Бесконфликтная синхронизация в распределенных системах

CRDT

Разрешай конфликты лучше, чем Cassandra

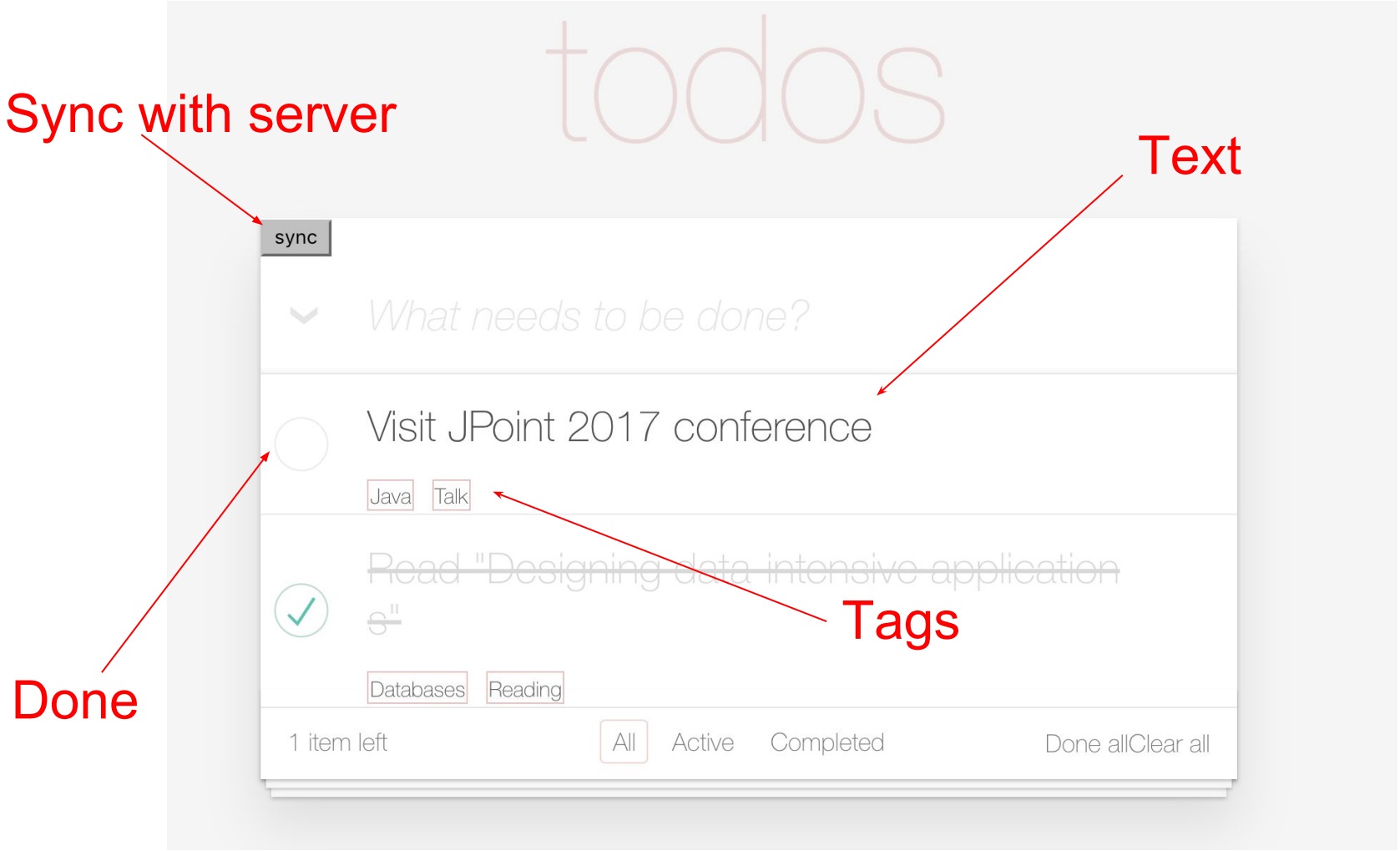
Андрей Ершов
andrershov@gmail.com
@andrershov

О докладчике

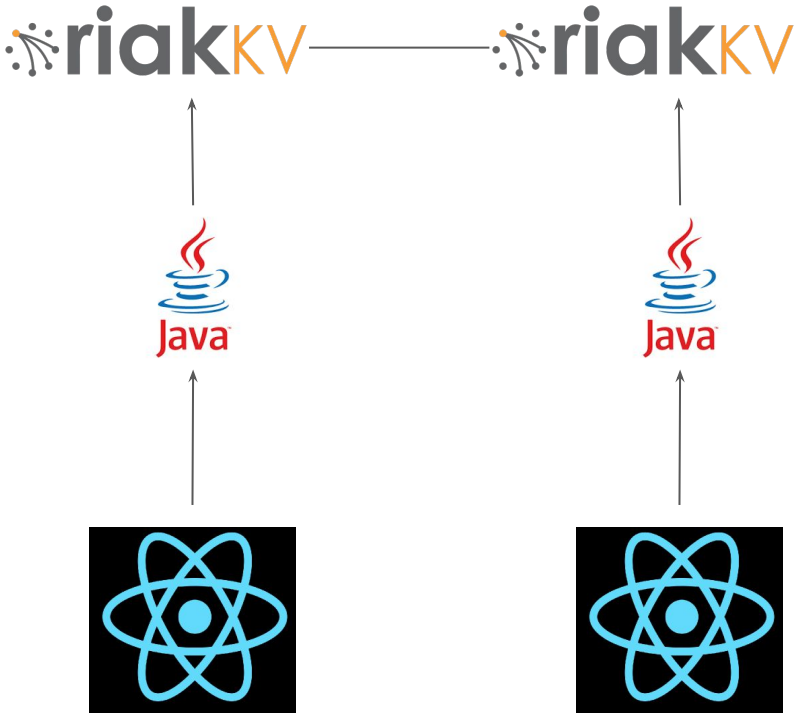
- 8+ лет программирования на Java
- Desktop, Android, Enterprise, Distributed Systems
- Lead в проекте видео-конференций
- Последний год подсел на чтение papers по распределенным системам

О чем доклад?

- ALPS (Available, Low Latency, Partition Tolerant, Scalable) Systems
- Конкурентные модификации неизбежны
- LWW не работает
- CRDT работают
- И даже поддерживаются БД
- Доклад - это обзор! Много ссылок и домашней работы


Sync with server
todos
Text
sync
What needs to be done?
Visit JPoint 2017 conference
Java
Talk
Read "Designing data-intensive applications"
Tags
Databases
Reading
Done
1 item left
All
Active
Completed
Done all
Clear all

Архитектура



Хранение

- Riak - key-value storage
- Будем хранить все items по одному ключу!

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

Хранение

- Для избежания конфликтов храним каждый item отдельно
- Для указания принадлежности к конкретному списку используем secondary indexes
- Перед модификацией тэгов, читаем текущее состояние из кэша

https://github.com/andrershov/riakcrdt

DEMO

Быстрый отклик

- Асинхронная репликация
- Оптимистичный UI

Сетевые проблемы

- Сетевая сегментация
- Оффлайн режим работы

Конкурентные изменения

INRIA
ROCQUENCOURT

Conflict-free Replicated Data Types *

Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal
Marek Zawirski, INRIA & UPMC, Paris, France

Thème COM — Systèmes communicants
Projet Regal

Rapport de recherche n° 7687 — Juillet 2011 — 18 pages

https://hal.inria.fr/inria-00609399v1/document

Eventual Consistency

Как только все ноды получили все операции, *то через некоторое время*, они придут в одно и то же состояние

Strong Eventual Consistency

Как только все ноды получили все операции, *то сразу,* они придут в одно и то же состояние - есть локальная функция conflict resolver

CRDT

1) Структуры данных
2) Обладают свойством SEC
3) Решение проблемы конкурентных изменений
4) Выполняют автоматическое разрешение конфликтов!

Структуры данных?

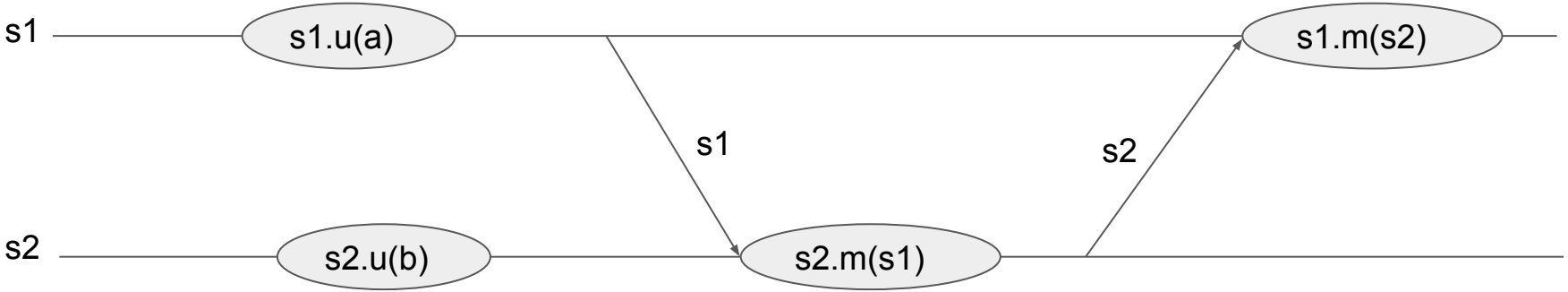
1) Counter
2) Register
3) Flag
4) List
5) Set
6) Map
7) Graph
8) ...

Типы CRDT

1) State based
2) Operation based
3) Delta

State based CRDT

При изменении состояния объекта передается все состояние

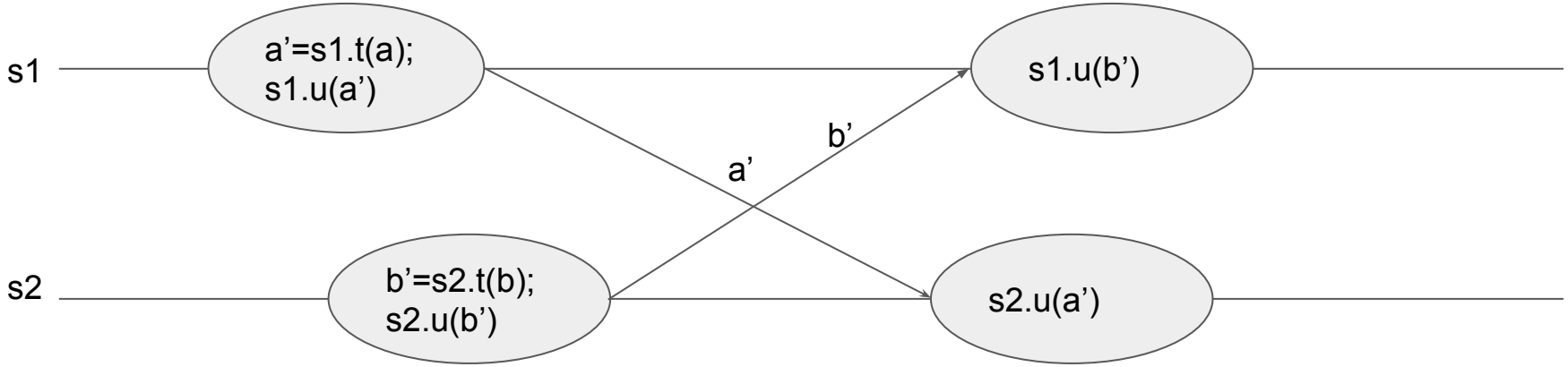


State based CRDT

1) Большой размер сообщения
2) Требования к транспорту минимальны:
   a) Потеря сообщений
   b) Дублирование сообщений
   c) Переупорядочивание сообщений

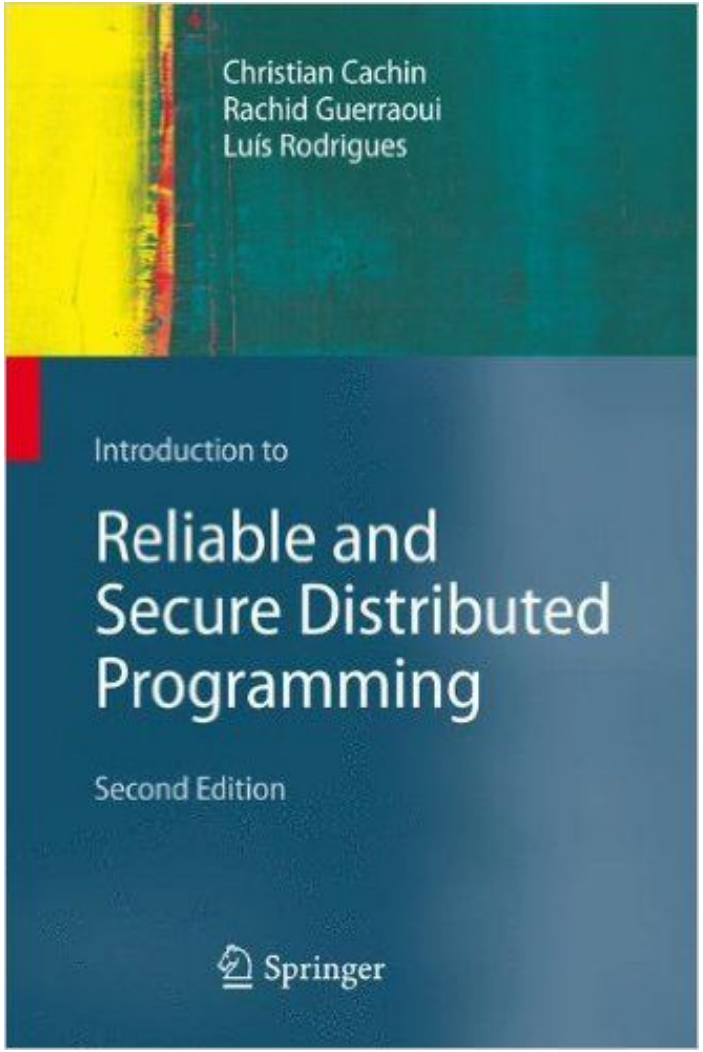
Op-based CRDT

1) Для каждой операции мутации две функции - t (prepare-update), u(effect-update)
2) a’ = s.t(a) (prepare-update has no side-effects)
3) По сети передается a’
4) s.u(a’) (effect-update mutates state)



Op-based CRDT

1) Маленький размер сообщения
2) Требования к транспорту - causal reliable broadcast


Christian Cachin
Rachid Guerraoui
Luís Rodrigues
Introduction to
Reliable and Secure Distributed Programming
Second Edition
Springer

Op-based CRDT

1) Подходит для несложных топологий.
2) Обычно подразумевает хранение op-log’a.

SwiftCloud

Making Operation-based CRDTs Operation-based

Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker

HASLab/INESC TEC and Universidade do Minho, Portugal

Abstract. Conflict-free Replicated Datatypes (CRDT) can simplify the design of eventually consistent systems. They can be classified into state-based or operation-based. Operation-based designs have the potential for allowing very compact solutions in both the sent messages and the object state size. Unfortunately, the current approaches are still far from this objective. In this paper, we introduce a new 'pure' operation-based framework that makes the design and the implementation of these CRDTs more simple and efficient. We show how to leverage the meta-data of the messaging middleware to design very compact CRDTs, while only disseminating operation names and their optional arguments.

http://haslab.uminho.pt/ashoker/files/opbaseddais14.pdf

Delta-CRDT

1) Требования к сети такие же как у state-based*
2) Размер сообщений такой же как у op-based*

Delta State Replicated Data Types

Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero

HASLab/INESC TEC and Universidade do Minho, Portugal

Abstract. CRDTs are distributed data types that make eventual consistency of a distributed object possible and non ad-hoc. Specifically, state-based CRDTs ensure convergence through disseminating the entire state, that may be large, and merging it to other replicas; whereas operation-based CRDTs disseminate operations (i.e., small states) assuming an exactly-once reliable dissemination layer. We introduce *Delta State Conflict-Free Replicated Data Types* (δ-CRDT) that can achieve the best of both worlds: small messages with an incremental nature, as in operation-based CRDTs, disseminated over unreliable communication channels, as in traditional state-based CRDTs. This is achieved by defining *δ-mutators* to return a *delta-state*, typically with a much smaller size than the full state, that to be joined with both local and remote states. We introduce the δ-CRDT framework, and we explain it through establishing a correspondence to current state-based CRDTs. In addition, we present an anti-entropy algorithm for eventual convergence, and another one that ensures causal consistency. Finally, we introduce several δ-CRDT specifications of both well-known replicated datatypes and novel datatypes, including a generic map composition.

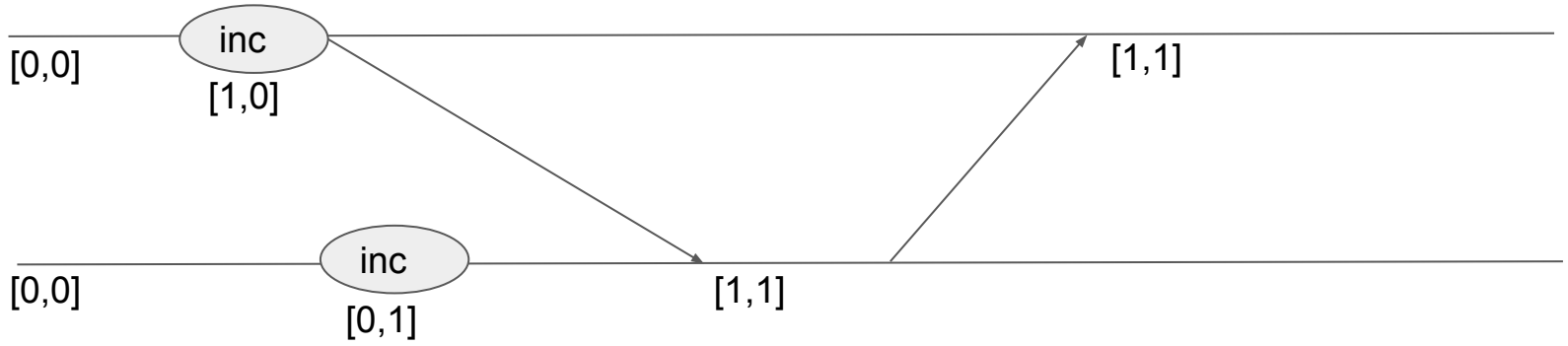
https://arxiv.org/pdf/1603.01529v1.pdf

State-based CRDTs

004
5

GCounter

1) GCounter = Grow-Only Counter
2) Для p(1)...p(n) процессов состояние = [x(1),...x(n)]
3) Каждый процесс инкрементирует свое значение
4) Операция merge(curr, recv) берет максимум для каждого компонента



PNCounter

1) PNCounter = Positive-Negative Counter

2) Одного массива не достаточно!

3) Возьмем два массива - один для сложений другой для вычитаний!

PNCounter = GCounter + GCounter

Counters. Сложности на практике

1) Размер массива == Количество процессов
2) Что является процессами? - клиенты
3) Если клиентов много, размер массива стремится в космос

Riak Counters

1) Состояние счетчика хранится в базе данных
2) Размер массива счетчика равен replication factor
3) Клиенты БД ничего не знают об их индивидуальном значении счетчика
4) Клиенты меняют значение счетчика, выполняя операции inc и dec
5) Эти операции не идемпотенты!

Counters

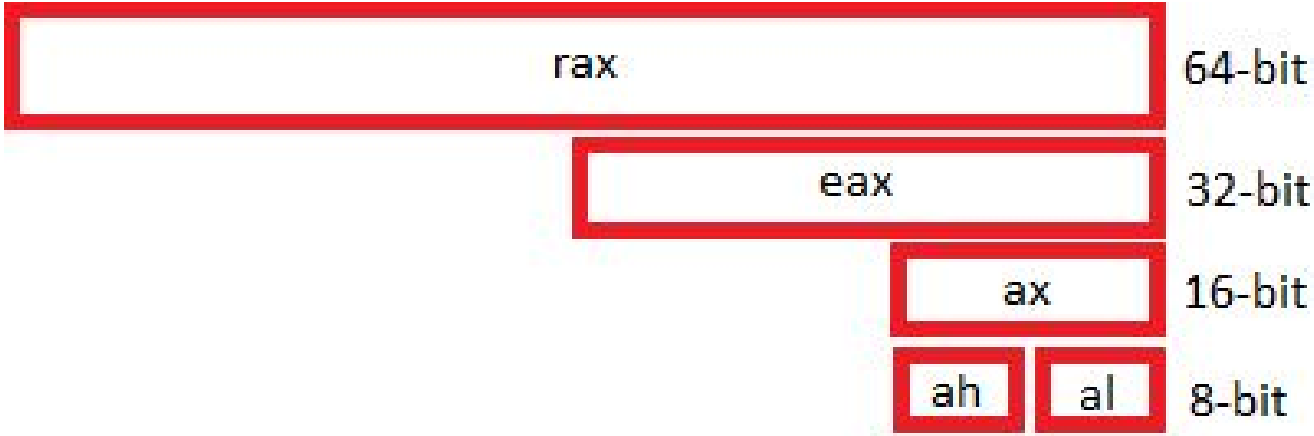
Riak 2 supports counters in the same way as other CRDTs. Counters are basically an integer you can increment or decrement.

CAVEAT: in error cases, there's no way to tell if a counter increment has happened or not. If you don't retry a counter increment, it may or may not have incremented. If you do retry a counter increment, it may be incremented once or more than once.

Counters. Что делать?

1) Если клиентов не много, использовать в лоб
2) Использовать Set вместо Counter
3) Пренебречь не-идемпотентными инкрементами

https://www.youtube.com/watch?v=YQGmJt_UP0E


rax
64-bit
eax
32-bit
ax
16-bit
ah
al
8-bit

LWWRegister

1) LWWRegister = Last-Write Wins Register
2) Каждую запись будем снабжать timestamp’ом!
3) У кого timestamp больше - тот и победил!
4) Да, это тоже CRDT.
5) Правда не самый лучший...

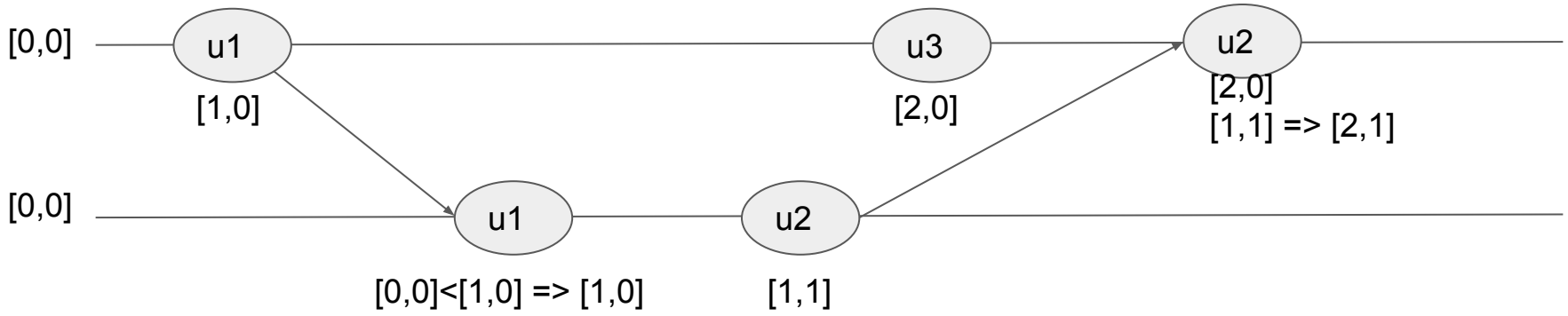
Cassandra LWW reading

1) www.datastax.com/dev/blog/why-cassandra-doesnt-need-vector-clocks
2) https://aphyr.com/posts/299-the-trouble-with-timestamps
3) https://aphyr.com/posts/294-jepsen-cassandra

Version Vector

1) Задача: сравнить версии объекта и детектировать конкурентные изменения состояния.

2) Для n процессов version vector это массив [x(1),...x(n)]. Каждый процесс при выполнении операции увеличивает значение своей компоненты.



Version Vector

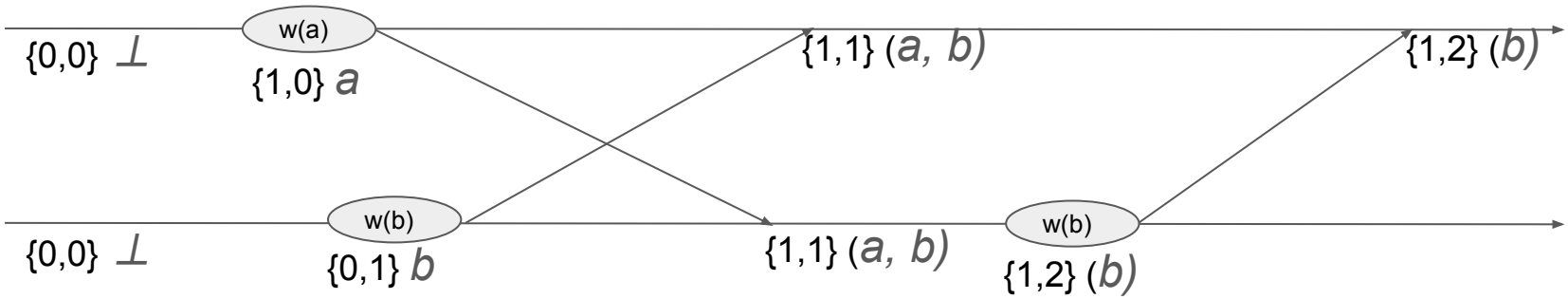
1) A = [2,1] vs B = [3, 1]. B > A. B учитывает все события, которые есть в A, плюс еще какое то.
2) A = [2, 1] vs B = [2, 1]. B >= A. B учитывает все события, которые есть в A.
3) A = [2,1] vs B = [1,2]. B *concurrent with* A. В каждом векторе есть события, неучтенные в другом => конфликт.
4) A = [2,1] merge B=[1,2] = [2,2]

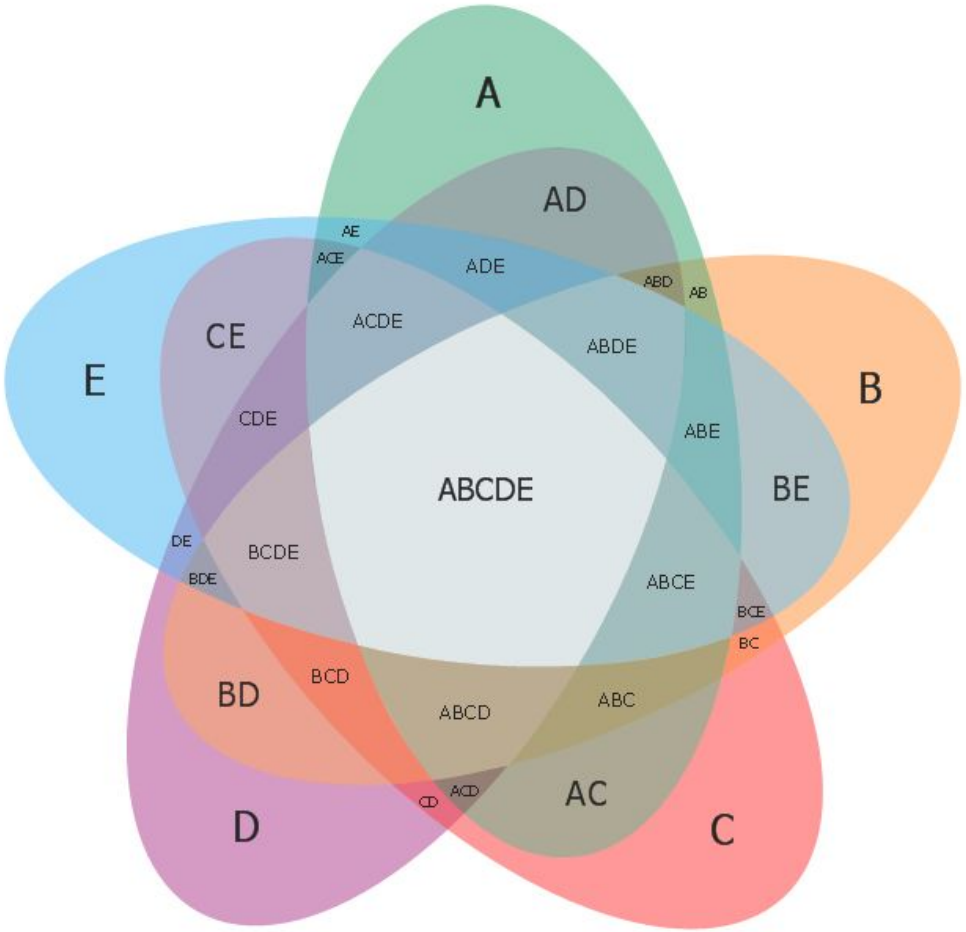
Version Vector. Reading

1) basho.com/posts/technical/why-vector-clocks-are-easy/
2) basho.com/posts/technical/why-vector-clocks-are-hard/
3) basho.com/posts/technical/vector-clocks-revisited/
4) basho.com/posts/technical/vector-clocks-revisited-part-2-dotted-version-vectors/
5) https://haslab.wordpress.com/2011/07/08/version-vectors-are-not-vector-clocks/

MVRegister

1) Для каждой записи в регистр, увеличиваем version vector.
2) При получении записи v_recv с vv_recv:
   a) Если vv_recv >= vv_curr. V_res = v_recv, vv_res = vv_recv.
   b) Если vv_curr >= vv_recv. V_res = v_curr, vv_res = vv_curr.
   c) Если vv_recv concurrent with vv_curr. V_res += v_curr, vv_res = merge(vv_recv, vv_curr)




A
AD
AE
ACE
ADE
ABD
AB
CE
ACDE
ABDE
E
B
CDE
ABE
ABCDE
BE
DE
BCDE
BDE
ABCE
BCE
BC
BCD
BD
ABCD
ABC
ACD
CD
AC
D
C

GSet

1) GSet = Grow-Only Set
2) Это обычный set! Но из него нельзя удалять.
3) Операция merge - это операция объединения множеств друг с другом!

2PSet

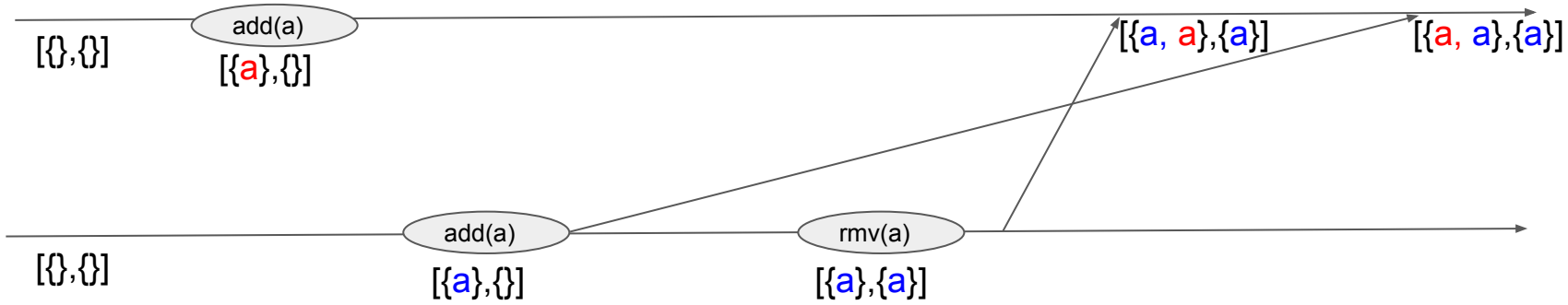
1) 2PSet - 2-Phase Set
2) 2PSet = G-Set (для добавлений) + G-Set (для удалений)
3) Если элемент есть в add set, но его нет в remove set, значит, элемент есть. Иначе, его нет.
4) Проблемы:
   a) Сплошной мусор
   b) Элементы нельзя повторно добавлять в 2PSet

Чем CRDT-Set отличается от обычного Set?

1) В sequential set операции строго упорядочены. История линейна.
2) В CRDT set есть только частичный порядок операций.
3) Если операции коммутативны, то особой разницы нет. Например, add(a), remove(b) = remove(b), add(a).
4) add(a), remove(a) != remove(a), add(a)
5) На уровне структуры нужно принять решение!
6) AWSet, RWSet

OR-Set

1) OR-Set - Observed-Remove Set (Add-Wins)
2) То же, что и 2P-Set, но позволяет выполнять повторное добавление
3) Каждый элемент пометим уникальным (для этого элемента) тэгом. Удаление удаляет только элемент с увиденным тэгом - observed-remove.



AWMap

1) AWMap - Add-Wins Map
2) В качестве значения любой CRDT!
3) Позволяет создавать вложенные структуры
4) AWMap<TaskId, AWMap<PropName, CRDT>>
   a) “Done” - EWFlag
   b) “Text” - MVRegister<String> (почти правда)
   c) “Tags” - AWSet<String>

The problem with embedded CRDT counters and a solution

Carlos Baquero
HASLab, INESC TEC &
Universidade do Minho
Braga, Portugal
cbm@di.uminho.pt

Paulo Sérgio Almeida
HASLab, INESC TEC &
Universidade do Minho
Braga, Portugal
psa@di.uminho.pt

Carl Lerche
Portland, Oregon
me@carllerche.com

https://arxiv.org/pdf/1011.5808.pdf

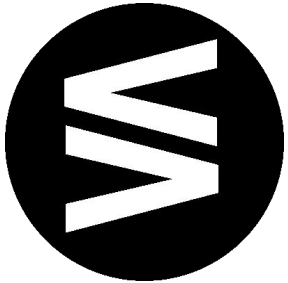
A Conflict-Free Replicated JSON Datatype

Martin Kleppmann and Alastair R. Beresford

Abstract—Many applications model their data in a general-purpose storage format such as JSON. This data structure is modified by the application as a result of user input. Such modifications are well understood if performed sequentially on a single copy of the data, but if the data is replicated and modified concurrently on multiple devices, it is unclear what the semantics should be. In this paper we present an algorithm and formal semantics for a JSON data structure that automatically resolves concurrent modifications such that no updates are lost, and such that all replicas converge towards the same state. It supports arbitrarily nested list and map types, which can be modified by insertion, deletion and assignment. The algorithm performs all merging client-side and does not depend on ordering guarantees from the network, making it suitable for deployment on mobile devices with poor network connectivity, in peer-to-peer networks, and in messaging systems with end-to-end encryption.

Index Terms—CRDTs, Collaborative Editing, P2P, JSON, Optimistic Replication, Operational Semantics, Eventual Consistency.

https://arxiv.org/pdf/1011.5808.pdf



SwiftCloud

Server-side databases

Riak

Pros:

- DynamoDb style
- DVV conflicts detection
- Early CRDT adopter

Cons:

- Non-reactive

Akka Distributed Data

Pros:

- Delta CRDT support
- Reactive

Cons:

- Replicated Storage based on Akka Cluster
- No sharding

AntidoteDB

Pros:

- Op-based CRDTs
- Multiple DCs support
- Causal Consistency (COPS, Eiger, GentleRain inspired)
- HA transaction support

Cons:

- Research project

Client-server databases

http://tonsky.me/blog/the-web-after-tomorrow/

Swarm.js

Pros:

- Op-based CRDTs (based on partially ordered op-log)
- JavaScript, Java libraries

Cons:

- Poor documentation
- Not ready
- Sharding is not supported
- ACL are not supported

SwiftCloud

Pros:

- CRDT support
- Transactions on the client

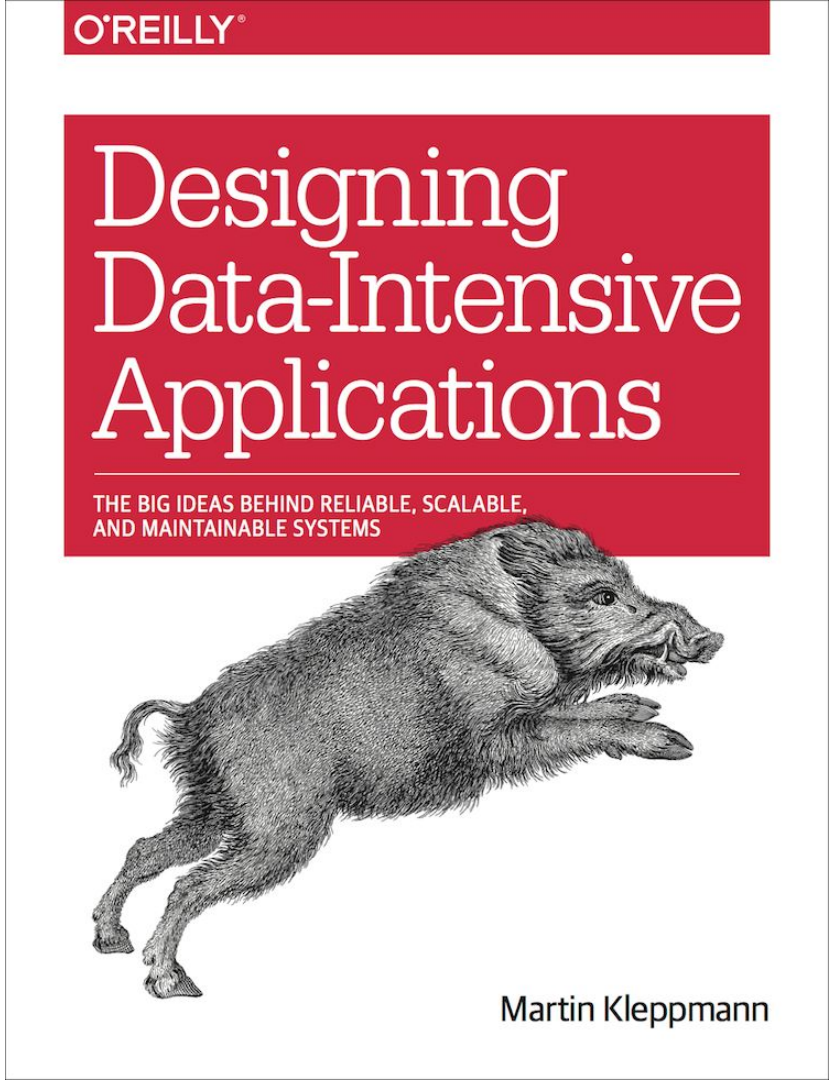
Cons:

- Research
- No sharding
- No ACLs

SwiftCloud

Выводы

1) Asynchronous replication, optimistic UI, partition tolerance, offline-work => конкурентные изменения
2) Конкурентные изменения приводят к конфликтам
3) CRDT - структуры данных для разрешения конфликтов
4) Нам нужны клиент-серверные БД с поддержкой CDRT


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann

Andrey Ershov

andrershov@gmail.com
@andrershov

DINS

Дискуссионная зона

1) Кто использует CDRT?
2) Обсуждение вашей любимой базы данных
3) CRDT и Eventual Consistency
4) Что такое полурешетка и причем здесь CDRT?
5) Почему LWW не работает
6) Как решить проблему мусора в ORSet?
7) Causal Reliable Broadcast
8) Чем отличается vector clock от version vector?

Discussion zone slides

Кто использует CRDT?

TOMTOM

bet365

ROVIO

RIOT
GAMES
LEAGUE of LEGENDS

basho

ANGRY BIRDS

riak

TRIFORK.
...think software

Tapjoy

SOUNDCLOUD

HELIUM

akka

PROJECTFIFO

CRDT и Eventual Consistency

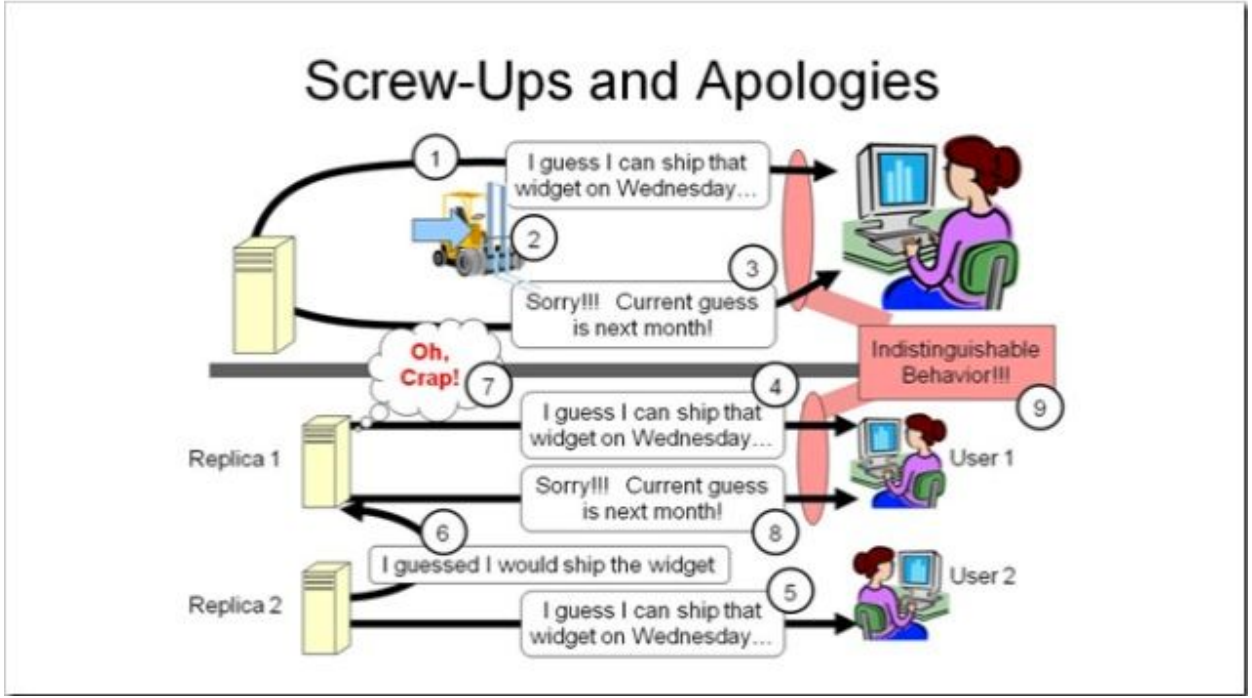
Достаточно ли нам eventual consistency?

1) Есть задачи, которые требуют координации
2) Пусть у нас есть склад с книгами
3) И две сегментированные реплики
4) Осталась только одна книга
5) Два пользователя заказывают эту книгу, потому что реплики не разговаривают друг с другом

Memories, Guesses, and Apologies – PatHelland's WebLog

https://blogs.msdn.microsoft.com/pathelland/2007/.../memories-guesses-and-apologies...

Pat Helland May 15, 2007 5 ... All computing can be considered as: "**Memories**, **Guesses**, and **Apologies**". This is a personal ... Screw-Ups and **Apologies**.



Альтернативный подход

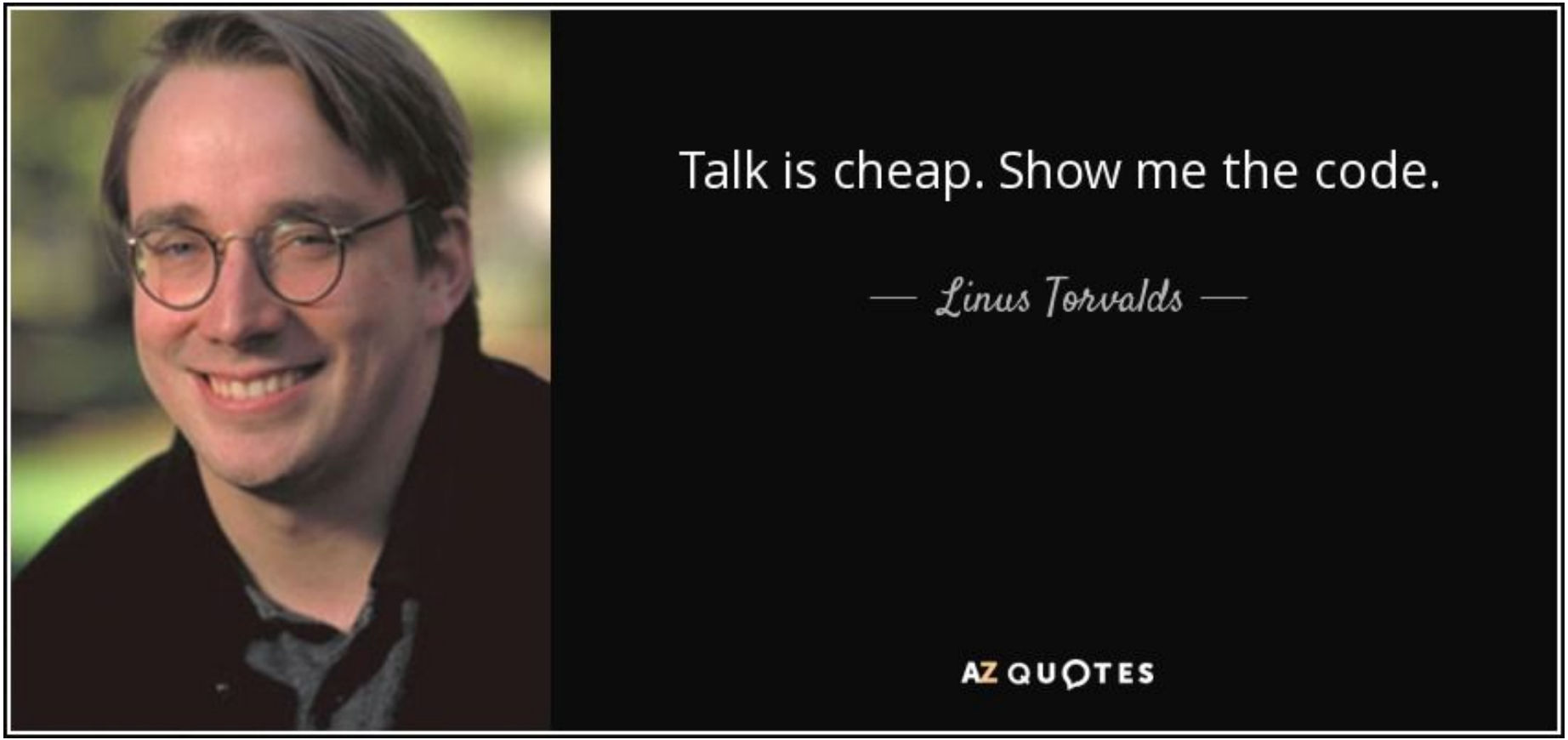
1) Использовать системы хранения с multiple consistency models
2) Использовать strong consistency только для операций, которые этого требуют

Invariant stability

1) Модель системы: causal consistency и операции коммутативны (CRDT!)
2) Выполнение каждой операции требует соблюдения инварианта
3) Доказано

Если для каждой пары операций из множества операций, их конкурентное выполнение не может нарушить инвариант системы, тогда выполнение всех операций не требует координации.

3) Проведя CISE (Cause I’m Strong Enough) анализ можно найти операции, которые требуют координации.


Talk is cheap. Show me the code.
— Linus Torvalds —
AZ QUOTES

```
@XPR("Int balance")
@XPR(value = "balance >= 0", type = XPR.Type.INVARIANT)
@Op(Account.Deposit.class)
@Op(Account.Debit.class)
public class Account extends AnnotatedSchema {

    @XPR(value = {"Int amount", "Int balance" }, type = XPR.Type.ARGUME
    @XPR(value = "amount >= 0", type = XPR.Type.PRECONDITION)
    @XPR(value = "balance := balance + amount", type = XPR.Type.EFFECT)
    public static class Deposit extends AnnotatedOperation { }

    @XPR(value = {"Int amount", "Int balance" }, type = XPR.Type.ARGUME
    @XPR(value = {" amount >= 0 "}, type = XPR.Type.PRECONDITION)
    @XPR(value = "balance := balance - amount", type = XPR.Type.EFFECT)
    public static class Debit extends AnnotatedOperation { }
}
```

’Cause I’m Strong Enough: Reasoning about Consistency Choices in Distributed Systems

Alexey Gotsman
IMDEA Software Institute, Spain

Hongseok Yang
University of Oxford, UK

Carla Ferreira
NOVA LINCS, DI, FCT,
Universidade NOVA de Lisboa, Portugal

Mahsa Najafzadeh
Sorbonne Universités, Inria,
UPMC Univ Paris 06, France

Marc Shapiro
Sorbonne Universités, Inria,
UPMC Univ Paris 06, France

http://dl.acm.org/citation.cfm?id=2837625

Extending Eventually Consistent Cloud Databases for Enforcing Numeric Invariants

Valter Balegas, Diogo Serra, Sérgio Duarte
Carla Ferreira, Rodrigo Rodrigues, Nuno Preguiça
NOVA LINCS/FCT/Universidade Nova de Lisboa

Marc Shapiro, Mahsa Najafzadeh
INRIA / LIP6

https://arxiv.org/pdf/1503.09052.pdf

Dotted Version Vectors: Logical Clocks for Optimistic Replication

Nuno Preguiça
CITI/DI
FCT, Universidade Nova de Lisboa
Monte da Caparica, Portugal
nmp@di.fct.unl.pt

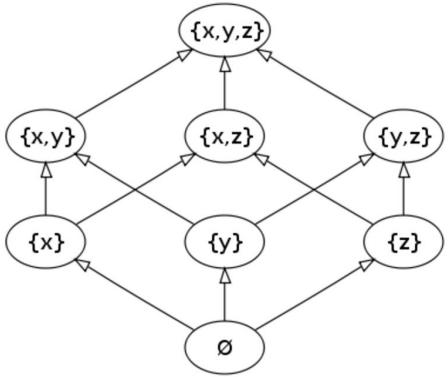
Carlos Baquero, Paulo Sérgio Almeida,
Victor Fonte, Ricardo Gonçalves
CCTC/DI
Universidade do Minho
Braga, Portugal
{cbm,psa,vff}@di.uminho.pt, rtg@lsd.di.uminho.pt

https://arxiv.org/pdf/1011.5808.pdf

CRDT и полурешетка

Свойства функции merge

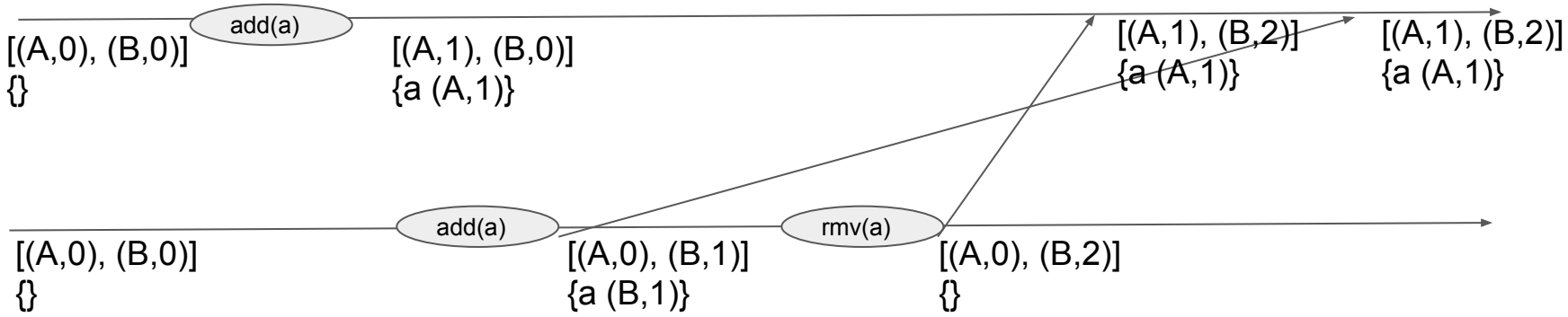
1) State-based CDRT образуют верхнюю полурешетку
2) Верхняя полурешетка - частично упорядоченное множество с операцией merge со следующими свойствами:
   a) m = merge(x, y), m - точная верхняя грань
   b) merge(x, x) = x - идемпотентность
   c) merge(x, y) = merge(y, x) - коммутативность
   d) merge(x, merge(y, z)) = merge(merge(x, y), z) - ассоциативность



Проблема мусора в множестве

ORWOT-Set

1) ORWOT-Set - Observed-Remove Set w/o tombstones!
2) Решение проблемы мусора.
3) В качестве version vector используем dotted version vector.
4) В качестве тэга используем dot.



Causal order broadcast

Broadcast 101. Модель системы

- p(1), …, p(n)
- Процессы посылают друг другу сообщения m(1), m(2), …
- Процесс *доставляет* сообщение, когда абстракция broadcast передает сообщение более высокому уровню приложения

Broadcast 101. Unreliable broadcast

p(i) посылает сообщение - некоторые процессы доставят это сообщение

Подходит для State-based CRDT!

Broadcast 101. Reliable broadcast

- Если процесс p(i) послал сообщение m, то все процессы доставят это сообщение.
- Для нескольких сообщений нет гарантии порядка доставки.

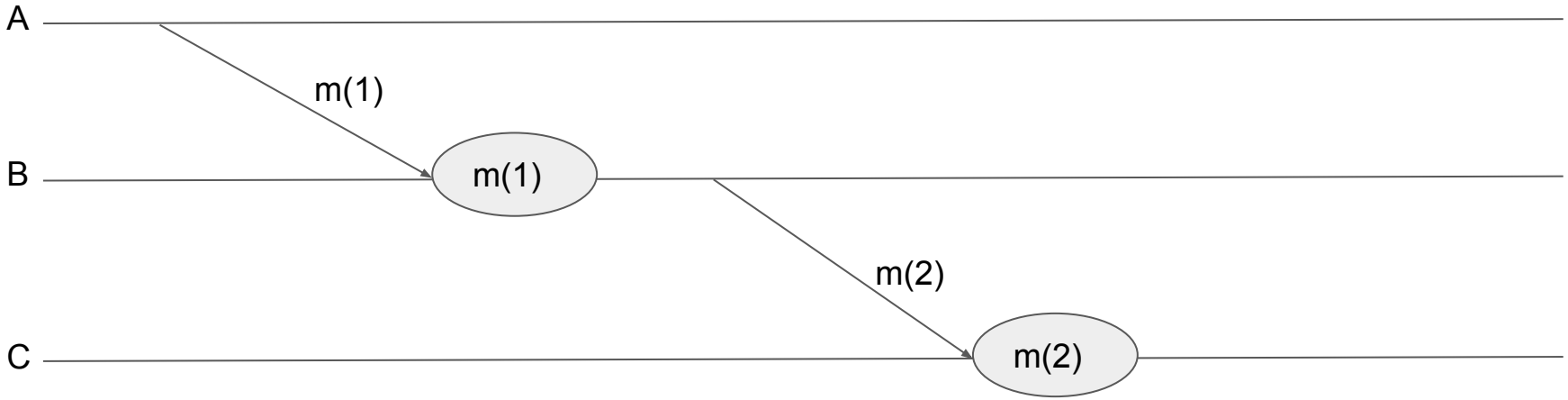
Broadcast 101. FIFO broadcast

- Все сообщения отправленные процессом p(i), будут доставлены остальными процессами в порядке отправки.
- Нет никаких ограничений на глобальный порядок следования сообщений.

Broadcast 101. Global order broadcast

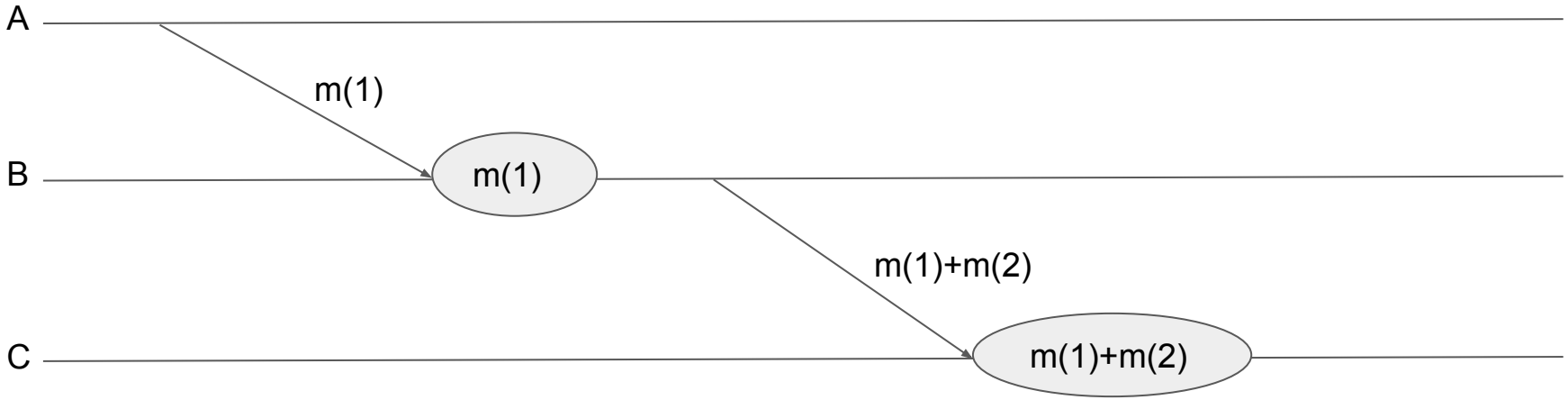
1) Все процессы посылают сообщения.
2) Все процессы доставляют сообщения в одном глобальном порядке.
3) Задача аналогична задаче консенсуса. Требует координации.

Broadcast 101. Causal broadcast



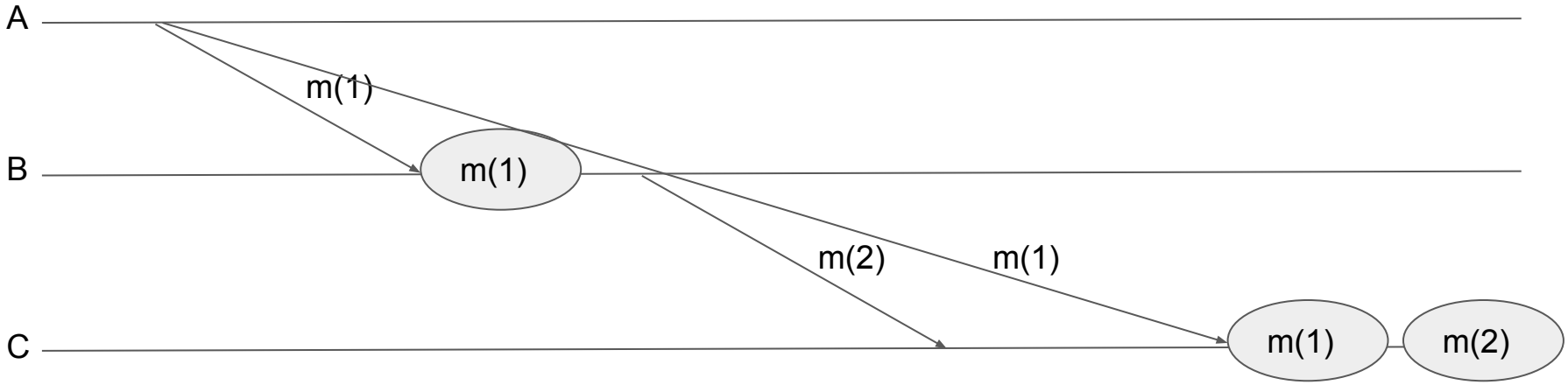
He causal broadcast!

Broadcast 101. Causal broadcast



Каждое сообщение включает в себя все предыдущие!

Broadcast 101. Causal broadcast



Нужно понять, что нужно ждать