PaPoC'23 Test of Time
"Bridging the Gap: Opportunities in Coordination-Avoiding Databases"

Presented by:
Peter Bailis, Sisu Data
Joe Hellerstein, UC Berkeley

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

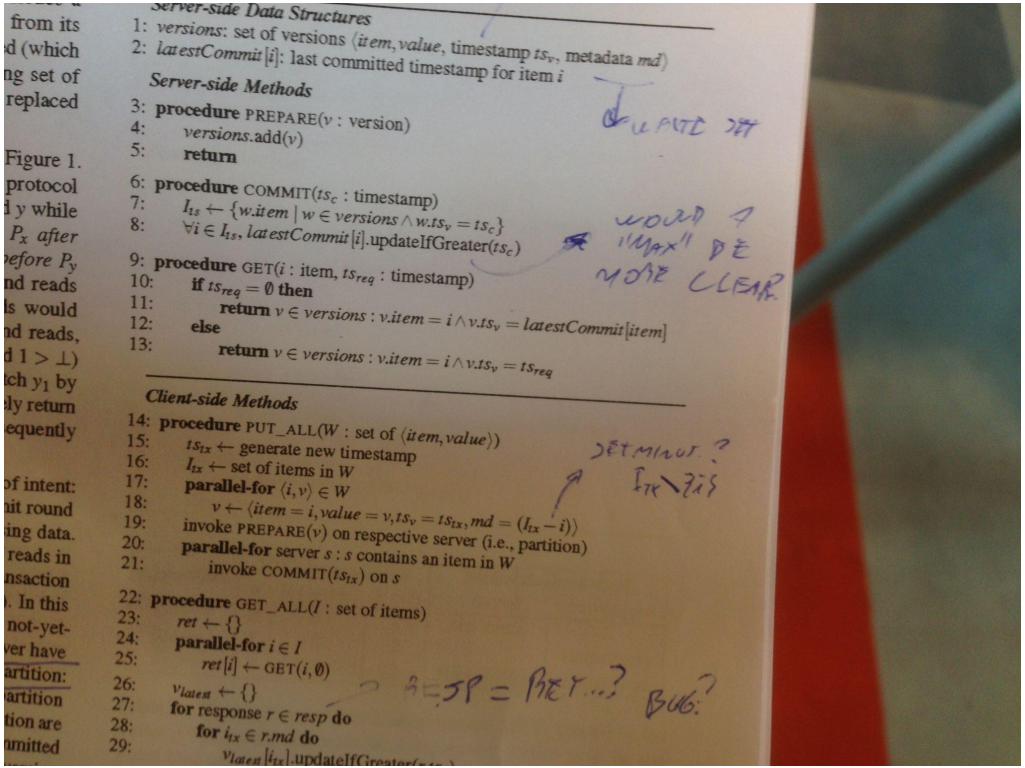
9 years: What we've learned

Lessons for the PaPoC community

PB


EUROSYS 2014
AMSTERDAM, THE NETHERLANDS
PaPEC 02-A06 2nd floor
WRSC 04-A04 4th floor
Euro-TM 06-A04 6th floor
SFMA 08-A04 8th floor
EuroDW 12-A36 12th floor
EuroSec 12-A33 12th floor
CloudDP 14-A33 14th floor


Overview
1 Introduction
2 Background
3 Solution
4 Current and Future Work
5 References
PAPERS



Server-side Data Structures

1: *versions*: set of versions $\langle item, value, \text{timestamp } ts_v, \text{metadata } md \rangle$
2: *latestCommit*$[i]$: last committed timestamp for item i

Server-side Methods

3: **procedure** PREPARE(v : version)
4: *versions*.add(v)
5: **return**

6: **procedure** COMMIT(ts_c : timestamp)
7: $I_{ts} \leftarrow \{w.item \mid w \in versions \wedge w.ts_v = ts_c\}$
8: $\forall i \in I_{ts}$, *latestCommit*$[i]$.updateIfGreater(ts_c)

9: **procedure** GET(i : item, ts_{req} : timestamp)
10: **if** $ts_{req} = \emptyset$ **then**
11: **return** $v \in versions : v.item = i \wedge v.ts_v = latestCommit[item]$
12: **else**
13: **return** $v \in versions : v.item = i \wedge v.ts_v = ts_{req}$

Client-side Methods

14: **procedure** PUT_ALL(W : set of $\langle item, value \rangle$)
15: $ts_{tx} \leftarrow$ generate new timestamp
16: $I_{tx} \leftarrow$ set of items in W
17: **parallel-for** $\langle i, v \rangle \in W$
18: $v \leftarrow \langle item = i, value = v, ts_v = ts_{tx}, md = (I_{tx} - i) \rangle$
19: invoke PREPARE(v) on respective server (i.e., partition)
20: **parallel-for** server $s : s$ contains an item in W
21: invoke COMMIT(ts_{tx}) on s

22: **procedure** GET_ALL(I : set of items)
23: $ret \leftarrow \{\}$
24: **parallel-for** $i \in I$
25: $ret[i] \leftarrow$ GET($i, \emptyset$)
26: $v_{latest} \leftarrow \{\}$
27: **for** response $r \in resp$ **do**
28: **for** $i_{tx} \in r.md$ **do**
29: $v_{latest}[i_{tx}]$.updateIfGreater(


Renaissance of EC?
...but embrace is not unanimous!

"Designing applications to cope with concurrency anomalies in their data is very error-prone, time-consuming, and ultimately not worth the performance gains."

F1: A Distributed SQL Database That Scales

Jeff Shute
Chad Whipkey
David Menestrina
Radek Vingralek
Eric Rollins
Stephan Ellner
Traian Stancescu
Bart Samwel
Mircea Oancea
John Cieslewicz
Himani Apte
Ben Handy
Kyle Littlefield
Ian Rae*

Google, Inc.
*University of Wisconsin-Madison

ABSTRACT

F1 is a distributed relational database system built at Google to support the AdWords business. F1 is a hybrid database that combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and usability of traditional SQL databases. F1 is built on Spanner, which provides synchronous cross-datacenter replication and strong consistency. Synchronous replication implies higher commit latency, but we mitigate that latency by using a hierarchical schema model with structured data types and through smart application design. F1 also includes a fully functional distributed SQL query engine and automatic change tracking and publishing.

1. INTRODUCTION

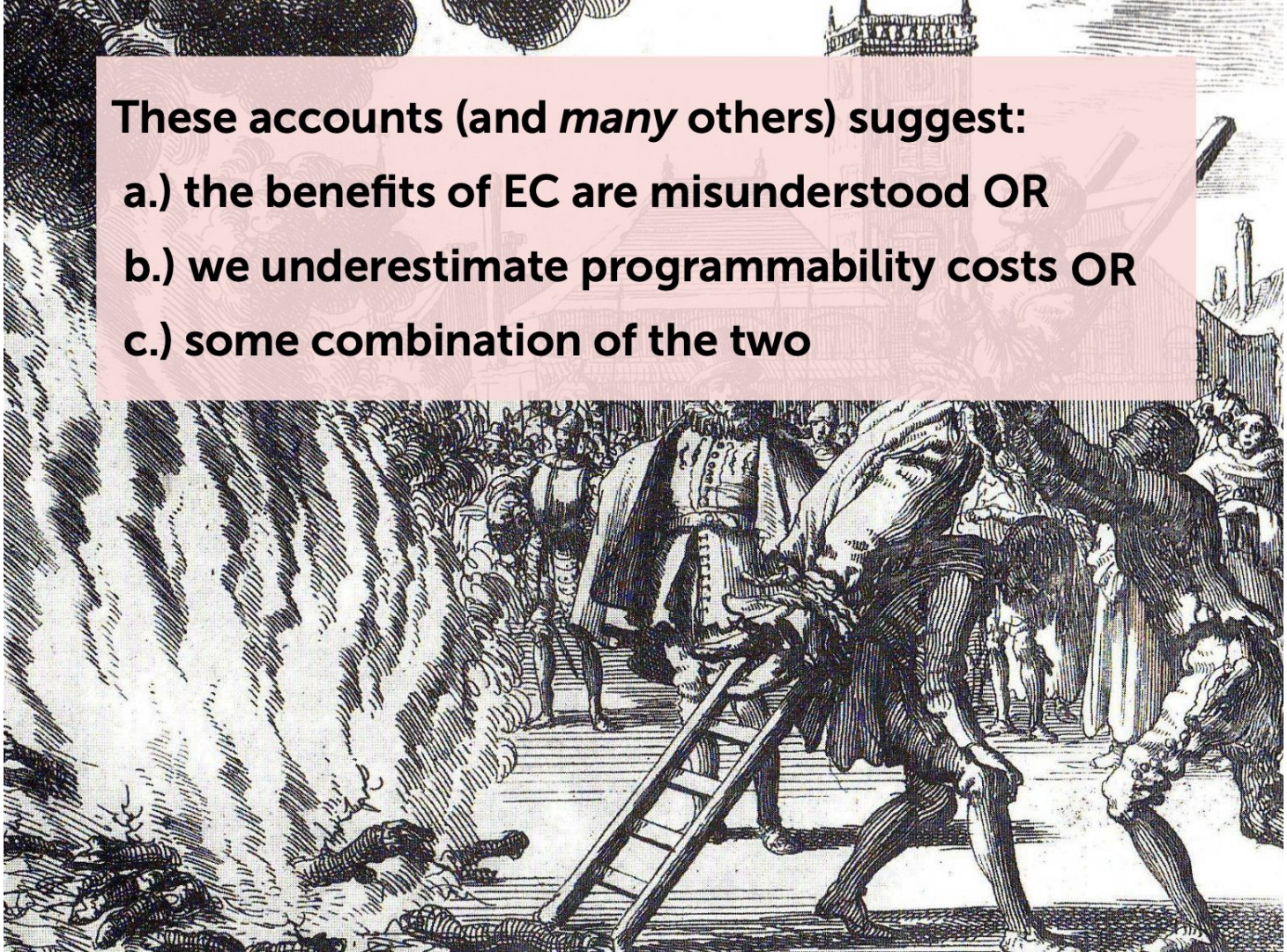
F1[1] is a fault-tolerant globally-distributed OLTP and OLAP database built at Google as the new storage system

consistent and correct data.
Designing applications to cope with concurrency anomalies in their data is very error-prone, time-consuming, and ultimately not worth the performance gains.

4. **Usability**: The system must provide full SQL query support and other functionality users expect from a SQL database. Features like indexes and ad hoc query are not just nice to have, but absolute requirements for our business.

Recent publications have suggested that these design goals are mutually exclusive [5, 11, 23]. A key contribution of this paper is to show how we achieved all of these goals in F1's design, and where we made trade-offs and sacrifices. The name F1 comes from genetics, where a *Filial 1 hybrid* is the first generation offspring resulting from a cross mating of distinctly different parental types. The F1 database system

VLDB 2013



These accounts (and *many* others) suggest:

a.) the benefits of EC are misunderstood OR

b.) we underestimate programmability costs OR

c.) some combination of the two

Questions in the Air, c. 2010

Pushing Further on 20th-Century, Storage-Centric Thinking:

1. What does it mean that transactions “don’t scale”?
2. How can we formalize weaker isolation levels?

Transitioning to 21st-Century Application-Centric Thinking:

3. What does eventual consistency mean for *program outcomes*?
4. When is coordination strictly required?

JMH

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

Talk Outline

PaPEC 2014: A look back

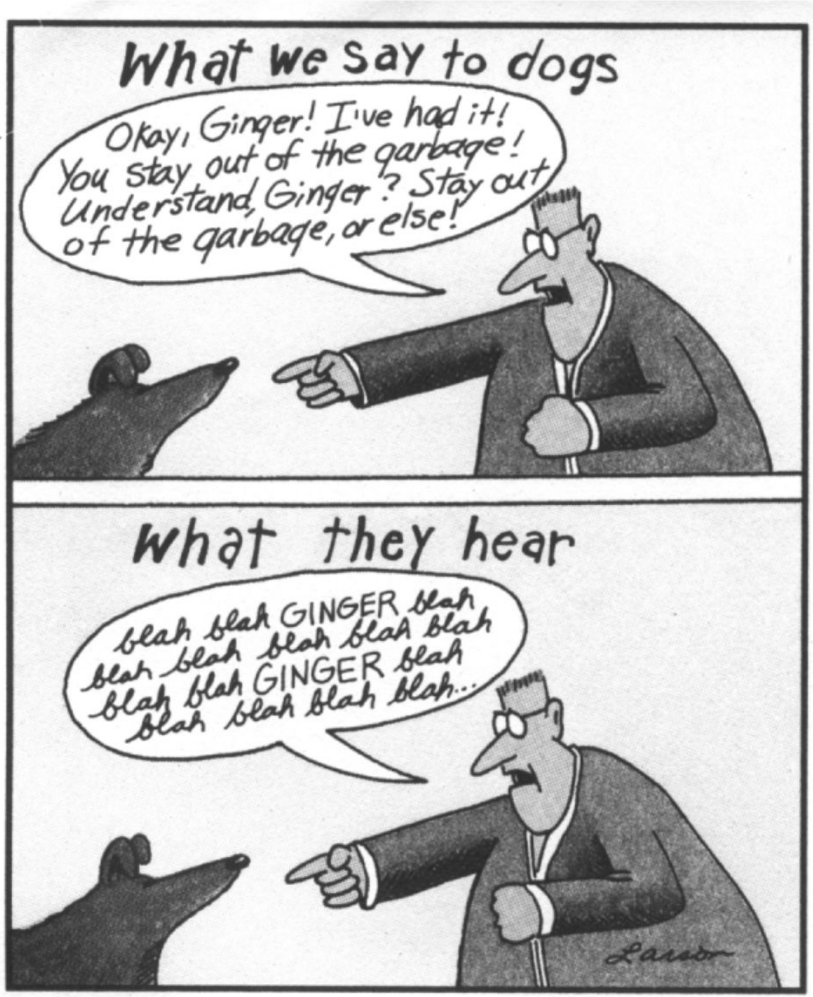
Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

Our PoV: It's all about the application

PB



The Far Side,
Gary Larson

"post on timeline"

"accept friend request"

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

CALM: CONSISTENCY AS LOGICAL MONOTONICITY

Theorem (CALM): A distributed program has a consistent, coordination-free distributed implementation if and only if it is monotonic.

Hellerstein JM. The Declarative Imperative: Experiences and conjectures in distributed logic. *ACM PODS* Keynote, June 2010 *ACM SIGMOD Record,* Sep 2010.

Ameloot TJ, Neven F, Van den Bussche J. Relational transducers for declarative networking. *JACM*, Apr 2013.

Ameloot TJ, Ketsman B, Neven F, Zinn D. Weaker forms of monotonicity for declarative networking: a more fine-grained answer to the CALM-conjecture. *ACM TODS,* Feb 2016.

Hellerstein, JM, Alvaro, P. Keeping CALM: When Distributed Consistency is Easy. C*ACM,* Sept, 2020.

JMH

CACM September 2020
http://bit.ly/calm-cacm

DOI:10.1145/3369736

In distributed systems theory, CALM presents a result that delineates the frontier of the possible.

BY JOSEPH M. HELLERSTEIN AND PETER ALVARO

Keeping CALM: When Distributed Consistency Is Easy

DISTRIBUTED SYSTEMS ARE tricky. Multiple unreliable

across many machines. Most scientific computing and machine learning systems work in parallel across multiple processors. Even legacy desktop operating systems and applications like spreadsheets and word processors are tightly integrated with distributed backend services.

The challenge of building correct distributed systems is increasingly urgent, but it is not new. One traditional answer has been to reduce this complexity with *memory consistency* guarantees—assurances that accesses to memory (heap variables, database keys, and so on) occur in a controlled fashion. However, the mechanisms used to enforce these guarantees—*coordination protocols*—are often criticized as barriers to high performance, scale, and availability of distributed systems.

The high cost of coordination. Coordination protocols enable autonomous, loosely coupled machines to jointly decide how to control basic behaviors, including the order of access to shared memory. These protocols are among the most clever and widely cited ideas in distributed computing. Some well-known techniques include the Paxos[33] and Two-Phase Commit (2PC)[25,34] protocols, and global barriers underly-

» key insights

- **Coordination is often a limiting factor in system performance. While sometimes necessary for consistent outcomes,**

Intuitively…

- Consistency: **same app outcome everywhere** regardless of NW shenanigans.
- Monotonicity: The **outcomes only grow** as inputs grow.

  Early outcomes are part of the final outcome! Stream without regret!

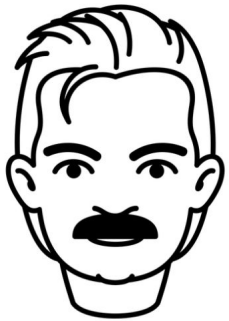
$$f\ monotone \triangleq$$
$$X \subseteq Y \rightarrow f(X) \subseteq f(Y)$$

- Coordination: Messages we **must wait for even though we have all the data.**

CALM

Monotonicity ⇔ Coordination-free Consistency

Database users express correctness criteria via database constraints



"usernames should be unique"

"account balances should remain positive"

"there should only be one administrator"

PB

"no duplicate users"

CAN WE USE CONSTRAINTS TO AVOID COORDINATION?

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

"no duplicate users"

CAN WE USE CONSTRAINTS TO AVOID COORDINATION?

WHAT THE APPLICATION SAYS

WHAT THE DATABASE HEARS

constraint

constraint

constraint

"no duplicate users"

constraint

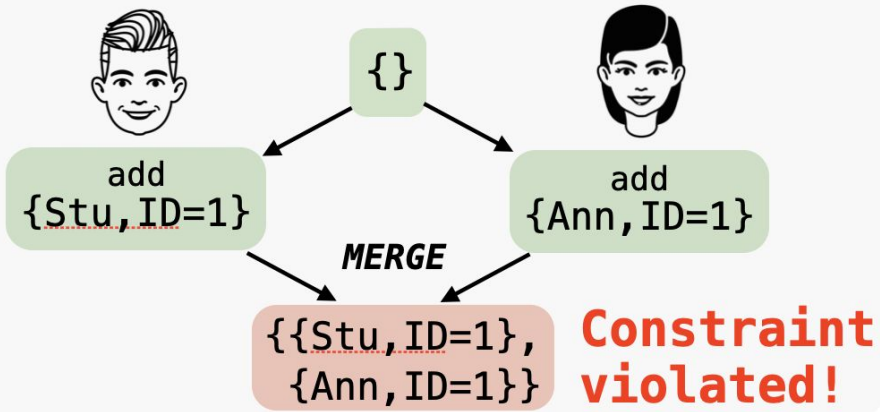
constraint

constraint

Key idea: Check if constraints can be violated by "merging" independent operations

ICT: Invariant Confluence Test

CONSTRAINT: User IDs are unique
OPERATION: Add users
MERGE: Set union



Key idea: Check if constraints can be violated by "merging" independent operations

ICT: Invariant Confluence Test

OUR CONTRIBUTION:

ICT $\iff$ safe, coordination-free execution possible

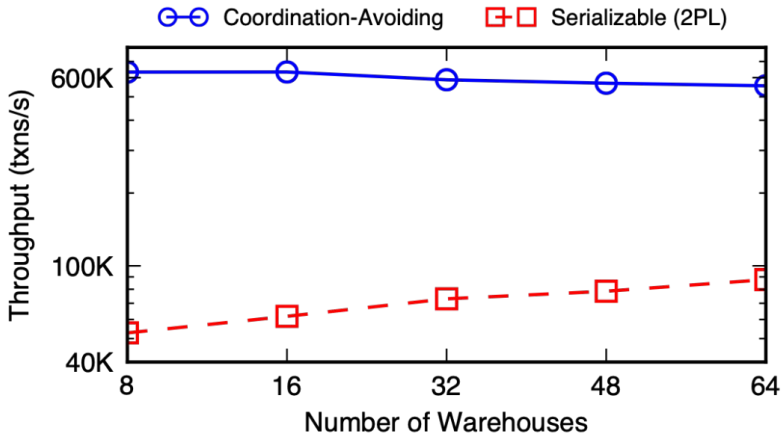
Theorem. A globally I-valid system can execute a set of transactions T with coordination-freedom, transactional availability, and convergence *if and only if* T are I-confluent with respect to I.

Generalizes classic partitioning-based indistinguishability arguments

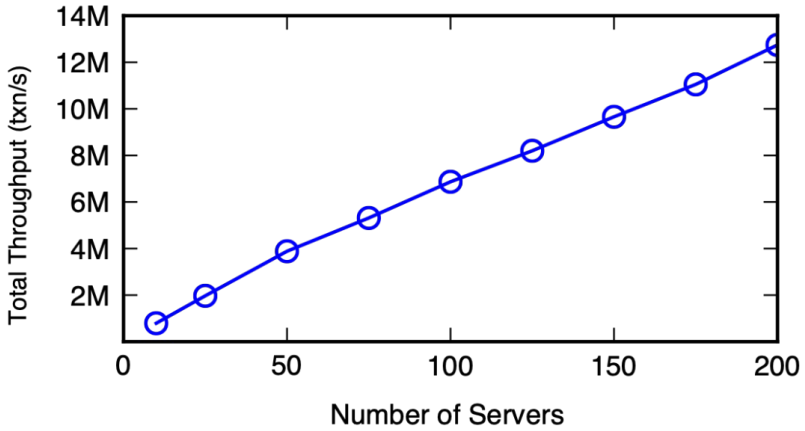
[VLDB 2015]

TPC-C 14/16 CONSTRAINTS PASS ICT

6-11x faster than ACID/serializability



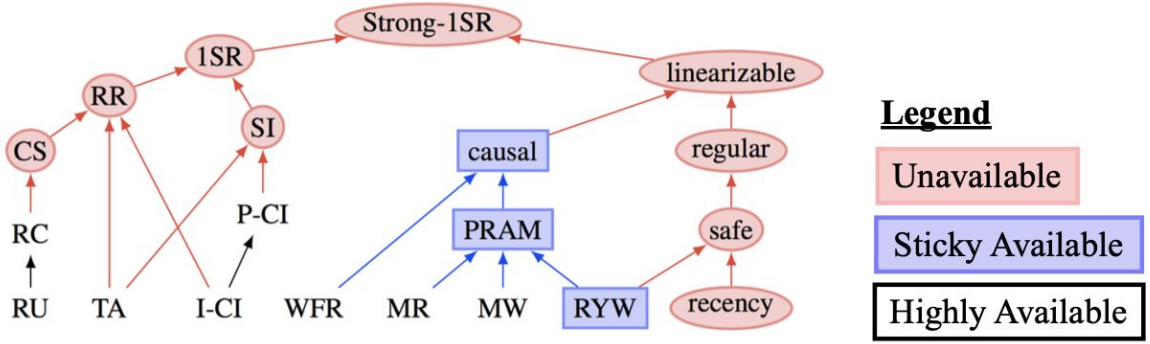
scale to over **25x** best listed result



Also: applied ideas back to transactional isolation!

Implementations typically use coordination (e.g. locking)

But is this fundamental to the isolation levels? **NO!**



Highly Available	Read Uncommitted (RU), Read Committed (RC), Monotonic Atomic View (MAV), Item Cut Isolation (I-CI), Predicate Cut Isolation (P-CI), Writes Follow Reads (WFR), Monotonic Reads (MR), Monotonic Writes (MW)
Sticky Available	Read Your Writes (RYW), PRAM, Causal
Unavailable	Cursor Stability (CS)†, Snapshot Isolation (SI)†, Repeatable Read (RR)†‡, One-Copy Serializability (1SR)†‡, Recency⊕, Safe⊕, Regular⊕, Linearizability⊕, Strong 1SR†‡⊕

prevents lost update†
prevents write skew‡
requires recency guarantees⊕

"Highly Available Transactions: Virtues and Limitations" VLDB 2014

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

Progress on all fronts!

App-centric:

- CRDT libraries: AutoMerge, Yjs.
- Research languages: Dedalus, Bloom, LVars, Lasp, Gallifrey, Hydro

RW-centric:

- Datacenter networks have gotten much faster
- NoSQL systems provide transactions
- Lots of work on scaling transactions, Paxos, etc.

idea: focus here solely on industry? then on next slide point out there's more research to do?

(On the App-centric side, CRDTs have been used in PayPal, League of Legends, SoundCloud, supported by Enterprise Redis, Riak.)

If so maybe format this slide and the next uniformly so the transition is clear.

Coordination remains a persistent challenge

CRDTs can be footguns

e.g. Keep CALM and CRDT On @ VLDB 2023

Developers still don’t add transactions to every line of code (and transactions are often underused)

e.g., ACIDRain @ SIGMOD 2017, Feral Concurrency Control @ SIGMOD 2015

Geo-distributed latency is still an issue, and especially at scale

from its
d (which
ng set of
replaced

Figure 1.
protocol
i y while
P_x after
before P_y
nd reads
ls would
nd reads,
d $1 > \perp$)
tch y_1 by
ly return
equently

of intent:
nit round
ing data.
reads in
nsaction
). In this
not-yet-
ver have
artition:
artition
tion are
nmitted

Server-side Data Structures

1: *versions*: set of versions $\langle item, value$, timestamp ts_v, metadata $md\rangle$
2: *latestCommit*$[i]$: last committed timestamp for item i

Server-side Methods

3: **procedure** PREPARE(v : version)
4: *versions*.add(v)
5: **return**

6: **procedure** COMMIT(ts_c : timestamp)
7: $I_{ts} \leftarrow \{w.item \mid w \in versions \wedge w.ts_v = ts_c\}$
8: $\forall i \in I_{ts}$, *latestCommit*$[i]$.updateIfGreater(ts_c)

9: **procedure** GET(i : item, ts_{req} : timestamp)
10: **if** $ts_{req} = \emptyset$ **then**
11: **return** $v \in versions : v.item = i \wedge v.ts_v = latestCommit[item]$
12: **else**
13: **return** $v \in versions : v.item = i \wedge v.ts_v = ts_{req}$

Client-side Methods

14: **procedure** PUT_ALL(W : set of $\langle item, value\rangle$)
15: $ts_{tx} \leftarrow$ generate new timestamp
16: $I_{tx} \leftarrow$ set of items in W
17: **parallel-for** $\langle i, v\rangle \in W$
18: $v \leftarrow \langle item = i, value = v, ts_v = ts_{tx}, md = (I_{tx} - i)\rangle$
19: invoke PREPARE(v) on respective server (i.e., partition)
20: **parallel-for** server s : s contains an item in W
21: invoke COMMIT(ts_{tx}) on s

22: **procedure** GET_ALL(I : set of items)
23: $ret \leftarrow \{\}$
24: **parallel-for** $i \in I$
25: $ret[i] \leftarrow$ GET($i, \emptyset$)
26: $v_{latest} \leftarrow \{\}$
27: **for** response $r \in resp$ **do**
28: **for** $i_{tx} \in r.md$ **do**
29: $v_{latest}[i_{tx}]$.updateIfGreater(

VLDB 2021 Best Industry Paper Award!

RAMP-TAO:

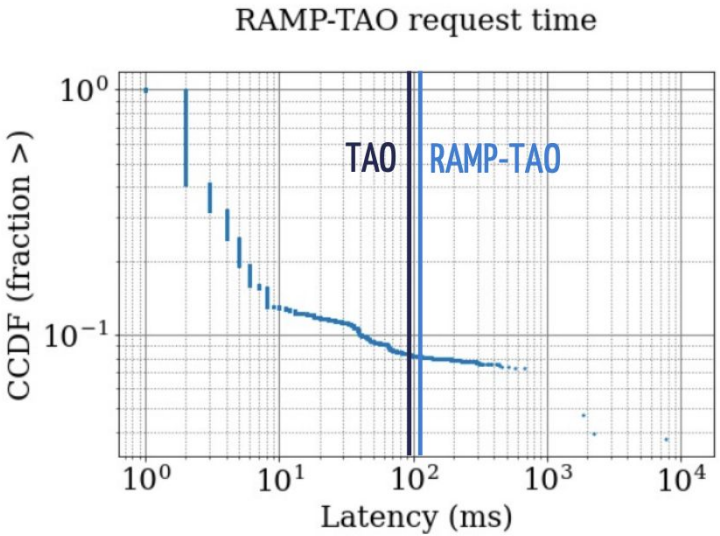
Layering Atomic Transactions on Facebook's Online TAO Data Store

Audrey Cheng (UC Berkeley),
Xiao Shi, Lu Pan, Anthony Simpson, Neil Wheaton, Shilpa Lawande (Facebook),
Nathan Bronson (Rockset),
Peter Bailis (Sisu Data),
Natacha Crooks, Ion Stoica (UC Berkeley)

We find: 1 in every 1,500 batched reads exhibit fractured reads anomalies…
…but coordination-avoiding protocols are light enough to work in practice!

RAMP-TAO can be feasibly deployed in production

- 99th percentile latency (114ms) comparable to TAO's (105ms)
- >99.9% of read transactions complete in one round
- 0.42% increase in memory overhead



Coordination remains a persistent challenge

Looking forward, reasons to expect even more load

- e.g., LLM agents could easily 10x transaction workloads
- Interplanetary networks are an inevitability…

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

Talk Outline

PaPEC 2014: A look back

Coordination Avoidance: Approach and results

9 years: What we've learned

Lessons for the PaPoC community

A Coordinated Call to Action

Goal: Deliver on the lessons of Application-Level semantics

Learning Lessons 1: CRDTs

- No footguns for reads! Needs to be correct under composition, too.

Learning Lesson 2: Languages like Bloom, Lasp, Gallifrey

- Needs to be approachable. Evolution, not Revolution.
- Performance matters. Target state-of-the-art speed.

Vision: A Programming Stack for Distributed Concerns

- A type system + compiler stack for fast, correct, robust distributed code
- Underlying a variety of surface languages / frameworks (a la LLVM)

Toward A Cloud Programming Stack

A language/compiler/debugger that addresses distributed concerns! E.g.:

- Is my program consistent even though it's non-monotone?
- How can I partition/replicate state safely?
- What failures can this tolerate and how many?
- What data is going where and who can see it?
- Tunable objective functions. Please optimize for:
  - $$, not latency.
  - 99'th percentile, not 95th
  - Etc.

Toward A Cloud Programming Stack

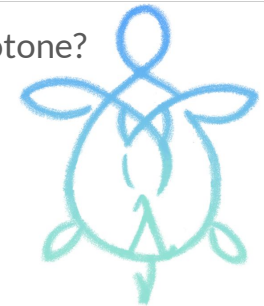
A language/compiler/debugger that addresses distributed concerns! E.g.:

- Is my program consistent even though it's non-monotone?
- How can I partition/replicate state safely?
- What failures can this tolerate and how many?
- What data is going where and who can see it?
- Tunable objective functions. Please optimize for:
  - $$, not latency.
  - 99'th percentile, not 95th
  - Etc.

Toward A Cloud Programming Stack

A language/compiler/debugger that addresses distributed concerns! E.g.:

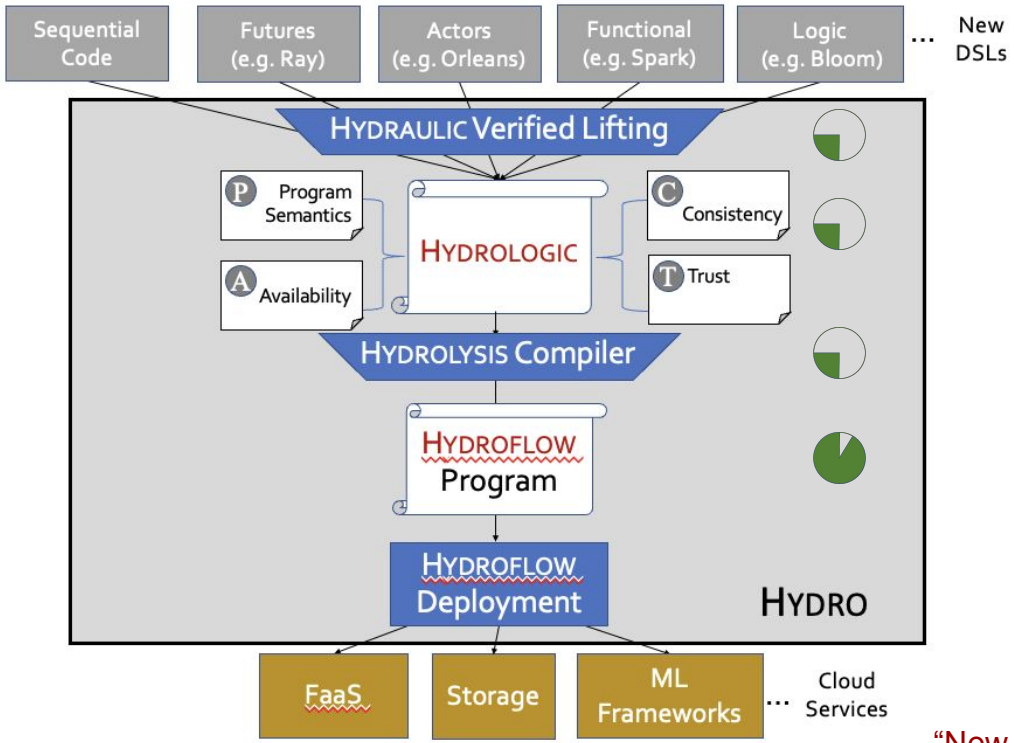
- Is my program consistent even though it's non-monotone?
- How can I partition/replicate state safely?
- What failures can this tolerate and how many?
- What data is going where and who can see it?
- Tunable objective functions. Please optimize for:
  - $$, not latency.
  - 99'th percentile, not 95th
  - Etc.



Hydro

HYDRO: A Programming Stack for the Cloud

https://hydro.run



Many languages for programmers

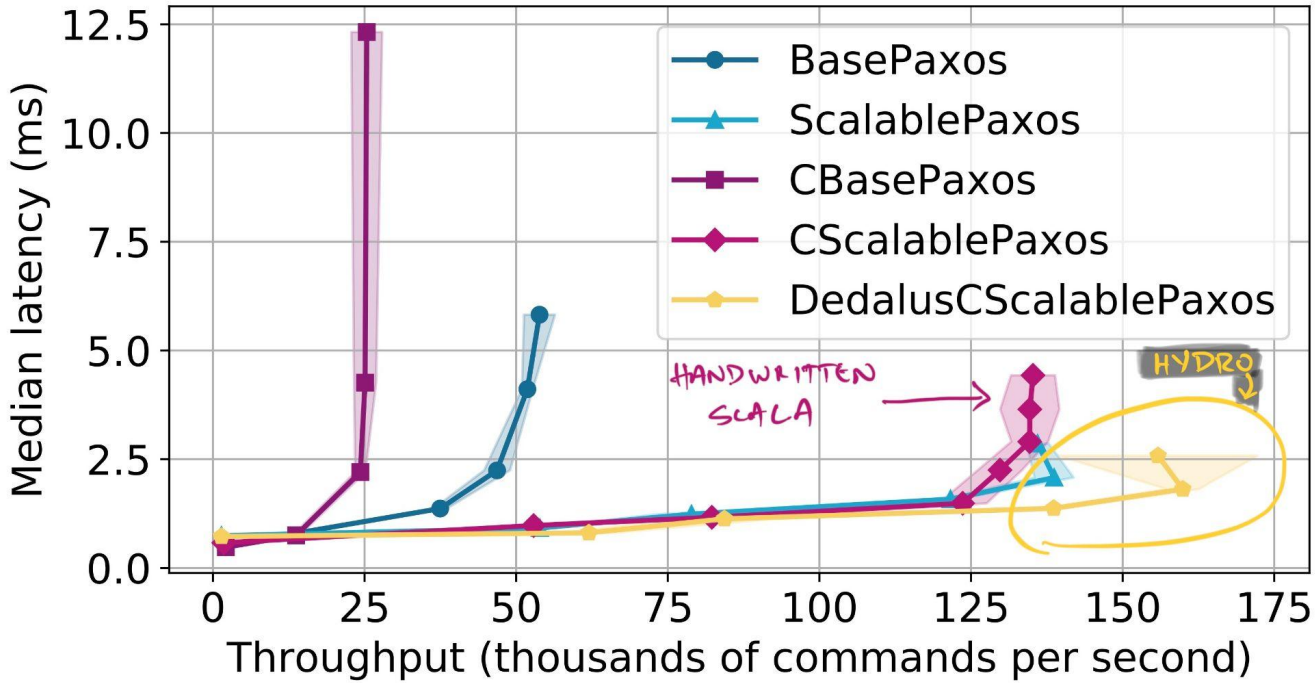
Declarative IR

Low-Level "Lattice-flow"

Adaptive self-deploying exe's

"New Directions in Cloud Programming"
CIDR 2021

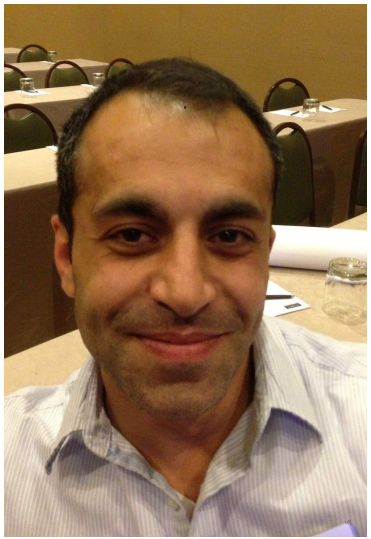
Initial Results: Compartmentalized Paxos



Hydroflow vs. code from “Scaling Replicated State Machines with Compartmentalization”, VLDB 2021

Special thanks to our co-authors

Alan Fekete



Ali Ghodsi

Mike Franklin

Ion Stoica

PB

Concluding Thoughts

9 years ago: battle between SQL and NoSQL

NoSQL held hints of truth around coordination-freedom

Our approach: go up the stack to the application to avoid trap of read/write

Has been a fruitful decade, with more to come!

Thank you!!!