Safe Combination of Data-Centric and Operation-Centric Consistency

Mirko Köhler
koehler@cs.tu-darmstadt.de
TU Darmstadt
Germany

Guido Salvaneschi
guido.salvaneschi@unisg.ch
University of St. Gallen
Switzerland

Abstract

Programming distributed systems requires maintaining consistency among data replicas. In recent years, various language-level abstractions have emerged for this issue that fall into two fundamental approaches: data-centric and operation-centric solutions. In the former developers explicitly assign consistency levels *to data*, in the latter they attach consistency constraints *to operations*. In practice, developers may benefit from both in the same application: data-centric consistency harmonizes well with object-oriented programming, yet one may need the flexibility to access the same data with a different consistency level depending on the operation. Currently, there is no solution that integrates both: it is a conceptual challenge to unify these two models and design a type system capable of ensuring their correct interaction.

We present ConOpY, a programming language that integrates both data-centric and operation-centric consistency into the same design. ConOpY's type system guarantees preventis consistency violations resulting from an improper mix of consistency models. ConOpY is implemented as a Java extension based on annotations.

CCS Concepts: • **Computing methodologies** → **Distributed programming languages**.

Keywords: replication, consistency, type systems, Java

ACM Reference Format:
Mirko Köhler and Guido Salvaneschi. 2023. Safe Combination of Data-Centric and Operation-Centric Consistency. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '23), October 22–27, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3618305.3623610

SPLASH Companion '23, October 22–27, 2023, Cascais, Portugal

ACM ISBN 979-8-4007-0384-3/23/10…$15.00
https://doi.org/10.1145/3618305.3623610

1 Background

In distributed applications exists a trade-off between consistency and availability [5]. Emphasizing consistency simplifies development due to a clearly-ordered view of events. Yet, this necessitates extensive inter-device coordination, impacting performance. To mitigate these issues, some systems prioritize availability by working on local states, merging them when feasible [13]. This approaches are efficient, offer consistency guarantees such as causality or convergence, and are referred to as *weakly* consistent [7]. In contrast, systems necessitating coordination are termed *strongly* consistent. Most software systems opt for weak consistency to enhance efficiency, although certain functionalities still demand strong consistency. For instance, in online stores, payments require strong consistency while browsing utilizes weak. In multiplayer online games, account data might need strong consistency whereas the game state is managed with weak [4]. Social networks require a mix, with strong consistency necessary for user index updates and weak for scalability [12].

Mixed consistency. In practice, neither strong nor weak are enough for an entire application. Developers must instead mix various levels of consistency. There are two strategies. (1) They can utilize multiple stores, each providing a distinct *data* consistency level [9, 10]. Here, the consistency level is determined upon assigning data objects to their respective stores, and objects are always accessed using the consistency level of the store. This limits flexibility as the same data can't be accessed with different consistency levels – a bank account is typically viewed as strongly consistent, but in some situations, reading the account balance with a weaker consistency level suffices. (2) Developers employ data stores that allow selection of consistency levels per *operation*, enabling access to a single data object with varying levels [2, 8]. This, however, necessitates to consider the consistency level of all operations applied to an object, not just the object itself. Developers must reason about the consistency of all modifications, from all processes involved in the application.

Both options, however, are prone to errors as incorrect mixing of consistency levels can lead to *consistency violations*, e.g., when a weakly consistent value (which might temporarily be inconsistent due to concurrent writes not visible to all processes simultaneously) is written into a strongly consistent object that guarantees a common global view. The

root problem is the flow of a weakly consistent value into a strongly consistent object. At the *data store* level, the system can't assist developers in circumventing these problematic flows as it can't reason about the context in which objects are used within the application.

Language approaches. Researchers have suggested elevating the selection of consistency levels from data stores to the programming language itself. In this setting, the flow of consistency levels can be tracked, and improper mixing of consistency levels, potentially leading to consistency violations, can be detected. Some approaches [3, 6, 9, 11] enable developers to annotate their replicated data with a *data-centric* consistency level. These approaches utilize type systems to prevent consistency violations. However, similar to the data store level, data-centric consistency constrains expressiveness, as all operations on an object execute with the same consistency. An alternative is to use *operation-centric* consistency levels, wherein developers specify consistency as a property of the operations accessing the data rather than the data itself. This allows for more nuanced specification of consistency, as different operations on an object can operate with different consistency levels.

2 Overview

We introduce ConOpY, an object-oriented programming language that offers integrated support for mixed consistent replication. The novelty lies in the innovative approach to incorporating operation-centric consistency, thereby enabling seamless reasoning about both data-centric *and* operation-centric consistency through a unified mechanism.

At the core of ConOpY lies the concept of *replicated objects*, objects that exist across multiple processes. Developers interact with replicated objects through operations, which are invoked as methods. Operations can modify and query the replicated state. Each operation is assigned a consistency level, which dictates how processes coordinate to execute the operation. To represent this, we define two consistency levels, Strong and Weak (corresponding to unavailability and high-availability, respectively [1]). Strong operations require coordination between processes, ensuring strong correctness guarantees through sequential execution of concurrent operations. In contrast, Weak operations can be executed by a single process and are later coordinated, allowing for temporarily inconsistent states between replicas, with the eventual guarantee of consistency.

ConOpY offers two fashions to define consistency: consistency for entire replicated objects (data-centric) or consistency for individual methods (operation-centric). These choices of consistency are enforced through the type system, enabling static reasoning about consistency. For the data-centric approach, developers annotate classes with specific consistency levels, resulting in replicated objects instantiated with the designated consistency level. All operations performed on these objects inherit the chosen consistency, ensuring a consistent treatment of the object's state. In the operation-centric approach, consistency is specified at the granularity of methods. Developers attach consistency annotations to method declarations, and ConOpY automatically infers the required consistency levels from these method specifications. Importantly, ConOpY facilitates the seamless interoperability of these two approaches, allowing developers to combine them within the same application as needed.

To ensure the correct mixing of data-centric and operation-centric consistency levels and prevent consistency violations, ConOpY is equipped with a consistency type system that verifies the correctness of specified consistency levels with respect to each other, specifically ensuring that Strong data does not depend on Weak(er) data.

We intend to evaluate ConOpY on both a local setup and the EC2 Cloud, aiming to demonstrate that it enables the correct integration of data-centric and operation-centric consistency. Additionally, we seek to showcase ConOpY's ability to introduce concurrent execution when feasible, all while relieving developers from the burden of reasoning about consistency correctness.

Acknowledgments

This work has been supported by the Swiss National Science Foundation and the LOEWE centre emergenCITY.

References

[1] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. 2013. Highly Available Transactions: Virtues and Limitations. *PVLDB*.

[2] Apache Cassandra. 2022. *Apache Cassandra Documentation.* https://cassandra.apache.org

[3] 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. https://doi.org/10.1016/j.jpdc.2020.05.010

[4] Ziqiang Diao. 2013. Consistency Models for Cloud-based Online Games: the Storage System's Perspective. In *Grundlagen von Datenbanken.*

[5] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News* 33, 2 (jun 2002), 51–59. https://doi.org/10.1145/564585.564601

[6] Mirko Köhler, Nafise Eskandani, Pascal Weisenburger, Alessandro Margara, and Guido Salvaneschi. 2020. Rethinking Safe Consistency in Distributed Object-Oriented Programming. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 188 (nov 2020), 30 pages. https://doi.org/10.1145/3428256

[7] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2011. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 401–416. https://doi.org/10.1145/2043556.2043593

[8] Microsoft. 2022. *Consistency levels in Azure Cosmos DB.* https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

[9] Matthew Milano and Andrew C. Myers. 2018. MixT: A Language for Mixing Consistency in Geodistributed Transactions. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 226–241. https://doi.org/10.1145/3192366.3192375

[10] MySQL. 2022. *MySQL 8.0: Configuring Transaction Consistency Guarantees*. https://dev.mysql.com/doc/refman/8.0/en/group-replication-configuring-consistency-guarantees.html

[11] Florian Myter, Christophe Scholliers, and Wolfgang De Meuter. 2018. A CAPable Distributed Programming Model. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Boston, MA, USA) *(Onward! 2018)*. Association for Computing Machinery, New York, NY, USA, 88–98. https://doi.org/10.1145/3276954.3276957

[12] Xiao Shi, Scott Pruett, Kevin Anthony James Doherty, Jinyu Han, Dmitri Petrov, Jim Carrig, John Hugg, and Nathan Grasso Bronson. 2020. FlightTracker: Consistency across Read-Optimized Online Stores at Facebook. In *OSDI*.

[13] George Zakhour, Pascal Weisenburger, and Guido Salvaneschi. 2023. Type-Checking CRDT Convergence. *Proc. ACM Program. Lang.* 7, PLDI, Article 162 (jun 2023), 24 pages. https://doi.org/10.1145/3591276

Received 2023-08-15; accepted 2023-08-30