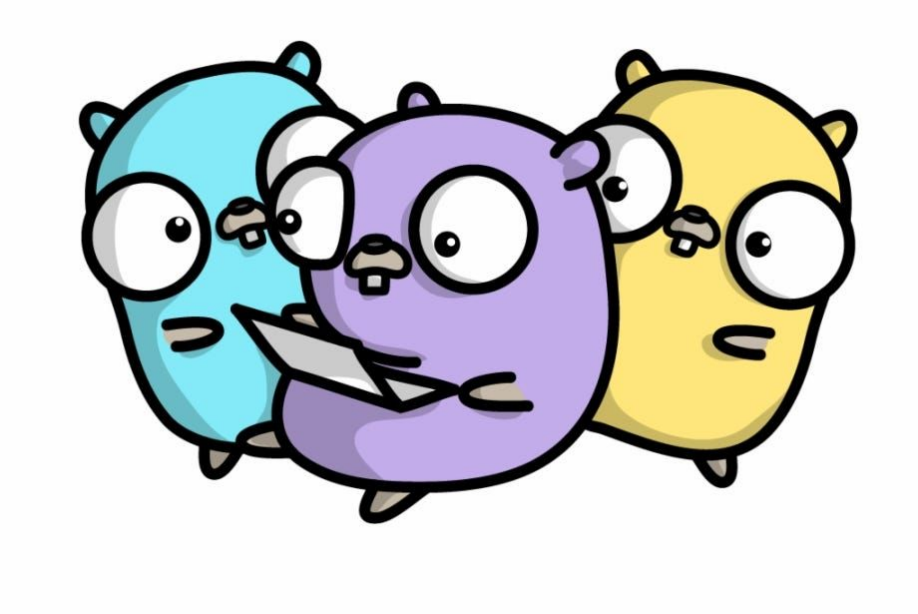
Data Movement and Replication Techniques in Go



Erhan Yakut / @yakuter

17 Şubat 2024

Biyografi

Ben Kimim?

Erhan YAKUT (yakuter)

İş / Görev

Binalyze isimli **DFIR** şirketinde **Staff Engineer** olarak çalışmaktayım.

b!nalyze

Tecrübe/Bilgi

Yaklaşık 15+ yıldır yazılım geliştirme ile ilgilenmekte olup, şu anda işletim sistemleri üzerinde olay sonrası delillerin toplanması üzerine yoğunlaşmış durumdayım.

Programlama Dilleri

Aktif olarak **Go** ile cross platform sistem araçları yazma üzerinde çalışıyorum.

İletişim Bilgisi

Twitter : **@yakuter**
Eposta : **yakuter@gmail.com**
Website: **yakuter.com**

Örnek İş İlanı

Skills and Knowledge:

- 6+ years experience delivering production code for backend services.
- Experience with Golang
- Experience with HTTP/REST and web technologies like gRPC, Protobufs
- Excellent knowledge of one of the databases such as PostgreSQL, Mongo, MYSQL
- Experience building and deploying Docker containers and working knowledge of Kubernetes
- Knowledge of data-driven architecture and systems design
- Knowledge of the philosophy of cloud and an experience maintaining cloud native services (AWS)
- Experience of designing, building, scaling and maintaining production services, and knowledge about how to compose a service oriented architecture.
- Experience creating, managing and maintaining Micro Services.
- Knowledge of the build, deploy pipelines using CI/CD tools for application deployment.
- Familiar with common logging, monitoring, alerting tools such as Datadog, Grafana,NewRelic, Prometheus, Icinga, Kibana, etc.
- Experience with .net core is a plus.

copy Function

copy slices

```go
src := []int{1, 2, 3, 4, 5}
dst := make([]int, 5)

// copy from slice to slice
copied := copy(dst, src)
// copied: 5
// dst: [1 2 3 4 5]

// copy from string to slice
var b = make([]byte, 5)
copy(b, "Hello, world!")
// b == []byte("Hello")
```

append Function

merge slices

```go
var s []int
// len=0 cap=0 []

s = append(s, 0) // append works on nil slices.
// len=1 cap=1 [0]

s = append(s, 1) // The slice grows as needed.
// len=2 cap=2 [0 1]

s = append(s, 2, 3, 4) // We can add more than one element at a time.
// len=5 cap=6 [0 1 2 3 4]

s = append(s, 5, 6)
// len=7 cap=12 [0 1 2 3 4 5 6]

anotherSlice := []int{7, 8, 9, 10, 11, 12}
s = append(s, anotherSlice...) // We can combine two slices
// len=13 cap=24 [0 1 2 3 4 5 6 7 8 9 10 11 12]
```

channel

Don’t communicate by sharing memory, share memory by communicating!

```go
messageChannel := make(chan string)

go func() {
    messageChannel <- "Hello, Channel!"
}()

message := <-messageChannel
fmt.Println("Received message:", message)
```

io.Writer ve io.Reader

Write ve Read ile basit i/o işlemleri

- io.Writer ve io.Reader interfaceleri, veri yazma ve okuma işlemleri için standart bir arayüz sağlar.

- Go'nun I/O işlemlerinin temelini oluşturur.

- io.Writer interface'i, bir veri akışına veri yazmayı sağlayan Write metodunu tanımlar. Bu interface, dosyalara, buffer'lara, ağ bağlantılarına ve daha fazlasına veri yazmak için kullanılabilir.

- io.Reader interface'i, bir veri akışından veri okumayı sağlayan Read metodunu tanımlar.

Kaynak kod

io.Reader’da En Çok Yapılan Hata

Tam okunduğunu sanmak!

Veri Kaynağında Yeterli Veri Yok: Eğer veri kaynağında sadece 5 byte veri kaldıysa, Read çağrısı yalnızca bu 5 byte’ı okur ve size geri döndürür. Bu durumda, n (okunan byte sayısı) değeri 5 olur.

Ağ Gecikmeleri ve Veri Parçalanması: Ağ üzerinden veri okurken, veri paketleri parçalara ayrılabilir veya gecikmeler yaşanabilir. Bu nedenle, her Read çağrısı farklı miktarda veri alabilir. Bir çağrıda 8 byte beklerken, gerçekte daha az veri okunabilir.

Dosya Sonu (EOF) Durumu: Dosya sonuna ulaşıldığında, kalan veri 8 byte’tan azsa, Read yalnızca kalan veriyi okur ve io.EOF hatası döndürür.

io.Reader Yanlış Kullanımı

Don’t communicate by sharing memory, share memory by communicating!

```go
reader := strings.NewReader("Test")
buf := make([]byte, 8)

// Wrong: Assuming exactly 8 bytes will be read
n, err := reader.Read(buf)
if err != nil {
    fmt.Println("Failed to read:", err)
    return err
}
fmt.Printf("Read data: %s\n", string(buf[:n]))
```

io.Reader Doğru Kullanımı

Don’t communicate by sharing memory, share memory by communicating!

```go
reader := strings.NewReader("This is a test message.")
buf := make([]byte, 8)
for {
    n, err := reader.Read(buf)
    if n > 0 {
        fmt.Printf("Read data: %s\n", string(buf[:n]))
    }
    if err == io.EOF {
        break // End of the file. Exit from loop.
    }
    if err != nil {
        fmt.Println("Failed to read:", err)
        return err
    }
}
```

bytes.Buffer

Hafızada veri saklamak

bytes.Buffer genellikle bellekte veri saklamak ve bu veri üzerinde çeşitli okuma/yazma işlemleri yapmak için kullanılır.

```go
var buffer bytes.Buffer

buffer.WriteString("Hello, bytes.Buffer!")

fmt.Println(buffer.String())
```

bufio

Hafızada i/o işlemleri

```go
file, err := os.Open("example.txt")
if err != nil {
    return err
}
defer file.Close()

// Create a buffered reader
scanner := bufio.NewScanner(file)

// Read the file line by line
for scanner.Scan() {
    line := scanner.Text()
    fmt.Println(line)
}

// Check for errors during scanning
if err := scanner.Err(); err != nil {
    fmt.Println("Error reading file:", err)
}
```

io.Pipe

Etkili I/O işlemleri

- io.Pipe bir pipe oluşturarak, bir yazma işlemi ile bir okuma işlemi arasında veri akışı sağlar.
- Bu, genellikle farklı goroutine'ler arasında veri aktarımı yaparken kullanılır.
- Pipe fonksiyonu, bir **io.PipeReader** ve bir **io.PipeWriter** döndürür.
- **PipeWriter** üzerine yazılan her şey, **PipeReader** tarafından okunabilir hale gelir.
- Bu yapı, özellikle stream edilen veriler veya büyük veri setlerinin işlenmesinde oldukça yararlıdır çünkü verilerin tamamının belleğe alınmasını gerektirmez.

io.Pipe

Etkili I/O işlemleri

```go
reader, writer := io.Pipe()

// PipeWriter'a yazma işlemini bir gorutinde gerçekleştir
go func() {
    _, err := writer.Write([]byte("Merhaba, io.Pipe kullanımı!"))
    if err != nil {
        fmt.Println("Yazma hatası:", err)
        return
    }
    writer.Close()
}()

// PipeReader'dan okuma işlemini ana gorutinde gerçekleştir
if _, err := io.Copy(os.Stdout, reader); err != nil {
    fmt.Println("Okuma hatası:", err)
    return
}
```

io.Copy

Etkili I/O kopyalama işlemleri

```
4   sourceFile, err := os.Open(sourcePath)
5   if err != nil {
6       fmt.Println("Failed to open source file:", err)
7       return
8   }
9   defer sourceFile.Close()
10
11  destinationFile, err := os.Create(destinationPath)
12  if err != nil {
13      fmt.Println("Failed to create destination file:", err)
14      return
15  }
16  defer destinationFile.Close()
17
18  _, err = io.Copy(destinationFile, sourceFile)
19  if err != nil {
20      fmt.Println("Failed to copy:", err)
21      return
22  }
```

HTTP Request ile Dosya Gönderme

```go
// Create reader and writer using io.Pipe
reader, writer := io.Pipe()

// Read the file and write to the pipe in a goroutine
go func() {
    file, err := os.Open(filePath)
    if err != nil {
        fmt.Println("Error opening file:", err)
        return
    }
    defer file.Close()

    // Copy file content to the pipe
    _, err = io.Copy(writer, file)
    if err != nil {
        fmt.Println("Error writing file to the pipe:", err)
        return
    }

    // Close the writer when the writing operation is done
    writer.Close()
}()

// Create HTTP request and send file content as body
resp, err := http.Post("http://example.com/upload", "application/octet-stream", reader)
if err != nil {
    fmt.Println("Error during HTTP request:", err)
    return
}
defer resp.Body.Close()
```

HTTP Server ile Dosya Stream Etme

```go
http.HandleFunc("/download", func(w http.ResponseWriter, r *http.Request) {
    file, err := os.Open(filePath)
    if err != nil {
        http.Error(w, "File not found", http.StatusNotFound)
        return
    }
    defer file.Close()

    // Stream the file content to the client
    // Set headers
    w.Header().Set("Content-Disposition", "attachment; filename="+filePath)
    w.Header().Set("Content-Type", "application/octet-stream")
    // Copy the file to the HTTP response
    _, err = io.Copy(w, file)
    if err != nil {
        http.Error(w, "Error streaming file", http.StatusInternalServerError)
    }
})

println("Server listening on port 8080...")
if err := http.ListenAndServe(":8080", nil); err != nil {
    panic(err)
}
```

Thank You

yakuter@gmail.com
@yakuter