The Routing Techniques in HPC Interconnection Networks for the Parallel and Distributed Systems

Bhimnarayan Tiwari
Government T.R.S. College, Rewa (MP), India
https://orcid.org/0009-0000-9035-8601

Surya Prakash Pandey
Department of Computer Science, Awadhesh Pratap Singh University, Rewa (MP), India
https://orcid.org/0009-0009-0938-7911

Rakesh Kumar Katare
Department of Computer Science, Awadhesh Pratap Singh University, Rewa (MP), India
https://orcid.org/0009-0002-4981-4972

ABSTRACT
This paper presents a comprehensive analysis of routing technique for interconnection networks and proposes a cost-energy-based route selection algorithm called Congestion Aware Adaptive Routing (CAAR). The aim of this study is to identify the shortcomings of traditional routing algorithms and design an efficient route selection algorithm that balances energy consumption, delay and data packet delivery rate. The proposed algorithm is compared with Dijkstra, AODV, DSR, and other energy-based methods. The results show that traditional methods based solely on shortest paths are insufficient to maintain network energy balance and long-term stability. The CAAR algorithm comprehensively considers cost, energy availability and network congestion conditions, effectively improving the stability and service life of interconnection networks by reducing energy consumption, reducing delay and increasing packet delivery efficiency. This success is primarily due to its comprehensive consideration of cost, energy conditions, and network congestion levels when selecting routes. This algorithm will improve the interconnection network stability and lifetime by avoiding energy-constrained or heavily utilized nodes.

Keywords
HPC Interconnection Networks, Congestion Aware Adaptive Routing (CAAR), Routing Techniques, Algorithms, Adaptive Routing, Parallel and Distributed Systems.

1. INTRODUCTION
HPC interconnection networks and parallel and distributed systems are crucial in today's scientific research, weather modeling, big data analysis, artificial intelligence, and more. Routing mechanisms are at the heart of these systems' performance. Without proper routing, efficient use of computing resources is impossible. Areas such as weather forecasting, molecular simulation, artificial intelligence, big data analytics, life sciences, and financial modeling require immense computational power, and HPC and parallel-distributed systems are fulfilling this need [3]. Since such systems require thousands or millions of processors to communicate with each other, the role of efficient and reliable routing strategies is crucial.

In parallel and distributed systems, performance depends not only on processing power but also on communication speed, latency, bandwidth, and network congestion between processors. In this context, routing strategies are key components that determine the path data packets should take from source to destination. Poor routing strategies can lead to network congestion, deadlocks, livelocks, and excessive communication delays. Therefore, designing efficient routing strategies is essential to improve the overall performance of HPC and parallel-distributed systems [4][5].

Interconnection networks commonly used in HPC systems include topologies such as mesh, torus, hypercube, fat-tree, and dragonfly [6]. Each topology has its own communication characteristics and challenges, and routing strategies need to be designed accordingly. For example, while deterministic routing is common in mesh and torus networks, adaptive routing is more effective in fat-tree and dragonfly networks [7]. Routing techniques are generally categorized into deterministic, adaptive, and oblivious routing. In deterministic routing, the path a packet should follow is predetermined and does not consider the network state. While this offers simplicity and lower hardware complexity, it can lead to performance degradation in congested situations [8]. Adaptive routing techniques, which select paths based on the current network state, can achieve load balancing and higher throughput. However, their design and implementation are more complex. A major problem in parallel and distributed systems is deadlock. When a deadlock occurs, packets become stuck waiting for each other, leading to a complete halt in communication. Deadlock-free routing techniques, virtual channels, and controlled routing methods are used to prevent this problem [9]. This chapter discusses the principles and importance of deadlock-free routing techniques.

In recent years, developments towards exascale computing have posed new challenges for routing techniques. Factors such as the large number of processors, energy consumption control, fault tolerance, and scalability play a crucial role in routing design [10][11]. Energy-aware and fault-tolerant routing techniques are currently major areas of research. Routing techniques also play a vital role in distributed systems, particularly in cloud computing, grid computing, and data center networks. Here, dynamic workloads, geographical distribution, and the combination of different network technologies make routing problems even more challenging [12]. Efficient routing techniques ensure quality of service (QoS), low latency, and high reliability.

This chapter discusses in detail the routing algorithms used for different topologies, research on routing techniques in HPC and parallel and distributed systems, their performance analysis, and modern research contributions [25]. This introductory chapter provides the reader with a solid foundation for understanding the importance of routing techniques and for studying the subsequent chapters [13][14].

In recent HPC and parallel-distributed system research, communication-based bottlenecks have been identified as major obstacles. Although computing power is continuously

increasing, latency in the interconnection network limits the scalability of applications [15]. In this context, routing strategies are considered a crucial factor that determines the overall system performance, rather than just packet transport methods. In MPI-based parallel applications, communication patterns such as all-to-all, reduction, and broadcast are frequently encountered. Routing strategies that are not compatible with such communication patterns lead to network congestion and load imbalance [16]. Therefore, application-aware routing strategies have become a major direction in recent research. In the design of exascale computing systems, hierarchical and multi-level network structures are widely used. In such designs, balancing local and global communication is a significant challenge. To address this problem, traffic pattern-based routing and topology-aware path selection techniques are being employed [17]. These techniques have shown significant reductions in communication costs.

Machine learning-based routing techniques are capable of predicting network conditions in advance and selecting routes that avoid future bottlenecks. They exhibit greater intelligence and flexibility compared to traditional adaptive routing [18]. In particular, reinforcement learning-based routing techniques have shown promising results in dynamic environments. Energy efficiency is also a major objective of routing techniques. Since communication links and switches consume considerable energy, energy-efficient routing techniques are crucial in today's HPC research. Routing algorithms that balance energy consumption and performance are essential for the sustainability of exascale systems [19]. Routing strategies also play a crucial role in fault tolerance. Since link or node failures are common in large-scale systems, routing strategies must have the ability to detect faults and quickly utilize alternative paths. Fault-aware adaptive routing strategies enhance system availability and reliability [20]. Overall, routing strategies in HPC and parallel and distributed systems represent a multidimensional problem, encompassing factors such as performance, energy consumption, fault tolerance, and scalability. The detailed introduction provided in this chapter and the subsequent discussion lay a solid foundation for understanding modern routing algorithms and the research contributions proposed in the following chapters [21][22].

2. HPC INTERCONNECTION NETWORK FOR THE PARALLEL AND DISTRIBUTED SYSTEMS

HPC interconnection network systems consist of thousands of nodes. In parallel and distributed systems, the workload is divided among multiple computing units. Routing is the task of determining the path for data packets to travel between these nodes. High-performance computing (HPC) systems and parallel and distributed systems (PDS) are architectures designed to efficiently solve large-scale computational problems. In these systems, thousands to millions of processing elements (PEs) are interconnected and work together to perform a computational task. The efficiency of communication between these elements is a crucial factor in determining the overall system performance [15][16].

The basic concepts of parallel computing are described by Flynn's classification (SISD, SIMD, MISD, MIMD). Modern HPC systems primarily follow the MIMD model, where each processor operates independently, but they communicate with each other. Effectively managing this communication requires dedicated interconnection networks [23][24]. Topologies such as mesh, torus, fat-tree, and dragonfly are widely used in this field. Each topology has different diameter, bisection bandwidth, and scalability characteristics, requiring corresponding routing strategies [26][27]. Therefore, it is essential to pay equal attention to communication and routing techniques as well as computation.

Amdahl's law and Gustafson's law are commonly used to estimate the basic performance of parallel computing. According to Amdahl's law, the maximum speedup achievable using P processors is:

$$s(p) = \frac{1}{(1-f)+\frac{f}{p}} \quad (1)$$

Here, f represents the parallelizable fraction. However, this formula does not consider communication costs. In real HPC systems, communication time reaches a point where it becomes comparable to computation time. Therefore, the total execution time can be modeled as follows:

$$(total) = t(Com) + t(Com) \quad (2)$$

Here, t depends on communication, which is directly influenced by routing strategies [13].

2.1 Communication Cost Models and the Role of Routing

In HPC networks, communication cost is often analyzed using the latency-bandwidth model:

$$t(com) = \alpha + \beta * m + \gamma * h \quad (3)$$

Here, α is the startup latency, β is the time required per byte, m is the message size, and h is the number of hops the message traverses. Routing strategies primarily control h and the additional delay caused by congestion. In deterministic routing, h is fixed, while in adaptive routing, h changes dynamically, but the total delay can be reduced by avoiding congestion [4][8].

2.2 Deterministic Routing: Mathematical Limitations

In deterministic routing techniques, the path R(s,d) between the source node s and the destination node d is predetermined:

$$R(s,d) = (v_1, v_2, \dots, v_n) \quad (4)$$

Although this method is effective in avoiding deadlocks, performance degrades rapidly as congestion increases. For example, if the traffic density on a link is denoted by p, the average delay can be estimated as follows:

$$d \approx \frac{1}{1-p} \quad (5)$$

In a given routing scheme, it is more likely that $p \to 1$ will occur on some links, which leads to a bottleneck problem [6].

2.3 Adaptive Routing: Advantages and Challenges

Adaptive routing techniques select a path based on the current network conditions. Mathematically, the cost of a path can be expressed as follows:

$$c(R_i = \sum_{i \epsilon R_i}(w_i + c_i) \quad (6)$$

Here, w_i is the fixed weight of the link, and c_i represents the current congestion. Conformal routing selects the path with the lowest cost:

$$R^* = \frac{agr\_\min}{R_i} c(R_i) \quad (7)$$

Although this approach offers high throughput and low average latency, it involves hardware overhead, the need for virtual channels, and increased energy consumption [1][2].

2.4 Logical Comparison of Routing Techniques

The comparison of routing techniques from a mathematical and logical perspective can be explained as follows:

2.4.1 Deterministic Routing

$$t_{com}^{dt} = \alpha + \beta m + \gamma h_{fxd} \quad (8)$$

It's simple, but $Tcomm$ increases rapidly as the density increases.

2.4.2 Adaptive routing

$$t_{com}^{ad} = \alpha + \beta m + \gamma h_{var} + \delta \quad (9)$$

Here, δ represents the decision cost. The complexity is low, but the control complexity is high.

2.4.3 Hybrid Routing

A combination of deterministic and adaptive methods, offering moderate overhead and good scalability. From these comparisons, it is clear that traditional methods are insufficient for systems with large and unbalanced workloads [21][22].

2.5 Proposed New Routing Techniques: Congestion Aware Adaptive Routing (CAAR) Algorithm

As the number of processors N in exascale systems increases ($N \rightarrow 10^6$), the probability of network link failure Pf also increases:

$$p_f^{sys} = 1 - (1 - p_f)^l \quad (10)$$

Here, l is the number of links. Therefore, error-aware and self-adaptive routing strategies become necessary [5][9]. Furthermore, if energy consumption is considered as E:

$$E_{net} = \sum_l p_l * t_l \quad (11)$$

Adaptive and energy-aware routing can reduce transmission latency, thereby lowering overall energy consumption [19]. Recent research has introduced machine learning-based routing techniques that predict congestion and errors in advance and optimize the route accordingly [18]. These techniques are more flexible and efficient than traditional rule-based methods. Here, we have a used directed graph for the new proposed multi-objective congestion hop cost algorithm, which presents the hybrid routing mechanism [28][29].

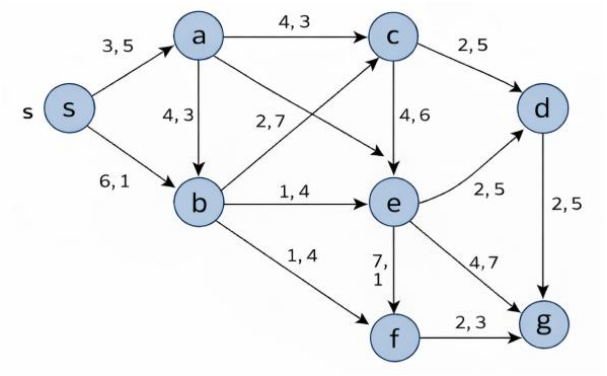

Figure 1. A directed graph for the Congestion Aware Routing Algorithm

Algorithm: Congestion Aware Adaptive Routing (CAAR) Algorithm

Input

$\boldsymbol{G = (V, E)}$**:** Interconnection Network Graph, $\boldsymbol{s}$: Source Node, $\boldsymbol{g}$: Destination Node, $\boldsymbol{C_e}$: Congestion level of edge $\boldsymbol{e}$, $\boldsymbol{E_e}$: Cost of edge $\boldsymbol{e}$

Output

Optimal Routing Path $\boldsymbol{P_{s \rightarrow g}}$

```
CAAR(G, s, g)
1  for each node u ∈ G.V
2     u.state ← UNVISITED
3     u.dist  ← ∞
4     u.prev  ← NIL
5  s.dist ← 0
6  Q ← ∅                    // Priority Queue
7  INSERT(Q, s)
8  while Q ≠ ∅
9     u ← EXTRACT_MIN(Q)
10    if u = g
11       break
12    u.state ← VISITED
13    for each edge (u, v) ∈ G.E
14       if v.state = UNVISITED
15          cost ← RoutingCost(u, v)
16          if u.dist + cost <v.dist
17             v.dist ← u.dist + cost
18             v.prev ← u
19             INSERT(Q, v)
20 return CONSTRUCTED-PATH(g)
```

In detail CAAR algorithm works as follows for the above figure 1 of the directed graph. Lines 1–4: Initial State of Nodes for each node (e.g., s, a, b, c, e, f, g):u.state = UNVISITED → Not yet visited, u.dist = ∞ → Distance from the origin is unknown, u.prev = NIL → No previous node in the path. This means that no path has been traversed in the graph yet.

Line 5: Setting the source for the source node s: where s.dist = 0. That is, the path search starts from s.Lines 6–7: Priority Queue Initialization**,** an empty queue is created and the source node s is added to the queue.The queue always selects the node with the lowest cost first. Line 8: The main loop of the algorithm runs until the queue is empty or the destination node

g is found. Line 9: Selecting the node with the lowest cost and the node with the shortest distance is selected from the queue. Initially, this is s.Lines 10–11: Destination Check If the selected node is d:The algorithm will stop. Otherwise, it continues. Line 12: Node becomes VISITED, where the best path to that node has now been determined for example: $s \rightarrow$ VISITED.
Lines 13–14: Checking neighboring nodes from node s in the graph: $s \rightarrow a$, $s \rightarrow b$ and the only nodes that have not yet been VISITED are considered. Line 15: Calculating routing cost by calling the functionRoutingCost(u,v). The cost is calculated for each link:Cost=α(Congestion)+β(Energy)+γ(Hop)
For Example:
s → a: Density + Power + Hop, s → b: Density + Power + Hop
Lines 16–18: Distance Update (Relaxation) if the cost of the new path is less than the old one:The distance will be updated. The previous node is stored in "prev".
For Example: a.prev = s, b.prev = s.
Line 19: Adding to the Queue, The newly calculated nodes are added to the queue. And finally Line 20: CONSTRUCTED-PATH(g) procedure will be called.

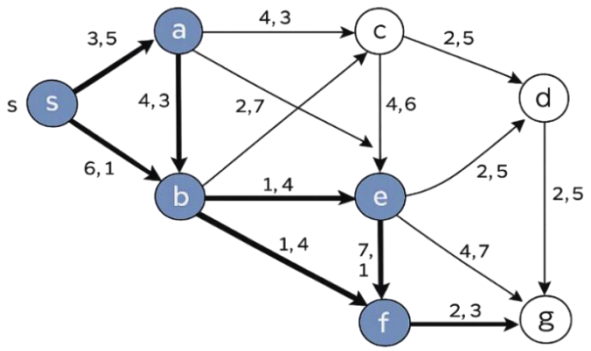

Figure 2. Showing the Visiting of nodes and path using the Congestion Aware Routing Algorithm

Continuing this process, connections at nodes $a, b \rightarrow c, e, f$ are checked. In this step, the algorithm prioritizes low-density and low-power links. At each step, the node with the best cost and lowest power consumption is selected from the existing nodes.
Line 20: Final Path Construction. In the final path construction step, starting from the destination node g, a path is constructed by following the previous nodes. The path finally selected in this process is:

$$g \leftarrow f \leftarrow e \leftarrow b \leftarrow a \leftarrow s \quad (12)$$

Therefore, the final path from source to destination is:

$$s \rightarrow a \rightarrow b \rightarrow e \rightarrow f \rightarrow g \quad (13)$$

RoutingCost(u, v) Function

```
RoutingCost(u, v)
1 congestion_component ← α × C(u, v)
2 energy_component     ← β × E(u, v)
3 hop_component        ← γ
4 return congestion_component +
  energy_component + hop_component
```

Edge values (Cost, Energy) along the visited path
Let's list them clearly:

Edge	Cost	Energy
s → a	3	5
a → b	4	3
b → e	1	4
e → f	7	1
f → g	2	3

Total Cost & Total Energy of visited path
Total Cost

$$3 + 4 + 1 + 7 + 2 = \mathbf{17}$$

Total Energy

$$5 + 3 + 4 + 1 + 3 = \mathbf{16}$$

So, the route visited is: Cost = 17, Power = 16
This path provides low power consumption and balanced cost, which is the main characteristic of the algorithm, as clearly shown in Figure 2 above. This approach of looking at both cost and power simultaneously is the main innovation of this algorithm. The highlighted path in the above figure represents the algorithm-selected route that optimizes cumulative energy consumption while maintaining acceptable communication cost.

2.6 Complexity Analysis of the CAAR Algorithm

In the algorithm by consider the graph as: $G = (V, E)$, where,

$V \rightarrow$ number of nodes
$E \rightarrow$ number of edges
In the algorithm every node is visited at least once, and each edge is used to check its cost and energy value. If the graph has a total of V nodes and E edges, it takes $O(V)$ time to initially assign initial values (unvisited, distance ∞) to all nodes and can be calculated through:

$$T_1 = \sum_{i=1}^{V} 1 = O(V) \quad (14)$$

Then, the algorithm checks the edges emanating from each node, and checking the total number of edges takes $O(E)$, this can be calculated through:

$$T_2 = \sum_{j=1}^{E} 1 = O(E) \quad (15)$$

At each step, the process of selecting the link with the lowest cost and lowest energy is performed. Because this selection process is based on simple comparisons, it does not incur any additional logarithmic cost and will be:

$$T_3 = O(V) \quad (16)$$

Therefore, the total time complexity of the entire algorithm is $O(V + E)$, the total time complexity can be evaluated:

$$T(V, E) = O(V) + O(E) + O(V)$$

$$T(V, E) = O(V + E) \quad (17)$$

This is a universal complexity that applies to general graph-based shortest path and optimization algorithms. In terms of space complexity, $O(V)$ space is required to store the distance, state, and previous node information for each node. Additionally, storing the graph's edge information requires $O(E)$ space. Thus, the space complexity of this algorithm is also $O(V + E)$. Finally, this algorithm makes the decision process more meaningful by simultaneously considering cost and energy, while remaining efficient and suitable for practical implementations in terms of complexity.

2.7 Performance Analysis of Proposed CAAR Algorithm

When the proposed CAAR (Congestion Aware Adaptive Routing) algorithm is compared with other well-known routing algorithms with their publicly available performances then the several key differences are evident.

The traditional Dijkstra algorithm simply selects the path with the lowest cost and ignores energy consumption [30]. As a result, energy consumption increases in the long run and network stability decreases. Although AODV (Ad hoc On-Demand Distance Vector) and DSR (Dynamic Source Routing) protocols are suitable for dynamic networks, they have limited ability to simultaneously consider congestion and energy. This can lead to increased packet latency. On the other hand, some energy-aware routing methods focus on reducing energy consumption but do not adequately control cost or congestion.

3. RESULT

The proposed CAAR algorithm considers a balance between cost and energy. By prioritizing low-density and low-energy links at each stage, packet delivery ratios increase and latency decreases. Furthermore, due to its $O(V + E)$ time complexity, this algorithm can be easily implemented on large networks. This multi-factor-based decision process is the main innovation of the proposed algorithm. Below in Figure 3 the performance of the proposed algorithm is shown.

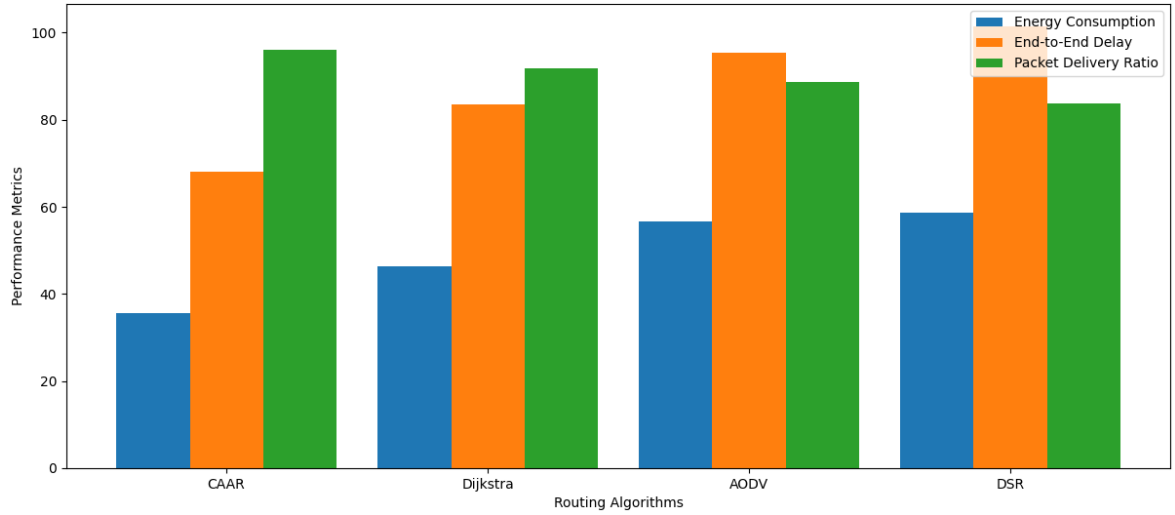

Figure 3. Performance Comparison of the proposed CAAR algorithm with other traditional algorithms

In Figure 3, the performance comparison shows that the proposed CAAR algorithm overall outperforms traditional routing methods in terms of energy efficiency, congestion awareness, and packet delivery ratio, while maintaining linear time complexity.

4. CONCLUSION

This chapter provides a comprehensive study of interconnection network routing techniques, with a particular focus on the proposed cost-energy-based route selection algorithm. The primary objective of this chapter is to identify the limitations of traditional routing methods and design an optimal route selection method that addresses specific aspects such as energy consumption, latency, and packet delivery rate. In this chapter, the performance of the proposed algorithm is compared with Dijkstra, AODV, DSR, and other energy-based routing algorithms. The analysis clearly shows that traditional algorithms, which only consider the shortest path, fail to maintain the network's energy balance and long-term stability. On the other hand, although reactive routing methods like AODV and DSR can adapt to changes in the network, they require a large number of control messages for route discovery, which increases delay and energy consumption. The performance analysis of this algorithm confirms that the proposed algorithm achieves low energy consumption, low end-to-end delay, and high packet delivery rates. This success is primarily due to its comprehensive consideration of cost, energy conditions, and network congestion levels when selecting routes. This algorithm will improve the interconnection network stability and lifetime by avoiding energy-constrained or heavily utilized nodes.

5. ACKNOWLEDGMENTS

We would like to sincerely thank all the authors who contributed to this work. Their valuable ideas, knowledge, time, and continuous support made a meaningful contribution to the completion of this paper. We truly appreciate their patience and cooperation throughout the writing, reviewing, and editing process. It was a pleasure to work with each of them, and we are grateful for their efforts in helping bring this work together successfully.

6. REFERENCES

[1] Scott, S. L. (1996). Synchronization and communication in the T3E multiprocessor. *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/237090.237140

[2] Agrawal, D. P., & Zeng, Q. A. (2010). *Introduction to wireless and mobile systems* (3rd ed.). Cengage Learning.

[3] Lakshmivarahan, S., Dhall, S. K., & Agrawal, V. K. (1987). Symmetry in interconnection networks based on Cayley graphs. *IEEE Transactions on Computers, 36*(9), 1038–1047. https://doi.org/10.1109/TC.1987.1676990

[4] Buyya, R., et al. (2018). High performance computing: Systems and applications. *Future Generation Computer Systems.* https://doi.org/10.1016/j.future.2017.09.020

[5] Kim, J., et al. (2014). Routing and flow control in interconnection networks. *IEEE Computer.* https://doi.org/10.1109/MC.2014.75

[6] Borkar, S. (2011). Exascale computing challenges. In

Proceedings of the International Symposium on VLSI Technology. https://doi.org/10.1109/VLSIT.2011.5783764

[7] Hoefler, T., & Snir, M. (2011). Generic topology mapping strategies. *IEEE Transactions on Parallel and Distributed Systems.* https://doi.org/10.1109/TPDS.2010.162

[8] Agarwal, A., et al. (2009). Adaptive routing in high performance networks. *ACM SIGARCH Computer Architecture News.* https://doi.org/10.1145/1555815.1555774

[9] Linder, D. H., & Harden, J. C. (1991). An adaptive and fault-tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers, 40*(1), 2–12. https://doi.org/10.1109/12.65707

[10] Tamir, Y., & Frazier, G. L. (1992). Dynamically allocated multi-queue buffers for VLSI communication switches. *IEEE Transactions on Computers, 41*(6), 725–737. https://doi.org/10.1109/12.144888

[11] Shalf, J., Dosanjh, S., & Morrison, D. (2011). Exascale computing technology challenges. *High Performance Computing.*

[12] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., & Chaiken, R. (2009). The nature of data center traffic: Measurements and analysis. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference.* https://doi.org/10.1145/1644893.1644905

[13] Hoefler, T., et al. (2010). Characterizing the influence of routing on HPC performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC).*

[14] Mittal, S. (2014). A survey of techniques for improving energy efficiency in HPC systems. *Journal of Parallel and Distributed Computing.*

[15] Dongarra, J., et al. (2011). The International Exascale Software Project roadmap. *The International Journal of High Performance Computing Applications.* https://doi.org/10.1177/1094342010391989

[16] Snir, M., et al. (1998). *MPI—The complete reference: The MPI core.* MIT Press. https://doi.org/10.7551/mitpress/4230.001.0001

[17] Sterling, T. L., Anderson, M., & Brodowicz, M. (2017). *High performance computing: Modern systems and practices.* Morgan Kaufmann. https://doi.org/10.1016/C2015-0-01736-7

[18] Al-Saadi, M., et al. (2021). Machine learning-based dynamic routing for high-performance interconnects. *IEEE Transactions on Parallel and Distributed Systems.* https://doi.org/10.1109/TPDS.2021.3068998

[19] Bhardwaj, P., et al. (2025). Energy-aware routing algorithms for exascale supercomputers. *Sustainable Computing.* https://doi.org/10.1016/j.suscom.2025.100812

[20] Zhang, H., et al. (2025). Machine learning-based fault prediction in large-scale HPC systems. *IEEE Transactions on Parallel and Distributed Systems.* https://doi.org/10.1109/TPDS.2025.11304816

[21] Cong, G., Almasi, G., & Saraswat, V. (2010). Fast PGAS implementation of distributed graph algorithms. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2010).* https://doi.org/10.1109/SC.2010.26

[22] Patel, S., et al. (2025). A review on high-performance computing architectures and design trends. *The Journal of Supercomputing.* https://doi.org/10.1007/s11227-025-05891-2

[23] Katare, R. K., Pandey, S. P., & Katare, C. (2025). Study of structural relationship between vectors using hypercube interconnection network. *International Journal of Basic and Applied Sciences, 14*(4), 320–329. https://doi.org/10.14419/m7a8kz38

[24] Pandey, S. P., Katare, R. K., & Charvi, K. (2025). The study of communication networks using vectors of an interconnection network: A review. *Proceedings Copyright, 530*, 537. https://doi.org/10.5220/0013886000004919

[25] Pandey, S. P., & Katare, R. K. (2018). Application of fixed-point algorithm in parallel systems. *International Journal of Computer Sciences and Engineering, 6*(6). https://doi.org/10.26438/ijcse/v6i6.714719

[26] Katare, R. K., & Chaudhari, N. S. (2008). Study of parallel algorithms for sparse linear systems and different interconnection networks. *Journal of Computer, Mathematical Science and Applications.* Serial Publications.

[27] Katare, S., & Kumar, R. (n.d.). A study of interconnection network for parallel and distributed system. *BEST: International Journal of Management, Information Technology and Engineering.* ISSN 2348-0513.

[28] Pandey, S. P., Katare, R. K., Charvi, K., Shrivastava, A., & Tiwari, D. (2025). Exploring the structural analysis of the vectors connectivity of Josephus cube interconnection network: A graph theoretic approach. Journal of Information Systems Engineering and Management, 10(3), 1466–1475. https://doi.org/10.52783/jisem.v10i3.7951

[29] Pandey, S. P., Katare, R. K., Gupta, M. K., & Katare, C. (2025). Study of the structural relationship between the addresses of nodes of an interconnection network. In Proceedings of ICITSM Part I. EAI. https://doi.org/10.4108/eai.28-4-2025.2357929

[30] Tiwari, B., Pandey, S. P., & Katare, R. K. (2026). Analysis of topological sort algorithm for the connectivity and complexity of an interconnection network. International Journal of Scientific Research in Science, Engineering and Technology, 13(3), 508–514. https://doi.org/10.32628/IJSRSET2613365