Fault-tolerant systems in microservices world

Tobiasz Heller
Ingrid

Agenda

- What can go wrong in distributed system?
- Preventing faults on infrastructure level
- Preventing faults on application level
- Idempotent operations

Fault tolerance is
ability of a system to continue working in the event of partial failure.

Wikipedia

Why do we need fault tolerance?

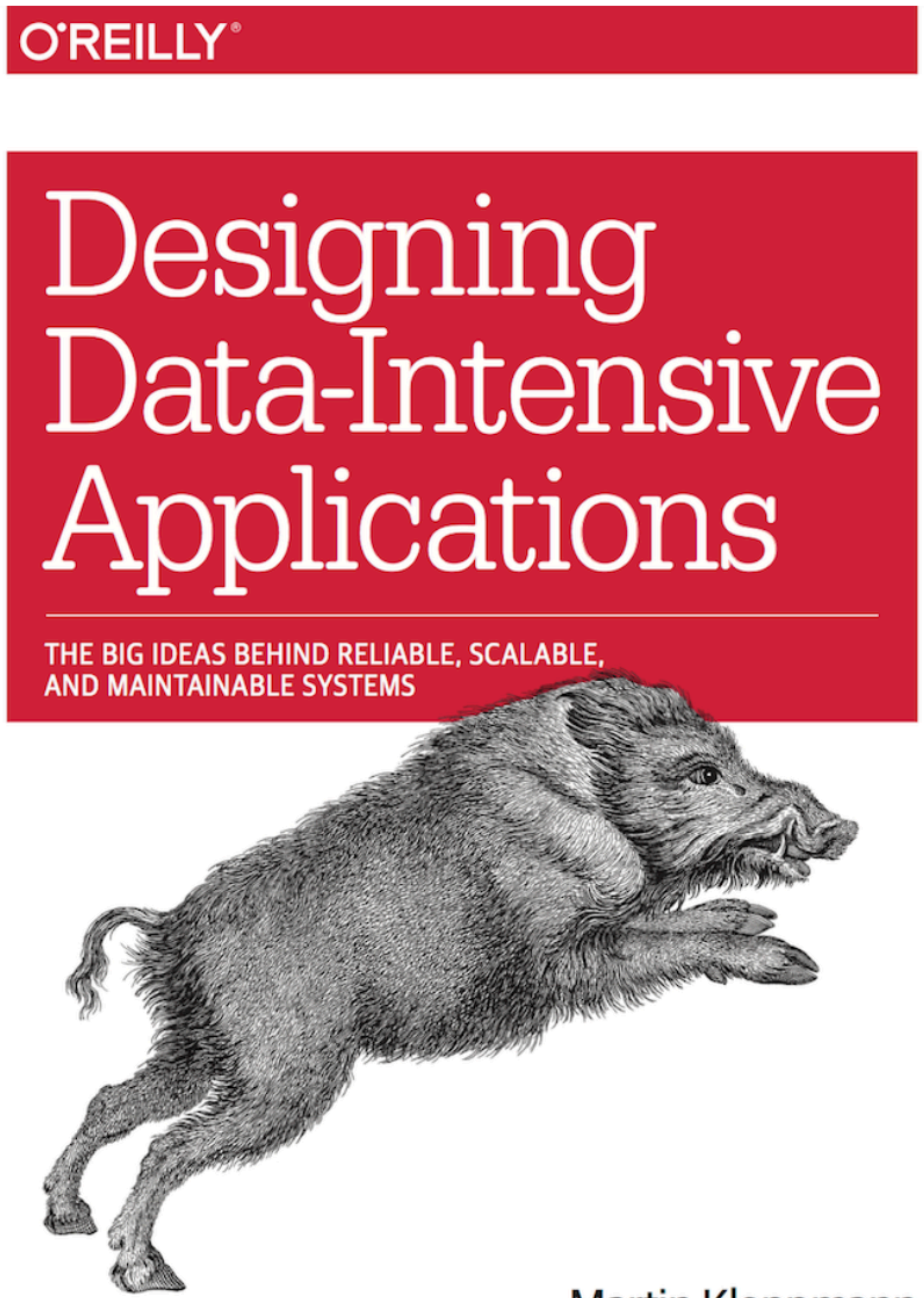
- Reliability

- Availability

- Security

Fault sources

- Program unrecoverable errors
- Node unavailable
- Traffic spikes
- Data or state inconsistency



O'REILLY®

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

Martin Kleppmann

Unrecoverable errors

Unrecoverable errors

Panic

Unrecoverable errors

Panic

```go
package main

func main() {
    var m map[string]float64
    m["pi"] = 3.1416
}
```

Unrecoverable errors

Panic

```go
package main

func main() {
    var m map[string]float64
    m["pi"] = 3.1416
}
```

Unrecoverable errors

Panic

- Defer & recover
- Panic from a goroutine, recover in the same goroutine
    - http middleware
    - gRPC interceptors

Unrecoverable errors

OOM

Unrecoverable errors

OOM

I NEED TO KNOW WHY MOVING OUR APP TO THE CLOUD DIDN'T AUTOMATICALLY SOLVE ALL OUR PROBLEMS.
Dilbert.com @ScottAdamsSays
YOU WOULDN'T LET ME RE-ARCHITECT THE APP TO BE CLOUD-NATIVE.
JUST PUT IT IN CONTAINERS.
11-08-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel
YOU CAN'T SOLVE A PROBLEM JUST BY SAYING TECHY THINGS.
KUBERNETES.

Unrecoverable errors

OOM

- Pod resources management
  - requested
  - limits
- Multiple replicas of service

Unrecoverable errors

OOM

- Pod resources management
  - requested
  - limits
- Multiple replicas of service
- Monitoring & alerting
  - Prometheus, Grafana, Slack

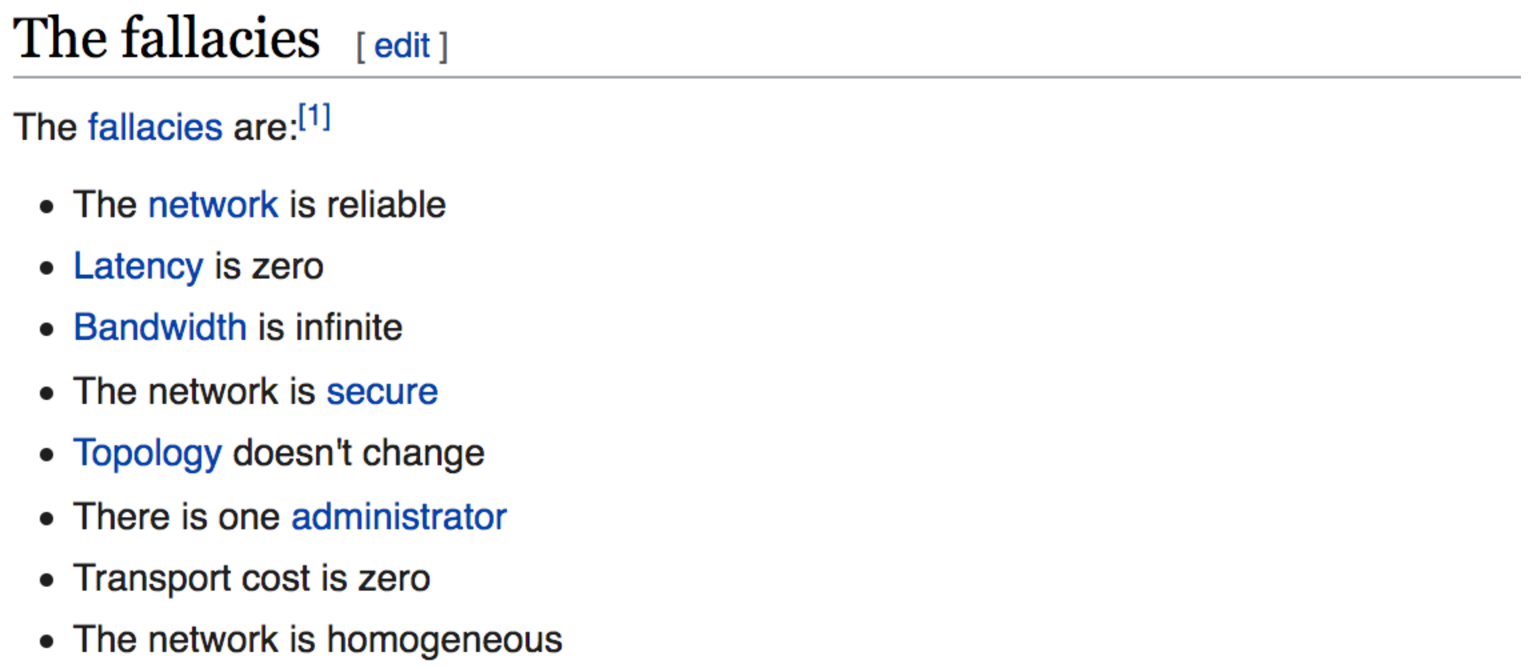
Node unavailable

Node unavailable

Network faults

Network faults

https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing



The fallacies [edit]

The fallacies are:[1]

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Network faults

Timeouts

Network faults

Timeouts

```go
func main() {
    ctx := context.Background()
    ctx, cancel := context.WithTimeout(ctx, 10*time.Second)
    defer cancel()
    req, err := http.NewRequest(http.MethodGet, "http://google.com", nil)
    if err ≠ nil {
        log.Fatal(err)
    }
    req = req.WithContext(ctx)
}
```

Network faults

Retries

Network faults

Retries

```
// grpc_retry "github.com/grpc-ecosystem/go-grpc-middleware/retry"
func Example_initializationWithExponentialBackoff() {
    opts := []grpc_retry.CallOption{
        grpc_retry.WithBackoff(grpc_retry.BackoffExponential(
            100 * time.Millisecond)),
    }
    grpc.Dial("myservice.example.com",
        grpc.WithUnaryInterceptor(
            grpc_retry.UnaryClientInterceptor(opts...)),
    )
}
```

Network faults

Circuit breaker

Protects system and remotely called function by not making calls at all, when certain failure threshold is reached.

https://martinfowler.com/bliki/CircuitBreaker.html

Network faults

Circuit breaker

Examples in Go

github.com/rubyist/circuitbreaker
github.com/sony/gobreaker
github.com/go-kit/kit

Network faults

Service mesh

- Istio
- Linkerd

envoy

Node unavailable

Multiple service replicas

- Specify replicas of pod
- Pod anti-affinity

Event notification

- Service publishes events on action
- Low coupling
- High availability using pub/sub

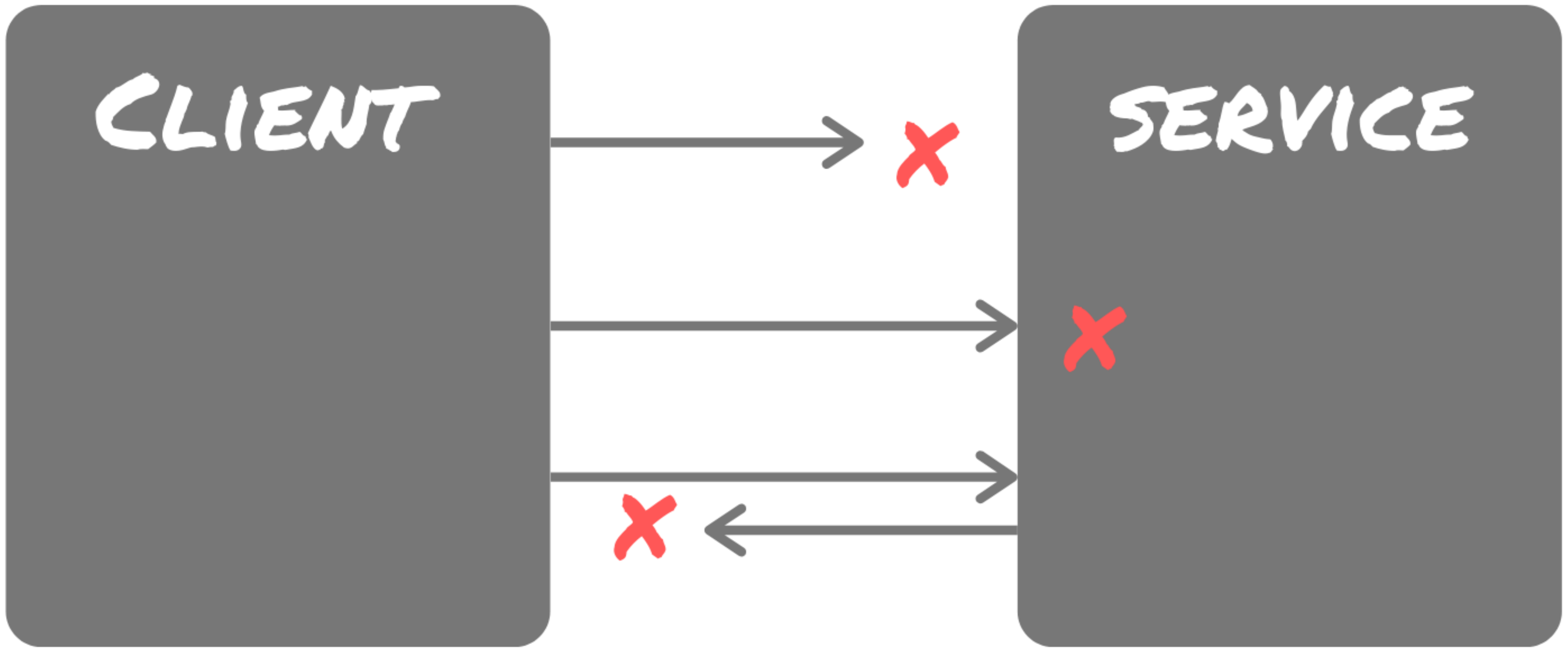
Idempotent operations

Idempotent operations

No matter how many times you call the operation, the result will be the same.

Stackoverflow

Idempotent operations



Idempotent operations

Book shipment operation:

- shipment_id
- merchant_id
- transaction_id*

Enforcing retries and idempotent design

- Chaos monkey (kubemonkey)
- github.com/luno/fate
- Preemptible nodes (last max 24h)

Traffic spikes

Traffic spikes

- Horizontal pod autoscaling
- Load balancing
- Node autoscaling

State inconsistency

State inconsistency

Single SQL database

- Use transactions
- ACID

State inconsistency

Single SQL database

- (A)tomicity - all or none will occur
- (I)solation - how concurrent execution of transactions are treated (different isolation level)

Distributed databases

Eventual consistency

If no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

- Vogels, W. (2009). "Eventually consistent”.

Distributed database

BASE

- (B)asically (A)vailable - basic reading and writing operations are available as much as possible
- (S)oft state: after some time we only have some probability of knowing the state
- (E)ventually consistent

Thank you