Unevenly Distributed

Adrian Colyer

@adriancolyer

QCon LONDON

Accel

the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Home

A Year in Papers

DECEMBER 14, 2015

We've reached the end of term again, and I'm taking a break from writing up papers over the holidays – a chance to replenish my backlog and start planning for 2016 too! I want to see what I can do to improve the readability of the site as well. The Morning Paper will resume on the 4th January.

In a moment I'll share with you the **top 10 most read** and **most tweeted** papers, plus **some of my own picks**. But first a quick look back over the year. Through the course of 2015 I've posted **206** paper write-ups on The Morning Paper plus a few original pieces and other miscellaneous posts. That means I'm now at over 300 paper reviews in total since #themorningpaper began. It's amazing how a little every day adds up over time!

I'd like to say a huge thank-you to everyone who's been following along, I love all the interaction that the papers lead to. And if you're not yet subscribed to The Morning Paper and you're looking for a *New Years Resolution*, **signing up to the mailing list will get you**

SUBSCRIBE

never miss an issue! The Morning Paper delivered straight to your inbox.

SEARCH

type and press enter

ARCHIVES

Select Month

MOST READ IN THE LAST FEW DAYS

- ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking

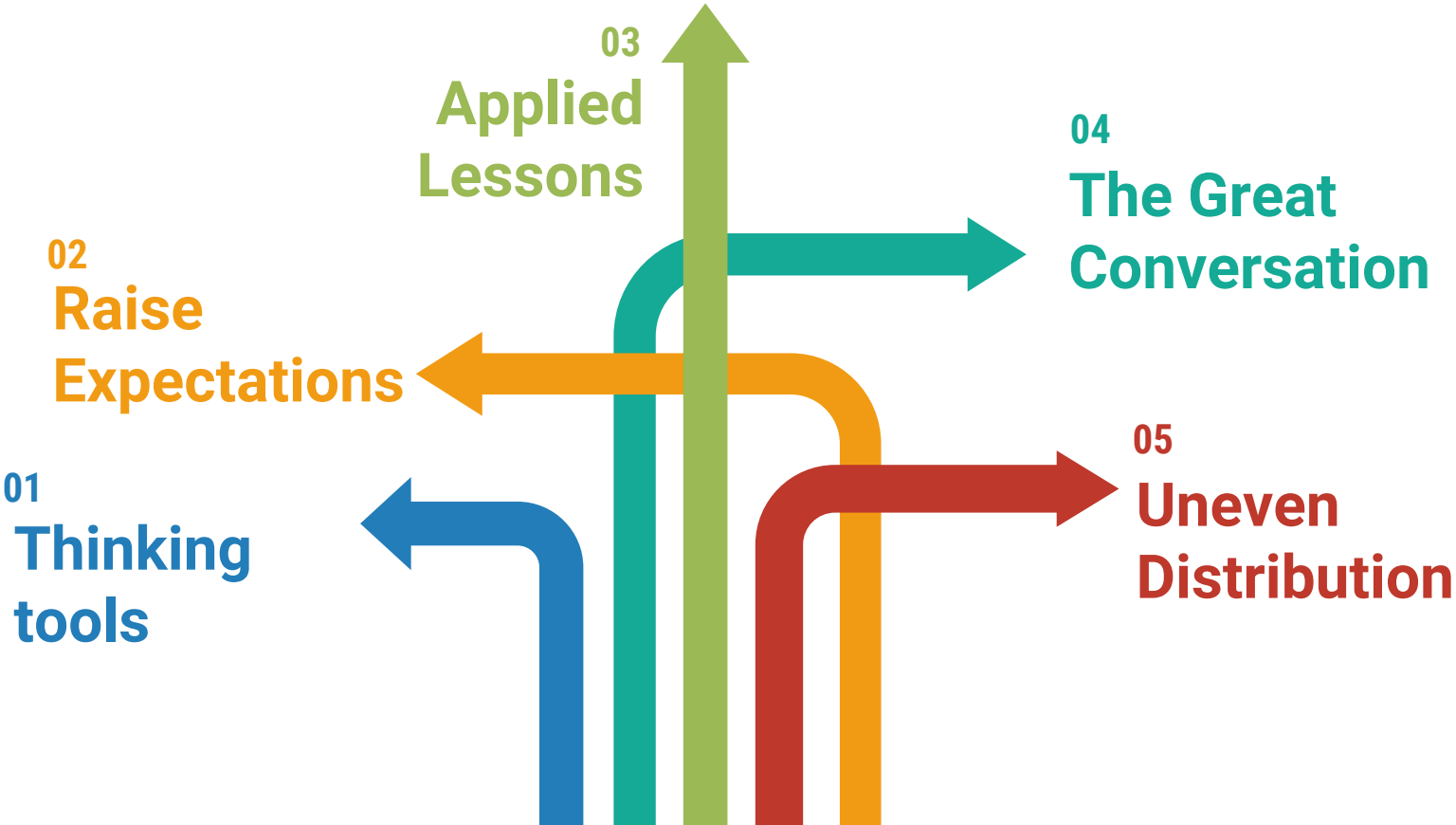
blog.acolyer.org

350

Foundations

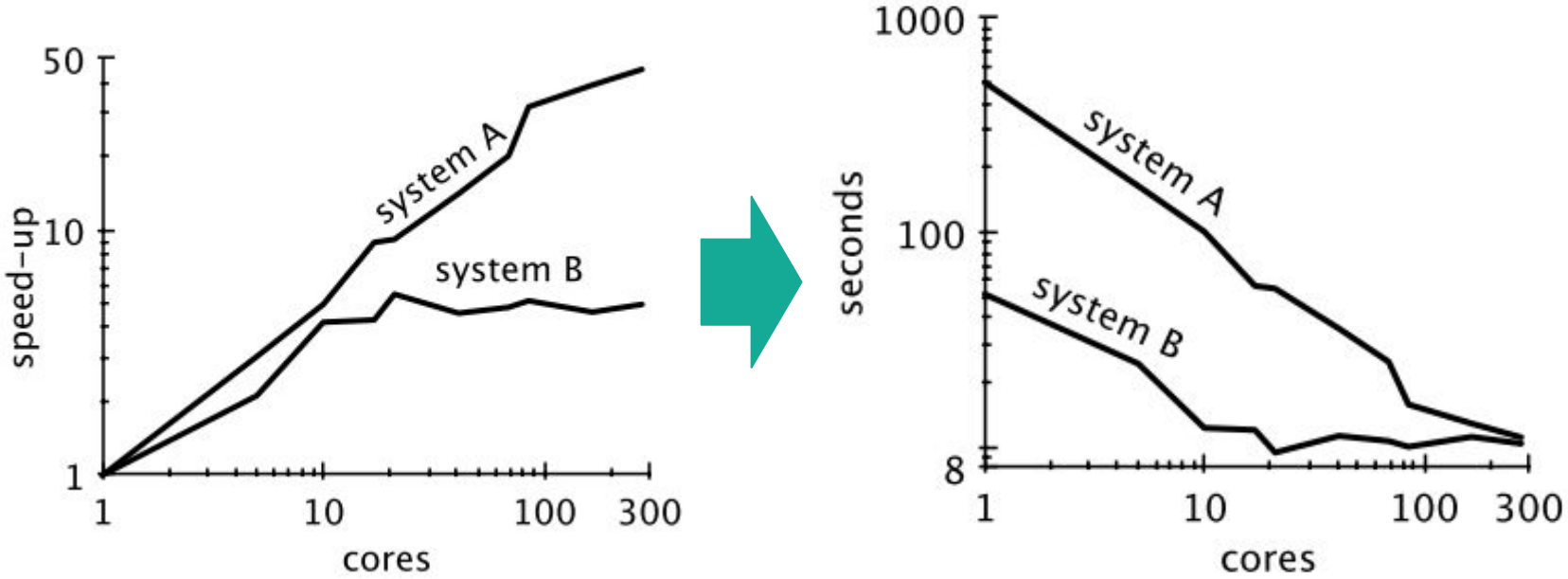
Frontiers

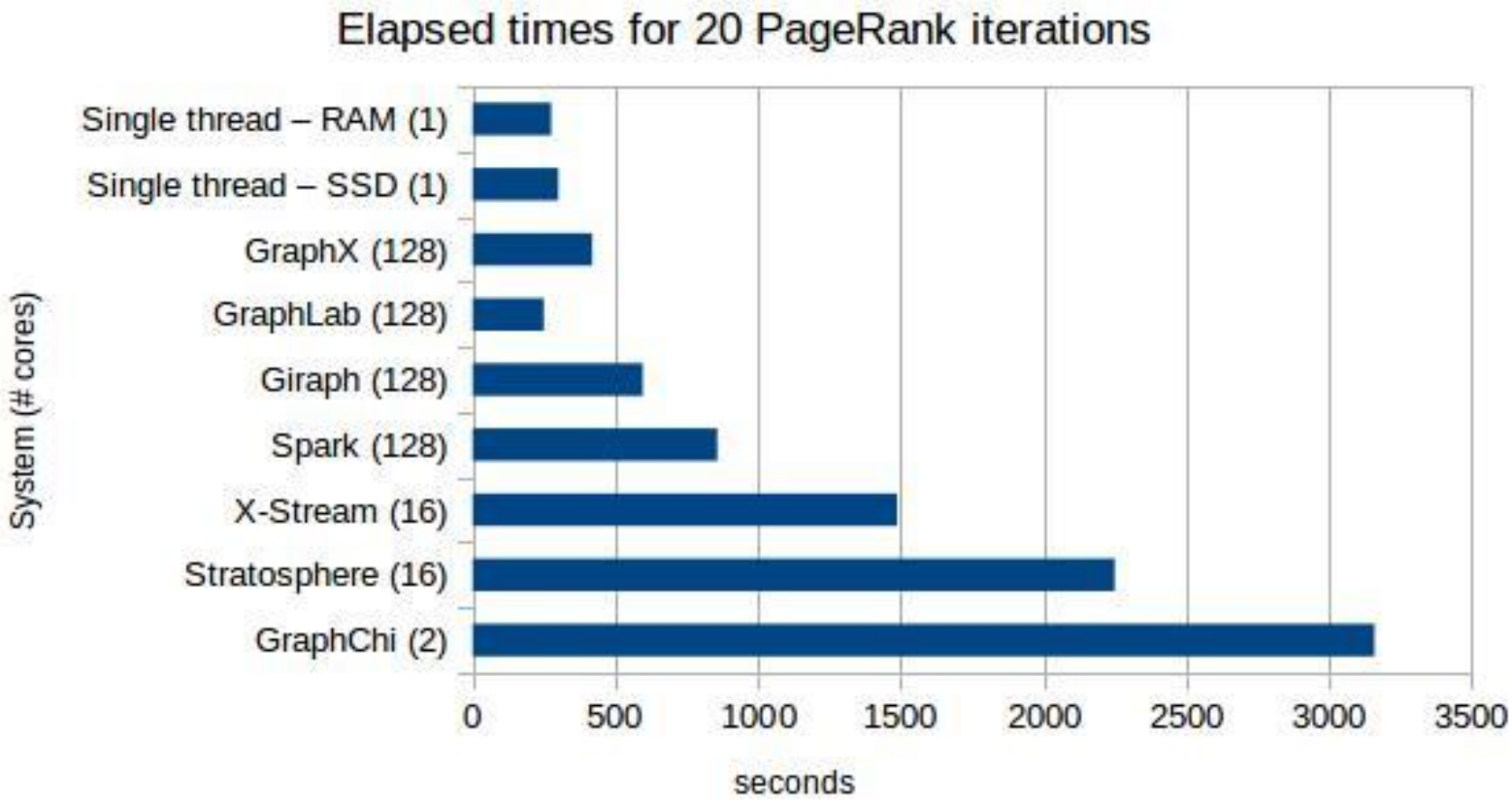
5 Reasons to <3 Papers



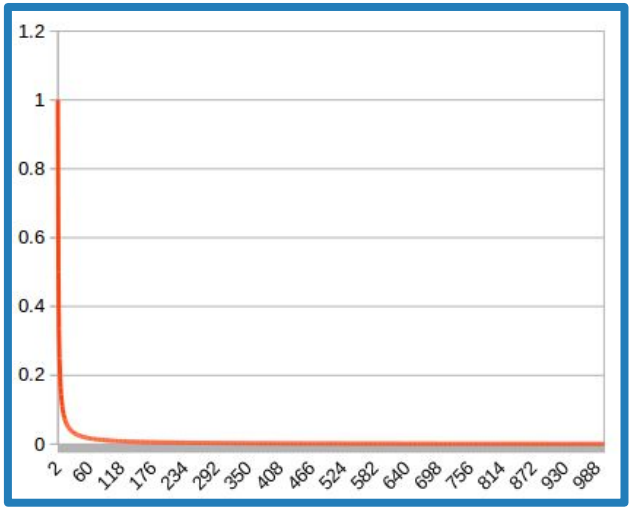
Scalability - but at what COST?

Frank McSherry




Elapsed times for 20 PageRank iterations
Single thread – RAM (1)
Single thread – SSD (1)
GraphX (128)
GraphLab (128)
Giraph (128)
Spark (128)
X-Stream (16)
Stratosphere (16)
GraphChi (2)
System (# cores)
0
500
1000
1500
2000
2500
3000
3500
seconds

But you have BIG Data!



Zipf Distribution

"Working sets are Zipf-distributed. We can therefore store in memory all but the very largest datasets."

Musketeer

One for all?

Musketeer: all for one, one for all in data processing systems

Ionel Gog Malte Schwarzkopf Natacha Crooks[†] Matthew P. Grosvenor
Allen Clement[†*] Steven Hand[*]
University of Cambridge [†] *Max Planck Institute for Software Systems*
* now at Google, Inc.

Abstract

Many systems for the parallel processing of big data are available today. Yet, few users can tell by intuition which system, or combination of systems, is "best" for a given workflow. Porting workflows between systems is tedious. Hence, users become "locked in", despite faster or more efficient systems being available. This is a direct consequence of the tight coupling between user-facing front-ends that express workflows (e.g., Hive, SparkSQL, Lindi, GraphLINQ) and the back-end execution engines that run them (e.g., MapReduce, Spark, PowerGraph, Naiad).

We argue that the ways that workflows are defined should

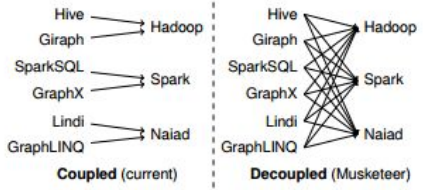
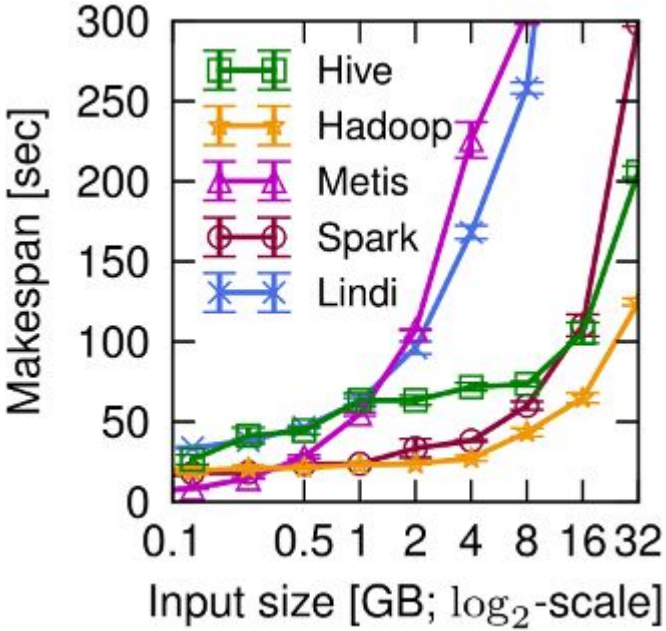
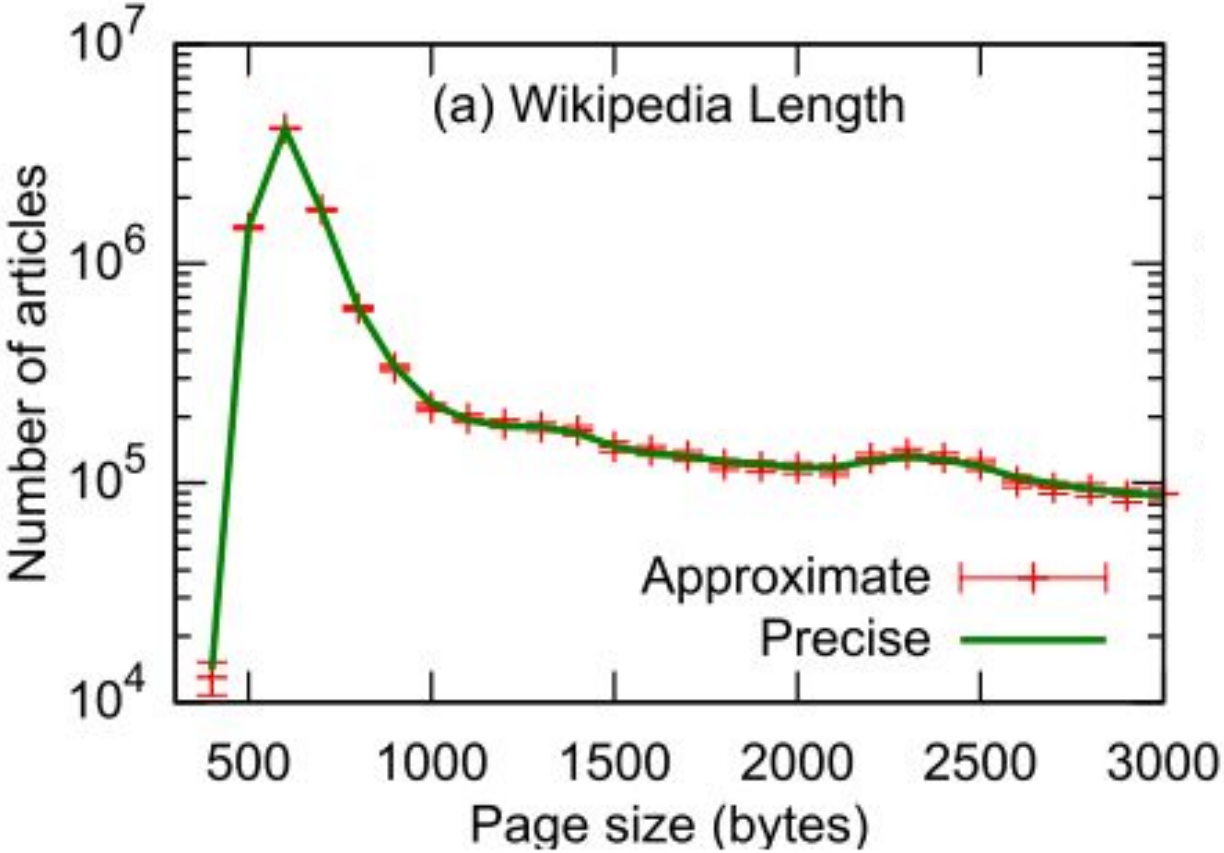


Figure 1: Decoupling front-end frameworks and back-end execution engines *(right)* increases flexibility.



Approx Hadoop



32x!

The Scalable Commutativity Rule

Improve your API Design

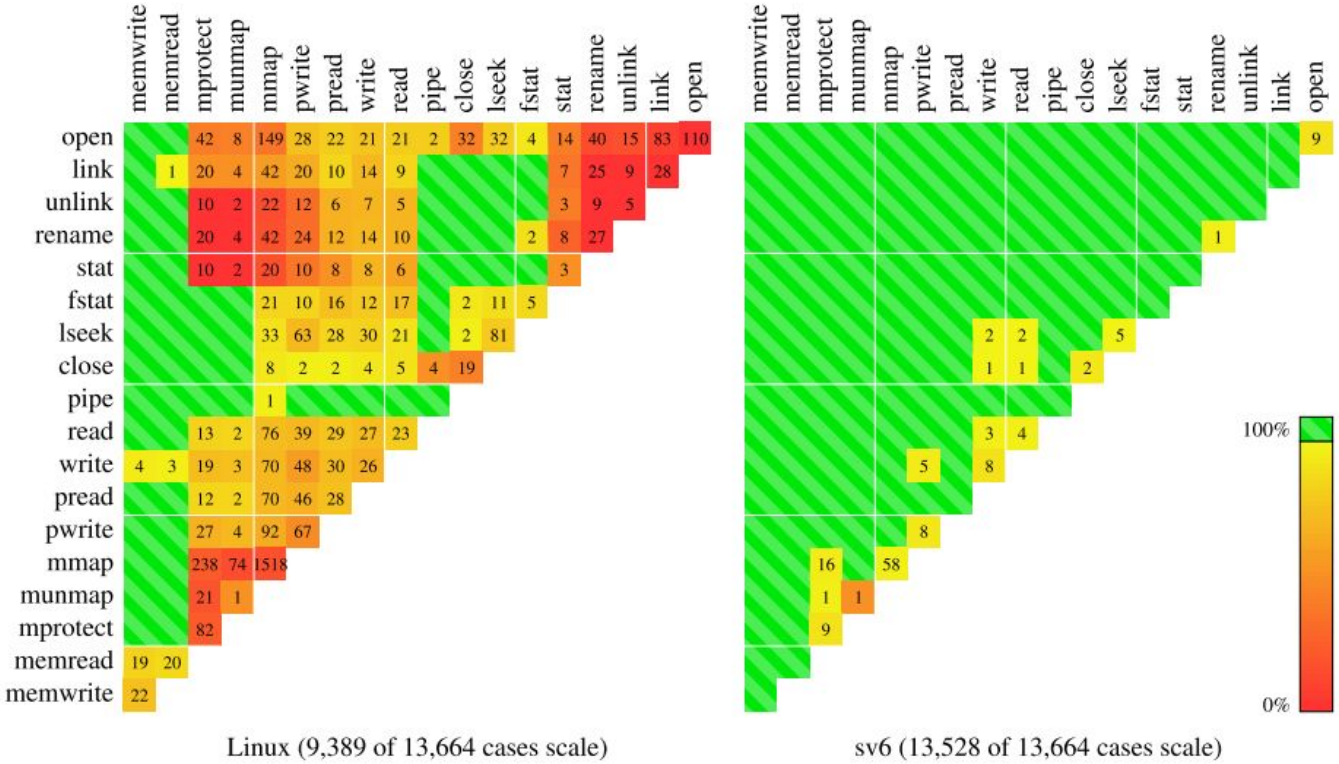


Figure 6: Scalability for system call pairs, showing the fraction and number of test cases generated by COMMUTER that are not conflict-free for each system call pair. One example test case was shown in Figure 5.

Raising Your Expectations

Not-quite-so-broken TLS: lessons in re-engineering a security protocol specification and implementation

David Kaloper-Meršinjak[†], Hannes Mehnert[†], Anil Madhavapeddy and Peter Sewell
University of Cambridge Computer Laboratory
first.last@cl.cam.ac.uk
[†] *These authors contributed equally to this work*

Abstract

Transport Layer Security (TLS) implementations have a history of security flaws. The immediate causes of these are often programming errors, e.g. in memory management, but the root causes are more fundamental: the challenges of interpreting the ambiguous prose specification, the complexities inherent in large APIs and code bases, inherently unsafe programming choices, and the impossibility of directly testing conformance between implementations and the specification.

We present *nqsb-TLS*, the result of our re-engineered approach to security protocol specification and implementation that addresses these root causes. The same code serves two roles: it is both a specification of TLS, executable as a test oracle to check conformance of traces from arbitrary implementations, and a usable implementation of TLS; a modular and declarative programming style provides clean separation between its components. Many security flaws are thus excluded by construction.

nqsb-TLS can be used in standalone Unix applications, which we demonstrate with a messaging client, and can also be compiled into Xen unikernels (specialised virtual machine image) with a trusted computing base (TCB) that is 4% of a standalone system running a standard Linux/OpenSSL stack, with all network

sensitive services, they are not providing the security we need. Transport Layer Security (TLS) is the most widely deployed security protocol on the Internet, used for authentication and confidentiality, but a long history of exploits shows that its implementations have failed to guarantee either property. Analysis of these exploits typically focusses on their immediate causes, e.g. errors in memory management or control flow, but we believe their root causes are more fundamental:

Error-prone languages: historical choices of programming language and programming style that tend to lead to such errors rather than protecting against them.

Lack of separation: the complexities inherent in working with large code bases, exacerbated by lack of emphasis on clean separation of concerns and modularity, and by poor language support for those.

Ambiguous and untestable specifications: the challenges of writing and interpreting the large and ambiguous prose specifications, and the impossibility of directly testing conformance between implementations and a prose specification.

In this paper we report on an experiment in developing a practical and usable TLS stack, *nqsb-TLS*, using a new approach designed to address each of these root-cause

54

CVEs
Jan '14 - Jan '15

! Error prone languages
! Lack of Separation
! Ambiguous and Untestable Spec

Surely we can do better?

Do Less Testing!

The Art of Testing Less without Sacrificing Quality

Kim Herzig[i] kimh@microsoft.com
Michaela Greiler[ii] mgreiler@microsoft.com
Jacek Czerwonka[ii] jacekcz@microsoft.com
Brendan Murphy[i] bmurphy@microsoft.com
[i] Microsoft Research, United Kingdom
[ii] Microsoft Corporation, Redmond, United States

Microsoft Windows 8.1

	Relative Improvement	Cost Improvement
Test Executions	40.58%	
Test Time	40.31%	$1,567,608
Test Result Inspection	33.04%	$61,533
Escaped Defects	0.20%	($11,971)
Total Cost Balance		**$1,617,170**

Failure Sketching: A Technique for Automated Root Cause Diagnosis of In-Production Failures

Baris Kasikci[1] Benjamin Schubert[1] Cristiano Pereira[2] Gilles Pokam[2] George Candea[1]

[1] {baris.kasikci,benjamin.schubert,george.candea}@epfl.ch [2] {cristiano.l.pereira,gilles.a.pokam}@intel.com
[1] School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne (EPFL)
[2] Intel Corporation

Abstract

Developers spend a lot of time searching for the root causes of software failures. For this, they traditionally try to reproduce those failures, but unfortunately many failures are so hard to reproduce in a test environment that developers spend days or weeks as ad-hoc detectives. The shortcomings of many solutions proposed for this problem prevent their use in practice.

We propose failure sketching, an automated debugging technique that provides developers with an explanation ("failure sketch") of the root cause of a failure that occurred in production. A failure sketch only contains program statements that lead to the failure, and it clearly shows the differences between failing and successful runs; these differences guide developers to the root cause. Our approach combines static program analysis with a cooperative and adaptive form of dynamic program analysis.

We built Gist, a prototype for failure sketching that relies on hardware watchpoints and a new hardware feature for extracting control flow traces (Intel Processor Trace). We show that Gist can build failure sketches with low overhead for failures in systems like Apache, SQLite, and Memcached.

1. Introduction

Debugging—the process of finding and fixing bugs—is time-consuming (around 50% [44] of the development time). This is because debugging requires a deep understanding of the code and the bug. Misunderstanding the code or the bug can lead to incorrect fixes, or worse, to fixes that introduce new bugs [30].

Traditionally, debugging is done in an iterative fashion: the developer runs the failing program over and over in a debugger, hoping to reproduce the failure, understand its root cause, and finally fix it. Fixing bugs generally requires the diagnosis of the root cause.

Intuitively, a root cause is the gist of the failure; it is a cause, or a combination of causes, which when removed from the program, prevents the failure associated with the root cause from recurring [74]. More precisely, a root cause of a failure is the negation of the predicate that needs to be enforced so that the execution is constrained to not encounter the failure [80].

The ability to reproduce failures is essential to traditional debugging, because developers rely on reproducing bugs to diagnose root causes. A recent study at Google [57] revealed that developers' ability to reproduce bugs is essential to fixing them. However, in practice, it is not always possible to reproduce bugs, and practitioners report that it takes weeks to fix hard-to-reproduce concurrency bugs [18].

The greatest challenge though, is posed by bugs that only recur in production and cannot be reproduced in-house. Diagnosing the root cause and fixing such bugs is truly hard. In [57], developers noted: *"We don't have tools for the once every 24 hours bugs in a 100 machine cluster."* An informal poll on Quora [54] asked *"What is a coder's worst nightmare,"* and the answers were *"The bug only occurs in production and can't be replicated locally,"* and *"The cause of the bug is unknown."*

A promising method to cope with hard to reproduce bugs is using record/replay systems [2, 46]. Record/replay sys-

Failure Sketch for Apache bug #21287

Type: Concurrency bug, double-free

Time	Thread T_1		Thread T_2		obj->refcnt
1	decrement_refcount(obj){	1	decrement_refcount(obj){	1	
2	if (!obj->complete) {	2	if (!obj->complete) {	2	
3	object_t *mobj = ...	3	object_t *mobj = ...	3	
4	dec(&obj->refcnt);	4		4	1
5		5	dec(&obj->refcnt);	5	0
6		6	if (!obj->refcnt) {	6	
7		7	free(obj);	7	
8	if (!obj->refcnt) {	8	}	8	
9	free(obj);	9	}	9	
	}				

Failure (double free)

Figure 8: The failure sketch of Apache bug #21287. The grayed-out components are not part of the ideal failure sketch, but they appear in the sketch that Gist automatically computes.

Lessons from the Field

A Masterclass in Config Mgt

at Facebook

Holistic Configuration Management at Facebook

Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, Akshay Chander, Zhe Wen, Aravind Narayanan, Patrick Dowell, and Robert Karl

Facebook Inc.

{tang, thawan, pradvenkat, akshay, wenzhe, aravindn, pdowell, robertkarl}@fb.com

Abstract

Facebook's web site and mobile apps are very dynamic. Every day, they undergo thousands of online configuration changes, and execute trillions of configuration checks to personalize the product features experienced by hundreds of million of daily active users. For example, configuration changes help manage the rollouts of new product features, perform A/B testing experiments on mobile devices to identify the best echo-canceling parameters for VoIP, rebalance

the many challenges. This paper presents Facebook's holistic configuration management solution. Facebook uses Chef [7] to manage OS settings and software deployment [11], which is not the focus of this paper. Instead, we focus on the home-grown tools for managing applications' dynamic runtime configurations that may be updated live multiple times a day, without application redeployment or restart. Examples include gating product rollouts, managing application-level traffic, and running A/B testing experiments.

Below, we outline the key challenges in configuration

Machine Learning Systems

lessons from Google

Ad Click Prediction: a View from the Trenches

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young,
Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov,
Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg,
Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica
Google, Inc.
mcmahan@google.com, gholt@google.com, dsculley@google.com

**Machine Learning:
The High-Interest Credit Card of Technical Debt**

01 Feature Management

02 Visualisation

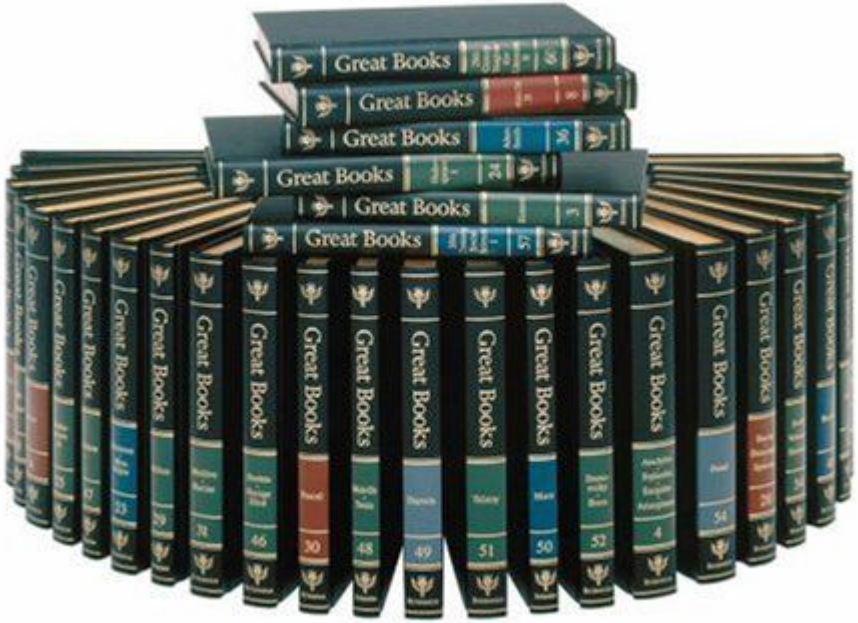
03 Relative Metrics

04 Systematic Bias Correction

05 Alerts on action Thresholds

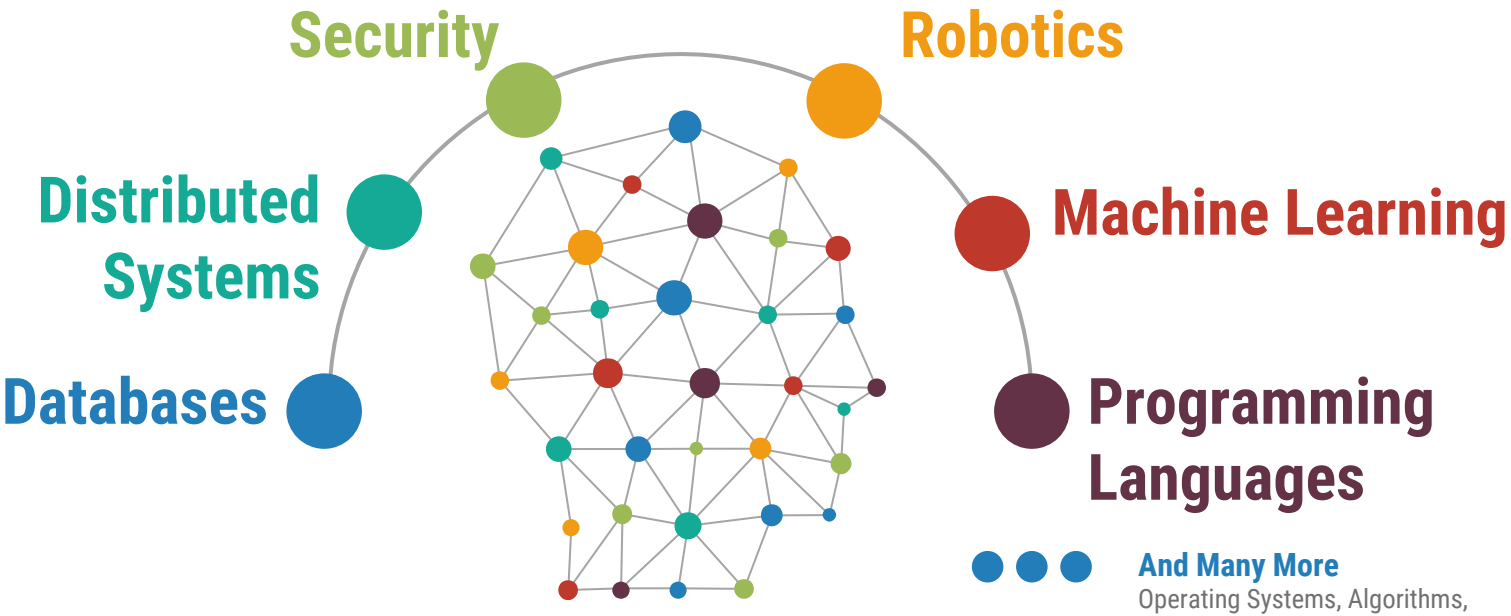
The Great Conversation

And the Syntopicon



Cross-Fertilization

Broad Exposure to Problems and their Solutions



And Many More

Operating Systems, Algorithms, Networking,Optimisation, SW Engineering,...

TPC-C - 1992

TPC

ry... The TPC is a non-profit corporation focused on developing data-centric benchmark standards and disseminating objective, verifiable perforn

- Home
- Results
- Benchmarks
- TPC Documentation
- Technical Articles
- Related Links
- What's New
- About the TPC

What's New

January 11, 2016 TPC announces a new database virtualization benchmark

January 11, 2016 TPC announces TPC DS 2.0

TPC Benchmarks & Benchmark Results

Please select any of the active TPC benchmarks below. All available options will be displayed.

(If your browser does not support all of the new features used, please select 'Results' in the navigation bar on the left for a 'text-only' version.)

TPC-C - Top Ten Performance Results
Version 5 Results As of 1-Mar-2016 at 9:45 AM [GMT]

Print Version

Note 1: The TPC believes it is not valid to compare prices or price/performance of results in different currencies.

All Results | Clustered Results | Non-Clustered Results | Currency All

Rank	Company	System	Performance (tpmC)	Price/tpmC	Watts/KtpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted
1	ORACLE	SPARC T5-8 Server	8,552,523	.55 USD	NR	09/25/13	Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning	Oracle Solaris 11.1	Oracle Tuxedo CFSR	03/26/13
2	SAP®	Dell PowerEdge T620	112,890	.19 USD	NR	11/25/14	SQL Anywhere 16	Microsoft Windows 2012 Standard x64	Microsoft COM+	11/25/14

TPC-Pricing

TPC-C Published Record Holder

Date	Mar 26th 2013
Database Manager	Oracle 11g r2 Enterprise Edition w. Partitioning
Performance (tpmC)	8,552,523 (8.5M)
Performance (tps)	**142,542 (143K)**
System Cost	$4,663,073
#Processors	8
#Cores	128
#Threads	1024

Coordination Avoidance

and I-Confluence Analysis

TPC-C

#	Informal Invariant Description	Type	Txns	$\mathcal{I}$-C
1	YTD wh sales = sum(YTD district sales)	MV	P	Yes
2	Per-district order IDs are sequential	S_{ID}+FK	N, D	No
3	New order IDs are sequentially assigned	S_{ID}	N, D	No
4	Per-district, item order count = roll-up	MV	N	Yes
5	Order carrier is set iff order is pending	FK	N, D	Yes
6	Per-order item count = line item roll-up	MV	N	Yes
7	Delivery date set iff carrier ID set	FK	D	Yes
8	YTD wh = sum(historical wh)	MV	D	Yes
9	YTD district = sum(historical district)	MV	P	Yes
10	Customer balance matches expenditures	MV	P, D	Yes
11	Orders reference New-Orders table	FK	N	Yes
12	Per-customer balance = cust. expenditures	MV	P, D	Yes

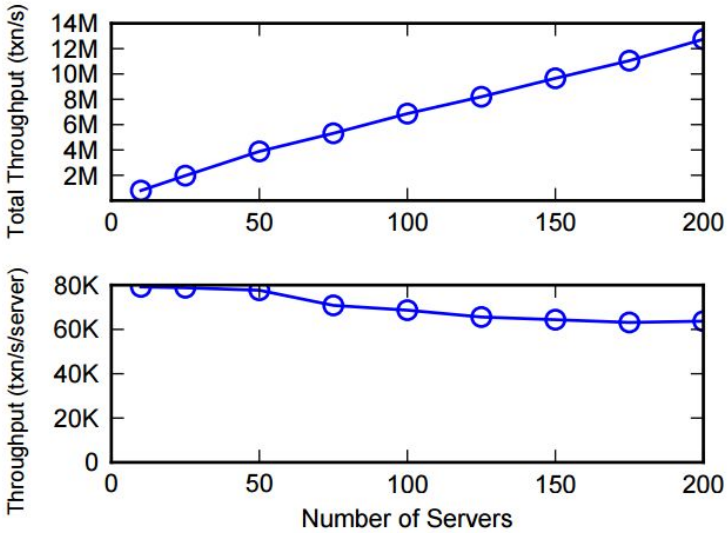


Figure 6: Coordination-avoiding New-Order scalability.

Multi-Partition Transactions at Scale

Scalable Atomic Visibility with RAMP Transactions

Peter Bailis, Alan Fekete[†], Ali Ghodsi, Joseph M. Hellers
UC Berkeley and [†]University of Sydney

ABSTRACT

Databases can provide scalability by partitioning data across several servers. However, multi-partition, multi-operation transactional access is often expensive, employing coordination-intensive locking, validation, or scheduling mechanisms. Accordingly, many real-world systems avoid mechanisms that provide useful semantics for multi-partition operations. This leads to incorrect behavior for a large class of applications including secondary indexing, foreign key

are fast but deliver incon
liver consistent results but
failure. Many of the large
protocols that guarantee f
few—if any—transactiona
sets of data items [11, 13,
rect behavior for use case
ing secondary indexing, fo

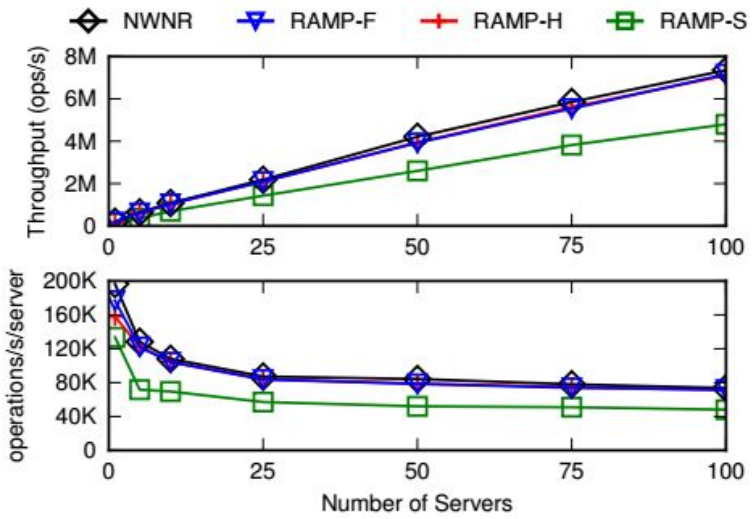


Figure 4: RAMP transactions scale linearly to over 7 million operations/s with comparable performance to NWNR baseline.

Turning your world Upside Down

Unevenly Distributed

Human computers

at Dryden by NACA (NASA) - Dryden Flight Research Center Photo Collection

http://www.dfrc.nasa.gov/Gallery/Photo/Places/HTML/E49-54.html. Licensed under Public Domain via Commons - https://commons.wikimedia.org/wiki/File:Human_computers_-_Dryden.jpg#/media/File:Human_computers_-_Dryden.jpg

Computing on a Human Scale

10s	File on desk
1:10s	Office filing cabinet
116d	Trip to the warehouse

10ns	Registers & L1-L3
70ns	Main memory
10ms	HDD

All Change Please

Next Generation Hardware

Compute

- HTM
- Persistent Memory NI
- FPGA
- GPUs

Networking

- 100GbE
- RDMA

Memory

- NVDIMMs
- Persistent Memory

Storage

- NVMe
- Next-gen NVM

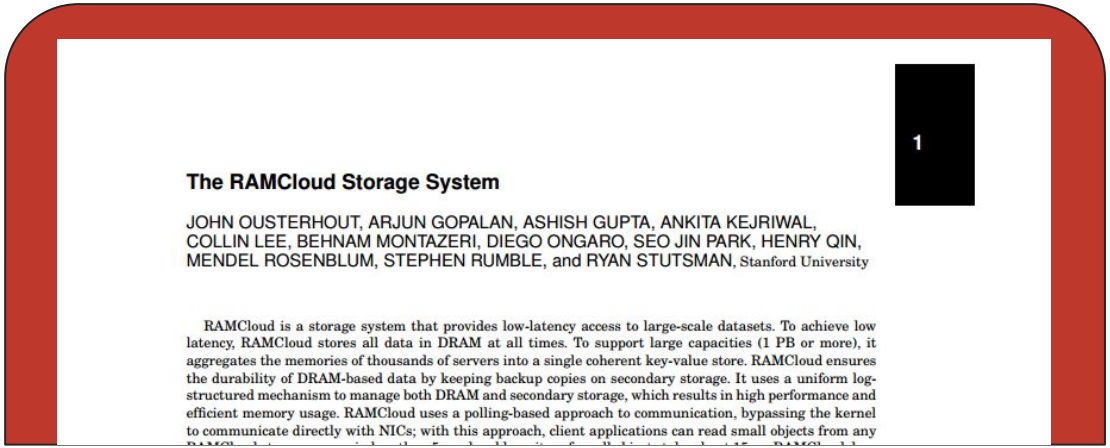
Computing on a Human Scale

10s	File on desk	2-10m	4x capacity fireproof local filing cabinets
1:10s	Office filing cabinet	23-40m	Phone another office (RDMA)
116d	Trip to the warehouse	3h20m	Next-gen warehouse

The New ~Numbers Everyone Should Know

	Latency	Bandwidth	Capacity/IOPS
Register	0.25ns		
L1 cache	1ns		
L2 cache	3ns		8MB
L3 cache	11ns		45MB
DRAM	62ns	120GBs	6TB - 4 socket
NVRAM' DIMM	620ns	60GBs	24TB - 4 socket
1-sided RDMA in Data Center	1.4us	100GbE	~700K IOPS
RPC in Data Center	2.4us	100GbE	~400K IOPS
NVRAM' NVMe	12us	6GBs	16TB/disk,~2M/600K
NVRAM' NVMf	90us	5GBs	16TB/disk, ~700/600K

Low Latency - RAMCloud



1

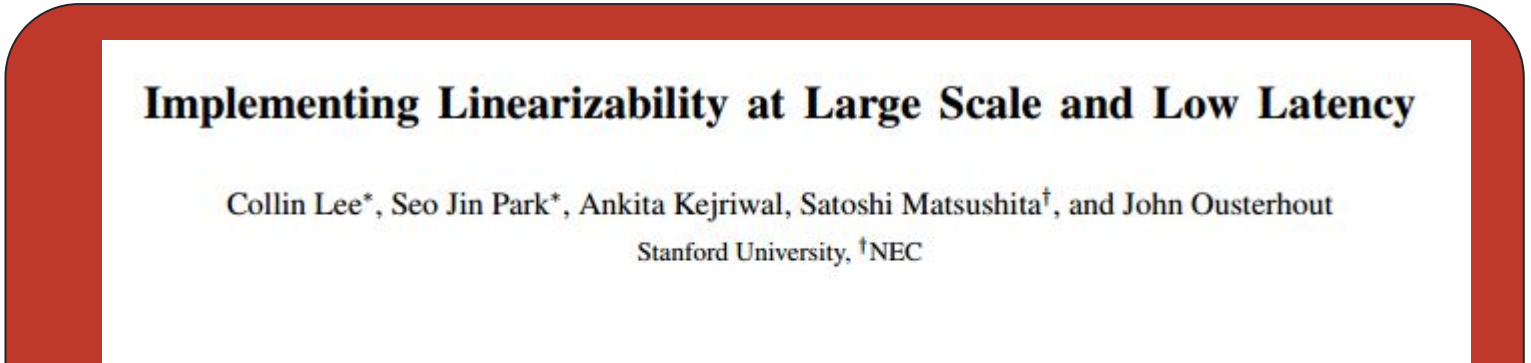
The RAMCloud Storage System

JOHN OUSTERHOUT, ARJUN GOPALAN, ASHISH GUPTA, ANKITA KEJRIWAL, COLLIN LEE, BEHNAM MONTAZERI, DIEGO ONGARO, SEO JIN PARK, HENRY QIN, MENDEL ROSENBLUM, STEPHEN RUMBLE, and RYAN STUTSMAN, Stanford University

RAMCloud is a storage system that provides low-latency access to large-scale datasets. To achieve low latency, RAMCloud stores all data in DRAM at all times. To support large capacities (1 PB or more), it aggregates the memories of thousands of servers into a single coherent key-value store. RAMCloud ensures the durability of DRAM-based data by keeping backup copies on secondary storage. It uses a uniform log-structured mechanism to manage both DRAM and secondary storage, which results in high performance and efficient memory usage. RAMCloud uses a polling-based approach to communication, bypassing the kernel to communicate directly with NICs; with this approach, client applications can read small objects from any

5μs	Reads
13.5μs	Writes
20μs	Transactions
27μs	5-object Txns
35K tps	TPC-C (10 nodes)

+



Implementing Linearizability at Large Scale and Low Latency

Collin Lee*, Seo Jin Park*, Ankita Kejriwal, Satoshi Matsushita†, and John Ousterhout

Stanford University, †NEC

No Compromises - FaRM

4.5M tps	TPC-C (90 nodes)
1.9ms	99%ile
6.3M qps	KV (per node)
41µs	at peak throughput

Life beyond Distributed Transactions: an Apostate's Opinion

Position Paper

Pat Helland

Amazon.Com
705 Fifth Ave South
Seattle, WA 98104
USA
PHelland@Amazon.com

The positions expressed in this paper are personal opinions and do not in any way reflect the positions of my employer Amazon.com.

ABSTRACT

Many decades of work have been invested in the area of distributed transactions including protocols such as 2PC, Paxos, and various approaches to quorum. These protocols provide the application programmer a façade of global serializability. Personally, I have invested a non-trivial portion of my career as a strong advocate

Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

This paper explores and names some of the practical approaches used in the implementations of large-scale mission-critical applications in a world which rejects distributed transactions. We discuss the management of fine-grained pieces of application data which may be repartitioned over time as the application grows. We also discuss the design patterns used in sending messages between these repartitionable pieces of data

No compromises: distributed transactions with consistency, availability, and performance

Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, Miguel Castro

Microsoft Research

Abstract

Transactions with strong consistency and high availability simplify building and reasoning about distributed systems. However, previous implementations performed poorly. This forced system designers to avoid transactions completely, to weaken consistency guaran-

No Compromises

"This paper demonstrates that new software in modern data centers can eliminate the need to compromise. It describes the transaction, replication, and recovery protocols in FaRM, a main memory distributed computing platform. FaRM provides distributed ACID transactions with strict serializability, high availability, high throughput and low latency. These protocols were designed from first principles to leverage two hardware trends appearing in data centers: fast commodity networks with RDMA and an inexpensive approach to providing non-volatile DRAM."

The Doctor will see you now

DrTM

Fast In-memory Transaction Processing using RDMA and HTM

Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, Haibo Chen

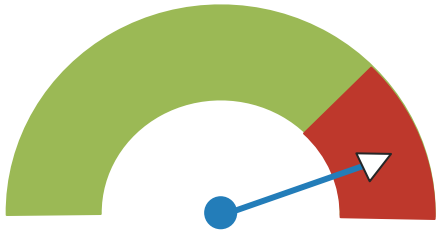
Shanghai Key Laboratory of Scalable Computing and Systems
Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University

Abstract

We present DrTM, a fast in-memory transaction processing system that exploits advanced hardware features (i.e., RDMA and HTM) to improve latency and throughput by over one order of magnitude compared to state-of-the-art distributed transaction systems. The high performance of DrTM are enabled by mostly offloading concurrency control within a local machine into HTM and leveraging the strong consistency between RDMA and HTM to ensure se-

build a transaction processing system that is at least one order of magnitude faster than the state-of-the-art systems without using such features. To answer this question, this paper presents the design and implementation of DrTM, a fast in-memory transaction processing system that exploits HTM and RDMA to run distributed transactions on a modern cluster.

Hardware transactional memory (HTM) has recently come to the mass market in the form of Intel's restricted



5.5M tps on TPC-C
6-node cluster.

Some things Change, Some stay the Same

From ARIES to MARS: Transaction Support for Next-Generation, Solid-State Drives

Joel Coburn* Trevor Bunker* Meir Schwarz Rajesh Gupta Steven S
Department of Computer Science and Engineering
University of California, San Diego
{jdcoburn,tbunker,rgupta,swanson}@cs.ucsd.edu

Abstract

Transaction-based systems often rely on write-ahead logging (WAL) algorithms designed to maximize performance on disk-based storage. However, emerging fast, byte-addressable, non-volatile memory (NVM) technologies (e.g., phase-change memories, spin-transfer torque MRAMs, and the memristor) present very different performance characteristics, so blithely applying existing algo-

1 Introduction

Emerging fast non-volatile memory (NV
such as phase change memory, spin-torque
and the memristor promise to be orders of
than existing storage technologies (i.e.,
Such a dramatic improvement shifts the
storage, system bus, main memory, and C
and will force designers to rethink storage
maximize application performance by exp

Blurred Persistence in Transactional Persistent Memory

Youyou Lu, Jiwu Shu*, Long Sun
Department of Computer Science and Technology, Tsinghua University, Beijing, China
luyouyou@tsinghua.edu.cn, shujw@tsinghua.edu.cn, sun-l12@mails.tsinghua.edu.cn

***Abstract*—Persistent memory provides data persistence at main memory level and enables memory-level storage systems. To ensure consistency of the storage systems, memory writes need to be transactional and are carefully moved across the boundary between the volatile CPU cache and the persistent memory. Unfortunately, the CPU cache is hardware-controlled, and it incurs high overhead for programs to track and move data blocks from being volatile to persistent.**

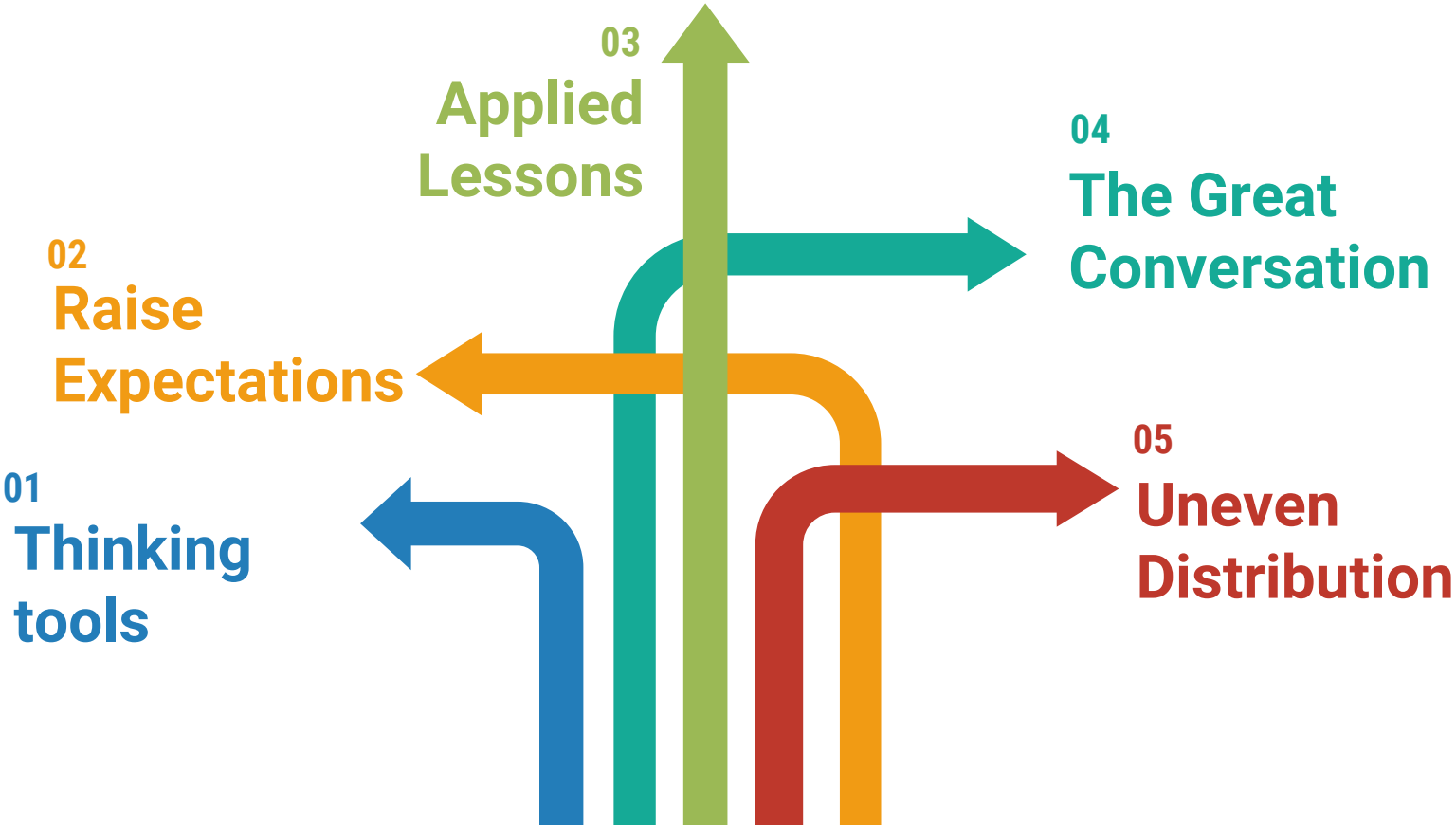
In this paper, we propose a software-based mechanism, *Blurred Persistence*, to blur the volatility-persistence boundary,

tems. As shown in Figure 1, buffer management in main memory is a white box for disk-based storage systems, while that in the CPU cache is a black box for persistent memory. Pages in main memory are managed by the operating system, and programs can know the status and perform the persistence operation on each page. With persistent memory, the CPU cache is hardware controlled, and programs find it cumbersome to track the status or perform the persistence operation for each cached block. In persistent memory, programs either keep the status of each page in the software, leading to extremely

A Brave New World

Fast RDMA networks +
Ample *Persistent* Memory +
Hardware Transactions +
Enhanced HW Cache Management +
Super-fast Storage +
On-board FPGAs + GPUs + … = ???

5 Reasons to <3 Papers



the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Home

Blurred Persistence: Efficient Transactions in Persistent Memory

JANUARY 21, 2016

Blurred Persistence: Efficient Transactions in Persistent Memory – Lu, Shu, & Sun, 2015

We had software transactional memory (STM), then hardware support for transactional memory (HTM), and now with persistent memory in which the **memory plays the role of stable storage**, we can have *persistent transactional memory*. And with persistent transactional memory, there's an issue that will surely make you smile with recognition: in-place of managing the relationship between volatile memory and disk, we now have to manage the relationship between the volatile CPU cache and memory! It's all the same considerations (forcing, stealing etc.) but in a new context and with a few new twists. Chief among those twists is that you have a lot less control over how and when the hardware moves data from cache to memory than you do over how and when you move data from memory to disk.

In case you find all these various permutations of non-volatile memory / storage confusing (I do!), then this might help:

SUBSCRIBE

never miss an issue! The Morning Paper delivered straight to your inbox.

SEARCH

type and press enter

ARCHIVES

Select Month

MOST READ IN THE LAST FEW DAYS

- Petuum: A New Platform for Distributed Machine Learning on Big Data
- Chimera: Large-Scale Classification Using Machine

01 **A new paper every weekday**

Published at http://blog.acolyer.org.

02 **Delivered Straight to your inbox**

If you prefer email-based subscription to read at your leisure.

03 **Announced on Twitter**

I'm @adriancolyer.

04 **Go to a Papers We Love Meetup**

A repository of academic computer science papers and a community who loves reading them.

Papers We Love $f(x)=x$

05 **Share what you learn**

Anyone can take part in the great conversation.

THANK YOU !

@adriancolyer