Towards Not Re-Inventing the Wheel: Managing Data Management Tools

Kathryn Dahlgren (June 2020)
Supervised by Peter Alvaro
Dept. Of Computer Science, University of California, Santa Cruz
1156 High St, Santa Cruz, CA 95064, U.S.A.
kmdahlgr@ucsc.edu

Abstract—NoSQL databases offer powerful abstractions for querying non-relational data. However, NoSQL products generally pursue superior flexibility, customizability, scalability, and performance goals while neglecting support for generally useful data management tools. In particular, products typically ship without integrated support for management features rendered conventional by the long history of RDBMSs, such as sophisticated query processing systems, join operations, aggregate functions, and integrity constraints. The design decision forces users of NoSQL technologies to find alternative methods for providing missing tools by engaging either directly or indirectly in a suboptimal design-implementation cycle as developers re-invent new instances of the same data management tools across NoSQL products. This paper articulates the problem associated with the lax regard for data management support currently defining the class of NoSQL databases and introduces the Piper package index and management system as an exploratory solution.

I. INTRODUCTION

The final decades of the twentieth century marked the inception, rise, and evolution of relational database management systems as products embodying some of the most appealing philosophies and techniques for interacting with data. RDBMSs offered powerful tools for viewing and manipulating data from a well-defined high-level perspective grounded in solid theory. The software systems also explored the highest standards for guaranteed correctness and performance goals using a wide variety of sophisticated tools resulting from decades of intense research in both the theoretical and practical limits of data storage, retrieval, and manipulation. Over time, the concepts and ideas fundamental to RDBMSs blossomed into a range of products offering a wide variety of features designed to do anything and everything a user may possibly need in the pursuit of a data management task. Accordingly, on the cusp of twenty-first century, databases encompassing a sizable suite of services, asserting strong correctness guarantees, and aligning inseparably with the relational model represented both the expectation and the norm. However, in the early 2000s, a number of heretics in the community rejected such an arguably limited vision of the world to pose existential questions regarding the deepest definitions of databases and data management and to propose new tools vindicated by time as steps toward a revolutionary arch in the history of Databases as a field of study.

The NoSQL movement resulted in tools rejecting different subsets of the more restricting characteristics of RDBMSs to acknowledge the heterogeneity of data collection, storage, interfacing, and manipulation needs. NoSQL products offered a perspective on databases emphasizing highly specialized and malleable designs streamlined for the idiosyncrasies and performance goals of targeted applications and resigned to the sacrifice of certain correctness guarantees to embody a philosophy fundamentally orthogonal to the basic principles of RDBMSs. More than 200 different NoSQL products emerged into existence since the early 2000s [1], each touting unique specializations for certain data models, workloads, system architecture compatibilities, scalability thresholds, performance ranges, and correctness assurances.

Flexibility and customizability represent the two most defining characteristics of NoSQL as a paradigmatic alternative to traditional RDBMSs. The space of NoSQL products demonstrates a comparably vast capacity for flexibility with respect to data models and implementation methods. Users enjoy a significant amount of freedom when choosing how to organize, access, and think about the data under consideration and have the power to infuse those considerations with arbitrary levels of simplicity or complexity. NoSQL products are also extremely customizable, meaning the resulting software may contain all and only the tools absolutely necessary for the creation of the most streamlined and performant incarnation of a particular system, with minimal or no support for extraneous features.

The abstract nature of the two major properties of NoSQL creates a number of problems. In particular, the properties do not inform rigid standards for data management functionality across products. The consequential systematic neglect of automatic support for suites of valuable and commonly needed management tools renders raw NoSQL products fundamentally classifiable as ‘database systems’, rather than ‘database management systems’. In response, users either implement the data management tools themselves; integrate the data management tools from third-party extensions written for specific NoSQL products; or wait until enough people complain about the missing feature(s) to pressure developers into finally adding the management tool(s) in a product version.
Accordingly, the NoSQL paradigm summarily thrusts the world into a k-implementation spiral in which users intent on transforming NoSQL products into database management systems engage in the endless re-implementation of the same management features across NoSQL products. The k-implementation suboptimality is obvious in the case of custom implementations because rebuilding tools such as AVG or COUNT for every NoSQL product is a cringingly clear deviation from good software reusability practices. The third-party approach is similarly suboptimal because people still write the same management tools per NoSQL product. Finally, the ‘pressuring the developers’ strategy is also k-implementation suboptimal by essentially manifesting the third-party approach pursued by the product developers.

A solution to the k-implementation of database management functionality problem would ideally minimize k while also minimizing compromises on the best qualities of the NoSQL paradigm. Specifically, the solution should not impede the target system’s flexibility with respect to data models or implementation options because handling unprincipled or complicated data with potentially unprincipled and complicated tools is one of the major appeals of the paradigm. Also, the solution should not jeopardize the customizability of the target system’s features because chasing designs streamlined for the goals and idiosyncrasies of particular applications is also one of the major appeals of the paradigm. An additional solution property unrelated to defending against compromises on the best qualities of NoSQL systems is ensuring easy dissemination of management tools across NoSQL products. Such a property is critical because the k-implementation problem is, fundamentally, a k-minimization problem, where k maps to the number of different NoSQL products in the world, which maps to a set of humans, and, as a general rule, humans do not enjoy working with inconvenient tools. The Piper package index and management system explores a framework for supporting such a list of solution constraints.

Piper [2] is an open source, freely available, package index and management system exclusively dedicated to hosting database management tools for NoSQL products. The package implementations of data management tools are as general as possible and only encode the high-level logic necessary for performing the associated data management tasks. Accordingly, the designs of package implementations should be general enough to work across different NoSQL products as long as the inputs from the underlying NoSQL database systems correspond to, or can be transformed into, some necessary format and the performance of such input correspondences and transformations occurs automatically within the Piper framework beyond the scope of user concerns. The Piper framework is responsible for using configuration information to automatically choose the data access methods appropriate for the underlying NoSQL system. As a result, the only product specializations appearing in Piper manifest in the adapter programs encoding the access logic per NoSQL product to describe mappings from raw input to appropriate general formats required by management tool implementations.

NoSQL products pursue superior flexibility, customizability, and performance goals while neglecting support for generally useful data management tools rendered conventional by the long history of RDBMSs, such as sophisticated query processing systems, join operations, aggregate functions, and integrity constraints. The NoSQL technology explosion echoing through the field since the early 2000s thrust the world into a viciously suboptimal k-implementation cycle forcing users to continuously re-invent solutions for commonly valuable data management tools across NoSQL products. While the resulting systems are highly streamlined and optimized for target applications, such a lack of built-in support for basic data management services defies the abstraction philosophy central to principled software design and ultimately increases the amount of work required to transform a NoSQL database system into a database management system. This paper articulates the problem associated with the lax regard for data management support currently defining NoSQL databases and proposes the Piper package index and management system as a valid solution according with several necessary and sufficient constraints.

**Problem Statement:** The k-implementation of database management functionality across the total set of instances of NoSQL products is suboptimal. Minimize k while avoiding compromises on the best qualities of the NoSQL paradigm. Specifically, develop a solution which:

1. Does not impede the flexibility of underlying NoSQL systems because using potentially unprincipled and complicated methods to handle unprincipled or complicated data represents one of the major paradigmatic appeals, and
2. Does not jeopardize the customizability of underlying NoSQL systems because developing highly streamlined designs also represents a major paradigmatic appeal.

The solution should additionally not render the dissemination of management tools across instances of NoSQL products inconvenient because k, in this context, represents the magnitude of a set of humans and humans generally dislike inconvenience.

**Current Contributions:** The current state of the Piper package index and management system is best described as ‘early’. At the moment, Piper is slightly more than a bare-bones proof-of-concept implementation and boasts an index of five different generalized implementations of data management tools representing varying degrees of common applicability and specialization. The index also supports two different NoSQL products, MongoDB and PickleDB, representative of two different NoSQL data models, a document store and a key-value store, respectively. Accordingly, the current version of Piper offers a contribution to the field by proving the existence of a principled framework for building data management tools such that the single implementations of those data management tools generalize across different NoSQL products aligning with different data models.

**Anticipated Contributions:** The ultimate vision for Piper is to provide a software collection and distribution service for
open-source implementations of data management tools capable of regarding database systems as black-box components to the same degree as other popular package management systems, such as MacPorts, HomeBrew, and PyPI, regard underlying system and architecture characteristics. Piper removes concerns regarding the inner workings of NoSQL database systems, which may vary widely and evolve over time with respect to implementations, data models, and built-in data management capabilities, to provide users with immediately runnable data management tools. The approach is reminiscent of the Foreign Data Wrapper technology used to glue PostgreSQL over arbitrary supported NoSQL systems [3]. However, Piper offers a far more flexible platform for building data management procedures outside of the SQL standard in a design decision echoing some of the core motivations for the inception of the NoSQL movement. Ultimately, the success of such an endeavor promises to rescue the world from an impending future in which the growing space of definitively diverse, highly heterogeneous, and stunningly specialized NoSQL systems force the endless re-implementation of valuable and commonly needed data management tools.

II. METHODS

The Piper package index and management system combines the principles and practices established by existing package indexes and management systems with the community-oriented and user-driven philosophies of open source software repositories to provide an ideal platform for building a catalog of data management tools generalizable across NoSQL systems. Developers from around the world can contribute data management tools to Piper by encapsulating the generalized logic associated with each application into Piper packages and, if necessary, providing adapters for new NoSQL systems describing mappings between package input schemes and the data access methods provided by the NoSQL product. Users pull all and only desired management functionality from the index and the functionality is immediately runnable on supported NoSQL products. Participation in the index requires management tool packages be as self-contained and independent from any NoSQL product as possible such that the functionality is available for use at any appropriate point during the course of a system run. Furthermore, installation and deinstallation processes compress into one-liners requiring little to no additional input. Finally, the framework requires package deinstallation processes impose absolutely no negative side effects on the destination environment, other than the loss of any capabilities provided by the de-installed Piper package.

Piper has a simple architectural design. As illustrated in Figure 1, Piper encompasses two major classes of software, adapters and packages. Adapters encode the data access methods provided by individual NoSQL products. Developers interested in utilizing Piper data management tools on an unsupported NoSQL product must contribute programs detailing appropriate data access methods and data format transformations to the suite of Piper adapters. Packages are implementations of data management tools general enough to function over any supported NoSQL product, regardless of unique underlying characteristics, such as implementation or capability idiosyncrasies or data models. Packages rely upon a standardized set of API calls to obtain data from the underlying NoSQL system. Piper utilizes information regarding the type of NoSQL system under consideration to automatically map the API calls to the logic provided in the corresponding adapter. By definition and design, Piper packages solely augment the data management capabilities of targeted instances of NoSQL systems.

To illustrate, suppose the user in Figure 2 is developing an application called SuperApp over MongoDB and realizes she wants to track the why-provenance of her input queries. She accordingly uses Piper to download the YProv package, along with the Quest dependency. Since the MongoDB adapter already provided in Piper details all the access logic required to run YProv and Quest over MongoDB, all the capabilities of the YProv and Quest tools are immediately runnable on the user’s instance of the MongoDB NoSQL database system within the context of her SuperApp. No further input required.

A. Adapters and Packages

The generalizability of data management tool implementations hosted by Piper depends upon the number and sophistication of adapters contributed to the Piper framework. As illustrated in the SuperApp example, adapters encompass intermediate code situated as a layer between the generalized data management tool implementations and the underlying instances of NoSQL products. The adapters encode exactly the set of data access methods provided by the underlying NoSQL systems.
products and required by the generalized management tools. At the moment, Piper contains adapters for both the MongoDB and PickleDB NoSQL database systems. The adapter implementations encode the most basic data access methods available for each product to allow the construction of data management packages runnable on arbitrary instances of either system.

Within the context of Piper, ‘packages’ represent implementations of data management tools completely self-contained and independent from the underlying environment and conceptually insulated from the characteristics, idiosyncrasies, and data models of underlying NoSQL systems. The Piper package index currently contains five packages representing different degrees of common value and specialization characteristics. All packages work on all NoSQL products supported by Piper, as demonstrated in unit tests accompanying the implementations of each package.

1) AggsPack: The AggsPack package contains a collection of generalized implementations of the five basic aggregate functions: AVG, SUM, COUNT, MIN, and MAX. Users specify a field of interest for the desired aggregation procedure and have the option of specifying at most one conditioning predicate per aggregate query. Observe the more recent incorporation of Quest and the aggregation support provided in C4 Overlog arguably overrides the utility of AggsPack at the cost of adhering to strict assumptions regarding data model interpretations. However, despite the questionable utility, the simplicity of the AggsPack implementation serves as an excellent template for designing new data management packages generalizable across NoSQL products.

2) SimpleJoin: SimpleJoin encodes the high-level logic for a sub-optimal, but generalizable implementation of a natural join. Users identify a single attribute as the join target and have the option of specifying at most one conditioning predicate. The workflow utilizes the data access methods provided in the adapters of underlying NoSQL systems to materialize all data relevant to the join attribute prior to applying any filtration logic and executing the join operation. While the tool possesses hilariously suboptimal complexity, SimpleJoin fulfills its original purpose as a pure proof-of-concept example of a straw-man approach to designing join methods generalizable across NoSQL data models.

3) Quest, A Datalog Query Interface: Quest is a general interface for constructing, issuing, and evaluating Datalog queries over an arbitrary NoSQL database, provided Piper contains the relevant adapter logic. Quest consumes Datalog queries according with the C4 Overlog syntax [4], [5] and pre-processes the queries to extrapolate relation names and associated schemas. Subsequently, the workflow uses the input data translations provided in the adapter for the targeted NoSQL database to materialize the encoded Datalog relations. Quest evaluates the input Datalog queries using the C4 Overlog Runtime. If expected data schemas match the corresponding Quest interpretations, then Quest evaluates the queries and outputs the results successfully. Otherwise, the execution provides a bad termination and outputs an explicit reason for the failure via the citation of an inevitably present erroneous syntactic or semantic inconsistency.

4) YProv, A Why-Provenance Derivation Tool: YProv is a Piper package for collecting the why-provenance of Datalog queries issued to an arbitrary supported NoSQL database. The implementation uses the provenance rewriting approach outlined in [5] to augment the input Datalog program with a series of additional queries collecting the ‘firing’ tuples responsible for causing the appearance of tuples in particular relations. Analyzing the collective provenance version of the input Datalog program informs detailed conclusions regarding exactly why particular tuples appear in specific relations.

5) OntoDS: OntoDS is a Piper package for enforcing domain subsumption constraints defined by relationship specifications encoded in ontologies on inserts issued to an arbitrary supported NoSQL database. The package uses the RDF [6] standard to define ontological relationships between raw data and semantically meaningful metadata and enforces the resulting subsumption connections as integrity constraints. Inserting new data into the underlying NoSQL system triggers an evaluation step designed to determine whether the new data accord with existing subsumption relationships. If so, the insert passes. If not, the insert fails and OntoDS generates an explicit reason for the failure.

III. Future Work

Piper enjoys a wide variety of avenues for future work. The ultimate vision of Piper as a NoSQL data management tool collection and distribution service, combined with the relatively early stage of current developments, suggests a well-defined and expansive horizon of immediate and long-term steps toward fully realizing the idea. Immediate directions encompass general extensions to the capabilities of the framework. Longer-term directions contemplate more existential questions regarding best practices for managing data management tools and for increasing the amount of automated work accomplished at all points in the Piper system.

A. Immediate Future Work

Two very fundamental and immediate segments of future work consist of expanding the number of packages supported in the index and of increasing the numbers and types of supported NoSQL databases. The five packages currently populating the Piper index are relatively limited in terms of common value and applicability. To better fulfill the vision of Piper and better minimize the k-implementation of data management tools across all NoSQL products, Piper must contain a wider variety of packages, such as fast and efficient joins, integrity constraint enforcement tools, and sophisticated query processing engines, generalizable across a greater diversity of NoSQL products and data models, including graph databases such as Neo4J, column stores such as Cassandra, and object stores such as Versant.

Another pillar for immediate future work is an investment in case studies quantifying the true magnitude of the k-implementation problem using evaluation dimensions such as
management tool re-implementation frequency and lines of code across NoSQL products as well as software tools relying upon NoSQL products as dependencies. Such insights promise to further solidify the motivation for Piper using hard numbers and to better characterize the urgency of the project mission.

A third branch of immediate future work stems toward the goal of exploring the space of potential efficiency limitations and guarantees of generalized data management packages implemented in Piper versus implementations of the same tools either built-in or built-over instances of NoSQL products. Comparing the performance and complexities of data management tool implementations provided in Piper versus equivalent implementations pasted over particular NoSQL instances represents an essential step toward qualifying Piper as a valid and efficient solution to the data management tool \(k\)-implementation problem. Furthermore, demonstrating how packages provided in Piper may or may not out-perform tools cooked into NoSQL products constitutes an invaluable data point for determining the overall contributions of Piper to the community.

B. Longer-Term Future Work

Directions for long-term future work address deeper explorations regarding the efficacy of Piper as a solution to a problem not entirely separated from the influence of human elements. Requiring data management package implementations manifest a level of generalizability sufficient enough to apply across all supported NoSQL products may be an unnecessarily strong condition for participation in the index. Future investigations will contemplate methods for incorporating non-generalizable third-party tools for NoSQL data management as Piper packages. The new associated challenge would be building enough sophisticated intermediate adapter code to render the tools generalizable across products, despite the inherent specializations of the original software designs. Such questions also inspire exciting future investigations into adapter generalization and data access API standards. What qualities make adapters sufficiently powerful to glue Piper packages onto a NoSQL product? Do those qualities form a pattern and, if so, is there any way to automate the recognition or extraction of adapter logic from product source codes?

The dynamic nature of the space of NoSQL products renders research questions exploring automation capabilities and limitations critical. Because the space of NoSQL products is dynamic, the library of adapters required by Piper to support management tool dissemination must also be dynamic. As the number and diversity of NoSQL products continues to grow, relieving pressure from developers by at least semi-automatically extracting valuable access methods from arbitrary NoSQL product implementations represents a vital obstacle for the utility of a solution such as Piper. The importance of such a feature manifests in the fact that adapters characterized by the most general patterns of data access logic possible for each supported NoSQL product reduces the total amount of adapter logic necessary to encode the data access methods needed in order to support all the data management packages provided in the index.

Additionally, because high capacities for flexibility and customizability represent key attractions for the NoSQL paradigm, data management package implementations may present bottlenecks in some contexts and not in others. For example, the performance of SimpleJoin may be perfectly fine in applications relying upon such an operation infrequently or utilizing the operation on small amounts of data. However, the performance is prohibitively horrible in any other context. Users working in more demanding environments may prefer a faster join method or a variation of SimpleJoin less reliant upon potentially huge materialized views. In the latter case, predicated upon the future augmentation of both SimpleJoin and the suite of NoSQL adapters with tune-able parameters capable of adjusting the behaviors of the join and data access logic, providing a means for at least semi-automatically tuning the package implementations or data access methods presents another set of major research questions. The growing space of specialized NoSQL products threatens the long-term utility of a solution such as Piper. Automatically optimizing at least some aspects of package and adapter implementations according to classes of performance challenges is an essential avenue of future work.

IV. Conclusion

Piper is a package index and management system exclusively dedicated to hosting data management tools generalizable to all varieties of NoSQL products, predicated upon the specification of appropriate data access methods per NoSQL system. Piper explores the qualities, qualifications, and constraints of a solution to problems surrounding the \(k\)-implementation of data management tools across the growing space and diversity NoSQL products. A plethora of avenues for future extensions seek answers to fundamental usability and optimization questions. Current progress on early stages of the project and a well-defined agenda of future work combine to offer compelling evidence for the promising nature of the approach and an optimistic outlook for the success of future research and development efforts.

Acknowledgments

Special thanks extend to the anonymous reviewers for all the helpful feedback and to members of the UCSC SRL for providing comments on the paper draft.

References