Maciej Kokociński

Poprawność Wysoko Dostępnych Ostatecznie Spójnych Systemów Zreplikowanych

Rozprawa doktorska

Przedłożono Radzie Dyscypliny
Informatyka Techniczna i Telekomunikacja
Politechniki Poznańskiej

Promotor: dr hab. inż. Paweł T. Wojciechowski, prof. PP

Poznań · 2022

Streszczenie

Globalne usługi, które stanowią serce dzisiejszego internetu, takie jak komunikatory, sieci społecznościowe, handel elektroniczny, bankowość, rachunki maklerskie, czy gry online, są realizowane przez skomplikowane systemy rozproszone. Aby sprostać rosnącemu obciążeniu generowanemu przez miliony użytkowników, te systemy muszą być skalowalne horyzontalnie, co oznacza, że można zwiększyć ich wydajność poprzez dodanie dodatkowych węzłów obliczeniowych. Spełnienie owych ogromnych wymagań dotyczących skalowalności jest dodatkowo utrudnione przez fakt, że systemy te muszą pozostawać sprawne przez cały czas. Jako, że żaden sprzęt komputerowy, czy sieciowy, nie są całkowicie odporne na usterki techniczne, same usługi muszą być zaimplementowane w taki sposób, który umożliwi im z łatwością tolerować awarie. W ten sposób systemy realizujące te usługi mogą stać się *wysoko dostępne*, tzn. mogą przetwarzać żądania klientów nawet gdy występują (częściowe) awarie systemu.

Powszechną techniką stosowaną w celu zwiększenia dostępności systemu jest *replikacja*, która polega na utrzymywaniu wielu kopii danych i kodu usługi, zwanych *replikami*, na fizycznie niezależnych węzłach, często rozproszonych geograficznie. Poza zapewnieniem odporności na awarie, replikacja poprawia skalowalność i obniża czasy odpowiedzi gdy repliki znajdują się geograficznie blisko klientów. Tradycyjne schematy replikacji, takie jak *zreplikowana maszyna stanowa* [15, 16] czy *replika główna – replika zapasowa* (ang. *primary-backup*) [17], utrzymują silną spójność pomiędzy replikami, tzn. repliki koordynują zmiany swoich stanów w taki sposób, aby system jako całość dla klientów sprawiał wrażenie pojedynczego scentralizowanego serwera. Jednakże, utrzymywanie replik w stanie synchronizacji jest kosztowne, ponieważ zazwyczaj wymaga rozwiązania problemu rozproszonego konsensusu. Z tego powodu przed wysłaniem odpowiedzi do klienta repliki muszą wymienić pomiędzy sobą wiele wiadomości, co znacząco zwiększa czas odpowiedzi. Ponadto, utrzymanie spójności replik jest niemożliwe gdy występują podziały sieci, a usługa powinna pozostać dostępna, jak stanowi słynne twierdzenie CAP [18]. Dlatego tradycyjne

silnie spójne schematy replikacji gwarantują jedynie dostępność w przypadku (ograniczonej liczby) awarii replik, ale nie w przypadku awarii sieci powodujących brak łączności pomiędzy grupami replik.

W celu pokonania powyższych ograniczeń, wymagania co do spójności mogą być osłabione. Kiedy stosowana jest jakaś odmiana *spójności ostatecznej* [19] repliki mogą synchronizować swoje stany jedynie ostatecznie. Znaczy to, że repliki mogą przetwarzać żądania klientów niezależnie i rozsyłać zmiany stanów asynchronicznie. Z tego powodu, nawet gdy występują podziały sieci, wysoka dostępność może być utrzymana. By osiągnąć ten cel, ostatecznie spójne systemy cechują się zdecentralizowaną architekturą i polegają na (asynchronicznej) komunikacji peer to peer (p2p). Jest to model wprowadzony pierwszy raz przez Amazon w ich wpływowym magazynie danych Dynamo [20] i który został powielony w wielu popularnych magazynach danych NoSQL (przykładowo w Apache Cassandra [21], Scylla [22], Riak [23], Voldemort [24] oraz Netflix Dynomite [25]).

Jednakże, osłabione modele spójności oferują słabsze gwarancje i z zasady dopuszczają pewną ilość niespójności. Jeżeli nie są obsłużone prawidłowo, mogą prowadzić to niepożądanych anomalii, w tym utraty danych. Z tego powodu programiści muszą ostrożnie projektować kod zreplikowanych usług tak by obsłużyć wszystkie przypadki graniczne i uwzględnić możliwość występowania anomalii. Aby zredukować obciążenie programistów stosowane są specjalne struktury danych, zwane *wolnymi od konfliktów, replikowanymi typami danych* (ang. *conflict-free replicated data types, CRDTs*) [26, 27, 28]. CRDT mogą być zaimplementowane w sposób całkowicie asynchroniczny i z zasady zapewniają ostateczną zbieżność stanu replik. Do popularnych CRDT zaliczają się *wielowartościowe rejestry* (ang. *multi-value registers, MVRs*), *rejestry typu ostatni-zapis-wygrywa* (ang. emphlast-write-wins registers, LWW-registers), *pozytywno-negatywne liczniki* (ang. *positive-negative-counters, PN-counters*), *zbiory zaobserwowane-usuń* (ang. *observed-remove sets, OR-sets*) [27], jak i struktury danych do współedytowania tekstu online [29].

Niestety semantyka CRDT jest bardzo ograniczona. Aby zapewnić wysoką dostępność, niskie czasy odpowiedzi oraz ostateczną zbieżność stanów replik, struktury te wymagają aby wszystkie operacje były naprzemienne, albo by istniały naprzemienne, asocjacyjne, idempotentne procedury scalania stanów replik. Dlatego struktury te nie nadają się do wszystkich zastosowań. Dla przykładu rozważmy pojedynczy nieujemny licznik całkowitoliczbowy. Operacja dodania może być trywialnie zaimplementowana w sposób wolny od konfliktów, ponieważ operacje dodawania są naprzemienne. Jednakże implementacja operacji odejmowania wymaga globalnego uzgodnienia żeby zapewnić, że wartość licznika nigdy nie spadnie poniżej zera. Podobnie w systemie aukcyjnym współbieżne oferty mogą być uznane za operacje niezależne, więc ich wykonanie nie wymaga synchronizacji. Jednakże operacja, która zamyka aukcję wymaga rozwiązania rozproszonego konsensusu by wybrać jedną zwycięską ofertę [30].

Z powodu ograniczeń CRDT, i spójności ostatecznej w ogólności, w ostatnim

czasie podejmowanych było wiele prób, zarówno w przemyśle [31, 32, 33, 34], jak i w nauce [35, 36, 37, 38, 39, 40, 41, 42], żeby wzmocnić semantykę systemów ostatecznie spójnych poprzez dopuszczenie do wykonywania części operacji z silniejszymi gwarancjami spójności, lub poprzez dodanie funkcji pseudotransakcyjnych. W ten sposób wyłoniła się nowa klasa systemów wysoko dostępnych, zwana *systemami o mieszanej spójności*, w której to jedynie część operacji musi być wysoko dostępna. Operacje wykonywane ze słabszymi gwarancjami spójności, zwane *słabymi*, pozostają wysoko dostępne nawet w obliczu występowania awarii, podczas gdy operacje wykonywane w trybie silnie spójnym, zwane *silnymi*, mogą się blokować, np z powodu awarii sieci. Analiza i formalizacja własności poprawności systemów o mieszanej spójności stanowią główny temat tej pracy.

Motywacje

Tak jak to zostało powyżej omówione, z powodu rosnącego znaczenia wysokiej dostępności w kontekście współczesnych globalnych usług w internecie, systemy wysoko dostępne, włączając w to rozwiązania warstwy pośredniczącej (ang. middleware) takie jak magazyny danych NoSQL, dynamicznie się rozpowszechniają. Z powodu naszego rosnącego uzależnienia od tych systemów, badanie i weryfikacja ich poprawności stanowi problem najwyższego znaczenia. Mimo, że istnieje obecnie znacząca liczba badań w tym obszarze, wciąż wiele pozostaje do zrobienia.

Przez długi czas spójność ostateczna unikała klarownego ujęcia i sformalizowania. Wiele definicji zostało zaproponowanych (zobacz np. [19, 43, 44, 45, 46, 2, 26, 37, 47, 48, 49]), które różniły się znacząco zarówno pod względem użytych technik formalizacji, jak i praktycznie oferowanych gwarancji. Z drugiej strony silna spójność, która jest stosowana od wielu dekad (zobacz np. [50, 51, 52]), jest dużo lepiej zrozumiana. Jest tak ponieważ silna spójność opiera się na bardzo prostej zasadzie: system silnie spójny wykonujący żądania współbieżnie powinien być nierozróżnialny od systemu wykonującego żądania sekwencyjnie. W porównaniu do spójności silnej, spójność ostateczna zapewnia gwarancje, które są nie tylko znacznie słabsze, ale również trudne do zrozumienia z powodu ich skomplikowania albo nieprecyzyjności. Dla przykładu definicja podana przez Vogelsa [19] mówi, że kiedy zapisy ustaną, ostatecznie wszystkie odczyty zwrócą tą samą wartość, ale definicja ta nie nakłada żadnych ograniczeń na zwrócone wartości gdy zapisy nigdy nie ustają. Znowuż *typy chmurowe* (ang. *cloud types*) [37] wymagają od użytkownika myślenia w kategoriach rewizji, które mogą się dzielić i łączyć jak w systemie kontroli kodu źródłowego. Jest to tzw model *spójności rewizji* (ang. *revision consistency*) [47], który został ostatecznie porzucony ze względu na nadmierne skomplikowanie [38]. Z tego powodu udowadnianie poprawności konkretnego systemu ostatecznie spójnego, jak również wnioskowanie na temat takich systemów w ogólności, jest bardziej wymagające. Co więcej, gdy ostatecznie spójne (słabe) operacje są mieszane z operacjami silnie spójnymi (silnymi) w jednym systemie o spójności mieszanej,

klarowność oferowanych gwarancji jest jeszcze mocniej obniżona. Istotnie, nie ma obecnie konsensusu co do oczekiwanej semantyki tego typu systemów.

Systemom o mieszanej spójności wykorzystywanym w przemyśle brakuje klarownie zdefiniowanej semantyki, albo mają tę semantykę znacząco ograniczoną. Dla przykładu wykonywanie *lekkich transakcji* w Apache Cassandra [21] na danych, które są w tym samym czasie modyfikowane przez zwykłe operacje ostatecznie spójne, prowadzi do niezdefiniowanego stanu systemu (ang. undefined behaviour) [53]. Z drugiej strony w Riaku [23] elementy, do których dostęp jest realizowany przez operacje słabe i elementy, do których dostęp jest realizowany przez operacje silne, muszą się mieścić w oddzielnych, niezależnych przestrzeniach, zwanych *kubełkami*, [34], przez co system tak naprawdę nie posiada semantyki spójności mieszanej. Inne systemy [39, 31, 32] umożliwiają dobór poziomu spójności jedynie dla operacji odczytu. Klienci mogą wybrać odczyty świeże albo potencjalnie przestarzałe (ang. stale). Z kolei zapisy w tym podejściu są zawsze wykonywane jako operacje silne.

Wszystkie znane podejścia, które faktycznie spełniają semantykę mieszanej spójności posiadają jakieś ograniczenia w kontekście ich działania w obliczu awarii. Dla przykładu w *typach chmurowych* [37], jak i w *globalnym protokole sekwencyjnym* [38], wszystkie operacje zapisu (zarówno słabe jak i silne) muszą być przekazane do scentralizowanego podsystemu, zwanego *chmurą*, którego zadaniem jest rozgłaszanie do wszystkich węzłów komunikatów o zmianach stanów w uporządkowanym strumieniu. Kiedy chmura jest niedostępna, np z powodu awarii większości serwerów działających w chmurze, albo z powodu podziału sieci, operacja zapisu wciąż może zostać wykonana i zaaplikowana lokalnie na którymś z węzłów, ale nie będzie widoczna dla innych węzłów. Jako kolejny przykład rozważmy *replikację leniwą* [54], *spójność niebiesko-czerwoną* [35], oraz *częściowe ograniczenia porządku* [36], w których to wszystkie repliki muszą pozostawać sprawne, aby możliwe było wykonanie operacji silnej na dowolnej z replik. W związku z tym awaria pojedynczej repliki może zablokować zdolność systemu do wykonywania operacji silnych, aż do naprawienia usterki. Typowe silnie spójne systemy zreplikowane wykorzystujące *nie-blokujące* protokoły uzgadniania, takie jak Paxos [55], mogą tolerować awarie aż do połowy wszystkich replik i wciąż przetwarzać operacje. Tak więc niemożność tolerowania nawet pojedynczej awarii w systemie wysoko dostępnym, który powinien z łatwością tolerować awarie, wydaje się głęboko niezadowalająca, nawet jeżeli owa niemożność dotyczy jedynie operacji silnych.

Rozwiązania omówione powyżej w obliczu awarii idą na kompromis osłabiając postęp, albo operacji słabych (nie propagując wytworzonych przez nie zmian), albo operacji silnych (blokując ich wykonanie). Takie kompromisy wynikające z mieszania operacji słabych i silnych są warte zbadania. W szczególności interesującym pytaniem, na które próbujemy znaleźć odpowiedź w tej pracy, jest to czy istnieje system o spójności mieszanej, który obsługuje operacje silne w sposób nie-blokujący (tzn toleruje przynajmniej jakąś liczbę awarii replik), jednocześnie nie ograniczając postępu operacji słabych. Taki system posiadałby najlepsze cechy systemów ostatecznie spójnych i silnie spójnych. Mianowicie,

oferowałby wysoką dostępność i niskie czasy dostępu w przypadku operacji słabych, oraz silne gwarancje i (ograniczoną) tolerancję awarii w przypadku operacji silnych. Następnym nasuwającym się pytaniem jest to jakie gwarancje poprawności taki system może oferować.

Niezależnie od tego czy dany system wysoko dostępny posiada dodatkowe operacje silne czy nie, zapewnienie jego poprawnego działania w obliczu awarii jest krytycznie istotne. Systemy te są celowo projektowane pod kątem scenariuszy, w których w każdej chwili mogą wystąpić awarie. Może być zatem zaskakującym fakt, że większość prac dotyczących poprawności systemów wysoko dostępnych, które można znaleźć w literaturze, całkowicie abstrahuje od problemu awarii replik lub sieci (zobacz np. [45, 35, 46, 56, 57, 58, 59, 60, 36, 61, 48, 49]). Analizy wykonane w ten sposób mogą być uważane za niekompletne: protokół, który działa poprawnie tylko gdy awarie nie występują, niekoniecznie działa poprawnie gdy awarie *występują*. Z tego względu przeprowadzenie szerokiej, wyczerpującej analizy poprawności systemów wysoko dostępnych z jawnym uwzględnieniem różnorodnych modeli awarii, może dostarczyć nowych wglądów i wiedzy na temat występujących kompromisów, które dotychczas pozostawały niezauważone w środowisku naukowym jak i w przemyśle.

Cele i wkład pracy

Biorąc pod uwagę powyższe motywacje, w następujący sposób formułujemy główną tezę dysertacji:

> *Ograniczenia i kompromisy występujące w osiągalnych gwarancjach poprawności systemów wysoko dostępnych wynikające z operacji o mieszanej spójności, oraz wynikające z występowania awarii serwerów i sieci, mogą zostać formalnie oznaczone i można na ich temat wnioskować.*

Potwierdzamy prawdziwość tezy w dwóch częściach. Po pierwsze oznaczamy i analizujemy kompromisy dotyczące poprawności wynikające z operacji o mieszanej spójności. Po drugie wykorzystując podejście holistyczne do analizy poprawności, którego częścią jest stworzenie wiernego modelu rzeczywistych systemów opartych o architekturę klient-serwer oraz uwzględnienie w sposób jawny występowania awarii, precyzyjnie określamy ograniczenia w gwarancjach poprawności tego typu systemów wynikające z występowania awarii.

Poniżej podsumowujemy osiągnięty wkład naukowy naszej pracy.

Po pierwsze w rozdziale 2 definiujemy *przenikliwe typy chmurowe* (ang. *acute cloud types*, ACT), jako abstrakcję dla systemów o spójności mieszanej, które łączą najlepsze cechy systemów spójnych ostatecznie oraz systemów silnie spójnych. ACT posiadają dwa rodzaje operacji: *operacje słabe* nacelowowane na nieograniczoną skalowalność i niskie czasy odpowiedzi (jak operacje w CRDT), oraz *operacje silne* używane gdy gwarancje spójności ostatecznej są niewystar-

czające. Operacje silne wykorzystują nie-blokującą synchronizację na bazie rozproszonego konsensusu. Zaproponowaliśmy i przeanalizowaliśmy przykładowy ACT o nazwie *przenikliwy licznik nieujemny* (ang. *acute non-negative counter*, ANNC). Przeanalizowaliśmy również wpływowy system Bayou i pokazaliśmy jak można go usprawnić aby stał się ACT ogólnego przeznaczenia o nazwie AcuteBayou. Dzięki temu zidentyfikowaliśmy niepożądane anomalie, które mogą się pojawić w Bayou. W szczególności odkryliśmy anomalię, którą nazwaliśmy *tymczasową zamianą kolejności operacji*, która okazała się nieodzowną, nieusuwalną cechą systemu Bayou i innych, które w podobny sposób co Bayou, używają dwóch niekompatybilnych ze sobą sposobów szeregowania operacji silnych i słabych.

Następnie w rozdziale 3 wyprowadziliśmy system formalny (ang. formal framework), który umożliwia wnioskowanie na temat ACT i innych systemów o spójności mieszanej. W ramach tego systemu sformalizowaliśmy kilka kryteriów poprawności, w tym *oscylującą spójność ostateczną* (ang. *fluctuating eventual consistency*, FEC), która adekwatnie oddaje gwarancje poprawności oferowane przez systemy dla których tymczasowa zamiana kolejności operacji jest nieodzowna. Korzystając z naszego systemu formalnego udowodniliśmy również poprawność ANNC i AcuteBayou.

Potem w rozdziale 4 uogólniliśmy nasze spostrzeżenia dotyczące AcuteBayou i zaproponowaliśmy rezultat formalny ukazujący ograniczenia w możliwych do osiągnięcia gwarancjach poprawności. Udowodniliśmy, że systemy o mieszanej spójności, które łączą najlepsze cechy systemów spójnych ostatecznie oraz systemów silnie spójnych, tak jak ACT, i które cechują się dowolnie skomplikowaną semantyką operacji, nie mogą uniknąć tymczasowej zamiany kolejności operacji. Wobec tego, systemy te nie spełniają własności podstawowej spójności ostatecznej (ang. basic eventual consistency) dla operacji słabych. Zbadaliśmy potencjalnie występujące kompromisy w gwarancjach poprawności oraz cechach systemów o mieszanej spójności poprzez poddanie analizie innych rozwiązań o mieszanej spójności, które nie posiadają wszystkich pożądanych cech ACT.

Dalej w rozdziale 5 pokazaliśmy jak adekwatnie modelować rzeczywiste systemy wysoko dostępne o architekturze klient-serwer w celu wnioskowania na temat ich poprawności w obliczu występowania awarii. Nakreśliliśmy możliwe scenariusze awarii i sklasyfikowaliśmy je w sześciu modelach awarii (ang. failure models). Podaliśmy również model formalny (ang. formal framework), w którym można jawnie specyfikować awarie. Model ten umożliwia formułowanie *świadomych awarii kryteriów poprawności*.

Kolejno w rozdziale 6 wyraziliśmy w naszym modelu gwarancje sesji i przeanalizowaliśmy ich własności i znaczenie. Pokazaliśmy, że klasyczne gwarancje sesji mogą być kontrproduktywne w przypadku niektórych typów danych i podaliśmy nowy substytut w ich miejsce, o nazwie *zachowanie kontekstu*.

Na koniec w rozdziale 7 przeanalizowaliśmy gwarancje poprawności, które są możliwe do osiągnięcia w sześciu różnych modelach awarii. W szczególności pokazaliśmy, że podstawowa spójność ostateczna nie jest osiągalna gdy

występują permanentne podziały sieci lub awarie replik, po których niemożliwe jest odtworzenie ich stanu. Zdefiniowaliśmy ostateczne warianty dwóch kluczowych gwarancji sesji i rodzinę świadomych awarii kryteriów poprawności, które precyzyjnie oddają gwarancje poprawności osiągalne w rozważanych modelach awarii. Zidentyfikowaliśmy również kilka niepożądanych anomalii, które mogą być zaobserwowane przez klientów gdy występują awarie. Pokazaliśmy jak można przeciwdziałać występowaniu niektórych z nich.

W przyszłości planujemy zaprojektować nowe przenikliwe typy chmurowe. W szczególności takie, które mogą być zastosowane praktycznie, np. w magazynie danych NoSQL. Ponadto, chcielibyśmy zbadać, które grupy operacji mogą być zaimplementowane w ACT bez tymczasowej zamiany kolejności operacji.

Bibliografia

[1] M. Kokociński, T. Kobus, and P. T. Wojciechowski, "On mixing eventual and strong consistency: Acute cloud types," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, 2022.

[2] M. Kokociński, T. Kobus, and P. T. Wojciechowski, "Brief announcement: Eventually consistent linearizability," in *Proceedings of PODC '15: the 34th ACM Symposium on Principles of Distributed Computing*, July 2015.

[3] M. Kokociński, T. Kobus, and P. T. Wojciechowski, "Brief announcement: On mixing eventual and strong consistency: Bayou revisited," in *Proc. of PODC '19*, July 2019.

[4] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Introduction to transactional replication," in *Transactional Memory. Foundations, Algorithms, Tools, and Applications* (R. Guerraoui and P. Romano, eds.), vol. 8913 of *Lecture Notes in Computer Science*, Springer, 2015.

[5] P. T. Wojciechowski, T. Kobus, and M. Kokociński, "State-machine and deferred-update replication: Analysis and comparison," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, 2017.

[6] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Relaxing real-time order in opacity and linearizability," *Elsevier Journal on Parallel and Distributed Computing*, vol. 100, 2017.

[7] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Hybrid transactional replication: State-machine and deferred-update replication combined," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, July 2018.

[8] P. T. Wojciechowski, T. Kobus, and M. Kokociński, "Model-driven comparison of state-machine-based and deferred-update replication schemes," in *Proceedings of SRDS '12: the 31st IEEE International Symposium on Reliable Distributed Systems*, Oct. 2012.

[9] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Hybrid replication: State-machine-based and deferred-update replication schemes combined," in *Proceedings of ICDCS '13: the 33rd IEEE International Conference on Distributed Computing Systems*, July 2013.

[10] M. Kokociński, T. Kobus, and P. T. Wojciechowski, "Make the leader work: Executive deferred update replication," in *Proceedings of SRDS '14: the 33rd IEEE International Symposium on Reliable Distributed Systems*, Oct. 2014.

[11] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "The correctness criterion for deferred update replication," in *Program of TRANSACT '15: the 10th ACM SIGPLAN Workshop on Transactional Computing*, June 2015.

[12] T. Kobus, M. Kokociński, and P. T. Wojciechowski, "Jiffy: A lock-free skip list with batch updates and snapshots," in *Proceedings of PPoPP '22: the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Apr. 2022.

[13] P. T. Wojciechowski, T. Kobus, and M. Kokociński, "A fault-tolerant data processing computer system and method for implementing a distributed two-tier state machine," 2017. EPO patent no. EP 3193256 B1, July 7, 2017.

[14] P. T. Wojciechowski, T. Kobus, and M. Kokociński, "A fault-tolerant data processing computer system and method for implementing a distributed two-tier state machine," 2018. USPTO patent no. US 10135929 B2, Nov. 20, 2018.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, July 1978.

[16] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Comput. Surv.*, vol. 22, Dec. 1990.

[17] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg, *The primary-backup approach*. ACM Press/Addison-Wesley Publishing Co., 1993.

[18] E. A. Brewer, "Towards robust distributed systems (abstract)," in *Proc. of PODC '00: the 19th Annual ACM Symposium on Principles of Distributed Computing*, July 2000.

[19] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, Jan. 2009.

[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Operating Systems Review*, vol. 41, Oct. 2007.

[21] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Operating Systems Review*, vol. 44, Apr. 2010.

[22] ScyllaDB documentation, "Scylla." https://www.scylladb.com/.

[23] Basho documentation, "Riak key-value store." http://basho.com/products/riak-overview/.

[24] Project Voldemort, "Voldemort key-value store." https://www.project-voldemort.com/voldemort/.

[25] Netflix, "Netflix dynomite distributed dynamo layer." https://github.com/Netflix/dynomite.

[26] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Proc. of SSS '11*, May 2011.

[27] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of convergent and commutative replicated data types," Tech. Rep. 7506, Inria–Centre Paris-Rocquencourt; INRIA, 2011.

[28] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski, "Replicated data types: Specification, verification, optimality," in *Proc. of POPL '14: the 41st Symposium on Principles of Programming Languages (POPL)*, Jan. 2014.

[29] H. Attiya, S. Burckhardt, A. Gotsman, A. Morrison, H. Yang, and M. Zawirski, "Specification and complexity of collaborative text editing," in *Proc. of PODC '16*, 2016.

[30] N. M. Preguiça, C. Baquero, and M. Shapiro, "Conflict-free replicated data types," *Computing Research Repository*, vol. abs/1805.06358, 2018.

[31] Amazon DynamoDB documentation, "Consistent reads in DynamoDB." https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html, 2019.

[32] Microsoft Azure documentation, "Consistency levels in Cosmos DB." https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels, 2019.

[33] Apache Cassandra documentation, "Light weight transactions in Cassandra." https://docs.datastax.com/en/cql/3.3/cql/cql_using/useInsertLWT.html, 2019.

[34] Basho documentation, "Consistency levels in Riak." https://docs.basho.com/riak/kv/2.2.3/developing/app-guide/strong-consistency, 2019.

[35] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Proceedings of ODSI '12: the 10th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2012.

[36] C. Li, N. Preguiça, and R. Rodrigues, "Fine-grained consistency for geo-replicated systems," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, July 2018.

[37] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood, "Cloud types for eventual consistency," in *Proceedings of the 26th European Conference on Object-Oriented Programming*, June 2012.

[38] S. Burckhardt, D. Leijen, J. Protzenko, and M. Fähndrich, "Global sequence protocol: A robust abstraction for replicated shared state.," in *Proc. of ECOOP '15*, July 2015.

[39] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS: Yahoo!'s hosted data serving platform," *Proc. of VLDB Endowment*, vol. 1, no. 2, 2008.

[40] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in *Proc. of SOSP '13*, Nov. 2013.

[41] M. Milano and A. C. Myers, "Mixt: a language for mixing consistency in geodistributed transactions," *ACM SIGPLAN Notices*, vol. 53, no. 4, 2018.

[42] V. Gavrielatos, A. Katsarakis, V. Nagarajan, B. Grot, and A. Joshi, "Kite: Efficient and available release consistency for the datacenter," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Feb. 2020.

[43] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Queue*, vol. 11, Mar. 2013.

[44] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser, "Managing update conflicts in Bayou, a weakly connected replicated storage system," in *Proceeding of SOSP '95: the 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[45] M. Serafini, D. Dobre, M. Majuntke, P. Bokor, and N. Suri, "Eventually linearizable shared objects," in *Proc. of PODC '10*, July 2010.

[46] R. Guerraoui and E. Ruppert, "A paradox of eventual linearizability in shared memory," in *Proc. of PODC '14*, July 2014.

[47] S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv, "Eventually consistent transactions," in *Proceedings of the 21st European Conference on Programming Languages and Systems*, Mar. 2012.

[48] S. Burckhardt, A. Gotsman, and H. Yang, "Understanding eventual consistency," Tech. Rep. MSR-TR-2013-39, Microsoft Research, Mar. 2013.

[49] S. Burckhardt, "Principles of eventual consistency," *Foundations and Trends in Programming Languages*, vol. 1, Oct. 2014.

[50] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers*, vol. C-28, Sept. 1979.

[51] C. H. Papadimitriou, "The serializability of concurrent database updates," *Journal of the ACM*, vol. 26, no. 4, 1979.

[52] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, 1990.

[53] Apache Cassandra Issues (Jira), "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`, 2016.

[54] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, "Providing high availability using lazy replication," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, Nov. 1992.

[55] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, 1998.

[56] S. Liu, M. Rahman, S. Skeirik, I. Gupta, and J. Meseguer, "Formal modeling and analysis of cassandra in maude," in *Formal Methods and Software Engineering* (S. Merz and J. Pang, eds.), vol. 8829 of *LNCS*, Springer International Publishing, 2014.

[57] A. Bouajjani, C. Enea, and J. Hamza, "Verifying eventual consistency of optimistic replication systems," in *Proc. of POPL '14*, Jan. 2014.

[58] H. Attiya, F. Ellen, and A. Morrison, "Limitations of highly-available eventually-consistent data stores," in *Proc. of PODC '15: the 34th ACM Symposium on Principles of Distributed Computing*, July 2015.

[59] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro, "'Cause I'm strong enough: Reasoning about consistency choices in distributed systems," in *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Jan. 2016.

[60] H. Attiya, F. Ellen, and A. Morrison, "Limitations of highly-available eventually-consistent data stores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, Jan. 2017.

[61] A. Girault, G. Gößler, R. Guerraoui, J. Hamza, and D. Seredinschi, "Monotonic prefix consistency in distributed systems," in *Proc. of FORTE '18*, June 2018.