 3.

## 3. Evaluation

### 3.1 Experimental Setup

The evaluation was conducted using two datasets. The first dataset specifies the workload of each application component in terms of request arrival rates. The second dataset contains the catalogue of available virtual machine (VM) types, including CPU capacity, memory capacity, hourly cost, and availability constraints.

Two optimization approaches were evaluated: Hill Climbing (HC) and Genetic Algorithm (GA). For the GA, the parameters were selected based on a prior sensitivity analysis, with a

population size of 80, 200 generations, and a mutation rate of 0.2. Due to the stochastic nature of the GA, results were averaged over multiple runs.

The evaluation focuses on the following metrics :

- Total infrastructure cost,
- Execution time,
- Robustness under increasing workload.

### 3.2 Performance Comparison

Table 1 reports the infrastructure cost and execution time obtained by Hill Climbing (HC) and the Genetic Algorithm (GA) under increasing workload levels. The workload scaling is represented by the number of applications, with higher values corresponding to higher demand and increased resource requirements.

**Table 1 – Performance Comparison under Increasing Workload**

Applications	HC Cost	GA Cost	HC Time	GA Time
1	1.7648	1.5722	0.83 s	5.89 s
2	2.6472	2.6472	0.93 s	5.98 s
5	7.0592	6.4832	0.78 s	5.99 s
10	14.298	13.1176	0.92 s	6.21 s

Hill Climbing consistently produces feasible solutions in less than one second across all scenarios, confirming its efficiency as a local search method. Its execution time remains stable as workload increases, making it suitable for time-critical deployment decisions. However, the solutions generated by HC tend to rely on homogeneous VM configurations, typically using a small number of large VM instances. While this strategy simplifies placement, it often results in resource over-provisioning and higher infrastructure cost.

In contrast, the Genetic Algorithm requires a higher but stable execution time of approximately six seconds per run due to its population-based nature. From a cost perspective, GA consistently achieves equal or lower infrastructure costs than HC. For low workloads (Applications = 1), this difference is already observable: although both approaches deploy the same number of replicas [5, 9, 6, 5], Hill Climbing selects two VMs of

type T10, resulting in a total cost of 1.7648, whereas GA identifies a heterogeneous configuration composed of one T00, one T07, and one T10 VM, reducing the cost to 1.5722.

As workload increases further (Applications = 5 and 10), the performance gap becomes more pronounced, with GA achieving cost reductions of approximately 8–9% compared to HC. This improvement can be explained by GA’s ability to explore a broader range of VM combinations and distribute replicas across different VM types, leading to a better match between resource demand and capacity. In contrast, Hill Climbing remains constrained to local improvements and tends to preserve suboptimal VM structures once they are reached.

Overall, these results demonstrate that the advantage of global search methods lies not only in replica scaling but also in their capacity to discover non-trivial and heterogeneous VM configurations, which significantly improves cost efficiency as problem complexity increases.

### 3.3 Scalability Study

The scalability study evaluates how both approaches respond to increasing workload levels. As the workload grows, a larger number of replicas is required to satisfy the stability constraints, which directly increases CPU and memory requirements and leads to a higher number of provisioned virtual machines.

Hill Climbing scales efficiently in terms of execution time, as its local search strategy remains computationally inexpensive even when the workload increases. However, its scalability in terms of cost is limited. As observed in the experimental results, Hill Climbing tends to preserve previously selected VM structures and relies on homogeneous configurations based on large VM instances. This behavior restricts its ability to reorganize the global deployment when demand increases, resulting in higher infrastructure costs for large workloads.

In contrast, the Genetic Algorithm adapts more effectively to workload scaling by jointly adjusting both the number of replicas and the composition of the VM pool. As workload increases, GA explores alternative VM combinations and redistributes replicas across heterogeneous VM types, allowing a finer match between resource demand and capacity. This flexibility explains the increasing cost advantage of GA observed for larger workloads, where cost reductions of up to 8–9% are achieved compared to Hill Climbing.

Overall, the scalability results indicate that while Hill Climbing is well suited for small to moderate workloads and fast decision-making, the Genetic Algorithm provides superior cost scalability for large-scale deployment scenarios. This confirms that global search methods are more effective when workload growth requires significant structural changes in the VM configuration.

### 3.4 Critical Analysis

The evaluation highlights a clear trade-off between computational efficiency and solution quality. Hill Climbing excels in speed and simplicity, making it an effective baseline and a practical choice for small-scale deployments or time-constrained scenarios. Its very low execution time enables rapid reconfiguration; however, its reliance on local improvements makes it prone to suboptimal solutions as problem complexity increases.

This limitation becomes particularly evident in scenarios with higher workload, where Hill Climbing tends to preserve homogeneous VM configurations based on large instances. Once such a structure is reached, the algorithm struggles to reorganize the deployment globally, even when more cost-efficient configurations exist. As a result, Hill Climbing sacrifices cost optimality in favor of simplicity and fast convergence.

The Genetic Algorithm, while computationally more expensive, provides better cost optimization and robustness, especially under high workload conditions. By maintaining a population of solutions and applying crossover and mutation operators, GA is able to explore a wider range of VM configurations and escape local optima. However, its performance strongly depends on appropriate parameter tuning. Poorly chosen parameters may lead to premature convergence or excessive computation time. Once properly configured, GA consistently outperforms Hill Climbing in terms of cost efficiency, as confirmed by the experimental results.

Several modeling assumptions influence the interpretation of the results. The model assumes fixed resource consumption per replica and constant service rates, and it does not account for network latency, inter-component communication overhead, or dynamic workload variations. While these simplifications enable tractable optimization and clear algorithmic comparison, they may limit direct applicability to real-world cloud environments.

Despite these limitations, the results clearly demonstrate the effectiveness of combining greedy placement and repair mechanisms with heuristic and metaheuristic optimization techniques. These components ensure feasibility and allow both local and global search strategies to operate efficiently. The comparison between Hill Climbing and Genetic Algorithm provides valuable insights into the fundamental trade-offs between local optimization speed and global solution quality in cloud application deployment and scaling.

## 4. Conclusion

This project addressed the problem of cost and performance optimization for application deployment and scaling in cloud environments. The objective was to determine an efficient configuration of replicas and virtual machines that ensures system stability while minimizing infrastructure cost. Due to the NP-hard nature of the problem, heuristic and metaheuristic approaches were adopted.

Two optimization techniques were evaluated: Hill Climbing and Genetic Algorithm. Hill Climbing proved to be extremely efficient in terms of execution time, consistently producing feasible solutions in under one second. Its simplicity and speed make it well suited for small-scale deployments or scenarios requiring rapid decision-making. However, its local search nature limits its ability to escape local optima, leading to higher infrastructure costs as workload complexity increases.

The Genetic Algorithm achieved lower or equal costs across all evaluated workloads, with a more pronounced advantage for larger-scale scenarios. By performing global exploration of the solution space, GA was able to identify more cost-efficient configurations, at the expense of higher but stable computation time. This makes GA particularly suitable for large-scale or offline optimization scenarios where cost efficiency is prioritized over execution speed.

The scalability analysis highlighted a clear trade-off between speed and solution quality. Hill Climbing scales well computationally but struggles to maintain cost efficiency under high workloads, whereas the Genetic Algorithm scales more effectively in terms of cost optimization.

Both approaches rely heavily on the greedy placement and repair mechanisms, which ensure feasibility with respect to stability and resource constraints. These components are essential to the effectiveness of the proposed framework.

Future work could extend this study by considering dynamic autoscaling, multi-objective optimization, and more realistic performance models including network effects. Overall, the results demonstrate that combining local and global optimization strategies provides valuable insights into cost-performance trade-offs in cloud application deployment.