KubeCon + CloudNativeCon NA 2026

Fencing Tokens & the Three-Party Contract

What Most Distributed Locks Get Wrong



Benjamin Wang (GitHub: @ahrtr)

• SIG-etcd Co-Tech Lead • WG etcd-operator Co-Organizer • Principal Software Engineer @ VMware by Broadcom

Salt Lake City, Utah, United States

Presentation Outline

Agenda

01 The Pitfalls of Distributed Locks

02 Fencing Tokens & Three-Party Contract

03 Introduce disco & Deep Dive into Lock Implementation

04 Live Demo: example/lock

05 Q & A

Distributed Systems Challenges

The Pitfalls of Distributed Locks

01. Process Pauses

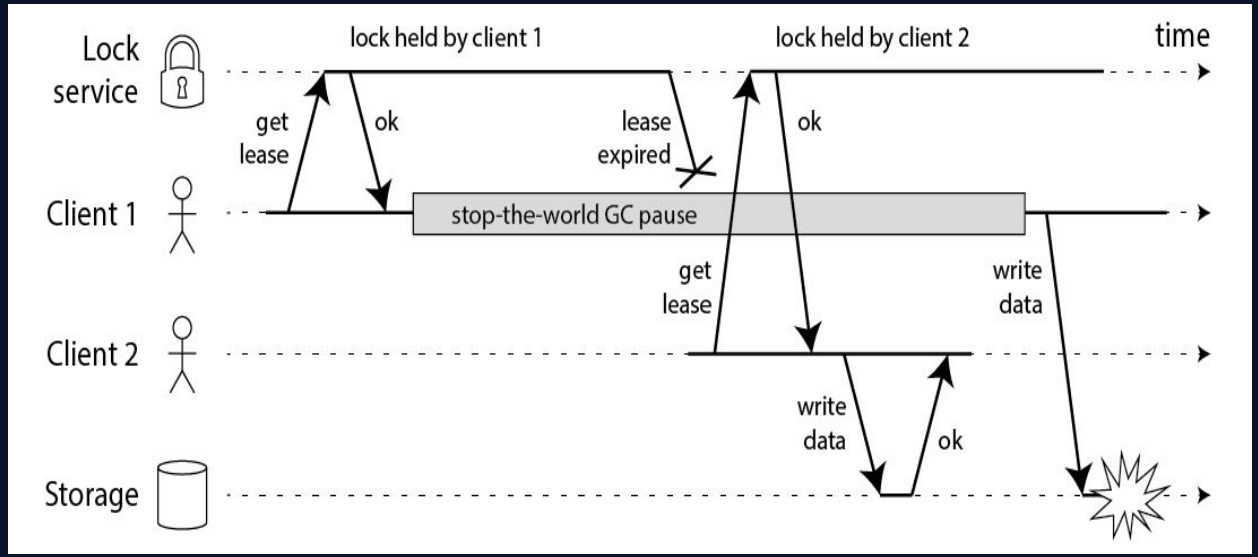
Stop-the-world GC pauses & execution delays

02. Network Partitions

Asynchronous delays & packet loss

03. Zombie Clients

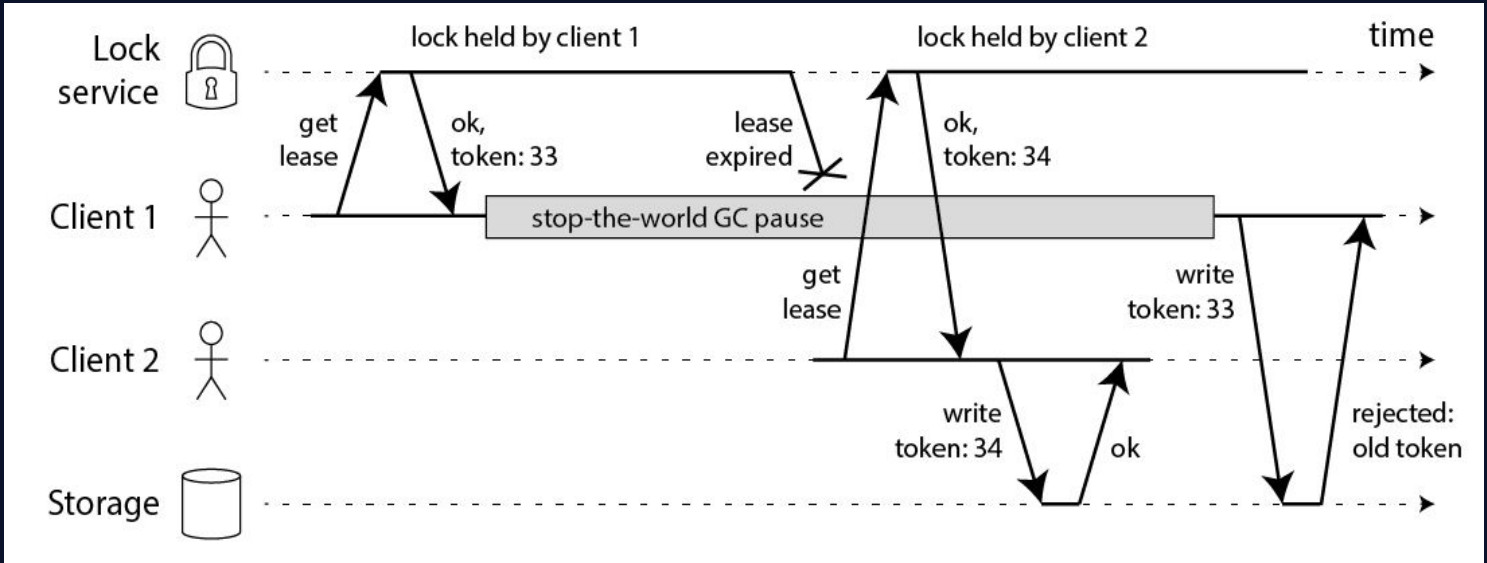
Clients acting on expired lock leases



Reference: https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html

● Distributed Systems Solution

Fencing Tokens



Reference: https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html

Distributed Systems Architecture

Three-Party Contract

01. Coordination Service

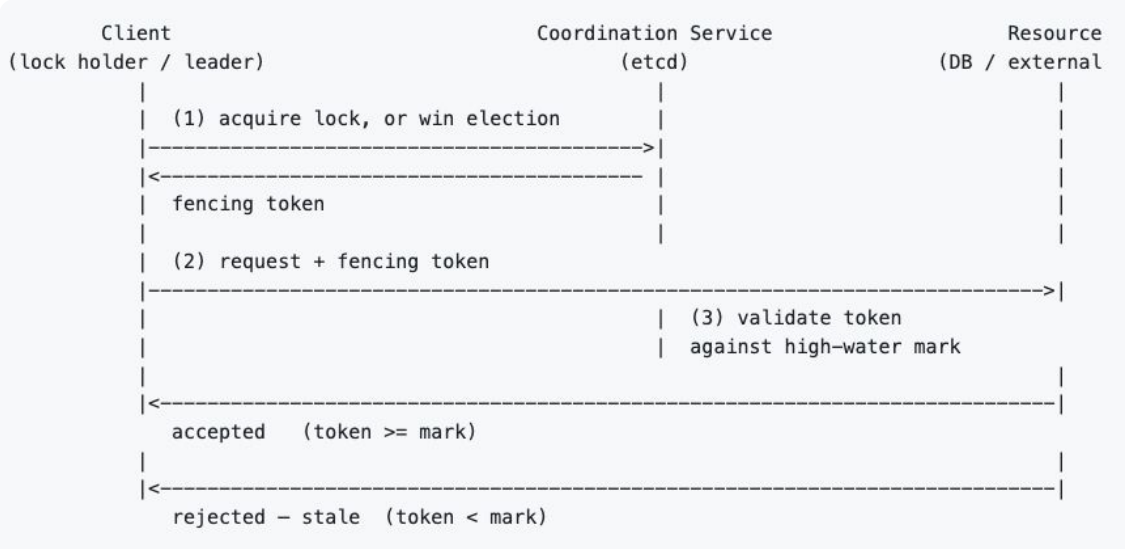
Assigns latest owner & issues fencing tokens (e.g., etcd)

02. Client Application

Attaches fencing token to every request sent to resource

03. Target Resource

Rejects stale owners by validating incoming fencing tokens



Distributed Coordination Framework

Introducing disco

01. Core System

A distributed coordination system fully implementing the Three-Party contract.

REPOSITORY

- https://github.com/ahrtr/disco

HIGHLIGHTS

- Implement Three-Party Contract
- Fencing token support
- Strict safety semantics

02. Supported Primitives

Out-of-the-box distributed synchronization tools.

- Lock
- RWLock
- Semaphore
- Leader Election

Want more? Contributions are welcome!

03. Coordination Service

Provider-based pluggable architecture.

Etcd
✔ Fully Supported

Zookeeper
Planned / Help wanted

Redis
Planned / Help wanted

● Deep Dive into Implementation

Lock Service Implementation

01. Lock Interface

```go
type Service interface {  11 usages  1 implementation  Benjamin Wang
    // Lock acquires the distributed lock, blocking until it is available or
    // ctx is canceled. Returns a Grant carrying the fencing token and lease
    // metadata for this acquisition.
    Lock(ctx context.Context) (*Grant, error)  1 implementation

    // TryLock attempts to acquire the lock without blocking.
    // Returns ErrLockTaken immediately if the lock is held by another owner.
    TryLock(ctx context.Context) (*Grant, error)  1 implementation

    // Unlock explicitly releases the lock. The underlying lease remains alive
    // so Lock can be called again without creating a new Service.
    Unlock(ctx context.Context) error  1 implementation

    // Done returns a channel that is closed when the underlying lease is lost
    // (expired or revoked). Once closed, callers must immediately stop
    // accessing guarded resources and must not call Lock again.
    // Call Close to release backend resources.
    Done() <-chan struct{}  1 implementation

    // Err returns ErrLockLost if the lease has been lost, nil otherwise.
    // Safe to call concurrently at any point in the Service lifetime.
    Err() error  1 implementation

    // Close revokes the lease and releases all backend resources.
    Close() error  1 implementation
}
```

02. etcd Provider Implementation

```go
func NewLock(client *clientv3.Client, key string, opts ...ProviderOption) (lock.Service, error) {
    session, err := newProviderSession(client, key, opts...)
    if err != nil { return nil, err }

    return &Provider{
        key:     key,
        session: session,
        mutex:   newMutex(session, key),
    }, nil
}
```

03. Grant Structure

```go
type Grant struct {  14 usages  Benjamin Wang
    // Key is the distributed lock key that was acquired.
    Key string

    // FencingToken is a monotonically increasing integer assigned at lock
    // acquisition time. It must be attached to every resource request so the
    // resource can reject requests from stale (lower-token) owners.
    FencingToken int64
}
```

● Deep Dive into Implementation

Client Application Implementation

01. gRPC Metadata Propagation

```
const GRPCKey = "x-fencing-token"  3 usages  Benjamin Wang

func ToGRPCMetadata(t Token) metadata.MD {  10 usages  Benjamin Wang
    return metadata.Pairs(GRPCKey, strconv.FormatInt(int64(t), base: 10))
}
```

```
Example:
outCtx := metadata.NewOutgoingContext(ctx, fencing.ToGRPCMetadata(grant.Token()))
resp, err := rc.DoSomething(outCtx, &pb.WriteRequest{...})
```

02. HTTP Header Injection

```
const HTTPHeader = "X-Fencing-Token"  4 usages  Benjamin Wang

func InjectHTTP(req *http.Request, t Token) {  13 usages  Benjamin Wang
    req.Header.Set(HTTPHeader, strconv.FormatInt(int64(t), base: 10))
}
```

Deep Dive into Implementation

Target Resource Implementation

01. gRPC Unary Interceptor

```go
func (g *Guard) UnaryInterceptor() grpc.UnaryServerInterceptor {  Benjan
    return func(
        ctx context.Context,
        req any,
        _ *grpc.UnaryServerInfo,
        handler grpc.UnaryHandler,
    ) (any, error) {
        token, err := fencing.FromGRPCContext(ctx)
        if err != nil {
            return nil, status.Error(codes.InvalidArgument, err.Error())
        }
        if err := g.Check(token); err != nil {
            return nil, status.Error(codes.Aborted, err.Error())
        }
        return handler(ctx, req)
    }
}
```

02. HTTP Middleware

```go
func (g *Guard) HTTPMiddleware(next http.Handler) http.Handler {  5 usages
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        token, err := fencing.ExtractHTTP(r)
        if err != nil {
            http.Error(w, err.Error(), http.StatusBadRequest)
            return
        }
        if err := g.Check(token); err != nil {
            http.Error(w, err.Error(), http.StatusConflict)
            return
        }
        next.ServeHTTP(w, r)
    })
}
```

03. Token Verification & Fencing

```go
func (g *Guard) Check(token fencing.Token) error {  Benjamin
    if token == fencing.Zero {
        return fencing.ErrNoToken
    }
    t := int64(token)
    for {
        cur := g.highWater.Load()
        if t < cur {
            return fencing.ErrTokenStale
        }

        if g.highWater.CompareAndSwap(cur, t) {
            return nil
        }
        // Another goroutine changed the mark; re-evaluate.
    }
}
```

Live Demo

Live Demo: example/lock

01. Lock Service

Start the lock service

Launches the core distributed lock coordinator process.

```
$ go run ./server/src/cmd/discod
```

02. Resource Service

Start target resource service

Runs the guarded resource API requiring fencing tokens.

```
$ go run ./examples/lock/http/resource
```

03. Client Applications

Start the client applications

Executes client instances contending for lock ownership.

```
$ go run ./examples/lock/http/client
```

Wrap Up

Questions & Answers

Thank You!