

Exploring Persistent Configuration Decisions for Adaptive Data Provisioning

Giovanni Donato Gallo^a, Federico Bergamini^b, Marco Napolitano and Daniela Micucci^c

Department of Informatics, Systems and Communication, University of Milano - Bicocca, Milan, Italy

Keywords: Software Architecture, Self-Adaptive Systems, Adaptive Data Provisioning, QoS-Aware Selection, Configuration Management.

Abstract: Adaptive data provisioning is a key challenge in modern systems operating over heterogeneous and dynamic data sources, where multiple providers may offer the same functionality with evolving quality characteristics. Existing approaches typically perform data source selection at each invocation based on current context and quality constraints. While this strategy maximizes responsiveness, it introduces runtime overhead and may lead to unstable behavior under noisy or delayed observations. The paper proposes ARES, a configuration-driven architecture for adaptive data provisioning. The approach elevates data source selection to an explicit architectural concern by separating the set of available providers from the active runtime configuration used to resolve requests, which is maintained over time and updated according to externally defined policies. We evaluate ARES through a controlled simulation involving multiple data sources with dynamic quality profiles. Preliminary results suggest that ARES can substantially reduce decision cost while maintaining acceptable levels of execution quality under the considered conditions. Moreover, configuration-level decisions exhibit higher effectiveness, enabling more robust adaptation despite imperfect estimated information. These findings provide initial evidence that configuration-level decision strategies may represent a promising architectural direction for balancing decision cost, execution quality, and adaptation stability in dynamic environments.

1 INTRODUCTION

Modern software systems increasingly operate in environments where the same information can be obtained from multiple heterogeneous and dynamically evolving data sources. This is particularly evident in mobile, IoT, and edge-based systems, where wearable sensors, smart devices, cloud services, and user-provided inputs coexist and provide alternative ways to obtain the same capability. Recent cloud-edge collaborative architectures further emphasize the need for adaptive QoS-aware resource and provider management under dynamic operational conditions (Fang, 2025). While this multiplicity enhances flexibility and resilience, it introduces significant architectural complexity, as data sources exhibit heterogeneous and time-varying non-functional properties such as reliability, latency, and availability (Atzori et al., 2010; Shi et al., 2016; Satyanarayanan, 2017).

Recent visions such as the Cognitive Computing

Continuum further highlight the emergence of highly dynamic distributed ecosystems in which heterogeneous cognitive devices autonomously collaborate across cloud, edge, and opportunistic networks while continuously adapting to environmental changes and QoS constraints (Ferrer et al., 2021). In these environments, service provisioning decisions cannot rely solely on static infrastructures or short-lived runtime optimizations, since providers, connectivity conditions, and resource availability may continuously evolve over time.

A central challenge in these systems is therefore not only *what* information is required, but also *how* it should be obtained and *when* decisions about provider selection should be taken. Existing approaches typically perform data source selection at each invocation based on current context observations and quality constraints. While this maximizes responsiveness, it introduces runtime overhead and may lead to unstable behavior under noisy or delayed observations (Alrifai and Risse, 2009; Herrera et al., 2023). Recent work on continuous QoS-aware adaptation in Cloud-IoT systems has also highlighted the importance of

^a <https://orcid.org/0009-0009-2413-5149>

^b <https://orcid.org/0009-0006-8503-1202>

^c <https://orcid.org/0000-0003-1261-2234>

limiting unnecessary re-optimization and preserving stable placements over time (Russo et al., 2024).

In parallel, research on self-adaptive systems has provided mechanisms for monitoring, analysis, and runtime reconfiguration, often structured around feedback loops such as MAPE-K. Surveys confirm the maturity of this area, highlighting extensive work on optimizing system behavior under dynamic conditions through continuous adaptation and decision-making (Wong et al., 2022; Krupitzer et al., 2015). However, although these approaches support runtime adaptation and long-term reconfiguration, the temporal persistence of selection decisions and the explicit modeling of decision validity over time are often not treated as primary architectural design dimensions.

Similarly, digital twin technologies provide synchronized virtual models to support monitoring, prediction, and proactive decision-making in IoT and cyber-physical systems (Hakiri et al., 2024). Nevertheless, existing approaches remain largely *entity-centric*, focusing on individual resources rather than supporting architectural reasoning over alternative providers and their selection over time. Existing approaches optimize *which provider to select*, but largely overlook *when* decisions should be taken and *how long* they should remain valid. To address this gap, we introduce ARES (Adaptive Runtime Execution over heterogeneous Sources), a configuration-driven architectural approach for adaptive data provisioning. The key idea is to elevate data source selection from per-invocation decisions to a persistent architectural configuration, maintained over time and updated at explicitly defined points according to external policies. In this context, persistence does not imply static deployment, but rather the explicit maintenance of provider-selection decisions across multiple invocations.

ARES adopts a configuration-driven perspective in which provider-selection decisions persist over time and are revised through explicit policies. This separation between execution and adaptation enables reasoning not only about which provider is optimal at a given time, but also about the trade-off between decision frequency, execution quality, and system stability. In particular, ARES makes explicit the *temporal dimension of decision making*, typically implicit in existing approaches. In ARES, the temporal dimension refers to explicitly modeling *i)* when provider-selection decisions are re-evaluated and *ii)* how long a selected configuration remains active before reassessment. More importantly, it introduces when decisions are taken and how long they remain valid as first-class architectural design dimensions.

The paper presents an initial architectural explo-

ration of persistent configuration decisions for adaptive data provisioning. In particular, the contributions of this work are:

- A *configuration-driven architectural model* that elevates data source selection to a persistent and explicit architectural concern.
- A *two-level adaptation model* separating configuration decisions from data production and quality management.
- A *policy-driven adaptation framework*, enabling customizable decision strategies.

The evaluation, conducted in a controlled digital health monitoring scenario, shows that ARES reduces decision cost by more than one order of magnitude while maintaining comparable execution quality. These preliminary results highlight that the placement and frequency of decision-making may represent relevant architectural design parameters.

2 ARES ARCHITECTURE

This section introduces ARES, an architectural approach for adaptive data provisioning in environments with heterogeneous data sources offering the same information under varying quality conditions. ARES is grounded on three principles: *(i)* treating data source selection as a configuration-level concern, *(ii)* separating candidate providers from the active runtime configuration, and *(iii)* distinguishing global decisions (which provider to select) from local ones (how data is produced and maintained).

2.1 Architectural Model

ARES is built on a set of architectural abstractions treated as first-class entities (Figure 1). These define a capability-driven model that decouples application requirements from concrete providers while enabling reasoning about data quality, context, and adaptation.

Tasks. A *task* represents an application-level request enriched with quality requirements:

$$t = (c, KPI_{req})$$

where c is the required capability and KPI_{req} denotes quality constraints. Tasks express application needs declaratively, without referring to concrete providers, thus decoupling application logic from data provisioning.

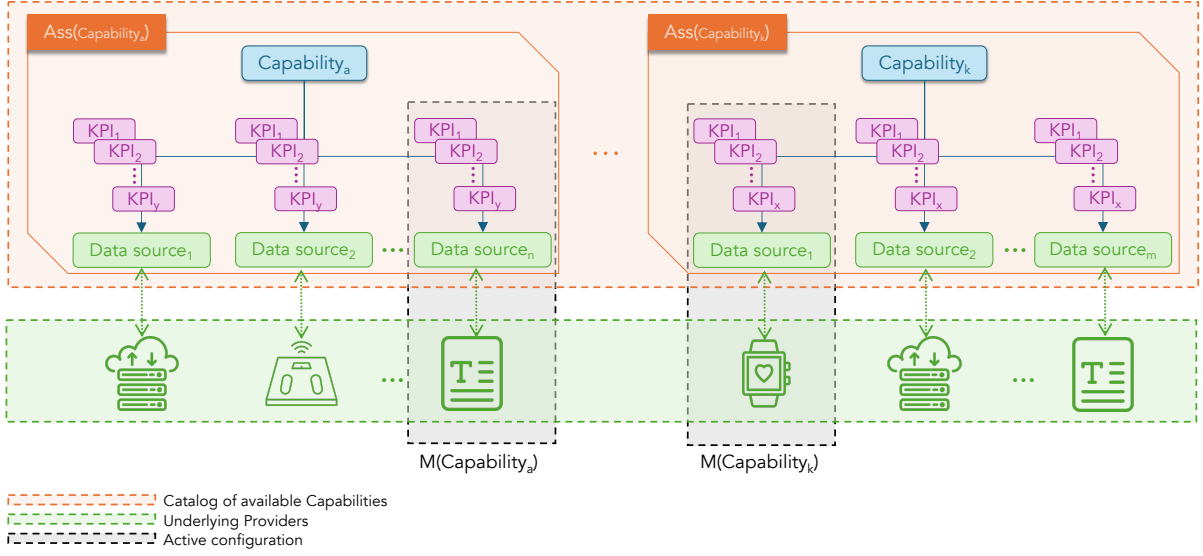


Figure 1: Architectural model of ARES.

Example: A monitoring application may require obtaining the patient's heart rate with a reliability of at least 0.9 and within a response time of at most 2 seconds.

Capabilities. A *capability* specifies what information is required, independently of how it is obtained. ARES adopts a unified model covering both domain-level and system-level concerns, enabling application data and environmental observations to be handled within the same abstraction.

Example: A monitoring application may require domain capabilities such as `getBloodPressure` and `getHeartRate`, while the system may rely on capabilities such as `getBandwidth` and `getBatteryLevel` to observe environmental conditions.

Data Sources. A *data source* is an abstraction representing an entity that provides one or more capabilities. Multiple data sources may offer the same capability with different quality. ARES does not assume that providers are strictly equivalent. Instead, providers expose alternative ways of obtaining information that satisfy the same application-level intent while exhibiting potentially different quality profiles, contextual limitations, and operational characteristics. This abstraction decouples provider selection from runtime realization, enabling different data provisioning and quality management strategies without affecting the rest of the architecture.

Example: The capability `getHeartRate` may be provided by a smartwatch (high accuracy, limited availability), a cloud service (medium accuracy, high availability), or manual input (low accuracy, always available).

Capability–Data Source Associations. For each capability c , ARES maintains the set of candidate data sources that can provide it:

$$Assoc(c) = \{(ds_i, KPI_{i,c})\}$$

where ds_i is a candidate data source and $KPI_{i,c}$ denotes the quality profile of ds_i when providing capability c . Each association captures the quality profile of a data source for a given capability, explicitly representing alternative providers and their trade-offs.

Example: For the capability `getHeartRate`, a smartwatch may exhibit high reliability and accuracy but limited availability, a cloud service moderate accuracy with high availability, and manual input lower reliability but full availability.

Active Configuration and Mapping. ARES maintains a single *active configuration*:

$$M : C \rightarrow D$$

At runtime, requests are resolved through $M(c)$, without per-invocation re-selection. This design separates decision-making from execution, ensuring predictable and lightweight runtime behavior while delegating adaptation to policy-driven mechanisms.

Example: If `getHeartRate` is mapped to a smartwatch, all requests are resolved through it without re-selection. Changes occur only when the configuration is updated by the adaptation mechanism.

Confidence-Aware Results. A capability invocation returns:

$$result = (value, confidence)$$

The confidence reflects the estimated confidence level associated with the produced data, derived from the quality profile of the selected data source and current context conditions.

Example: A heart rate acquisition is 72 bpm from a smartwatch and may have confidence 0.95, while the same value from manual input may have confidence 0.6.

2.2 Policy-Driven Adaptation

ARES adopts a two-level adaptation model driven by externally defined policies. At the global level, *configuration policies* determine when and how the mapping M is updated, while at the local level, *data source policies* control how data is produced and how quality estimates evolve over time.

The two levels are loosely coupled but interact through quality estimates and contextual conditions: local variations in quality may trigger global reconfiguration, while global decisions determine which local behavior becomes active.

This design supports both *intra-system variability* (different behaviors within the same system, including multiple instances of the same data source configured with context-specific policies) and *inter-system variability* (reuse across applications with different policies).

This design reframes data provisioning as a configuration-level architectural concern, enabling explicit reasoning over alternative providers and their quality trade-offs while preserving efficient execution. However, the effectiveness of the approach depends on the accuracy of quality estimates and on the adequacy of the adopted policies. The current version of ARES focuses on single-provider selection in order to isolate and study the architectural implications of configuration persistence and decision timing. Multi-provider fusion strategies, redundancy mechanisms, and aggregated capability composition are intentionally out of scope of this study and represent a key direction for future work.

2.3 Layered Realization

The ARES architectural model is independent of specific implementation choices. To illustrate its operationalization, we outline a possible layered realization, which does not constrain the model or its abstractions.

In this view, the architecture can be organized into four conceptual layers. The *Domain Layer* expresses application requirements through tasks, remaining independent of data provisioning concerns. The *Capability Layer* provides the abstract interface through which information is requested. The *Configuration and Provisioning Layer* maintains capability–data source associations and the active mapping M , and resolves capability invocations. Finally, the *Runtime Layer* contains the executable counterparts of data sources. Within this layer, data sources implement their behavior according to associated policies, including data generation strategies (e.g., acquisition, caching, or simulation) and quality estimation mechanisms. Adaptive reconfiguration operates on the mapping M and can be realized through standard feedback-loop mechanisms (e.g., MAPE-K (Kephart and Chess, 2003), AWARE (Sanwouo Chekam et al., 2025)). Similarly, data sources can be implemented using runtime abstractions such as Digital Twins to encapsulate provider behavior and maintain quality estimates.

3 EVALUATION

This section evaluates the effectiveness of ARES in supporting adaptive data provisioning in the presence of multiple alternative providers. The evaluation considers a controlled scenario involving a single capability (`getHeartRate`) provided by three alternative sources with different quality profiles. Focusing on a single capability allows isolating the behavior of the selection and adaptation mechanisms while preserving the core challenge of choosing among alternatives under dynamic conditions.

The evaluation should be interpreted as a preliminary proof-of-concept study aimed at isolating the architectural implications of configuration persistence under controlled conditions, rather than as a comprehensive assessment of adaptive data provisioning in real-world deployments.

3.1 Research Questions

The study addresses the following research questions:

- **RQ1 – Execution Quality:** To what extent does ARES maintain application-level constraint satisfaction when capability invocations are resolved through a configuration-level mapping under dynamic data source conditions?
- **RQ2 – Configuration Adequacy:** To what extent are reconfiguration decisions effective in restoring constraint satisfaction under changing data source conditions?
- **RQ3 – Decision–Quality Trade-off:** What trade-off arises between decision cost and execution quality when adopting configuration-level selection instead of per-request selection?

3.2 Experimental Setup

We consider a digital health monitoring scenario in which the system repeatedly invokes the capability `getHeartRate` under explicit quality constraints: reliability ≥ 0.9 and latency ≤ 2 s.

3.2.1 System Model

The system model defines the evaluated data sources, the quality representations used in the experiment, and the observation model adopted to derive estimated values from ground truth.

Data Sources. The capability `getHeartRate` is provided by three alternative data sources, each characterized by a capability-specific quality profile defined in terms of reliability and latency. The initial quality profiles are as follows:

- smartwatch: reliability = 0.95, latency = 1.2s
- cloud-based service: reliability = 0.9, latency = 1.8s
- manual input: reliability = 0.85, latency = 1.0s

Ground Truth and Estimated Quality. For each data source, we distinguish between: (i) *ground-truth quality values*, representing the actual provider behavior, and (ii) *estimated quality values*, used by the system for decision making.

In this study, reliability denotes the probability that a provider returns a semantically correct and usable result for the requested capability. Reliability therefore captures successful service execution and result correctness rather than measurement accuracy. Accuracy-related aspects are intentionally abstracted into the overall quality profile of a provider and are not modeled as a separate dimension. In other words, reliability represents the probability that a capability invocation successfully returns a usable result.

The system never accesses ground-truth values directly. All decisions rely exclusively on estimated values, while all evaluation metrics are computed against ground-truth values. This models realistic conditions in which monitoring may be partial, delayed, or noisy.

Estimated Quality Generation. Estimated KPI values are derived from ground-truth values by introducing both delay and noise. Specifically, reliability and latency are observed with a fixed delay of 5 and 3 time steps, respectively. In addition, uniform noise is applied in the ranges $[-0.03, 0.03]$ for reliability and $[-0.2, 0.2]$ for latency. This models realistic monitoring conditions in which observations are delayed and imprecise.

3.2.2 Workload and Execution Model

The evaluation is based on a discrete-event simulation consisting of $N = 1,200$ invocations of the `getHeartRate` capability, indexed by a logical time step $t \in \{0, \dots, N-1\}$.

Each time step represents a request issued by the application. For readability, steps can be interpreted as occurring at regular intervals, although the evaluation does not rely on real-time assumptions.

At each time step t :

1. the application invokes `getHeartRate`;
2. a data source is selected according to the decision strategy under evaluation;
3. the selected data source produces a value;
4. ground-truth values are used only to assess constraint satisfaction.

3.2.3 Decision Strategies

We compare two alternative strategies that differ in how decisions are applied over time: (i) a configuration-level strategy implemented by ARES, and (ii) a per-request selection baseline.

Configuration Update Policy in ARES.

To isolate the architectural effect of configuration-level selection, ARES adopts a periodic reassessment policy. The active mapping is re-evaluated every 25 invocations. The reassessment interval was intentionally fixed to a coarse-grained value in order to emphasize the architectural trade-off between responsiveness and decision stability. The goal of the study is not to identify an optimal interval, but to investigate the effects of persistent configuration decisions compared to continuous reselection. At each reassessment:

1. the system retrieves the latest estimated quality values;
2. it evaluates whether the currently mapped provider is still adequate;
3. if needed, it selects a new provider;
4. the mapping is updated accordingly.

Between reassessment points, the mapping remains unchanged.

Selection Policy. Both *ARES* and the baseline adopt the same deterministic selection policy. Given a capability c and the set of candidate data sources providing c , the policy operates at each time step t as follows:

1. *Feasibility filtering.* Only candidates whose estimated quality values satisfy all required constraints are considered:

$$Feasible(c, t) = \{ds_i \mid KPI_{i,c}^{est}(t) \models KPI_{req}\}$$

2. *Ranking.* Among feasible candidates, the selected source is the one maximizing:

$$score(ds_i, t) = w_r \cdot rel_i^{est}(t) - w_l \cdot \widehat{lat}_i^{est}(t)$$

where $rel_i^{est}(t) \in [0, 1]$ is the estimated reliability and $\widehat{lat}_i^{est}(t) \in [0, 1]$ is the normalized estimated latency of ds_i at time t . The weights are set to $w_r = 0.7$ and $w_l = 0.3$. Latency is normalized as:

$$\widehat{lat}_i^{est}(t) = \min\left(\frac{lat_i^{est}(t)}{L_{max}}, 1\right)$$

where $L_{max} = 4s$ accounts for the maximum observed latency across all scenarios and estimation noise.

3. *Fallback.* If no candidate satisfies all required constraints, the source with the highest score among all candidates is selected.

This policy is intentionally simple and domain-independent, so that observed differences can be attributed to the architectural strategy rather than to the complexity of the selection heuristic.

Baseline (Per-request Selection). As a baseline, we consider a *per-request selection strategy*. At each invocation, the system selects a data source by applying the same feasibility filtering and ranking policy adopted by *ARES*, based on the current estimated values.

Differently from *ARES*, the baseline does not maintain a persistent configuration: the selection procedure is executed independently at each time step.

The baseline does not aim to represent a specific industrial system, but rather an intentionally reactive

strategy that approximates continuous reselection under noisy observations. The comparison is intended to isolate the effects of decision persistence and reassessment frequency.

3.2.4 Evaluation Scenarios

We define three controlled scenarios that model different operating conditions affecting provider quality:

- **Stable.** Provider quality profiles remain constant over time.
- **Degradation.** The quality of the initially selected provider (smartwatch) progressively deteriorates over time. At each time step, its reliability decreases by 0.0003 while its latency increases by 0.0015, up to predefined bounds.
- **Dynamic.** Abrupt performance changes affect one provider at a time through non-overlapping degradation intervals of fixed duration (150 invocations). During each interval, the affected provider experiences a temporary drop in quality, while the others remain stable.

All perturbations are injected at predefined logical time steps and affect ground-truth quality values. Estimated values evolve accordingly based on the adopted observation model.

3.3 Evaluation Methodology

Let N be the total number of invocations of the `getHeartRate` capability. We denote by $ds(t)$ the data source selected at time step t according to the decision strategy. For each time step $t \in \{0, \dots, N-1\}$, we define binary indicators capturing whether the selected data source satisfies the required quality constraints:

$$Comp(t) = \begin{cases} 1 & \text{if } KPI_{ds(t),c}^{real}(t) \models KPI_{req} \\ 0 & \text{otherwise} \end{cases}$$

$$Viol(t) = 1 - Comp(t).$$

3.3.1 RQ1: Execution Quality

RQ1 evaluates the extent to which the system satisfies application-level quality constraints during execution.

Constraint Satisfaction Rate (CSR).

$$CSR = \frac{1}{N} \sum_{t=0}^{N-1} Comp(t)$$

It measures the fraction of invocations satisfying all required constraints.

Violation Burst Length. Let $\{B_1, \dots, B_k\}$ be the sequences of consecutive violating invocations (i.e., $Viol(t) = 1$). The average burst length is:

$$Burst_{avg} = \frac{1}{k} \sum_{i=1}^k |B_i|$$

This metric captures how long violations persist once they appear.

3.3.2 RQ2: Configuration Adequacy

RQ2 evaluates how effective reconfiguration decisions are in restoring constraint satisfaction under changing provider conditions.

Reconfiguration Effectiveness. For each reconfiguration event i , we define:

$$Eff(i) = \begin{cases} 1 & \text{if } Comp(t_{reconf}^{(i)}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The overall effectiveness is:

$$Effectiveness = \frac{1}{R} \sum_{i=1}^R Eff(i)$$

where R is the total number of reconfigurations.

This metric evaluates whether decisions based on estimated values successfully restore constraint satisfaction according to ground-truth conditions immediately after a reconfiguration takes place. The metric intentionally evaluates the immediate correctness of reconfiguration decisions rather than their long-term stability, since the goal of RQ2 is to isolate the local effectiveness of reassessment under imperfect and delayed estimated information. Consequently, the metric captures whether the reassessment process selects an appropriate provider at the time the decision is taken, independently from subsequent environmental variations or future provider degradations. This metric complements CSR by focusing on local decision correctness rather than long-term execution behavior.

3.3.3 RQ3: Decision-Quality Trade-off

RQ3 evaluates the trade-off between decision cost and execution quality.

Total Selection Operations. Let

$$Sel(t) = \begin{cases} 1 & \text{if a selection operation is executed at time } t \\ 0 & \text{otherwise} \end{cases}$$

Then:

$$TotalSelOps = \sum_{t=0}^{N-1} Sel(t)$$

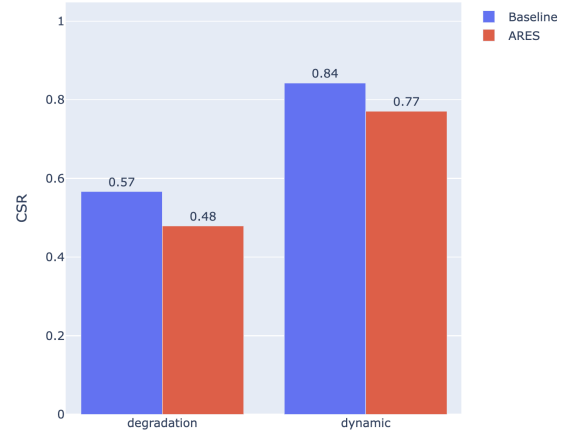


Figure 2: RQ1 – Constraint Satisfaction Rate (CSR) across scenarios.

This metric reflects the computational effort required to perform decision making. In the baseline, selection is performed at every invocation, while in ARES it is performed only when the configuration is initialized or updated.

Number of Constraint-Satisfying Invocations.

$$TotalComp = \sum_{t=0}^{N-1} Comp(t)$$

Decision Cost per Constraint-Satisfying Invocation.

$$Cost_{comp} = \begin{cases} \frac{TotalSelOps}{TotalComp} & \text{if } TotalComp > 0 \\ \infty & \text{otherwise} \end{cases}$$

This metric measures how many selection operations are required, on average, to obtain one constraint-satisfying invocation. Lower values indicate a better trade-off between decision cost and execution quality.

Since both strategies use the same selection policy, this metric isolates the impact of decision timing (configuration-level vs per-request) from that of the selection heuristic.

3.4 Results

3.4.1 RQ1: Execution Quality

Figure 2 reports the constraint satisfaction rate (CSR) for the degradation and dynamic scenarios.

In both scenarios, the baseline achieves higher constraint satisfaction than ARES. This is expected, as the baseline performs a selection at every invocation and can immediately react to changes in provider quality. In contrast, ARES updates the configuration

Table 1: RQ1 – Violation bursts in the degradation scenario.

	#V	Min	Max	Avg	Med	Mode
Baseline	263	1	14	1.98	1	1 (137)
ARES	12	25	150	52.08	37.5	25 (6)

#V: total violations. Min/Max/Avg/Med: burst length statistics. Mode: most frequent burst (count in parentheses).

only periodically, introducing a delay in reacting to changes.

Despite this limitation, ARES maintains acceptable execution quality within the controlled experimental setting. CSR decreases by about 16% in the degradation scenario and 8% in the dynamic scenario compared to the baseline. This indicates that configuration-level selection can sustain acceptable execution quality under changing conditions despite reduced responsiveness.

To better understand the nature of violations, Table 1 reports violation-burst statistics for the degradation scenario. The baseline exhibits many violations, but they are typically short, reflecting frequent but quickly resolved issues. In contrast, ARES produces far fewer violation bursts, but with substantially longer duration. This behavior is a direct consequence of the periodic reconfiguration policy: once a configuration becomes inadequate, it is maintained until the next reassessment point.

Overall, these results highlight a key trade-off: the baseline minimizes violation duration through continuous adaptation, whereas ARES reduces the frequency of violations at the cost of longer persistence when they occur.

3.4.2 RQ2: Configuration Adequacy

Figure 3 reports reconfiguration effectiveness in both scenarios.

ARES consistently achieves higher reconfiguration effectiveness than the baseline, with an improvement of about 22% in the degradation scenario and up to 73% in the dynamic scenario. The larger gain in the dynamic case highlights the robustness of configuration-level decisions under rapidly changing conditions.

These results indicate that, although ARES reacts less frequently, its decisions are generally more reliable. Since reconfiguration is performed less often, each decision has a stronger impact and is based on more stable estimated conditions. In contrast, the baseline performs frequent selections based on potentially noisy estimates, increasing the likelihood of choosing providers that do not satisfy ground-truth constraints.

The periodic reassessment policy of ARES also

explains the longer violation bursts observed in RQ1. As reconfiguration is evaluated every 25 invocations, a delay between the onset of inadequacy and the corresponding reconfiguration is expected, reflecting the trade-off between responsiveness and stability.

Moreover, because ARES performs fewer reconfiguration actions, it is less exposed to ineffective adaptations caused by transient fluctuations in estimated quality values, although this aspect is not directly quantified.

Overall, these results show that configuration-level strategies improve the effectiveness and robustness of adaptation decisions, while trading some responsiveness for greater stability. This result complements RQ1: although ARES achieves lower overall compliance due to delayed reactions, its individual reconfiguration decisions are more reliable.

3.4.3 RQ3: Decision-Quality Trade-off

Figure 4 shows that ARES shifts the operating point towards significantly lower decision cost with only a moderate decrease in execution quality.

The baseline achieves slightly higher CSR values, but at a significantly higher decision cost, as it performs a selection at every invocation. In contrast, ARES drastically reduces the number of selection operations while maintaining acceptable execution quality.

In the degradation scenario, decision cost is reduced by more than one order of magnitude, while CSR decreases by about 16%. In the dynamic scenario, a similar cost reduction is achieved with only an 8% decrease in constraint satisfaction.

Overall, ARES provides a significantly more efficient solution, trading a limited reduction in execution quality for a major decrease in decision cost.

It is important to note that ARES is not intended to outperform per-request selection in terms of execution quality. Rather, its goal is to expose and manage the architectural trade-off between decision frequency, adaptation stability, and decision cost. The results suggest that reducing decision frequency may lead to lower responsiveness, but can also improve decision robustness and significantly reduce adaptation overhead.

Taken together, the results across all research questions show that ARES shifts adaptation from continuous, reactive decision making to a more stable and efficient configuration-level approach, achieving acceptable execution quality while substantially improving decision effectiveness and reducing computational overhead.

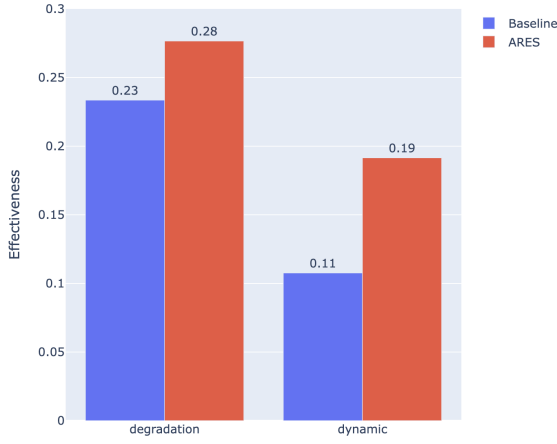


Figure 3: RQ2 – Reconfiguration effectiveness across scenarios.

4 THREATS TO VALIDITY

This section discusses the main threats to the validity of the evaluation and the adopted mitigation measures.

Internal Validity. A first threat lies in the simulation-based nature of the evaluation. Data source behavior, including degradation and dynamic variations, is modeled through controlled perturbations that may not capture the full complexity of real-world systems. Furthermore, the numerical values adopted for provider reliability, latency, observation delays, and reassessment intervals were not derived from a specific deployment. Rather, they were intentionally selected to create controlled conditions in which provider quality evolves over time and delayed observations may affect adaptation decisions. Consequently, the reported results should not be interpreted as quantitative predictions for a particular application domain, but as evidence of the architectural behavior induced by persistent configuration decisions. Although the specific values are synthetic, they were chosen to create non-trivial situations in which delayed observations, quality degradation, and reconfiguration decisions interact, thereby enabling the study of the architectural trade-offs targeted by ARES.

A second threat concerns the generation of estimated quality values: KPIs are derived from ground truth by introducing delay and noise, whose parameters may influence the observed behavior.

Execution quality is measured through constraint satisfaction rate and violation burst length, while configuration adequacy is assessed through reconfiguration effectiveness, and decision cost through cost per compliant invocation. Although these metrics capture

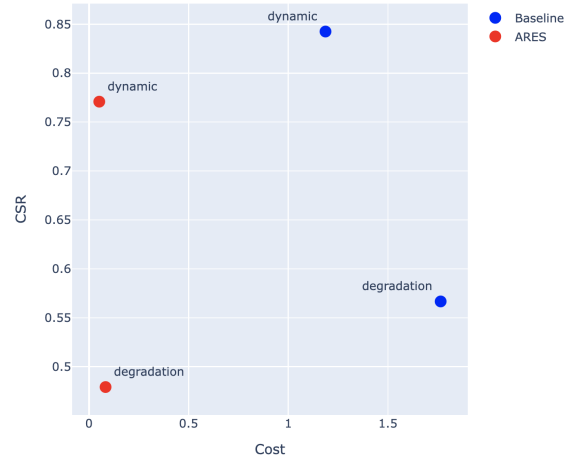


Figure 4: RQ3 – Trade-off between decision cost and execution quality.

the main trade-offs targeted in this study, they do not cover all possible system qualities.

The evaluation is based on a fixed number of invocations ($N = 1,200$) and deterministic settings, which support reproducibility but do not account for statistical variability across runs.

To mitigate these threats, we use a controlled setting where all factors except the decision strategy are kept constant. Both ARES and the baseline rely on the same estimated information and selection policy, ensuring that differences are attributable to the architectural strategy rather than the selection heuristic.

External Validity. The evaluation intentionally abstracts away from several aspects commonly found in real deployments, including multi-capability dependencies, provider fusion, heterogeneous workloads, contextual correlations among providers, and application-specific criticality constraints. Consequently, the current results should not be interpreted as evidence of general superiority of configuration-level strategies, but rather as an initial characterization of their architectural trade-offs.

The evaluation focuses on a single capability and a limited set of data sources. While this isolates the effect of decision strategies, it may limit generalizability to scenarios with multiple capabilities, more complex dependencies, or richer provider ecosystems. Moreover, the simulated setting may not fully capture the irregular behavior of real-world systems. In addition, aspects such as energy consumption, user effort, or economic cost are not considered, which may further limit the applicability of the results in real-world settings.

However, the goal of the evaluation is not to provide a comprehensive performance assessment, but

to isolate the architectural impact of decision timing under controlled conditions. We expect the observed trends to generalize qualitatively to more complex settings, although further validation in real-world deployments is required.

5 RELATED WORK

This section discusses the main research areas related to this work, focusing on self-adaptive systems, QoS-aware selection, and digital twin technologies.

Self-Adaptive Systems. Self-adaptive systems provide mechanisms for dynamically adjusting system behavior in response to changes in the environment or system state. The MAPE-K feedback loop (Kephart and Chess, 2003) remains the canonical reference model for structuring adaptation processes, while runtime models support reasoning and reconfiguration decisions (Morin et al., 2009). Recent surveys highlight the maturity of this field across cloud, IoT, and cyber-physical systems (Wong et al., 2022; de Lemos et al., 2013). These approaches focus on optimizing performance and quality of service through continuous monitoring and runtime decision-making, with reusable mechanisms for monitoring, planning, and execution (Krupitzer et al., 2015). Despite these advances, adaptation is typically treated as a *continuous control problem*, where decisions are taken at runtime in response to observed changes. As a result, the temporal dimension of decision-making (i.e., how often decisions should be taken and how long they should remain valid) is rarely modeled explicitly as an architectural concern.

More recent work has also investigated continuous QoS-aware adaptation in Cloud-IoT systems, focusing on incremental placement adaptation and mechanisms aimed at limiting unnecessary re-optimization under dynamic conditions (Herrera et al., 2023). These approaches partially acknowledge the importance of adaptation stability over time, although the persistence and validity of configuration decisions are still not treated as first-class architectural concerns.

QoS-Aware Selection and Resource Management. Quality-aware selection has been extensively studied in service computing, cloud-edge systems, and IoT environments. Classical approaches formulate service selection and composition as optimization problems over QoS attributes such as latency, reliability, and cost (Alrifai and Risse, 2009; Oh et al., 2008). More recent work extends this paradigm to distributed

and cloud-edge environments, proposing adaptive resource management and offloading strategies that dynamically select providers based on current conditions (Mao et al., 2017; Russo et al., 2024). In IoT settings, heuristic and multi-objective optimization techniques are used to handle heterogeneous resources and dynamic constraints (Hamzei et al., 2023).

QoS-aware dynamic fog service provisioning approaches have also explored runtime deployment and release decisions to maintain latency and QoS guarantees in IoT environments (Yousefpour et al., 2019).

Recent approaches employ probabilistic inference, optimization, and reinforcement learning techniques to support QoS-aware service composition and dynamic service distribution under uncertain IoT conditions (Sefati et al., 2026; Chen et al., 2025).

These approaches effectively identify suitable providers under changing conditions, but typically operate at the level of individual invocations or short decision horizons. Consequently, they do not address the architectural implications of maintaining a persistent configuration over time, nor the trade-offs between decision frequency, stability, and execution quality. In particular, existing solutions optimize *which provider to select*, but do not explicitly consider *when* decisions should be performed or how long they should persist.

Digital Twins. Digital twins provide synchronized virtual representations of physical or logical entities, enabling monitoring, simulation, and predictive decision-making. The concept was introduced in manufacturing and cyber-physical systems (Grieves, 2014), and recent surveys highlight its adoption across domains such as IoT, smart infrastructures, and healthcare (Barricelli et al., 2019; Hakiri et al., 2024). Recent work emphasizes their role in data-driven adaptation and real-time analytics in dynamic environments (Lu et al., 2020). However, most approaches remain *asset-centric*, focusing on individual entities rather than supporting architectural reasoning over multiple alternative providers. Furthermore, although digital twins enable predictive decision-making, they do not address the management of selection decisions as persistent architectural configurations, nor the impact of decision frequency on system stability and cost.

Positioning of ARES. ARES explores the implications of treating data provider selection as a persistent architectural concern and explicitly models the temporal dimension of decision-making. While prior work focuses on optimizing selection at runtime, ARES adopts a configuration-driven approach in

which decisions are maintained over time and updated through explicit policies. This enables reasoning not only about which provider is optimal, but also about how often decisions should be updated and how long they should remain valid.

By elevating selection from a local runtime mechanism to a global architectural construct, ARES re-frames adaptive provisioning as an architectural configuration problem rather than solely a runtime optimization task. This perspective enables reasoning about stability, adaptation cost, and decision persistence as explicit architectural dimensions.

6 CONCLUSION

This paper presented ARES, a preliminary architectural exploration of configuration-driven adaptive data provisioning in environments with multiple alternative data sources. The key idea is to shift decision making from per-request selection to configuration-level decisions, explicitly treating when decisions are taken and how long they remain valid as architectural design dimensions.

The evaluation shows that this design leads to a different trade-off between responsiveness and efficiency. While per-request selection achieves slightly higher execution quality due to its immediate reactivity, it incurs a significantly higher decision cost and is more sensitive to noise in estimated quality values. In contrast, ARES reduces decision overhead by more than one order of magnitude while maintaining comparable levels of execution quality, and exhibits more robust adaptation behavior. These preliminary results suggest that configuration-level selection may provide a viable trade-off between responsiveness, stability, and decision overhead in dynamic provisioning environments.

Future work will investigate adaptive configuration policies, richer decision models including additional non-functional properties, and more complex scenarios involving multiple capabilities and tasks.

DATA AVAILABILITY

The artifacts supporting the evaluation of the proposed approach are publicly available at: <https://github.com/sal-unimib/ares-simulation-evaluation>. The repository contains all materials required to reproduce the experimental evaluation, including the source code for the simulated scenarios, the components used for simulation data generation, and the

scripts and datasets employed to compute the evaluation metrics. Documentation is provided to support the setup of the experimental environment, the execution of the simulations, and the replication of the reported results.

ACKNOWLEDGEMENTS

This research is funded by the National Plan for NRRP Complementary Investments (PNC, established with the decree-law 6 May 2021, n. 59, converted by law n. 101 of 2021) in the call for the funding of research initiatives for technologies and innovative trajectories in the health and care sectors (Directorial Decree n. 931 of 06-06-2022) - project n. PNC0000003 - AdvANced Technologies for Human-centrEd Medicine (project acronym: ANTHEM).

REFERENCES

- Alrifai, M. and Risse, T. (2009). Combining global optimization with local selection for efficient qos-aware service composition. In *Proceedings of the International Conference on World Wide Web (WWW)*.
- Atzori, L., Iera, A., and Morabito, G. (2010). *Computer Networks*, 54(15):2787–2805.
- Barricelli, B. R., Casiraghi, E., and Fogli, D. (2019). A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access*, 7:167653–167671.
- Chen, S., Li, D., and Mohajer, A. (2025). Qos-aware multi-agent ddpg for adaptive edge service distribution in intelligent wireless communication networks. *Ain Shams Engineering Journal*, 16(9):103543.
- de Lemos, R., Giese, H., Müller, H. A., and al. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer.
- Fang, Z. (2025). Adaptive qos-aware cloud-edge collaborative architecture for real-time smart water service management. In *Proceedings of the International Conference on Management Science and Computer Engineering (MSCE)*.
- Ferrer, A. J., Becker, S., Schmidt, F., Thamsen, L., and Kao, O. (2021). Towards a cognitive compute continuum: An architecture for ad-hoc self-managed swarms.
- Grieves, M. (2014). Digital twin: Manufacturing excellence through virtual factory replication. *White Paper*.
- Hakiri, A., Gokhale, A., Yahia, S. B., and Mellouli, N. (2024). A comprehensive survey on digital twin for future networks and emerging internet of things industry. *Computer Networks*, 244:110350.
- Hamzei, M., Khandagh, S., and Jafari Navimipour, N. (2023). A quality-of-service-aware service composition method in the internet of things using a multi-

- objective fuzzy-based hybrid algorithm. *Sensors*, 23(16):7233.
- Herrera, J. L., Berrocal, J., Forti, S., et al. (2023). Continuous qos-aware adaptation of cloud-iot application placements. *Computing*, 105:2037–2059.
- Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.
- Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., and Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206.
- Lu, Y., Liu, C., Wang, K. I.-K., Huang, H., and Xu, X. (2020). Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and Computer-Integrated Manufacturing*, 61:101837.
- Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358.
- Morin, B., Barais, O., Jézéquel, J.-M., Fleurey, F., and Solberg, A. (2009). Models@run.time to support dynamic adaptation. *Computer*, 42(10):44–51.
- Oh, S.-C., Lee, D., and Kumara, S. R. T. (2008). Effective web service composition in diverse and large-scale service networks. *IEEE Transactions on Services Computing*, 1(1):15–32.
- Russo, G. R., Ferrarelli, D., Pasquali, D., Cardellini, V., and Presti, F. L. (2024). Qos-aware offloading policies for serverless functions in the cloud-to-edge continuum. *Future Generation Computer Systems*, 156:1–15.
- Sanwouo Chekam, B. P., Quinton, C., and Temple, P. (2025). *Breaking the Loop: AWARE is the New MAPE-K*.
- Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.
- Sefati, S. S., Sefati, S. T., Vulpe, A., and Fratu, O. (2026). Adaptive qos-aware service composition in the internet of things using a hybrid bayesian network-based optimization algorithm. *IEEE Internet of Things Journal*, pages 1–1.
- Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.
- Wong, T., Wagner, M., and Treude, C. (2022). Self-adaptive systems: A systematic literature review across categories and domains. *Information and Software Technology*, 148:106934.
- Yousefpour, A., Patil, A., Ishigaki, G., Kim, I., Wang, X., Cankaya, H. C., Zhang, Q., Xie, W., and Jue, J. P. (2019). Qos-aware dynamic fog service provisioning.