Understanding Tradeoffs of Replicated Data Library Integration Strategies in Multilingual Environments

Provakar Mondal and Eli Tilevich
Software Innovations Lab, Virginia Tech, USA
{provakar,tilevich}@cs.vt.edu

ABSTRACT

Modern distributed systems replicate data across multiple execution sites by means of special-purpose replicated data libraries (RDLs), which provide read-write data access and synchronization. Programming languages often need to be mixed across replica sites to meet business requirements and resource constraints. Because RDLs are typically written in a single language, integrating them in multilingual environments requires special-purpose code, whose characteristics are poorly understood. We aim to bridge this knowledge gap by reviewing two key strategies for integrating RDLs in multilingual environments: (1) foreign-function interface (FFI) and (2) common data format (CDF). Our preliminary results indicate performance and implementation tradeoffs: CDF offers latency and memory consumption advantages, while incurring an additional implementation burden. With modern distributed systems utilizing multiple languages, our findings can inform the design of RDLs for multilingual replicated data systems.

CCS CONCEPTS

• **Software and its engineering → Interoperability**.

KEYWORDS

Cross-Language Interoperability, Replicated Data Systems

ACM Reference Format:
Provakar Mondal and Eli Tilevich. 2025. Understanding Tradeoffs of Replicated Data Library Integration Strategies in Multilingual Environments. In *26th International Middleware Conference (MIDDLEWARE Demos Posters and Doctoral Symposium '25), December 15–19, 2025, Nashville, TN, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3721464.3777425

1 PROBLEM CONTEXT AND RELATED WORK

In distributed systems, data is often replicated across multiple execution sites, known as *replicas*, to lower access latency, increase data availability, and prevent a single point of failure [6, 7]. Modern distributed systems commonly deploy replicated data libraries (RDLs) to manage all interactions with and synchronization of data. RDL provides API methods for accessing and updating replicated data, synchronizing the updates behind the scenes [8]. The need to replicate data in modern distributed applications has led to the creation of various RDLs, most of which are implemented in a single programming language [2]. If written in the same language, application code and an RDL can interact straightforwardly. When the business needs demand that application code at different replicas be implemented in dissimilar languages, developers can follow two strategies for integrating an RDL [5]. **Strategy I**: the application code interfaces with a monolingual RDL via the foreign function interfaces (FFIs). **Strategy II**: the RDL supports multiple languages; so the application code interacts with the RDL natively, while the replicas synchronize their state by exchanging messages in a common data format (CDF).

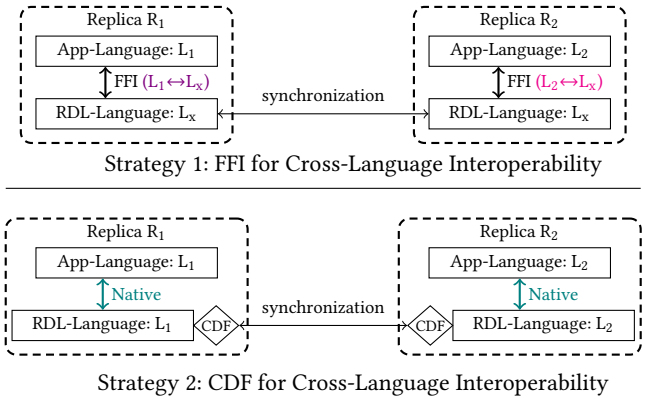
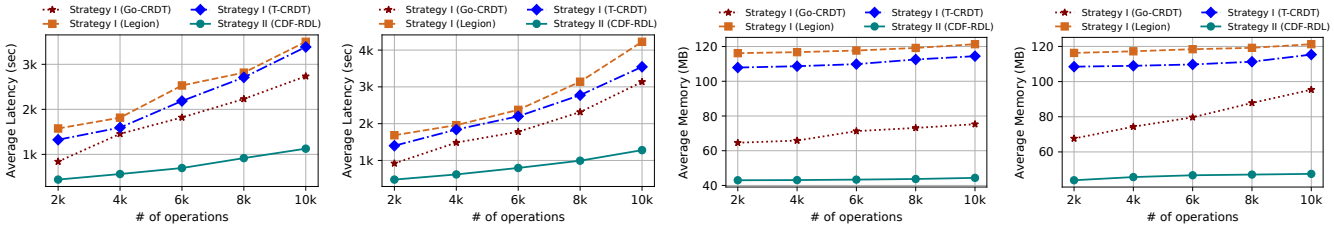


Figure 1: Cross-Language RDL Integration Strategies

The strategies appear in Figure 1. The application logic in R_1 and R_2 replicas is written in languages L_1 and L_2, respectively. In Strategy I (top), the RDL is monolingual, written in language L_x, with the application interacting with the RDL via FFIs. In this case, two different FFIs are required: $L_1 \leftrightarrow L_x$ for R_1 and $L_2 \leftrightarrow L_x$ for R_2. In Strategy II (bottom), the RDL is multilingual, so written in L_1 and L_2 for R_1 and R_2, respectively. Hence, the application code interacts natively with the library through regular local function calls. When exchanging messages across replicas, the message format needs to be mapped to a common data format (CDF).

When mixing languages within replicated data applications, it remains unclear which of the aforementioned strategies should be used, as their characteristics remain unexplored. How would employing FFI to interoperate with RDLs affect system performance? How would the implementation effort be affected if RDLs are implemented in different languages to communicate via CDF? To address this knowledge gap, this work reviews Strategies I and II, assessing how their performance characteristics influence their suitability for overcoming language barriers in this domain. Our results demonstrate a performance—implementation tradeoff; Strategy II offers performance advantages while requiring additional implementation effort to implement the RDL in all system languages.

This work is licensed under a Creative Commons Attribution 4.0 International License.
MIDDLEWARE Demos Posters and Doctoral Symposium '25, December 15–19, 2025, Nashville, TN, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1556-3/2025/12.
https://doi.org/10.1145/3721464.3777425



(a) Set Latency **(b) Map Latency** **(c) Set Memory** **(d) Map Memory**

Figure 2: Performance Characteristics for Strategy I and II

Related Work: Mixing different languages is a critical requirement for modern distributed systems, with numerous prior approaches. In Common Object Request Broker Architecture (CORBA), the Interface Definition Language (IDL) is used to define interfaces implemented in multiple languages [10]. CORBA's inter-language Remote Procedure Call (RPC) marshals arguments/return types, and maps errors reporting to language-specific constructs. Java-based MapReduce provides language bindings to Python and C++ [3]. TruffleVM, a virtual machine-based runtime, executes and combines multiple programming languages, with the languages interoperating via FFI [4]. Code-interoperability mode treats JVM and Java as a unified programming platform for prototyping heterogeneous code that can execute in parallel on various hardware and from different languages [9]. To our knowledge, these techniques have not been studied in our target domain, a shortcoming this work aims to address.

2 PRELIMINARY RESULTS

We created a multilingual replicated data system comprising three replicas in Go, JavaScript, and Java. To evaluate Strategy I, we experimented with three open-source third-party RDLs. Each library is monolingual; Go-CRDT is Go's built-in RDL package, Legion provides a JavaScript implementation of various replicated data types, and T-CRDT is a Java RDL. From each RDL, we used Set and Map. For these monolingual RDLs, we provided FFIs to each library's API. To evaluate Strategy II, we implemented a multilingual RDL in Go, JavaScript, and Java, enabling these languages to communicate via CDF, provided by Google's Protocol Buffer (protobuf) [1].

For performance evaluation, we executed several benchmarks with different numbers of RDL operations. We report the average latency and memory usage values measured for Strategy I's Go-CRDT, Legion, and T-CRDT, and Strategy II's CDF-RDL, with the results depicted in Figure 2. CDF-RDL exhibited the lowest latency and memory consumption as compared to the other three evaluation subjects of Strategy I. This efficiency is due to CDF-RDL's capability to interact with the application logic in the same languages, whereas the other RDLs rely on FFI.

We also measured the uncommented lines of code (ULOC) required for each strategy. The more code the application programmer has to write to integrate an RDL with application code, the higher the resulting programming burden is. All measurements appear in Table 1. Strategy II requires implementing the RDL in all system languages along with the implementation of CDF. Hence, Strategy II incurs an additional programming burden compared to Strategy I, which requires providing FFIs for target languages.

Table 1: Implementation Effort Across Strategies

Libraries	Uncommented Lines of Code (ULOC)			
	Go	JavaScript	Java	Total
Strategy I (Go-CRDT)	None	358	406	764
Strategy I (Legion)	214	None	345	559
Strategy I (T-CRDT)	231	165	None	396
Strategy II (CDF-RDL)	708	564	493	1763

Having reviewed two strategies for integrating RDL into multilingual replicated data systems, we assessed their respective performance and implementation tradeoffs. These findings inspire us to explore novel RDL designs that can achieve superior performance and software engineering characteristics.

REFERENCES

[1] David Castro. 2001. Protocol Buffers. https://protobuf.dev/.
[2] Kevin De Porre, Florian Myter, Christophe Scholliers, and Elisa Gonzalez Boix. 2020. CScript: A distributed programming language for building mixed-consistency applications. *J. Parallel and Distrib. Comput.* 144 (2020), 109–123.
[3] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[4] Matthias Grimmer, Roland Schatz, Chris Seaton, Thomas Würthinger, Mikel Luján, and Hanspeter Mössenböck. 2018. Cross-language Interoperability in a Multi-Language Runtime. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 40, 2 (2018), 1–43.
[5] Wen Li, Austin Marino, Haoran Yang, Na Meng, Li Li, and Haipeng Cai. 2024. How are multilingual systems constructed: Characterizing language use and selection in open-source multilingual software. *ACM Transactions on Software Engineering and Methodology* 33, 3 (2024), 1–46.
[6] David Mealha, Nuno Preguiça, Maria Cecilia Gomes, and João Leitão. 2019. Data Replication on the Cloud/Edge. In *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data*. 1–7.
[7] Provakar Mondal and Eli Tilevich. 2023. Undoing CRDT Operations Automatically. In *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 246–251.
[8] Provakar Mondal and Eli Tilevich. 2025. ER-π: Exhaustive Interleaving Replay for Testing Replicated Data Library Integration. In *Proceedings of the 26th International Middleware Conference*.
[9] Athanasios Stratikopoulos, Florin Blanaru, Juan Fumero, Maria Xekalaki, Orion Papadakis, and Christos Kotselidis. 2023. Cross-Language Interoperability of Heterogeneous Code. In *Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming*. 17–21.
[10] Steve Vinoski. 1997. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications magazine* 35, 2 (1997), 46–55.