# Configuration Management Systems

Rohan Murali Nair
Vrije Universiteit Amsterdam, The Netherlands
rohan.murali.nair@student.vu.nl

Matthijs Jansen
Vrije Universiteit Amsterdam, The Netherlands
m.s.jansen@vu.nl

Daniele Bonetta
Vrije Universiteit Amsterdam, The Netherlands
d.bonetta@vu.nl

## ABSTRACT

As modern computing systems become increasingly complex and dynamic, managing performance, scalability, and reliability under diverse workloads has become a significant challenge. Configuration Management Systems (CMS) play a critical role in addressing these issues by tuning the system parameters throughout the software lifecycle. However, the diverse nature of system architectures and workloads makes it difficult to implement generic, one-size-fits-all configuration strategies. To address this, the survey introduces a structured classification of CMS to improve configuration strategies for specific programs and environments. The classification spans three key dimensions: lifecycle stages (predeployment, deployment, runtime), system scope (application-level, infrastructure-level, cross-level), and execution models (offline, online, hybrid), making it easier to understand how different CMS approaches tackle various configuration challenges. The survey also examines the most common techniques, such as telemetry-based monitoring, heuristic and search-based optimization, and machine learning-driven approaches that enable adaptive configuration across diverse environments. In conclusion, it identified pressing research challenges, including multi-cloud interoperability and the development of self-healing CMS. These insights highlight the growing need for intelligent, automated, and cost-effective configuration management in today's heterogeneous computing ecosystems. A well-defined classification becomes especially important, as it allows CMS solutions to target specific configuration scenarios more precisely, based on lifecycle stages, scope, and execution models, ultimately enabling more adaptive context-aware optimization strategies.

## KEYWORDS

Configuration Management Systems, Lifecycle, Taxonomy, Execution, Techniques

## 1 INTRODUCTION

Configuration Management Systems (CMS) have played an important role in computing since the 1950s, when early systems required a consistent methods for software versioning and hardware provisioning[13]. Over time, as computing environments evolved, ranging from on-premise servers to today's cloud native services, the importance of configuration management has only been increased. In modern systems, misconfigurations are one of the most seen causes of outages, poor performance, and and leading to security vulnerabilities. Modern computing systems have large number of configurable parameters that can be adjusted, these parameters includes virtual machine instance types, container CPU and memory resource limits, thread pool sizes in multi-threaded applications, etc. These parameters have an impact on performance, scalability, and operational cost. But in real-world scenarios, these parameter tuning are often done manually, left with default values, or require someone with a lot of domain knowledge. As systems become more complex and dynamic, these manual or static approaches often lead to poor performance, wasted resources, or unrealiable systems [30][3].

Achieving optimal configuration parameters is most essential, but what does this mean in this domain and why is it so important? An optimal configuration parameters is a specific combination of configuration values that enables a system to meet one or more defined objectives, such as minimizing execution time, maximizing throughput, or reducing cost, while operating within given resource and workload constraints [31]. This is important because without proper tuning, it can lead to under-utilization of resources, performance degradation, or high operational cost. However, identifying these optimal values for the parameters is very challenging because of the large configuration space, the interdependencies between parameters, and the unpredictable nature of changing workloads.

The consequences of misconfiguration extend beyond theory, having caused notable disruptions in real-world scenarios. In 2017, *Amazon Web Services (AWS)* suffered a massive outage in its S3 storage service due to a misconfigured input in an internal maintenance tool [4]. An engineer was required to remove a small number of servers from one of the S3 subsystems for debugging, but incorrectly entered a parameter value that removed a much larger set of servers, including those handling metadata, thus causing service disruption across the US-East-1 region.

Similarly, in 2012, *Knight Capital Group* deployed new trading software with a legacy feature flag (a configuration parameter) accidentally reactivated on several servers [28]. This led to the execution of an outdated code which is responsible for an old order routing system, which quickly executed a series of unintended trades, resulting in a $440 million loss in 45 minutes. These incidents shows how failures in configuration management, whether due to unsafe tooling defaults or parameter misuse, can lead to huge system-wide issues. The examples demonstrate the critical need for robust, intelligent and context-sensitive configuration management solutions that can mitigate such risks.

Modern computing environments are complex and dynamic, requiring configuration management systems (CMS) that can operate effectively across a variety of use cases. Although CMS have become more adaptive and intelligent, it remains unrealistic to expect a single system to handle every configuration challenge. Different systems are designed for different phases of operation, target various layers of the stack, rely on different execution models of configurations, and adopt a range of underlying techniques. These differences make it hard to compare systems and see common patterns in CMS design. To better understand this diverse landscape, this paper introduces a structured perspective that captures the essential variations in how CMS are built and used. The following are the main contributions of this paper:

(1) **Lifecycle and Scope Classification**: The study presents a structured classification of CMS across lifecycle stages (pre-deployment, deployment, runtime) and system scope (application level, infrastructure level, cross level), enabling targeted and layer-aware configuration strategies. This helps in clarifying how CMS differ depending on when and where configurations are applied. By structuring the CMS along these dimensions, the user can better match tools and techniques to specific system needs.
(2) **Execution Model Classification**: A separate classification based on execution models such as offline, online, and hybrid is introduced to capture how configuration changes are applied over time. This classification is essential for understanding trade-offs between stability, adaptability, and performance. It complements the lifecycle classification by focusing on how changes are made, rather than when or where, highlighting operational behavior under varying workload conditions.
(3) **Survey of Key Techniques**: The study identifies the most common and dominant techniques used in CMS, such as telemetry-based monitoring, heuristic and search methods, and machine learning. These methods enable automated and adaptive configuration across different environments. Understanding these techniques allows to choose appropriate mechanisms based on performance needs, scalability goals, and available data.
(4) **Future Research Directions**: The study also highlights open challenges, including the lack of multi-cloud interoperability and the need for self-healing CMS that can automatically detect and recover from misconfigurations. These gaps present significant barriers to the deployment of CMS in large-scale real-world systems. Future work should focus on building more robust, autonomous and portable configuration solutions.

By addressing these aspects, this study provides valuable information for researchers and practitioners, contributing to the development of more adaptive, scalable, and intelligent configuration management solutions.

## 2 RELATED WORK

Configuration Management Systems (CMS) have been studied from multiple angles, including historical overviews, tool-specific evaluations (e.g., Ansible, Terraform, Puppet), and process of models. These works have contributed to understanding the evolution of CMS, best practices, and implementation challenges. However, most existing works focus narrowly on specific tools and fail to examine CMS as a unified concept. In many cases, the focus is limited to specific tools or domains, making it difficult to generalize the findings across broader CMS applications.

A key limitation across the literature is the lack of systematic classification.Most existing studies look at specific areas, like deployment or maturity, without considering how CMS work across different stages, layers, and methods of execution and optimization. These dimensions are essential in understanding how CMS function in various environments. Without such a classification, it becomes difficult to compare approaches or design systems that are adaptable to varying configuration scenarios.

| **Survey** | **Lifecycle** | **Scope** | **Execution** | **Techniques** |
|---|---|---|---|---|
| Estublier (2000) [10] | ✗ | ✗ | ✗ | ✗ |
| Delaet et al. (2010) [7] | ✓ | ✓ | ✓ | ✗ |
| Ali & Kidd (2013) [2] | ✗ | ✗ | ✓ | ✗ |
| Hintsch et al. (2016) [16] | ✗ | ✗ | ✗ | ✗ |
| Serrano & Pereira (2020) [24] | ✗ | ✗ | ✗ | ✗ |
| Sherman et al. (2020) [25] | ✓ | ✗ | ✓ | ✗ |
| Kostromin (2020) [18] | ✗ | ✓ | ✓ | ✗ |
| Farayola et al. (2023) [14] | ✗ | ✗ | ✗ | ✓ |
| **This survey** | ✓ | ✓ | ✓ | ✓ |

**Table 1: Related works in the domain, ordered by year.**

For example, Estublier (2000) [10] provides a historical perspective on Software Configuration Management (SCM), focusing on early version control systems and process control techniques, but does not address modern cloud-native practices or adaptive CMS. Similarly, Serrano and Pereira (2020) [24] examine maturity models in CMS but focus primarily on process evaluation, without discussing how CMS are applied or executed across different system layers. Hintsch et al. (2016) [16] and Farayola et al. (2023) [14] analyze popular tools and discuss challenges in distributed environments, yet they lack a structured classification of how these tools operate across different stages or execution models. While these studies contribute important insights, they do not provide a unified framework for understanding CMS in a broader, system-wide context, which is necessary for addressing today's dynamic and heterogeneous computing environments.

Table 1 summarizes how recent surveys compare across these dimensions. While each study provides value in its own domain, they do so inconsistently and often overlook key relationships between CMS characteristics. This paper addresses that gap by offering a high-level classification that links lifecycle stages, execution models, and optimization strategies, providing a broader and more structured understanding of CMS.

## 3 TAXONOMY OF THE LITERATURE STUDY

Configuration Management Systems (CMS) differ widely in how they handle misconfigurations, adjust to changing workloads, and balance goals like performance and cost. To better compare these solutions, a taxonomy is introduced.This taxonomy classifies CMS based on the most critical aspects and highlights the dominant techniques, providing a structured way to understand and evaluate different approaches.

This taxonomy has four main dimensions, each containing subpoints:

(1) **Lifecycle**: This concerns with "when configuration changes occur". Pre-deployment sets parameters before the system

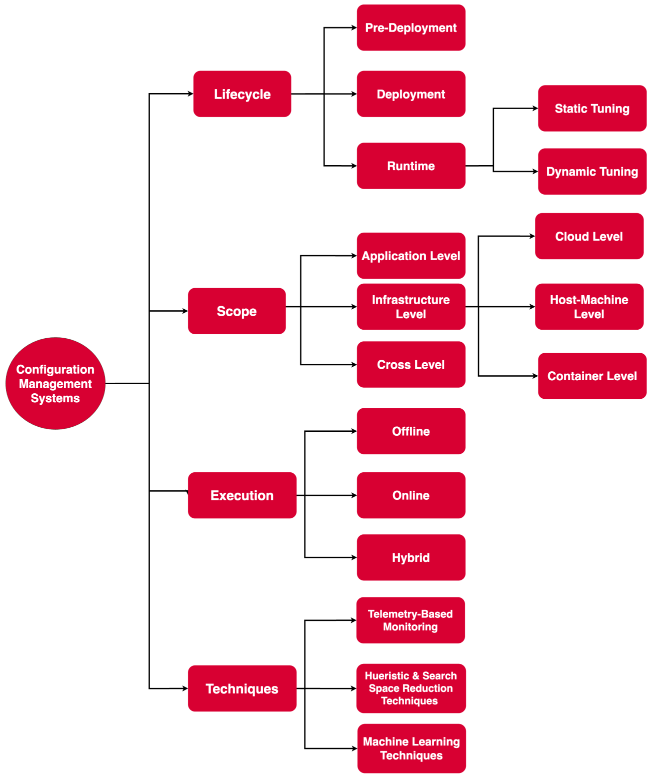

**Figure 1: Concluded taxonomy of Configuration Management Systems**

is deployed, deployment refines them in the deployment phase, and runtime fine-tunes settings continuously once the system is running.

(2) **Scope**: This focuses on where configuration applies in the stack. The application level targets application-specific parameters, the infrastructure level covers domains such as VMs, containers, and the cross-level coordinates both layers for end-to-end optimization.

(3) **Execution**: This dimension refers to when and how configuration changes are applied. Offline execution applies configurations before runtime, often based on historical data. Online execution adjusts configurations dynamically during system operation using real-time feedback. Hybrid execution combines both, starting with offline-tuned defaults and refining them at runtime through adaptive techniques.

(4) **Techniques**: This refers to the dominant strategies used in modern CMS to optimize configurations. The most widely adopted methods include "telemetry-based monitoring" which provides real-time feedback for continuous tuning, "heuristic and search-based techniques" which efficiently explore large spaces configuration, and "machine learning approaches" which have become increasingly dominant due to their ability to predict and adapt configurations automatically based on data-driven insights.

By organizing CMS solutions across these four dimensions,we gain a clearer understanding of how to target specific system phases, layers, execution methods, and optimization strategies. This structure makes it easier to identify potential overlaps, uncover research gaps, and guide practitioners in selecting approaches that best fit their needs, whether they require offline stability, online adaptability, or an intelligent combination of both.

## 4 STUDY DESIGN

The main goal of this literature study is to collect, organize and present information from different sources to better understand Configuration Management Systems (CMS). As shown in Figure 2, the study is designed in three main phases: Planning, Execution and Extraction. Each phase helps refine the search and ensure that only the most relevant information is gathered.

(1) **Planning:**
- *Objectives:* The main objectives were to understand how Configuration Management Systems (CMS) work across different lifecycle stages, their scope, identifying the dominant techniques and execution models they employ, and explore existing research gaps in the field.
- *Topics:* Based on these objectives, key topics were selected, including lifecycle stages (pre-deployment, deployment, runtime), system scope (application level, infrastructure level, cross level), execution models (offline, online, hybrid) and optimization techniques (telemetry, heuristics, machine learning).
- *Research Questions:* With the objectives in mind, the next task was to formulate research questions that would help guide the entire review. These questions included, for example: 'What are the primary dimensions along which CMS can be classified?" or "How do different execution models (offline, online, hybrid) affect CMS performance?".
- *Keywords:* To ensure a complete search, a set of keywords was identified based on the objectives and research questions. Examples of keywords included "Configuration Management Systems", "Dynamic Tuning in Configuration Management Systems", "Machine Learning in CMS", etc. These keywords were used to query papers from academic databases such as IEEE Xplore and Google Scholar.

(2) **Execution**
- *Selection*: Using keywords that were found, the next step was to select a broad set of papers which are relevant to the domain. Initial inclusion criteria typically required papers to focus on types of cms, system tuning, or automation in the context of CMS.
- *Filtering:* After collecting an initial set of papers, an inclusion-exclusion process was used to narrow down the selection. In the inclusion process, abstracts were read to include papers that discussed CMS more generally. In the exclusion papers, they were excluded if they focused only on a single tool without broader insights or did not explain their methods clearly. Conclusions were also scanned to verify the relevance and quality of each paper.
- *Citations*: To avoid missing influential research, the references of selected papers were also checked. If any previously unknown but relevant publications emerged from this process, they were added to the pool. This step helps

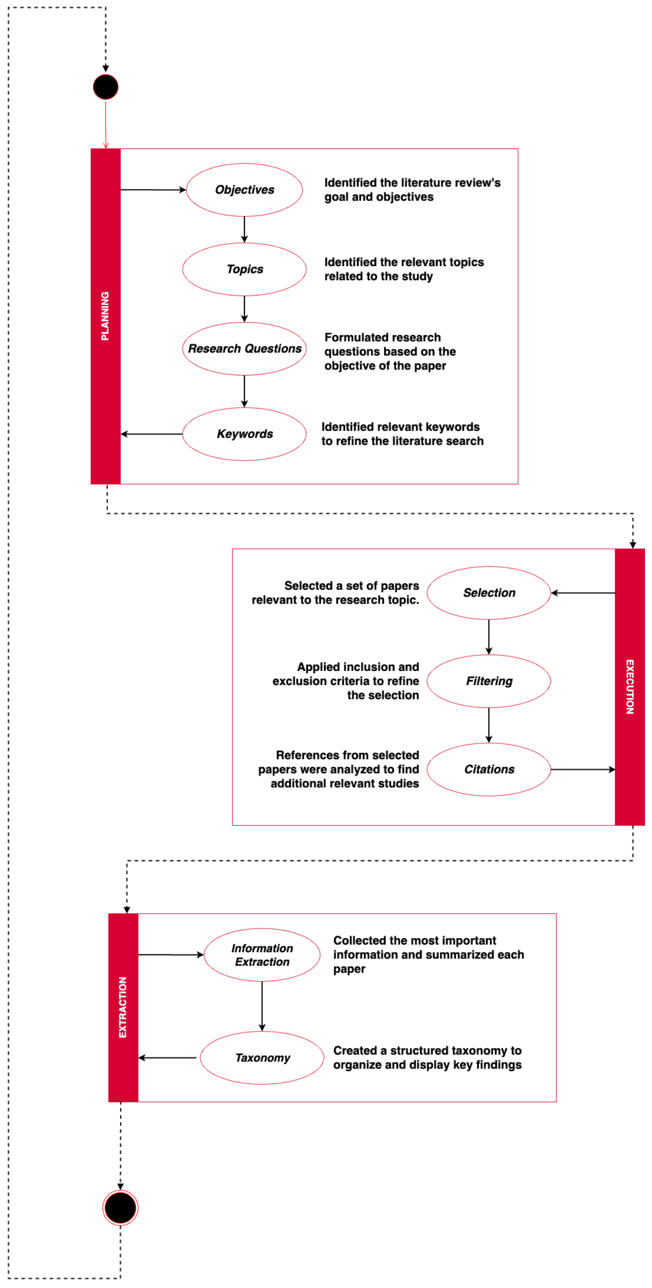
PLANNING
Objectives
Identified the literature review's goal and objectives
Topics
Identified the relevant topics related to the study
Research Questions
Formulated research questions based on the objective of the paper
Keywords
Identified relevant keywords to refine the literature search
EXECUTION
Selection
Selected a set of papers relevant to the research topic.
Filtering
Applied inclusion and exclusion criteria to refine the selection
Citations
References from selected papers were analyzed to find additional relevant studies
EXTRACTION
Information Extraction
Collected the most important information and summarized each paper
Taxonomy
Created a structured taxonomy to organize and display key findings

**Figure 2: Study Design**

to snowball the most impactful and frequently cited works, ensuring that the overall review remains well-rounded and up to date.

(3) **Extraction**
- *Information Extraction*: In this phase, papers that passed the filtering stage was read in detail. Key findings, such as system goals, performance metrics, methodology and results, were extracted and summarized. The information was organized according to the main areas of interest: lifecycle stages, scope, execution models, and techniques.
- *Taxonomy*: Once the essential data from all included studies was collected, it was organized into a structured taxonomy. This taxonomy (elaborated in a later section) groups CMS approaches along four main dimensions: Lifecycle, Scope, Execution, and Techniques, giving a clear framework for comparing different solutions.

Through these phases, the study followed a clear path, from setting objectives and defining search queries, to narrowing down to the relevant literature set and finally organizing the results into a taxonomy that classifies different types of Configuration Management Systems (CMS) and highlights research gaps. Figure 2 visually shows this process, highlighting how each phase builds on the previous one and leads to a better understanding of how configuration management systems can be organized and explored.

## 5 LIFECYLCE STAGES

The configuration parameters have a significant impact on the performance of how an application or computing environment performs. A configuration parameter that works in one environment may not be optimal in a different environment or under different workloads. To address this, different approaches have been introduced in various works, focusing on different phases of configuration management. Some works optimize configurations before deployment to avoid misconfigurations, others optimize parameters at deployment to balance cost and performance, while others focus on runtime tuning to adapt to changing workloads. This section examines how configuration management is addressed in different phases and how existing research contributes to optimizing system performance over time.

### 5.1 Pre-Deployment Configuration

The pre-deployment phase is the foundation of configuration management, where the initial parameters are defined before deployment. These parameters are based on expected workloads and system specifications. Earlier approaches primarily focused on pre-deployment configuration. The main goal is to prevent misconfigurations and tune parameters to meet system requirements. This helps improve performance, reliability, and cost efficiency by testing parameters in advance.

Even though pre-deployment configuration is the safest and most widely used approach, it comes with its own challenges. The default configuration parameters set during the pre-deployment phase are often sub-optimal. Many systems are deployed with default configurations, which become ineffective as the workload changes. This is mainly because workloads cannot be accurately predicted in advance. They are dynamic and may shift significantly after deployment, making pre-deployment tuning difficult. Manual parameter tuning is time-consuming and makes it difficult to identify optimal settings. Additionally, configuration parameters must be tested, which adds overhead.

Several research works have explored techniques to enhance pre-deployment configuration management by optimizing parameter selection and preventing misconfigurations. "*ConfProf*" employs a white-box performance profiling approach, analyzing and ranking configuration options based on execution profiles, helping developers pinpoint inefficient configuration-dependent code and improve system efficiency [15]. Similarly, "*ATConf*" utilizes Bayesian Optimization with dropout-based dimensionality reduction to optimize big data processing framework (BPDF) parameters before execution, significantly reducing execution time by up to 46.52% compared to default configurations [8]. Another example, *FLASH*, utilizes a sequential model-based approach to identify the near-optimal configurations while using far lesser measurements than traditional methods [23]. By reflecting on the measurements seen so far, FLASH dynamically refines its configuration search, making it well-suited for pre-deployment tuning. Finally another study introduced "*CherryPick*", applies Bayesian Optimization with a guided trial-and-error method, launching targeted trial jobs on selected cloud configurations before execution [3]. By iteratively measuring performance and refining its model, CherryPick identifies near-optimal configurations with minimal testing, ensuring efficient offline tuning without impacting mission-critical workloads. These methods demonstrate how machine learning and automated tuning strategies improve pre-deployment configuration, making systems more reliable before they go live.

However, despite these advancements, pre-deployment tuning alone is not sufficient to handle real-world variability. Default configurations, even if optimized before deployment, may become sub-optimal when exposed to dynamic workloads. Many systems rely on pre-defined configurations that are tested in controlled environments, but fail to maintain efficiency under changing operational conditions. Additionally, manual tuning remains a significant challenge, as identifying the best configuration parameters requires extensive domain expertise and incurs additional computational overhead. While pre-deployment techniques reduce initial errors and enhance system stability, they cannot fully account for workload fluctuations, infrastructure variations, and unexpected performance bottlenecks that may arise after deployment. This limitation necessitates the need for deployment and runtime adjustments, which are explored in the following sections.

**Observation 5.1**

Pre-deployment configuration is used for optimizing system parameters before deployment to avoid misconfigurations and improve the performance. However, these configurations often become suboptimal as the workloads changes, thus requiring additional tuning during deployment or runtime for longer efficiency.

## 5.2 Deployment Configuration

The deployment phase configuration involves setting and optimizing parameters as the system is deployed into a real-world environment. Unlike the pre-deployment phase, where tuning is based on expected workloads, the deployment phase adjusts parameters based on actual system conditions. This phase is critical as the mismatches between the pre-defined configurations on real-world infrastructure can lead to performance bottlenecks or unexpected failures. The main goal is to balance performance and cost, avoid performance bottlenecks, and find the optimal configuration parameters for real-world environment.

Despite careful pre-deployment tuning, deployment-time configuration faces several challenges due to the unpredictable nature of real-world environments. The configuration parameters optimized before the deployment may not have the best performance when resource availability, latency, and network conditions are drastically different from the test environment. For example, cloud environments introduce variable resource constraints, thus making static configurations unreliable. Poor deployment configurations can increase execution time and cost, particularly in large data and cloud environments [8]. Selecting incorrect VM instance types, storage configurations, or database settings can lead to wasted resources [3]. Another challenge is manually optimizing the deployment configuration parameters, which requires a lot of knowledge about the system [21], and the vast number of configuration parameters makes it more difficult to scale.

Several key studies have explored deployment-phase configuration techniques to enhance performance and resource efficiency. "*EDLB*" (Enhanced Dynamic Load Balancing) optimizes cloud resource allocation by dynamically distributing workloads across virtual machines during deployment, achieving a 22.46% reduction in execution time and improved resource utilization [32]. "*SOAVM*" employs an adaptive resource allocation approach by leveraging service-oriented virtualization models, which adjust virtualized application resources dynamically based on real-time system conditions, ensuring performance stability even in fluctuating workloads [5]. Another research also introduces "*Apollo*", a predictive configuration deployment framework that estimates the impact of changes before rollout, while ensuring a safer deployment and reduced likelihood of runtime misconfigurations [26]. These frameworks highlight how intelligent deployment-phase tuning strategies can ensure optimal configurations, reducing the need for post-deployment adjustments while improving scalability and operational stability.

These works highlight the importance of the configuration in the deployment phase. They demonstrate how automated data-driven techniques can refine configurations based on real-world conditions, improving efficiency, reducing costs, and improving system stability from the moment of deployment.

**Observation 5.2**

Deployment phase configuration adjusts the system's parameters during the deployment to match the real-world conditions, ensuring better performance and resource efficiency. Unlike pre-deployment tuning, it adjusts settings based on actual infrastructure constraints, network conditions, and resource availability, preventing performance bottlenecks, inefficiencies, and misconfigurations that may arise during deployment.

## 5.3 Runtime Configuration

The runtime phase is the dynamic optimization of the configuration parameters while the system is running to maintain performance. Unlike the pre-ceployment phase (5.1) and the deployment phase (5.2), which focus on tuning before deployment or during deployment, the runtime configuration focuses on optimizing the configuration parameters based on the real-time workloads of the system. This phase is crucial for ensuring that the application's scalability, efficiency, and cost are balanced in dynamic computing environments such as cloud environments (AWS, Azure, GCP, etc.). The runtime configuration can be implemented in two ways: Static Tuning and Dynamic Tuning.

(1) **Static Tuning**: Static Tuning requires human involvement to adjust the configurations based on workloads. System administrators or engineers monitor performance metrics and manually modify parameters when required. This approach is effective in scenarios that require manual adjustments. It is used in critical applications where automated adjustments could be risky. Additionally, manual tuning is applied to legacy systems that lack support for real-time adaptive tuning. There are many limitations to this approach, such as slow response time, high operational cost, and susceptibility to human errors.
(2) **Dynamic Tuning**: Dynamic Tuning is an approach for making real-time changes to configuration parameters based on workloads, system performance, and external conditions. It is widely used in cloud computing, databases, and auto-scaling environments. Dynamic tuning can use various techniques, including machine learning, control theory methods, and deep learning. The advantage of dynamic tuning is that it ensures consistency in performance despite changing workloads. It also helps minimize resource wastage through trial and error and reduces downtime and operational costs by continuously optimizing system workloads.

While runtime configuration enables systems to adapt dynamically to workload changes, it also introduces several challenges that must be carefully managed. Constant monitoring is required for real-time reconfiguration, which leads to system overhead, and frequent adjustments to configuration parameters can cause instability and high resource consumption [22]. Although dynamic tuning provides flexibility, excessive changes can negatively impact performance, requiring a balance between rapid reconfiguration and its consequences [22]. Another challenge is the trade-off between performance and cost, as dynamic tuning can optimize both, but poorly timed adjustments may lead to inefficiency. For instance, OptimusCloud employs a cost-benefit analysis approach to ensure that reconfigurations justify their computational expense, optimizing performance-per-dollar (Perf/$) in cloud-based NoSQL databases [20]. Several research works have explored runtime configuration, especially in dynamic tuning. Although static methods have been widely used, some systems still require manual tuning during operation. "*OptimusCloud*" achieves 4.5× lower latency by jointly tuning database and VM configurations at runtime, ensuring efficient adaptation to workload variations[20]. Similarly, "*SmartConf*" introduces a control-theoretic approach to dynamically fine-tune CPU, memory, and I/O allocations, ensuring system stability while responding to changing workloads[29]. Another study, "*DeepHill*" integrates deep learning-based workload prediction with real-time instance reconfiguration, reducing power consumption by 13.33% while optimizing resource utilization[1]. These studies highlight the importance of runtime configuration in modern computing environments, where workloads are unpredictable and require continuous optimization.

**Observation 5.3**

Runtime phase configuration optimizes the system parameters dynamically while the system is running, while ensuring scalability, efficiency, and cost-effectiveness in cloud and dynamic environments. It can be static, requiring manual adjustments by system administrators, or dynamic, leveraging techniques like machine learning, control theory approaches, and deep learning for real-time optimization.

## 6 SCOPE OF CONFIGURATION

The configuration management plays an important role in ensuring the stability, performance, scalability and saving cost of the systems. There are various systems, applications and environments have distinct configuration parameters which require optimization in order to get better performance. The scope of configuration can be organized into various levels depending on where the tuning is implemented:

(1) **Application-Level Configuration** – Focuses on optimizing the configuration parameters at application level.
(2) **Infrastructure-Level Configuration** – Managing system-wide resources such as containers, virtual machines (VM) and cloud infrastructure. This level can further classified into:
   (a) *Container-Level Configuration* – Optimizing configurations within containerized environments to ensure isolation, scalability and efficiency.
   (b) *Host-Level Configuration* – Managing system hardware, networking, and virtualization settings.
   (c) *Cloud-Level Configuration* – Tuning cloud-based services such as provisioning, scaling, and resource optimization.
(3) **Cross level Configuration** – A hybrid approach integrating both application and infrastructure-level tuning for system-wide optimization.

Each configuration level plays a important role in system management, and their combined effectiveness shows the overall performance, adaptability and reliability of the modern computing environments.

### 6.1 Application Level Configuration

Application-level configuration is the most commonly used aspect of configuration management, as it directly affects application performance, effective execution, and resource optimization. This level of configuration is about optimizing application's configuration parameters such as memory allocation, concurrency control, caching mechanisms, and query optimization to ensure the application is running efficiently under workloads. Given its impact on system behavior, application-level configuration is widely studied, with various optimization techniques proposed to automate and enhance this process.

One key area where application-level configuration has an important role is in database management systems. Databases require carefully tuned parameters such as buffer pool sizes, indexing strategies, transaction isolation levels, and query execution plans to achieve optimal performance. As mentioned in the previous section 5, configuration management can also be done in pre-deployment, deployment, and runtime phases. Studies such as *iTuned* have developed automated methods leveraging adaptive sampling to dynamically improve database configurations [9]. Similarly, web applications depend on application-level configuration management to optimize request handling, session management, etc. Poorly configured parameters in applications can lead to longer response times, inefficient resource allocation, and potential service failures.

Frameworks for Big Data, including Hadoop, Spark, and Flink, rely on application-level configurations to optimize job scheduling, manage memory efficiently and adjust parallelism levels. The complexity of configuring these frameworks has led to optimization techniques such as *ATConf*, which applies Bayesian Optimization to improve execution performance by pre-tuning processing parameters [8]. These techniques highlight the importance of application configuration management, as workload-specific optimization can enhance efficiency and cost-effectiveness.

Since configuration at the application level plays a crucial role in system performance, tackling challenges with automated, workload-aware tuning methods guarantees efficiency and reliability across different environments.

**Observation 6.1**

Application-level configuration directly influences the system performance, by optimizing application parameters such as memory allocation, concurrency control, caching mechanisms, load balancing etc. Proper tuning of these parameters improves performance, resource utilization, and responsiveness, while ensuring stability and high performance. Application-level configuration is necessary for ensuring reliability, improving responsiveness and optimizing resource utilization, as it directly impacts the system performance.

## 6.2 Infrastructure Level

Infrastructure-level configuration is fundamental in optimizing resource allocation, ensuring system scalability, and saving cost in computing environments. Unlike application-level configuration, which optimizes application parameters, infrastructure-level configuration is used for provisioning and managing hardware resources, virtual machines (VMs), networking policies, etc. Optimizing parameters at this level helps avoid resource bottlenecks, reduce latency, and minimize operational overhead, thus directly influencing system performance and reliability. Infrastructure-level configuration is divided into three key dimensions: **Container-level**, **Host-level**, and **Cloud-level**.

*6.2.1 Container Level Configuration.* Container-level configuration is used for optimizing parameters such as CPU, memory, and storage within container environments. The adoption of technologies such as Docker and Kubernetes serves as an example of how containers are prioritized in modern computing because of characteristics such as being lightweight, easily scalable, and deployable. However, as mentioned earlier, to obtain optimal container performance, it is necessary to effectively manage resources, especially under varying and dynamic workloads. Without proper optimization of configuration parameters, there can be performance degradation, inefficient resource utilization, security vulnerabilities, and scaling issues, leading to high operational costs and reduced application reliability. Challenges such as resource contention, inefficient auto-scaling, and complex networking can impact container performance and scalability. Security misconfigurations, persistent storage issues, and configuration drift further complicate deployment consistency and container reliability.

One notable piece of research that focuses on managing container configurations is *ConfAdvisor* [11], which uses a white-box approach to automate detection and correction of misconfigurations by analyzing real-time telemetry data, container metrics, and deployment logs. This continuous analysis enables adaptation to various workloads, significantly improving performance, reliability, and resource utilization within Kubernetes-managed environments [11].

Another significant contribution is *ATConf*, an automated configuration framework designed for big data processing frameworks. It employs a black-box optimization technique, dynamically tuning both internal parameters (such as Spark's execution parameters) and external parameters (such as JVM settings and kernel-level configurations). This approach effectively addresses the complexity of high-dimensional configuration spaces, adapting resource allocation to fluctuating workloads to achieve enhanced performance compared to default settings [8].

More recent research, *conTuner*, auto-tunes the resource parameters of Docker containers using a combination of offline profiling and online adaptation [6]. By learning from workload patterns and adjusting configurations in real time, it improves resource utilization and container performance in dynamic environments. It uses clustering and a two-phase random search to efficiently explore configuration spaces, helping to avoid manual tuning while ensuring better scalability and deployment efficiency.

These studies underline the importance and potential of advanced configuration management systems in optimizing container-level performance, reliability, and resource efficiency within modern, dynamic infrastructure environments.

*6.2.2 Host-Level Configuration.* Host-level configuration involves managing and tuning physical hardware resources, virtualization layers (such as hypervisors), and host operating systems. Proper optimization at this level is important, as it forms the foundation that directly impacts the performance, efficiency, and reliability of higher-level services such as containers and virtual machines.

A key challenge at this level is the overhead introduced by hypervisor resource management tasks, which can degrade performance if not properly managed. Research on hypervisor systems like *VMware ESXi* shows the critical importance of properly setting CPU resource allocations, memory management strategies, and scheduling policies. Effective optimization of these configurations reduces hypervisor overhead and optimizes resource usage, significantly improving overall system scalability and performance [17].

In addition to addressing host-level configuration challenges, frameworks such as *iTuned* introduce solutions for automated performance optimization, particularly focusing on database systems running directly on physical or virtualized hosts. iTuned uses an adaptive sampling strategy and a Gaussian process representation of response surfaces to dynamically identify and set optimal configurations for critical database parameters, such as buffer pool size, concurrency levels, and query optimization settings. By systematically conducting low-overhead experiments, iTuned efficiently identifies configurations that significantly improve performance metrics over default or manually tuned settings, reducing complexity and enhancing operational efficiency at the infrastructure level [9].

These approaches show the importance of configuration management at the host level. By effectively optimizing hypervisor and database system parameters, systems can significantly improve resource utilization, reduce operational costs, and ensure robust performance and reliability for host-level environments.

*6.2.3 Cloud-Level Configuration.* Cloud-level configuration involves strategic optimization of computing resources, storage infrastructure, network settings, and security policies in cloud environments such as AWS, Azure, and Google Cloud. Effective management at this level is critical due to the dynamic and heterogeneous nature of workloads and resource configurations, directly impacting operational costs, performance, and reliability of cloud-hosted systems.

Many studies have been conducted on cloud-level configuration. One of them is *Deep-Hill*, a framework that addresses cloud-level configuration challenges by optimizing resource allocation for SaaS applications in dynamic environments [1]. Unlike traditional methods, Deep-Hill uses a five-layer deep neural network (DNN) to predict optimal parameters for cloud servers based on real-time user demand, and applies a Hill-Climbing (HC) algorithm to fine-tune these parameters step by step, ensuring optimal performance under varying workloads. This combined approach helps minimize power consumption, reduce the number of active servers, and prevent underutilized resources by adapting to workload changes. As a result, Deep-Hill improves system scalability, reduces operational costs, and enhances overall cloud performance, making it a key advancement in cloud-level configuration management.

Additionally, recent studies highlight the importance of efficiently distributing workloads at this level. The *Enhanced Dynamic Load Balancing (EDLB)* algorithm proactively assigns tasks to virtual machines by constantly evaluating current system states and Service Level Agreement (SLA) requirements [12]. Unlike traditional approaches, EDLB continuously adjusts task allocations to avoid resource bottlenecks and minimize SLA violations, achieving significant improvements in overall performance metrics such as execution time and system responsiveness. This approach greatly improves resource utilization and operational efficiency, emphasizing the crucial role of adaptive load balancing in maintaining optimal performance and ensuring compliance with service quality standards in cloud environments [12].

Cloud-level configuration plays a crucial role in ensuring scalability, cost efficiency, and optimization for various workloads in modern cloud infrastructures. Proper optimization enables dynamic resource allocation, effective load balancing, and strong security, helping prevent performance issues and reduce costs. Frameworks such as Deep-Hill and EDLB demonstrate the importance of adaptive and predictive configuration strategies for managing cloud resources efficiently.

**Observation 6.2**

Infrastructure configuration optimizes configuration parameters at infrastructure level and they are categorized across three key dimensions: container-level, host-machine and cloud-level configurations. Container-level focuses on CPU, memory, and storage efficiency within the container environment or the containers itself, Host-level tuning enhances hypervisor and database performance, and cloud-level ensures dynamic workload distribution and auto-scaling. Proper infrastructure configuration is crucial for system reliability, cost efficiency, and sustained performance.

## 6.3 Cross Level Configuration

Cross-level configuration is the bridge that connects application-level and infrastructure-level tuning to provide an effective approach for optimizing configuration parameters at each layer. In an isolated environment, cross-level approaches coordinate settings across the entire software stack. The main goal of this approach is to ensure that adjustments at one level do not conflict with configurations at another level, thereby improving overall efficiency, reliability, and cost-effectiveness.

A main advantage of cross-level configuration is the ability to manage complex interactions among various parameters. For example, changes to an application's concurrency settings can affect the CPU scheduling of the underlying container, or vice versa. By using monitoring techniques, cross-level approaches reduce the risk of misalignment between applications and infrastructure resources.

Many studies have been conducted in this domain. *OptimusCloud* tunes both virtual machine and NoSQL database parameters by adjusting configurations across the entire stack according to changing workloads. This study has shown that coordination of application- and infrastructure-level settings can significantly improve performance [20]. Another study introduces *OPPerTune*, which explores post-deployment tuning of large-scale services by dynamically reconfiguring both application-level parameters (e.g., cache limits, data path optimizations) and infrastructure-level parameters (e.g., CPU isolation, container scaling) to perform well under evolving workloads [27]. By aligning changes across all layers, the system ensures that the specific needs of each workload at the application level are matched with appropriate container-level configuration parameters.

Despite its advantages, cross-level configuration adds complexity. This level of configuration demands a unified view of the entire system, which can be challenging when dealing with different types of environments. Moreover, aligning priorities across layers requires well-designed conflict resolution policies to prevent compromises in system stability. As systems become more complex, cross-layer configuration becomes increasingly important for effective infrastructure management. It ensures that applications benefit from globally optimized settings rather than limited, layer-specific adjustments.

**Observation 6.3**

Cross-level configuration bridges the gap between application-level and infrastructure-level tuning by coordinating adjustments across all layers of the software stack. This approach helps prevent conflicts between configurations at different levels, reducing misalignment and improving performance, efficiency, and cost-effectiveness. By aligning configuration priorities across components such as databases, virtual machines, and containers, cross-level strategies enable globally optimized settings that adapt better to complex dynamic workloads.

## 7 TYPES OF EXECUTION

There are different ways in which the execution of configuration management can implemented, these are categorized on how configuration settings are applied across various environments. These execution strategies are typically classified into three types: **Offline execution**, **Online execution**, and **Hybrid execution**, each used for specific system needs, dynamic workloads, and operational conditions.

### 7.1 Offline Execution

Offline execution in Configuration Management Systems (CMS) refers to the process of determining and optimizing system parameters before execution or deployment. In this approach, parameters such as historical workload data, performance logs, etc., are considered, and predefined optimization strategies are used to establish configuration values before the system runs. This type of execution ensures that optimal configurations are set up prior to deployment. Offline execution is especially useful in environments that prioritize stability, predictability, and minimal runtime overhead. It is crucial for systems with predictable workload patterns, such as batch processing, enterprise applications, and high-performance computing (HPC). By predefining configurations, this approach minimizes runtime delays, maintains consistent performance, and prevents configuration drift. It is particularly beneficial in low-latency environments, where real-time tuning could introduce unnecessary computational overhead. Additionally, offline execution reduces the risk of ad-hoc misconfigurations, ensuring system reliability and efficiency.

Several studies have explored offline execution to enhance configuration management by optimizing settings before deployment. "*ATConf*" applies Bayesian optimization to analyze historical workload data, using dropout-based dimensionality reduction to identify critical configuration parameters, minimizing execution time and improving predictability[8]. "*ConfProf*" profiles system configurations before runtime, leveraging machine learning-based performance modeling to predefine optimal resource allocation, reducing performance variability and improving efficiency where real-time tuning is infeasible [15]. "*CherryPick*" employs a guided trial-and-error approach with Bayesian Optimization, running targeted trial jobs on different resource configurations before execution. By iteratively measuring performance and refining its model, CherryPick finds near-optimal configurations with minimal testing, ensuring efficient offline tuning without impacting mission-critical workloads [11]. These studies illustrate how offline execution enhances stability, predictability, and efficiency in structured workload environments.

The primary challenge of offline execution is its inability to adapt to real-time workload variations, making it unsuitable for dynamic environments. Predefined configurations can lead to resource mismatches, either through over-provisioning or under-provisioning. Additionally, its effectiveness relies on the accuracy of historical data and outdated workload profiling, which may result in suboptimal configurations. Despite these limitations, offline execution remains beneficial for predictable workloads, where pre-tuned configurations reduce runtime overhead and improve system stability.

**Observation 7.1**

Offline execution optimizes system parameters before deployment, ensuring stability, predictability, and minimal runtime overhead. It is particularly effective for batch processing, enterprise applications, and HPC environments, where predefined configurations reduce runtime delays and misconfigurations. However, offline execution cannot adapt to real-time workload changes, making it less suitable for dynamic environments that require continuous tuning and flexibility.

### 7.2 Online Execution

Online execution involves dynamically adjusting configuration parameters while the system is running, ensuring that resources are allocated in real time based on various workloads and performance metrics. Unlike offline execution, which relies on historical data, online execution monitors system behavior in real time and applies necessary reconfigurations. This approach is particularly helpful in cloud computing, distributed systems, and real-time applications where demands are unpredictable and require frequent adjustments. Online execution is essential to handle varying workloads, minimize resource utilization, and improve system response. By adapting to real-time telemetry and workload demands, this approach optimizes performance while minimizing manual effort. It has an advantage in scenarios where latency-sensitive applications and auto-scaling strategies require reconfiguration to maintain system efficiency.

There are several studies that show the importance of online execution in configuration management by dynamically adjusting system parameters in real time based on workload variations. "*iTuned*" applies an adaptive database tuning system that continuously monitors workload patterns and dynamically adjusts database configurations, improving query performance and resource efficiency [9]. "*EDLB (Enhanced Dynamic Load Balancing)*" dynamically distributes workloads across virtual machines by analyzing real-time system load, reducing execution time by 22.46% and increasing resource utilization by 3.10% [32]. "*SmartConf*" applies a control-theoretic feedback mechanism to fine-tune CPU, memory, and I/O allocations, ensuring system stability under changing workloads while preventing performance degradation[29]. These studies illustrate how online execution enables continuous self-adaptation, reducing manual intervention while optimizing performance, efficiency, and reliability in dynamic computing environments.

There are several challenges involved in real-time reconfiguration. One major challenge is the computational overhead associated with continuous monitoring and reconfiguration based on workloads, which can lead to latency and over-utilization of resources. Additionally, not accounting for unexpected workload spikes can cause instability in configuration tuning, leading to suboptimal performance if the system fails to adapt quickly. System reliability is another concern, as frequent changes to configuration parameters can result in temporary performance degradation or inconsistencies in configuration. Furthermore, online execution heavily relies on accurate and efficient decision-making algorithms, making it vulnerable to incorrect adjustments caused by noisy data or prediction errors. There are also security concerns, as real-time changes to system configurations can introduce vulnerabilities if not properly managed. Overcoming these challenges requires effective monitoring mechanisms, optimization strategies, and efficient feedback loops to ensure that online execution maintains high performance, stability, and security in dynamic environments.

Online execution is necessary for systems that require real-time adaptability, enabling automated performance tuning and resource optimization. Although it provides better flexibility compared to offline execution, it also requires strong monitoring, predictive modeling, and security mechanisms. As computing environments continue to evolve, AI-driven and control-theoretic approaches will play an increasingly critical role in enhancing online execution strategies.

**Observation 7.2**

Online execution dynamically adjusts configurations in real time, ensuring optimal resource allocation, system responsiveness, and performance based on live workload metrics. It is crucial for cloud computing, distributed systems, and real-time applications, where workloads change unpredictably. However, challenges such as computational overhead, instability risks, and security concerns must be managed with robust monitoring, predictive modeling, and adaptive decision-making to maintain system stability and efficiency.

## 7.3 Hybrid Execution

Hybrid execution is the combination of offline and online execution, offering the benefits of both to optimize configuration parameters and improve performance. Unlike offline execution, which focuses solely on optimizing configuration parameters prior to execution, or online execution, where tuning occurs during runtime using real-time data, hybrid execution combines both approaches to configure parameters, thus ensuring enhanced performance. This execution approach is particularly beneficial in cloud environments, high-performance computing (HPC), and distributed systems, where workloads can exhibit both predictable and unpredictable behavior.

Hybrid execution is essential for systems that require both stability and adaptability. By predefining optimal configuration parameters and continuously making runtime adjustments, hybrid execution ensures that workloads operate efficiently without runtime overhead. This approach is particularly useful in cloud-based applications, where real-time resource allocation is required to handle variable workloads, and in database management systems, where predefined configurations prevent performance degradation while real-time adjustments ensure query efficiency. Additionally, hybrid execution minimizes configuration drift by maintaining a strong baseline while allowing controlled reconfiguration to dynamically optimize resource utilization.

Several research studies highlight the impactfulness of hybrid execution in configuration management by combining the techniques of optimizing configurations before execution (offline) along with real-time adaptive tuning (online) to enhance system performance and efficiency. "*SOPHIA*" uses a hybrid workload-aware reconfiguration system for NoSQL databases, where predefined configurations ensure stability, while real-time adjustments help prevent performance degradation, resulting in a 30% improvement in throughput [21]. Similarly, another research study, "*Optimus-Cloud*", optimizes cloud-hosted databases by predefining baseline configurations while dynamically adjusting resource allocations online based on real-time performance monitoring, leading to improvements in latency and efficiency[20]. "*Deep-Hill*" combines deep neural network-based offline training with an online adaptive resource allocation mechanism, reducing power consumption by 13.33% while ensuring efficient SaaS instance configuration [1]. These studies show the importance of hybrid execution in balancing stability and adaptability, optimizing performance, scalability, and system reliability in dynamic computing environments.

Although hybrid execution is effective in balancing predefined configurations and dynamically reconfiguring parameters according to changing workloads, it also introduces several challenges in configuration management and system efficiency. Maintaining consistent performance across both execution methods is challenging, as any misconfigurations may result in performance drops or conflicts with existing configurations. Computational overhead remains a concern, as real-time tuning still requires processing resources, especially in large-scale distributed systems. Additionally, balancing stability and adaptability is a difficult task—relying too much on offline methods can limit responsiveness, while depending heavily on online execution can lead to instability. As mentioned in the online execution section 7.2, cost optimization must also be considered, as frequent reconfigurations can increase operational costs, particularly in cloud environments. Addressing these challenges requires an intelligent framework or approach that can predict workloads, automate orchestration, and perform cost-aware reconfigurations.

Hybrid execution balances efficiency and adaptability, making it well suited for modern CMS architectures. By integrating offline execution with online execution, it improves performance, scalability, and reliability while minimizing runtime overhead. However, successful hybrid execution implementation depends on intelligent workload prediction, automated orchestration, and cost awareness. Future advancements in AI-driven configuration tuning and cloud-native optimization strategies may further refine hybrid execution, making it even more effective in highly dynamic environments.

**Observation 7.3**

Hybrid execution combines offline and online approaches, ensuring stability through pre-defined configurations while adapting dynamically to workload changes. It is particularly useful in cloud environments, HPC, distributed systems, etc., where workloads vary between predictable and unpredictable patterns. However, challenges such as balancing stability with adaptability, computational overhead, and cost optimization must be addressed through intelligent workload prediction and automated orchestration for effective implementation.

## 8 TECHNIQUES

The core of effective configuration management lies in the techniques, they are the necessary to optimization, adaptation, and performance stability. Without a proper techniques, even the most sophisticated systems would struggle to achieve efficiency, scalability, or responsiveness to the varying workloads. The configuration is only as good as the method used to manage it. This study highlights three fundamental categories of configuration management techniques that define modern systems:

(1) **Telemetry-based monitoring systems**
(2) **Heuristic and Search-Based Optimization approaches**
(3) **Machine Learning–based methods**

The core of effective configuration management lies in the techniques; they are necessary for optimization, adaptation, and performance stability. Without proper techniques, even the most sophisticated systems would struggle to achieve efficiency, scalability, or responsiveness to varying workloads. Configuration is only as good as the method used to manage it. A well-chosen technique can determine whether a system adapts smoothly or fails under pressure. The study highlights three commonly used techniques:

### 8.1 Telemetry-Based Monitoring

Configuration tuning is only effective if there is a feedback system. Systems evolve, workloads change, and static configurations remain optimal in only a handful of scenarios before quickly becoming sub-optimal. Telemetry-based monitoring ensures that configurations remain optimal by actively tracking CPU usage, memory consumption, latency, and other key performance metrics. Unlike manual tuning, telemetry enables automated and real-time reconfiguration, avoiding inefficiencies before performance is impacted.

Several studies have used this technique to optimize configuration parameters in cloud environments. *ConfAdvisor*, for example, continuously analyzes container logs and runtime metrics, detecting misconfigurations, and automatically recommends or applies adjustments to optimize resource allocation [11]. Similarly, another study introduces *OptimusCloud*, which extends this approach by dynamically tuning cloud database parameters and VM configurations based on real-time workload demands, thus reducing costs while maintaining efficiency.

Telemetry-based monitoring is essential for infrastructure optimization. Studies on VMM overhead show that without real-time monitoring, resource inefficiencies go unnoticed, leading to performance bottlenecks and wasted resources [17]. By continuously analyzing system behavior, telemetry-driven frameworks prevent degradation, optimize hypervisor settings, and ensure sustained efficiency—making telemetry not just a tool, but the foundation of self-adaptive configuration management.

This makes telemetry-based monitoring a key part of adaptive configuration tuning. It helps close the feedback loop between how the system behaves and how its settings are adjusted. Without telemetry, configurations remain fixed and cannot respond to changes. With telemetry, systems can keep their settings in sync with changing workloads, ensuring better performance and efficiency.

**Observation 8.1**

Telemetry-based monitoring enables real-time configuration adjustments by continuously tracking system performance metrics such as CPU usage, memory consumption, and latency. Unlike static tuning, it automates reconfiguration to prevent performance degradation and inefficiencies. This approach is essential for adaptive systems, optimizing resources, enhancing efficiency, and maintaining performance in dynamic environments.

### 8.2 Heuristic and Search-Based Techniques

As the configuration space becomes increasingly complex, brute-force search becomes less effective. Exploring all possible parameter combinations is not feasible. Instead, heuristic and search-based optimization techniques efficiently navigate the vast configuration space by using adaptive search, approximation, and evolutionary algorithms. These methods quickly identify near-optimal parameters and are scalable, which is essential for modern systems.

The heuristic and search based optimization techniques are necessary for navigating efficiently across the vast configuration spaces of the modern system/applications. The "*AtConf*" leverages Bayesian optimization methods combined with dimension dropout. This approach enables the tuning of configurations within a high-dimensional space, minimizing the computational load associated with exhaustive searching in big data frameworks like Spark [8]. In containerized environments, a two-stage optimization approach is used, where the first applies a greedy algorithm to distribute workloads for various VM's and then the second approach uses genetic algorithms to resource allocations, which significantly enhances the CPU utilization. and workload balancing [19]. "*EDLB (Enhanced Dynamic Load Balancing Algorithm)*" which improves cloud-based task scheduling by using a heuristic-based method which dynamically allocates workloads based on execution time predictions, optimizing system throughput without requiring expensive reconfigurations [32]. These heuristic-based methods are essential, as this helps in tackling the manual effort of going through he vast complex configuration parameters, thus maintaining performance and making the system more reliable.

In general, these search-based and heuristic techniques are crucial for managing the complexity of modern systems. They help maintain system performance and reliability while efficiently handling large configuration spaces.

**Observation 8.2**

Heuristic and search-based techniques efficiently optimize configuration parameters by navigating large, complex configuration spaces using adaptive search, approximation, and evolutionary algorithms. Unlike brute-force methods, these techniques quickly find near-optimal configurations, improving performance, scalability, and reliability. They are essential for automating configuration tuning, reducing manual effort, and ensuring efficient resource utilization in modern systems.

### 8.3 Machine Learning Techniques

Machine learning-based techniques dynamically optimize configuration parameters by training models with system data and then using the trained models to tune configurations. These methods utilize approaches such as supervised learning, reinforcement learning, and Gaussian process modeling to improve configuration tuning.

Several researches uses the machine learning for optimizing the configuration parameters. The tool "*iTuned*" uses Guassian processes and adaptive sampling to optimize the configurations of database. By running the controlled low-overhead experiments in production environments, ituned helps in reconfiguration without requiring extensive manial intervention [9]. "*Deep-Hill*" is another example which uses Deep Nueral Networks (DNNs) along with Hill climibng algorithms to predict the cloud resource allocation for SaaS workloads. The model is trained extensively on historical workload data, it continuously reconfigures at runtime, thus improving efficiency and cost effectiveness [1].

Machine learning techniques offer a powerful and effective adaptive approach to configuration tuning by enabling systems to learn from historical or real-time data. These methods significantly reduce manual intervention and improve responsiveness to dynamic workloads. As the complexity of cloud-native and distributed systems continues to grow, machine learning-driven configuration management becomes essential and will play a critical role in achieving self-optimization, scalability, and system robustness in modern computing environments.

**Observation 8.3**

Machine learning-based techniques dynamically optimize configuration parameters by leveraging trained models on system data to predict and adjust configurations in real time. These approaches enable automated tuning, reducing manual intervention. By continuously adapting to workload changes, machine learning enhances efficiency, cost-effectiveness, and system performance in complex computing environments.

## 9 FUTURE WORKS AND RESEARCH GAPS

Despite the advancements in the configuration management techniques, several key challenges remain open for future research. One major gap is that there are only limited works in addressing the interoperability of configuration management across the multi-cloud environments. Existing tools often focus on the vendor-specific systems, making configuration optimization in multi-platforms difficult. The Future works should involve exploring the unified configuration frameworks which is capable of adapting to heterogeneous cloud infrastructures while ensuring the seamless integration and policy enforcement across multiple providers.

Another key challenge is self-healing configuration frameworks remain an research challenge. Despite having advancements in AI-driven and automated configuration management, misconfigurations can still occur due to unexpected parameter, unforeseen system states or model inaccuracies in the case of AI-driven approach. Configuration errors can lead to severe performance degradation, vulnerabilities or system failures. Future research should focus on fault-tolerant, self-repairing configurations that can autonomously detect, diagnose and revert faulty settings in real-time. The use of techniques such as causal inference to find the root cause of issues and automated rollback mechanisms would allow systems to correct themselves before problems spread. This would make configuration management more reliable, stable, and resilient in complex computing environments.

**Observation 9**

Key future directions in configuration management include enabling interoperability across multi-cloud platforms and developing self-healing systems. Fault-tolerant configurations that can autonomously detect, diagnose, and recover from errors in real-time are essential for improving system resilience, stability, and security in dynamic environments.

## 10 CONCLUSION

This report explored Configuration Management Systems (CMS) in various dimensions, such as lifecycle stages, scope of the system, execution models and optimization techniques, with the goal of understanding how modern systems can be better configured for performance, scalability, and efficiency. Although the study began by categorizing CMS along structured lines, the most valuable insights emerged from examining how these systems function in real-world dynamic environments.

A key finding is that configuration management must be lifecycle aware to be effective. Pre-deployment strategies help prevent initial misconfigurations, but their static nature limits their long-term value. Deployment-phase adjustments offer some responsiveness, but it is during the runtime phase that the full adaptability of a CMS is tested. Systems capable of dynamically tuning parameters during execution—especially using automated or learning-based techniques—demonstrate significantly better resilience and efficiency.

In addition, the scope of the configuration plays a critical role. Application-level tuning has direct and visible effects on performance, but without supporting infrastructure-level configuration, such as container, host, or cloud-level tuning, the system can remain unbalanced. The cross-level configuration, while more complex to manage, enables better coordination across layers and prevents conflicting adjustments, ultimately improving stability and performance.

The execution model of a CMS also affects its adaptability. Offline execution offers stability and is suitable for predictable environments, but lacks flexibility. Online execution, though more adaptable, introduces overhead and complexity. Hybrid execution emerged as a promising direction, combining the predictability of offline methods with the responsiveness of online techniques, making it suitable for today's dynamic native cloud systems.

Finally, the review highlighted the growing importance of advanced optimization techniques. Telemetry-based monitoring supports real-time feedback loops, heuristic methods help navigate large configuration spaces efficiently, and machine learning approaches, especially those using Gaussian processes and deep learning, are leading the shift toward intelligent, self-adaptive systems. These techniques reduce reliance on manual tuning, improve response to change, and allow systems to optimize themselves based on historical and real-time data.

In summary, modern CMS must be intelligent, adaptive, and context-aware. The path forward lies in building systems that are not only capable of cross-layer coordination and dynamic tuning, but also resilient and autonomous: capable of detecting, correcting, and learning from misconfigurations without human intervention. Future research should focus on enabling interoperability across heterogeneous environments and developing robust self-healing mechanisms to ensure system reliability at scale.

## REFERENCES

[1] Mahmoud Abouelyazid. 2024. Deep-Hill: An Innovative Cloud Resource Optimization Algorithm by Predicting SaaS Instance Configuration Using Deep Learning. *IEEE Access* 12 (2024), 92573–92582. https://doi.org/10.1109/ACCESS.2024.3423339

[2] Usman Ali and Callum Kidd. 2013. Critical success factors for configuration management implementation. *Industrial Management & Data Systems* 113, 2 (2013), 250–264. https://doi.org/10.1108/02635571311303569

[3] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. USENIX.

[4] Amazon Web Services. 2017. Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region. Available at https://aws.amazon.com/message/41926/.

[5] Qifeng Zou Chaoze Lu, Jianchao Zhou. 2025. An optimized approach for container deployment driven by a two-stage load balancing mechanism. *PLOS ONE* 20, 1 (2025), e0317039. https://doi.org/10.1371/journal.pone.0317039

[6] Ting-Chun Chen, Hung-Wei Shen, Yi-Cheng Hung, Chung-Ta Wang, Der-Jiunn Liu, and Chih-Hung Hsu. 2023. conTuner: Improving Resource Usages of Containers Through Auto-Tuning Container Resource Parameters. In *Proceedings of the 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 220–229. https://doi.org/10.1109/CCGrid57596.2023.00032

[7] Thomas Delaet, Wouter Joosen, and Bart Vanbrabant. 2010. A Survey of System Configuration Tools. In *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*. https://www.usenix.net/events/lisa10/tech/full_papers/Delaet.pdf

[8] Hui Dou, Kang Wang, Yiwen Zhang, and Pengfei Chen. 2023. ATConf: auto-tuning high dimensional configuration parameters for big data processing frameworks. *Cluster Computing* 26 (2023), 2737–2755.

[9] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with iTuned. In *Proceedings of the VLDB Endowment*, Vol. 2. VLDB Endowment, 1246–1257.

[10] Jacky Estublier. 2000. Software Configuration Management: A Roadmap. (2000), 279–289. https://doi.org/10.1145/336512.336576

[11] Chiba et al. 2019. Configuration Management in Control Systems. In *Proceedings of the IEEE Conference on Control Systems*. IEEE.

[12] R. Zhanuzak et al. 2024. Optimizing Cloud Computing Performance With an Enhanced Dynamic Load Balancing Algorithm for Superior Task Allocation. *IEEE Access* 12 (2024), 183121–183130. https://doi.org/10.1109/ACCESS.2024.3424321

[13] Syahrul Fahmy, Aziz Deraman, Jamaiah Yahaya, Akhyari Nasir, and Nooraida Shamsudin. 2020. The Evolution of Software Configuration Management. *International Journal of Advanced Trends in Computer Science and Engineering* 9, 1.3 (2020), 50–63. https://doi.org/10.30534/ijatcse/2020/0891.32020

[14] Oluwatoyin Ajoke Farayola, Azeez Olanipekun Hassan, Olubukola Rhoda Adaramodu, Ololade Gilbert Fakeyede, and Monisola Oladeinde. 2023. Configuration Management in the Modern Era: Best Practices, Innovations, and Challenges. *Computer Science IT Research Journal* 4, 2 (2023), 140–157. https://doi.org/10.51594/csitrj.v4i2.613

[15] Xue Han, Tingting Yu, and Michael Pradel. 2021. ConfProf: White-Box Performance Profiling of Configuration Options. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21)*. ACM, Virtual Event, France, 1–8. https://doi.org/10.1145/3427921.3450255

[16] Johannes Hintsch, Carsten Görling, and Klaus Turowski. 2016. A Review of the Literature on Configuration Management Tools. In *Proceedings of the International Conference on Information Resources Management (CONF-IRM)*. http://aisel.aisnet.org/confirm2016/71

[17] Xiaofei Huang, Xiaoying Bai, and Richard M. Lee. 2013. An Empirical Study of VMM Overhead, Configuration Performance, and Scalability. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*. 359–364. https://doi.org/10.1109/SOSE.2013.72

[18] R.O. Kostromin. 2020. Survey of software configuration management tools of nodes in heterogeneous distributed computing environment. *arXiv preprint arXiv:2010.10924* (2020). https://www.researchgate.net/publication/344967752_Survey_of_software_configuration_management_tools_of_nodes_in_heterogeneous_distributed_computing_environment

[19] Chaoze Lu, Jianchao Zhou, and Qifeng Zou. 2025. An optimized approach for container deployment driven by a two-stage load balancing mechanism. *PLoS ONE* 20, 1 (2025), e0317039. https://doi.org/10.1371/journal.pone.0317039

[20] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OptimusCloud: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *Proceedings of the 2020 USENIX Annual Technical Conference*. USENIX.

[21] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. 2019. SOPHIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads. In *Proceedings of the 2019 USENIX Annual Technical Conference*. USENIX.

[22] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. 2019. Understanding and Auto-Adjusting Performance-Sensitive Configurations. In *Proceedings of the 2019 USENIX Annual Technical Conference*. USENIX.

[23] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding Faster Configurations using FLASH. *IEEE Transactions on Software Engineering* (2018). arXiv preprint arXiv:1801.02175.

[24] João P. Serrano and Rúben F. Pereira. 2020. Improvement of IT Infrastructure Management by Using Configuration Management and Maturity Models: A Systematic Literature Review and a Critical Analysis. *Organizacija* 53, 1 (2020), 1–18. https://doi.org/10.2478/orga-2020-0001

[25] Alex Sherman, Philip A. Lisiecki, Andy Berkheimer, and Joel Wein. 2005. ACMS: The Akamai Configuration Management System. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)*. USENIX Association, USENIX, Boston, MA, USA, 245–258.

[26] Akshay Somashekar, Leonardo S. Santiago, Mark Russinovich, Jacob Nelson, Siddhartha Sen, Hitesh Ballani, Justine Sherry, and Anirudh Sivaraman. 2024. Apollo: Predictable and Safe Configuration Deployment for Cloud Applications. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association. https://www.usenix.org/conference/nsdi24/presentation/somashekar

[27] Gagan Somashekar, Karan Tandon, Anush Kini, Chieh-Chun Chang, Petr Husak, Ranjita Bhagwan, Mayukh Das, Anshul Gandhi, and Nagarajan Natarajan. 2024. OPPerTune: Post-Deployment Configuration Tuning of Services Made Easy. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. USENIX Association, Santa Clara, CA, USA.

[28] U.S. Securities and Exchange Commission. 2013. Order Instituting Administrative and Cease-and-Desist Proceedings: Knight Capital Americas LLC. Available at https://www.sec.gov/litigation/admin/2013/34-70694.pdf.

[29] Shu Wang, Chi Li, William Sentosa, Henry Hoffmann, Shan Lu, and Achmad Imam Kistijantoro. 2018. Understanding and Auto-Adjusting Performance-Sensitive Configurations. In *Proceedings of 2018 Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. ACM.

[30] Yanpei Xu, Zachary Musgrave, Yuxing Bao, Min Liu, Jordan Ristorcelli, and Allan Snavely. 2015. Automatic configuration tuning with dependencies. In *Proceedings of the ACM Symposium on Cloud Computing*. https://doi.org/10.1145/2806777.

2806842

[31] Xin Zhang, Yuyang Mao, Mohammad Alizadeh, and Scott Shenker. 2019. Optimal configuration of datacenter applications under dynamic workloads. In *Proceedings of the ACM Symposium on Cloud Computing*. https://doi.org/10.1145/3357223.3362703

[32] Raiymbek Zhanuzak, Mohammed Alaa Ala'anzy, Mohamed Othman, and Abdulmohsen Algarni. 2024. Optimizing Cloud Computing Performance With an Enhanced Dynamic Load Balancing Algorithm for Superior Task Allocation. *IEEE Access* 12 (2024), 183117–183134. https://doi.org/10.1109/ACCESS.2024.3508793