Применение TLA+ для эффективного тестирования распределенных систем

Никита Синяченко

Евгений Чернацкий

BARSiC

Рассмотрим наш путь к верификации распределенной системы **BARSiC** (**B**inlog **A**synchronous **R**eplication **S**ystem **и** **C**onsensus)

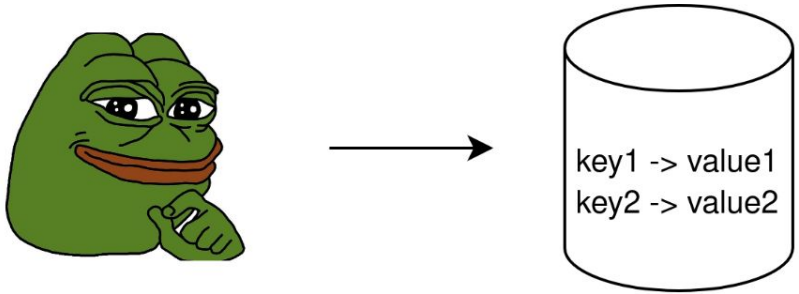
BARSiC

Рассмотрим наш путь к верификации распределенной системы **BARSiC** (**B**inlog **A**synchronous **R**eplication **S**ystem **и** **C**onsensus)

- Мотивация к созданию
- Как реализовывался
- Почему хотелось верифицировать

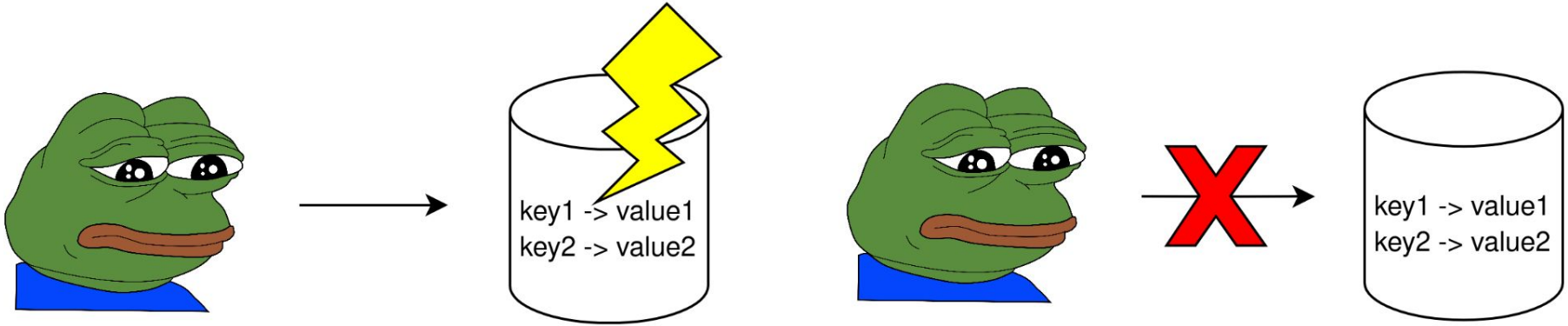
Рассматриваемая задача

- Данные хранятся в некоторой БД
- Хочется максимизировать доступность системы



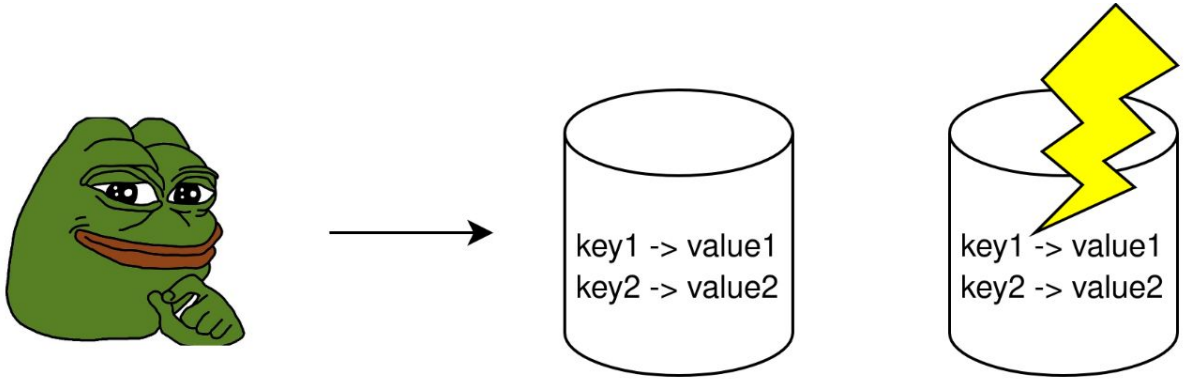
Рассматриваемая задача

- Данные хранятся в некоторой БД
- Хочется максимизировать доступность системы
- Возможные проблемы:
  - Упала машина с БД
  - Сетевые неполадки



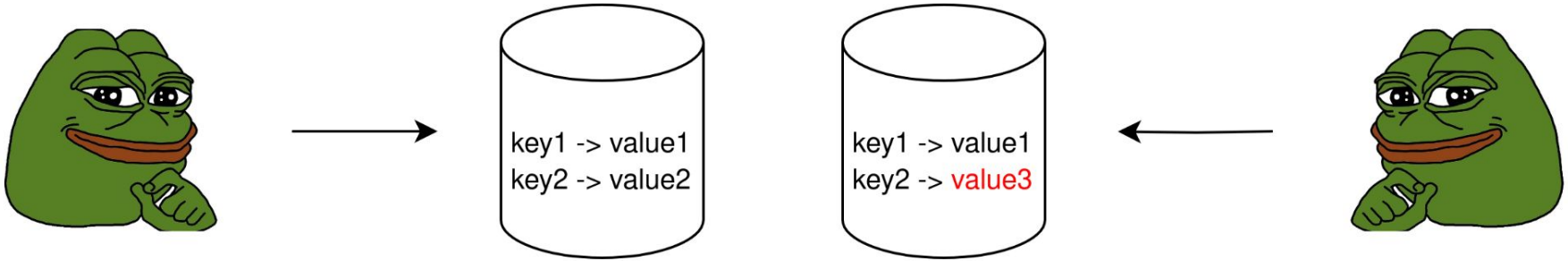
Рассматриваемая задача

- Данные хранятся в некоторой БД
- Хочется максимизировать доступность системы
- Возможные проблемы:
  - Упала машина с БД
  - Сетевые неполадки
- Решение: **репликация данных**



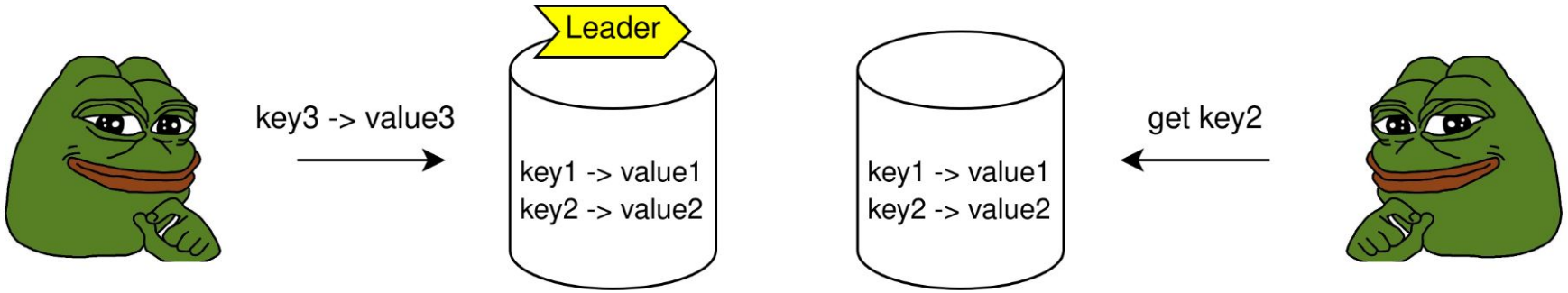
Репликация данных

- Данные в системе могут изменяться
- Требуется поддерживать согласованность между копиями



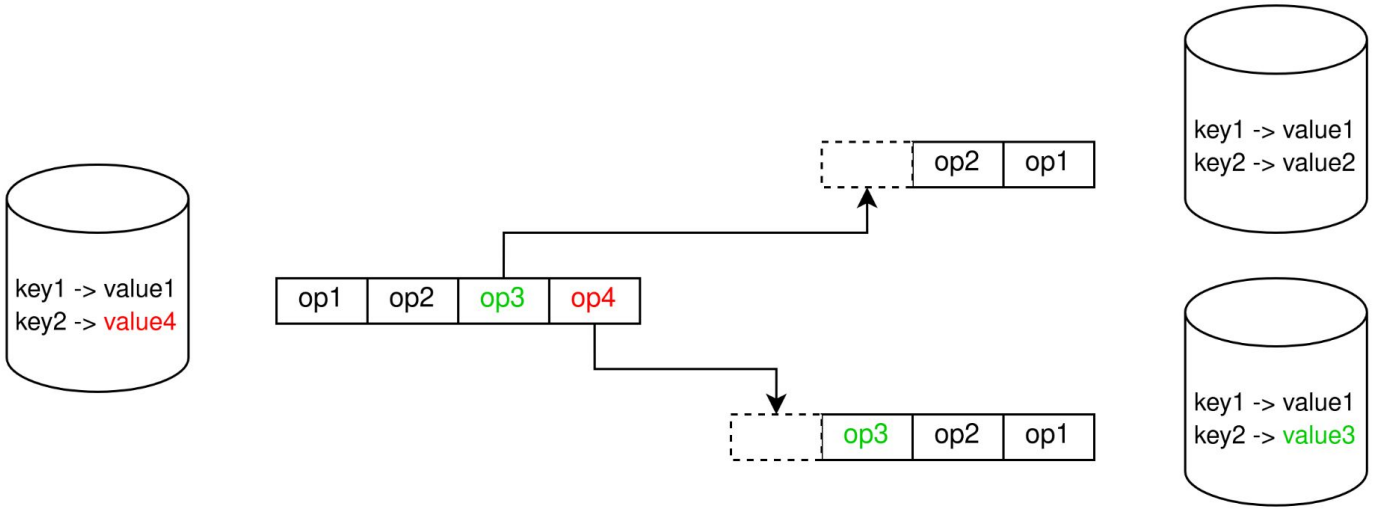
Leader-follower репликация

- Операции **чтения** будем совершать над **любой копией**
- Операции **записи** – только над выделенной **копией-лидером**



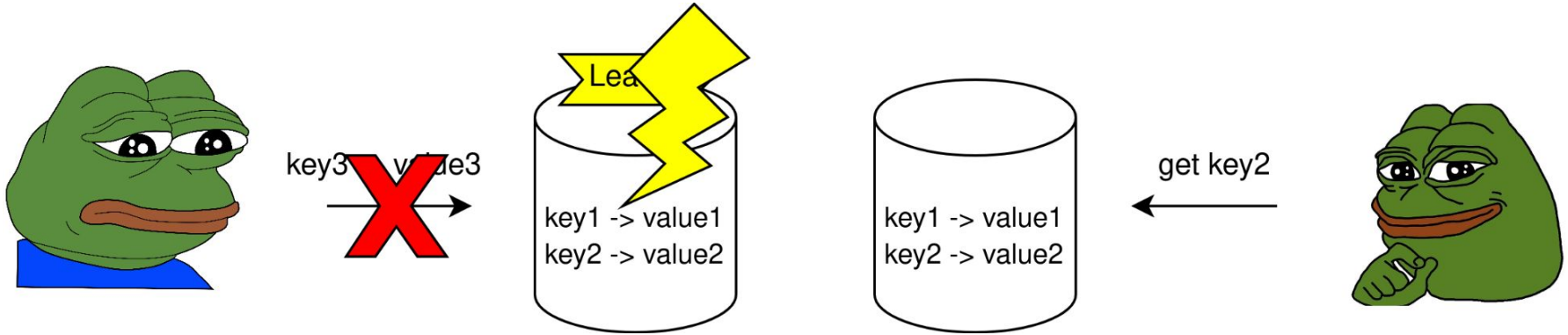
Leader-follower репликация

- Операции **чтения** будем совершать над **любой копией**
- Операции **записи** – только над выделенной **копией-лидером**
- Данные записываем в последовательный журнал операций
- Рассылаем элементы журнала



Leader-follower репликация

- При сбое лидера система перестает работать на запись
- Можно назначать нового лидера вручную админами
- Но хочется быстро, автоматически и без ручной работы



Leader-follower репликация

- Как выбирать нового лидера при сбое прежнего?
- Как согласовывать данные между копиями?
- Как восстанавливать данные после сбоя копии?


Я ВЫШЕЛ В ИНТЕРНЕТ
С ТАКИМ ВОПРОСОМ
memesmix.net

Leader-follower репликация

- Как выбирать нового лидера при сбое прежнего?
- Как согласовывать данные между копиями?
- Как восстанавливать данные после сбоя копии?

Ответ: **алгоритмы консенсуса**

Алгоритмы консенсуса

Позволяют согласованно принять решение на нескольких агентах с учётом возможных сбоев

Примеры:

- **Paxos, MultiPaxos** (Megastore, Spanner, Chubby, …)
- **Raft** (etcd, TIDB, Consul, …)

Team Paxos vs Team Raft

Heidi Howard @ Hydra

Amazon ECS

cassandra

PaxosStore: High-availability Storage Made Practical in WeChat

Jianjun Zheng[1] Qian Lin[1,*] Jiatao Xu[1] Cheng Wei[1]
Chuwei Zeng[1] Pingan Yang[1] Yunfan Zhang[1]
[1]Tencent Inc. [2]National University of Singapore
{rockzheng, sunnyxu, dengoswei, eddyzeng, ypaapyyang, ifanzhang}@tencent.com
[2]linqian@comp.nus.edu.sg

Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency

Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas

Microsoft

Clustrix

ceph

infinit

Megastore: Providing Scalable, Highly Available Storage for Interactive Services

Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, Vadim Yushprakh
Google, Inc.

Large-scale cluster management at Google with Borg

Abhishek Verma[1] Luis Pedrosa[2] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Doozer

LogDevice

The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, *Google Inc.*

neo4j

log Cabin

TiKV

HashiCorp Consul

neo4j

RethinkDB

hazelcast

Atomix

etcd

Cockroach DB

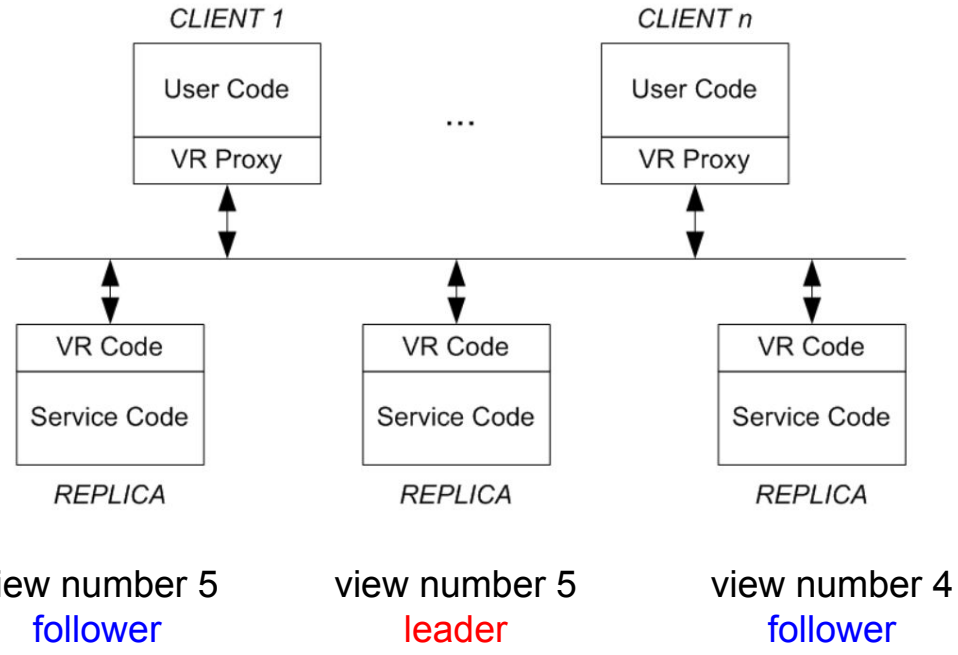
Алгоритмы консенсуса

- **Paxos, MultiPaxos** (Megastore, Spanner, Chubby, …)
- **Raft** (etcd, TIDB, Consul, …)
- **ZAB** (Zookeeper)
- …
- **Viewstamped Replication** (*BARSiC*)

Viewstamped Replication

- Основан на раундах (view)
- В каждом раунде один **предопределенный лидер**
- В начале каждого раунда реплики решают, будет ли он действующим
- **Действующий лидер** принимает запросы и реплицирует их остальным
- При сбое лидера реплики начинают новый раунд

[Viewstamped Replication Revisited]



Написали алгоритм по статье.
Отладили, протестировали.
Всё взлетело.


Ага, поверил, ну и бредятина

Имплементить статьи умеют все

Viewstamped Replication не практичен

2. When replica i receives STARTVIEWCHANGE messages for its *view-number* from f other replicas, it sends a ⟨DOVIEWCHANGE v, l, v', n, k, i⟩ message to the node that will be the primary in the new view. Here v is its *view-number*, l is its log, v' is the view number of the latest view in which its status was *normal*, n is the *op-number*, and k is the *commit-number*.

В сообщениях передается весь журнал операций

Всё состояние хранится в памяти

- The *log*. This is an array containing *op-number* entries. The entries contain the requests that have been received so far in their assigned order.

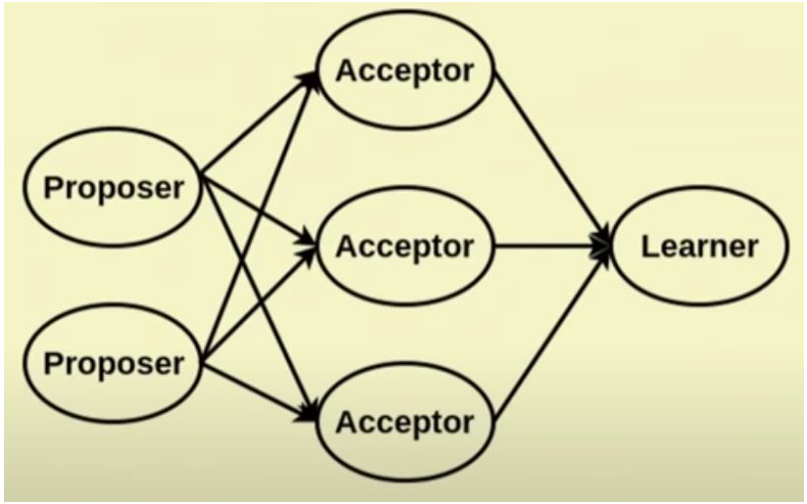
Улучшения алгоритма

- Пересылаем в сообщениях не весь лог, а по частям
- Храним состояние персистентно, чтобы не скачивать его полностью при рестарте узла
- Начинаем новые выборы не по таймеру, а при наборе кворума желающих (prevote)
- …

Написали Барсика по статье, получили совсем другой алгоритм,
корректность не доказана.
Проблема в нас?

Другие уже сталкивались с таким!

Вспомним хрестоматийный пример – Paxos



Paxos легко описывается

A Paxos Algorithm

This summarises our simplified, Raft-style Paxos algorithm. The text in red is unique to Paxos.

State

Persistent state on all servers: (Updated on stable storage before responding to RPCs)
currentTerm latest term server has seen (initialized to 0 on first boot, increases monotonically)
log[ ] log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)
Volatile state on all servers:
commitIndex index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied index of highest log entry applied to state machine (initialized to 0, increases monotonically)
Volatile state on candidates: (Reinitialized after election)
entries[] Log entries received with votes
Volatile state on leaders: (Reinitialized after election)
nextIndex[ ] for each server, index of the next log entry to send to that server (initialized to leader commit index + 1)
matchIndex[ ] for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries RPC

Invoked by leader to replicate log entries; also used as heartbeat
Arguments:
term leader's term
prevLogIndex index of log entry immediately preceding new ones
prevLogTerm term of prevLogIndex entry
entries[ ] log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit leader's commitIndex
Results:
term currentTerm, for leader to update itself
success true if follower contained entry matching prevLogIndex and prevLogTerm
Receiver implementation:
1. Reply false if term < currentTerm
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex: set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes
Arguments:
term candidate's term
leaderCommit candidate's commit index
Results:
term currentTerm, for candidate to update itself
voteGranted true indicates candidate received vote
entries[] follower's log entries after leaderCommit
Receiver implementation:
1. Reply false if term < currentTerm
2. Grant vote and send any log entries after leaderCommit

Rules for Servers

All Servers:
- If commitIndex > lastApplied: increment lastApplied and apply log[lastApplied] to state machine
- If RPC request or response contains term T > currentTerm: set currentTerm = T and convert to follower

Followers:
- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates:
- On conversion to candidate, start election: increase currentTerm to next t such that $t \bmod n = s$, copy any log entries after commitIndex to entries[], and send RequestVote RPCs to all other servers
- Add any log entries received from RequestVote responses to entries[]
- If votes received from majority of servers: update log by adding entries[] with currentTerm (using value with greatest term if there are multiple entries with same index) and become leader

Leaders:
- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts
- If command received from client: append entry to local log, respond after entry applied to state machine
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry
- If there exists an N such that N > commitIndex and a majority of matchIndex[i] ≥ N: set commitIndex = N

Корректность Paxos доказана

P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

Since numbers are totally ordered, condition P2 guarantees the crucial safety property that only a single value is chosen.

To be chosen, a proposal must be accepted by at least one acceptor. So, we can satisfy P2 by satisfying:

P2^a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v.

We still maintain P1 to ensure that some proposal is chosen. Because communication is asynchronous, a proposal could be chosen with some particular acceptor c never having received any proposal. Suppose a new proposer "wakes up" and issues a higher-numbered proposal with a different value. P1 requires c to accept this proposal, violating P2^a. Maintaining both P1 and P2^a requires strengthening P2^a to:

P2^b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

Но проблемы такие же

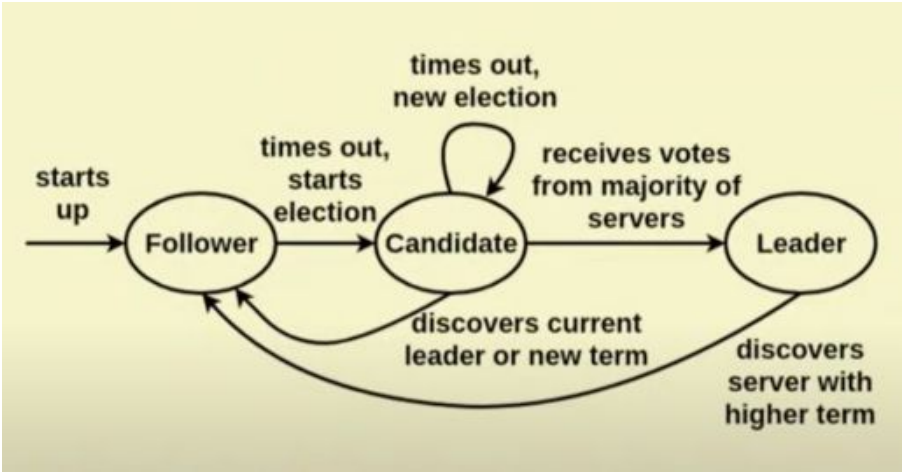
Paxos Made Live - An Engineering Perspective

9 Summary and open problems

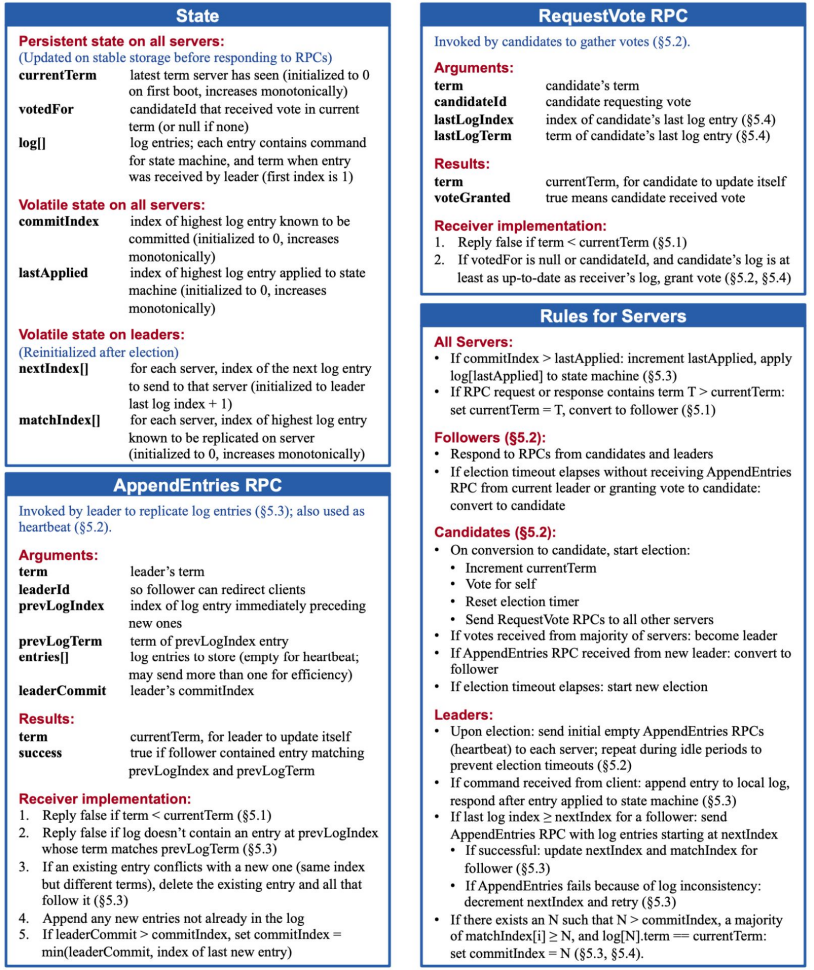
We have described our implementation of a fault-tolerant database, based on the Paxos consensus algorithm. Despite the large body of literature in the field, algorithms dating back more then 15 years, and experience of our team (one of us has designed a similar system before and the others have built other types of complex systems in the past), it was significantly harder to build this system then originally anticipated. We attribute this to several shortcomings in the field:

- There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system. In order to build a real-world system, an expert needs to use numerous ideas scattered in the literature and make several relatively small protocol extensions. The cumulative effort will be substantial and the final system will be based on an unproven protocol.
- The fault-tolerance computing community has not developed the tools to make it easy to implement their algorithms.
- The fault-tolerance computing community has not paid enough attention to testing, a key ingredient for building fault-tolerant systems.

Ещё один алгоритм – Raft



Raft легко описывается



State

Persistent state on all servers:
(Updated on stable storage before responding to RPCs)

currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)

Volatile state on all servers:

commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied	index of highest log entry applied to state machine (initialized to 0, increases monotonically)

Volatile state on leaders:
(Reinitialized after election)

nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

Rules for Servers

All Servers:

- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):

- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Leaders:

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).

Figure 2: A condensed summary of the Raft consensus algorithm (excluding membership changes and log compaction). The server behavior in the upper-left box is described as a set of rules that trigger independently and repeatedly. Section numbers such as §5.2 indicate where particular features are discussed. A formal specification [31] describes the algorithm more precisely.

Корректность Raft доказана

9.2 Correctness

We have developed a formal specification and a proof of safety for the consensus mechanism described in Section 5. The formal specification [31] makes the information summarized in Figure 2 completely precise using the TLA+ specification language [17]. It is about 400 lines long and serves as the subject of the proof. It is also useful on its own for anyone implementing Raft. We have mechanically proven the Log Completeness Property using the TLA proof system [7]. However, this proof relies on invariants that have not been mechanically checked (for example, we have not proven the type safety of the specification). Furthermore, we have written an informal proof [31] of the State Machine Safety property which is complete (it relies on the specification alone) and rela-

Снова такие же проблемы

Доклад на Hydra 2022 - Solving Raft's practical problems in Tarantool

“... there is quite a big difference between the Raft implementation and the production-ready Raft implementation. The difference is that the system used on the production-ready requires more from the system than can be provided by the canonical Raft.”

Проблема фундаментальная

Реализация протокола на практике является другим недоказанным протоколом

Как убедиться в корректности реализации протокола?

Такой доклад уже был…

От алгоритма до прода: как подойти к верификации распределенных систем

Важные особенности распределенных систем

Состояние не определено

Работают 3 машины, обмениваются сообщениями. Их нельзя остановить строго одновременно и проверить корректность состояний.

Нельзя сделать Stop The World.

Исполнение не детерминировано

События на каждом агенте могут происходить в любом порядке:

- Получение сообщения
- Срабатывание таймера
- Запись на диск

Чтобы убедиться в корректности необходимо проверить все возможные сценарии исполнения

Нужно писать систему в определенном виде

- В виде, похожем на протокол
- В виде, удобном для тестирования

Event-based модель (акторная модель)

- Все агенты системы представляются функциями вида
  `(State, Event) -> (State, []Event)`

- Текущее состояние актора + событие = новое состояние актора + пачка событий

- События – это получение сообщения, срабатывание таймера, fsync записи на диск, …

Event-based модель (акторная модель)

- Все агенты системы представляются функциями вида
  `(State, Event) -> (State, []Event)`

- `func OnRequestMsg(msg RequestMsg) []PrepareMsg`

- `func OnCommitTimeout() []StartViewChangeMsg`

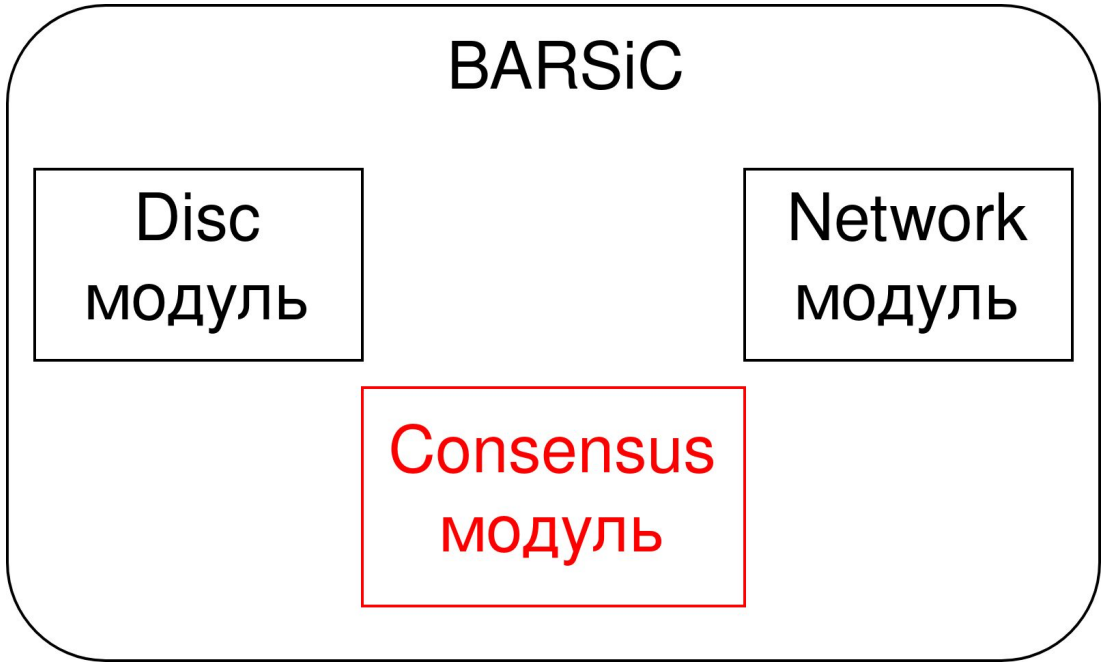
- `func OnFsync() PrepareOkMsg`

BARSiC

Consensus модуль реализует протокол консенсуса

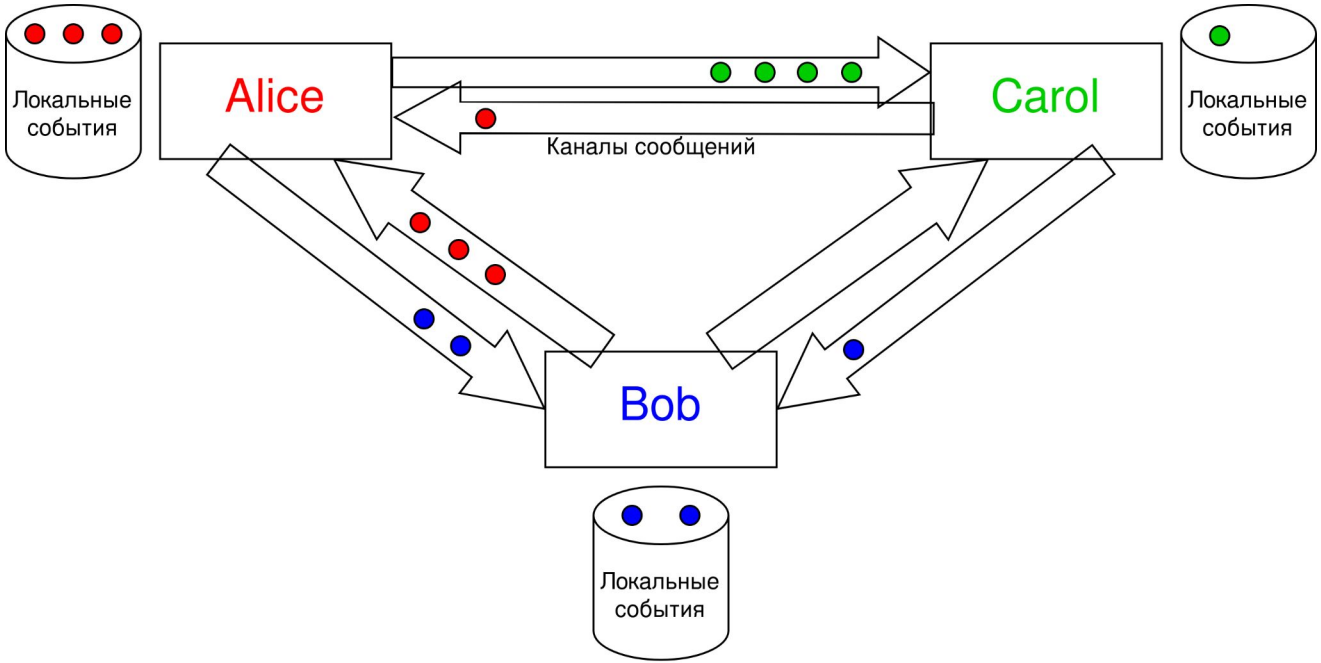
Написан в акторном стиле

Его и требуется верифицировать



Акторы легко тестировать

- Поднимаем локально несколько акторов
- Храним очередь событий до каждого
- Выбираем, в каком порядке события случаются с акторами
- После каждого события проверяем инварианты системы



Fuzzing testing

Такую модель удобно тестировать с помощью **фаззинга**.

Fuzzing testing

Фаззинг – это тестирование кода на случайных входных данных

Функциям на вход даются случайные аргументы и анализируются их реакции на это

Таким образом генерируем и тестируем случайные исполнения кода

Fuzzing testing

Такую модель удобно тестировать с помощью **фаззинга**:
Берём случайные данные, интерпретируем их как id агента + событие из очереди всех событий

Таким образом генерируем случайные поведения распределенной системы

Легко имитировать сбои

- Сбросить состояние актора в начальное
- При получении сообщения проигнорировать его с некоторой вероятностью

Имплементить статьи умеют все

Фаззить код умеют многие

Fuzzing не доказывает корректность

- Fuzzing полезен для отладки и нахождения нетривиальных ошибок и корнер кейсов
- Перебираются не все возможные исполнения, а только случайные
- Нет гарантии корректности системы на всех исполнениях

Формальное описание системы

- Язык спецификаций **TLA+**
- Позволяет описывать программы и системы на формальном языке
- Основан на темпоральной логике
- Инструмент **TLC** проверяет корректность модели с конечным числом состояний

Пример спецификации TLA+

Специфицируем **принцип Дирихле:**

“Если голуби рассажены в клетки, причем число голубей больше числа клеток, то хотя бы в одной из клеток находится более одного голубя”

Пример спецификации TLA+

Специфицируем **принцип Дирихле:**

```
---- MODULE DirichletPrinciple ----
EXTENDS Integers

CONSTANTS MaxPigeonCount, BoxCount
ASSUME MaxPigeonCount > BoxCount

VARIABLES pigeonCount, boxes

Init ==
    /\ pigeonCount = 0
    /\ boxes = [i \in 1..BoxCount |-> 0]

AddPigeon(i) ==
    /\ pigeonCount < MaxPigeonCount
    /\ pigeonCount' = pigeonCount + 1
    /\ boxes' = [boxes EXCEPT ![i] = boxes[i] + 1]

Next ==
    \E i \in 1..BoxCount:
        AddPigeon(i)

Spec ==
    Init /\ [][Next]_<<pigeonCount, boxes>>

DirichletPrinciple ==
    (pigeonCount = MaxPigeonCount) =>
        \E i \in 1..BoxCount: boxes[i] > 1

====
```

Пример спецификации TLA+

Константы – параметры системы:

- `MaxPigeonCount` – максимальное суммарное число голубей
- `BoxCount` – число клеток

Переменные – состояние системы:

- `pigeonCount` – суммарное число голубей на данный момент
- `boxes` – число голубей в каждой из клеток

```
CONSTANTS MaxPigeonCount, BoxCount
ASSUME MaxPigeonCount > BoxCount

VARIABLES pigeonCount, boxes
```

Пример спецификации TLA+

Начальное состояние:

- `pigeonCount = 0`
- `boxes` – массив длины `BoxCount`, состоящий из нулей

```
Init ==
    /\ pigeonCount = 0
    /\ boxes = [i \in 1..BoxCount |-> 0]
```

Пример спецификации TLA+

Переходы – возможные события в системе:

- `AddPigeon(i)` – добавить одного голубя в клетку под номером `i`

```
AddPigeon(i) ==
    /\ pigeonCount < MaxPigeonCount
    /\ pigeonCount' = pigeonCount + 1
    /\ boxes' = [boxes EXCEPT ![i] = boxes[i] + 1]

Next ==
    \E i \in 1..BoxCount:
        AddPigeon(i)
```

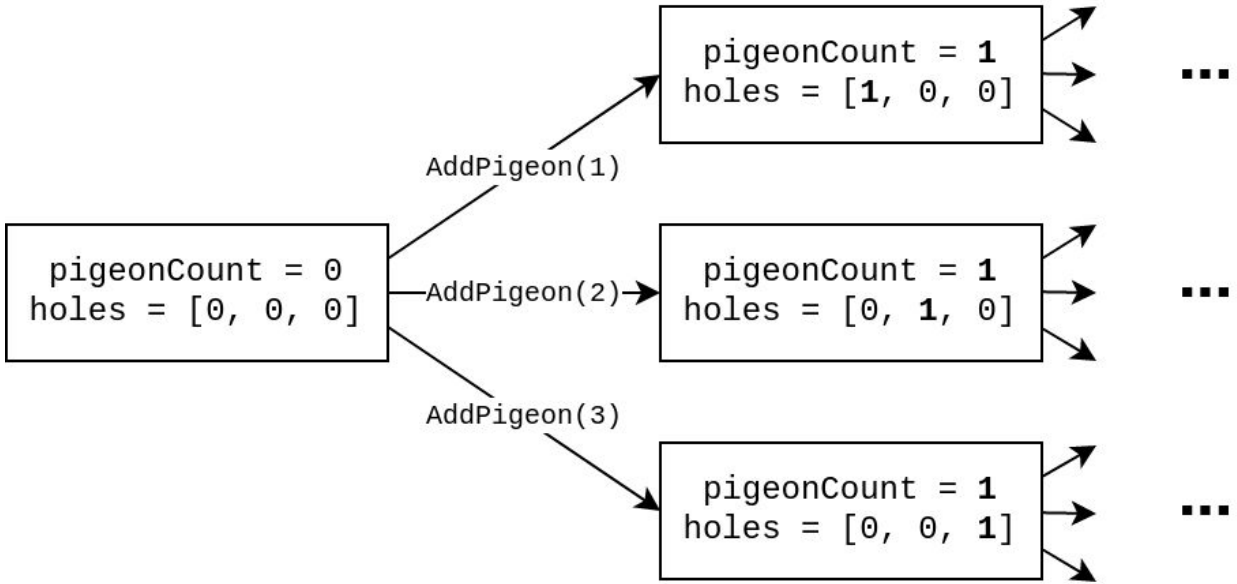
Пример спецификации TLA+

Спецификация – совокупность начального состояния `Init` и возможных переходов `Next`

```
Spec ==
    Init ∧ [][Next]_<<pigeonCount, boxes>>
```

Пример спецификации TLA+

Спецификация также задается своим **графом состояний** (не обязательно конечным)



Пример спецификации TLA+

Инварианты – ожидаемые утверждения про систему:

- `DirichletPrinciple` – после рассадки всех голубей в клетки существует клетка, в которой находится больше одного голубя

```
DirichletPrinciple ==
    (pigeonCount = MaxPigeonCount) ⇒
        \E i \in 1..BoxCount: boxes[i] > 1
```


```
CONSTANT MaxPigeonCount = 4
CONSTANT BoxCount = 3
SPECIFICATION Spec
INVARIANT DirichletPrinciple
```

General

Status: Succeeded (Fingerprint collision probability: 4.900000E-17)
Start: 2023-09-14 13:09:25
End: 2023-09-14 13:09:25

States

Time	Diameter	Found	Distinct	Queue
00:00:00	5	61	35	0

Coverage

Module	Action	Total	Distinct
DirichletPrinciple	Init	1	1
DirichletPrinciple	AddPigeon	60	34

Проверяемые инварианты для BARSiC

- `PrefixLogConsistency` – операции, которые были когда-то закоммичены, должны присутствовать на кворуме узлов

```
PrefixLogConsistency ==
    \E Q \in Quorum:
        \A r \in Q:
            IsPrefixOf(committedLog, Log(r))
```

Проверяемые инварианты для BARSiC

- `CommitedLogMonotonic` – если некоторая операция была когда-либо закоммичена, то она будет продолжать быть закоммиченной (нельзя раскоммитить уже закоммиченную операцию)

```
CommittedLogMonotonic ==
    [][Next ⇒ IsPrefixOf(committedLog, committedLog')]_vars
```

Инструмент проверки TLC

- Строит **граф состояний** по спецификации TLA+
- Проверяет выполнение свойств и инвариантов на построенном графе состояний
- Так как граф строится явно, то проверке подлежат только **конечные** модели

Особенности модели BARSiC

- Размер журнала операций и число выборов не ограничены сверху, а значит число состояний бесконечно
- Для получения верифицированной модели требуется ее ограничить
- В TLA+ модели Барсика размер журнала и число выборов ограничиваются сверху константами
- Как показывает практика и интуиция, ограниченных моделей достаточно, чтобы находить различные баги

Имплементить статьи умеют все

Фаззить код умеют многие

Писать на TLA+ умеют немногие

Корректности TLA+ модели недостаточно

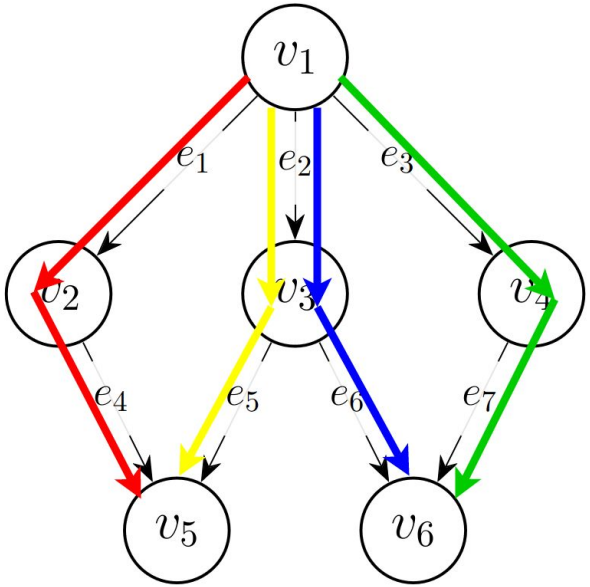
- Формально описали систему на TLA+
- С помощью TLC model checker убедились в корректности специфицированной системы

- **Проблема:** только по корректности модели нельзя утверждать о корректности конкретной реализации системы

Идея решения

- В TLC модель задается конечным графом состояний
- Путь в графе состояний, начинающийся в стартовой вершине, соответствует некоторому исполнению системы

- **Идея:** найдем такое множество путей, чтобы каждое ребро в графе состояний покрывалось как минимум одним путем, а затем экспортируем и применим для тестирования реальной системы



Похожие решения

- **Modelator** – GitHub

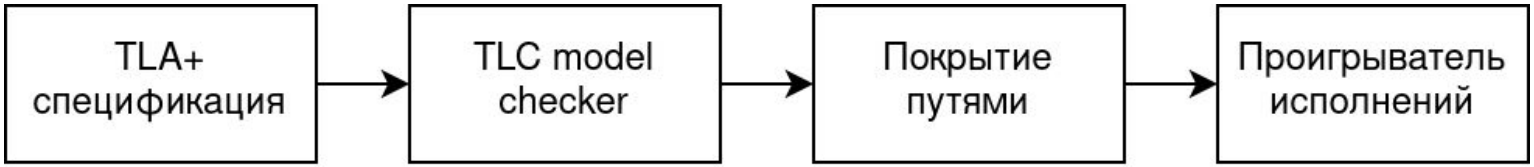
- Инструмент автоматической генерации тестов по модели TLA+
- Не строит граф состояний, а лишь генерирует случайные исполнения в рамках модели

Похожие решения

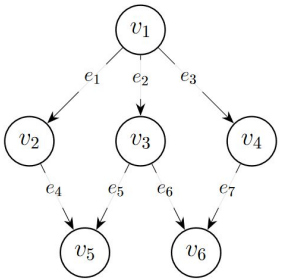
- **Kayfabe** – Model-Based Program Testing with TLA+/TLC

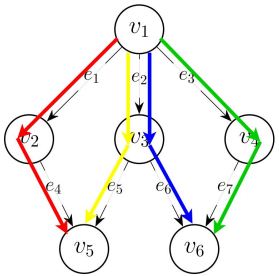
- Не поддерживает действия, имеющие ненулевое число аргументов
- Использует сведение к задаче о коммивояжере, хотя существует более оптимальное решение
-
- Отсутствует в открытом доступе, существует только в виде статьи

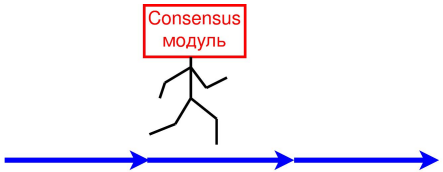
Архитектура решения




```
MODULE Intro
EXTENDS Naturals, Sequences
CONSTANT Keys, N   Keys and N are both defined in the config file
VARIABLES list
ASSUME (N ∈ Nat) ∧ (N > 0)
TypeInvariant ≜ list ∈ Seq(Keys)

Init ≜ list = ⟨⟩   List initialized empty

Insert(key) ≜ ∧ Len(list) < N
              ∧ list' = Append(list, key)

Delete ≜ ∧ Len(list) > 0
         ∧ list' = Tail(list)

Next ≜ ∨ ∃ k ∈ Keys : Insert(k)
       ∨ Delete

Spec ≜ Init ∧ □[Next]_list

THEOREM Spec ⟹ □TypeInvariant
```

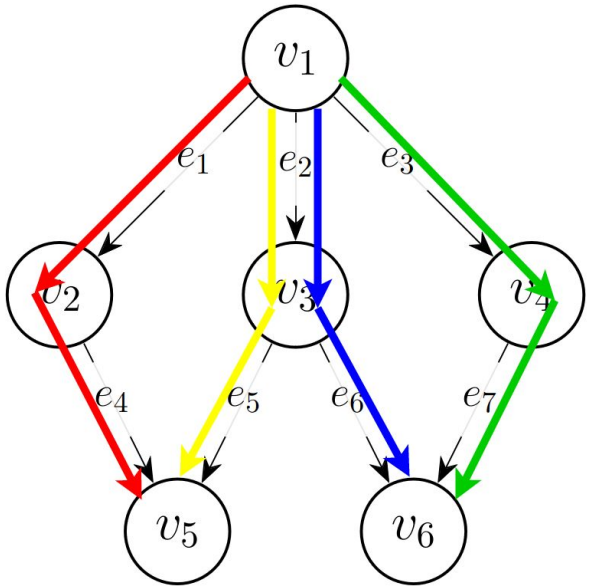






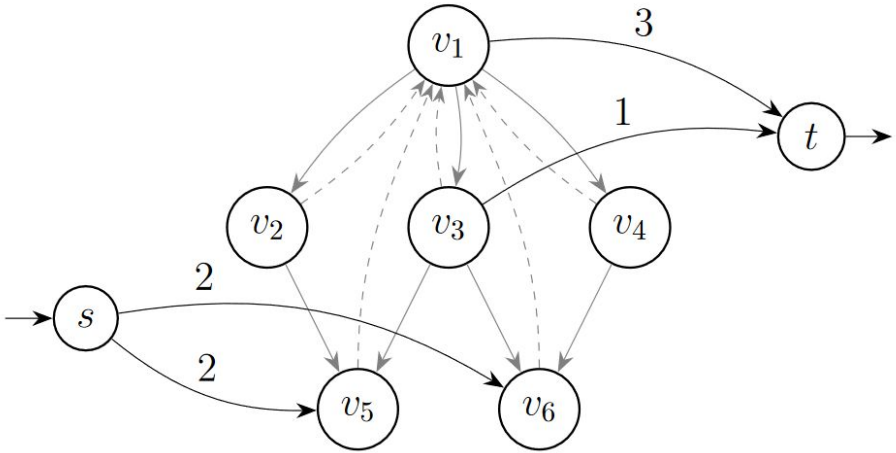
Покрытие путями

- Дан ориентированный граф
- Требуется найти оптимальное по размеру множество путей, начинающихся в стартовой вершине, такое, что каждое ребро покрывается как минимум одним из путей



Покрытие путями

- Сводится к задаче о максимальном потоке и поиску эйлерова цикла в графе
- Асимптотическая сложность $\mathcal{O}(D \cdot |E| + D^2 \cdot |V|)$
- Полученные пути экспортируются на диск в формате JSON



Проигрыватель исполнений

- JSON файл содержит массив исполнений (путей)
- Каждое исполнение состоит из сериализованных состояний (вершин) и переходов (ребер)
- Таким образом, каждое исполнение представляет собой очередной тест
- В ходе тестирования требуется посетить все переходы и убедиться в согласованности системы и модели в каждом состоянии

Проигрыватель исполнений

- Все тесты имеют общий контекст, содержащий тестируемую реализацию системы, а также несколько методов:
  - Метод инициализации системы
  - Метод сброса системы в начальное состояние
  - Метод отображения состояния системы в модельное состояние
  - Метод вызова соответствующего действия по id с заданными аргументами

```
type ModelMapping[S any] interface {
    Init()
    Reset()
    State() S
    PerformAction(int, []any)
}
```

Алгоритм тестирования

1. Считывание JSON файла с покрытием графа путями и вызов метода `Init()`
2. Вызов метода `State()`, сравнение результата с ожидаемым и переход к шагу 4 если это конец пути
3. Вызов метода `PerformAction(...)` для исполнения очередного действия и переход к шагу 2 для обработки следующего состояния на пути
4. Вызов метода `Reset()` для сброса системы в начальное состояние и переход к шагу 2 для обработки следующего пути (если он существует)

Процесс тестирования

Имплементить статьи умеют все
Фаззить код умеют многие
Писать на TLA+ умеют немногие
Тестировать на основе TLA+ умеют единицы

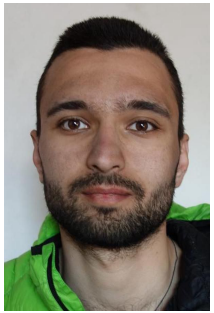
Результаты

- Самая большая проверенная модель, на основе которой был протестирован BARSiC, содержала более 10^8 вершин
- Удалось выявить несколько багов, которые не ловились привычными методами тестирования

Респект пацанам

- Григорий Бутейко
- Николай Климов
- Григорий Петросян
- Илья Кокорин
- Виталий Аксёнов
- Никита Галушко






спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание
спасибо за внимание