10.48047/jocaaa.2023.31.04.67

Multi-Dimensional SLA Contracts: A Specification Language and Verification Framework Beyond Single-Number Uptime

Murali M. Chittabathini†, Saketh Gowerneni‡, Pradeep Kandepaneni§, Satya Sampreeth Boppudi¶, Dedeepya Yarra*

†Independent Researcher, USA, murali.mohan.rao@gmail.com
‡Independent Researcher, USA, gsaketh369@gmail.com
§Independent Researcher, USA, pradeepkandepaneni@gmail.com
¶Independent Researcher, USA, sampreeth.boppudi@gmail.com
*Independent Researcher, USA, dedeepyachowdary99@gmail.com

Abstract—Service-level agreements are usually stated as a single number — 99.99% availability, or p99 latency under 100 ms — but real services must satisfy several requirements at once, and those requirements interact: raising availability can weaken consistency, and lowering latency can raise cost. A one-dimensional SLA cannot express these simultaneous requirements or the trade-offs between them, so the trade-offs are made implicitly and inconsistently. This paper presents SLAspec, a formal language for multi-dimensional SLA contracts together with a framework that checks a contract for feasibility, verifies whether an infrastructure meets it, and recommends the cheapest infrastructure that satisfies it. A contract specifies requirements across dimensions such as availability, latency, consistency, and fairness, each with a target and a confidence level, plus explicit trade-offs and priorities. The contribution is the specification language, the feasibility and compliance-checking model, the trade-off-aware optimal-design formulation, and a comparative analysis against single-dimensional SLAs and manual review; empirical validation on production services is scoped as future work. All quantitative results reported in this paper are projected design targets illustrated on sample scenarios, not measured outcomes.

Index Terms—service-level agreements, SLA, SLO, formal specification, multi-objective optimisation, Pareto frontier, compliance verification, DevOps

Introduction

A service-level agreement is the contract between a service and its users, and it is almost always written as a single number: a percentage of uptime, or a latency percentile. That single number is easy to state and easy to report, but it does not describe what a real service must actually deliver. A payment service must be available and fast and consistent and fair to all users at the same time, and these are distinct requirements that a single availability figure cannot capture.

The requirements also interact, often in tension. Stronger consistency can reduce availability under partition, by the well-known trade-off between them; lower latency can raise cost, by demanding more or faster hardware; tighter fairness can reduce peak throughput. A one-dimensional SLA forces these trade-offs to be made somewhere, but because it cannot express them, they are made implicitly — in code, in configuration, in the heads of engineers — and therefore inconsistently and without review.

We argue that SLAs should be multi-dimensional and explicit. A contract should state each requirement, the confidence with which it must hold, the trade-offs the service is willing to make between dimensions, and the priorities among them, in a form precise enough to check automatically. Making the contract formal turns vague intentions into verifiable requirements and turns implicit trade-offs into stated, reviewable choices.

The main contributions of this work are: 1) SLAspec, a formal language for expressing multi-dimensional SLA contracts with targets, confidence levels, trade-offs, and priorities; 2) a feasibility and conflict checker that detects contradictory requirements before deployment; 3) a compliance-verification model that checks an infrastructure against a contract with confidence intervals; and 4) a trade-off-aware optimal-design formulation that recommends the cheapest infrastructure satisfying a contract. We make no measured claim; the empirical study is scoped as future work in Section X.

The remainder of this paper is organized as follows. Section II reviews SLAs, formal specification, and multi-objective optimisation. Section III presents the specification language. Section IV covers feasibility and conflict checking. Section V describes compliance verification. Section VI develops optimal design and the trade-off surface. Section VII gives the comparative analysis. Section VIII discusses conflicting dimensions and fairness, Section IX states limitations, Section X the evaluation plan, and Section XI concludes.

10.48047/jocaaa.2023.31.04.67

II. Background

A. SLAs, SLOs, and Error Budgets

Site-reliability practice distinguishes the externally-promised SLA from the internally-targeted SLO and manages the gap with an error budget [1]. This discipline is mature for a single dimension — availability is tracked against a budget and decisions are made when the budget is spent — but it is applied one dimension at a time, with no formal account of how availability, latency, and consistency budgets interact. SLAspec extends the same rigour to several dimensions at once.

B. Formal Specification

Formal specification languages express system requirements precisely enough to be checked mechanically, replacing prose that can be misread with notation that cannot [2]. The lesson for SLAs is that a contract written in a formal language can be checked for internal consistency and verified against an implementation, neither of which is possible for a prose SLA. SLAspec applies this idea specifically to the structure of service-level requirements.

C. Multi-Objective Optimisation

When several objectives conflict, the set of best compromises is the Pareto frontier, on which no objective can improve without another worsening [3]. Constraint solvers and integer programming can find the cheapest configuration meeting a set of requirements [4]. SLAspec uses both: the frontier expresses the trade-off space a contract permits, and integer programming finds the minimal-cost infrastructure on it that satisfies the contract.

D. SLO Tooling

A recent line of tooling lets teams declare service-level objectives as code and track them against error budgets, and emerging open specifications standardise how an SLO is expressed [11]. These tools are a clear advance over prose targets, but they remain organised around independent objectives, each tracked on its own, with no formal notion of the trade-offs and priorities that relate them. SLAspec builds on the same declarative spirit and adds the missing structure: the cross-dimension trade-offs, the joint feasibility check, and the cost-optimal design that a collection of independent SLOs cannot express. It is therefore complementary to SLO-as-code tooling, supplying the contract-level reasoning above the individual objectives those tools already track well.

III. The SLA Specification Language

Fig. 1 shows the structure of a contract. An SLAspec contract has three parts. The requirements name each dimension — availability, latency, consistency, fairness — with a metric, a target, and a confidence level, so that a requirement reads as "availability at least 99.99%, with 95% confidence." The trade-offs state which dimensions may be sacrificed for which, such as relaxing consistency to raise availability under partition. The priorities give a weight vector over the dimensions, expressing what matters most when not everything can be maximised.

The confidence level on each requirement is what makes the language honest about uncertainty. A real service does not meet a target deterministically; it meets it with some probability over a period. Stating "99.99% availability with 95% confidence" expresses both the target and the certainty with which it must hold, which a bare "99.99%" leaves implicit. This couples naturally to a probabilistic compliance check, described in Section V, that reports not just whether a target is met but how confidently.

Including trade-offs and priorities in the contract is the language's distinctive move. A prose SLA records targets but is silent on what to do when they conflict; SLAspec records the conflict resolution in advance, as part of the agreement. This means that when availability and consistency cannot both be maximised under a partition, the contract already says which to favour, so the runtime decision is a matter of following the agreed policy rather than improvising under pressure. Encoding the trade-offs turns a source of operational ambiguity into a stated term.

A concrete contract makes the structure tangible. A payment service might declare availability of at least 99.99% at 95% confidence, p99 latency under 100 ms, read consistency above 99%, and a fairness rule that no user sees more than twice the median latency; a trade-off clause permitting relaxed consistency in exchange for availability under partition; and a priority vector weighting availability above latency above consistency. Read together, these say not only what the service promises but how it behaves when the promises collide and which promise yields first. The same information, scattered across a prose SLA and the implementation, would be neither checkable nor consistent; gathered into one contract, it is both.

10.48047/jocaaa.2023.31.04.67

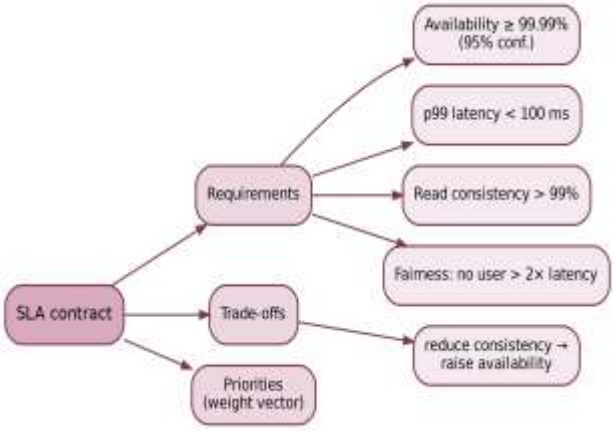

Fig. 1. Structure of an SLAspec contract: requirements with targets and confidence, trade-offs between dimensions, and priorities.

IV. Feasibility and Conflict Checking

A multi-dimensional contract can be internally contradictory in ways a single number never can — requiring, for instance, strong consistency and full availability under partition, which a fundamental result shows cannot both hold [5]. SLAspec checks a contract for such conflicts before any infrastructure is built. The checker examines the requirements for combinations that are known to be jointly unsatisfiable and reports them, so that an impossible contract is caught at authoring time rather than discovered as a chronic, unexplained violation in production.

Feasibility checking is valuable precisely because contradictory requirements are easy to write and hard to spot. Each requirement looks reasonable in isolation, and the conflict emerges only from their combination, which is exactly the kind of error a formal checker excels at and a human reviewer often misses. Catching it early saves the considerable cost of building toward a target that cannot be met and then debugging the resulting failures as if they were faults rather than impossibilities.

V. Compliance Verification

Given an infrastructure and a contract, the compliance checker measures each dimension and reports whether the requirement is met at its stated confidence. Because requirements carry confidence levels, the check is probabilistic: it estimates the probability that each target holds over the relevant period and compares it to the required confidence, producing a per-dimension verdict and an overall compliance report. A dimension that is met with lower confidence than required is flagged even if its point estimate looks acceptable, because the contract demanded a level of certainty the data does not support.

Reporting compliance per dimension, rather than as a single pass or fail, is what makes the verification actionable. A service might meet availability and latency comfortably while just missing the consistency requirement; a one-number SLA would either hide this or fail the whole service, whereas the multi-dimensional report points precisely at the dimension that needs attention. This turns SLA verification from a blunt verdict into a diagnostic that says which promise is at risk and by how much.

The per-dimension view extends the error-budget discipline to multiple dimensions. Each requirement defines a budget — the allowable shortfall before the promise is broken — and tracking these budgets separately tells a team which dimension it is spending down fastest. A service burning its latency budget while its availability budget sits untouched should invest in latency, not redundancy, and only a multi-dimensional account reveals this; a single availability budget would say nothing about the latency risk. Multi-dimensional error budgets thus turn the contract into an ongoing management tool, not merely a verification gate, by showing where reliability effort will buy the most margin against the promises that matter.

VI. Optimal Infrastructure Design

Given a feasible contract, the natural question is the cheapest infrastructure that satisfies it. SLAspec formulates this as a constrained optimisation: minimise cost subject to each requirement being met at its confidence, with the trade-offs and priorities shaping which compromises are admissible. Integer programming solves this to recommend a configuration — instance types, replication, placement — that meets the contract at minimal cost [4], turning the contract directly into an infrastructure decision.

Fig. 2 shows the trade-off surface the optimisation explores, here projected to cost against availability. Each point on the frontier is the cheapest infrastructure achieving a given availability; moving toward higher availability climbs the frontier at increasing cost, and the priorities in the contract select where on it to sit. The figure uses illustrative values to convey the shape, not measurements: the steep rise near the top is the familiar reality that each additional nine of availability costs disproportionately more than the last.

Exposing the frontier rather than a single recommendation lets stakeholders make the trade-off knowingly. Shown that moving from three nines to four

10.48047/jocaaa.2023.31.04.67

roughly multiplies cost, a team can decide whether the extra reliability is worth it for a given service, rather than discovering the cost only after committing to the target. The optimisation thus supports a conversation about what to promise, grounded in the actual cost of each promise, which is far more useful than a recommendation handed down without its price.

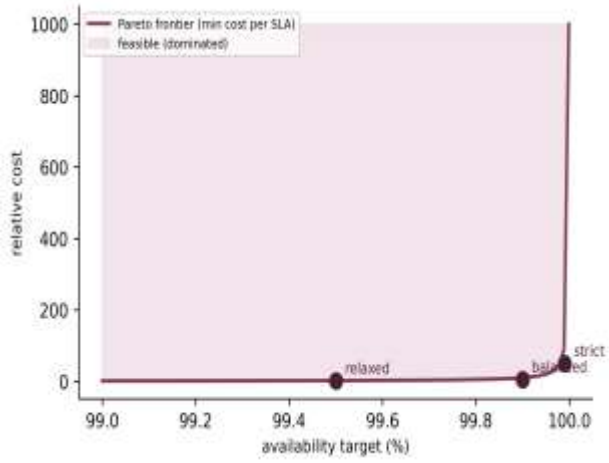

Fig. 2. Cost-versus-availability frontier. Each point is the cheapest infrastructure meeting that target; values are illustrative of the shape, not measured.

VII. Comparative Analysis

Because we make no measured claim, we compare SLAspec against the prevailing approaches analytically, across the criteria that determine whether multi-dimensional service requirements can be stated, checked, and met. Table I summarises the comparison. A single-dimensional SLA cannot express multiple requirements or their trade-offs; manual multi-dimensional review can express them but cannot check feasibility or compliance mechanically; only a formal language supports automatic feasibility checking, probabilistic compliance, and cost-optimal design across dimensions.

VIII. Discussion

Conflicting dimensions are not a flaw to be eliminated but a reality to be expressed. The value of stating trade-offs in the contract is that the conflict is resolved deliberately and in advance, rather than by whoever happens to be on call when a partition forces the choice. A contract that says, ahead of time, to favour availability over consistency under partition gives the runtime an unambiguous policy and gives the organisation a record of a decision it actually made, instead of one that emerged by accident from the implementation.

Fairness deserves particular attention as a dimension because it is rarely stated and easily violated. A service can hit its aggregate latency target while delivering far worse latency to a subset of users, so an average-based SLA hides unfairness that the affected users experience acutely. Making fairness an explicit dimension — no user receives more than twice the median latency, say — forces it to be measured and met rather than averaged away, which is only possible once the SLA can carry more than one dimension. This connects multi-dimensional SLAs to the broader concern of equitable service that single-number agreements structurally cannot address.

Finally, a formal multi-dimensional contract changes how SLAs are negotiated. Today a customer and a provider agree on a headline number and leave the rest to interpretation; with SLAspec they can negotiate over an explicit set of dimensions, see the cost of each target on the frontier, and agree on the trade-offs in writing. The contract becomes a shared, checkable artefact rather than a marketing figure, and both sides can verify compliance against the same definition. This shifts the SLA from a promise that is argued about after a breach to a specification that is agreed and monitored from the start, which is ultimately the point of making it formal.

IX. Limitations and Threats to Validity

The language is only as good as the dimensions it can express, and some service qualities resist precise specification; a contract can capture availability and latency cleanly but may struggle with softer notions of quality. This paper contributes a language and framework, not measurements, so no claim about verification accuracy or achieved cost savings is made. With respect to construct validity, the probabilistic compliance check depends on statistical assumptions about the measured data that a deployment must validate. Externally, the cost-optimal design depends on a cost model of the available infrastructure that varies by provider and must be supplied.

X. Evaluation Plan (Future Work)

A sound empirical study would express the real SLAs of several production services in SLAspec, run the feasibility checker to find any latent contradictions, verify the running infrastructure against the contracts and compare the per-dimension verdicts against ground-truth incidents, and apply the optimal-design formulation to measure whether the recommended configurations meet the contracts at lower cost than the deployed ones. A usability study would assess whether engineers can

10.48047/jocaaa.2023.31.04.67

express their intended SLAs in the language accurately. We leave this study to follow-up work, noting that eliciting the true multi-dimensional requirements of a service — including the trade-offs teams make tacitly — is itself part of the research.

XI. Conclusion

We presented SLAspec, a formal language for multi-dimensional SLA contracts and a framework that checks them for feasibility, verifies infrastructure against them with confidence, and recommends the cheapest infrastructure that satisfies them. Through a comparative analysis we showed that its contribution is expressing and satisfying several interacting requirements and their trade-offs together, which single-number SLAs cannot represent and manual review cannot verify mechanically. The natural next step is empirical: expressing real service SLAs in the language, checking and verifying them against production, and measuring the cost of optimal designs, so that the trade-offs services make implicitly today become explicit, reviewable terms.

XII. References

[1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly, 2016.

[2] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA, USA: Addison-Wesley, 2002.

[3] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester, U.K.: Wiley, 2001.

[4] L. Perron and V. Furnon, "OR-Tools," Google, 2023. [Online]. Available: https://developers.google.com/optimization/

[5] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[6] A. Keller and H. Ludwig, "The WSLA framework: specifying and monitoring service level agreements for web services," *J. Netw. Syst. Manag.*, vol. 11, no. 1, pp. 57–81, 2003.

[7] N. R. Murphy, B. Beyer, D. Rensin, K. Nukala, and S. Thorne, Eds., *The Site Reliability Workbook*. Sebastopol, CA, USA: O'Reilly, 2018.

[8] P. Bailis and K. Kingsbury, "The network is reliable," *Commun. ACM*, vol. 57, no. 9, pp. 48–55, 2014.

[9] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, 2012.

[10] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[11] Nobl9, "OpenSLO: a specification for defining service level objectives," 2023. [Online]. Available: https://openslo.com/

[12] H. Ludwig et al., "Web service level agreement (WSLA) language specification," IBM, Tech. Rep., 2003.

[13] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.

[14] L. Leite et al., "A survey of DevOps concepts and challenges," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–35, 2019.