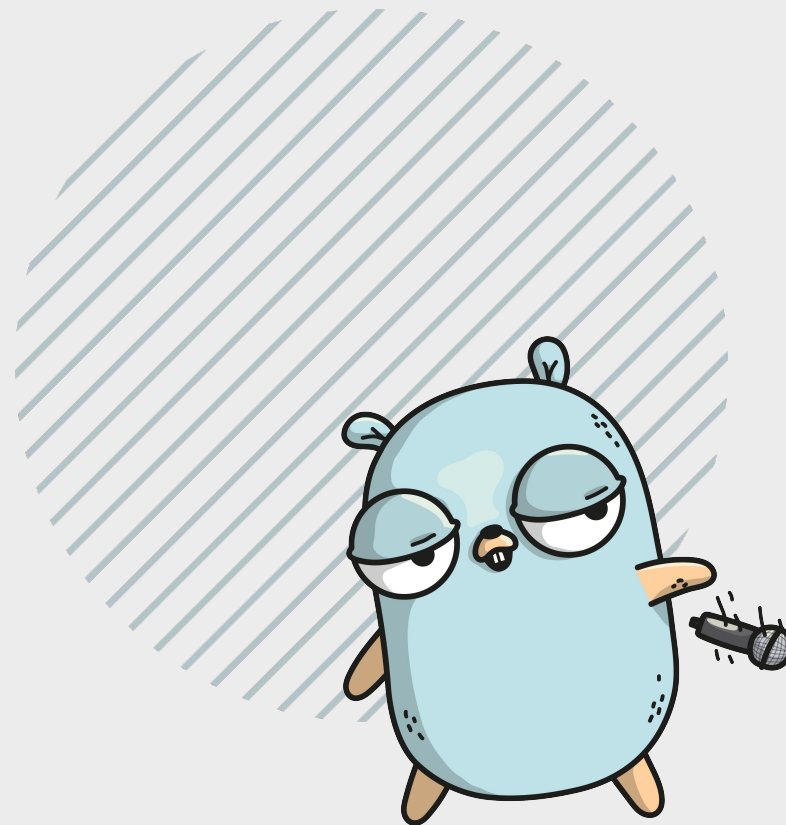


Go로 배우는 분산 시스템

Distributed systems in Go

김수빈 / 당근마켓



Speaker



김수빈

당근마켓

- 당근마켓에서 인터널 프로덕트 개발을 하고 있습니다.
- 플랫폼 개발을 좋아합니다.
- 주로 **Go**와 **Python**을 사용합니다.



github.com/sudosubin




GopherCon Korea 2024

Index

- 분산 시스템이란?
- 차근차근 만들어보는 분산 시스템
- 챌린지
- 마무리





분산 시스템이란?



분산 시스템

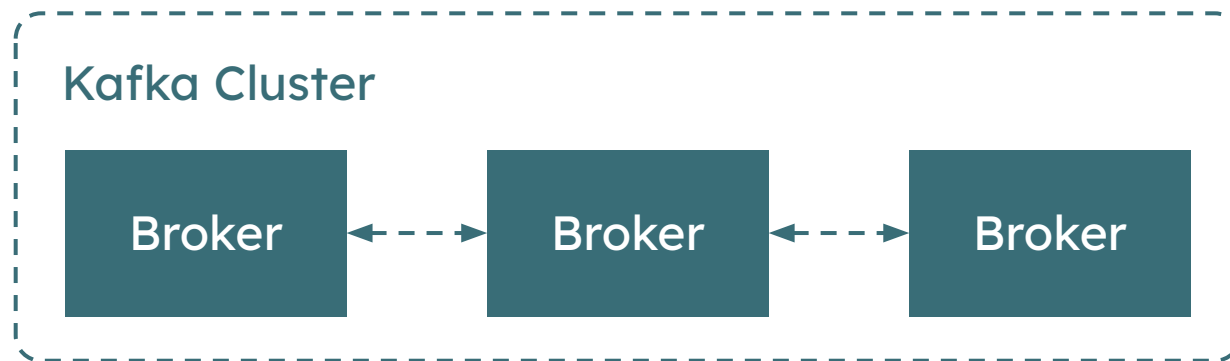
- 특정 작업을 여러 노드에서 실행
- 네트워크로 연결



Apache Kafka

Distributed Messaging System

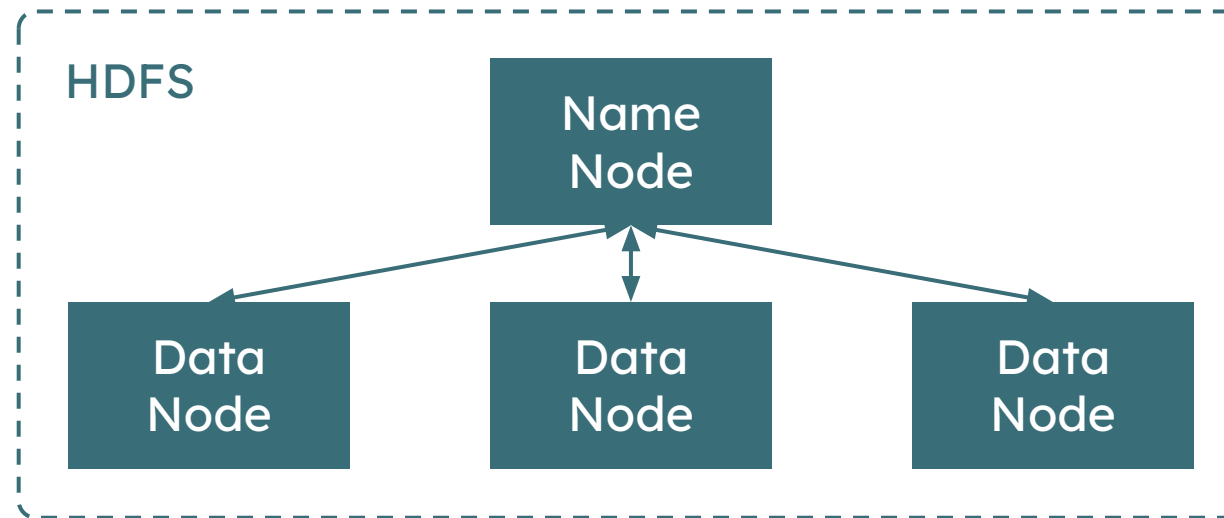
- 여러 개의 메시지 브로커가 클러스터의 구조로 협력
- 여러 개의 **Producer, Consumer**가 메시지 처리를 위해 협력
- 수평 확장 가능



Hadoop (HDFS)

Distributed File System

- Name Node는 클라이언트의 요청을 받아 Data Node에 요청을 전달
- 여러 개의 Data Node를 통해 데이터 저장, 읽기, 쓰기를 분산 처리



차근차근 만들어보는 분산 시스템



Maelstrom

- 분산 시스템을 구현하고 테스트하기 위한 도구
- STDIN, STDOUT을 이용한 단순한 구현
- Go 언어로 구현된 **Demo** 패키지로 빠르게 시작!

 github.com/jepsen-io/maelstrom



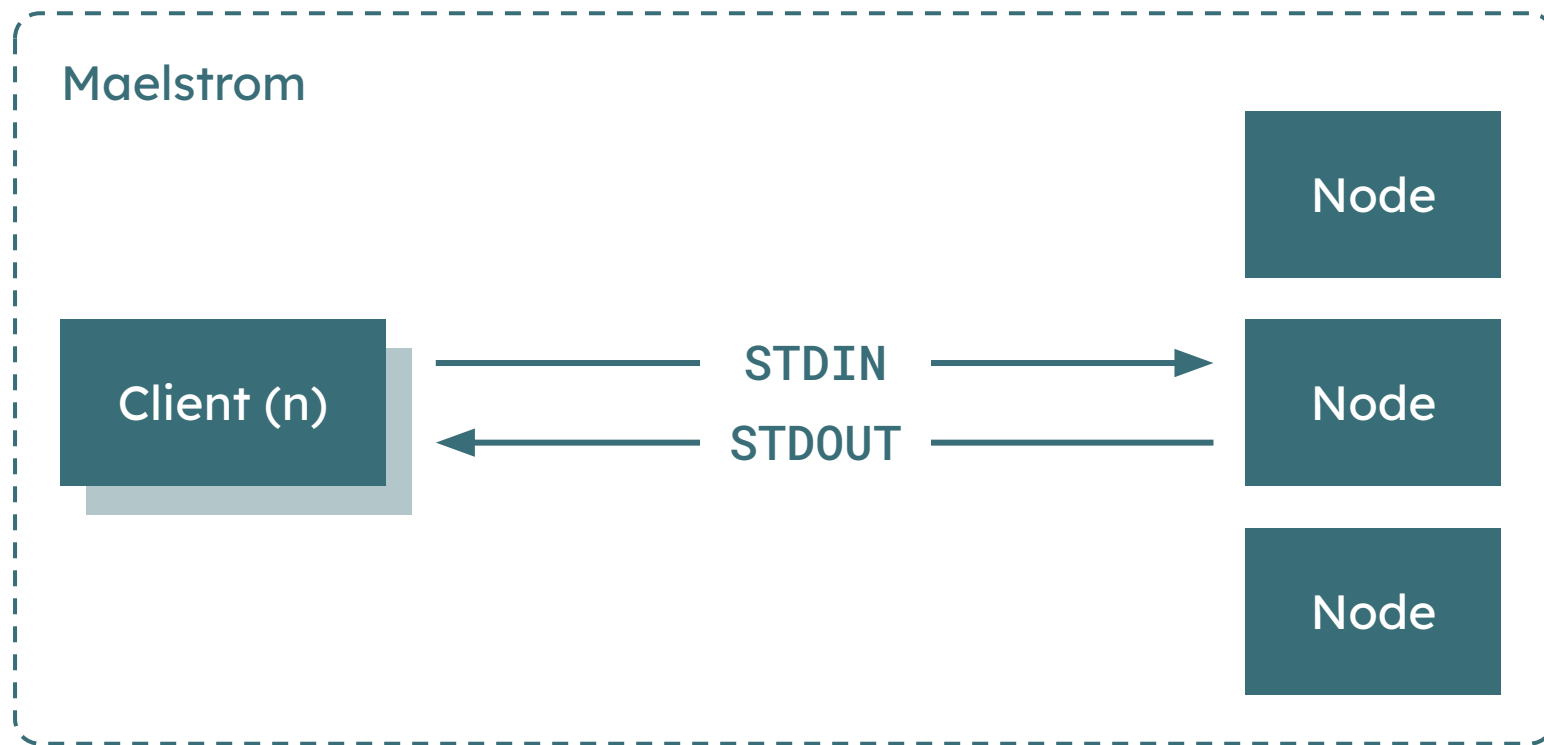
순서

1. 구조 알아보기
2. 단일 노드
3. 분산 시스템



구조 알아보기

Maelstrom의 시스템 디자인



STDIN

```
{
  "src": "c1",
  "dest": "n1",
  "body": {
    "msg_id": 1,
    "type": "echo",
    "echo": "hello there"
  }
}
```

STDOUT

```
{
  "src": null,
  "dest": "c1",
  "body": {
    "type": "echo_ok",
    "echo": "hello there",
    "msg_id": 2,
    "in_reply_to": 1
  }
}
```



```
func main() {  
    n := maelstrom.NewNode()  
  
    n.Handle("echo", func(msg maelstrom.Message) error {  
        var body map[string]any  
        if err := json.Unmarshal(msg.Body, &body); err != nil {  
            return err  
        }  
        body["type"] = "echo_ok"  
        return n.Reply(msg, body)  
    })  
  
    if err := n.Run(); err != nil {  
        log.Fatal(err)  
    }  
}
```



단일 노드

Broadcast

- broadcast, read 구현
- broadcast 요청으로 받은 값들 → read 요청 받으면 모두 반환



STDIN

```
{  
  "type": "broadcast",  
  "message": 8  
}
```

STDOUT

```
{  
  "type": "broadcast_ok"  
}
```

STDIN

```
{  
  "type": "read"  
}
```

STDOUT

```
{  
  "type": "read_ok",  
  "messages": [1, 8, 72, 25]  
}
```



```
func main() {  
    n := maelstrom.NewNode()  
    mu := new(sync.Mutex)  
    messages := []int{}  
  
    n.Handle("broadcast", func(msg maelstrom.Message) error {  
        ...  
    })  
  
    n.Handle("read", func(msg maelstrom.Message) error {  
        return n.Reply(msg, map[string]any{  
            "type":      "read_ok",  
            "messages": messages,  
        })  
    })  
}
```




```
n.Handle("broadcast", func(msg maelstrom.Message) error {  
    var body map[string]any  
  
    if err := json.Unmarshal(msg.Body, &body); err != nil {  
        return err  
    }  
  
    mu.Lock()  
    messages = append(messages, int(body["message"].(float64)))  
    mu.Unlock()  
  
    return n.Reply(msg, map[string]any{"type": "broadcast_ok"})  
})
```



분산 시스템

Broadcast

- 여러 노드가 각각 **broadcast, read** 요청 처리
- 모든 값은 몇 초 이내에 전체 노드에 전파되어야
- 다른 노드가 받은 값은 어떻게 보여주지?



STDIN

```
{  
  "type": "topology",  
  "topology": {  
    "n1": ["n2", "n3"],  
    "n2": ["n1"],  
    "n3": ["n1"]  
  }  
}
```

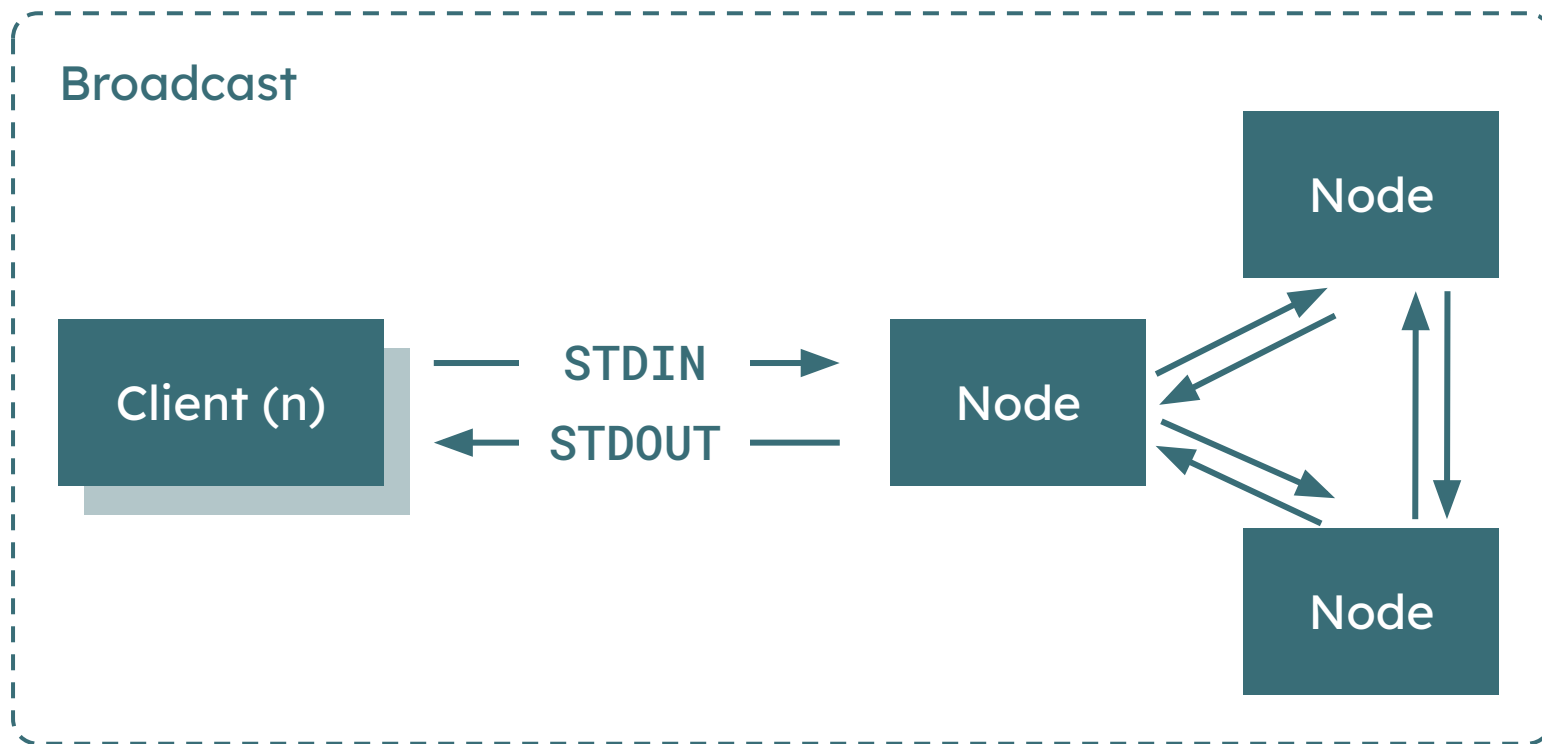
STDOUT

```
{  
  "type": "topology_ok"  
}
```



분산 시스템

Broadcast



```
...  
if err := json.Unmarshal(msg.Body, &body); err != nil {  
    return err  
}
```

```
for _, id := range getNodes(n, topology) {  
    if id != msg.Src {  
        go func() {  
            if err := n.Send(id, body); err != nil {  
                panic(err)  
            }  
        }()  
    }  
}
```

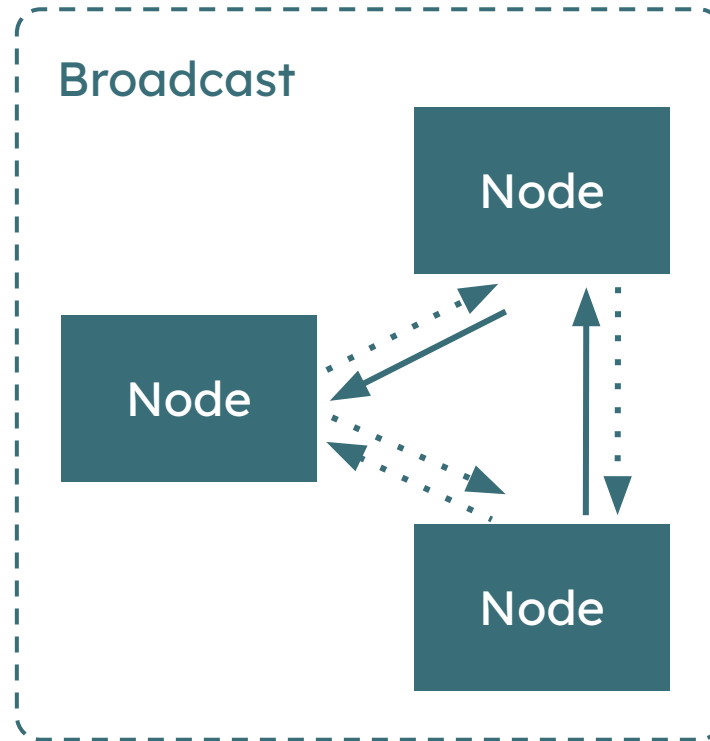
```
mu.Lock()  
messages = append(messages, int(body["message"].(float64)))  
...
```



분산 시스템

Broadcast, with Fault Tolerant

- Network Partition 발생
- 종종 누락되는 Node 간 메시지
- → Eventual consistency!



분산 시스템

Broadcast, with Fault Tolerant

- 보내는 도중 실패한 경우 → 재시도 가능
- 보낸 것은 성공했지만 받는 쪽에서 실패한 경우 → 재시도 어려움
- 더군다나 우리의 플랫폼은 완전 비동기



```
go func() {  
    for { <- ticker.C  
        for _, id := range getNodes(n, topology) {  
            if id == n.ID() {  
                continue  
            }  
  
            body = map[string]any{  
                "type": "custom_messages_sync",  
                "messages": messages,  
            }  
  
            if err := n.Send(id, body); err != nil {  
                panic(err)  
            }  
        }  
    }  
}()
```




```
n.Handle("custom_messages_sync", func(msg maelstrom.Message) error {  
    var body map[string]any  
  
    if err := json.Unmarshal(msg.Body, &body); err != nil {  
        return err  
    }  
  
    mu.Lock()  
    messages = // messages와 body["messages"] 잘 합치기  
    mu.Unlock()  
  
    return n.Reply(msg, map[string]any{"type": "custom_messages_sync_ok"})  
})
```





챌린지



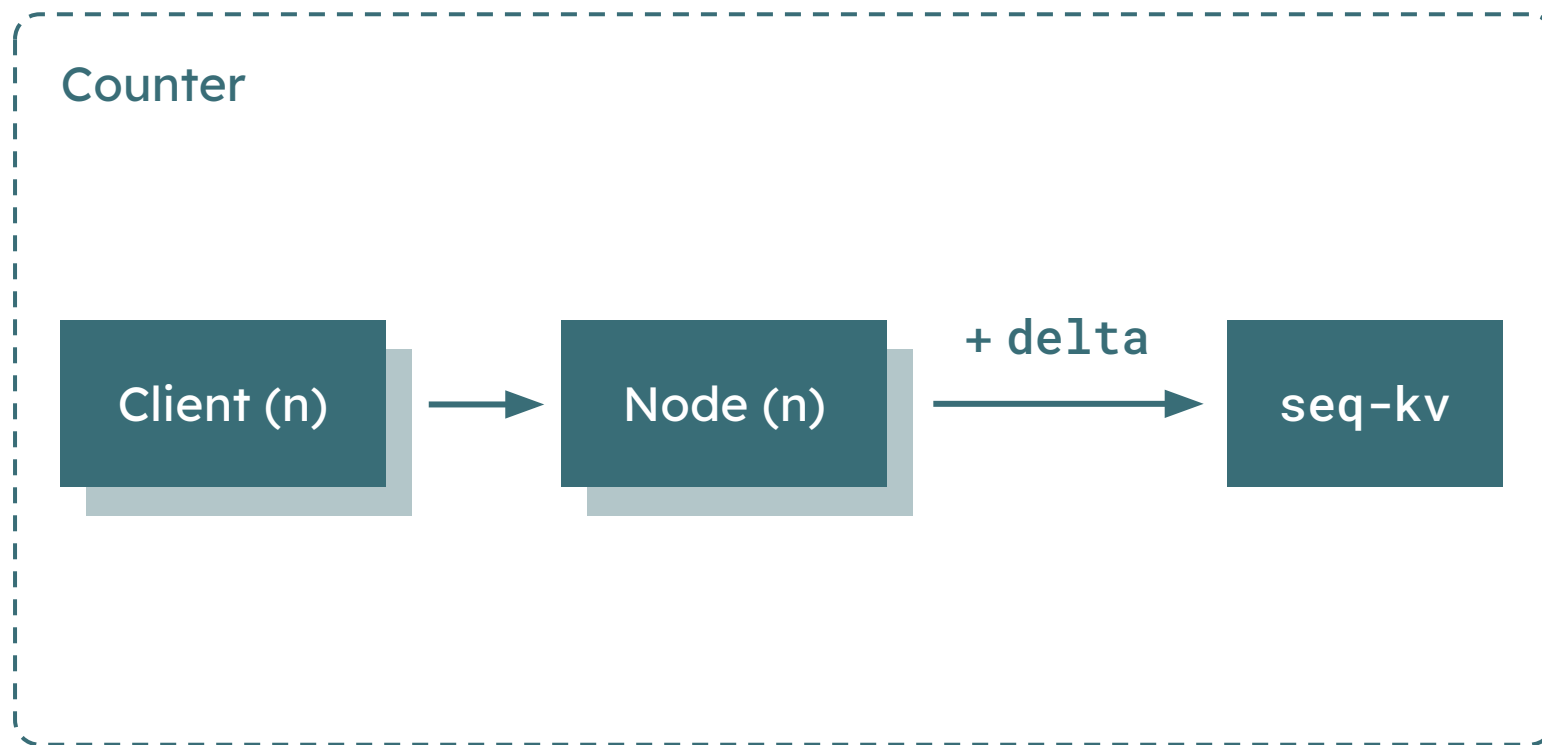
Counter 구현

- seq-kv 제공
 - Sequential Consistency
 - Read, Write, CAS
- **add** 요청이 들어오면, 중앙의 값 저장소를 **delta** 만큼 증가
- **read** 요청이 들어오면, 값 반환



Counter

Using seq-kv



Counter

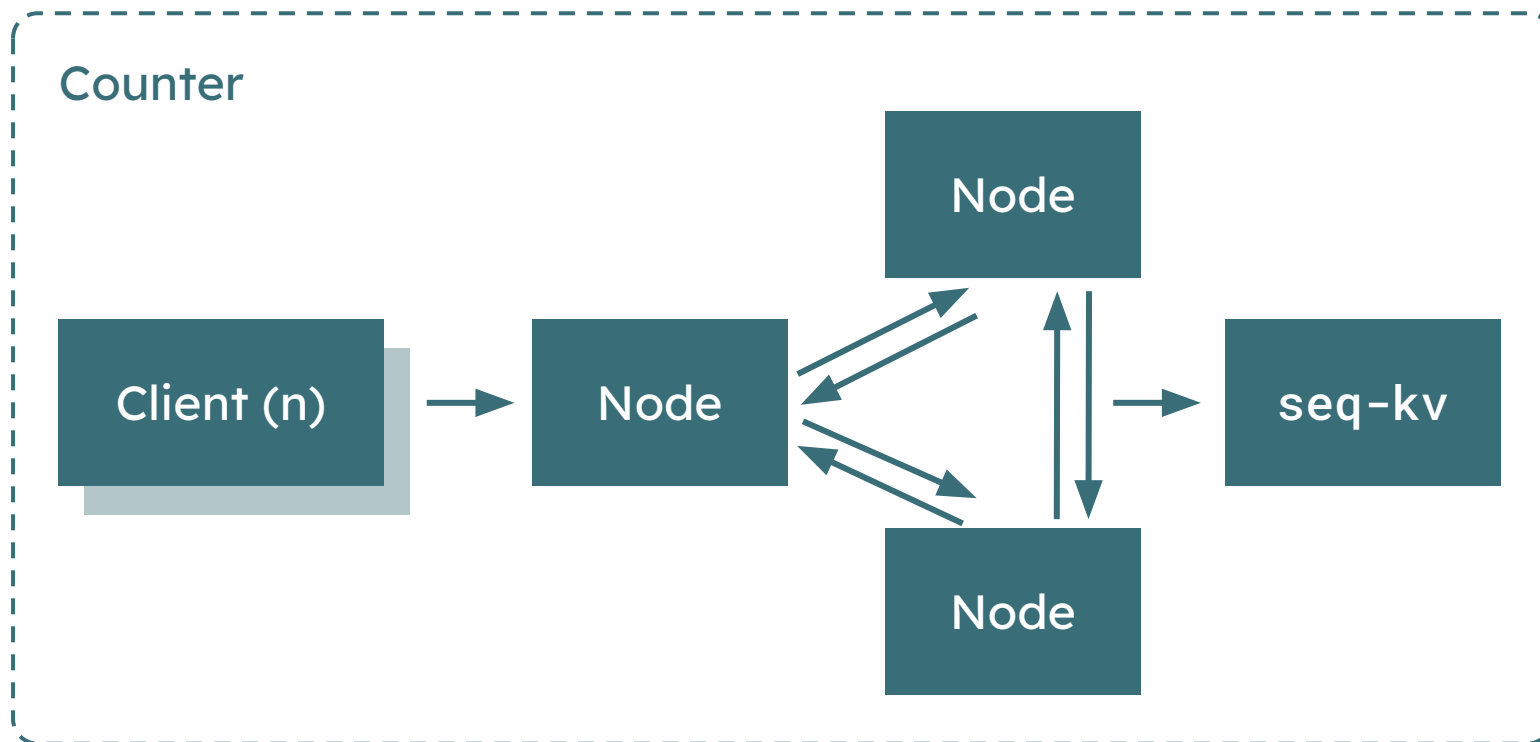
Using seq-kv

- “Sequential Consistency”
- 일부 노드에서는 **CAS**까지 잘 했는데, 조회 시에는 예전 값 반환



Counter

Using seq-kv



```
kv := maelstrom.NewSeqKV(n)

n.Handle("add", func(msg maelstrom.Message) error {
    // var body ~ json.Unmarshal(msg.Body, &body) ...

    old, err := kv.ReadInt(ctx, KEY)
    if err != nil {
        return err
    }

    delta := int(body["delta"].(float64))
    if err := kv.CompareAndSwap(ctx, KEY, old, old + delta, true); err != nil {
        return err
    }

    return n.Reply(msg, map[string]any{"type": "add_ok"})
})
```



```
n.Handle("read", func(msg maelstrom.Message) error {  
    // var body ~ json.Unmarshal(msg.Body, &body) ...  
  
    value, err := kv.ReadInt(ctx, KEY)  
    if err != nil {  
        return err  
    }  
  
    return n.Reply(msg, map[string]any{  
        "type": "read_ok",  
        "value": value,  
    })  
})
```



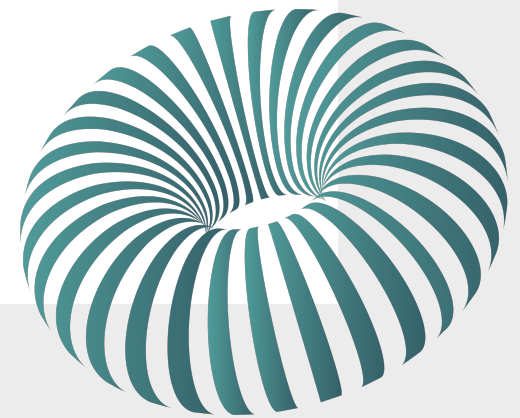
Counter

Using seq-kv

- seq-kv가 한참이 지나서도 계속해서 오래 된 값을 반환한다면?
- seq-kv에 장애 상황이 발생한다면?



마무리



분산 시스템, 어렵지 않죠?

- Availability
- Fault Tolerance
- Scalability
- Consistency



https://en.wikipedia.org/wiki/Bob_Ross#/media/File:Bob_at_Easel.jpg



JEPSEN

Blog Analyses Talks Consistency Services

Distributed Systems Safety Research



About Jepsen

Jepsen is an effort to improve the safety of distributed databases, queues, consensus systems, etc. We maintain an open source [software library](#) for systems testing, as well as [blog posts](#) and [conference talks](#) explaining distributed systems failure

News

Recent research, analyses, and announcements.

jetcd 0.8.2
2024-08-07

Thanks to References

- Maelstrom
 - github.com/jepsen-io/maelstrom
- Fly.io - Gossip Glomers
 - fly.io/dist-sys
- Medium - 분산 시스템 직접 만들어보기
 - medium.com/@sudosubin/9fbc3f2a9684






GopherCon Korea 2024


Fly.io Discourse



















Community Support

- 전 세계의 같이 챌린지를 진행하는 사람들
- 궁금한 부분은 물어보거나, 다른 사람들의 고민과 해결 방법을 찾아보면서 문제를 풀 수 있어요

 Fly.io [Log In](#)  

categories ▾ dist-sys-challenge ▾ [Latest](#) [Top](#) [Categories](#)



Topic		Replies	Activity
 Challenge #2: Unique ID Generation: is there anything smarter/better than flake ids? dist-sys-challenge		1	Jun 24
New synchronous Rust Maelstrom crate  Show & Tell dist-sys-challenge		3	Jun 11
Challenge #5c: Efficient Kafka-Style Log dist-sys-challenge		15	Jun 8
 Did I solve Challenge #3d and #3e  Questions / Help dist-sys-challenge		1	Mar 4
 <input checked="" type="checkbox"/> Challenge 6: tips  Questions / Help dist-sys-challenge		2	Jan 30
 The clock in the real world for Challenge 2 dist-sys-challenge		1	Jan 2
 Work through the Fly.io Distributed Systems Challenge in your browser! dist-sys-challenge , distributed		1	Dec 2023
 Client read timeout  Questions / Help dist-sys-challenge		1	Dec 2023

Q&A



Thank you!

