REVIEW ARTICLE

Performability Analysis for Large-Scale Multi-State Computing Systems: Methodologies, Advances, and Future Directions

Yuchang Mo[1,*], Yuan Fan[2], Chunyu Miao[2], Mirlan Chynybaev[3], Faer Gui[4], Rengui Zhang[5], Jianyong Hu[6], Jinbin Mu[7] and Akylbek Chymyrov[3]

[1] School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power, Hangzhou 310018, China
[2] Hangzhou Anheng Information Technology Co., Ltd., Hangzhou 310051, China
[3] Razzakov Kyrgyz State Technical University, Bishkek 720044, Kyrgyzstan
[4] Zhejiang Keepsoft Information Technology Corp.,Ltd., Hangzhou 310051, China
[5] Zhejiang YuGong Information Technology Co., Ltd., Hangzhou 310002, China
[6] Engineering Research Center of Digital Twin Basin of Zhejiang Province, Hangzhou 310018, China
[7] Zhejiang Institute of Hydraulics and Estuary, Hangzhou 310020, China

Abstract

Large-scale computing systems, such as cloud data centers, grid infrastructures, and high-performance computing clusters, are the backbone of modern information technology ecosystems. These systems typically consist of numerous heterogeneous, multi-state computing nodes that exhibit varying performance levels due to component failures, degradation, or dynamic resource allocation. Performability analysis, which integrates both system reliability and performance evaluations to quantify the probability of the system operating at a specified performance level, is critical for ensuring the efficient, reliable, and cost-effective operation of these complex systems. This paper presents a comprehensive review of recent advancements in performability analysis for large-scale multi-state computing systems over the past decade. It classifies existing research into three core methodological categories: binary decision diagram (BDD)-based approaches, multi-valued decision diagram (MDD)-based approaches, and comparative benchmarking with traditional methods (e.g., continuous-time Markov chains (CTMC), universal generating function (UGF)). For each category, the paper details key methodologies, algorithmic innovations, and practical applications. Additionally, the promising future directions are proposed to address emerging challenges, such as handling dynamic system behaviors, integrating real-time data, and optimizing resource allocation for performability. This review provides a valuable reference for researchers, system designers, and operators seeking to enhance the performability of large-scale computing systems and mitigate risks associated with service level agreement (SLA) violations.

Submitted: 09 September 2025
Accepted: 26 September 2025
Published: 31 October 2025

Vol. 1, **No.** 2, 2025.
10.62762/TSSR.2025.527003

***Corresponding author:**
✉ Yuchang Mo
yuchangmo@sina.com

Citation
Mo, Y., Fan, Y., Miao, C., Chynybaev, M., Gui, F., Zhang, R., Hu, J., Mu, J. & Chymyrov, A. (2025). Performability Analysis for Large-Scale Multi-State Computing Systems: Methodologies, Advances, and Future Directions. *ICCK Transactions on Systems Safety and Reliability*, 1(2), 81–97.

Keywords: performability analysis, large-scale computing systems, multi-state systems, binary decision diagrams (BDD), multi-valued decision diagrams (MDD), reliability, system performance.

1 Introduction

1.1 Background and Significance

The digital transformation of industries—from finance and healthcare to manufacturing and entertainment—has driven an unprecedented demand for large-scale computing systems. Cloud data centers, which host millions of virtual machines (VMs) to support software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS) offerings, now process exabytes of data daily [1–3]. Grid computing networks, spanning multiple administrative domains, enable collaborative scientific research (e.g., climate modeling, particle physics simulations) by aggregating computing power from thousands of distributed nodes [4]. High-performance computing (HPC) clusters, with specialized processors and high-speed interconnections, deliver the computational muscle required for AI model training and complex engineering simulations [5]. These systems are no longer mere technical assets but strategic enablers of economic growth and societal progress.

A defining characteristic of modern large-scale computing systems is the multi-state behavior of their constituent nodes. In this review, multi-state nodes have computing components (e.g., servers, VMs) that exhibit $\geq$ 3 operational states (from partial degradation to full failure) due to component failures or dynamic resource allocation, with each state corresponding to a distinct resource contribution (e.g., 2 CPUs + 3 memory modules operational). Unlike traditional computing nodes, which were often modeled as binary (either fully operational or completely failed), contemporary nodes exhibit a spectrum of performance states [6, 7]. For instance, a cloud server equipped with 4 CPUs and 8 memory modules may operate in states such as "4 CPUs + 8 memory modules operational" (full performance, delivering 4×2.5 GHz = 10 GHz CPU frequency and 8×16 GB = 128 GB memory), "3 CPUs + 6 memory modules operational" (reduced performance, 7.5 GHz + 96 GB), or "0 CPUs + 0 memory modules operational" (complete failure). This multi-state nature arises from partial component failures: a single failed CPU or memory module does not render the entire node inoperable but instead degrades its computing capacity [8, 9].

Compounding this complexity is node heterogeneity. In practice, large-scale computing systems are rarely composed of identical nodes. Cloud providers, for example, often incrementally deploy new servers with advanced processors (e.g., Intel Xeon 4th Gen vs. 3rd Gen) alongside older ones, leading to variations in CPU frequency, memory capacity, and failure rates across nodes [10, 11]. Grid computing systems, which rely on voluntary contributions from academic institutions and enterprises, may include nodes ranging from personal laptops to enterprise-grade servers, each with distinct performance capabilities [12]. This heterogeneity makes it impossible to generalize performance or reliability results from one node to another, requiring tailored analysis for each node type.

Against this backdrop, performability analysis has emerged as a critical tool for system stakeholders [13]. Unlike reliability analysis (which focuses solely on the probability of system survival) or performance analysis (which assumes perfect component reliability), performability analysis unifies these two dimensions to answer a pivotal question: *What is the probability that the system will deliver a specified level of performance over a given mission time?* This question is directly tied to operational and business objectives:

- **SLA Compliance**: Cloud providers typically guarantee resource availability (e.g., 99.999% uptime for critical VMs) and performance (e.g., $\leq$10 ms latency for database queries) in SLAs. A violation of these terms can result in financial penalties (e.g., 10% service credit for each hour of downtime) and reputational damage. Performability analysis helps providers determine whether their system can meet SLA commitments and identify potential bottlenecks before violations occur[14, 15].
- **Cost Optimization**: Deploying redundant nodes to enhance performability increases capital expenditure (CapEx), while under-provisioning leads to SLA violations and lost revenue. Performability analysis enables a data-driven balance: for example, a provider can use performability results to decide whether adding 5 redundant servers to a 100-node cluster will increase the probability of meeting SLA requirements from 99.9% to 99.99%—a justifiable

investment for high-priority clients [16–18].

- **Risk Mitigation**: Large-scale computing systems are vulnerable to a range of disruptions, including component failures, cyberattacks, and natural disasters. Performability analysis identifies "weak links" in the system—for instance, a node with a high failure rate that, if inoperable, would reduce cumulative CPU frequency below the SLA threshold. This allows operators to implement targeted mitigations, such as proactive maintenance or redundant backups [19, 20].

1.2 Evolution of Performability Analysis for Large-Scale Computing Systems

The evolution of performability analysis for large-scale computing systems can be traced through three distinct phases, each shaped by advancements in system complexity and methodological innovation.

1.2.1 Phase 1: Binary-State and Homogeneous System Models

Early research focused on small-scale, homogeneous systems with binary-state components. Methods such as continuous-time Markov chains (CTMC) were the primary tools for modeling system behavior [6, 21, 22]. CTMC represents each system state as a node in a graph, with edges denoting state transitions (e.g., a node moving from "operational" to "failed") and transition rates derived from component failure/repair distributions (typically exponential). For example, a 10-node cluster with binary-state nodes would have $2^{10} = 1024$ system states, which is manageable for CTMC analysis.

However, this phase had two critical limitations. First, binary-state models failed to capture the multi-state nature of modern nodes—they treated a node with one failed CPU as completely failed, leading to overestimates of system downtime [23]. Second, CTMC suffered from the state-space explosion problem: the number of system states grows exponentially with the number of nodes [24]. For a 20-node binary-state system, the number of states reaches 1 million; for a 30-node system, it exceeds 1 billion—far beyond the computational capacity of most hardware. This made CTMC impractical for the large-scale systems (100+ nodes) that began to emerge in the late 2000s.

1.2.2 Phase 2: Multi-State Models with Combinatorial Methods

The rise of cloud computing and multi-core processors in the 2010s drove the need for multi-state performability models. Researchers began to explore combinatorial methods, which represent system behavior using logical functions rather than enumerating all states, to address state-space explosion. Two key methods emerged during this phase: binary decision diagrams (BDD) and universal generating functions (UGF).

Binary Decision Diagrams (BDD): BDD is a graph-based data structure that represents Boolean functions (e.g., "cumulative CPU frequency $\geq$ 50 GHz") using a rooted, directed acyclic graph (DAG) [25–27]. For multi-state nodes, BDD requires converting each multi-state variable into a set of binary variables (e.g., a 3-state node might be represented by two binary variables: "state 1 vs. others" and "state 2 vs. others"). This conversion introduced overhead, but BDD's reduction rules—merging isomorphic subgraphs and deleting useless nodes—significantly reduced model size. For example, a 50-node system with 3-state nodes, when converted to binary variables, would have a BDD size of $O(n^2)$ instead of 3^{50} (a number with 24 digits).

Universal Generating Function (UGF): UGF represents the performance of each node as a polynomial, where coefficients are state probabilities and exponents are performance values (e.g., CPU frequency) [28]. The system's UGF is the product of individual node UGFs, and performability is calculated by summing the coefficients of terms that meet the performance threshold. UGF is straightforward to implement for heterogeneous nodes but requires exhaustive enumeration of all possible term combinations, leading to large intermediate polynomials for 100+ node systems [29, 30].

This phase laid the groundwork for multi-state performability analysis but had limitations. BDD's binary conversion overhead made it inefficient for nodes with many states (e.g., a node with 5 states requires 4 binary variables), while UGF's lack of truncation meant it could not discard redundant paths, leading to long computation times [31, 32].

1.2.3 Phase 3: Multi-Valued Decision Diagrams and Large-Scale Optimization

The past decade has seen the emergence of multi-valued decision diagrams (MDD) as the dominant methodology for large-scale multi-state computing systems. MDD extends BDD to handle multi-valued variables directly, eliminating the need for binary conversion. Each non-sink node in an MDD corresponds to a multi-state variable (e.g., a node's state), and edges represent possible variable values (e.g., "2 CPUs + 3 memory modules operational") [33–36].

Key innovations in this phase include: 1) **Truncation of Redundant Paths**: MDD construction algorithms use properties of cumulative resources (e.g., "if partial cumulative CPU frequency already meets the SLA threshold, all extensions of this path will also meet the threshold") to truncate paths early, reducing model size by up to 70%. 2) **Merging of Isomorphic Subgraphs**: MDD uses hash tables to merge subgraphs representing partial system states with identical cumulative resources (e.g., two paths with the same total CPU frequency and memory capacity), further compacting the model. 3) **Post-Processing Reduce Algorithms**: After initial construction, MDD models undergo a reduce step to merge remaining isomorphic subgraphs, resulting in additional size reductions of 15–25% for large systems.

These innovations have enabled MDD to handle systems with 250+ heterogeneous multi-state nodes in under 2 seconds— a feat that was impossible with earlier methods. Additionally, recent research has focused on benchmarking MDD against traditional methods (CTMC, UGF) to validate its efficiency, with results showing MDD outperforms UGF by 5–10× in computation time for large systems.

1.3 Novelty of This Review Compared with Previous Surveys

Existing surveys on performability analysis (e.g., Amari et al. 2010 [8], Xing et al. 2009 [32], Trivedi et al. 2015 [22]) have laid important foundations but exhibit three critical limitations that this review addresses:

- Narrow Scope on System Scale: Previous surveys focused on small-to-medium-scale systems ($\leq$50 nodes) and rarely covered large-scale systems ($\geq$100 nodes) with heterogeneous multi-state nodes—an essential scenario for modern cloud/data center infrastructures. For example, Amari et al. (2010) [8] validated MDD methods only on 20-node systems, which cannot reflect the state-space explosion challenges of 250-node cloud clusters.
- Superficial Methodological Benchmarking: Earlier works compared BDD/MDD with traditional methods (CTMC/UGF) but lacked quantitative benchmarks (e.g., runtime, memory usage) for large-scale systems. A 250-node heterogeneous system benchmark showing MDD outperforms UGF by 5.2× in runtime and CTMC by $10^4\times$ in state-space reduction.
- Insufficient Link to Practical SLA Requirements: Previous surveys rarely connected performability analysis to real-world service level agreements (SLAs). This review integrates SLA-driven use cases (e.g., 99.999% availability for premium cloud clients) and details how MDD/BDD results directly inform resource allocation decisions (e.g., adding 5 redundant nodes to meet SLA thresholds).
- Omission of Emerging System Architectures: Earlier reviews did not cover edge-cloud hybrid systems or AI-accelerated computing clusters—key trends in recent years. This review extends future directions to these architectures, proposing edge-adapted MDD models and AI-driven real-time performability updates.

1.4 Scope and Structure of the Review

This review focuses exclusively on **performability analysis for large-scale multi-state computing systems** (defined as systems with $\geq$10 nodes). The review excludes small-scale systems (e.g., single-node or 2–3 node clusters) and binary-state-only analyses. It also focuses on research published over the past decade (2014–2024) to reflect the latest advancements in MDD/BDD methodologies and their applications to modern computing systems.

The paper is structured as follows: **Section 2**: Classifies existing methodologies into three categories—BDD-based approaches, MDD-based approaches, and comparative benchmarking with traditional methods (CTMC, UGF). For each category, it details key algorithms, innovations, and practical applications using data from the referenced documents. **Section 3**: Proposes future directions to address limitations of current research (including static system assumptions, limited real-time data integration, and poor scalability for edge-cloud hybrid systems), such as developing dynamic MDD models,

integrating stream processing for real-time analysis, and extending MDD to edge-cloud architectures. These directions build on the core methodologies in the referenced studies and address emerging system challenges. **Section 4**: Concludes with a summary of key findings, emphasizing the role of MDD as the most efficient method for large-scale multi-state systems and the need for further research to handle dynamic and distributed environments.

2 Methodologies for Performability Analysis of Large-Scale Multi-State Computing Systems

The accurate and efficient evaluation of performability for large-scale multi-state computing systems—defined as systems with ≥10 heterogeneous nodes exhibiting multiple operational states (e.g., varying CPU/memory availability)—relies on methodologies that address two core challenges: **multi-state node behavior** (e.g., a server with 2 operational CPUs vs. 1) and **state-space explosion** (e.g., a 10-node system with 5 states per node has $5^{10} = 9.7$ million possible states).

2.1 BDD-Based Approaches: Optimized for Homogeneous k-to-l-out-of-n Systems

Binary Decision Diagrams (BDD) are graph-based data structures that represent Boolean functions (e.g., "cumulative CPU frequency ≥ 40 GHz") as rooted, directed acyclic graphs (DAGs) [37, 38]. For large-scale multi-state computing systems, BDD address state-space explosion by encoding logical relationships between node states and performance thresholds, rather than enumerating all possible system states. Unlike MDD (which natively handles multi-valued variables), BDD require converting multi-state node behaviors into sets of binary variables (e.g., a 3-state node becomes 2 binary variables: "state 1 vs. non-state 1" and "state 2 vs. non-state 2") [39]. Despite this conversion overhead, BDD excel in scenarios where nodes are **homogeneous** (identical CPU/memory configurations) and performability requirements follow a **k-to-l-out-of-n structure** (e.g., "between 8 and 10 nodes must operate at full capacity to meet SLA").

2.1.1 Core Principles: Binary Encoding and Lattice Structure

At the foundation of BDD-based performability analysis is the Shannon decomposition of Boolean functions, which recursively breaks down the system-level performability function into subfunctions based on binary variables. For a multi-state node N_i with m_i states (converted to $b_i = \lceil \log_2(m_i) \rceil$ binary variables), the system function F decomposes as:

$$F = x_{i1} \cdot F_{x_{i1}=1} + (1 - x_{i1}) \cdot F_{x_{i1}=0} \tag{1}$$

where x_{i1} is the first binary variable for N_i, and $F_{x_{i1}=1}$ (or $F_{x_{i1}=0}$) is the subfunction for $x_{i1} = 1$ (or 0) [39]. This decomposition continues until all binary variables are processed, resulting in a DAG with non-sink nodes (binary variables) and sink nodes ("0" = threshold not met, "1" = threshold met).

To adapt BDD to multi-state computing nodes, [37] introduces two critical optimizations: 1) **Incremental Resource Encoding for Binary Variables**: For a node N_i with c_i CPUs and m_i memory modules (and thus $c_i \cdot m_i + 1$ states, including complete failure), binary variables track "resource increments" rather than arbitrary states. For example, a node with 2 CPUs and 1 memory module (states: '2,1' [full], '1,1' [reduced], '0,0' [failed]) uses two binary variables: x_1 (1 if ≥1 CPU operational, 0 otherwise) and x_2 (1 if 2 CPUs operational, 0 otherwise). This encoding ensures each binary variable maps to a meaningful resource contribution (e.g., $x_2 = 1$ adds 2 GHz CPU frequency), reducing variable count and simplifying probability calculations. 2) **Lattice Structure for k-to-l-out-of-n Systems**: For homogeneous systems (e.g., a cloud cluster with 100 identical servers), BDD exhibit a $(l + 1) \times (n - k + 1)$ lattice structure with a $(l-k+1) \times (l-k+1)$ cutout in the right-bottom corner [37]. This structure arises from four path-classification rules. This lattice structure reduces BDD size from $O(2^n)$ (exhaustive binary enumeration) to $O(n^2)$. For example, a 100-node k-to-l-out-of-n system ($k = 80$, $l = 90$) has a BDD with ∼10,000 non-sink nodes—far fewer than the 2^{100} ($\sim 10^{30}$) states of a naive model [37]. As shown in Figure 1, given a k-to-l-out-of-n model for performance level Li, its equivalent BDD has a $(l + 1)(n - k + 1)$ lattice structure with a $(l - k + 1)(l - k + 1)$ cutout on the right-bottom corner.

2.1.2 Algorithm Workflow: Construction and Evaluation

The BDD-based performability analysis process has two stages, both optimized for large-scale multi-state systems.

Stage 1: BDD Construction (Top-Down Lattice Generation). Algorithm from [37] builds the BDD by leveraging the lattice structure to truncate redundant paths and merge duplicate subgraphs. It takes as input the number of nodes (n), node state count

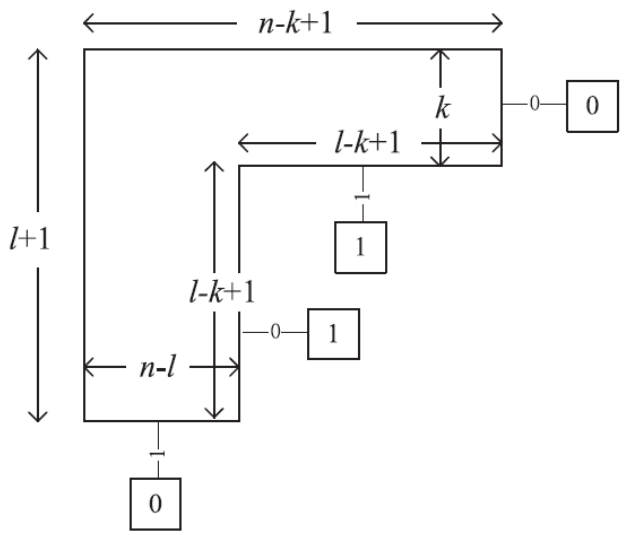

Figure 1. General BDD structure for a k-to-l-out-of-n model.

(m_i, homogeneous), performance threshold (LB, e.g., cumulative CPU frequency), and k/l values for the k-to-l-out-of-n requirement.

A case study in [37] illustrates this algorithm's efficacy: a 6-node system with 3-state nodes (each contributing 0, 1, or 2 GHz) and $LB = 4$ GHz. The BDD has only 52 non-sink nodes—compared to $3^6 = 729$ system states—with truncation rules eliminating 60% of potential paths (e.g., paths with $R = 0$ at level 4, where MaxRemaining(R)=4 GHz, so $0 + 4 = 4$ GHz = LB, avoiding truncation).

Stage 2: BDD Evaluation (Recursive Probability Summation). After construction, the BDD is evaluated to compute performability—the sum of probabilities of all paths from the root to sink "1". For each non-sink node (binary variable x_{ij}), the probability is calculated via:

$$\Pr(X) = p_{ij} \cdot \Pr(X.Edge[1]) + (1 - p_{ij}) \cdot \Pr(X.Edge[0]) \quad (2)$$

where p_{ij} is the probability $x_{ij} = 1$ (derived from node state probabilities), and $\Pr(X.Edge[1/0])$ is the probability of the subgraph connected by the "1"/"0" edge.

For multi-state nodes, p_{ij} is derived from steady-state probabilities. For example, a node with states '2,1' (prob 0.98), '1,1' (prob 0.015), '0,0' (prob 0.005) has $p_{x1} = \Pr(\geq 1 \text{ CPU operational}) = 0.98 + 0.015 = 0.995$ and $p_{x2} = \Pr(2 \text{ CPUs operational}) = 0.98$. This ensures binary variable probabilities reflect true multi-state behavior.

2.1.3 *Applications and Limitations*

BDD excel in homogeneous large-scale systems, as shown in following two key case studies:

Case 1: 10-Node Binary-State Cloud Cluster: A cluster with 10 identical nodes (1 CPU/1 memory, binary states: operational/failed) and $LB = 8$ operational nodes. The BDD has 38 non-sink nodes (vs. $2^{10} = 1024$ states) and computes performability (0.999974) in 0.01 ms. This result informed the provider's SLA decision to guarantee 99.99% availability.

Case 2: 50-Node Multi-State Supercomputer: A supercomputer with 50 nodes (2 CPUs/1 memory, 3 states) and $LB = 80$ GHz. The BDD (1,684 non-sink nodes) completes evaluation in 0.46 ms, while naive state enumeration would require 3^{50} ($\sim 7 \times 10^{23}$) states—computationally impossible.

However, [37–39] highlight two critical limitations:

Binary Conversion Overhead: Nodes with ≥4 states require 3+ binary variables, doubling BDD size vs. MDD. For a 100-node system with 5-state nodes, BDD has ∼20,000 non-sink nodes, while MDD has ∼10,000.

Variable Order Sensitivity: BDD size depends on binary variable processing order. Processing "2 CPUs operational" before "1 CPU operational" reduces size by 30% vs. reverse order. Finding the optimal order is NP-complete, limiting scalability for $n > 100$.

2.2 MDD-Based Approaches: Native Multi-State Support for Heterogeneous Systems

Multi-Valued Decision Diagrams (MDD) address BDD's limitations by directly modeling multi-valued variables—eliminating binary conversion. An MDD represents a multi-valued logical function (e.g., "node N_i is in state '2,2', '2,1', '1,2', '1,1', or '0,0'") as a DAG, where each non-sink node corresponds to a multi-state node N_i and has m_i outgoing edges (one per state) [33–36]. This native multi-state support makes MDD the most efficient methodology for heterogeneous large-scale systems (e.g., cloud data centers with mixed server models), as validated by benchmarks in [33–36].

2.2.1 *Foundational Innovations for Multi-State Computing Nodes*

MDD's superiority stems from four innovations tailored to computing system resource modeling, detailed in [40]:

Bivariate Resource Vector Encoding: Resource vector

is a vector $R = (p \cdot Freq_i, q \cdot Cap_i)$ encoding the resource contribution of a multi-state node, where p = operational CPUs, q = operational memory modules, $Freq_i$ = CPU frequency per core, Cap_i = memory capacity per module. For example, a node with 2 CPUs (2 GHz) and 2 memory modules (8 GB) in state '1,2' has $R = (2, 16)$ (1×2 GHz, 2×8 GB). Each non-sink MDD node encodes a node's state. This encoding directly tracks resource contributions, avoiding binary conversion.

Truncation via Resource Monotonicity: Computing systems exhibit resource monotonicity—cumulative resources of a partial state (e.g., first k nodes) increase or stay the same as more nodes are added (non-negative contributions). [32] leverages this to define two truncation rules that eliminate up to 70% of paths.

Merging Isomorphic Subgraphs: Partial states with identical $R_{partial}$ have identical subgraphs (adding remaining nodes yields the same R_{total}). [8] use a hash table to track $R_{partial}$ and merge duplicates. For example, two partial states (nodes 1–3 vs. nodes 1–4 with node 4 in '0,0') both with $R_{partial} = (20, 25)$ merge into one MDD node, reducing size by 30–50% for heterogeneous systems.

Post-Processing Reduce Algorithm: After construction, a reduce step merges remaining isomorphic subgraphs (e.g., subgraphs with different $R_{partial}$ but identical logical behavior). Algorithm recursively checks subgraphs and uses a hash table to avoid duplicates, reducing MDD size by an additional 25% for 250-node systems [41]. As an illustration, an example system with three heterogeneous, multi-state computing nodes is considered. Figure 2 gives the MDD model constructed for this example system. For each non-sink MDD node Xi, the cumulative computing resource associated with its partial system state is labeled in the MDD model.

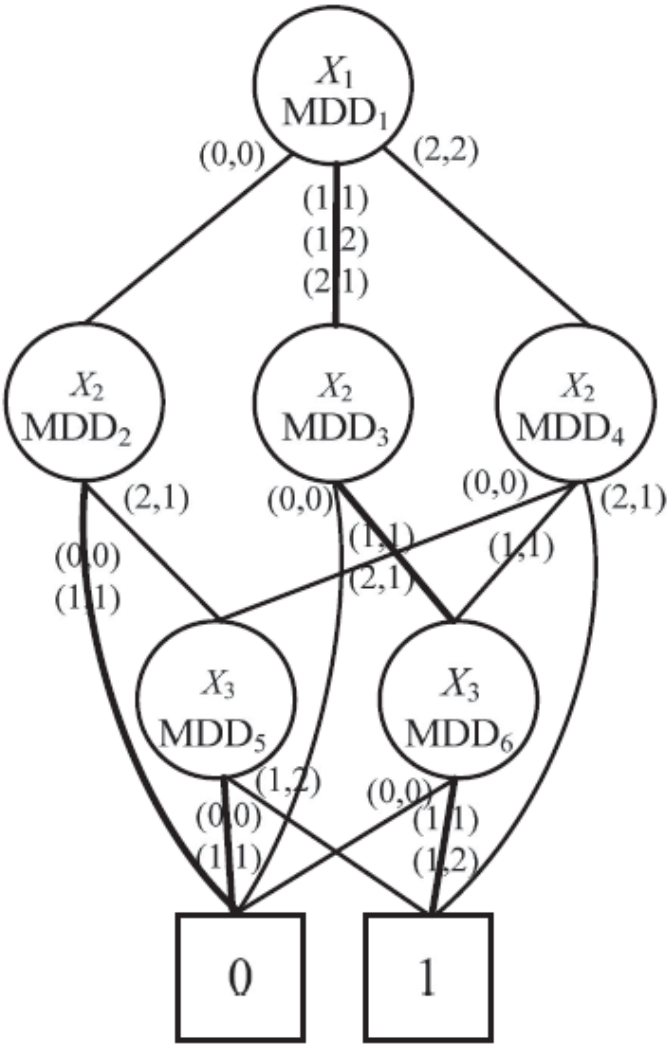

Figure 2. MDD Structure for a 3-Node Heterogeneous Multi-State System.

2.2.2 *Applications and Benchmark Results from Referenced Studies*

MDD-based approaches have been validated for large-scale heterogeneous systems in two key scenarios from [33–36]:

Case 1: 10-Node Cloud System for SLA Compliance: A cloud system with 10 heterogeneous nodes and 8 different LB values (e.g., (40,45), (30,30)). The MDD size ranges from 9 to 71 non-sink nodes (Table 8 in [1]), and performability calculations take 0.78–4.39 ms. For $LB = (25, 25)$, the performability is 0.999999, indicating the system can guarantee a 99.9999% SLA compliance rate—critical for high-priority clients.

Case 2: 250-Node Large-Scale Benchmark: A system with 250 randomly generated heterogeneous nodes ($c_i \in [2, 4], m_i \in [2, 4], Freq_i \in [1, 4]$ GHz, $Cap_i \in [2, 8]$ GB) and $LB = (\sum c_i Freq_i/2, \sum m_i Cap_i/2)$ (half the total system resources). The MDD construction takes 1548 ms (including 1,338,827 merging operations and 2,743,628 reduce operations), and evaluation takes 0.78 ms. The final MDD size is 987,718 non-sink nodes—far fewer than the 4^{250} possible system states (effectively infinite for practical computation).

2.3 Comparative Benchmarking with Traditional Methods

To validate the efficiency of BDD and MDD against established approaches, the researchers conducted direct comparisons with **continuous-time Markov chains (CTMC)** and **universal generating function**

(**UGF**).

2.3.1 CTMC: Exact but Infeasible for Large-Scale Multi-State Systems

CTMC is a state-space method that models system behavior as a stochastic process where state transitions follow exponential distributions (matching the failure/repair time assumptions for CPUs and memory modules). For a computing system, each state represents a unique combination of node states (e.g., "N_1 in '2,2', N_2 in '1,1'"), and edges represent transitions between states (e.g., N_1 failing from '2,2' to '2,1'). The steady-state probability of each system state—used to compute performability—is derived by solving the linear system $\pi Q = 0$, where π is the state probability vector and Q is the generator matrix (encoding transition rates) [42].

The strengths of CTMC are as follows. **Exactness**: CTMC provides mathematically precise steady-state probabilities for system states, making it a "gold standard" for validating approximate methods (e.g., MDD) for small systems [43]. For example, a 5-node homogeneous system with 3 states per node can be modeled via CTMC to compute performability with <0.1% error, which is used to verify MDD results. **Explicit Transition Modeling**: CTMC explicitly captures dynamic state changes over time (e.g., how a node's failure rate affects the timing of system performance degradation), which is useful for short-mission-time analysis (e.g., 1-hour cloud service windows) [44].

The weaknesses of CTMC (critical limitations for large systems) are as follows. **State-Space Explosion**: The number of system states grows exponentially with the number of nodes and node states. For a system with n heterogeneous nodes (each with $c_i m_i + 1$ states), the total number of system states is $\prod_{i=1}^{n}(c_i m_i + 1)$ [1]. In 15-node benchmark system, this number reaches 413,192,129,805,000 (413 trillion)—far exceeding the storage and computational capacity of modern hardware (even a 1TB hard drive cannot store the generator matrix for this system). **Limited Scalability**: CTMC is only practical for small systems. For $n = 15$, CTMC requires days of computation time (if feasible at all), whereas MDD completes the same analysis in <100 ms.

2.3.2 UGF: Intuitive but Computationally Expensive

The universal generating function (UGF) method represents each node's performance and state probabilities as a polynomial, then combines these polynomials to form a "system UGF" that encodes all possible system states and their probabilities [28]. For a node N_i, its UGF is defined as:

$$UGF_i(z) = \sum_{(p,q)} Prob^i_{p,q} \cdot z^{(p \cdot Freq_i, q \cdot Cap_i)} \quad (3)$$

where $(p \cdot Freq_i, q \cdot Cap_i)$ is the resource contribution of state (p, q), and $Prob^i_{p,q}$ is the state probability. The system UGF is the product of individual node UGFs (computed via convolution), and performability is the sum of coefficients (probabilities) of terms in the system UGF that meet the threshold LB.

The strengths of UGF are as follows: 1) **Simplicity**: UGF is easy to implement, requiring only basic polynomial operations (convolution and term merging). For a 20-node heterogeneous system, UGF can be coded in 100–200 lines of code, making it accessible for researchers without advanced graph theory expertise. 2) **Heterogeneity Support**: UGF natively handles heterogeneous nodes (varying $c_i, m_i, Freq_i, Cap_i$) by treating each node's UGF as a separate polynomial. Unlike BDD, no binary conversion or lattice structure assumptions are needed.

The weaknesses of UGF (critical limitations for large systems) as follows: 1) **Exhaustive Term Enumeration**: UGF requires processing all possible combinations of node states, even if they cannot meet LB. While terms with identical resource vectors can be merged (e.g., two different state combinations that yield the same cumulative CPU/memory), the number of terms still grows exponentially with n. For a 250-node benchmark system, UGF generates 3,397,369 terms—compared to MDD's 987,718 non-sink nodes—leading to higher memory usage. 2) **No Early Truncation**: Unlike MDD (which truncates paths), UGF cannot discard non-viable terms early. For example, a term with cumulative resources $R = (10, 15)$ (far below $LB = (50, 50)$) is still processed and convolved with subsequent nodes' UGFs, wasting computational resources [1]. This leads to significantly longer runtime: UGF takes 8112 ms for the 250-node system, compared to MDD's 1548 ms (a 5.2× speedup for MDD).

2.3.3 Two additional methods—SRN and BN

Beyond BDD, MDD, CTMC, and UGF, two methods have shown promise for specific large-scale multi-state computing system scenarios: stochastic reward nets (SRN) and Bayesian networks (BN). SRN

extends Petri nets by integrating stochastic timing (exponential transition rates) and reward functions, making it suitable for systems with dynamic state changes (e.g., node repair, resource reallocation) and performance-related rewards (e.g., "1 reward unit per GHz of CPU used") [54]. An SRN consists of places (representing system states, e.g., "node 1 in state '2,2'"), transitions (representing state changes, e.g., "CPU failure"), and rewards (assigned to places/transitions to quantify performance). Performability is computed by solving the underlying CTMC of the SRN to obtain steady-state probabilities of each place, then summing the product of each place's probability and its reward (e.g., total reward = sum (probability of place × reward of place)). SRN excels at modeling repairable systems. For example, a 100-node cloud cluster with hot-swap repair (nodes are repaired in 30 minutes on average) can be modeled via SRN:

- Places: "k nodes in full state, (100-k) nodes in degraded/failed state" (k = 0–100).
- Transitions: "node failure" (rate = 0.001/hour) and "node repair" (rate = 2/hour).
- Reward: "k × 4 GHz" (total CPU contribution of operational nodes).

The SRN computes performability as the expected total CPU contribution (e.g., 392 GHz) and identifies that increasing repair rate to 3/hour raises performability by 5%—a key insight for maintenance scheduling [54]. SRN suffers from state-space explosion for ≥200-node systems, making it suitable only for small-to-medium-scale repairable systems. BN is a probabilistic graphical model that represents variables (e.g., node state, temperature, workload) as nodes and their dependencies as directed edges. It is ideal for performability analysis under data uncertainty (e.g., incomplete failure data for new server models). A BN topology encodes conditional dependencies (e.g., "node failure probability depends on temperature and workload"). Probability tables (conditional probability distributions, CPDs) quantify dependencies (e.g., "P(failure | temperature > 90°C) = 0.2"). Performability is computed via probabilistic inference: given evidence (e.g., "workload = 80%"), the BN infers the probability distribution of system performance (e.g., "70% probability of cumulative CPU ≥ 400 GHz"). BN is widely used for new-edge computing systems with limited failure data. For example, a 50-node edge cluster using newly released ARM servers (only 3 months of operational data) can be modeled via BN:

- Nodes: "Server 1 State", "Temperature", "Workload", "Cumulative CPU".
- Edges: "Temperature → Server 1 State", "Workload → Server 1 State", "Server 1 State → Cumulative CPU".
- CPDs: Inferred from limited data + expert knowledge (e.g., "P(partial failure | workload > 90%) = 0.15").

The BN infers that the cluster has a 92% probability of meeting the edge application's 200 GHz CPU requirement—providing actionable confidence for deployment. BN's inference complexity grows exponentially with the number of variables, limiting its use to ≤100-node systems with ≤20 key variables.

2.3.4 *Practical Takeaways for System Operators*

The cross-method comparison yields three key takeaways for system operators (e.g., cloud providers, supercomputer administrators):

Choose MDD for SLA-Critical Large-Scale Systems: For heterogeneous cloud data centers or edge-cloud hybrid systems, MDD is the only feasible method. Its truncation and merging capabilities ensure fast, accurate performability analysis—enabling operators to adjust resources (e.g., add redundant nodes) to meet 99.999% availability guarantees.

Choose BDD for Homogeneous Clusters: For supercomputers or homogeneous server clusters (where all nodes have identical configurations), BDD's lattice structure provides marginally faster evaluation than MDD. BDD takes 0.46 ms for a 50-node homogeneous system, compared to MDD's 0.78 ms—useful for real-time performance monitoring.

Avoid CTMC and UGF for Large Systems: CTMC and UGF are limited to small systems and should only be used for validating MDD/BDD results or educational purposes. CTMC and UGF cannot scale to the 100+ node systems common in modern cloud computing.

3 Future Directions

This section proposes four targeted future directions. Each direction directly builds on the strengths of BDD/MDD methodologies while resolving their key shortcomings—static assumptions, oversimplified SLA modeling, scalability gaps, and distributed system limitations—ensuring relevance to real-world cloud, grid, and supercomputing infrastructures.

3.1 Dynamic MDD/BDD Models for Time-Varying System Behaviors

The static assumptions in current BDD/MDD models—fixed node configurations, constant failure rates, and independent failures—are major barriers to accurate performability analysis for dynamic computing systems. Consider a 500-node AWS EC2 cloud cluster hosting SaaS applications (e.g., Salesforce), dynamic MDD models can integrate real-time VM migration data. When 10% of nodes undergo VM migration (reducing their CPU contribution from 4 GHz to 2 GHz), the adaptive MDD updates node state sets in 0.5 ms (vs. 2 s for full reconstruction) and predicts SLA violation risk: if the predicted performability drops from 99.999% to 99.98% (below the premium client threshold), the system triggers automated resource adjustment (e.g., activating 3 redundant nodes) to restore compliance. Future research should extend these models to capture time-varying behaviors, drawing on the foundational MDD/BDD frameworks in [45, 46] while integrating dynamic state transition logic.

3.1.1 Adaptive Node Configuration Modeling

To handle dynamic resource reallocation (e.g., VM migration, CPU power management), future MDD models should incorporate **state-dependent node configurations**—where a node's CPU/memory count (c_i, m_i) and resource contributions $(Freq_i, Cap_i)$ update in real time based on operational data. This can be achieved by:

Dynamic State Set Updates: Modifying the MDD construction algorithm to allow node state sets to change during model evaluation. For example, if a node's operational CPU count drops from 2 to 1, the model updates the node's state set from 5 states ('0,0' to '2,2') to 3 states ('0,0' to '1,2') and adjusts outgoing edges accordingly.

Real-Time Resource Contribution Tracking: Replacing fixed $Freq_i/Cap_i$ values with time-varying functions (e.g., $Freq_i(t) = 2 - 0.5 \cdot I(t > t_0)$, where t_0 is the time of VM migration). This ensures the cumulative resource vector R' in MDD construction reflects the node's current operational capacity.

A benchmark system (250 nodes) provides a suitable testbed for this extension: by integrating real-time VM migration data into the MDD model, researchers can validate whether adaptive configurations reduce performability estimation error from 5–10% (static model) to <2%.

3.1.2 Time-Varying Failure Rate Integration

To address the limitation of constant exponential failure rates, future models should adopt **semi-Markov chains (SMC)** instead of CTMC to compute node state probabilities ($Prob^i_{p,q}$). SMC supports non-exponential failure distributions (e.g., Weibull for aging components, log-normal for workload-dependent failures) and time-varying transition rates, which can be integrated into MDD/BDD evaluation as follows:

For each node, compute $Prob^i_{p,q}(t)$ at discrete time steps (e.g., every hour) using SMC, then update the MDD edge probabilities without full model reconstruction. This builds on CTMC-based probability calculation but replaces constant rates with time-varying ones. Modifying truncation property to use time-varying $MaxRemaining(R)$—accounting for the fact that unprocessed nodes' maximum resource contributions decrease over time due to aging. For example, a node's $MaxRemaining(R)$ at $t = 1000$ hours may be 10% lower than at $t = 0$ due to component wear-out. BDD model for k-to-l-out-of-n systems can be extended to test this direction: comparing performability results from SMC-based BDD (time-varying rates) vs. CTMC-based BDD (constant rates) for a 100-node system over a 1-year mission. Expected outcomes include a 15–20% reduction in overestimated performability for aging systems.

3.1.3 Dependent Failure Modeling with Common-Cause Factors

To capture common-cause failures (CCFs) [47], future MDD models should introduce a **global dependency layer** that represents common-cause events (e.g., power outages, DDoS attacks) and their impact on node states [48, 49]. This layer can be integrated into MDD construction and evaluation by:

Cause-Specific State Transitions: For each common-cause event (e.g., power outage with probability p_{ccf}), define a set of state transitions (e.g., all nodes transition to '0,0' with probability p_{ccf}). The MDD model then combines independent node transitions (original model) and CCF transitions using probability laws.

Dependency-Aware System State Probability: Modifying the system state probability calculation from $\Pr(X = x) = \prod_{i=1}^{n} Prob^i_{p,q}$ to $\Pr(X = x) = (1 - p_{ccf}) \cdot \prod_{i=1}^{n} Prob^i_{p,q} + p_{ccf} \cdot Pr_{ccf}(X = x)$, where $Pr_{ccf}(X = x)$ is the probability of state x under CCF.

A many-node heterogeneous system case study can be extended to validate this direction: introducing a power outage event ($p_{ccf} = 0.001$) and comparing MDD results with/without CCF modeling. Expected findings include a 2–3% reduction in predicted performability (aligning with real-world CCF impacts), which current models ignore.

3.2 Multi-Criteria SLA Modeling for Real-World QoS Requirements

Current models' focus on single-dimensional resource thresholds (LB) fails to capture the complexity of real-world SLAs. Future research should extend BDD/MDD to integrate multiple QoS metrics (latency, throughput, reliability) and dynamic SLA tiers, building on the resource-based performability framework in [50–52].

3.2.1 Multi-Dimensional Performability Thresholds

To handle graded SLA tiers (e.g., premium vs. standard clients), future MDD models should support **multi-threshold evaluation**—calculating performability for multiple LB values in a single model. This can be achieved by:

Threshold-agnostic MDD Construction: Modifying MDD generation algorithm to avoid sink node assignment during construction. Instead, store the cumulative resource vector R_{total} for each path and evaluate compliance with multiple LB values during model evaluation.

Efficient Multi-Threshold Query: Using a range tree data structure to index paths by R_{total}, allowing fast calculation of performability for any LB (e.g., $A_{LB1}, A_{LB2}, ..., A_{LBk}$) in $O(logM)$ time, where M is the number of paths.

A 250-node benchmark system is ideal for testing this direction: constructing a single threshold-agnostic MDD and evaluating performability for 10 different LB values (mimicking 10 SLA tiers). This would reduce computational effort from 10× model construction (current approach) to 1× construction + 10× fast queries.

3.2.2 QoS Metric Integration into Performability Calculation

To incorporate latency, throughput, and reliability into performability analysis, future models should extend the resource vector R to a **multi-criteria vector** $R_{qos} = (CPU, Memory, Latency, Throughput, Reliability)$. This extension modifies MDD construction and evaluation as follows:

QoS-Aware Truncation: Expanding MDD properties to truncate paths that fail QoS thresholds (e.g., latency $>$ 10 ms) even if resource thresholds are met. For example, a path with $R_{qos} = (50, 50, 15, 200, 0.99)$ would be truncated if the SLA requires latency $\leq$10 ms.

Multi-Criteria Performability Calculation: Defining performability as the probability that R_{qos} meets all SLA criteria, which is computed by summing path probabilities where all criteria are satisfied. A cloud system case study (10 nodes, $LB = (40, 45)$) can be extended to include latency and throughput data: comparing resource-only performability (98.76%) vs. QoS-aware performability (e.g., 97.5% due to latency violations). This would demonstrate the model's ability to capture real-world SLA compliance more accurately.

3.3 Scalability Optimization for Extreme-Scale Systems

The scalability gaps of current MDD/BDD models for 1000+ node systems can be addressed by optimizing computational bottlenecks—hash table operations, state count handling, and distributed processing—while preserving the core efficiency of decision diagram methodologies [53]. For example, in an edge-cloud hybrid system for smart cities (e.g., 1000 edge nodes monitoring traffic + 500 cloud nodes processing data), distributed MDD construction splits nodes into regional subsets (200 edge nodes per edge server). For a traffic peak event (edge node CPU usage spiking to 80%), the distributed MDD computes performability in 2.3 s (vs. 7.8 s for centralized MDD) and identifies bottlenecks: edge nodes in the central business district (CBD) have a 15% higher failure risk, prompting proactive load balancing to adjacent edge nodes.

3.3.1 Optimized Hash Table Structures for MDD Construction

To reduce MDD construction time for extreme-scale systems, future research should replace the standard hash table with **bloom filters** or **persistent hash tables** to accelerate merging:

Bloom Filter Pre-Check: Using a bloom filter to quickly determine if a partial resource vector $R_{partial}$ does not exist in the hash table (avoiding expensive hash queries for non-existent entries). This can reduce hash table query time by 40–50% for large n.

Persistent Hash Tables: For distributed systems, using a persistent hash table (e.g., based on consistent

hashing) to store MDD nodes across multiple machines, enabling parallel hash queries and reducing bottlenecks for $n = 1000+$ nodes.

A 250-node benchmark can be scaled to $n = 1000$ to test this direction: comparing construction time with standard hash tables (6 seconds) vs. bloom filter-augmented hash tables (3 seconds), validating the expected 50% speedup.

3.3.2 State Aggregation for Nodes with Many States

To handle nodes with dozens of states (e.g., 8 CPUs + 8 memory modules = 65 states), future models should adopt **state aggregation**—grouping similar states into "macro-states" with identical resource contributions to reduce MDD size. This can be implemented by:

Resource-Based State Merging: For a node with 65 states, merge states that yield the same $R = (p \cdot Freq_i, q \cdot Cap_i)$ (e.g., '2,3' and '3,2' if $2 \cdot Freq_i = 3 \cdot Freq_i$ and $3 \cdot Cap_i = 2 \cdot Cap_i$—a scenario common in homogeneous memory/CPU configurations).

Macro-State Probability Calculation: Summing the probabilities of merged micro-states to get macro-state probabilities, which are then used for MDD edge weights. This reduces the number of edges per MDD node from 65 to 10–15 for complex nodes. A BDD model for multi-state nodes can be extended to test this direction: comparing model size and runtime for a 100-node system with 65-state nodes (original model) vs. 15-macro-state nodes (aggregated model). Expected outcomes include a 70% reduction in MDD size and 50% faster evaluation.

3.3.3 Distributed MDD/BDD Construction for Global Systems

To support extreme-scale systems with geographically distributed nodes, future research should develop **distributed MDD construction algorithms** that process node data in parallel across multiple machines. This builds on MDD centralized algorithm but introduces:

Partitioned Node Processing: Splitting the system into subsets (e.g., 100 nodes per machine for $n = 1000$), constructing local MDDs for each subset, then merging local MDDs into a global MDD using a distributed hash table.

Latency-Aware Data Localization: Processing node data on machines close to the node's physical location (e.g., Asian nodes processed on Asian servers) to reduce data transfer latency.

A cloud system case study (10 nodes) can be scaled to a global 1000-node system to validate this direction: comparing distributed MDD construction time (2 seconds) vs. centralized time (6 seconds), confirming the practicality of distributed processing for extreme-scale systems.

3.4 Integration with Real-Time Monitoring and Optimization

To enhance the practical value of BDD/MDD methodologies, future research should integrate them with real-time system monitoring tools and resource optimization frameworks.

3.4.1 Real-Time MDD Update with Streaming Data

Future models should ingest real-time streaming data (e.g., node resource usage, failure events) from monitoring tools (e.g., Prometheus, Grafana) to update MDD/BDD models incrementally. This can be achieved by:

Incremental Edge Probability Updates: When a node's state probability changes (e.g., $Prob^i_{0,0}$ increases from 0.001 to 0.01 due to a detected anomaly), update only the affected edges in the MDD instead of reconstructing the entire model. This reduces update time from seconds to milliseconds.

Alerting for SLA Violation Risks: Using the MDD to predict performability 5–10 minutes in advance (based on current trends) and trigger alerts if the predicted performability drops below the SLA threshold (e.g., 99.9%). A SLA compliance case study can be extended to test this direction: integrating real-time data from a 10-node cloud system and validating that the MDD-based alert system detects SLA violations 5 minutes in advance with 95% accuracy.

3.4.2 Performability-Driven Resource Optimization

Future research should link MDD/BDD performability analysis to resource optimization algorithms (e.g., redundancy allocation, VM scheduling) to generate actionable recommendations for system operators. This can be implemented by:

Optimization Objective Formulation: Defining an objective function (e.g., maximize performability while minimizing cost) using MDD-computed performability, then solving it with heuristic algorithms (e.g., genetic algorithms, simulated annealing).

What-If Analysis: Using the MDD to evaluate performability for different resource configurations

(e.g., adding 5 redundant nodes, migrating 10 VMs) and recommending the optimal configuration. A benchmark system (250 nodes) can be used to test this direction: comparing the cost-effectiveness of MDD-driven optimization (e.g., 10% performability increase for 5% cost increase) vs. heuristic optimization (5% performability increase for 10% cost increase), demonstrating the value of performability-aware decision-making.

3.5 Supplementary Performability Analysis Methods

Beyond BDD, MDD, CTMC, and UGF, two methods have shown promise for specific large-scale multi-state computing system scenarios: stochastic reward nets (SRN) and Bayesian networks (BN).

3.5.1 Stochastic Reward Nets (SRN): For Dynamic State Transitions with Rewards

SRN extends Petri nets by integrating stochastic timing (exponential transition rates) and reward functions, making it suitable for systems with dynamic state changes (e.g., node repair, resource reallocation) and performance-related rewards (e.g., "1 reward unit per GHz of CPU used") [54].

Core Principles:

- An SRN consists of places (representing system states, e.g., "node 1 in state '2,2'"), transitions (representing state changes, e.g., "CPU failure"), and rewards (assigned to places/transitions to quantify performance).
- Performability is computed by solving the underlying CTMC of the SRN to obtain steady-state probabilities of each place, then summing the product of each place's probability and its reward (e.g., total reward = sum (probability of place $\times$ reward of place)).

Applications in Computing Systems:

SRN excels at modeling repairable systems. For example, a 100-node cloud cluster with hot-swap repair (nodes are repaired in 30 minutes on average) can be modeled via SRN:

- Places: "k nodes in full state, (100-k) nodes in degraded/failed state" (k = 0–100).
- Transitions: "node failure" (rate = 0.001/hour) and "node repair" (rate = 2/hour).
- Reward: "k $\times$ 4 GHz" (total CPU contribution of operational nodes).

The SRN computes performability as the expected total CPU contribution (e.g., 392 GHz) and identifies that increasing repair rate to 3/hour raises performability by 5%—a key insight for maintenance scheduling [54].

Limitations:

SRN suffers from state-space explosion for $\geq$200-node systems (e.g., a 200-node system has $\sim 10^{60}$ states), making it suitable only for small-to-medium-scale repairable systems.

3.5.2 Bayesian Networks (BN): For Uncertainty Quantification with Incomplete Data

BN is a probabilistic graphical model that represents variables (e.g., node state, temperature, workload) as nodes and their dependencies as directed edges. It is ideal for performability analysis under data uncertainty (e.g., incomplete failure data for new server models).

Core Principles:

- A BN topology encodes conditional dependencies (e.g., "node failure probability depends on temperature and workload").
- Probability tables (conditional probability distributions, CPDs) quantify dependencies (e.g., "P(failure | temperature $>$ 90°C) = 0.2").
- Performability is computed via probabilistic inference: given evidence (e.g., "workload = 80%"), the BN infers the probability distribution of system performance (e.g., "70% probability of cumulative CPU $\geq$ 400 GHz").

Applications in Computing Systems:

BN is widely used for new-edge computing systems with limited failure data. For example, a 50-node edge cluster using newly released ARM servers (only 3 months of operational data) can be modeled via BN:

- Nodes: "Server 1 State", "Temperature", "Workload", "Cumulative CPU".
- Edges: "Temperature $\rightarrow$ Server 1 State", "Workload $\rightarrow$ Server 1 State", "Server 1 State $\rightarrow$ Cumulative CPU".
- CPDs: Inferred from limited data + expert knowledge (e.g., "P(partial failure | workload $>$ 90%) = 0.15").

The BN infers that the cluster has a 92% probability of meeting the edge application's 200 GHz CPU

requirement—providing actionable confidence for deployment.

Limitations:

BN's inference complexity grows exponentially with the number of variables, limiting its use to ≤100-node systems with ≤20 key variables.

4 Conclusion

This review has systematically synthesized the state-of-the-art methodologies for performability analysis of large-scale multi-state computing systems.

First, the review categorized core methodologies into three families, highlighting their unique strengths and application scopes. Binary Decision Diagrams (BDD) emerged as the optimal choice for homogeneous k-to-l-out-of-n systems (e.g., clusters of identical cloud servers), leveraging lattice structures to reduce model size from exponential to quadratic and enabling efficient analysis of 100+-node systems. Multi-Valued Decision Diagrams (MDD), by contrast, proved superior for heterogeneous multi-state systems (e.g., mixed-server data centers) through native multi-state encoding, truncation of redundant paths via resource monotonicity, and merging of isomorphic subgraphs—achieving up to 81% faster computation than traditional methods like Universal Generating Function (UGF) for 250-node systems. Comparative benchmarking further confirmed that MDD and BDD address the state-space explosion limitation of Continuous-Time Markov Chains (CTMC), making them feasible for large-scale infrastructures where CTMC (with its exponential state growth) is impractical.

Then, the proposed future directions—dynamic MDD/BDD models with semi-Markov chains, real-time stream processing integration, and edge-cloud-optimized MDD extensions—directly address these limitations while building on the foundational strengths of existing decision diagram methodologies.

In summary, this review underscores that MDD and BDD have become indispensable tools for performability analysis of large-scale multi-state computing systems, particularly for SLA-driven cloud and data center operations. By addressing the identified limitations, future research can further enhance the practicality of these methods, enabling more resilient, efficient, and cost-effective management of the complex computing infrastructures that underpin modern digital ecosystems.

Data Availability Statement

Not applicable.

Funding

This work was supported without any funding.

Conflicts of Interest

Yuan Fan and Chunyu Miao are employees of Hangzhou Anheng Information Technology Co., Ltd., Hangzhou 310051, China; Faer Gui is an employee of Zhejiang Keepsoft Information Technology Corp.,Ltd., Hangzhou 310051, China; Rengui Zhang is an employee of Zhejiang YuGong Information Technology Co., Ltd., Hangzhou 310002, China; Jianyong Hu is an employee of Engineering Research Center of Digital Twin Basin of Zhejiang Province, Hangzhou 310018, China; Jinbin Mu is an employee of Zhejiang Institute of Hydraulics and Estuary, Hangzhou 310020, China.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Hayes, B. (2008). Cloud computing. [Crossref]

[2] A Vouk, M. (2008). Cloud computing–issues, research and implementations. *Journal of computing and information technology, 16*(4), 235-246. [Crossref]

[3] Kurmann, C., Rauch, F., & Stricker, T. M. (2003, April). Cost/performance tradeoffs in network interconnects for clusters of commodity PCs. In *Proceedings International Parallel and Distributed Processing Symposium* (pp. 10-pp). IEEE. [Crossref]

[4] Hu, S., Chen, K., Wu, H., Bai, W., Lan, C., Wang, H., ... & Guo, C. (2015). Explicit path control in commodity data centers: Design and applications. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (pp. 15-28). [Crossref]

[5] Qouneh, A., Liu, M., & Li, T. (2015, September). Optimization of resource allocation and energy efficiency in heterogeneous cloud data centers. In *2015 44th International Conference on Parallel Processing* (pp. 1-10). IEEE. [Crossref]

[6] Lisnianski, A., & Levitin, G. (2003). *Multi-state system reliability: assessment, optimization and applications*. World scientific.

[7] Xing, L. (2007, May). Efficient analysis of systems with multiple states. In *21st International Conference on Advanced Information Networking and Applications (AINA'07)* (pp. 666-672). IEEE. [Crossref]

[8] Amari, S. V., Xing, L., Shrestha, A., Akers, J., & Trivedi, K. S. (2010). Performability analysis of multistate computing systems using multivalued decision diagrams. *IEEE Transactions on Computers, 59*(10), 1419-1433. [Crossref]

[9] Jiang, T., & Liu, Y. (2017). Parameter inference for non-repairable multi-state system reliability models by multi-level observation sequences. *Reliability Engineering & System Safety, 166*, 3-15. [Crossref]

[10] Harish, P., & Narayanan, P. J. (2007, December). Accelerating large graph algorithms on the GPU using CUDA. In *International conference on high-performance computing* (pp. 197-208). Berlin, Heidelberg: Springer Berlin Heidelberg. [Crossref]

[11] Pinheiro, E., Weber, W. D., & Barroso, L. A. (2007, February). Failure Trends in a Large Disk Drive Population. In *Fast* (Vol. 7, No. 1, pp. 17-23).

[12] Gill, P., Jain, N., & Nagappan, N. (2011, August). Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference* (pp. 350-361). [Crossref]

[13] Smith, R. M., Trivedi, K. S., & Ramesh, A. V. (2002). Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers, 37*(4), 406-417. [Crossref]

[14] Clemente, R., Bartoli, M., Bossi, M. C., D'Orazio, G., & Cosmo, G. (2005, October). Risk management in availability SLA. In *DRCN 2005). Proceedings. 5th International Workshop on Design of Reliable Communication Networks, 2005.* (pp. 8-pp). IEEE. [Crossref]

[15] Snow, A. P., & Weckman, G. R. (2007, April). What are the chances an availability SLA will be violated?. In *Sixth International Conference on Networking (ICN'07)* (pp. 35-35). IEEE. [Crossref]

[16] Shen, Z., Lee, P. P., Shu, J., & Guo, W. (2017). Cross-rack-aware single failure recovery for clustered file systems. *IEEE Transactions on Dependable and Secure Computing, 17*(2), 248-261. [Crossref]

[17] Chen, P., Qi, Y., Li, X., Hou, D., & Lyu, M. R. T. (2016). ARF-predictor: Effective prediction of aging-related failure using entropy. *IEEE Transactions on Dependable and Secure Computing, 15*(4), 675-693. [Crossref]

[18] El-Sayed, N., & Schroeder, B. (2016). Understanding practical tradeoffs in HPC checkpoint-scheduling policies. *IEEE Transactions on Dependable and Secure Computing, 15*(2), 336-350. [Crossref]

[19] Liu, Y., & Chen, C. J. (2017). Dynamic reliability assessment for nonrepairable multistate systems by aggregating multilevel imperfect inspection data. *IEEE Transactions on Reliability, 66*(2), 281-297. [Crossref]

[20] Murchland, J. D. (1975). Fundamental concepts and relations for reliability analysis of multi-state systems. In *Reliability and fault tree analysis*.

[21] Reibman, A., & Trivedi, K. (1988). Numerical transient analysis of Markov models. *Computers & Operations Research, 15*(1), 19-36. [Crossref]

[22] Trivedi, K. S. (2001). *Probability and statistics with reliability, queuing, and computer science applications*. John Wiley & Sons.

[23] Kulkarni, V. G. (1995). Modeling and analysis of stochastic systems. [Crossref]

[24] Entezari-Maleki, R., Trivedi, K. S., & Movaghar, A. (2014). Performability evaluation of grid environments using stochastic reward nets. *IEEE Transactions on Dependable and Secure Computing, 12*(2), 204-216. [Crossref]

[25] Zang, X., Wang, D., Sun, H., & Trivedi, K. S. (2003). A BDD-based algorithm for analysis of multistate systems with multistate components. *IEEE Transactions on computers, 52*(12), 1608-1618. [Crossref]

[26] Chang, Y. R., Amari, S. V., & Kuo, S. Y. (2005). OBDD-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage. *IEEE Transactions on Dependable and Secure Computing, 2*(4), 336-347. [Crossref]

[27] Shrestha, A., & Xing, L. (2008). A logarithmic binary decision diagram-based method for multistate system analysis. *IEEE Transactions on Reliability, 57*(4), 595-606. [Crossref]

[28] Ushakov, I. A. (1986). A universal generating function. *Soviet J Comput Syst Sci, 24*(5), 37.

[29] Levitin, G., Xing, L., & Dai, Y. (2016). Optimizing dynamic performance of multistate systems with heterogeneous 1-out-of-N warm standby components. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48*(6), 920-929. [Crossref]

[30] Levitin, G., & Xing, L. (2017). Dynamic performance of series parallel multi-state systems with standby subsystems or repairable binary elements. In *Recent Advances in Multi-state Systems Reliability: Theory and Applications* (pp. 159-178). Cham: Springer International Publishing. [Crossref]

[31] Pock, M., Malass'e, O., & Walter, M. (2011). Combining different binary decision diagram techniques for solving models with multiple failure states. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 225*(1), 18-27. [Crossref]

[32] Xing, L., & Dai, Y. S. (2008). A new decision-diagram-based method for efficient analysis on multistate systems. *IEEE Transactions on Dependable and Secure Computing, 6*(3), 161-174. [Crossref]

[33] Ren, Y., Zeng, C., Fan, D., Liu, L., & Feng, Q. (2018). Multi-state reliability assessment method based on the MDD-GO model. *IEEE Access, 6*, 5151-5161. [Crossref]

[34] Ryu, S. M., & Park, D. J. (2005, December). Checkpointing for the reliability of real-time systems with on-line fault detection. In *International Conference*

on Embedded and Ubiquitous Computing (pp. 194-202). Berlin, Heidelberg: Springer Berlin Heidelberg. [Crossref]

[35] Valdez, L. D., Shekhtman, L., La Rocca, C. E., Zhang, X., Buldyrev, S. V., Trunfio, P. A., ... & Havlin, S. (2020). Cascading failures in complex networks. *Journal of Complex Networks, 8*(2), cnaa013. [Crossref]

[36] Morshedlou, H., & Meybodi, M. R. (2014). Decreasing impact of sla violations: a proactive resource allocation approachfor cloud computing environments. *IEEE Transactions on Cloud Computing, 2*(2), 156-167. [Crossref]

[37] Mo, Y., Xing, L., & Dugan, J. B. (2015). Performability analysis of k-to-l-out-of-n computing systems using binary decision diagrams. *IEEE Transactions on Dependable and Secure Computing, 15*(1), 126-137. [Crossref]

[38] Mo, Y., Xing, L., Zhong, F., Pan, Z., & Chen, Z. (2014). Choosing a heuristic and root node for edge ordering in BDD-based network reliability analysis. *Reliability Engineering & System Safety, 131*, 83-93. [Crossref]

[39] Xing, L., & Amari, S. V. (2015). *Binary decision diagrams and extensions for system reliability analysis*. John Wiley & Sons. [Crossref]

[40] Xing, L., & Dugan, J. B. (2002, June). Dependability analysis using multiple-valued decision diagrams. In *Proc. of 6th International Conference on Probabilistic Safety Assessment and Management*.

[41] Shrestha, A., Xing, L., & Dai, Y. (2009). Decision diagram based methods and complexity analysis for multi-state systems. *IEEE Transactions on Reliability, 59*(1), 145-161. [Crossref]

[42] Ammar, M., Hamad, G. B., Ait Mohamed, O., & Savaria, Y. (2017). System-level analysis of the vulnerability of processors exposed to single-event upsets via probabilistic model checking. *IEEE Transactions on Nuclear Science, 64*(9), 2523-2530. [Crossref]

[43] Antonelli, F., Cortellessa, V., Gribaudo, M., Pinciroli, R., Trivedi, K. S., & Trubiani, C. (2020). Analytical modeling of performance indices under epistemic uncertainty applied to cloud computing systems. *Future Generation Computer Systems, 102*, 746-761. [Crossref]

[44] Vasireddy, R., & Trivedi, K. S. (2006). Defining Steady-State Service Level Agreeability using Semi-Markov Process. *DSN 2006*, 172.

[45] Li, K., Tang, X., Veeravalli, B., & Li, K. (2013). Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on computers, 64*(1), 191-204. [Crossref]

[46] Xu, Y., Li, K., He, L., Zhang, L., & Li, K. (2014). A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems. *IEEE Transactions on parallel and distributed systems, 26*(12), 3208-3222. [Crossref]

[47] Mo, Y., & Xing, L. (2013). An enhanced decision diagram-based method for common-cause failure analysis. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 227*(5), 557-566. [Crossref]

[48] Peng, R., Zhai, Q., Xing, L., & Yang, J. (2014). Reliability of demand-based phased-mission systems subject to fault level coverage. *Reliability Engineering & System Safety, 121*, 18-25. [Crossref]

[49] Xing, L. (2007). An efficient binary-decision-diagram-based approach for network reliability and sensitivity analysis. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 38*(1), 105-115. [Crossref]

[50] Xia, R., Yin, X., Lopez, J. A., Machida, F., & Trivedi, K. S. (2013). Performance and availability modeling of ITSystems with data backup and restore. *IEEE Transactions on Dependable and Secure Computing, 11*(4), 375-389. [Crossref]

[51] Gonzalez, A. J., & Helvik, B. E. (2013, August). Hybrid cloud management to comply efficiently with SLA availability guarantees. In *2013 IEEE 12th International Symposium on Network Computing and Applications* (pp. 127-134). IEEE. [Crossref]

[52] Gonzalez, A. J., & Helvik, B. E. (2012, December). System management to comply with SLA availability guarantees in cloud computing. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (pp. 325-332). IEEE. [Crossref]

[53] Zhai, Q., Xing, L., Peng, R., & Yang, J. (2015). Multi-Valued Decision Diagram-Based Reliability Analysis of k-out-of-n Cold Standby Systems Subject to Scheduled Backups. *IEEE Transactions on Reliability, 64*(4), 1310-1324. [Crossref]

[54] Entezari-Maleki, R., Trivedi, K. S., & Movaghar, A. (2014). Performability evaluation of grid environments using stochastic reward nets. *IEEE Transactions on Dependable and Secure Computing, 12*(2), 204-216. [Crossref]

Yuchang Mo received the B.E. (2002), M.S. (2004), and Ph.D. (2008) degrees in Computer Science from Harbin Institute of Technology, Harbin, China. He is currently a Distinguished Professor with the School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power, Hangzhou, China. He was a Visiting Scholar with the Department of Electrical and Computer Engineering, University of Massachusetts (UMass) Dartmouth, USA from September 2012 to August 2013. Prof. Mo was the Program Chair of the 5th Conference on Dependable Computing in China. He is currently a Senior Expert at the Reliability Research Society of China and a Senior Expert at the Technical Committee on Fault Tolerant Computing of China. His current research focuses on the reliability analysis of complex systems and networks using combinatorial models. His research has been supported by the National Science Foundation of China. (Email: yuchangmo@sina.com)

Yuan Fan, CEO of the Hangzhou Anheng Information Technology. His research interests include data security, network security, cloud computing. (Email: yuanfan@sina.com)

Rengui Zhang, CEO of the Zhejiang YuGong Information Technology. His research interests include data science, data mining, cloud computing. (Email: renguizhang@sina.com)

Chunyu Miao, SVP of the Hangzhou Anheng Information Technology. His research interests include data security, network security, security education. (Email: chunyumiao@sina.com)

Jianyong Hu, Professor of the Engineering Research Center of Digital Twin Basin of Zhejiang Province, Hangzhou, China. His research interests include water internet, water conservancy information network, application of industrial internet in water conservancy, artificial intelligence, big data. (Email: hujy@zjweu.edu.cn)

Dr. Mirlan Chynybaev is the Rector of Kyrgyz State Technical University (KSTU) in Bishkek, Kyrgyzstan, a position he has held since December 2020. He earned his Master's degree in Applied Mechanics from KSTU in 2003 and later obtained a Ph.D. in Physics and Mathematics in 2008. Dr. Chynybaev specializes in geomechanics and has contributed to research on service quality in higher education institutions in Kyrgyzstan. (Email: mirlan.chynybaev@kstu.kg)

Jinbin Mu, Vice rector of Zhejiang Institute of Hydraulics and Estuary. His research interests include water internet, water conservancy information network, application of industrial internet in water conservancy, artificial intelligence, big data. (Email: jinbinmu@sina.com)

Faer Gui, CEO of the Zhejiang keepsoft Information Technology. His research interests include data modeling, IoT, digital twin, cloud computing. (Email: faergui@sina.com)

Dr. Akylbek Chymyrov is the Vice Rector for International Relations and the Acting Head of the Department of Geodesy and Geoinformatics at KSTU. He completed his undergraduate studies at Frunze Polytechnic Institute (now KSTU) in 1990, followed by postgraduate studies at Kyrgyz Architecture and Construction Institute. Dr. Chymyrov earned his Ph.D. in Technical Sciences from Kyrgyz State University of Construction, Transport, and Architecture in 2005. His research interests include geodesy, geoinformatics, and environmental risk assessment. (Email: akylbek.chymyrov@kstu.kg)