Brief Announcement: On Mixing Eventual and Strong Consistency: Bayou Revisited*

Maciej Kokociński
Maciej.Kokocinski@cs.put.edu.pl

Tadeusz Kobus
Tadeusz.Kobus@cs.put.edu.pl
Institute of Computing Science
Poznan University of Technology
Poznań, Poland

Paweł T. Wojciechowski
Pawel.T.Wojciechowski@cs.put.edu.pl

ABSTRACT

In this paper we study the properties of eventually consistent distributed systems that feature arbitrarily complex semantics and mix eventual and strong consistency. These systems execute requests in a highly-available, weakly-consistent fashion, but also enable stronger guarantees through additional inter-replica synchronization mechanisms that require the ability to solve distributed consensus. We use the seminal Bayou system as a case study, and then generalize our findings to a whole class of systems. We show dubious and unintuitive behaviour exhibited by those systems and provide a theoretical framework for reasoning about their correctness. We also state an impossibility result that formally proves the inherent limitation of such systems, namely *temporary operation reordering*, which admits interim disagreement between replicas on the relative order in which the client requests were executed.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; • **Computer systems organization → Reliability**; **Availability**;

KEYWORDS

eventual consistency, mixed consistency, fault-tolerance

ACM Reference Format:
Maciej Kokociński, Tadeusz Kobus, and Paweł T. Wojciechowski. 2019. Brief Announcement: On Mixing Eventual and Strong Consistency: Bayou Revisited. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3293611.3331583

1 INTRODUCTION

Modern replicated services running on the Internet are designed for high availability and low latency in serving client requests. These desirable traits come at the expense of weaker consistency guarantees, e.g., eventual or causal consistency [17]. Such weak consistency models are often sufficient and thus embraced by the widely employed NoSQL data stores. However, there are many use cases in which stronger guarantees are needed, in the form of, e.g., serializable transactions. This is why in recent years various NoSQL vendors started adding (quasi) transactional support to their systems [6] [7], albeit these add-on mechanisms are often very limited.[1]

In this paper we study the inherent limitations of replicated systems which mix eventual consistency with strong consistency, and at the same time retain rich transactional semantics. More precisely, we are interested in eventually consistent systems which facilitate serializable transactions that can operate on the same data as highly-available, weakly consistent operations. Weakly consistent operations are guaranteed to progress, and are handled in such a way that the processes eventually converge to the same state within each network partition, even when strongly consistent transactions cannot complete due to network and process faults. The full version of the paper is available in [11].

2 BAYOU – A CASE STUDY

In the past several mixed consistency systems have been proposed, most notably [16] [8] [4]. To showcase some of the problems which result from having multiple consistency models coexisting in a system we study the seminal Bayou system [16], one of the first always available, eventually consistent data stores.

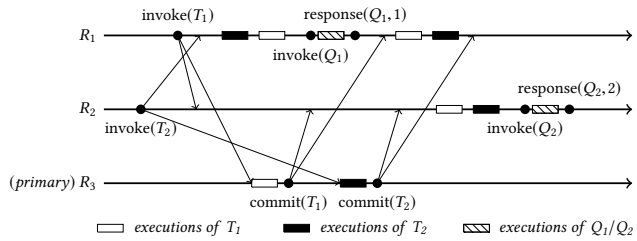
Overview. Each Bayou server speculatively total-orders all received client requests using a simple timestamp-based mechanism and without prior agreement with other servers. This way a Bayou process (which we call a *replica*) can respond to a request even in the presence of network partitions in the system. In the background, Bayou replicas synchronize to enforce the final request execution order, as established by a primary replica (the primary periodically commits the operations it has received). When desired, a client may wait until the request *stabilizes*, i.e., it is processed according to the final execution order, so the response can never change. Logically, the state of each replica corresponds to the following list l of received requests: the prefix of l contains stabilized requests, arranged in the order established by the primary; other requests are kept further on l, arranged according to their timestamps. Stabilization of a request means that the request is moved from the second part of the list to the end of the first part of the list. Note that sometimes this operation might involve rolling back

*This work was supported by the Foundation for Polish Science, within the TEAM programme co-financed by the European Union under the European Regional Development Fund (grant No. POIR.04.04.00-00-5C5B/17-00). Kokociński and Kobus were also supported by the Polish National Science Centre (grant No. DEC-2012/07/B/ST6/01230) and partially by the internal funds of the Faculty of Computing, Poznan Univ. Techn.

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07.
https://doi.org/10.1145/3293611.3331583

[1] For instance, Riak allows strongly consistent (serializable) operations to be performed only on distinct data [7], whereas using the so called light weight transactions in Apache Cassandra on data that are accessed also in the regular (eventually consistent) fashion leads to undefined behaviour [10].




```
T1 {                T2 {                Q1/Q2 {
   x = 1               y = 1               return z
   if y = 1 then       if x = 1 then    }
      z = 1               z = 2
}                   }
```

Figure 1: Example execution of Bayou showing temporary operation reordering and circular causality. Initially, replica R_1 executes transactions T_1 and T_2 in order T_2, T_1, which corresponds to T_1's and T_2's timestamps. This transaction execution order is observed by the client who issues query Q_1. On the other hand, R_2 executes the transactions according to the final execution order (T_1, T_2), as established by the primary replica R_3. Hence, the client who issued query Q_2 observes a different execution order than the client who issued Q_1. The execution orders perceived by the clients form a circular dependency between T_1 and T_2: T_1 depends on T_2 as evidenced by Q_1's response, while T_2 depends on T_1 as evidenced by Q_2's response. Note that replicas execute the operations with a delay (e.g., caused by CPU being busy). Also, notice that R_1 reexecutes the operations once it gets to know the final order.

and reexecuting some of the requests. Intuitively, Bayou combines timestamp-based eventual consistency [17] and serializability [13].

For the purpose of our analysis we assume that requests in Bayou are arbitrary but deterministic *operations* that produce some response values. An operation might correspond to, e.g., a transaction specified in SQL. When a client is interested in the stable response, we say that the operation is *strong*. Otherwise it is *weak*, and the response is returned to the client before the final operation execution order is established. This way any weak operation is non-blocking (with respect to network communication), but its ultimate impact on the system's state might differ from what the client can infer from the response (if the final execution order differs from the tentative one). On the other hand, a strong operation returns a response only after the final execution order is established. Hence the guarantees on the execution order of strong operations are more stringent compared to the guarantees of weak operations. By abstracting a system described above, we obtain a model of a distributed system with arbitrarily complex semantics that mixes eventual and strong consistency. The semantics are formalized by a specification of a sequential data type exporting a set of operations available to the clients.

Anomalies. We demonstrate two phenomena present in Bayou that may come as dubious and unintuitive. Interestingly, they are never exhibited by popular NoSQL systems (which guarantee only eventual consistency), nor by strongly consistent solutions (e.g., DBMS). In the first phenomenon, which we call *temporary operation reordering*, the replicas may temporarily disagree on the relative order in which the requests (modeled as operations) submitted to the system were executed. The difference in operation execution order can be observed by the clients. The second phenomenon, *circular causality*, signifies a situation in which, by examining the return values of the operations processed by a system, one may discover a cycle in the causal dependency between the operations. Both of these anomalies are present in Bayou, because it features two incompatible ways of ordering operation executions (see Figure 1). Interestingly, circular causality can be avoided in Bayou [11], however, this is not the case with temporary operation reordering.

Correctness guarantees. Because of the phenomena described above, the guarantees provided by Bayou cannot be formalized using the correctness criteria used for contemporary eventually consistent systems. For example, *basic eventual consistency (BEC)* by Burckhardt [3] directly forbids circular causality. BEC also requires the relative order of any two operations, as perceived by the client, to be consistent and to never change. Similarly, *strong eventual consistency (SEC)* by Shapiro *et al.* requires any two replicas that delivered the same updates to have equivalent states. Obviously, Bayou neither satisfies BEC nor SEC. Bayou satisfies the operational specification in [8]. However, we aim to provide a declarative specification, similar in style to popular consistency criteria, such as *sequential consistency* [12], or *serializability* [13].

In order to formalize the guarantees of systems that, similarly to Bayou, admit temporary operation reordering, we introduce a new correctness criterion called *fluctuating eventual consistency (FEC)*, which can be viewed as a generalization of BEC. More precisely, BEC requires that the following three conditions are satisfied: EV, NCC, and RVal($\mathcal{F}$). EV requires every operation to eventually become visible to all subsequent ones. NCC forbids circular causality. Finally, RVal($\mathcal{F}$) requires the return values to be consistent with the replicated data type $\mathcal{F}$ specification,[2] i.e., for any given operation *op*, the value returned, and the value predicted by $\mathcal{F}$ (based on the set of operations visible to *op* and a single total order of all operations), have to match.

FEC retains EV and NCC but relaxes RVal, so that different operations can perceive different operation orders. However, we require that the different perceived operation orders converge to one final execution order.

For strong operations Bayou provides more stringent guarantees, namely sequential consistency (Seq). A response of a strong operation *op* always reflects the serial execution of all stabilized operations up to the point of *op*'s commit.

When weak and strong operations are mixed, we denote by FEC(*weak*, $\mathcal{F}$) $\wedge$ Seq(*strong*, $\mathcal{F}$) the combination of guarantees including FEC for weak operations and sequential consistency for strong operations.

[2] A replicated data type specification is a generalization of a sequential data type specification, that is suitable for certain replicated data types such as MVRs [15]. Although this generalization is not necessary in case of Bayou, which always processes all operations sequentially, we adopt it for generality and compatibility with BEC.

Wait-freedom. Now we show that, unlike many existing eventually consistent data stores, Bayou does not guarantee bounded wait-free execution of operations, i.e., there does not exist a bound on the number of steps of the Bayou protocol that a replica takes before returning a response to the client [9]. Consider an infinite execution in which new operations are being invoked on each replica with a constant rate. Assume that one of the replicas is unable to keep up executing the operations at the same rate as it receives them over network. The replica starts lagging behind. Each new operation invoked at the replica will be executed only after the current stash of lagging requests is executed. Since the lag is increasing, so is the response time for the operations invoked on the replica. Thus no bound on response time exists. It is surprising, because the result holds also when the clients do not wait for the operations to stabilize.

Fault-tolerance. Bayou's reliance on the primary makes it not fault-tolerant (the primary is the single point of failure). Although the primary may recover, when it is down, operations do not stabilize, and thus no strong operation can complete. Alternatively, the primary could be replaced by a distributed commit protocol. If two-phase-commit (2PC) [1] is used, the phenomena illustrated in Figure 1 are not possible. However, in this approach, a failure of any replica blocks the execution of strong operations (in 2PC all the replicas need to be operational). On the other hand, if a *non-blocking* commit protocol is used, e.g., based on a *total order broadcast (TOB)* [14], the system may stabilize operations despite (a limited number of) failures. Any scheme which admits failures of at least one replica, and thus do not depend on synchronous communication with all the replicas, is necessarily prone to the phenomena described above.

3 IMPOSSIBILITY RESULT

Now we proceed to our central contribution. Informally, our result shows that it is impossible to devise a fault-tolerant system that mixes strong and weak consistency and implements arbitrary semantics (as defined by a specification of a replicated data type $\mathcal{F}$) but which never admits temporary operation reordering. Before we state our theorem, we discuss our approach to formulating it.

We are interested in the behaviour of systems, both in the fully asynchronous environment, when timing assumptions are consistently broken (e.g., because of prevalent network partitions), and in a stable one, when the minimal amount of synchrony is available so that consensus eventually terminates. Thus, we consider two kinds of *runs*: *asynchronous* and *stable*. Replicas are not aware which kind of a run they are currently executing. In the stable runs, we augment the system with the failure detector Ω, the weakest failure detector capable of solving distributed consensus in the presence of failures [5]. This way replicas can use, e.g., TOB for communication. In the asynchronous runs, a system which waits for TOB to complete may block forever.

We make some basic assumptions about the systems considered (we expressed the assumptions implicitly in Section 1). In case of weak operations the system needs to behave as a regular eventually consistent data store as defined by the *write-propagating data store* abstraction by Attiya *et al.* [2]. Intuitively, each replica processes weak operations as soon as possible, and eagerly synchronizes with other replicas, to ensure eventual consistency. The system cannot rely on a central coordinator for the propagation of the weak operations (as in [4]). In case of strong operations, we consider systems with *non-blocking strong operations*, which means that the execution of a strong operation never blocks on communication with all replicas. This assumption eliminates protocols such as 2PC for inter-replica synchronization required to complete strong operations; quorum-based mechanisms, such as TOB, are permitted.

If a mixed-consistency system could avoid temporary operation reordering, it would mean that it ensures BEC for weak operations and also provides at least sequential consistency (Seq) for strong operations. However, as we show, it is impossible.

Theorem 1. *Let $\mathcal{F}$ be an arbitrary replicated data type. If a system guarantees BEC($weak, \mathcal{F}$) in asynchronous runs, then it does not guarantee BEC($weak, \mathcal{F}$) $\wedge$ Seq($strong, \mathcal{F}$) (neither in the asynchronous, nor in the stable runs).*

If we replaced BEC with FEC above, the theorem would not longer be true. In fact, in [11] we formally prove that a fault-tolerant version of Bayou that avoids circular causality guarantees FEC($weak, \mathcal{F}$) in asynchonous runs and FEC($weak, \mathcal{F}$)$\wedge$Seq($strong, \mathcal{F}$) in stable runs. Thus our result shows that admitting temporary operation reordering is the inherent cost of mixing eventual and strong consistency when we make no assumptions about the semantics of $\mathcal{F}$. Naturally, for certain replicated data types achieving both BEC($weak, \mathcal{F}$), and Seq($strong, \mathcal{F}$) is possible (it is easy to show that it is the case for, e.g., a single distributed register).

REFERENCES

[1] Philip A., Bernstein, Vassco Hadzilacos, and Nathan Goodman. 1987. *Concurrency control and recovery in database systems.* Addison-Wesley.
[2] Hagit Attiya, Faith Ellen, and Adam Morrison. 2015. Limitations of Highly-Available Eventually-Consistent Data Stores. In *Proc. of PODC '15.*
[3] Sebastian Burckhardt. 2014. Principles of Eventual Consistency. *Foundations and Trends in Programming Languages* 1, 1-2 (Oct. 2014), 1–150.
[4] Sebastian Burckhardt, Daan Leijen, Jonathan Protzenko, and Manuel Fähndrich. 2015. Global Sequence Protocol: A Robust Abstraction for Replicated Shared State.. In *Proc. of ECOOP '15.*
[5] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. 1996. The Weakest Failure Detector for Solving Consensus. *J. ACM* 43, 4 (July 1996), 38.
[6] Apache Cassandra documentation. 2019. Light Weight Transactions in Cassandra. https://docs.datastax.com/en/cql/3.3/cql/cql_using/useInsertLWT.html.
[7] Basho documentation. 2019. Consistency levels in Riak. https://docs.basho.com/riak/kv/2.2.3/developing/app-guide/strong-consistency.
[8] Alan Fekete, David Gupta, Victor Luchangco, Nancy Lynch, and Alex Shvartsman. 1996. Eventually-serializable Data Services. In *Proc. of PODC '96.*
[9] Maurice Herlihy. 1991. Wait-free Synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 13, 1 (Jan. 1991), 124–149.
[10] Apache Cassandra Issues (Jira). 2016. Mixing LWT and non-LWT operations can result in an LWT operation being acknowledged but not applied. https://jira.apache.org/jira/browse/CASSANDRA-11000.
[11] Maciej Kokociński, Tadeusz Kobus, and Paweł T. Wojciechowski. 2019. On mixing eventual and strong consistency: Bayou revisited. *ArXiv preprint arXiv:1905.11762 [cs.DC]* (2019). https://arxiv.org/abs/1905.11762
[12] L. Lamport. 1979. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Comput.* C-28, 9 (Sept. 1979).
[13] Christos H. Papadimitriou. 1979. The Serializability of Concurrent Database Updates. *J. ACM* 26, 4 (1979).
[14] Fernando Pedone, Rachid Guerraoui, and André Schiper. 1998. Exploiting Atomic Broadcast in Replicated Databases. In *Proc. of Euro-Par '98.*
[15] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free Replicated Data Types. In *Proc. of SSS '11.*
[16] Douglas Terry, Marvin Theimer, Karin Petersen, Alan Demers, Mike Spreitzer, and Carl Hauser. 1995. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. of SOSP '95.*
[17] Werner Vogels. 2009. Eventually Consistent. *Commun. ACM* 52, 1 (Jan. 2009), 5.