Proceedings of the 2nd Workshop on

Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED)

Held in conjunction with DISC 2019

Budapest, Hungary
October 14, 2019

Editors: Yanhong Annie Liu, Miguel Matos

Contents

Foreword

It is our great pleasure to welcome you to the 2019 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems — ApPLIED 2019. The purpose of this workshop is to bring together designers and practitioners of distributed systems from both academia and industry to share their point of views and experiences on implementing and evaluating distributed algorithms and systems. This second installment of the workshop was co-located with the 33rd International Symposium on Distributed Computing (DISC 2019), and took place on October 14th, 2019, in Budapest, Hungary.

The workshop featured keynote lectures, discussion panels, and presentations of short research papers.

The program included six keynote lectures by Ethan Buchman (Interchain Foundation, Switzerland), Jing Chen (Algorand, Inc.), Idit Keidar (Technion, Israel), Gadi Taubenfeld (Interdisciplinary Center, Israel), Zoltán Turányi (Ericsson, Hungary), Nitin Vaidya (Georgetown University, USA).

There were three panel and discussion sessions by invited panelists, other experts, and all participants, on (1) Challenges and Current State, (2) Consensus and Distributed Ledger: Scalability and Assurance, and (3) Summary and Directions.

Organization

General Chairs

- Chryssis Georgiou, University of Cyprus, Cyprus
- Elad Michael Schiller, Chalmers University of Technology, Sweden

Program Committee Chairs

- Yanhong Annie Liu, Stony Brook University, USA
- Miguel Matos, INESC-ID & IST, University of Lisboa, Portugal

Technical Program Committee

- Amy Babay, University of Pittsburgh, USA
- Ken Birman, Cornell University, USA
- Ioannis Chatzigiannakis, Sapienza University of Rome, Italy
- John Field, Google, New York City, USA
- Seif Haridi, Royal Inst. of Technology and Swedish Inst. of Computer Science, Sweden
- Wolfgang John, Ericsson Research, Sweden
- Marc Shapiro, Sorbonne-Universit-LIP6 and INRIA, France
- Srikumar Venugopal, IBM Research, Ireland

Sponsors

- University of Cyprus, Cyprus
- Chalmers University of Technology, Sweden
- Stony Brook University, USA
- Universidade de Lisboa & Angainor project (PTDC/CCI-COM/31456/2017), Portugal

Program

9:00 Opening and introduction: Chryssis Georgiou

Morning Sessions: Chair: Miguel Matos

9:15 Invited talk: Idit Keidar
Transactional Data Structure Libraries

10:00 Coffee break

10:30 Invited talk: Zoltan Turanyi
Approaches to Data Sharing in Edge FaaS

11:15 Invited talk: Nitin Vaidya
To build, or Not to Build, That Is the Question

11:45 ***Invited panel: Challenges and Current State***
Chryssis Georgiou, Zoltán Turányi, and Nitin Vaidya

12:30 Lunch

Afternoon Sessions: Chair: Annie Liu

14:00 Invited talk: Ethan Buchman
Building and Testing Byzantine Fault Tolerant State Machines

14:45 Invited talk: Jing Chen
Algorand: From Theory to Practice

15:25 Discussion session
Consensus and Distributed Ledger: Scalability and Assurance

16:00 Coffee break

16:30 Invited talk: Gadi Taubenfeld
Weak Models for Distributed Computing

17:10 Student experience session: Axel Niklasson
plcli - a Tool for Running Distributed Applications on PlanetLab

17:20 ***Invited panel and open discussion: Summary and Directions***
Ethan Buchman, Jing Chen, and Gadi Taubenfeld

17:50 Closing remarks

Invited Talk Abstracts and Speaker Bios

Invited Speakers

- Ethan Buchman, Interchain Foundation, Switzerland
- Jing Chen, Algorand, USA
- Idit Keidar, Technion, Israel
- Gadi Taubenfeld, Interdisciplinary Center, Israel
- Zoltán Turányi, Ericsson, Hungary
- Nitin Vaidya, Georgetown University, USA

Building and Testing Byzantine Fault Tolerant State Machines

Ethan Buchman, Interchain Foundation, Switzerland

Abstract: Building and testing fault tolerant state machines, let alone Byzantine Fault Tolerant (BFT) ones, is notoriously tricky business. The handful of industrial solutions, like Apache Zookeeper, CoreOS's etcd, and Hashicrop's Consul, are not BFT, and their state machines are restricted to relatively simple service-discovery oriented key-value stores. Blockchain solutions, on the other hand, add a myriad of complex state machines, including new virtual machine designs, but fail to provide adequate performance and mature development environments.

Here we present Tendermint, a production-grade Byzantine Fault Tolerant State Machine Replication engine written in Go. Tendermint supports BFT replication for state machines written in any language by using a socket protocol to communicate between the state machine and the replication engine, allowing applications to be built and tested in a developer's language of choice. Tendermint is being used on the public Internet today to secure upwards of 1 Billion USD in value, with deployments supporting hundreds of consensus nodes. Here we provide an overview of the Tendermint system and how to build Byzantine Fault Tolerant applications in Go and Javascript.

Biography Ethan is an Internet Biophysicist. He received a B.Sc. in Physical Science and a M.Sc. in Engineering Systems and Computing, both from the University of Guelph. With background in cell biology, neuroscience, mathematics, machine learning, and distributed computing, his goal is to build tools that encourage humans to self-organize into functional systems, much like molecules managed to self-organize into Life. Ethan is a co-founder of the Tendermint and Cosmos projects, and currently serves as Technical Director of the Interchain Foundation, a non-profit with a mission to research, develop, and promote open, decentralized networks. He is focused primarily on building general purpose technology to support interoperable replicated state machine on the public Internet, and in proving their correctness. He also runs CoinCulture CryptoConsulting, which offers in-depth courses on blockchain technology for both non-technical and technical audiences.

Building and Testing Byzantine Fault Tolerant State Machines

Ethan Buchman
Interchain Foundation
https://interchain.io/

Informal Systems
https://informal.systems

Budapest, 2019

Acknowledgements

Significant material in this presentation came directly or indirectly from:

Zarko Milosevic
Research Scientist
Interchain Foundation

Sunny Aggarwal
Research Scientist
All in Bits Inc.

Peng Zhong
Chief Design Officer
All in Bits Inc.

Contents

Blockchain Consensus

Blockchain Consensus

Bitcoin and Ethereum solves (a variant) of BFT consensus problem in novel network and adversarial setting:

- High number of nodes (thousands)
- Wide area network
- Open, permissionless network, i.e., not a single administration domain
- Byzantine fault tolerant
- No direct connectivity between nodes, i.e., nodes communicate over p2p gossip network

Satoshi's Invention

- Optimize BFT using hash-linked transaction batches
  - "Blockchain"
  - Batching amortizes the cost of BFT over many transactions
  - Hash links provide cheap integrity checks for clients
- Economic incentives for safety and liveness
  - "Proof-of-Work"
  - Use economics to secure a synchrony assumption and a random lottery to hedge its failure
  - Byzantine behaviour has high opportunity cost
  - Variable set of anonymous participants

Blockchain Consensus Yesterday (PoW)

Pros

- Simple to understand, prove, and implement correctly
- Based on cryptographic puzzle
- Economic security

Cons

- Probabilistic and Eventual Agreement
  - Is a committed transaction really committed?
- Low throughput
- High latency
- Not energy efficient

Blockchain Consensus Today (Tendermint)

Pros

- Adapt classic BFT consensus algorithms for the new system model
- High performance
- Instant finality
- Energy efficient
- Shift economic concerns to state machine layer
- Support state machines written in any language (!)
- Support efficient proofs for clients

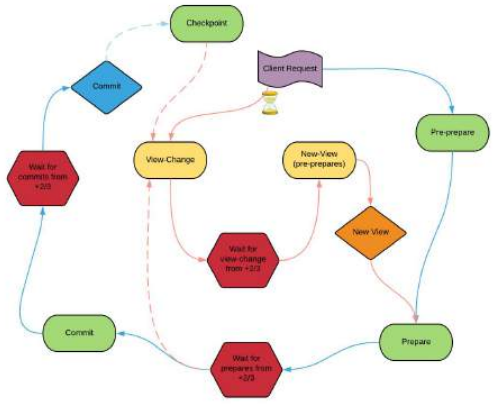
Cons

- More complex
- Thresholds for consensus

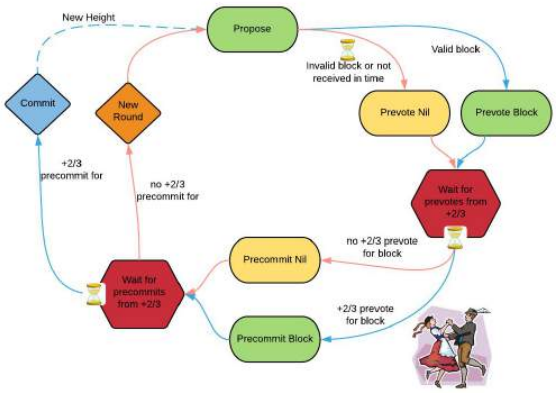
Tendermint consensus algorithm

- Proceeds in a sequence of rounds
  - Round contains multiple communication steps (similar to PBFT view)
- Every round has a dedicated proposer
  - Proposer defined by a deterministic function
  - Proposer could change every round (similar to Spinning, Veronese et al, 2009)
  - https://github.com/cwgoes/tm-proposer-idris
  - https://github.com/tendermint/tendermint/blob/master/docs/spec/reactors/consensus/proposer-selection.md
- Single execution path
  - There is no separate recovery protocol
  - Similar to normal case of PBFT
- Novel termination mechanism
  - Relies on gossip based communication
  - Message size does not depend on system size
  - Does not require additional information exchange between processes

PBFT



Tendermint

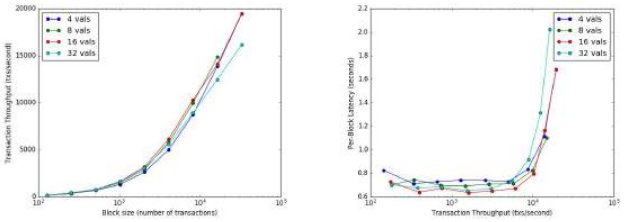


Tendermint/Cosmos History:

- Tendermint: Consensus without mining
  - 2014 - Whitepaper by Jae Kwon
- Tendermint: Byzantine Fault Tolerance in the Age of Blockchains
  - 2016 - MSc Thesis by Ethan Buchman
- Cosmos: A Network of Blockchains
  - 2016 - Whitepaper by Kwon and Buchman
- The Latest Gossip on BFT Consensus
  - 2018 - Technical Report by Buchman, Milosevic, Kwon
  - https://arxiv.org/abs/1807.04938
  - https://github.com/tendermint/spec

Tendermint Performance (2016)

7 datacenters on 5 continents (AWS - c3.8xlarge)



https://github.com/tendermint/network_testing (deprecated, sorry)

Robustness (2017)

https://jepsen.io/analyses/tendermint-0-10-2

JEPSEN

Analyses | Talks | Consistency | Services | Ethics

Tendermint 0.10.2
2017-09-05

Tendermint is a distributed, byzantine fault-tolerant consensus system designed to replicate arbitrary state machines. We experimentally verify Tendermint's safety properties using its built-in key-value store, Merkleeyes, as the hosted state machine, while creating simple and complex network partitions, clock skew, process crashes, write-ahead-log truncation, simple byzantine faults, and dynamic membership reconfiguration. We discovered a single-node data corruption issue in Merkleeyes, a fatal crash in Merkleeyes WAL recovery, and a potential data-loss issue in the Tendermint WAL, but otherwise found no cases of nonlinearizable behavior, so long as byzantine validators control less than 1/3 of the vote. This work was funded by the Tendermint team, and conducted in accordance with the Jepsen ethics policy.

Testnets (2018)

https://medium.com/tendermint/2018-year-in-review-the-top-5-most-epic-cosmos-testnets-that-will-restore-your-faith-in-3b060974f9a7

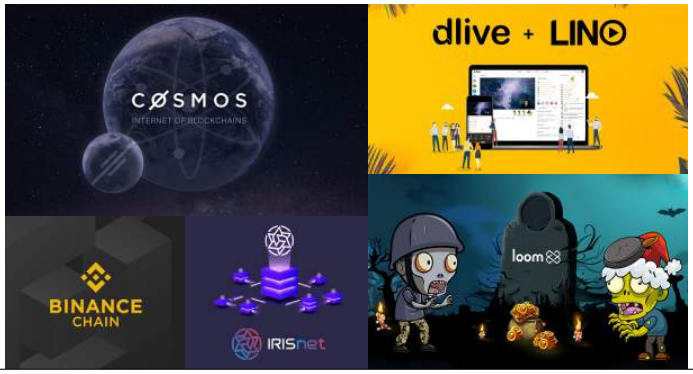
Adversarial Cosmos Testnet (2019)

https://github.com/cosmos/game-of-stakes

https://forum.cosmos.network/
https://riot.im/app/#/room/#gameofstakes:matrix.org

Mainnets (2019)

Over $5B in collective market cap



Open Research Challenges

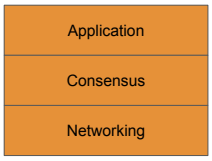
https://interchain.io/careers https://interchain.io/funding/

- Formal verification of Tendermint algorithms and implementation
- Optimal gossip protocol for
  - Transaction broadcasting
  - Votes dissemination and fast round synchronisation
- Optimal algorithm for blockchain and state synchronisation
- Use of advanced cryptography (eg. aggregate signatures) for improving performance and scalability
- Pipelining blocks and opportunities for concurrent execution
- Interaction between application-level staking logic and Tendermint behaviour
- Light client performance and advanced cross chain communication

Blockchain Applications

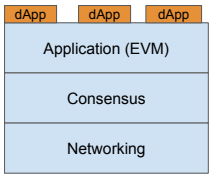
Somebody Else's State Machine

Eg. Zookeeper, Bitcoin



Somebody Else's State Machine

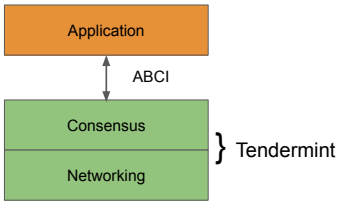
Eg. Ethereum, Aion, Tezos



Your State Machine

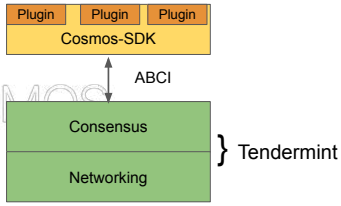
Eg. built on Tendermint

Application Blockchain Interface (ABCI):
*State Machines in **any***
Programming Language



Your State Machine

Eg. built with an ABCI Application Framework



ABCI Ecosystem

https://tendermint.com/ecosystem

VMs & Smart Contract Languages

Why?

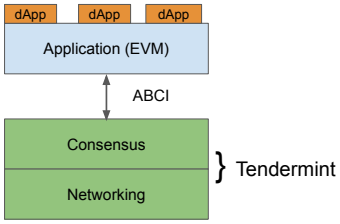
- Restricted and “secure” environment for running untrusted code
- Determinism and termination
- Easy interoperability between contracts in the same language / VM

Why not?

- Maturity, ecosystem, tooling, etc.
- Attack surface
- Customization, performance tuning
- Scalability

VMs & Smart Contract Languages ...

… over ABCI!



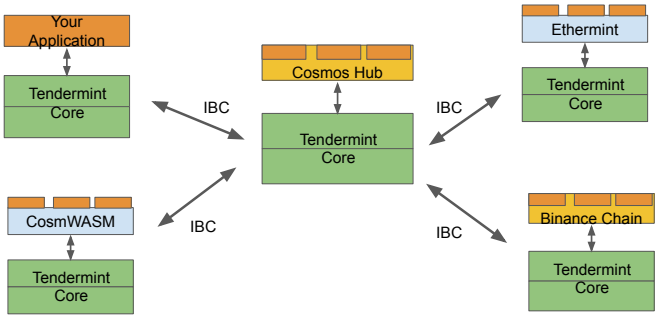
ABCI Virtual Machines

https://tendermint.com/ecosystem

Virtual Machine	Project	Framework	Developer
EVM	https://github.com/cosmos/ethermint	Cosmos-SDK	https://chainsafe.io/
WASM	https://github.com/cosmos-gaians	Cosmos-SDK	https://github.com/cosmos-gaians
Secure EcmaScript	https://github.com/Agoric/cosmic-swingset	Cosmos-SDK + Agoric SwingSet	https://agoric.com/
Pact	(https://github.com/f-o-a-m/hs-abci-server)	hs-abci-server	https://foam.space/ + https://kadena.io/
Move	https://github.com/open-libra/movemint	rust-abci	https://www.openlibra.io

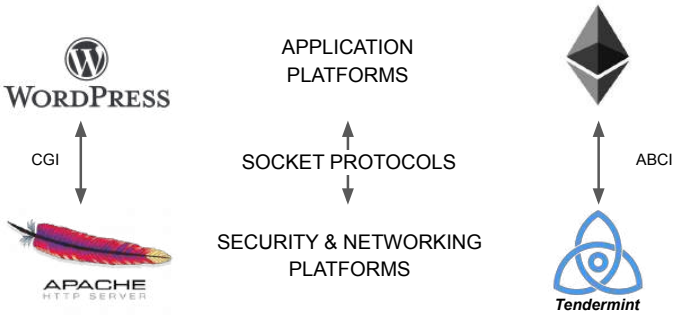
Cross Chain Interoperability

Eg. Cosmos Network

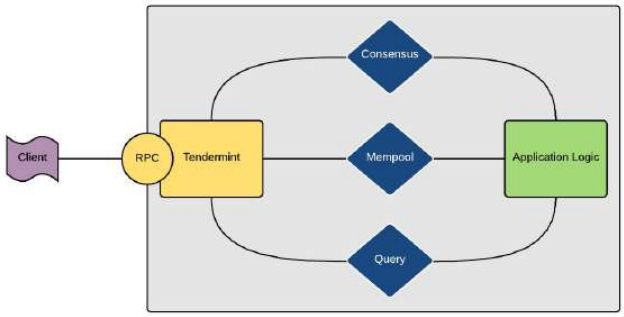


ABCI

Application Blockchain Interface (ABCI)



Application Blockchain Interface (ABCI)



Application Blockchain Interface (ABCI)

```
// Application is an interface that enables any finite, deterministic state machine
// to be driven by a blockchain-based replication engine via the ABCI.
// All methods take a RequestXxx argument and return a ResponseXxx argument,
// except CheckTx/DeliverTx, which take `tx []byte`, and `Commit`, which takes nothing.
type Application interface {
    // Info/Query Connection
    Info(RequestInfo) ResponseInfo                // Return application info
    SetOption(RequestSetOption) ResponseSetOption // Set application option
    Query(RequestQuery) ResponseQuery             // Query for state

    // Mempool Connection
    CheckTx(tx []byte) ResponseCheckTx // Validate a tx for the mempool

    // Consensus Connection
    InitChain(RequestInitChain) ResponseInitChain    // Initialize blockchain with validators and other info from TendermintCore
    BeginBlock(RequestBeginBlock) ResponseBeginBlock // Signals the beginning of a block
    DeliverTx(tx []byte) ResponseDeliverTx           // Deliver a tx for full processing
    EndBlock(RequestEndBlock) ResponseEndBlock       // Signals the end of a block, returns changes to the validator set
    Commit() ResponseCommit                          // Commit the state and return the application Merkle root hash
}
```

Cosmos SDK

Cosmos SDK

Ruby on Rails for Blockchains

- Abstract away low-level ABCI concerns

- Golang
  - Tiny language, static and interface types, high performance, compiles everywhere, standardized formatting, built-in testing, etc.

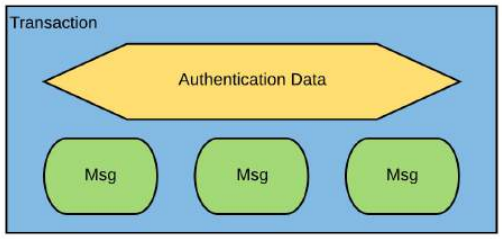
- Composable modules

- Object-Capability based security - “Principle of Least Authority”

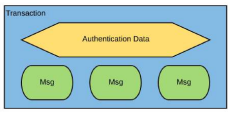
Cosmos SDK

Ruby on Rails for Blockchains - batteries included

Txs and Msgs

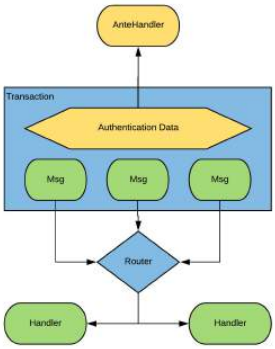


Txs and Msgs




```
// A standard way to wrap Msgs
// with a Fee and Signatures.
type StdTx struct {
    Msgs       []sdk.Msg       `json:"msg"`
    Fee        StdFee          `json:"fee"`
    Signatures []StdSignature `json:"signatures"`
    Memo       string          `json:"memo"`
}
```

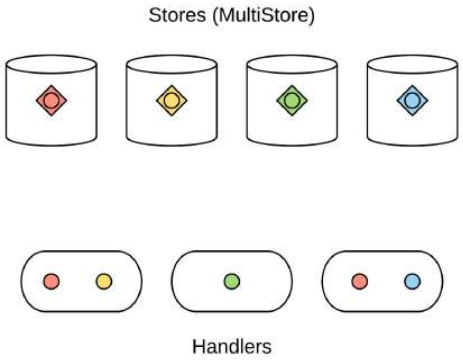
Handlers and AnteHandler



Handlers, AnteHandler, Result

```
// Handler defines the core of the application's
// state transition function
type Handler func(ctx Context, msg Msg) Result

// AnteHandler authenticates transactions before
// their internal messages are handled.
type AnteHandler func(ctx Context, tx Tx) (newCtx Context, result Result, abort bool)

// Result is the result of a transaction
type Result struct {
    Code ABCICodeType
    Data []byte
    Log  string

    GasWanted int64
    GasUsed   int64
    Fee       sdk.Coins

    Tags Tags
}
```

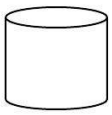
Object-Capability Stores



Object-Capability Stores

```
// Create keys, register handlers, mount stores
keyAccount := sdk.NewKVStoreKey("acc")
keyIssue := sdk.NewKVStoreKey("issue")
app.Router().
    AddRoute("send", handleMsgSend(keyAccount)).
    AddRoute("issue", handleMsgIssue(keyAccount, keyIssue))
app.MountStoresIAVL(keyAccount, keyIssue)
```

Simple Key-Value Store



```
// KVStore is a key-value store to get/set data.
type KVStore interface {
    Get(key []byte) []byte
    Has(key []byte) bool
    Set(key, value []byte)
    Delete(key []byte)
    Prefix(prefix []byte) KVStore
    Iterator(start, end []byte) Iterator
    ReverseIterator(start, end []byte) Iterator
}
```

Simple Key-Value Store

```
// Load the store, fetch an account.
store := ctx.KVStore(key)
accBytes := store.Get(from)
```

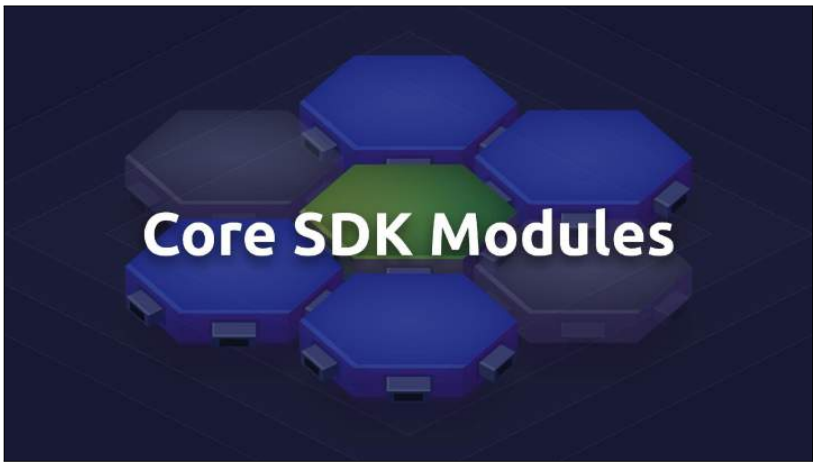
Mappers and Keepers

Mappers and Keepers

```
type accountMapper struct {
    key sdk.KVStoreKey
}

type AccountMapper interface {
    GetAccount(sdk.Context, sdk.Address) Account
    SetAccount(sdk.Context, sdk.Address, Account)
}

type coinKeepr struct {
    mapper AccountMapper
}

type CoinKeeper interface {
    GetCoins(sdk.Context, sdk.Address) sdk.Coins
    SetCoins(sdk.Context, sdk.Address, sdk.Coins)
}
```

BeginBlock & EndBlock

```
type RequestBeginBlock struct {
    Hash                []byte
    Header              Header
    Validators          []SigningValidator
    ByzantineValidators []Evidence
}

type ResponseEndBlock struct {
    ValidatorUpdates      []Validator
    ConsensusParamUpdates *ConsensusParams
    Tags                  []common.KVPair
}
```



Core Module: Tendermint

- Mature, state-of-the-art BFT consensus & networking protocol
- Vertical scaling solution for your base-layer chain
- Instant confirmation: 1 block finality
- 170+ validators on latest testnet
- Formally proven

Core Module: Proof of Stake

- Use your own coin for slashing and staking mechanics
- Maximize decentralization with delegation (through skin in the game)
- Mitigate long range attacks with an unbonding period

Core Module: Governance

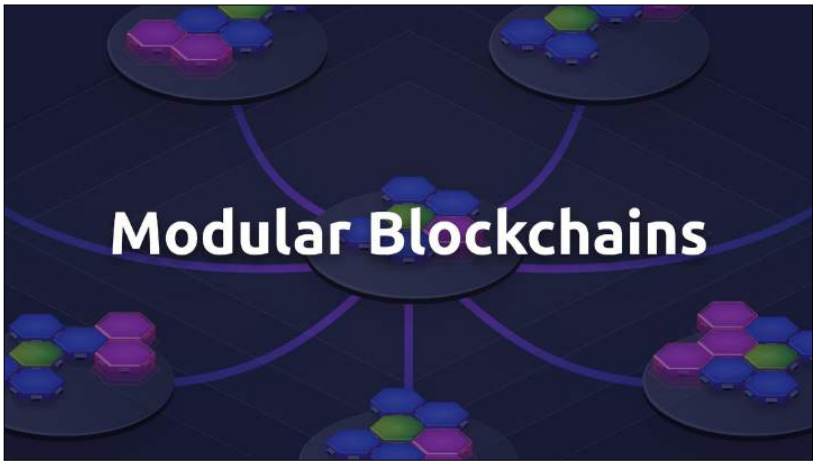
- On-chain proposal system:
  - Text Updates
  - Automatic Parameter Changes
  - Software Upgrades
- Liquid democracy for bonded stakeholders

Core Module: Rewards & Fees

- Use your own coin to pay fees when running a transaction - no lock-in.
- Multi-coin support for improved user experience.

Core Module: IBC

- The Inter-Blockchain Communication (IBC) Protocol enables the Internet of Blockchains
- Move coins & data between different blockchains
- Basis for horizontal scaling and interoperability
- Analogous to TCP/IP for the Internet



SDK + Peggy == Cosmos Hub

COSMOS

cosmos.network

SDK + EVM == Ethermint

github.com/cosmos/ethermint

SDK + Microservices == IRISnet

irisnet

irisnet.org

Rep, Auctions, BW Fees == LINO

LINO

lino.network

Channels + ILP == Kava

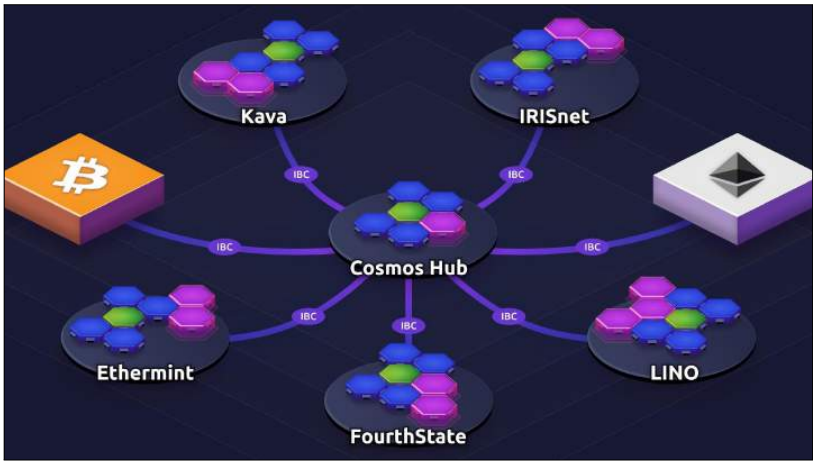
Kava

kava.io

Peggy + Plasma == Fourth State

4TH STATE LABS

github.com/fourthstate



Application: Regen Network

https://www.regen.network/

Ecological State

Regen Network monitors on-the-ground conditions and generates trusted attestations about the ecological state of a piece of land.

Verified Change of State

Network members assess the impact of their ecological actions on the land using Ecological State Protocols.

Direct Rewards

Ecological Contracts allow organizations to allocate funds and distribute them according to specific verified outcomes.

Application: TruStory

https://www.trustory.io/

Lotion JS

Lotion JS

Lotion JS

lotion

Smooth, easy blockchain apps. Powered by Tendermint consensus.

https://lotionjs.com

Simple Lotion App

```
let app = require('lotion')({
  initialState: { count: 0 }
})

app.use((state, tx) => {
  state.count++
})

app.listen(3000)
```

JS Smart Contracts for Bitcoin!

https://nomic.io/

NOMIC

Write Bitcoin smart contracts in JavaScript.

Nomic makes it easy and fun to write smart contracts for the world's most popular cryptocurrency, using the language of the web.

JS Smart Contracts for Bitcoin!

```
module.exports = {
  lastDepositor: Contract.getAddress(),
  blocksUntilPayout: 1000,

  deposit(address) {
    this.lastDepositor = address
    this.blocksUntilPayout = 1000
  },

  onBlock() {
    this.blocksUntilPayout -= 1
    if (this.blocksUntilPayout === 0) {
      /**
       * Send all held funds to last depositor when time runs out
       */
      let balance = Bitcoin.getBalance()
      Bitcoin.send(this.lastDepositor, balance)
      this.blocksUntilPayout = 1000
    }
  }
}
```

https://github.com/nomic-io/bitcoin-peg/blob/master/bitcoinPeg.md

But Javascript is Insecure!

Hacker News new | past | comments | ask | show | jobs | submit

▲ Backdoor in event-stream library dependency (github.com)

1034 points by cnorthwood 7 months ago | hide | past | web | favorite | 491 comments

The npm Blog

Blog about npm things.

Details about the event-stream incident

But Javascript is Insecure!

bitpay / copay

Code | Issues 140 | Pull requests 7 | Projects 0 | Wiki | Security | Insights

`event-stream` dependency attack steals wallets from users of copay #9346

Closed deanveloper opened this issue on Nov 26, 2018 · 29 comments

deanveloper commented on Nov 26, 2018

dominictarr/event-stream#116

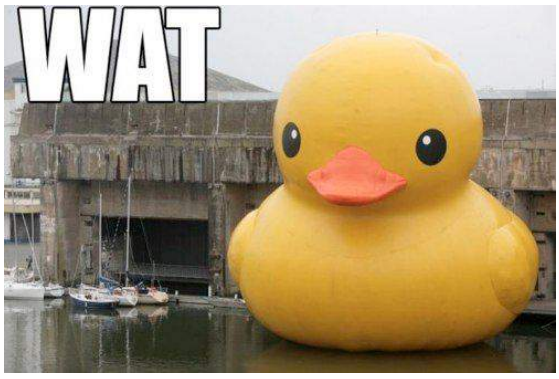
63 10 4 3

deanveloper referenced this issue on Nov 26, 2018

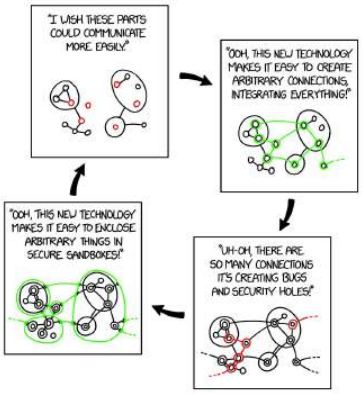
I don't know what to say. #116 Closed

scottrobertson referenced this issue on Nov 26, 2018

npm package 'event-stream' pulls in malware #9347 Closed



https://www.destroyallsoftware.com/talks/wat



https://xkcd.com/2044/

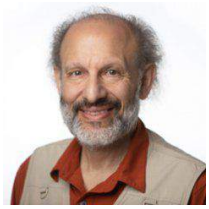
AGORIC

Origins of Smart Contracts

Markets and Computation: Agoric Open Systems

Mark S. Miller
Xerox Palo Alto Research Center,
3333 Coyote Hill Road, Palo Alto, CA 94304
K. Eric Drexler
MIT Artificial Intelligence Laboratory,
545 Technology Square, Cambridge, MA 02139*

The Ecology of Computation
B.A. Huberman (editor)



Mark S. Miller

Principle of Least Authority

AGORIC ABOUT EVENTS BLOG

POLA Would Have Prevented the Event-Stream Incident

> *The mistake is in asking "How can we prevent attacks?" when we should be asking "How can we limit the damage that can be done when an attack succeeds?". The former assumes infallibility; the latter recognizes that building systems is a human process.*
>
> *— Alan Karp, "POLA Today Keeps the Virus at Bay", HP Labs*

The JavaScript world was rocked this week by news that the popular npm package event-stream included malicious code that attempted to steal the private keys of certain Bitcoin users.

Since the attack was discovered, both the JavaScript community and the cryptocurrency community have been passionately debating how to prevent such an attack. At Agoric, we think this attack was entirely preventable, and the answer is POLA, the Principle of Least Authority.

https://agoric.com/event-stream-exploit-was-preventable-pola/

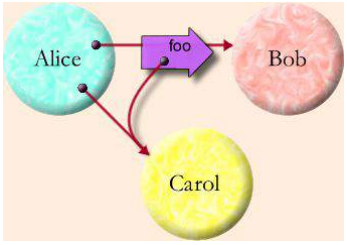
Principle of Least Authority

```
1  // hmm, why does this need access to the file system and the network?
2  const addHeader = require('./addHeader', {fs, https});
3
4  // no authority is being passed here, this is probably safe
5  const addFooter = require('./addFooter');
```

newIndex.js hosted with ❤ by GitHub view raw

https://medium.com/agoric/pola-would-have-prevented-the-event-stream-incident-45653ecbda99

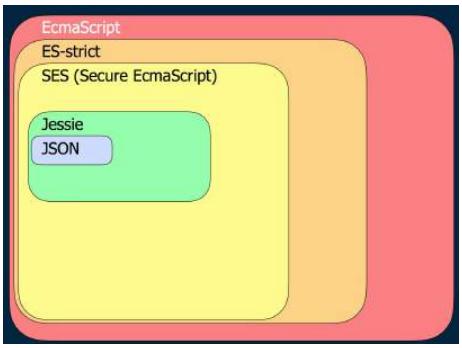
Object Capabilities

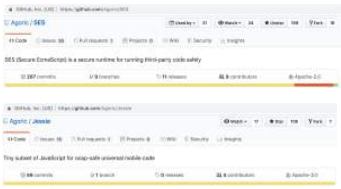


http://erights.org/elib/capability/overview.html

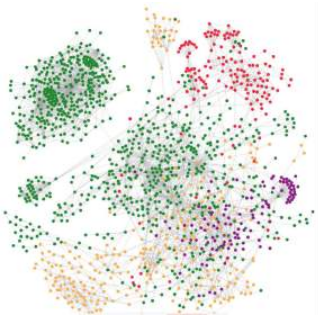
Securing Javascript

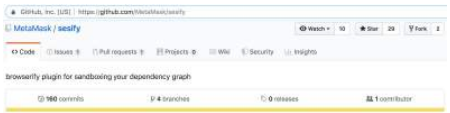
https://agoric.com/





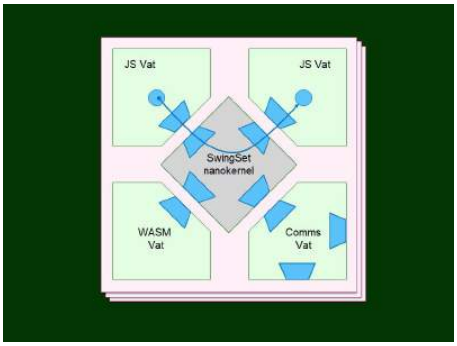
Securing Javascript





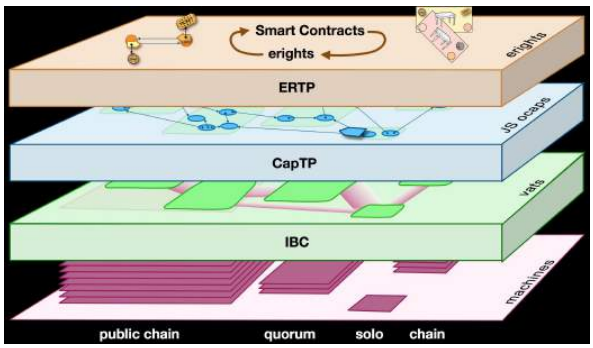
Agoric Stack

https://agoric.com/

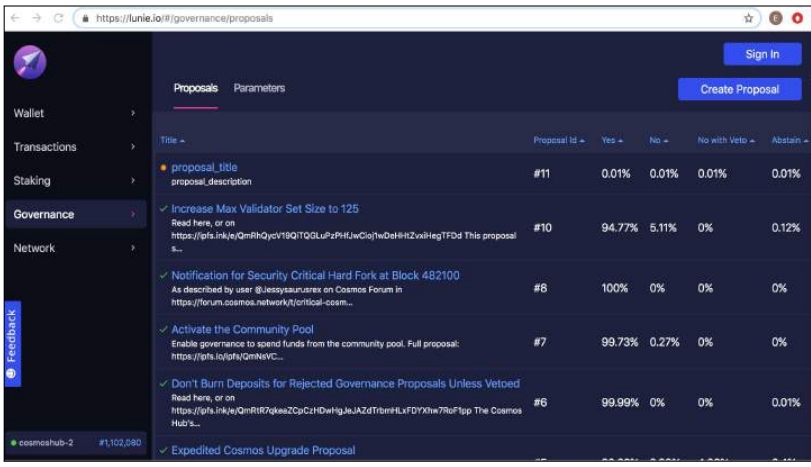


Agoric Stack

https://agoric.com/



Governance & Proof of Stake



Conclusion

Key Takeaways (Tendermint)

- Tendermint BFT
  - Simple, robust, gossip based
- Tendermint Apps
  - State machines in any language via ABCI
  - Go, JS, Rust, Python, Haskell, Erlang, etc.
  - Many VMs coming

Key Takeaways (Cosmos, Go)

- Cosmos
  - “Internet of Blockchains”
  - Application Specific Blockchains connected via IBC
  - Sovereignty, Diversity, Security, Scalability, Interoperability
- Cosmos-SDK
  - “Ruby on Rails” for blockchains
  - Modular framework for experimenting with economic systems in Go
- Cosmos Hub
  - Largest & Most Advanced Bonded Proof of Stake
  - Hub for interoperability between blockchains

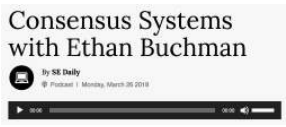
Key Takeaways (Javascript)

- LotionJS
  - Simple JS framework for building Tendermint blockchains
  - Pegs to Bitcoin facilitate Bitcoin smart contracts in JS
- Agoric
  - JS can be secure!
  - Smart Contracts in JS via Object Capabilities
  - Tight integration with Cosmos-SDK and Cosmos ecosystem

Key Takeaways (Organizations)

- Interchain Foundation
  - Apply for funding or a job!
  - https://interchain.io/careers
  - https://interchain.io/funding
- Informal Systems
  - R&D spin out from the Interchain Foundation
  - Coming soon!
  - https://informal.systems

If you like the sound of my voice ...



Software Engineering Daily
“Consensus Systems with Ethan Buchman”

Epicenter Podcast
“Launching the Internet of Blockchains”

Rebuild Conference
“Stakeholders & State Machines”


Thank You!
https://interchain.io
https://interchain.io/careers
https://interchain.io/funding
https://informal.systems
https://cosmos.network
https://tendermint.com
COSMOS
INTERNET OF BLOCKCHAINS

Algorand: From Theory to Practice

Jing Chen, Algorand, USA

Abstract: Blockchains stand to revolutionize the way a modern society operates. They can secure all kinds of traditional transactions, such as payments, in the exact order in which the transactions occur; and enable totally new transactions, such as cryptocurrencies and smart contracts. They can remove intermediaries and usher in a new paradigm for trust. As currently implemented, however, blockchains scale poorly and cannot achieve their enormous potential. Algorand is the first blockchain that is truly secure, scalable and decentralized. It is permissionless and works in a highly asynchronous environment. It dispenses with proof of work and miners and requires only a negligible amount of computation. Moreover, its transaction history does not fork, guaranteeing immediate finality of a transaction the moment the transaction enters the blockchain. In this talk, I will introduce Algorands core technology, recent development and roadmap.

Biography: Jing Chen is Chief Scientist and Head of Theory Research at Algorand, and Assistant Professor in the Computer Science Department at Stony Brook University. Her main research interests are distributed ledgers, game theory, and algorithms. Jing received her bachelor and masters degrees in computer science from Tsinghua University, and her PhD in computer science from MIT. She did a one-year postdoc at the Institute for Advanced Study, Princeton. Jing received the NSF CAREER Award in 2016.

Transactional Data Structure Libraries

Idit Keidar, Technion, Israel

Abstract: We introduce transactions into libraries of concurrent data structures; such transactions can be used to ensure atomicity of sequences of data structure operations. By focusing on transactional access to a well-defined set of data structure operations, we strike a balance between the ease-of-programming of transactions and the efficiency of custom-tailored data structures. We exemplify this concept by designing and implementing a library supporting transactions on any number of maps, sets (implemented as skiplists), queues, stacks, producer-consumer pools, and logs. Our library offers efficient and scalable transactions, which are an order of magnitude faster than state-of-the-art transactional memory toolkits.

We further introduce nesting into our transactional data structure library. Nested transactions create checkpoints within a longer transaction, so as to limit the scope of abort. We then conduct a case study of pipelined network intrusion detection. In this benchmark, nesting improves throughput by up to 15x. Finally, we discuss cross-library nesting, namely dynamic composition of transactional data structure libraries.

(Based on joint works with Alexander Spiegelman, Gal Assa, Guy Golan-Gueta, and Hagar Meir.)

Biography: Idit Keidar received her B.Sc. (summa cum laude), M.Sc. (summa cum laude), and Ph.D. from the Hebrew University of Jerusalem in 1992, 1994, and 1998, respectively. She was a Postdoctoral Fellow at MITs Laboratory for Computer Science. She is currently a Professor at the Technions Viterbi Faculty of Electrical Engineering, where she holds the Lord Leonard Wolfson Academic Chair. She serves as the Head of the Technion Rothschild Scholars Program for Excellence, and also heads the EE Facultys EMET Excellence Program. Her research interests are in fault-tolerant distributed and concurrent algorithms and systems, theory and practice. Recently, shes mostly interested in distributed storage and concurrent data structures and transactions.

Transactional Libraries

Idit Keidar, Technion

with

Alexander Spiegelman, Guy Golan-Gueta, Gal Assa, and Hagar Meir



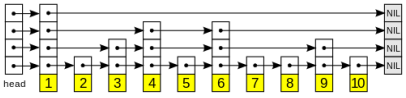

Agenda

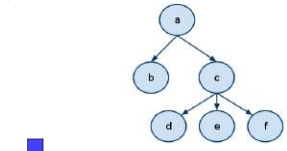
- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

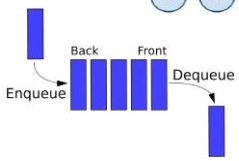
Multi-Threading is Everywhere

Data Structures (DS)

- Essential building blocks in modern SW
  - Map, skiplist, queue, etc.

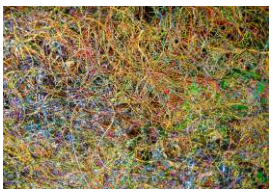






But Are They "Thread Safe"?

Correct under concurrency?



OR

?

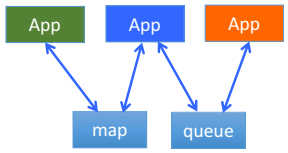
"Thread-Safe" Concurrent DS Libraries

- Widely used in real life software
- Numerous research papers
  - Concurrent Skipklist
    [Herlihy, Lev, Luchangco, Shavit: A simple optimistic skiplist algorithm] [Fraser: Practical lock freedom] …
  - Concurrent queue
    [Michael, Scott: Simple, fast, and practical nonblocking and blocking concurrent queue algorithms] [Gramoli, Guerraoui: Reusable concurrent data types]
  - Concurrent binary tree
    [Bronson, Casper, Chafi, Olukotun: A practical concurrent binary search tree] [Drachsler, Vechev, Yahav: Practical concurrent binary search trees via logical ordering]

Concurrent Data Structure Libraries (CDSLs)

- Each operation executes atomically
- Custom-tailored implementation preserves semantics

```
balance←map.get (key=Yoni)
newBalance← balance + deposit
map.set(key=Yoni, newBalance)
repQ.enq("Yoni's balance=new")
```



Are They Really Thread Safe??

```
balance ←map.get (key=Yoni)
newBalance← balance + deposit
map.set(key=Yoni, newBalance)
repQ.enq("Yoni's balance=new")
```

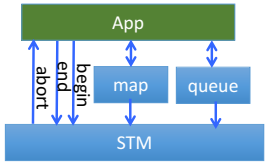
```
balance ←map.get(key=Yoni)
newBalance← balance + deposit
map.set(key=Yoni, newBalance)
repQ.enq("Yoni's balance=new")
```

- Oops! Atomic operations are not enough

Software Transactional Memory (STM)

- TL2, TinySTM, SwissTM,…
- Transactions (TXs) = atomic sections including multiple DS operations

```
Begin_TX
    balance←map.get(key=Yoni)
    newBalance ← balance + deposit
    map.set(key=Yoni, newBalance)
    repQ.enq("Yoni's balance=new")
End_TX
```



CDSL vs STM

Why?

	CDSL	STM
Performance	✓	×
Exploit DS semantics	✓	×
Used in practice	✓	×
Generality	×	✓
Composability	×	✓

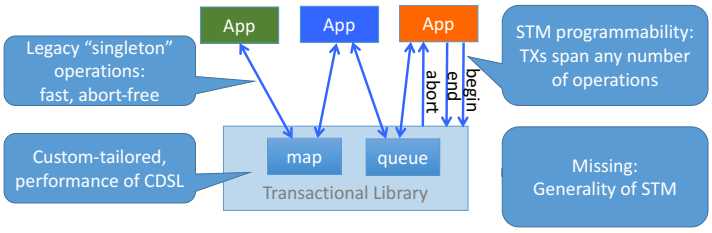
STM Overhead Explained

- Common STM solution:
  - Each object has a version
  - TX reads a "consistent snapshot"
    - Validates versions have not changed by commit time
    - Otherwise aborts
  - TX updates occur during commit
- Sources of overhead:
  - Global version clock (GVC) ⇒ contention
  - Read- and write-sets ⇒ tracking and validation overhead
  - Conflicts ⇒ aborts

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

TDSL: Bringing Transactions into CDSL



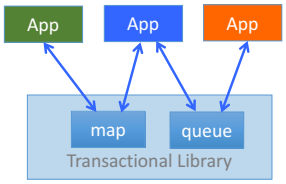
Why TDSL?

- Power of transactions
- Optimizations using DS semantics and structure
  - Use semantics to reduce overhead aborts
  - E.g., read-set reduction, non-transactional index
- Different CC mechanisms for different DSs
  - Optimistic maps, sets
  - Pessimistic queues
- Fast abort-free singletons
  - As fast as in CDSL
  - No contention on global version clock
  - Support legacy code

TDSL Benefit 1: Programmability

- Support for legacy code
  - Fast, abort-free singletons
- Power of transactions

```
Begin_TX
    val←map.get(key=Yoni)
    new← val+deposit
    map.set(key=Yoni,new)
    repQ.enq("Yoni's balance=new")
End_TX
```

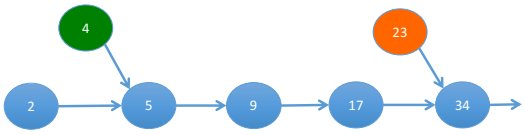


TDSL Benefit 2: Semantic Optimization

- Use known transactional solutions
- But, take advantage of semantics & structure of each DS to reduce aborts & overhead
  - Reduce read-set
  - Do some of the work non-transactionally

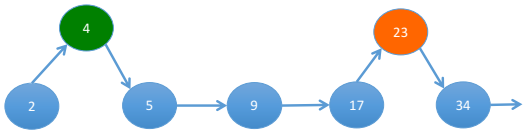
Example of Abort Reduction

- Two concurrent put operations
- Put(23) traverses the list, reads nodes that put(4) updates
  - Conflict, STM would abort at least one
  - But no semantic conflict



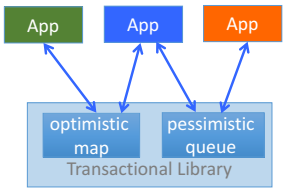
Example of Abort Reduction

- Two concurrent put operations
- Put(23) traverses the list, reading nodes put(4) updates
  - Conflict, STM would abort at least one
  - But no semantic conflict



TDSL Benefit 3: Mix & Match

- Maps allow lots of concurrency
  - Amenable to optimistic, fine-grain synchronization
- Queues do not
  - Pessimistic synchronization better
  - Coarse-grain
- STM picks one
- Our TDSL can mix & match



19

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

20

Skiplist Roadmap

1. Add STM-like transaction support to simple linked list
   - Based on TL2 STM algorithm
2. Optimization: remove redundant validation and tracking
   - Use semantics and structure
3. Optimization: shorten transactions
   - Non-transactional index
   - Lazy GC

21

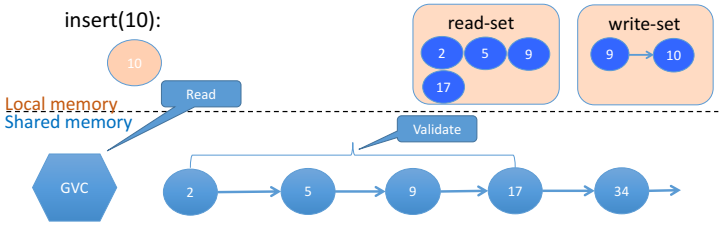
Step 1: Standard STM

- Take a simple linked list

- Add TL2 mechanism to support TXs:
  - Add a version to each node
  - Get versions from global version clock (GVC)
  - Maintain read-set with read versions
  - Defer updates, track in write-set
  - To commit: lock write-set, validate read-set, increment GCV, update, release locks

22

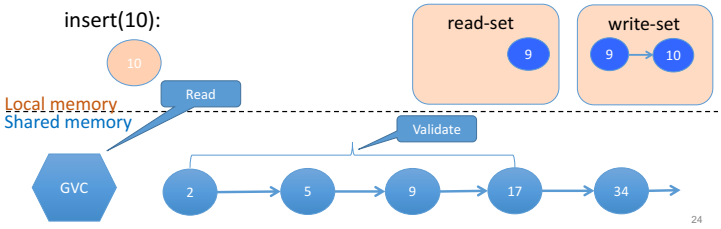
Step 1: Standard STM



23

Step 2: Reduce Read-Set

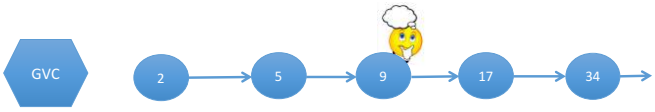
- Exploit structure and semantics



24

Step 3: Non-Transactional Index

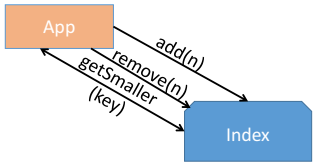
- Imagine we could guess the right node



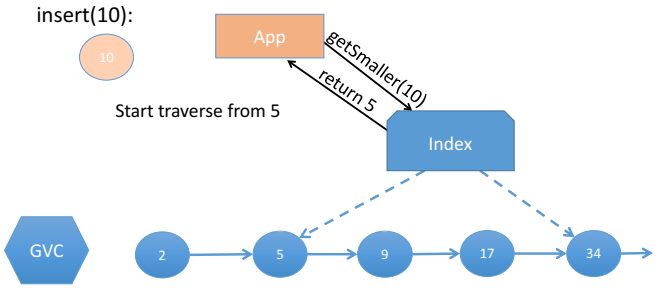
- So, we could be faster and save aborts during the traversal

Step 3: Non-Transactional Index

- getSmaller
  - Returns some node with a smaller key
  - For performance, not much smaller
- Implemented as (concurrent) skiplist
  - For example



Step 3: Non-Transactional Index



Step 3: Non-Transactional Index

- Updated outside the TX (if completes successfully)
  - Reduces aborts, overhead
- But:
  - May return nodes with smaller keys than predecessor
    - ⇒ longer traversals
  - May return removed nodes

Handling Removed Nodes

- Add *deleted* bit per node
- Commit sets this bit instead of removing the node
  - Use epoch-based memory reclamation
- Check deleted bit of node returned from index
  - If true, call getSmaller with returned node's key
  - Eventually converges
- After adding a node to the index, check if it is deleted, and if yes, remove it

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

Queues

- Optimistic dequeue likely to lead to aborts
  - Instead, use single lock and version per queue
- Pessimistic coarse-grain dequeue
  - Lock queue on first dequeue, release at commit time
  - Read from shared queue, track locally (write-set)
- Optimistic enqueue
  - Enqueue locally (in write-set)
- Commit
  - Lock queue (if needed) and update based on write-set

31

Stacks

- Optimistic while tx has pushed more than it popped
  - Push to write-set
  - Pop from write-set
- Pessimistic if at any time tx has popped more than it pushed
  - Lock stack
  - Push to write-set
  - Pop – read from shared stack, track locally
- Coarse-grain

32

Logs

- Optimistic read(pos), hasNext(pos)
  - If hasNext(pos) returns false - track in read-set "last = pos"
  - No other tracking
- Pessimistic append(val)
  - Lock the entire log
  - Track appends in write-set until commit time

33

Producer-Consumer Pools

- Fine-grain: multiple slots
- Each with its own lock
- Pessimistic produce/consume
  - Lock only the affected slot
  - Track locally until commit
- Optimistic consume from earlier produce
  - Produce and consume cancel each other out

34

Composition

- One GVC shared among all objects
  - Any number of skiplists, queues, stacks, logs, pools
- Commit:
  - Lock all write sets or respective objects
  - Validate all read sets
  - Increase GVC
  - Update objects and release locks

35

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

36

Fast Singletons

- Challenges:
  - TDSL transactions are faster than general transactions but slower than custom-tailored CDSLs
  - GVC a contention point
  - Legacy code might not be compatible with aborts
- Goal: make TDSL singletons just like CDSL operations
  - Fast, abort-free, avoid global contention point
  - Yet preserve transaction semantics – linearizability, opacity

Singletons: Avoiding GVC Contention

- GVC is needed for opacity –
  - All of a transaction's writes have the same version
  - All reads see a consistent snapshot as of some version
- Singletons read/write at a unique point in time
  - GVC gratuitous
  - But must make concurrent transactions aware of singleton updates

Singleton Updates

- Add *singleton* bit per node
  - Last update by singleton?
- Read version from GVC without incrementing it
- Use index
- Do not defer work to "commit time"
  - No read- and write- sets
  - Lock, validate unchanged/unlocked/undeleted, update, unlock
  - In case of failure, restart (do not abort)

Transaction Adjustments

- Transactions validate that no node in read-set has
  - a version equal to the transaction's snapshot version &
  - a true singleton bit
- If yes, abort and increment GVC before retrying

Fast Abort-Free Singletons: Summary

- Use the index
- Do not increment GVC
  - No contention
  - As fast as in CDSL
  - Make transactions aware of singletons using designated fields

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Additional objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

Composing TDSLs – Revised API

- TX-begin
- Divide TX-commit into
  - TX-lock
  - TX-verify
  - TX-finalize
- TX-abort

43

Static Composition

L1.BeginTX
L2.BeginTX
    Operations in L1 and L2
L1.lock
L2.lock
L1.verify
L2.verify
L1.finalize
L2.finalize

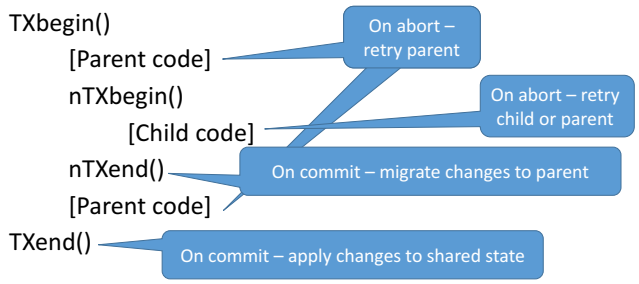
44

Dynamic Composition Requires Validation

L1.BeginTX
    Operations in L1
    L2.BeginTX
    L1.verify
        Operations in L1, L2
L1.lock
L2.lock
L1.verify
L2.verify
L1.finalize
L2.finalize

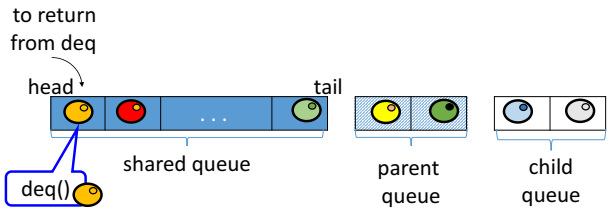
45

Nesting – Limit Scope of Abort



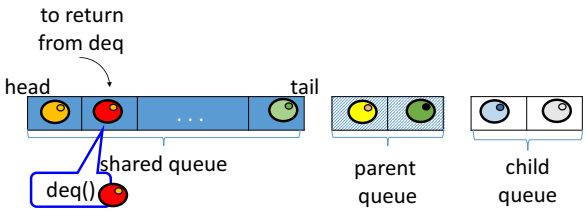
46

Example: Nested Queue



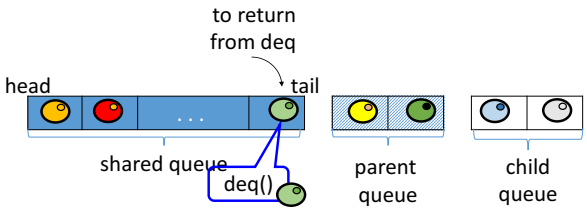
47

Example: Nested Queue



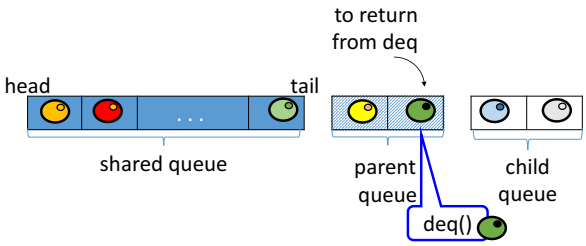
48

Example: Nested Queue



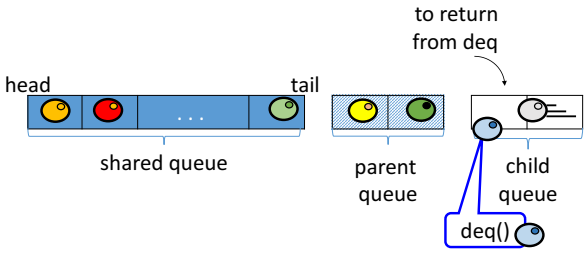
49

Example: Nested Queue



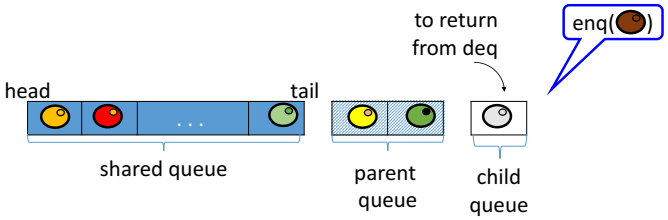
50

Example: Nested Queue



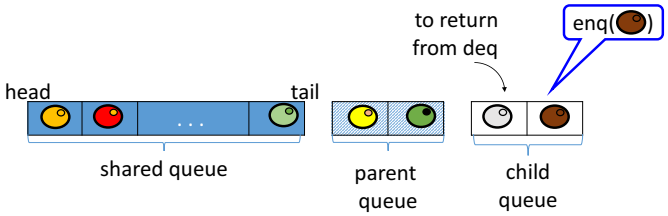
51

Example: Nested Queue



52

Example: Nested Queue



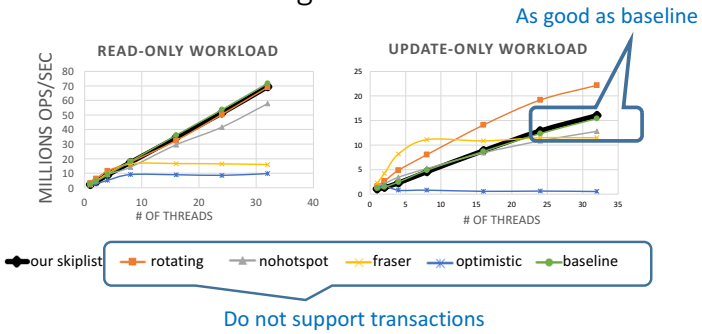
53

Agenda

- Motivation
  - Concurrent Data Structure Libraries (CDSLs) vs Transactional Memory
- Introducing: Transactional Data Structure Libraries (TDSL)
- Example TDSL algorithm
  - Skiplist
  - Composition of multiple objects
  - Fast abort-free singletons
- Library composition & nesting
- Evaluation

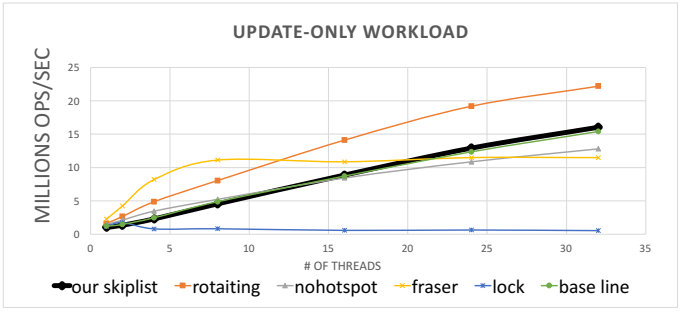
54

Singletons



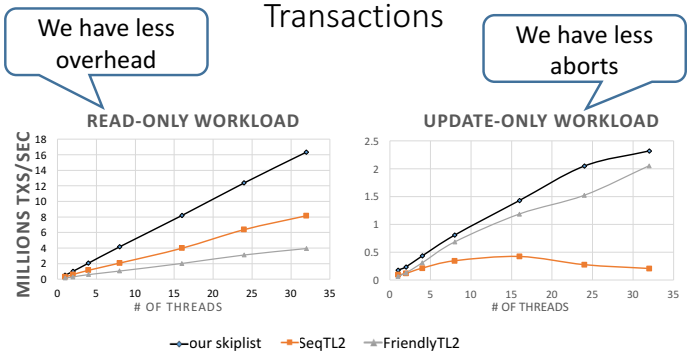
55

Singletons



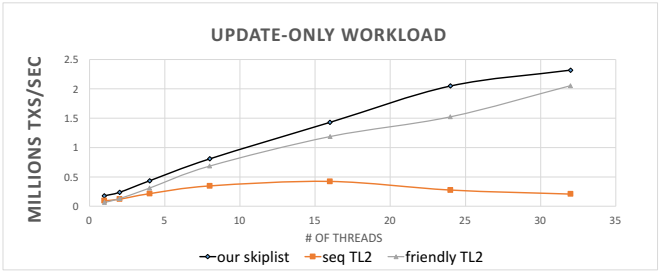
56

Transactions



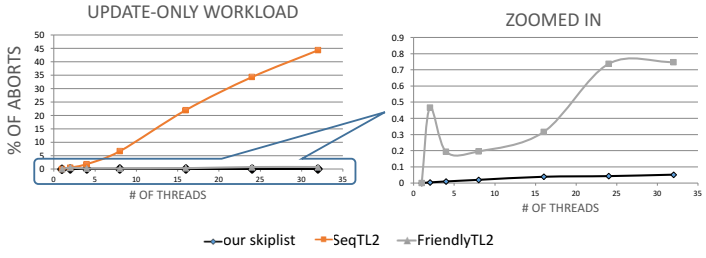
57

Transactions



58

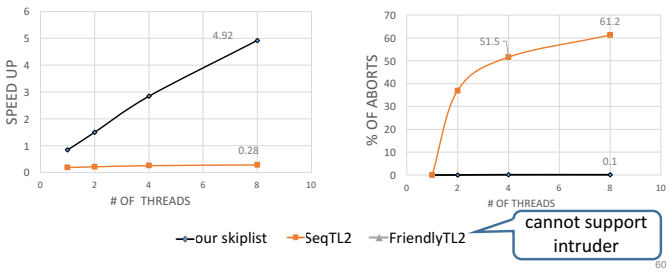
Transactions (Aborts)



59

Intruder

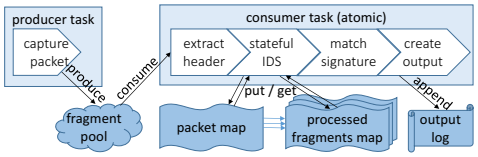
Testing multiple skiplists and queues in a real application



60

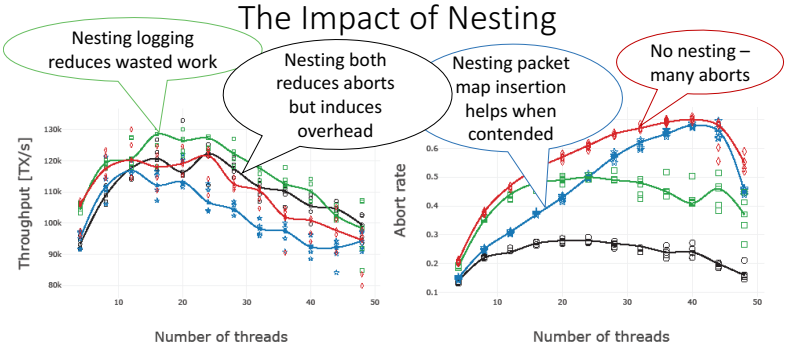
Nesting in NIDS Benchmark

- Network Intrusion Detection System
  - Producers model network-interfacing processes (stubs)
  - Consumers require CPU and DRAM resources



61

The Impact of Nesting



62

Conclusion

- New concept for concurrent programing
  - Composable DSs supporting TX as well as fast singletons
- Support for nesting
- A prototype library with a host of DSs
  - Map (based on skiplist), queue, stack, log, pool
- We hope that the community will adopt this concept
  - Build and use more such libraries

63

Weak Models for Distributed Computing

Gadi Taubenfeld, Interdisciplinary Center, Israel

Abstract: The talk has three parts. In the first part, I will show how to model the process of genome-wide epigenetic modifications, which allows cells to utilize their DNA, as an anonymous shared memory system. This is done by formulating a particular consensus problem and presenting algorithms for solving the problem. In the second part, I will discuss results for anonymous shared memory systems which are composed of shared objects for which there is no a priori agreement between the processes on the names of the objects. In the third part, I will motivate and explore the new notion of weak failures, which should be viewed as fractions of traditional failures.

Biography: Gadi Taubenfeld is a professor and past dean of the School of Computer Science at the Interdisciplinary Center in Herzliya, Israel. He is an established authority in the area of concurrent and distributed computing and has published widely in leading journals and conferences. He authored the books "Synchronization Algorithms and Concurrent Programming" and "Distributed Computing Pearls". His primary research interests are in concurrent and distributed computing. Gadi was the head of the computer science division at Israel's Open University; member of technical staff at AT&T Bell Laboratories; consultant to AT&T Labs - Research; and a research scientist and lecturer at Yale University. Gadi served as the program committee chair of PODC 2013 and DISC 2008 and holds a Ph.D. in Computer Science from the Technion - Israel Institute of Technology.

Weak Models For Distributed Computing

Gadi Taubenfeld

IDC, Israel

Weak Models For Distributed Computing

Gadi Taubenfeld

IDC, Israel

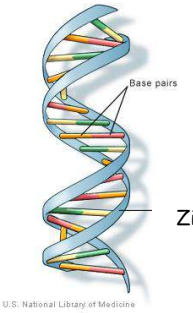
Gadi: I am not an implementor of tools, programming languages, or platforms!

Annie: … pls mention where computers can help you except from text editor and a slides editor …

New 2019

Part I

Genome-Wide Epigenetic Modifications as a Shared Memory Consensus Problem



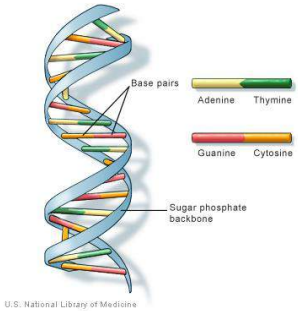
Ziv Bar-Joseph
CMU

Sabrina Rashid
CMU

Gadi Taubenfeld
IDC

The human genome

The entire DNA of a single human cell



- Two meters long
- 3 billion base pairs
- About 25,000 genes

(Only about 1 percent of DNA is made up of protein-coding genes)

Chromatin

Package DNA into a small volume to fit into the nucleus of a cell

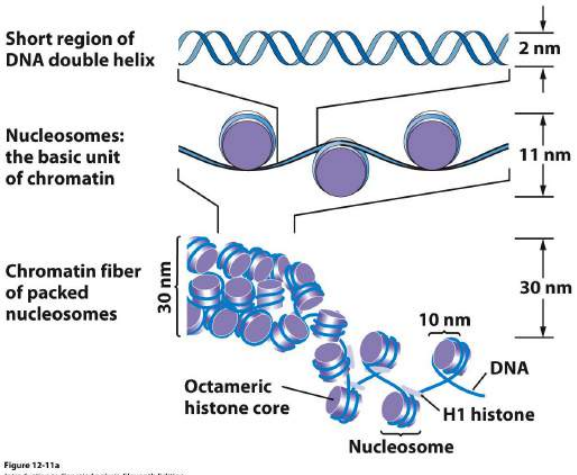
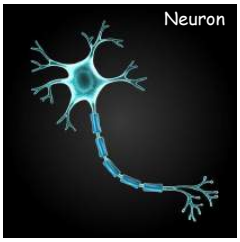
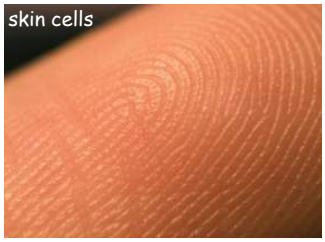


Figure 12-11a
Introduction to Genetic Analysis, Eleventh Edition
© 2015 W. H. Freeman and Company

Cells types & DNA

Q: How can an organism have different cell types yet one genome?

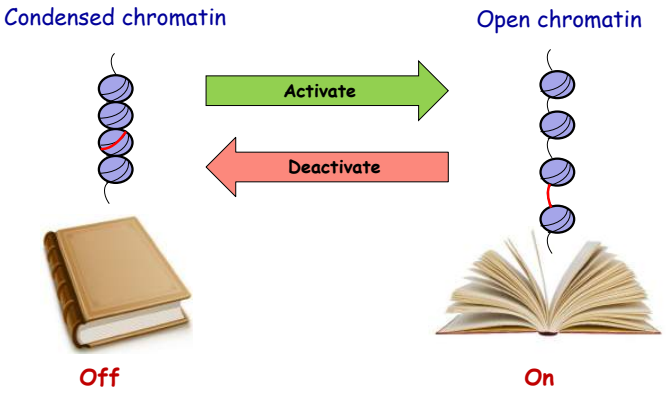


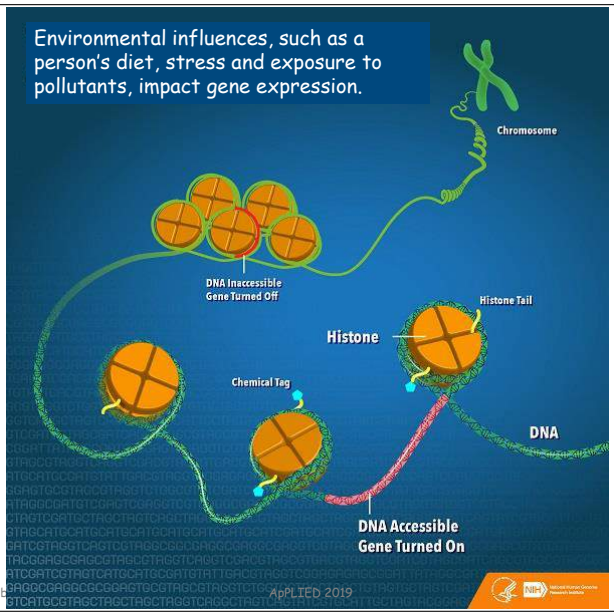


A: Each cell expresses, or turns on, only a fraction of its genes. The rest of the genes are repressed, or turned off.

Regulation of gene expression

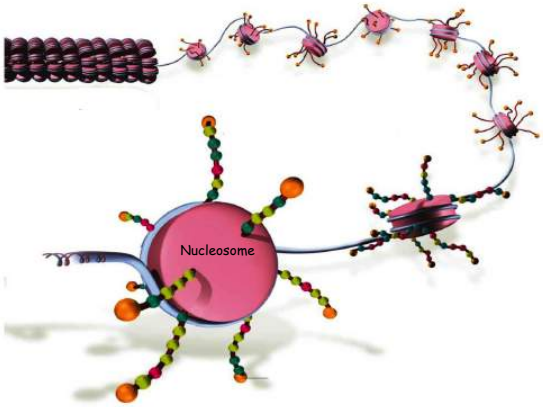
Turning genes on and off

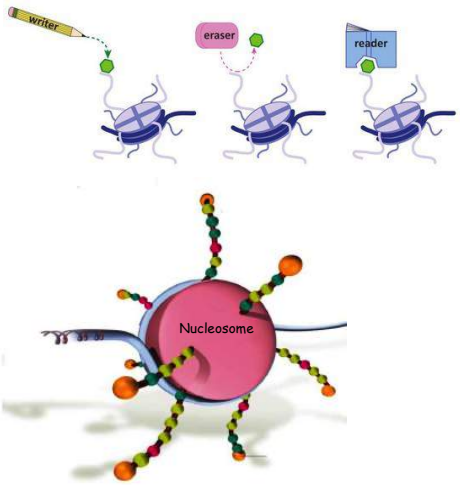


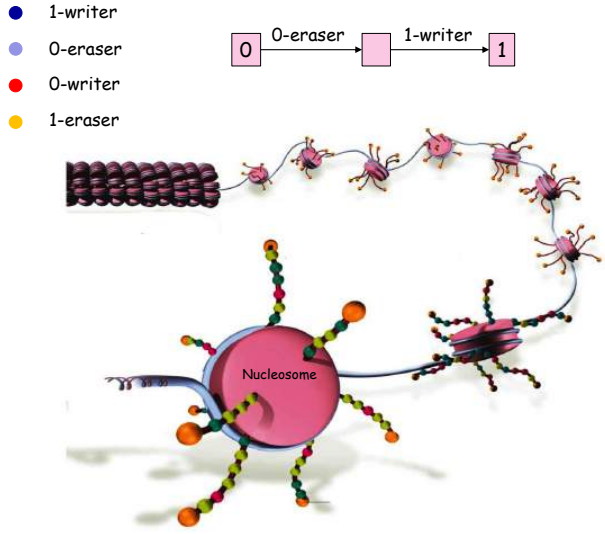


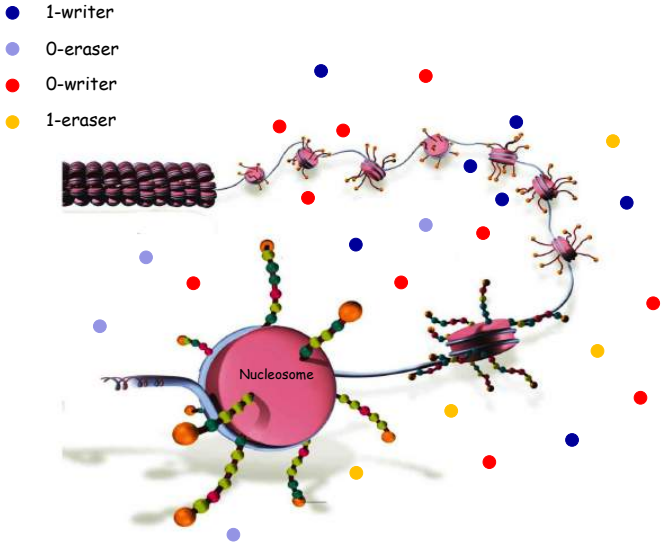
Epigenetics

Modifications that do not change the DNA and affect gene activity



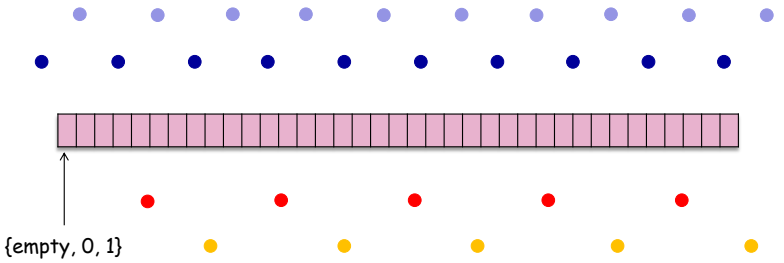






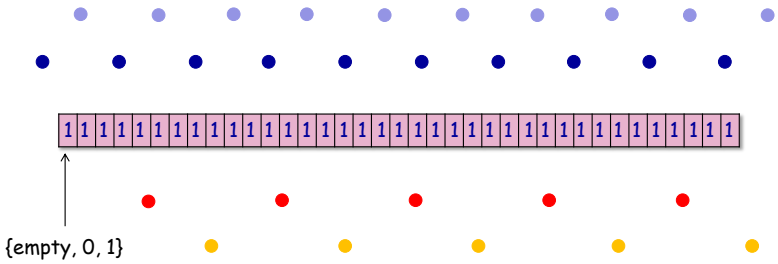
The epigenetic consensus problem

- 1-writer
- 0-eraser
- 0-writer
- 1-eraser



The epigenetic consensus problem

- 1-writer
- 0-eraser
- 0-writer
- 1-eraser



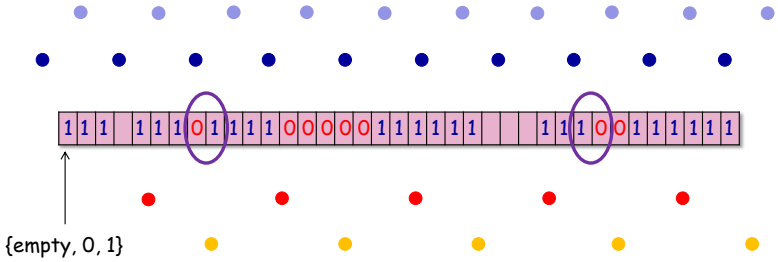
Very weak model

- Randomization
- Anonymous processes (no identifiers)
- Anonymous shared memory
- Memory-less processes (well may 1-2 bits)
- A transition from 0 to 1 cannot occur directly
- No sense of direction
- Self-stabilization

We present an algorithm that matches the biological assumptions, prove it correctness and derive bounds on its expected run time both theoretically and in simulations.

The epigenetic consensus problem

- 1-writer
- 0-eraser
- 0-writer
- 1-eraser

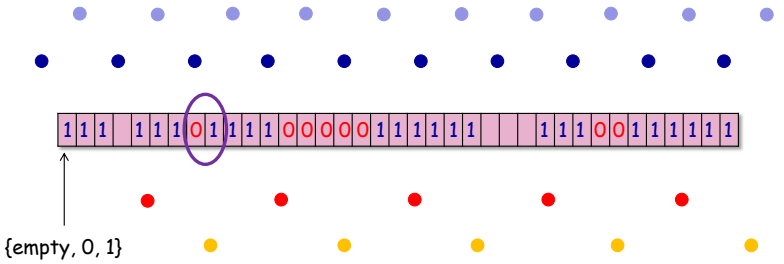


The epigenetic consensus problem

Corollary 6.2 *Assume $W_1/W_0 \geq 3$. The probability that the final decision value is 1 is more than*

$$(1-(1/3)^N) \times \left(1-e^{-N/12}\right)$$

Corollary 7.2 *Assuming $W_1/W_0 \geq 3$, $E[T] \leq 6.4N^2$*



Annie's question: Where computers can help you except from text editor and a slides editor ?

Simulations

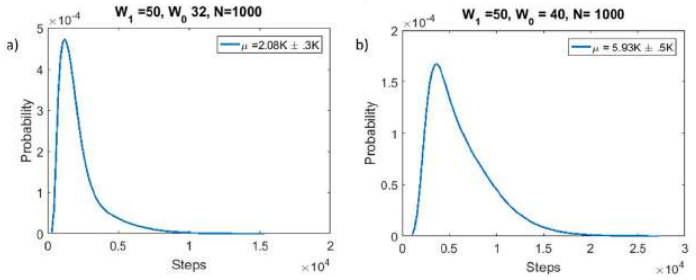


Figure 1: Distribution of number of steps to reach consensus. Plots summarize 300 random runs of the algorithm. a) low and b)high level of competitions between 1-writers and 0-writers. μ denotes the average time to reach consensus.

Simulations

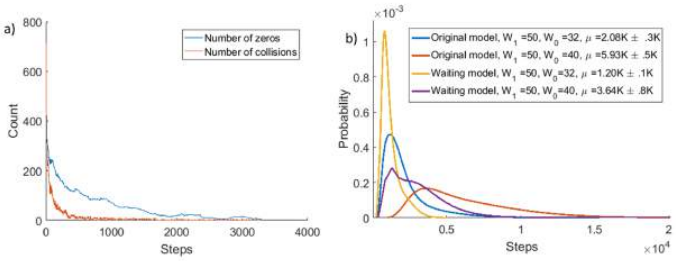


Figure 2: a) Number of zeros and collisions vs steps in the algorithm. While the initial number of collisions is a linear function of the number of 0's, we observe that towards the end of the algorithm there are very few collisions while the number of 0's remains relatively high. b) Comparison between the proposed original model and a revised model that allows writers to attach themselves with the erasers and the erasers wait until a collision is resolved. Here we can see that the waiting version is faster at both competition level compared to the original model.

Conclusion #1

Weak models are interesting!

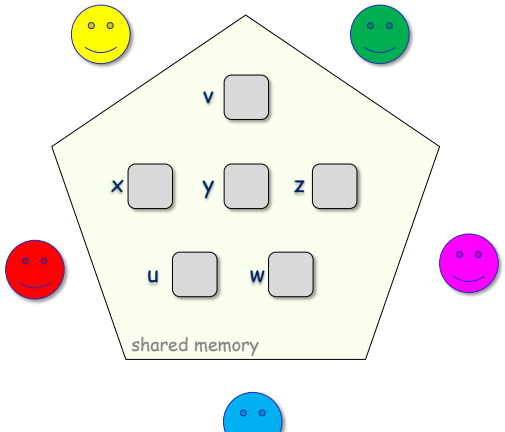
Part II

Anonymous Shared Memory

Classical view of SM

Standard Model

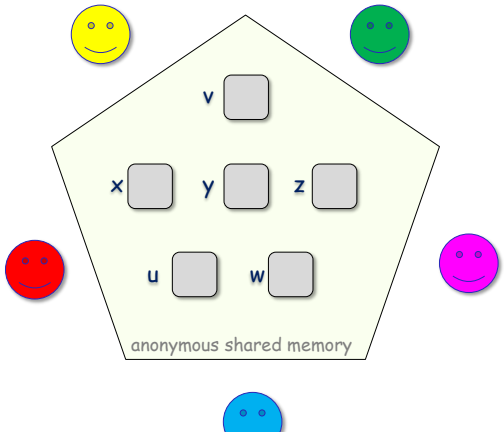
Objects have names



Anonymous shared memory

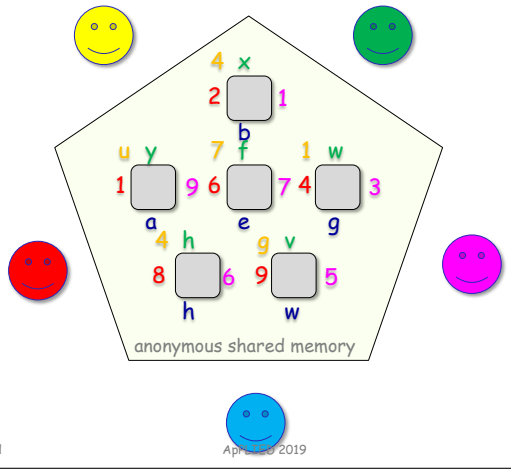
New Model

NO prior agreement on the names of the objects!



Anonymous shared memory

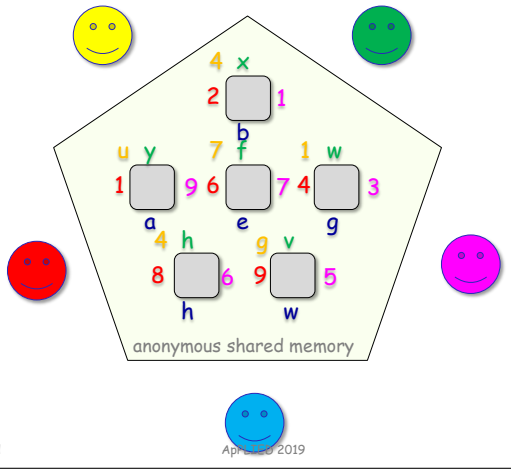
New Model



Anonymous shared memory

PODC 2017

Coordination without prior agreement
by *Gadi Taubenfeld*



Algorithms & space bounds

PODC 2017
R/W registers

Algorithms	Can do ✓	Cannot do ✗
Deadlock-free symmetric mutual exclusion for two processes	odd # of registers	even # of registers
Obstruction-free consensus for n ≥ 2 processes	2n-1 or more	n or less
Obstruction-free adaptive perfect renaming for n ≥ 2 processes	2n-1 or more	n or less

(The # of registers is not 1)

Optimal Memory-Anonymous Symmetric Deadlock-Free Mutual Exclusion

PODC 2019

- Theorem. For every n ≥ 1, there is a symmetric deadlock-free mutual exclusion algorithm for n processes using m ≥ 1 anonymous R/W registers if and only if for every positive integer 1< k ≤n, m and k are relatively prime.
- The same result holds also for RMW registers ! *

Zahra Aghazadeh, Damien Imbs, Michel Raynal, Philipp Woelfel, Gadi Taubenfeld

* It is trivial to do also with one RMW register.

Resolving two open problems

SIROCCO 2019

For a universe which includes (also) anonymous objects,

- Are atomic read/write registers the weakest objects ?
- Are deterministic (oblivious) objects with the same set-consensus number have the same computational power ?

NO

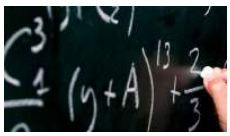
Conclusion #2

Weak models are interesting!

Part III
Fractions in Distributed Computing

Egypt 1600 B.C.

Europe 17th century



Dist. Comp. ???

Fractions were studied by Egyptians mathematicians around 1600 B.C. However, fractions, as we use them today, didn't exist in Europe until the 17th century.

Part III
Fractions in Distributed Computing

- We understand what it means to tolerate one process failure.
- But what does it mean to tolerate 0.8 process failure ?

Motivation
Something is better than nothing

- FLP: Impossibility of consensus in the presence of a single failure.

- Is consensus possible in the presence of a single weak failure?

NETYS 2018

Weak Failures: Definitions, Algorithms and Impossibility Results by *Gadi Taubenfeld*

- Is consensus possible in the presence of a single weak failure?

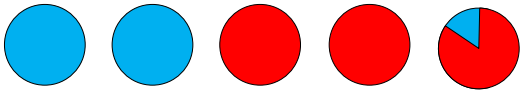
YES !!!

Motivation
Generalizing from the previous example

- Suppose you can solve a problem in the presence of f traditional failures, but not in the presence of f+1 such failures.

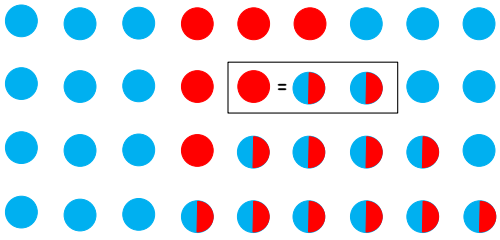
(f=2)

- Maybe it is possible to solve the problem in the presence of f traditional failures plus several weak failures.



SSS 2018

Set agreement and renaming in the presence of contention-related crash failures




Anaïs Durand


Michel Raynal


Gadi Taubenfeld

Conclusion #3

Weak models are interesting!

Approaches to Data Sharing in Edge FaaS

Zoltán Turányi, Ericsson, Hungary

Abstract: Function-as-a-service systems are stateless by nature - every function starts with no memory of what happened before. Such context is typically placed in an external database, where functions can read and update as side effect. The time to execute functions significantly depends on access time to this database - accessing data in a different location is much slower than locally. It is thus beneficial to co-locate data and execution. This is especially relevant for the Edge scenario, where remote means geographically remote. In this talk we overview some approaches, how such placement can be combined with methods to ensure data consistency among various locations. Our key goals are high performance and ease of use.

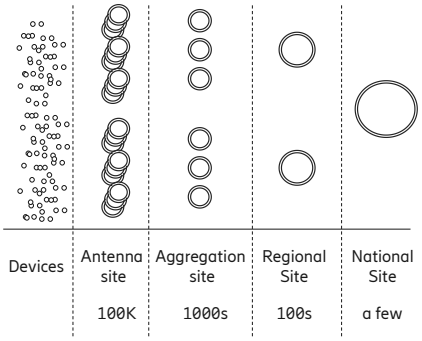
Biography: Zoltn Richard Turnyi is currently an expert of 5G Network Architectures within Ericsson Research. He currently works on a Function-as-a-Service concept built on a fast, distributed, in-memory key-value store. His background is in IP networking and mobile core with recent addition of Software Defined Networking and Network Function Virtualization. He is the author of more than 50 patent applications and works with Ericsson Research for more than 20 years. He holds an M.Sc. in Computer Science from the Technical University of Budapest. He is a scout master for 25 years.

Approaches to Data Sharing in Edge FaaS

Zoltán Richárd Turányi
Expert, Ericsson Research
Hungary

Problem Statement: Mobile Cloud Apps

- Cellular networks are hierarchical
  - Centralized componets are cheap to build and maintain
  - But for radio reasons nodes must be distributed
- Other reasons to deploy application components distributed
  - Low latency towards end-user
  - Local processing to save bandwidth
  - Fate sharing with user
- 5G introduces new, low-latency modes
  - Ultra Reliability Low Latency Communication (URLLC)

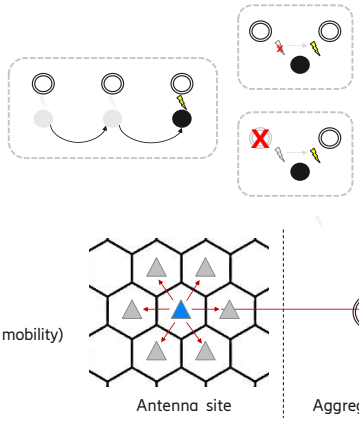


Low latency use cases

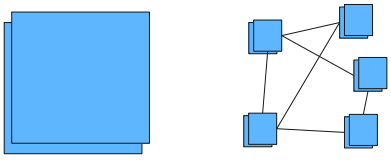
1. Cloud Virtual & Augmented Reality – Real-time Computer Rendering Gaming/Modeling
2. Connected Automotive – ToD, Platooning, Autonomous Driving
3. Smart Manufacturing – Cloud Based Wireless Robot Control
4. Connected Energy – Feeder Automation
5. Wireless eHealth – Remote Diagnosis With Force-Feedback
6. Wireless Home Entertainment – UHD 8K Video & Cloud Gaming
7. Connected Drones – Professional Inspection & Security

Mobility

- Users move
  - Physical mobility
  - Change in radio conditions
  - Node and link failures
- States related to users need to be
  - moved
  - replicated
- Replicaton can be
  - To neighbouring edge sites (handy at mobility)
  - To central site

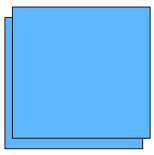


Problem Statement: Function-as-a-Service

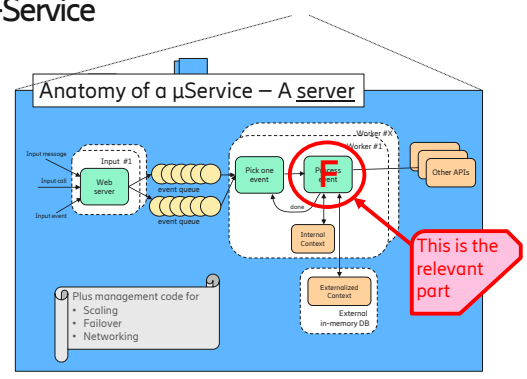


Monolithic apps MicroServices

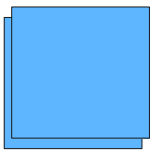
Problem Statement: Function-as-a-Service



Monolithic apps

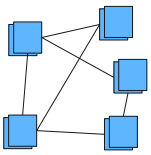


Problem Statement: Function-as-a-Service



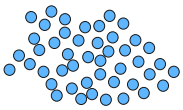
Monolithic apps

- All-in-one
- Scales in big blocks
- Upgrades monolithically



MicroServices

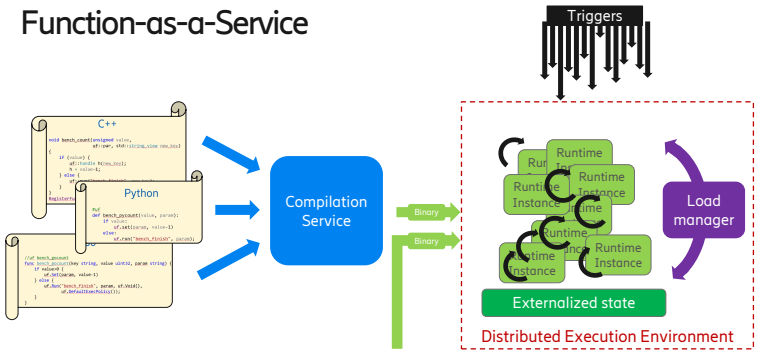
- Loosely coupled (hard)
- Data enclosed
- Overhead: Web servers, HTTP, sidecars
- Individual scaling, failover
- Developers do a lot besides business logic



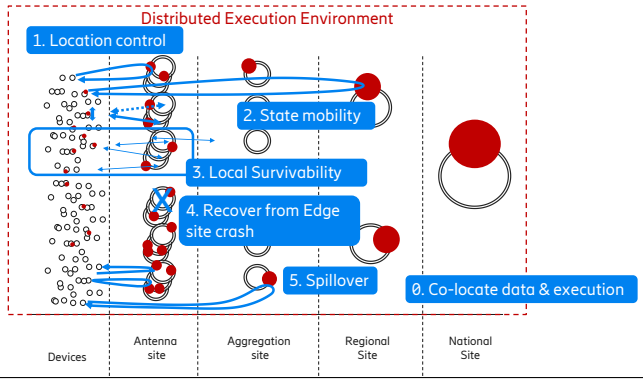
Functions

- Externalized state
- Developer focus
- Platform does scaling, failover
- Very fluid
- Full interworking with uServices

Problem Statement: Function-as-a-Service

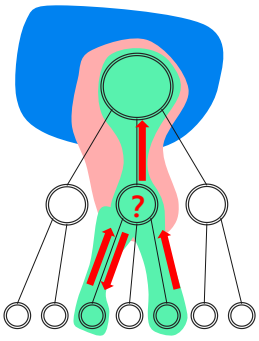


Problem Statement: Mobile FaaS Apps



CloudPath

- Hierarchical execution model: nodes may have children and parents
  - Children are usually less capable than parents
- Developers may mark functions to execute at specific hierarchy levels
- PathStore
  - Children cache a part of the parent's database locally
    - The root has everything
  - Reads fetch the relevant part (and subscribe updates)
    - Cold entries are automatically removed
  - Writes take effect locally then propagate upwards
    - Tightly synchronized GPS clocks are used to timestamp writes
    - Write conflicts are resolved using the timestamps
  - Eventually consistent

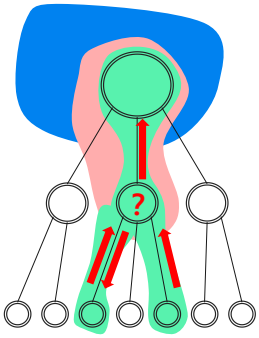


CloudPath: A Multi-Tier Cloud Computing Framework
Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, Eyal de Lara
2nd ACM/IEEE Symposium on Edge Computing (SEC), San Jose, CA, October 2017

CloudPath

- Good reliability
  - Data is stored at multiple levels
- Fast reads after caching
- Fast local writes
- Possible to add mobility
- May handle local survivability
  - If all needed data is locally cached

- Does not handle simultaneous writes very well
- No atomic updates possible (like a counter)



CloudPath: A Multi-Tier Cloud Computing Framework
Seyed Hossein Mortazavi, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips, Eyal de Lara
2nd ACM/IEEE Symposium on Edge Computing (SEC), San Jose, CA, October 2017

What should the ideal database be like?

Note: possible to have more than one in an app.

Assumptions

- Most requests come from the edge
  - Goal is to serve these fast
- Execution
  - Functions are short lived and partake serving one request
  - Function Execution is possible everywhere
  - Already running functions do not move
- Database has the ability to move data around
  - Result in two phase lookups
  - Distributed hash tables are out
  - Caching of locations and subscribing to location updates help

Merging or serializing database

- Merging
  - Let local writes diverge the history
  - Merge changes in a distributed fashion

Super fast locally
Good merging strategy is needed.
Application dependent, custom merging logic.

- Serializing
  - Maintain a logical order of updates same everywhere
  - Results in a single location handling all updates for the same data

Super fast locally at master site.
Slow remotely.
Good with dominant accessor.
Versioning enables atomic read-update-write operations.

Replication

- Conflicting requirements
  - Handle Edge site failure
  - Provide Local Survivability

- Inter-site and intra site
- Robustness
  - Wait for replication to complete; or
  - Proceed logic in the meantime
- Location
  - From Edge to Central
  - Edge to Edge
    - Predict mobility or not
    - Controlled handover process

Fine control is needed by the programmer.
- Future-like mechanisms to have writes in parallel
- API to control replica locations & master mobility

Function Execution Location

- Programmer may designate both data and execution in the system by hand
  - Does not support e.g., spillover or edge site failure
- Two kind of automatic strategies
- Function mobility
  - Move the function's execution where its data is
  - Need to know what kind of data the function accesses
    - Provided by developer, Statically analysed, Measured
- Data mobility
  - Move the data to where functions execute
  - Best if there is some consistent execution of functions (including sharding)
  - Data may migrate to servers not functions

As simple as falling back to centralized execution if data not available locally

Optimization:
Co-locate functions working on same data

Multi-key or single-key transactions

- Single-key transactions
  - Each transaction affects only one addressable data element
    - E.g., plain Key-Value stores

Workarounds
- Large, composite values
- Multi-step updates via 'lock' keys
  1. Write into a key to take a lock
  2. Update several keys
  3. Release the lock

- Multi-key transactions
  - Easy to program
  - Difficult and complex to implement
  - Very slow for data scattered all around

We can send code around
- Decompose transaction code
- Execute close to data in parallel
- Have the ability to roll back if needed

Summary

- 5G and Edge computing will enable many exciting low-latency use cases
- FaaS is emergent programming paradigm for the Cloud
- Selecting the right external database for Edge FaaS is a challenge

To build, or Not to Build, That Is the Question

Nitin Vaidya, Georgetown University, USA

Abstract: The talk will explore necessity and sufficiency of experimental evaluations, using examples of past projects, and make suggestions for future directions in experimental research in distributed computing.

Biography: Nitin Vaidya is the Robert L. McDevitt, K.S.G., K.C.H.S. and Catherine H. McDevitt L.C.H.S. Chair of Computer Science at Georgetown University. He received Ph.D. from the University of Massachusetts at Amherst. He previously served as a Professor and Associate Head in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He has co-authored papers that received awards at several conferences, including 2015 SSS, 2007 ACM MobiHoc and 1998 ACM MobiCom. He is a fellow of the IEEE. He has served as the Chair of the Steering Committee for the ACM PODC conference, as the Editor-in-Chief for the IEEE Transactions on Mobile Computing, and as the Editor-in-Chief for ACM SIGMOBILE publication MC2R.

Invited Talk, ApPLIED Workshop @ DISC 2019, Budapest

To Build, Or Not To Build, That Is The Question …

Nitin Vaidya
Georgetown University

Outline

- Brief overview of past work
- Some thoughts on building testbeds
- PG-13

Caveat

- I don't always follow my own advice

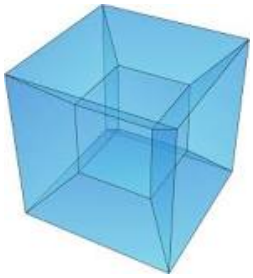
Brief History of Time

ECE @ UMass Ph.D
➔ CS @ Texas A&M
➔ ECE @ UIUC
➔ CS @ Georgetown

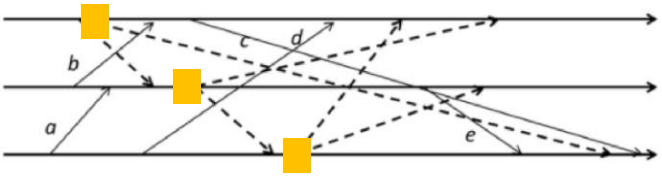
Fault-tolerant computing
➔ Wireless networks … systems
➔ Distributed algorithms … theory

Fault-Tolerance

Checkpointing & Rollback Recovery

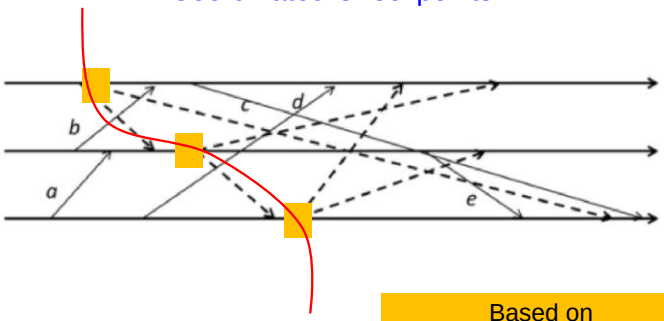


Coordinated Checkpoints



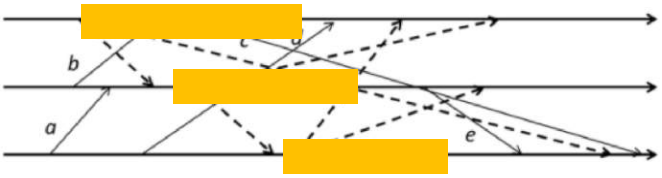
Application messages
Control messages

Coordinated Checkpoints

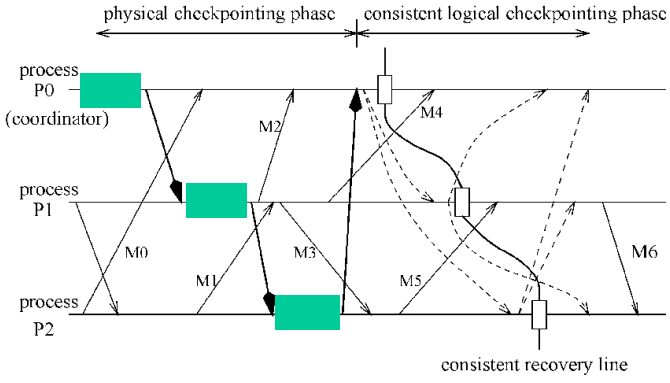


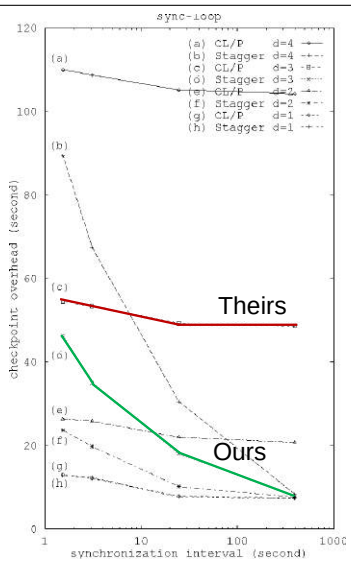
Based on
Chandy-Lamport Snapshot

Coordinated Checkpoints



Consistent Logical Checkpoints
Staggered Checkpointing





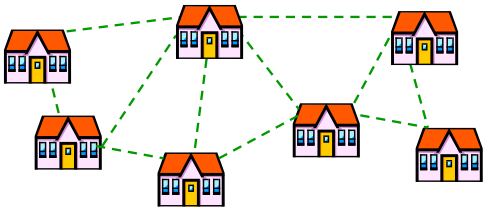
12

Multi-Level Checkpointing

- Different (cost) checkpoints for different faults

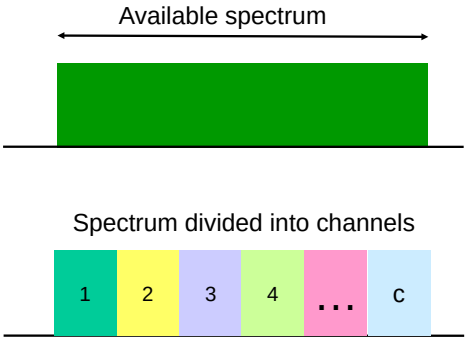
13

Mesh Networks

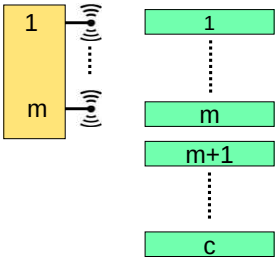


15

Multi-Channel Systems

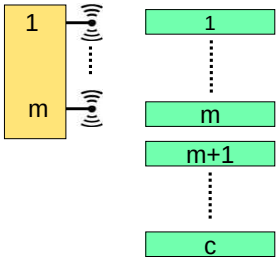


Practical Scenario



c–m unused channels at each node

Practical Scenario

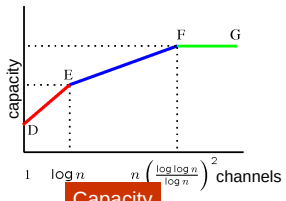


c–m unused channels at each node

How does mesh performance scale with c and m ?

Net-X: Multi-Channel Mesh

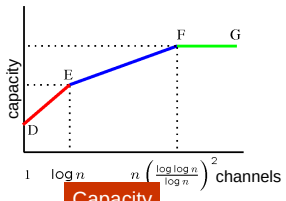
Theory to Practice



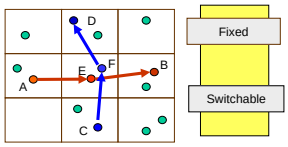
Capacity bounds

Net-X: Multi-Channel Mesh

Theory to Practice



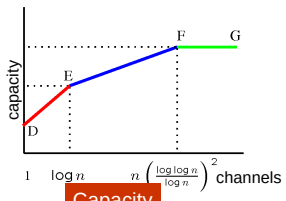
Capacity bounds



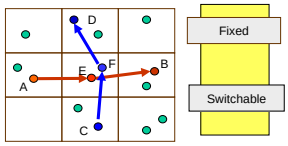
Insights on protocol design

Net-X: Multi-Channel Mesh

Theory to Practice

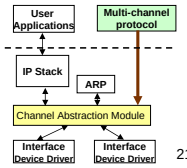


Capacity bounds



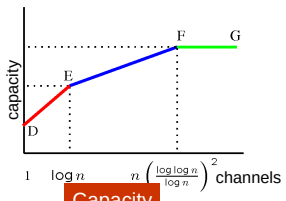
Insights on protocol design

OS improvements Software architecture

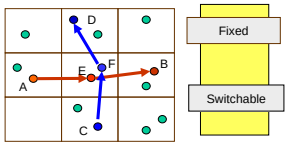


Net-X: Multi-Channel Mesh

Theory to Practice



Capacity bounds

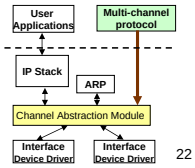


Insights on protocol design

OS improvements Software architecture

Net-X testbed

Linux box



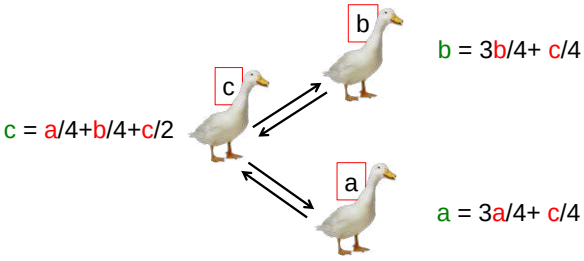
Distributed Algorithms

"Local Computations"

Average Consensus

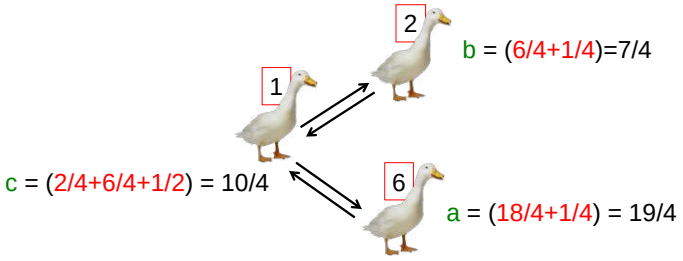
Average Consensus

Initially, state = input



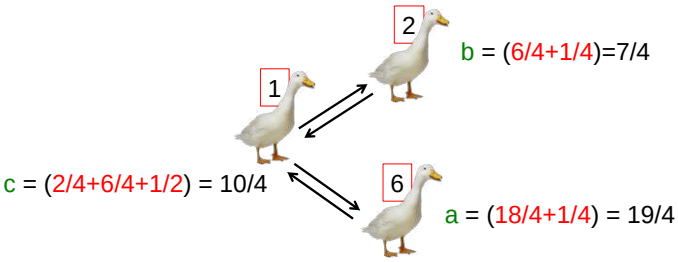
25

Average Consensus



Average Consensus

Values converge to *average* of inputs



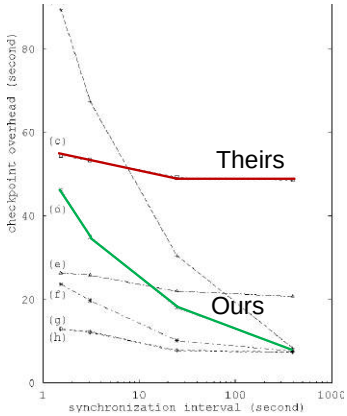
Implementing Local Algorithms

- Too much work to implement (in wireless networks)
- Software environment to make life easier
- Programmer provides pseudo-code
- Rest automated

28

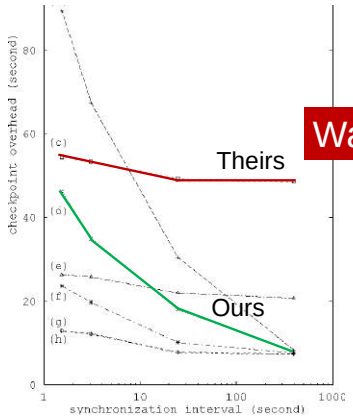
To Build, Or Not To Build,
That Is The Question …

Staggered Checkpointing



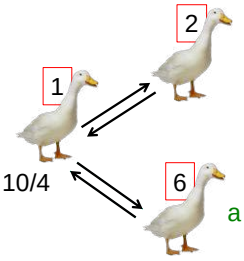
30

Staggered Checkpointing



Waste of time

Average Consensus
Software Toolkit for Local Computations



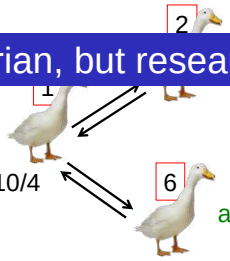
b = (6/4+1/4)=7/4

c = (2/4+6/4+1/2) = 10/4

a = (18/4+1/4) = 19/4

Average Consensus
Software Toolkit for Local Computations

Utilitarian, but research value limited

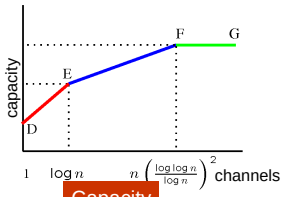


c = (2/4+6/4+1/2) = 10/4

a = (18/4+1/4) = 19/4

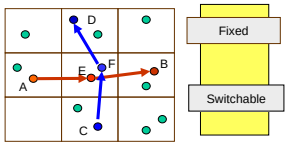
Net-X: Multi-Channel Mesh

Theory to Practice

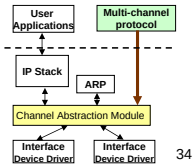


Capacity bounds

Insights on protocol design



OS improvements
Software architecture

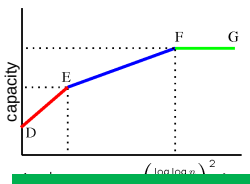


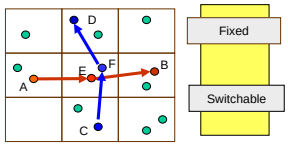
Net-X testbed

Linux box

Net-X: Multi-Channel Mesh

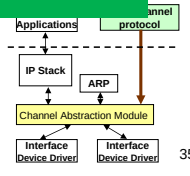
Theory to Practice



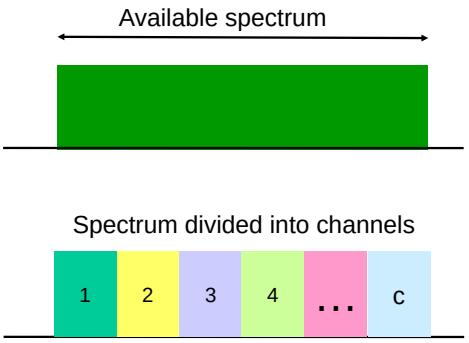


Best Case Scenario

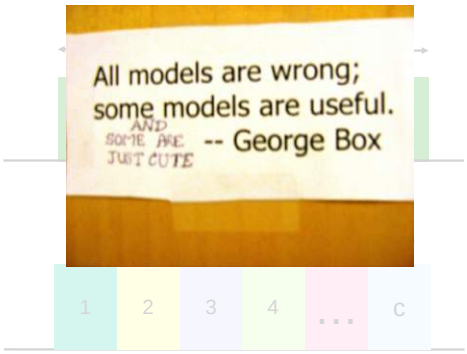
Linux box

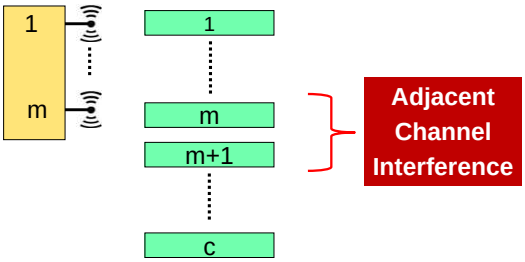


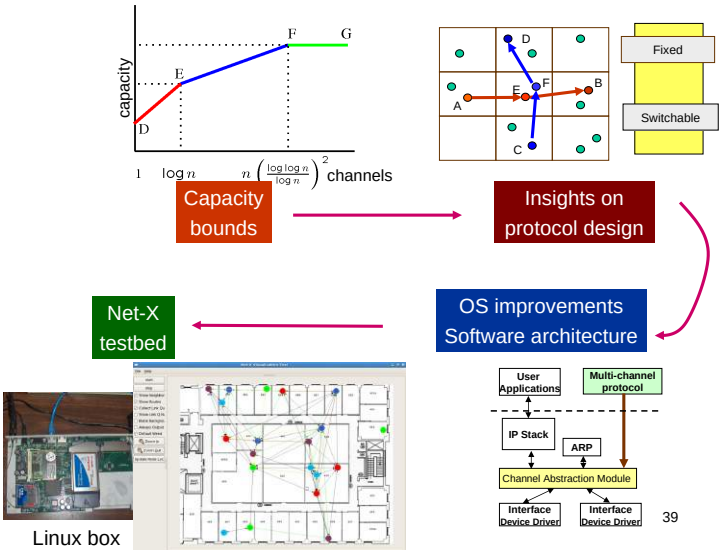
Multi-Channel Systems

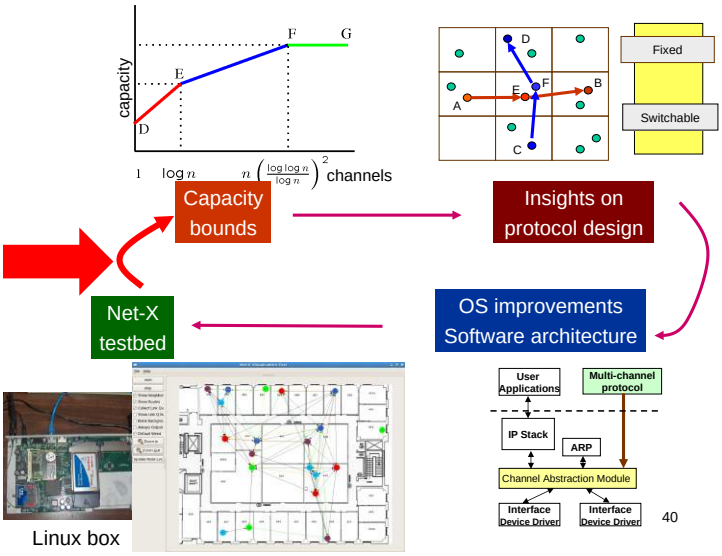


Multi-Channel Systems









When To Build

When To Build

Theory

Systems

When To Build

- When results are not predictable from theory
- For theory & simulations to suffice, need accurate system & workload models

When To Build

- When results are not predictable from theory
- For theory & simulations to suffice, need accurate system & workload models

➔ Academic architects rarely build physical systems anymore

Bad Reasons To Build

Bad Reasons To Build

- So we can publish the paper

Bad Reasons To Build

- So we can publish the paper ← Much of the "systems" literature

Bad Reasons To Build

- So we can publish the paper ← Much of the "systems" literature
- Make simple ideas appear "substantive"

Bad Reasons To Build

- So we can publish the paper ← Much of the "systems" literature
- Make simple ideas appear "substantive"
- Everybody is doing it
  … so what's wrong with you?

Bad Reasons To Build

- So we can publish the paper ← Much of the "systems" literature
- Make simple ideas appear "substantive"
- Everybody is doing it
  … so what's wrong with you? ← Funding agencies prone to this

"The MSR Effect" *

"The MSR Effect" *

* Disclaimers:
  Replace your favorite lab here
  Some of my best friends are at MSR

"The MSR Effect"

- In the *good old days*, industry research labs aspired to do relevant but academic quality research
  - Fundamental research
  - Long timescales
  - "Independence" from products

"The MSR Effect"

- In the *good old days*, industry research labs aspired to do relevant but academic quality research
  - Fundamental research
  - Long timescales
  - "Independence" from products

Today …

- Research labs dominate many conferences
- Academics aspire to emulate industry labs
  ➔ "Systems" communities have succumbed to this

How to unwind this clock?

Break Artificial Boundaries

Theory

Systems

Minimalism

- Often less is more
- Don't build just because you can
- There may be better things to do with your time and resources

57

Litmus Test

- Would you be willing to publicly post the *exact problem statement* ?

  … before developing the solution

- If not, find something better to do

Thanks!

disc.georgetown.domains

Invited Talk Summaries

Algorand: from Theory to Practice

Jing Chen, Algorand Inc., USA

Algorand: from Theory to Practice

Jing Chen
Algorand Inc., Boston, MA 02116
jing@algorand.com

Abstract

A summary of my talk at the ApPLIED Workshop at DISC 2019.

Introduction

Blockchains stand to revolutionize the way a modern society operates. They can secure all kinds of traditional transactions, such as payments, in the exact order in which the transactions occur; and enable totally new transactions, such as cryptocurrencies and smart contracts. They can remove intermediaries and usher in a new paradigm for trust. As currently implemented, however, blockchains scale poorly and cannot achieve their enormous potential. Algorand is the first blockchain that is truly secure, scalable and decentralized. It is permissionless and works in a highly asynchronous environment. It dispenses with "proof of work" and "miners" and requires only a negligible amount of computation. Moreover, its transaction history does not "fork", guaranteeing immediate finality of a transaction the moment the transaction enters the blockchain. In this talk, I will briefly introduce Algorand's core technology, recent development and roadmap. The readers may refer to [7, 8, 4, 6] for more details.

Underlying the Algorand blockchain is a new Byzantine Agreement protocol that is highly efficient. It works under an adversarial model where the adversary can dynamically corrupt any user at any time, control the actions of a corrupted user, and perfectly coordinate the actions of all corrupted users. Even with such a strong adversary, the protocol achieves asynchronous safety and guarantees that the Algorand blockchain doesn't fork even when the underlying propagation network is partitioned. Accordingly, any transaction

that appears on the blockchain is immediately final and can be relied upon, without the need of waiting for more blocks being added after it. On the other hand, the protocol achieves liveness as long as messages propagated by honest users are received by other honest users within a known time bound.

Another important idea that makes the Algorand blockchain scalable is *cryptographic self-selection*. Indeed, having millions of users participate in the Byzantine Agreement in order to select the next block is unrealistic. One possibility is to *publicly* select, at random, a subset of users to form a *committee* and participate on behalf of everybody. However, once the identities of the committee members become public, the adversary can corrupt them so that they behave maliciously. Instead, the Algorand blockchain has each user self-select herself into a committee. Thanks to cryptographic primitives such as unique signatures, cryptographic hash functions and verifiable random functions, a user can privately generate a unique "lottery ticket", which she can use to prove her membership in the committee if she is selected, but cannot cheat and convince others to accept her proof otherwise. When participating in the Byzantine agreement, a committee member propagates her winning ticket together with her proposal or voting message. No other communication is needed to find out who is selected. In this way, the adversary learns the fact that a particular user is selected only after the user has sent out her message in the protocol instead of before, and corrupting the user doesn't let the adversary control the message being sent out.

One more idea is needed here. If a selected committee participates in multiple steps of the Byzantine agreement, the adversary can learn the identities of its members and corrupt all of them after seeing their first messages, so that they behave maliciously in remaining steps. The Algorand blockchain is immune to this problem because its Byzantine agreement has an important property referred to as *user replaceability*: the protocol doesn't rely on users keeping private states, and the message that a user should send in a step can be determined solely based on messages that have been propagated to him/her in previous each steps. As such, the protocol has a committee randomly and independently selected for every step. Corrupting the committee members for a step does not give the adversary more power than random corruption in terms of controlling committee members for future steps.

In order to deal with Sybil attacks, the selection probability is the same for every token: that is, in effect, tokens are selected at random, and users that own the selected tokens participate in the Byzantine agreement. The users do not need to delegate their right of participation to a small group of

super nodes, neither do they need to lock up their tokens for a long time in order to participate in the consensus protocol. Indeed, the approach here is a pure form of proof-of-stake.

Many other ideas have been introduced in the Algorand blockchain, but rather than covering them all today, I'd like to report on some recent developments on actually implementing the blockchain and putting it to work. The Algorand MainNet launched in mid-June 2019, and has been running smoothly since. The TestNet has been running since April, 2019; it runs the same version of the blockchain as the MainNet, so that developers can test their software (such as wallets) before running them on the MainNet. The code has been audited by third parties, and the entire code repository is open-sourced and available at [3]. Various tools for developers, such as SDKs for multiple programming languages and tutorials, can be found at [2]. We continue to enlarge the tool set, and have also launched a bug bounty program [1]. In addition to a pen-and-paper analysis, with collaborators at Runtime Verification, we have begun formal verification of the consensus protocol using the Coq proof assistant. Our model explicitly incorporates timing issues and adversarial actions, reflecting a more realistic environment that may be faced by a public blockchain. We have proved asynchronous safety of the protocol under this model, and a paper reporting on the progress is available at [5].

Since the MainNet launch, we continue to develop new technology to enable important applications on the blockchain. For example, in forthcoming versions, users will be able to issue their own fungible tokens directly in layer-1 of the blockchain. Moreover, users will be able to clear multiple transfers, among arbitrary sets of users and for arbitrary sets of layer-1 currencies, in a single transaction: that is, in a truly atomic way without relying on devices such as hashed time-locks. And there is still more to come. Interested readers can find the most up-to-date information at `https://www.algorand.com/`. Stay tuned!

References

[1] `https://bugcrowd.com/algorand`.

[2] `https://developer.algorand.org/`.

[3] `https://github.com/algorand/go-algorand`.

[4] Algorand blockchain features. https://github.com/algorandfoundation/specs/blob/master/overview/Algorand_v1_spec-2.pdf, 2019.

[5] M. A. Alturki, J. Chen, V. Luchangco, B. Moore, K. Palmskog, L. Peña, and G. Roşu. Towards a verified model of the Algorand consensus protocol in Coq. In *FMBC'19: Workshop on Formal Methods for Blockchains, 3rd Formal Methods World Congress*, 2019.

[6] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos. Algorand Agreement: Super fast and partition resilient Byzantine agreement. Cryptology ePrint Archive, Report 2018/377, 2018. https://eprint.iacr.org/2018/377.

[7] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[8] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

Panel and Discussion Summaries

These sessions were well attended and had live and involved discussions. During each session, discussion questions are addressed by each panelist or expert and also discussed among all participants.

Various aspects of the current state were discussed, from tools for simulation to work on verification. The conclusion is that, overall, a lot of progress is being made, but significant future work is needed.

Invited panel: Challenges and Current State

Panelists: Chryssis Georgiou, Zoltán Tuányi, and Nitin Vaidya

Chair: Miguel Matos

Discussion questions:

1. On the processes and practices from designing an algorithm to developing a prototype implementation suitable for running on real systems. What are the current practices and best current practices?

2. On languages, libraries, platforms, and tools for distributed algorithms and systems, addressing correctness, performance, ease of use, and result reproducibility. What are the current ones and best current ones?

Discussion session: Consensus and Distribute Ledger: Scalability and Assurance

Chair: Annie Liu

Discussion questions:

1. What is the current state and problems for scalability?

2. What is the current state and problems for assurance?

Invited panel and open discussion: Summary and Directions

Panelists: Ethan Buchman, Jing Chen, and Gadi Taubenfeld

Chair: Annie Liu

Discussion questions:

1. What are the learned lessons on the design and evaluation of distributed algorithms and systems? Include both positive and negative lessons.

2. What are future directions? Consider both short term and longer terms.

Short Papers

plcli - a Tool for Running Distributed Applications on PlanetLab

Axel Niklasson, Chalmers University of Technology, Sweden

plcli - a Tool for Running Distributed Applications on PlanetLab

Axel Niklasson
axelni@student.chalmers.se
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden

Abstract

Implementing and testing distributed applications on platforms with many servers such as PlanetLab is made easier through the help of tooling. Various solutions exist, but most tools are either outdated or lack sufficient documentation. In this paper, a tool referred to as *plcli* is introduced which, among other things, enables engineers working with PlanetLab to deploy and run distributed applications on PlanetLab servers. The intended functionality of *plcli* has been summarised in three use cases; running distributed experiments, distributed debugging and node health monitoring. Furthermore, a pilot implementation written in the programming language Go is offered with this paper that implements support for experiment deployment. To see how *plcli* performs, an evaluation has been carried out that has shown that the deployment time is nearly kept constant as the number of application instances and servers are scaled up to as much as 120 instances on fifteen servers. For example, a 400% increase in the number of servers only resulted in a 7% increase in deployment time.

1 Introduction

When implementing and testing applications that are meant to run as distributed systems, the need for tooling to deploy these applications arise. In order to understand how the application in question performs as a real system, it must be deployed as such; while simulating a physically distributed system using tools such as NS-3 [1] [2] is a great alternative during the development phase, testing must be performed in a real-world environment as well. Testing applications for production usage often requires ten or more servers to be part of the deployments, which emphasises the importance of automating the process of deployment. Manually connecting to servers or residing to ad-hoc implemented scripts is not a scalable solution, which is what this kind of tooling aims to provide.

This paper focuses on applications and experiments run on the PlanetLab EU platform [3]. PlanetLab is a platform used for deploying, running and accessing distributed applications in a planetary-scale system [4] [5] [6]. More than 300 universities and research institutes, referred to as *sites*, are providing servers to the network. These servers are called *nodes* and are what makes out the computing capacity of the enormous cluster that is PlanetLab. Users are assigned a *slice*, which essentially is a distributed virtualised virtual machine, that they can use to deploy services to. Since nodes are provided by the

different sites, no guarantees on homogeneity on the machines can be given and downtime is to be expected. The varying quality of nodes makes PlanetLab a suitable platform for testing systems in a real-world setting. However, users of this platform need to make sure that desired nodes are actually functioning as intended, which can at times be tedious work.

1.1 Related work

There are public user tools listed on the PlanetLab website that may be used to decrease workload and enable easier usage when working with PlanetLab [7]. The listed tools offer various functionalities such as slice management, package management and others. Most of the tools (8/12) are not available anymore and the list is gravely outdated, but two tools that are still accessible and exhibit similar functionality as *plcli* are *PLDeploy* and *pssh*. PLDeploy functions as a "utility to deploy, configure and control PlanetLab services" which is rather similar to Use Case 1, presented in Section 1.2. However, when deploying services using PLDeploy, it needs to be done in a very special fashion by constructing and attaching what is called *cogs* that are used to deploy services and pulling the results back. There is not much information related to this tool and its documentation has not been updated in the last decade, making a more in-depth comparison hard to perform. pssh on the other hand provides a parallel version of OpenSSH and related tools [8] and its main features of providing parallel execution of commands over ssh is also present in *plcli*. For example, when users want to run a command on several nodes, this is done in a concurrent fashion without the users having to consider it.

1.2 Use Cases

The first use case, **Use Case 1**, targets support for deploying an application to a given number of physical PlanetLab nodes, launching a specified amount of instances on each node and upon termination, gathering of log files. As outlined in Section 2, engineers using *plcli* are able to quickly deploy their experiments through adding a configuration file to their application repository. Use Case 1 is considered the main feature of *plcli* and is the only one implemented in the preliminary version of *plcli*.

Apart from running experiments, being able to carry out distributed debugging is also an attractive feature and is expressed as **Use Case 2**. Finding out what is happening in a distributed system is a very complex task, as clearly outlined by Joyce *et al.* [9] and support for distributed debugging could increase engineering productivity a great deal as well as aiding in finding out why certain problems occur in production systems. For example, an approach similar to D3S presented by Liu *et al.* in [10] or the solution based on the MINHA platform presented by Jorge *et al.* [11] could be taken to implement support for this use case.

The duration of experiments might range from a few minutes to hours or even days and it is important that these experiments are performed on healthy nodes. Features such as healthy node discovery and automatically fixing some of the more simple problems of nodes (for example problems that may be resolved through a simple reboot) could be added to *plcli* to provide a richer toolset when working with PlanetLab nodes, which is referred to as **Use Case 3**. This could be taken even further by considering the PlanetLab platform as a system in itself and deploy planning agents that make decisions based on repair plans and carry out node repairs, which is an approach heavily based on the one presented by Dashofy *et al.* [12].

1.3 Contribution

In this paper a tool for deploying and running distributed applications using the PlanetLab platform known as *plcli* has been introduced, along with three main use cases representing the three main features of the tool. A preliminary implementation of *plcli* is bundled with this paper which provides functionality for Use Case 1. An evaluation of the performance of the tool with respect to Use Case 1 has been carried as well, which is presented and discussed. An implementation satisfying Use Case 1 is offered alongside this paper [13].

2 System Design

plcli is a command-line interface written in the programming language Go, which is a programming language introduced by Google in 2009, designed for fast compilation of source code and easier programming [14] [15] [16]. The Go programming language provides functionality for implementing efficient applications with scalable concurrency mechanisms known as *goroutines*. Goroutines are essentially lightweight threads associated with less overhead and the Go runtime is very efficient in the handling of these goroutines. As shown by Togashi *et al.*, Go outperforms for example Java when it comes to concurrency handling [17]. Mechanisms for efficient concurrency handling are naturally of high interest when developing a tool such as *plcli* that must be able to communicate with tens, or even hundreds, of nodes efficiently. These goroutines are further referred to as *workers* and are used whenever there are I/O-bound operations that need to be executed concurrently, such as calling the PlanetLab API or executing commands on nodes over SSH.

plcli is a preliminary implementation of a full-fledged tool intended to provide support for all three use cases outlined in Section 1.2. Through the implementation of support for Use Case 1, various features have been added to *plcli* such as deployment of applications, file transfer to nodes as well as concurrent command execution. An extensive list of available commands at the time of writing can be found in the README in the project repository [13].

Since the main functionality of the current implementation is to deploy code, a more in-depth explanation of how a deployment is performed is provided. *plcli* makes use of what is called a configuration file which is required to be placed in the root of the public repository of applications that should be deployed using *plcli*. This file contains information about environment variables, how the application is prepared for launch and how to launch an instance. *plcli* downloads this file from the application repository and performs the needed steps in order to prepare the PlanetLab nodes for deployment of the given application. The structure of this file is not finalised at the time of writing and omitted for brevity. However, an example can be found in the GitHub repository for the demo application [18].

plcli aims to reduce the time and effort needed to deploy and run experiments on PlanetLab and consequently, it is highly dependant on the performance of the PlanetLab API. In order to retrieve information about what nodes are available and decide which nodes to use in a deployment, the API is queried for up to date information. However, the API is only used internally by *plcli* in an effort to enable users and researchers to focus on what experiments to run rather than how these experiments actually are run.

3 Evaluation Plan

In order to investigate the performance of *plcli*, the time taken to perform a full deployment - the *deployment time* - is evaluated in three different experiments. All three experiments

presented below utilise a basic demo application written in Python 3.7, which can be found on GitHub [18] and were run on a MacBook Pro 15" with a 2,2 GHz Intel Core i7 processor and 16 GB of RAM.

In the first experiment, one application instance was deployed to a varying amount of physical nodes with one worker allocated per node. This was done to investigate changes in deployment time as the deployment grows with respect to the number of nodes. Furthermore, experiments with one instance deployed to a fixed amount of nodes with a varying amount of workers were also conducted, which aided in investigating the performance gain of using workers. Lastly, due to severe problems with finding more than fifteen suitable nodes for the demo application, a varying amount of application instances were launched on the same set of physical nodes to try and investigate the performance at scale.

4 Evaluation Results

The first experiment measuring the performance of *plcli* when deploying to an increasing amount of physical nodes exhibits a nearly constant deployment time, as can be seen in Figure 1. There is a small increase of the trendline as the number of nodes increase, but it is very small and indicates that the overhead introduced by adding more nodes than fifteen will be minimal. For example, the time taken to deploy to three nodes is around fifteen seconds while deploying to fifteen nodes takes around sixteen seconds; that is a 400% increase in the number of nodes with merely 7% increase in deployment time which indicates promising performance when scaling. These results are expected, since one worker is allocated per instance and consequently a lot of work can be carried out in an efficient fashion.

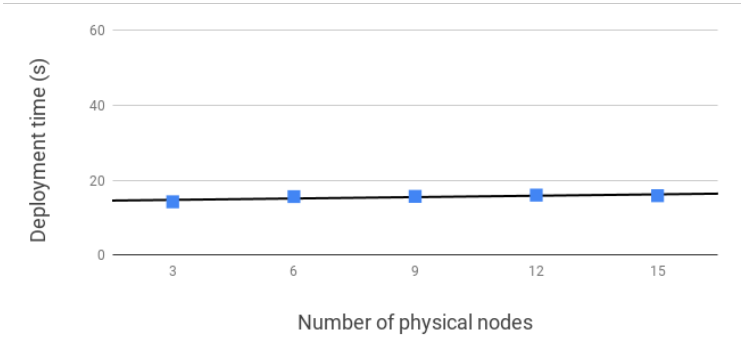


Figure 1: Deploying one instance to an increasing number of physical nodes with one worker per instance.

The results from the second experiment, investigating the effect of workers with respect to the deployment time, can be seen in Figure 2. As can be seen, the deployment time is nearly cut in half as the number of workers are doubled. The reason for the time being a bit unevenly reduced as the number of workers are doubled is most likely due to network latency and other operations that are subject to variation in execution time. The reason for not deploying using more than twelve workers is that since twelve physical nodes are used, more than twelve workers would not make any sense since there would not be any work for the additional workers. These results are expected, since in theory, a 100% increase in workers should yield a 50% decrease in execution time since there are twice as many

workers available to perform the deployments.

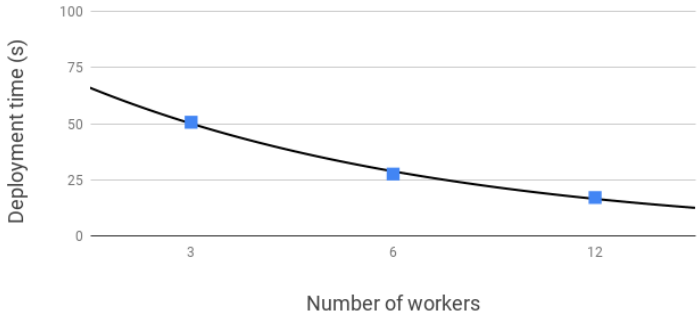


Figure 2: Deploying one instance to twelve nodes with an increasing number of workers.

Results from the third experiment, examining the change in deployment time when increasing the amount of instances on a constant amount of nodes, can be seen in Figure 3. It shows the increase of deployment time as the number of instances is scaled up to as much as 120 instances launched on fifteen physical nodes. Much similar to Figure 1, the deployment time is slowly growing but almost kept constant as the number of instances is multiplied by 8x. Due to one worker being used per instance launch, an almost constant deployment time is to be expected.

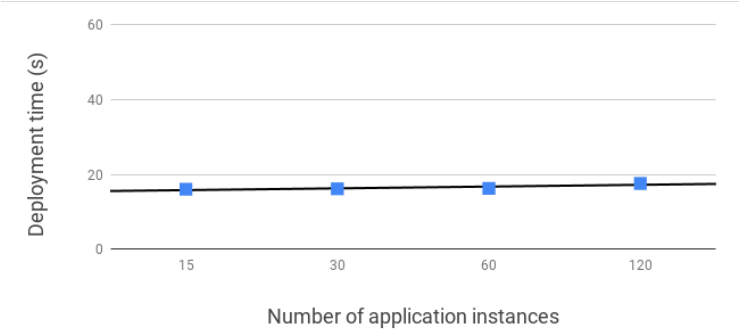


Figure 3: Scaling deployments virtually with one worker per instance to fifteen physical nodes.

5 Conclusion

A first implementation of *plcli* supporting Use Case 1 has been implemented and shown to provide a nearly constant deployment time for up to as much as 120 application instances on 15 physical nodes on the PlanetLab platform. The approach of using workers has been shown to be directly linked to the deployment time and shown to be very beneficial when performing concurrent deployments. Furthermore, *plcli* is bundled with many useful features for engineers using PlanetLab to run applications. When it comes to further work, *plcli* would benefit greatly from implementation of Use Case 2 and Use Case 3, as well as a more rigorous evaluation with respect to scalability (i.e. deploying to more physical nodes). This preliminary implementation is to be considered as a pilot and a foundation for continued work.

References

[1] ns-3 — a discrete-event network simulator for internet systems. URL: https://www.nsnam.org/, Accessed: 2019-07-12.

[2] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator 2 (NS2)*, pages 1–18. Springer US, Boston, MA, 2009.

[3] Planetlabeurope. URL: https://www.planet-lab.eu/, Accessed: 2019-07-12.

[4] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[5] Larry Peterson and Timothy Roscoe. The design principles of planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):11–16, January 2006.

[6] Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir. Experiences building planetlab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 351–366, Berkeley, CA, USA, 2006. USENIX Association.

[7] User tools. URL: https://www.planet-lab.org/tools, Accessed: 2019-09-14.

[8] parallel-ssh. URL: https://code.google.com/archive/p/parallel-ssh/, Accessed: 2019-07-03.

[9] Jeffrey Joyce, Greg Lomow, Konrad Slind, and Brian Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, March 1987.

[10] Xuezheng Liu, Zhenyu Guo, Xi Wang, Feibo Chen, Xiaochen Lian, Jian Tang, Ming Wu, M. Kaashoek, and Zheng Zhang. D3s: Debugging deployed distributed systems. In *Proc. NSDI*, pages 423–437, 01 2008.

[11] Tiago Jorge, Francisco Maia, Miguel Matos, José Pereira, and Rui Oliveira. Practical evaluation of large scale applications. In Alysson Bessani and Sara Bouchenak, editors, *Distributed Applications and Interoperable Systems*, pages 124–137, Cham, 2015. Springer International Publishing.

[12] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. Towards architecture-based self-healing systems. In *Proceedings of the First Workshop on Self-healing Systems*, WOSS '02, pages 21–26, New York, NY, USA, 2002. ACM.

[13] plcli github repository. URL: https://github.com/axelniklasson/plcli, Accessed: 2019-09-14.

[14] The go programming language. URL: https://golang.org/, Accessed: 2019-07-09.

[15] Rob Pike. The go programming language. *Talk given at Google's Tech Talks*, 2009.

[16] Alan AA Donovan and Brian W Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015.

[17] N. Togashi and V. Klyuev. Concurrency in go and java: Performance analysis. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 213–216, April 2014.

[18] plcli demo app github repository. URL: https://github.com/axelniklasson/plcli-demo-app, Accessed: 2019-09-14.

A Existing PlanetLab tools

Tables 1 and 2 list all the tools listed on the PlanetLab website[1] as of July 2, 2019.

Name	**Brief description**	**State**
Plush	Users describe experiments or computation in XML, and Plush uses it to locate, contact, and prepare resources. It includes a Nebula GUI that allows users to build, visualize and run their applications without using the command-line interface.	Can't access webpage.
PIMan	PlanetLab Experiment Manager is designed to simplify the deployment, execution and monitoring of your PlanetLab experiment. The application presents a simple GUI to perform common tasks.	Can access webpage, but all links are broken.
Stork	A software installation utility akin to yum and apt available for both users of PlanetLab and for home use. It includes a Stock Slice Manager GUI that simplifies package management and Stork installation on your PlanetLab slices.	Broken link.
pShell	A Linux shell like interface providing a few basic commands to interact with a Planetlab slice, works as a command center at the local machine and interact with slice nodes.	Broken link.
AppManager	PlanetLab Application Manager is designed to help deploy, monitor, and run applications on PlanetLab. The package gives you the ability to centrally manage, install, upgrade, start, stop, and monitor of applications on a PlanetLab slice.	Broken link.

Table 1: Tools listed on the PlanetLab website and their state as of July 2, 2019 (Table 1/2)

[1] https://www.planet-lab.org/tools

Name	Brief description	State
Emulab	A network testbed, giving researchers a wide range of environments in which to develop, debug, and evaluate their systems.	Accessible, but different purpose than *plcli*.
plDist	A tool for parallel distribution of files to Planetlab nodes using BitTorrent or rsync.	Broken link.
Nixes	A set of bash scripts to install, maintain, control and monitor applications on PlanetLab.	Broken link.
PLDeploy	PlanetLab Slice Deploy Toolkit is a set of scripts to help users manage their slices.	Accessible, comparison with *plcli* can be found in Section 1.1.
pssh	Provides the parallel versions of the openssh tools. It can be used to control large collections of nodes in the wide-area network.	Accessible, comparison with *plcli* can be found in Section 1.1.
vxargs	Inspired by xargs and pssh, it provides the parallel versions of any arbitrary command, including ssh, rsync, scp, wget, curl, etc.	Broken link.
PlanetLab broadband link emulator	A link emulator for PlanetLab that can be configured with few important measured characteristics of broadband links, such as their asymmetric link bandwidths and queue sizes.	Accessible, but different purpose than *plcli*.

Table 2: Tools listed on the PlanetLab website and their state as of July 2, 2019 (Table 2/2)