Automated Curation of Variational Data in NoSQL Databases through Metric-driven Self-adaptive Migration Strategies

Andrea Hillenbrand¹² and Uta Störl¹²

¹Darmstadt University of Applied Sciences, Darmstadt, Germany
²University of Hagen, Hagen, Germany

Keywords: Databases, NoSQL, Data Migration, Schema Evolution, Schema Migration, Automated Curation.

Abstract: Schema-flexible NoSQL databases have become popular backends in agile development. They allow developers to write code flexibly while assuming a new database schema different from the current one. The co-evolution of the schema with the software code, together with requirements for performance and cost efficiency, require subtle management decisions regarding the migration of variational legacy data persisted in the production database. Project managers have to deal with the consequences of schema evolution in order to comply with service-level agreements, especially if metrics specified in the SLAs compete with each other in tradeoffs. We present self-adaptive schema migration strategies that curate just as much variational data so that competing metrics can be balanced out, thus making continuous management interventions superfluous.

1 INTRODUCTION

Schema-flexible NoSQL databases allow developers to write code assuming a new schema different from the current one. Then, new software releases can be deployed without application downtime for the schema migration. Though, eventually it has to be addressed how to handle the variational data that is already persisted in the production database. Managing the repercussions of schema evolution comes up repeatedly, oftentimes in unison with changing requirements from service-level agreements (SLAs). The most crucial issue is an oftentimes overwhelming complexity of the schema evolution problem, i.e., predicting the impact of schema changes on the metrics that are required by SLAs despite the multitude of influencing factors. Due to this complexity, there seem only two approaches pragmatic in general.

First, a heuristic allows an adequate solution in form of an approximation considering the most important migration situation characteristics that influence the impact (Hillenbrand et al., 2021a). However, although the existence of such a heuristic saves stakeholders time, evaluating the metrics and applying the heuristic repeatedly is still time-consuming.

Especially at the beginning of a software project, just when stakeholders are most busy and presumably workflows still inefficient, reorganizing a database schema is often necessary due to emerging requirements. We have identified schema modification operations affecting multiple types as the most definite cost driver of schema evolution (Hillenbrand et al., 2021a). Moreover, we have found in our probabilistic experiments that metrics can vary easily by factor 100 in case of high cardinalities of the entity-relationships, sometimes resulting in exceptionally high migration costs and tail latencies. The uncertainty gets worse if the migration situation is unknown or scenarios hard to predict.

Contribution. We contribute an approach that overcomes the necessity of navigating a heuristic in order to deal with the impact of schema evolution. We present the automated curation of variational data in NoSQL databases through self-adaptive migration that balances out competing metrics automatically, thus making stakeholder interventions and elicitation of migration situations superfluous.

The Bigger Picture. Migration strategies vary with respect to how much and when they migrate legacy data that is structured according to earlier schema versions. In an eager approach, all of the legacy data is curated right away at the release of schema changes, which produces maximal charges with the cloud provider. The upside of this investment is that a structurally homogeneous database instance does not...
produce migration-induced runtime overhead, which is especially crucial for the performance of cloud-hosted applications (Barker et al., 2012; Curino et al., 2011; Difallah et al., 2013). However, if saving costs is defined being most important in the SLAs, then a lazy strategy minimizes migration costs, as data remains unchanged in the event of schema changes. The downside is that a considerable runtime overhead is introduced (Klettke et al., 2016; Saur et al., 2016). The metrics migration costs and latency compete in a tradeoff, schematically depicted in Figure 1, which opens up the possibility of alternative strategies at different opportunity costs.

In (Hillenbrand et al., 2021a), we have investigated the impact of schema evolution in terms of common migration strategies and software release strategies. We uncovered the correlations of migration situation characteristics with the impact of schema evolution for each strategy by searching the solution space with a probabilistic Monte Carlo method of repeated sampling of our schema migration advisor tool MigCast in order to bring the complexity of the problem under control (Fishman, 2013). A cost model in MigCast takes all relevant migration situation characteristics into account as detailed in Section 2. Then we could distill a heuristic by means of which stakeholders can mitigate the impact of schema evolution and pace releasing schema changes in order to ascertain compliance with cost- and latency-related SLAs (Hillenbrand et al., 2021a).

In order to put stakeholders in a position of directly controlling the impact of schema evolution, we present a self-adaptive migration strategy that can simply be parameterized with the required thresholds of the competing metrics to balance them out. We have integrated it into our schema migration advisor tool MigCast to demonstrate the effectiveness in the context of different migration situations. We have already analyzed possible options of self-adaptation from a theoretical stance in (Hillenbrand et al., 2020; Hillenbrand et al., 2021b). In this paper, we flesh out the methodology of self-adaptive migration by specifying and validating the most promising algorithm.

2 ARCHITECTURE

Definitions and Terminology. In this paper, we use the following metrics: We refer to the time that a read access takes, i.e. the time to retrieve a requested data entity, as data access latency. Latency competes in a tradeoff with the monetary charges occasioned by migrating the data with a cloud service provider, which we refer to as migration costs. The migration costs consist of on-release and on-read migration costs, which together make up the cumulated migration costs. On-release migration costs are caused when entities are migrated in the event of a schema modification operation (SMO). They depend on the number of affected entities and how these legacy entities are being handled. In contrast, on-read migration costs are caused when entities are accessed that exist in older versions than the current schema.

Architecture of MigCast. MigCast is our NoSQL schema migration advisor tool, which we presented in earlier work (Hillenbrand et al., 2019; Hillenbrand et al., 2020). For each migration strategy, MigCast generates the migration scenarios according to the parameterized characteristics and calculates various cost and classification metrics by means of the schema management middleware Darwin (Störler et al., 2018). Specifically, MigCast takes all metric-relevant situation characteristics into account: i. intensity and distribution of data entity accesses, ii. different kinds of schema changes, and iii. cardinality of the entity-relationships of the data set, and calculates the monetary migration charges and the data access latency, as well as classification metrics like precision and recall in order to assess the structural heterogeneity of the database instance, while persisting all data in the MigCast database. For more details, be referred to (Hillenbrand et al., 2021b; Hillenbrand et al., 2021a).

3 MIGRATION STRATEGIES

As illustrated in Figure 1, the migration strategies eager and lazy span the space of opportunity costs on the metrics. Different compromises on this tradeoff can be settled by means of alternative strategies.

A common compromise is the incremental strategy which fluctuates between these extremes at the disadvantage of an unstable, yet not unpredictable, latency. With the incremental strategy, schema changes are usually treated like with lazy migration, the disarray in the database increasing accordingly. However, with the incremental strategy, legacy entities are completely migrated at certain periodic increments to
match the current schema and thus, getting rid of the runtime overhead which is caused by updating legacy entities on-the-fly when being accessed. In terms of a heuristic, it is advisable to apply the incremental strategy in case that the entity accesses to be served, the workload, is evenly distributed.

If the workload concentrates on hot data, i.e., is Pareto-distributed, a predictive strategy settles on a compromise with a more stable latency and better cost-benefit ratio of invested migration costs for improved latency. The predictive strategy is controlled through adapting the cardinality of the set of entities to be migrated after an SMO, the prediction set size. In order to keep a steady balance between metrics, we devised the predictive strategy which keeps hot entities in a prediction set and orders them according to their actuality and access frequency (Hillenbrand et al., 2021b). Legacy entities not included in the prediction set, are migrated lazily when being accessed.

Through the complexity-adaptive strategy we avoid tail latencies which would occur when legacy entities are accessed that have been affected by a sequence of SMOs, typically during major schema changes at the beginning of a software development life-cycle. This backlog of complex schema changes can be prevented by increasing the prediction set size automatically when a certain number of multi-type SMOs have accrued (Hillenbrand et al., 2021b).

In the remainder of this section, we newly devise the algorithm of the requirement-adaptive migration strategy in Algorithm 1 that automatically adapts the prediction set size (psSize). This strategy is parameterized by the thresholds of migration costs ($c_{\text{max}}$) and latency ($l_{\text{max}}$) as inputs. The metrics ($c$, $l$) are monitored and the prediction set size is increased or decreased accordingly in order for both metrics to comply with their thresholds, or find a suitable compromise. The monitored migration costs consist of measurements of the on-read migration costs caused by accessing legacy entities when serving the workload in between two schema changes and the on-release migration costs caused by the last schema change.

A balanced compromise can be reached through an average of the measured metrics in respect of their maximal thresholds, i.e., $l_{\text{max}}$ and $c_{\text{max}}$.

In Figure 2 A, tuple $(c_1, l_1)$ represents the measured metrics since the past release with its auxiliary line to indicate the used resources. Analogously, the red tuple $(c_{\text{max}}, l_{\text{max}})$ marks the thresholds of the metrics to indicate the available resources. The red tuple can be connected to the origin, the red dotted line, and can be described by the function $c = \frac{l}{l_{\text{max}}}$ which expresses the compromise between the metrics. Consequently, all three averages in green ($F_1$ for harmonic, $G$ for geometric, and $\sigma$ for arithmetic means) of the quotients of measured metrics and required thresholds can be located on this red dotted line, with the harmonic mean $F_1$ as the furthest and the arithmetic mean $\sigma$ as the compromise closest to the red tuple.

The intersections of the green lines with the x-axis are a geometric solution to determine how to adapt the prediction set size. These represent the predicted costs that are caused if the quotients of the measured-to-required metrics are assumed to be equally high for both migration costs and latency, depending on their correlation being arithmetic, geometric, or harmonic. Naturally, the green lines all turn out to be parallel to the red line, the available resources. This is an interesting way of picturing the metrics being balanced out: the blue line can be rotated at certain angles to be identical to the green lines. Not by coincidence the angle, at which the blue line is rotated to become the harmonic mean, lie on the red dotted line, and the angle, at which it becomes the arithmetic mean, equals half of the measured metrics ($\frac{c_1}{2}, \frac{l_1}{2}$).
Algorithm 1: Requirement-adaptive strategy adapts the prediction set size at each release based on metrics migration costs and latency and their arithmetic mean.

**Result:** Updates prediction set size psSize fulfilling given requirements regarding the maximum migration costs \( c_{\text{max}} \) and maximum latency \( l_{\text{max}} \) and regarding the prioritization of these thresholds \( \text{prio} \).

**Input:** psSize, \( c_{\text{max}} \), \( l_{\text{max}} \), \( \text{prio} \)

**Output:** psSize

```plaintext
1  \( q_c \leftarrow \frac{c_{\text{max}}}{c_1} \);
2  \( q_l \leftarrow \frac{l_{\text{max}}}{l_1} \);
3  \( a \leftarrow \frac{1}{2}(q_c + q_l) \);
4  \( m_c \leftarrow a \cdot c_{\text{max}} \);
5  \( m_l \leftarrow a \cdot l_{\text{max}} \);
6  \( \text{if } c_{\text{max}} \neq 0 \wedge l_{\text{max}} = 0 \text{ then} \)
7      psSize \leftarrow \text{psSize} \cdot q_c^{-1} ;
8  \text{else if } c_{\text{max}} = 0 \wedge l_{\text{max}} \neq 0 \text{ then} \)
9      \( \text{if } 2l_{\text{max}} \leq l \text{ then} \)
10         psSize \leftarrow 1 ;
11     \text{else } /\ast 2l_{\text{max}} > l \text{ */} \)
12         psSize \leftarrow \text{psSize} \cdot \frac{l_{\text{max}}}{2l_{\text{max}}-1} ;
13 \text{end}
14 \text{else } /\ast c_{\text{max}}, l_{\text{max}} \neq 0, \neg (\text{prio}(c_{\text{max}}) \wedge \text{prio}(l_{\text{max}})) \)
15 \text{if } \text{prio} (c_{\text{max}}) \text{ then} \)
16     \text{if } m_c > c_{\text{max}} \text{ then} \)
17         psSize \leftarrow \text{psSize} \cdot q_c^{-1} ;
18     \text{else } /\ast m_c \leq c_{\text{max}} \text{ */} \)
19         psSize \leftarrow \text{psSize} \cdot a \cdot q_c^{-1} ;
20 \text{end}
21 \text{else if } \text{prio}(l_{\text{max}}) \text{ then} \)
22     \text{if } m_l > l_{\text{max}} \text{ then} \)
23         psSize \leftarrow 1 ;
24     \text{else } /\ast 2l_{\text{max}} > l \text{ */} \)
25         psSize \leftarrow \text{psSize} \cdot \frac{l_{\text{max}}}{2l_{\text{max}}-1} ;
26 \text{end}
27 \text{else } /\ast m_l \leq l_{\text{max}} \text{ */} \)
28         psSize \leftarrow \text{psSize} \cdot a \cdot q_c^{-1} ;
29 \text{end}
30 \text{else } /\ast \neg \text{prio} (c_{\text{max}}) \wedge \neg \text{prio} (l_{\text{max}}) \text{ */} \)
31         psSize \leftarrow \text{psSize} \cdot a \cdot q_c^{-1} ;
32 \text{end}
33 \text{end}
34 \text{end}
35 \text{return psSize}
```

1. **Line 9/10:** If the measured latency is at least twice the amount of its threshold, then the situation de-
picted in Figure 3 B shows that the highest increase of the psSize is required (100%), which is tantamount to an eager migration of all legacy entities. This is the case, when we assume an arithmetic correlation of the quotients, where the projected migration costs diverge if the latency is much higher than required by the threshold.

2. Line 11/12: If the measured latency stays below twice the amount of the threshold, then the migration costs can be projected as shown in Figure 4 A by means of the red line which is a straight line from $l_{\text{max}}$ intersecting the center of the blue rectangle (used resources) to the intersection with the x-axis $c_{\text{proj}}$ as defined by $\frac{c_{\text{max}}}{2l_{\text{max}} - l_1}$. 

Line 14: If, however, there are two thresholds migration costs and latency, and at most one metric is prioritized no matter the consequences to the other metric, then three mutually exclusive cases have to be distinguished:

1. Line 15: If the threshold for cost is mandatory, then it is distinguished whether the compromise with regard to the costs $m_c$ is greater than the maximal costs $c_{\text{max}}$ or not.

(a) Line 16/17 $m_c > c_{\text{max}}$: If $m_c$ exceeds the maximal costs $c_{\text{max}}$, then the psSize is multiplied by the inverse migration costs quotient $q_-^{-1}$, scaling down the psSize to $c_{\text{max}}$ (see Figure 4 B).

(b) Line 18/19 $m_c \leq c_{\text{max}}$: If $m_c$ is at most $c_{\text{max}}$, then the psSize is adapted by the compromise $a$ and the inverse migration costs quotient $q_-^{-1}$. This is illustrated in Figure 2 A. It balances out the metrics to become equal quotients.

2. Line 21: If the threshold for latency is mandatory, then it is distinguished whether the compromise with regard to the latency $m_l$ is greater than the maximal latency $l_{\text{max}}$ or not.

(a) Line 22 $m_l > l_{\text{max}}$: If $m_l$ exceeds the maximal $l_{\text{max}}$ as shown in both Figures 5 A and B, then two mutually exclusive cases have to be distinguished:

i. Line 23/24 $2l_{\text{max}} \leq l$: If the measured latency surpasses twice the amount of the threshold, see Figure 5 A, then the highest increase of the psSize is required, i.e., 100%, which is tantamount to an eager migration.

ii. Line 25/26 $2l_{\text{max}} > l$: If the measured latency stays below, see Figure 5 B, then the migration costs can be projected by means of the green line which is a straight line from $l_{\text{max}}$ intersecting the center of the blue rectangle to the abscissa $c_{\text{proj}}$ defined by $\frac{c_{\text{max}}}{2l_{\text{max}} - l_1}$.

(b) Line 28/29 $m_l \leq l_{\text{max}}$: If the compromise with regard to the latency $m_l$ is at most $l_{\text{max}}$, as illustrated in Figure 2 A, then the psSize is adapted by the compromise $a$ and the inverse migration costs quotient $q_-^{-1}$, like in Line 19.

3. Line 31/32: If neither threshold is mandatory, then the psSize is adapted by the compromise $a$ and the inverse quotient $q_-^{-1}$ to balance out the metrics and become equal quotients, like in Lines 19/29. This is illustrated in Figure 2 depending on the metrics surpassing the thresholds (B) or not (A).

4 IMPLEMENTATION

We integrated the requirement-adaptive strategy into MigCast to demonstrate its effectiveness in terms of controlling the impact of schema evolution. We calculated the migration costs and latency for all strate-
The **requirement-adaptive** strategy stays consistently below the **predictive** and **complexity-adaptive** strategies with respect to on-release and cumulated migration costs. The threshold for the latency is kept in roughly half on the releases, which is a good balance as the metrics are set as equally important.

At release 4, latency is significantly higher than the threshold and thus the prediction set size is increased and the on-release migration increase as well, yet not at the same measure as the latency threshold is surpassed. The increase is relatively moderate, because the sum of on-release cost of release 3 and on-read costs of release 4 have also already surpassed the threshold of USD20. At release 6, latency also surpasses its threshold and now the prediction set size is decreased and on-release cost shrink, because the on-read costs have already exhausted the budget. At releases 11 and 12, latency peaks again, which is also visible in high on-read migration costs, such that the on-release costs are dialed down accordingly.

Altogether, throughout the releases, the **requirement-adaptive** strategy has stable on-release migration costs and acceptable latency at relatively low cumulated costs. Thus, depending on the SLAs, stakeholders are well advised to choose the **requirement-adaptive** strategy if they want to decide for a compromise between the competing metrics, avoid budget peaks like with the incremental strategy, avoid constantly high costs with the eager strategy, and avoid constantly high latency with the lazy strategy. With the **requirement-adaptive** strategy stakeholders can reckon with very stable on-release migration costs and relatively low cumulated charges. We argue that once cost transparency is established and the option of self-adaptive strategies exists for stakeholders, their preferences regarding the tradeoffs will converge to the best compromise with regard to the SLAs, which would have been considered too risky without automation.

### 5 RELATED WORK

**Schema Evolution.** Frameworks managing schema changes in relational databases have been studied in (Aulbach et al., 2009; Cleve et al., 2015; Curino et al., 2013; Herrmann et al., 2017). Schema evolution in XML has been investigated in (Bertino et al., 2009; Curino et al., 2013; Herrmann et al., 2017). Schema-facile NoSQL database systems have been researched sparsely. The
schema is declared implicitly within the application code, such that schema changes can be observed by analyzing the code (Meurice and Cleve, 2017; Scherzinger and Sidortschuck, 2020). There is evidence that the NoSQL schema evolves more continuously throughout the project than with relational databases (Scherzinger and Sidortschuck, 2020). A study revealed that projects undergo different volumes of schema changes (Vassiliadis, 2021).

**Data Migration.** (Ellison et al., 2018) investigates costs and duration of migrating entire databases to the cloud. When stakeholder consider a database to be hosted in a cloud, concerns about high costs are most often cited, tail latency being cited as an issue as well (3T Software Labs Ltd., 2020). Because **eager** migration causes considerable costs in cloud-hosted settings (Ellison et al., 2018; Hillenbrand et al., 2021a), other migration strategies such as **lazy** (Klettke et al., 2016; Saur et al., 2016) and **proactive** (Hillenbrand et al., 2021b) strategies have been proposed.

**Self-adaptation.** A recommender system is presented in (Mior et al., 2017) that maps the application’s conceptual data model to a NoSQL schema. Ongoing research focuses on providing automated schema optimization between different NoSQL data stores (Conrad et al., 2021). In (Preuveneers and Joosen, 2020), tuning deployment parameters at runtime for scalability and performance of data-intensive applications is addressed for NoSQL systems. A survey on parameter-tuning approaches for SQL-on-Hadoop systems examined throughput and resource utilization, response time, and cost-effectiveness (Filho et al., 2021). Cloud-enabled frameworks to perform automatic resizing of NoSQL clusters were presented in (Tsoumakos et al., 2013).

### 6 CONCLUSION

We presented the automated curation of variational data in NoSQL databases through a metric-driven self-adaptive migration strategy, which are independent from stakeholder intervention and from concrete situation characteristics. By means of a self-adaptive migration strategy, we equip stakeholders with an option to control schema migration automatically. Thereby, stakeholders can replace navigating a time-consuming heuristic and moderate the impact on SLA-derived metrics when reorganizing the database, even if migration situation characteristics are unknown or hard to predict.
ACKNOWLEDGEMENTS

This work has been funded by the German Research Foundation (#385808805). We thank Jan-Christopher Mair, Kai Pehns, Tobias Kreiter, Shamil Nabiyev, and Maksym Levchenko (Darmstadt University of Applied Sciences) for their contributions to MigCast.

REFERENCES


