Applied Computer Science 2026, vol. 22, no. 2, pp. 105–116
https://doi.org/10.35784/acs_8923

Submitted: 2025-12-30 | Revised: 2026-02-28 | Accepted: 2026-03-07

Keywords: cluster optimisation; Tabu Search; server selection

Andrzej IMIEŁOWSKI [1*], *Łukasz BANAŚ* [2], *Bogusław TWARÓG* [3], *Janusz BYTNAR* [4]

[1] State University of Applied Sciences in Jaroslaw, Poland, andrzej.imielowski@pansjar.edu.pl
[2] Rzeszow University of Technology, Poland, lukasz.banas84@gmail.com
[3] University of Rzeszow, Poland, btwarog@ur.edu.pl
[4] State University of Applied Sciences in Jaroslaw, Poland, janusz.bytnar@pansjar.edu.pl
[*] Corresponding author: andrzej.imielowski@pansjar.edu.pl

Optimisation of the corporate cluster structure using the Tabu Search method

Abstract

The paper presents a new approach to optimising cluster structure by selecting servers to meet specified performance requirements while minimising costs. Modern applications are placing increasing demands on performance and are critical elements of business operations. As a result, the operation of such applications increasingly relies on server clusters. Selecting the type and number of servers is not a trivial task. The problem is further complicated by the widespread use of layered application architectures, which means that different hardware solutions may be optimal for handling different layers. The article proposes a technique that uses the Tabu Search heuristic in conjunction with a BCMP-based application model. An optimisation algorithm for the cluster structure is presented in two versions: minimising the solution cost while meeting performance requirements, and maximising performance while meeting budget constraints.

1. INTRODUCTION

Contemporary multi-tier applications play a key role in today's IT world. Organisations' growth largely depends on the availability and reliability of various digital services. Many of these applications are increasingly critical to stakeholders, who cannot afford downtime, as it can lead to revenue loss and decreased productivity. Ensuring high performance and reliability requires not only advanced software but also appropriate infrastructure. The proper selection of server hardware is crucial, not only for quality and reliability but also for expected performance. This task is non-trivial and often boils down to following the software manufacturer's recommendations or expert opinions, then adjusting the configuration through trial and error.

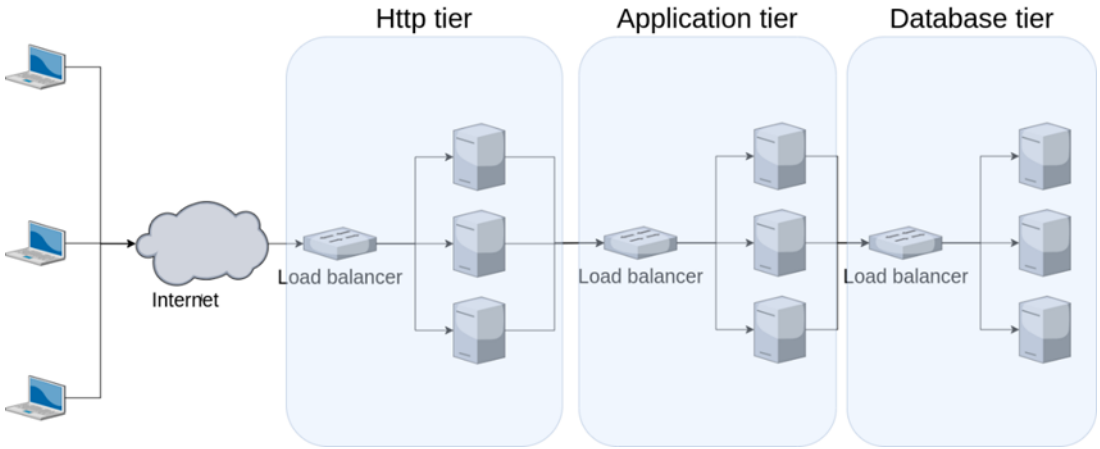

Fig. 1. 3-tier application cluster diagram

This approach often leads to over-dimensioning of systems and, as a result, excessive spending on hardware. It should be noted that there is a lack of scientific literature specifically addressing this particular topic. Therefore, our research draws upon existing work on performance modelling (Imiełowski, 2009; Shoaib & Das, 2011; Urgaonkar et al., 2005) and the application of the Tabu Search algorithm to similar optimisation problems (De Werra & Hertz, 1989; Glover & Laguna, 2013).

In this paper, we propose a method for selecting server hardware based on the heuristic Tabu Search method, which can help in more effectively matching the infrastructure to the actual needs of the applications. Contemporary advanced applications requiring high performance are most often built as three-tier Java applications (Oracle, 2025) , consisting of the presentation layer (HTTP server), the application layer, and the database layer.

Due to increasing performance requirements, horizontal clustering solutions are used—multiplying servers with the same functionality, as well as vertically clustering separate servers to handle individual layers (Apache Software Foundation, 2025). In the most advanced solutions, both approaches are combined. Crucially, this study focuses on a "bare-metal cluster" scenario, in which applications run directly on physical servers. This approach continues to be used in production environments due to the elimination of virtualisation overhead and the ability to ensure maximum control over system performance and reliability.

Figure 1 shows a general diagram of the organisation of a cluster for a typical three-tier application. For the proper functioning of the cluster, in addition to servers, it is also necessary to distribute tasks between individual nodes. This role is usually performed by specialised devices or dedicated servers with appropriate software.

Tab. 1. Solutions in building three-tier applications

Web Servers	Load Balancers	Application Servers	Database Servers
Apache Web Server	Oracle LBaaS	Oracle Weblogic Server	Oracle
MS IIS	Big-IP F5	IBM Websphere	MySql
Oracle OHS	Fortigate	Redhat Jboss	MS SQL
IBM Http Server	Kemp	Apache Tomcat	PostgreSQL
Nginx	Radware's Alteon	GlassFish	MariaDB
Tomcat	Citrix ADC		
	Avi Vantage		

Table 1 summarises the most commonly used solutions for building three-tier applications. Currently, software solutions used in clusters utilize both proprietary and open-source software (Dhanny & Atiim, 2020).

Figure 2 presents an exemplary schematic diagram of a Java application cluster hosted by Oracle WebLogic Cluster solutions. This is one of the most commonly used corporate solutions. In the example configuration, a single database server is used, but there are no obstacles to having the database layer be a horizontal cluster as well.

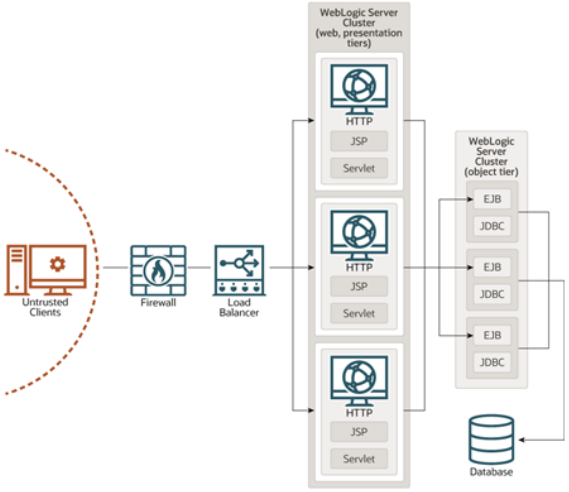

Fig. 2. Logical diagram of the Oracle WebLogic cluster (Oracle, 2025).

2. ANALYTICAL MODEL OF THE CLUSTER

The analytical model described by Imiełowski (2009) has been used to present the above solutions. BCMP-type queuing networks are also used by other authors (Balsamo et al., 2015; Mizuno & Ohba, 2023; Mizuno et al., 2025) to model the performance of cluster systems; however, in this case, an open network was utilised, which allows for greater flexibility in modelling the arrival process of requests to the system. This model is based on a queuing network depicted in Figure 3.

In the modelled cluster system, three classes of requests can be distinguished:

- Requests requiring handling only in the presentation layer of the system (e.g., static web pages).
- Requests requiring handling by the application layer.
- Requests requiring access to the database.

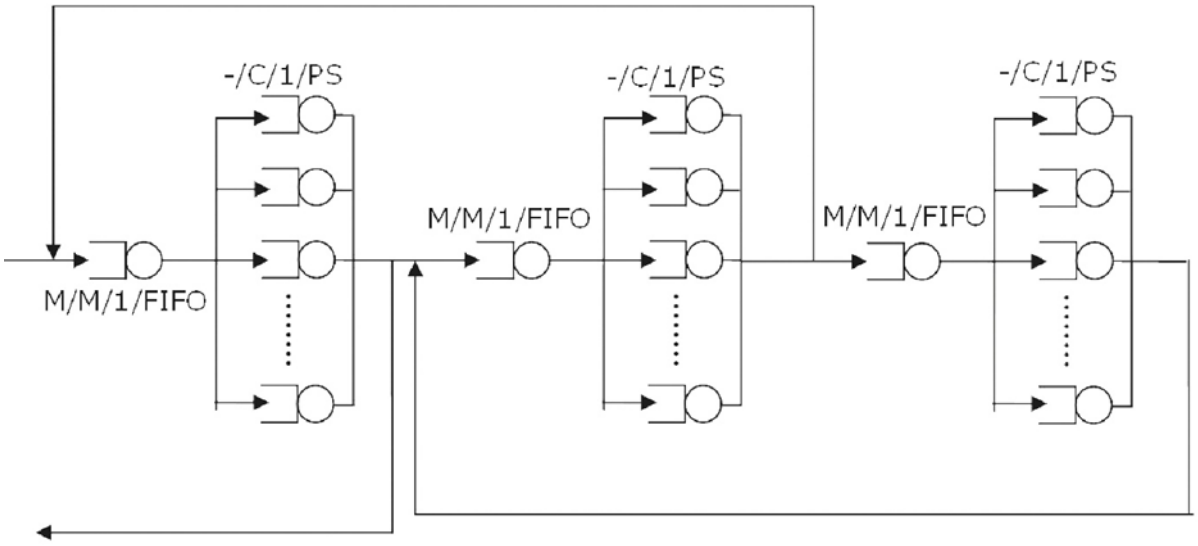

Fig. 3. Cluster queueing network diagram

The following assumptions were formulated for the described system:

- The system is supplied with requests arriving in a Poisson stream at a constant rate λ, independent of the number of clients.
- Each request, independent of the others, is of type *1*, *2*, or *3* with probabilities p_1, p_2 , and p_3, respectively.
- The system consists of three layers.
- Within each layer, there is one load balancer and m_i identical servers (i - layer number).
- Requests, independent of the others, are directed by the load balancer of the i-th layer to one of the servers of that layer with probability $1/m_i$ (m_i - *number of servers in the i-th layer*).
- The length of all queues in the system is unlimited.
- Load balancers in all layers are represented as *˜/M/1/FIFO* stations.
- All servers in the system are represented as *˜/C2/PS* stations with identical service time distributions within a layer.
- The distribution of the request service time is independent of the request type and only depends on the node number.

A queuing network that meets the above assumptions and aligns with Figure 3 satisfies the conditions specified for BCMP (Baskett, Chandy, Muntz, Palacios) queuing networks (Baskett et al., 1975). Considering the above, the queuing network depicted in Figure 3 can be described by the following vector:

$$S' = \left(Q_0, R, L, W, (\mu_{i0}, m_i, \mu_{i01}, \mu_{i02}, A_i)_{(i,j)\epsilon W}\right) \quad (1)$$

where:

- Q_0 - matrix describing the traffic of requests in the network,
- R - set of client classes,
- L - set of layer numbers in the system, $L = \{1,2,3\}$,
- W - set of node numbers in the i-th layer, $W = \{(i,j): i = 1,2,3; j = 0,1,\dots,m_i\}$,
- μ_{i0} - service rate at the dispatcher of the i-th layer,
- m_i - number of servers in the i-th layer,
- $\mu_{i01}, \mu_{i02}, A_i$ - parameters of the Cox distribution describing the service time distribution of requests in the server of the i-th layer.

The solution to the above model allows, among other things, the determination of the expected response time for individual client classes. The details of the solution and the calculable formulas are described in (Imiełowski, 2009).

3. OPTIMISATION TASK

The analytical model described earlier can be used to select appropriate server hardware for a multi-tier application. The task facing the system designer is to determine the optimal cluster structure. Optimal, assuming that certain requirements are met. On one hand, these will be requirements defining minimum performance parameters, such as the maximum acceptable response time or minimum system throughput. On the other hand, the system design must usually meet financial constraints. Consequently, two fundamental questions can be formulated:

- What system structure will ensure the fulfilment of performance requirements with minimal expenditure?
- What are the maximum performance parameters achievable without exceeding the cost limit?

In this context, the system structure refers to the number and type of servers used in the cluster configuration. These questions allow the formulation of optimisation tasks. It should be noted that these are not trivial questions because, for example, if ten possible server types are considered and the possibility of using different server types for handling layers is allowed, the number of possible solutions reaches several million. To solve the optimisation task described above, the Tabu Search algorithm is proposed. This method was introduced by F. Glover (1986) and subsequently developed and described in Glover (1989; 1990). Tabu Search is a heuristic search method that navigates the solution space using a sequence of moves. The algorithm avoids oscillation around local optima by employing a list of forbidden moves (tabu). The tabu list is constructed from already-checked solutions.

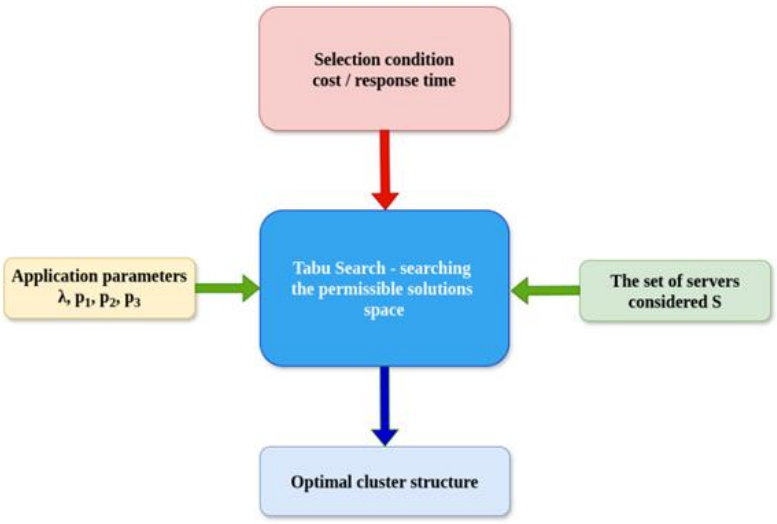

Fig. 4. Decision process

The method includes various search strategies, with the most well-known and popular being intensification and diversification of the search process. Intensification involves “densifying” the sampling of a certain area

of the solution space, while diversification involves dispersing the sampling so that no significant area is omitted. The search concludes when the time limit is exceeded, the allowable number of iterations is completed, or the allowable number of moves without improvement in the objective function value is reached. The Tabu Search algorithm is widely used for optimisation in many technical problems (Lu et al., 2018; Braiki & Youssef, 2024; Shuling & Renping, 2023). The diagram in Figure 4 illustrates the procedure for solving the previously stated tasks.

3.1. First variant - Minimising cluster cost

First, we will consider the first task: minimising the cluster's cost while meeting the specified performance parameters.

For the purpose of the task, it is proposed to describe the cluster configuration d as follows:

$$d = (k_1, m_1, k_2, m_2, k_3, m_3) \tag{2}$$

where:

- k_i – type of server in *i*-th layer $i = (1, 2, 3)$,
- m_i- number of servers in *i*-th layer.

The set of considered servers S can be defined as:

$$S = \{s_i = (k_i, w_i, c_i) : i \epsilon \{1, \dots, n\}\} \tag{3}$$

where:

- n - is the number of server types,
- k - is the type number of the server,
- w - is the performance coefficient of the server *(described in more detail below)*,
- c - is the price of the server.

The performance coefficient w will be understood as a multiple of the performance of a baseline server, and can be expressed as:

$$\mu_i = w_i \cdot \mu_0 \tag{4}$$

where:

- μ_i - performance of the described server,
- w_0 - performance coefficient of the described server,
- μ_0 - performance of the baseline server.

The baseline server is the server for which tests were conducted to determine its performance parameters for the considered multi-tier application. Based on these tests, coefficients w_1, w_2, w_3 will also be determined, adjusting the performance depending on the layer in which the server is used. These coefficients define the ratios of the baseline server's performance across the system layers in which it was used. It is assumed that these coefficients will be the same for all servers, meaning, for example, if the baseline server had half the performance in the database layer compared to the web layer, the same will apply to all considered servers. Using the formulas for the expected response times described in the analytical model, we can calculate the response times in the individual system layers $T^{(1)}, T^{(2)}, T^{(3)}$.

The server performance coefficient w is proposed to be calculated based on the SPECjbb2015 benchmark (SPEC, 2025). The tests in this benchmark evaluate server performance using an advanced Java application. SPEC.org is a non-profit organisation that develops hardware testing methods and publishes test results (Bays & Lange, 2012). From the organisation's website, one can download test results for the servers of interest and calculate the w coefficient based on the proportionality to the baseline server's test results. The value of w for the baseline server is assumed to be $w = 1$.

Algorithm 1: Cost minimization for assumed performance

Input : D, λ, p_1, p_2, p_3, T_0, S
Output : D^*

Inicjalization
$D^* = D$
$TABU = \{D\}$

for $i = 1$ to N
  for $k = 1$ to 12
    if $D_k \epsilon TABU$ then /*checking the tabu list
      go to next k
    else
      add D_k to L /*adding a neighbor to the candidate list
  end
  if $L = \oslash$ then +
    break /*finish
  else
    $K = \left\{\hat{D} \epsilon L : T(\hat{D}) <= T_0\right\}$
    if $K = \oslash$ then
      $D = \hat{D} \epsilon L : min(T(\hat{D}))$
    else
      $D = \hat{D} \epsilon K : min(cost(\hat{D}))$
  if $cost(D) < cost(D^*)$ i $T(D) <= T_0$ then $D^* = D$
  add D to $TABU$ /*adding a solution to the Tabu list
end

Considering the above, we can define the conditions of the optimization task. The solution space can be written as:

$$D = \{d = (k_1, m_1, k_2, m_2, k_3, m_3): k_i \epsilon s, s \epsilon S, i = 1,2,3\} \tag{5}$$

The permissible solution space is:

$$D_0 = \{d \epsilon D: T(d) \leq T_0\} \tag{6}$$

where:

$$T(d) = \sum_{i=1}^{3} p_i \cdot T^i(d) \tag{7}$$

Can be defined as the weighted average response time of the system, with the weight determined by the proportion of requests of a given class in the total stream of requests arriving at the system. T_0 is the assumed maximum system response time.

Consequently, the optimisation task can be formulated as the minimisation of the cost function while meeting the constraints defining the permissible solution space:

$$min\ \mathrm{K}(d) \qquad dla \quad d \epsilon D_0 \tag{8}$$

where $K(d)$ is the function defining the cost of the selected solution, d belonging to the set D_0 of permissible solutions.

Based on Glover et al. (1993) and Laguna (1994), a Tabu Search algorithm for the above task has been developed. The input data for the algorithm includes:

– D - an array describing the cluster structure $[k_1, m_1, k_2, m_2, k_3, m_3]$,

- Structure of the request stream: λ, p_1, p_2, p_3 in the order: arrival rate of the request stream, probabilities of occurrence of requests of class 1,2,3 ,
- T_0 - the maximum weighted response time as determined above,
- S - an array of considered servers for selection $[k_i, w_i, c_i]$.

The algorithm starts the search for the optimal solution from a given cluster structure D and then checks neighbouring solutions. A neighbouring solution differs from the current one by either having one more or one fewer server in a single layer (but only in that layer) or by having a different server type in a single layer by one unit. Thus, each solution has twelve neighbouring points. Neighbouring points that are on the TABU list are immediately excluded, forming a list of aspirant points L for the next move. Next, a list K is created, containing candidates that meet performance requirements. From this list, the next-lowest-cost cluster is chosen. If K is empty (no neighboring clusters meet performance requirements), the move with the shortest response time, wielka litera T, nawias otwierający wielka litera D, nawias zamykający , is selected from the list, $T(D)$ is selected from list L. The selected cluster is added to the TABU list. The algorithm then checks if the proposed cluster is better than the best solution found so far. If L is empty and no move can be made that is not on the TABU list, the algorithm terminates. The variable N represents the given number of iterations. To diversify search areas, the algorithm is repeated a set number of times, starting each attempt from random starting points.

The algorithm is presented in pseudocode in the printout of Alg. 1.

3.2. Second variant - Maximising cluster performance

Similar to the approach outlined in section 3.1, the optimisation task can be formulated as minimising the system's response time function:

$$\min \mathrm{T(d)} \quad \text{dla d} \in \mathrm{D}_0 \tag{9}$$

where function $T(d)$ is defined as:

$$T(d) = \sum_{i=1}^{3} p_i \cdot T_i(d) \tag{10}$$

and D_0 is the set of permissible solutions:

$$D_0 = \{d \in D\colon K(d) \leq K_0\} \tag{11}$$

where the set of solutions D can be written analogously to section 3.1:

$$D = \{d = (k_1, m_1, k_2, m_2, k_3, m_3)\colon k_i \epsilon s, s \epsilon S, i = 1,2,3\} \tag{12}$$

and the cost function can be defined analogously to section 3.1 as:

$$K(d) = \sum_{i=1}^{3} m_i \, c_{k_i} \tag{13}$$

The notations used above are analogous to those used in section 3.1.

Using the Tabu Search method, an algorithm implementation for the above task was developed. The algorithm is presented in pseudocode in Alg. 2.

Both versions of the algorithm presented above differ slightly. The mechanism for searching the set of permissible solutions is the same - the variants differ in the part of selecting a solution to be written to the tabu set. The difference lies in checking whether the solution satisfies the admissibility condition. In one case, it checks that the maximum response time is not exceeded; in the other, that the maximum cost is not exceeded.

Algorithm 2: Minimizing response time at the assumed cost of the solution

Input : $D, \lambda, p_1, p_2, p_3, K_0, S$
Output : D^*

Inicjalization
$D^* = D$
$TABU = \{D\}$

for $i = 1$ to N
  for $k = 1$ to 12
    if $D_k \epsilon TABU$ then /*checking the tabu list
      go to next k
    else
      add D_k to L /*adding a neighbor to the candidate list
  end
  if $L = \oslash$ then
    break /*finish
  else
    $K = \left\{\hat{D} \epsilon L : cost(\hat{D}) <= K_0\right\}$
    if $K = \oslash$ then
      $D = \hat{D} \epsilon L : min(koszt(\hat{D}))$
    else
      $D = \hat{D} \epsilon K : min(T(\hat{D}))$
  if $T(D) < T(D^*)$ i $cost(D) <= K_0$ then $D^* = D$
  add D to $TABU$ /*adding a solution to the Tabu list
end

The presented implementation of the Tabu Search algorithm ensures high scalability. The primary performance constraints in Tabu Search applications are the number of neighbouring points and the length of the tabu list. In the analysed case, the number of neighbours was limited to a small, constant value (12), thereby maintaining low computational costs even for larger-scale problems. The length of the tabu list can be adjusted depending on the size of the feasible solution space.

4. ALGORITHM TESTING

The performance of the proposed Tabu Search-based method was verified through dozens of test cases, in which the algorithm consistently reached the optimal solution. This was confirmed each time by an exhaustive search of the feasible solution space. Due to this deterministic convergence and 100% effectiveness within the tested scope, the error variance was zero; consequently, traditional measures of statistical dispersion and confidence intervals were omitted as they would not provide additional informative value. Furthermore, robustness analysis (illustrated in Tables 3 and 4) demonstrates the algorithm's stability across different starting configurations (parameters N and D). Even in potential sub-optimal scenarios, the relative error remains negligible, as the "gap" between the best and subsequent solutions in the ranking is minimal.

To empirically ground these findings, the algorithm's functionality was validated using real-world data. We selected ten server configurations and determined their market prices based on current hardware distributor data (expressed in PLN). Their performance coefficients (w) were evaluated using the SPECjbb2015 benchmark, a standard for measuring Java server performance. The specifications and performance metrics for these units are summarised in Table 2. This high repeatability, combined with standardised benchmarking, ensures that the method reliably matches physical infrastructure to application requirements without the risk of significant deviation from the optimum.

Tab. 2. List of servers selected for the experiment

	Server type	SPECjbb2015	W	Price
1	Dell Inc. PowerEdge R750xs	203484	1	88525
2	Dell Inc. PowerEdge T550	204568	1.0053272	115000
3	Dell Inc. PowerEdge R650xs	205936	1.012050087	87156
4	Dell Inc. PowerEdge C6615	212674	1.045163256	77000
5	Dell Inc. PowerEdge R750	247011	1.21390871	175000
6	Dell Inc. PowerEdge R7515	249575	1.22650921	67000
7	Dell Inc. PowerEdge R7625	327929	1.611571426	114500
8	Dell Inc. PowerEdge R7615	366259	1.799940044	124000
9	Dell Inc. PowerEdge R760	421268	2.070275796	189000
10	Dell Inc. PowerEdge R6525	471724	2,318236323	214000

For this hardware set, the algorithm was used to select servers for applications with varying performance requirements. The results obtained using the proposed algorithm were compared with those from a full search of the permissible solution space. In all considered cases, the algorithm found the optimal solution. In most cases, the optimal solution was found after no more than 100 steps, and at most 150. The article presents results for one dataset related to an application:

- Arrival rate $\lambda = 800[1/s]$,
- Distribution of requests into classes in the following proportion
    a. $p_1 = 0.3$
    b. $p_2 = 0.3$
    c. $p_3 = 0.4$
- maximum response time $T_0 = 0.065s$

An additional constraint was imposed that the maximum number of servers in a layer cannot exceed 15. This limitation was introduced to allow the algorithm's results to be compared with those of a full search of the solution set.

4.1. Cluster cost minimisation

For the formulated task, the first step was to determine the cluster structure by minimising costs while meeting performance constraints. The results are presented in a tab. 3. Each row of the table represents a single solution and contains, for each layer, the selected server type number, the quantity of units, the total price, and the resulting response time. The first row presents the optimal solution, followed by subsequent solutions ranked by cost.

Tab. 3. Optimisation results - variant 1

Servers						Price	T
Tier 1		Tier 2		Tier 3			
Type	N	Type	N	Type	N	[k PLN]	[s]
8	9	6	12	8	3	2 292	0.0645
8	8	8	8	6	5	2 319	0.0642
8	9	8	7	6	5	2 319	0.0635
8	8	6	13	7	4	2 321	0.0649
8	9	6	12	6	6	2 322	0.0627
8	9	6	11	6	7	2 322	0.0649
8	9	6	13	6	5	2 322	0.0632
8	8	8	7	6	7	2 329	0.0646
8	8	6	14	6	6	2 332	0.0640
8	8	6	13	6	7	2 332	0.0644

The results of applying the proposed algorithm were compared with the results of a full search of the set of permissible solutions. Under the assumptions used (a maximum of 15 servers per layer), this set consists of 3,375,000 possible solutions. The 10 best solutions according to the algorithm coincide with the 10 best solutions from the set of all possible solutions.

4.2. Minimising response time at the assumed cost of the solution

Similar assumptions were used to test the algorithm for minimising system response time under a budget constraint. A maximum cluster cost of PLN 2,300,000 was assumed to enable comparison with the results from the previous section. The results are presented in a tab. 4.

Tab. 4. Optimisation results - variant 2

Servers						Price	T
Tier 1		Tier 2		Tier 3			
Type	N	Type	N	Type	N	[k PLN]	[s]
8	9	6	12	8	3	2 292	0.0645
8	8	6	12	8	4	2 292	0.0653
8	9	6	12	6	5	2 255	0.0655
8	8	6	13	6	6	2 265	0.0657
8	8	8	7	6	6	2 262	0.0659
8	9	6	11	6	6	2 255	0.0662
8	8	6	12	6	7	2 265	0.0667
8	8	6	14	6	5	2 265	0.0668
8	8	6	12	7	4	2 254	0.0672
8	8	6	13	8	3	2 235	0.0674

s in the previous example, the results were compared with those from a full search of the set of permissible solutions, and the same result was obtained.

Figures 5 and 6 show the graphs of the successive steps of the Tabu Search algorithm, with a 5-fold diversification of the search area and 200 steps per cycle.

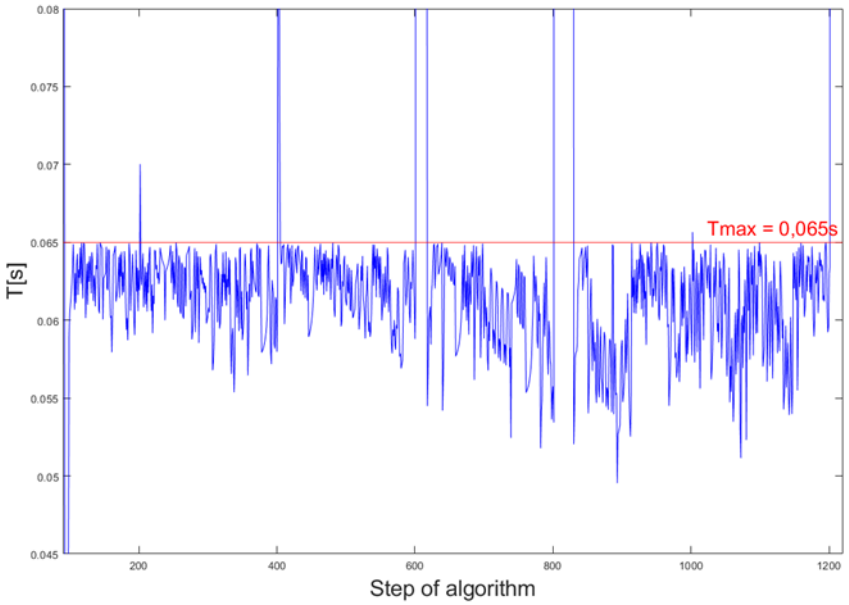

Fig. 5. Graph of response time T_0 in step of the algorithm

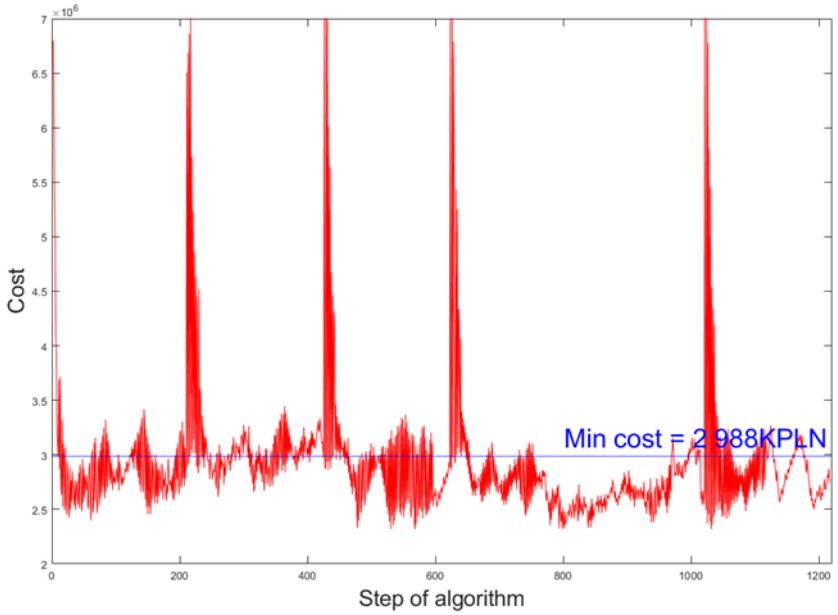

Fig. 6. Graph of cluster cost in step of the algorithm

It is worth noting that, in both task variants, the optimal solutions and solutions close to optimal differ by relatively small values. This applies to both costs and achieved performance parameters. In both task variants, the optimal solutions and solutions close to optimal differ by relatively small values. This applies to both costs and achieved performance parameters. It should be noted that the diversification cycles are clearly visible in the graphs, with each cycle starting the search from random values that are far from optimal. The algorithm approaches optimal values in a small number of steps. Due to the scale of the graphs, the values in the initial steps go beyond the graph.

5. SUMMARY

This article proposes applying a heuristic algorithm to solve a typical engineering design problem. Traditionally, such problems are attempted to be solved through successive trials and improvements of the existing infrastructure, utilising expert knowledge. This often results, on the one hand, in achieving system performance parameters lower than those assumed. Only after a certain period of time and appropriate infrastructure adjustments are the desired effects achieved. On the other hand, the project is also relatively often over-dimensioned, resulting in high costs for the end user. It seems that the proposed method can, to some extent, streamline the design process. On the other hand, the proposed method should be extended to include hardware selection, accounting for system virtualisation.

Conflicts of Interest

The authors declare no conflict of interest.

REFERENCES

Apache Software Foundation. (2025, December 15). *Apache Tomcat 10 configuration reference: The clustering how-to*. https://tomcat.apache.org/tomcat-10.0-doc/cluster-howto.html

Balsamo, S., Rossi, G., & Marin, A. (2015). Applying BCMP multi-class queueing networks for the performance evaluation of hierarchical and modular software systems. *International Journal of Computer Aided Engineering and Technology, 7*, 145. https://doi.org/10.1504/IJCAET.2015.068328

Baskett, F., Chandy, K. M., Muntz, R. R., & Palacios, F. G. (1975). Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM, 22*(2), 248–260. https://doi.org/10.1145/321879.321887

Bays, W., & Lange, K.-D. (2012). SPEC: Driving better benchmarks. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering* (pp. 249–250).

Braiki, K., & Youssef, H. (2024). An experimental and comparative study examining resource utilization in cloud data center. *Cluster Computing*, 1–18.

De Werra, D., & Hertz, A. (1989). Tabu search techniques: A tutorial and an application to neural networks. *OR Spectrum, 11*(3), 131–141. https://doi.org/10.1007/BF01720782

Dhanny, D., & Atiim, S. B. (2020). Free open-source high–availability solution for Java web application using Tomcat and MySQL. *ACMIT Proceedings, 7*(1), 68–77.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. (1990). Tabu search—Part II. *ORSA Journal on Computing, 2*(1), 4–32.

Glover, F., & Laguna, M. (2013). Tabu search. In P. M. Pardalos, D.-Z. Du, & R. L. Graham (Eds.), *Handbook of Combinatorial Optimization* (pp. 3261–3362). Springer. https://doi.org/10.1007/978-1-4419-7997-1_17

Glover, F., Taillard, E., & de Werra, D. (1993). A user's guide to tabu search. *Annals of Operations Research, 41*(1–4), 3–28.

Imiełowski, A. (2009). Modelling of multi-tier internet applications with the use of BCMP queueing networks and simulation model in Simulink. In *Proceedings of the International Conference on Computer Networks* (pp. 200–209).

Laguna, M. (1994). A guide to implementing tabu search. *Investigacion Operativa, 4*, 5–25.

Lu, Y., Cao, B., Rego, C., & Glover, F. (2018). A Tabu search based clustering algorithm and its parallel implementation on Spark. *Applied Soft Computing, 63*, 97–109. https://doi.org/10.1016/j.asoc.2017.11.038

Mizuno, S., Komiyama, Y., & Ohba, H. (2025). Proposal for optimizing number of servers in closed BCMP queueing network. *International Journal of Data Science and Analytics, 20*(3), 2605–2614. https://doi.org/10.1007/s41060-024-00621-x

Mizuno, S., & Ohba, H. (2023). *Optimal nodes placement using a closed BCMP queueing network*. https://doi.org/10.21203/rs.3.rs-2522053/v1

Oracle. (2025, December 15). *Planning and configuring clusters*. https://docs.oracle.com/en/middleware/standalone/weblogic-server/14.1.1.0/clust/planning.html

Shoaib, Y., & Das, O. (2011). Web application performance modeling using layered queueing networks. *Electronic Notes in Theoretical Computer Science, 275*, 123–142. https://doi.org/10.1016/j.entcs.2011.09.009

Shuling, Y., & Renping, Y. (2023). A QoS-aware resource allocation method for Internet of Things using ant colony optimization algorithm and tabu search. *International Journal of Advanced Computer Science and Applications, 14*(9).

Standard Performance Evaluation Corporation [SPEC]. (2025, December 15). *SPECjbb2015 benchmark*. https://www.spec.org/jbb2015/

Urgaonkar, B., Shenoy, P., Chandra, A., & Goyal, P. (2005). Dynamic provisioning of multi-tier internet applications. In *Proceedings of the Second International Conference on Autonomic Computing (ICAC'05)* (pp. 217–228). IEEE. https://doi.org/10.1109/ICAC.2005.27