S3 - many objects

S3 - many objects

Create 30.000 objects as fast as possible

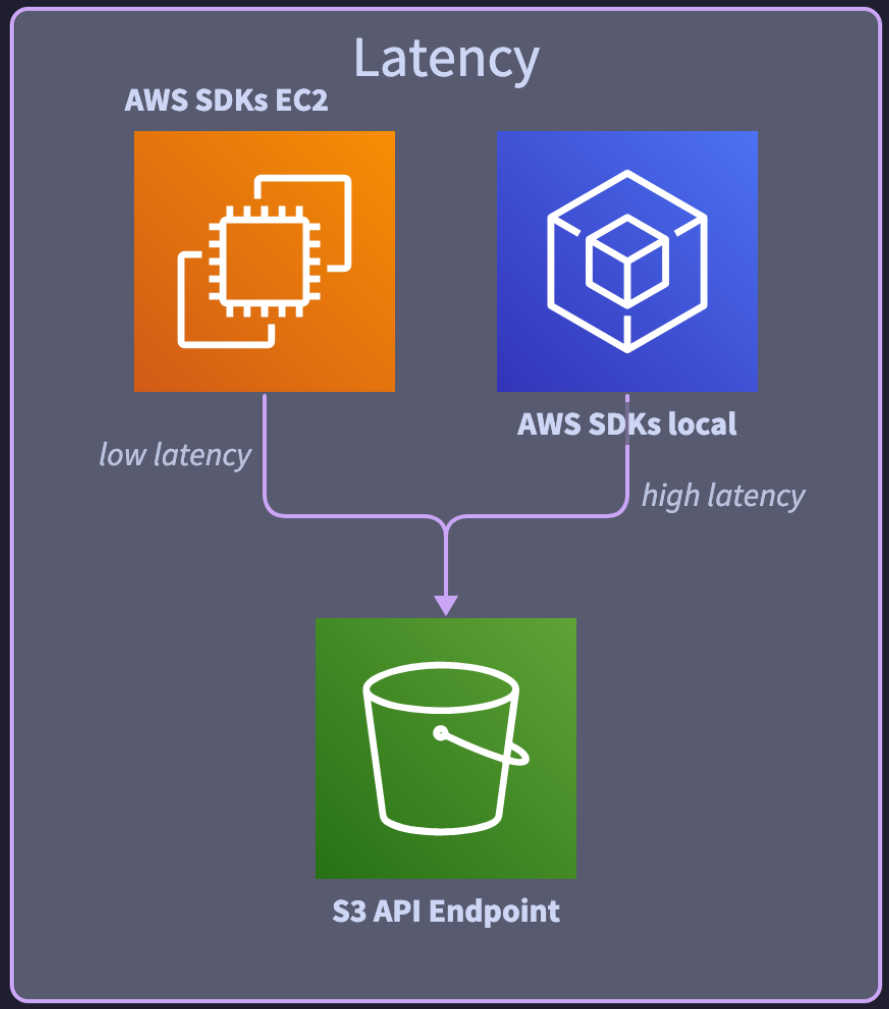
Actions on many objects

Usecase: Create 30.000 objects as fast as possible

- A bash loop with `aws s3 cp`
- A GO program with a single goroutine
- A GO program with 100 goroutines
- A GO program with 100 goroutines on AWS cloudshall

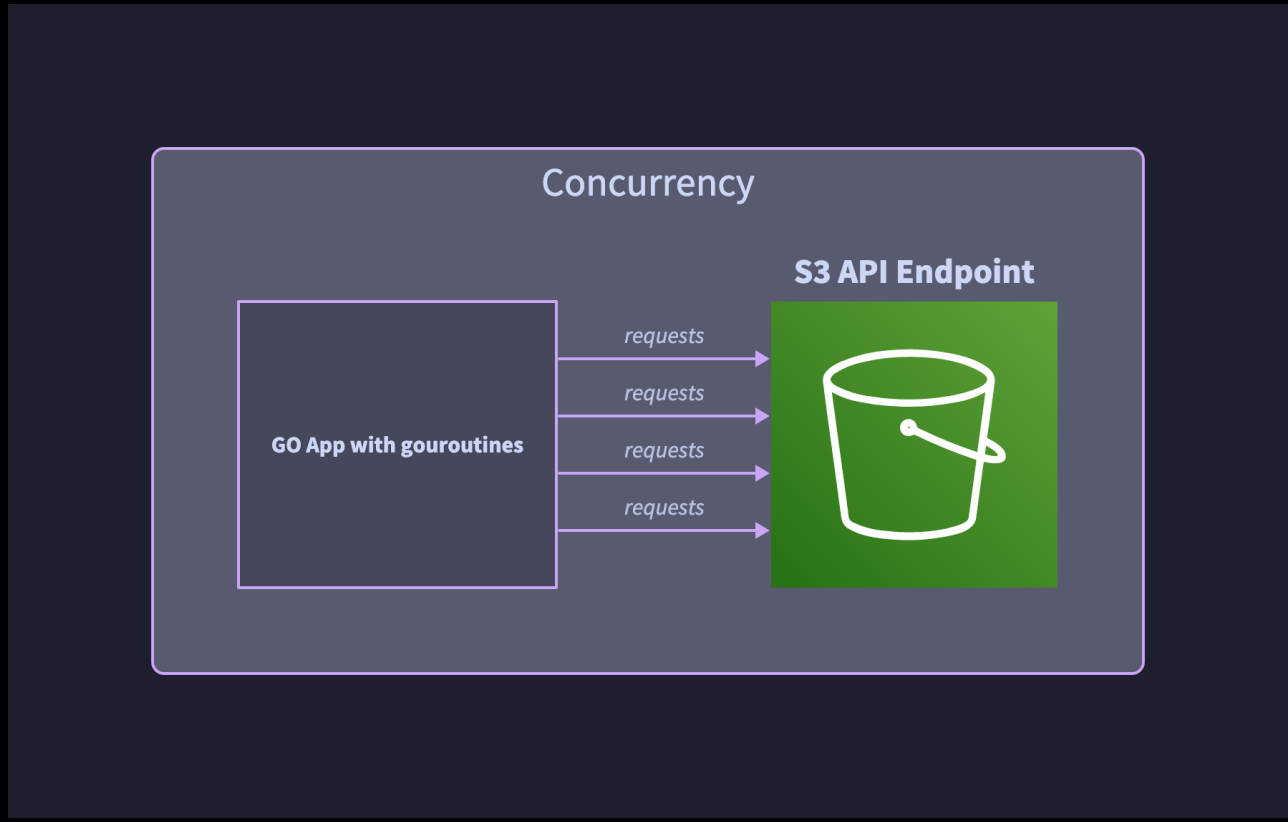
AWS performance considerations

- The latency is much smaller on AWS compute units
- Traffic is routed directly on AWS network
- Consider size of data transfered

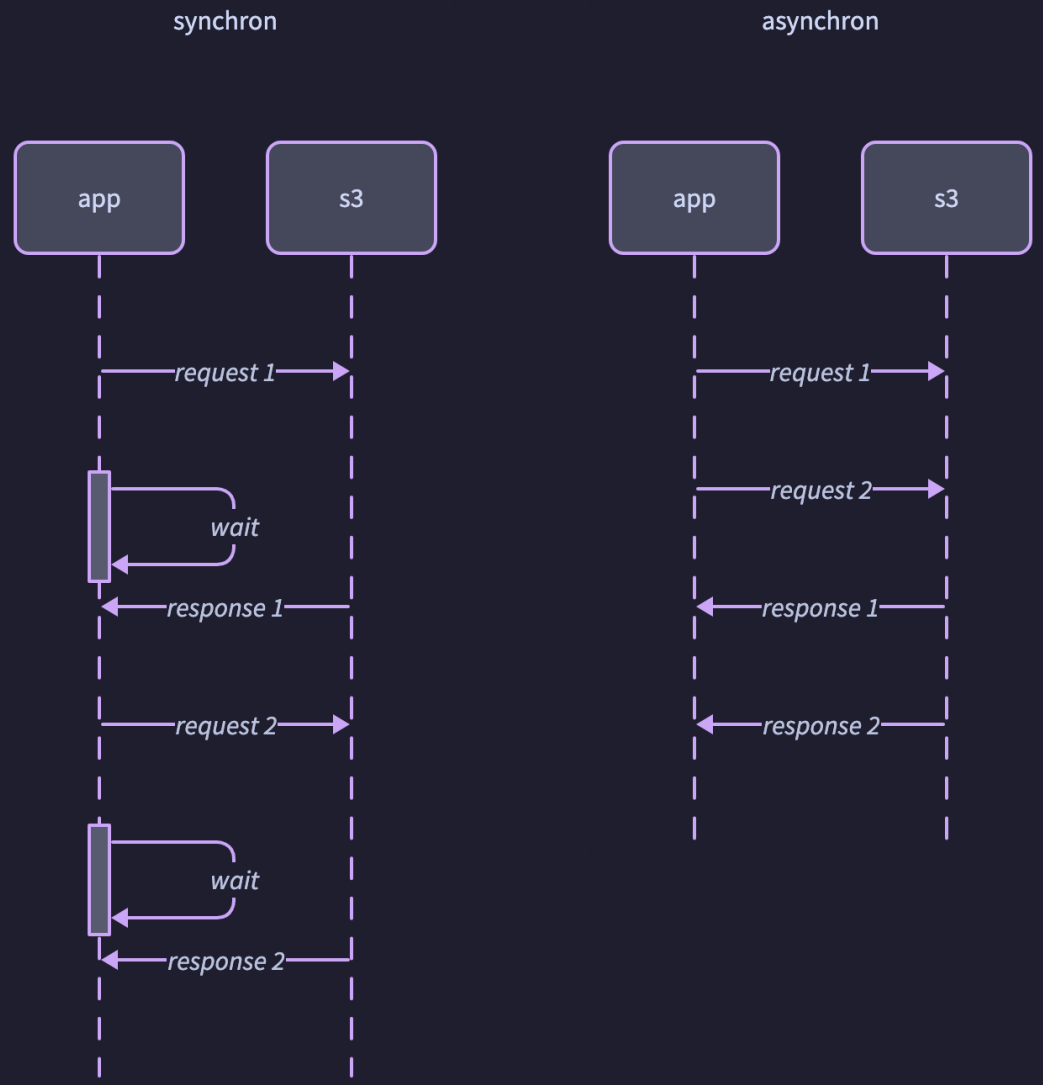


AWS performance considerations

- AWS services can be called in parallel
- S3 doesn't have any limits for the number of connections
- Consider compute power, memory and network bandwidth



Use asynchronous calls



Concurrency in GO

- Parallel functions in GO are called goroutines
- Goroutines are lightweight threads managed by GO runtime
- A "go" statement starts the execution of a function call as a goroutine
- Goroutines run in the same address space, so access to shared memory must be synchronized
- Wait for goroutines to finish with `sync.WaitGroup`

Concurrency in GO

```
1:      for i := 0; i < numGoroutines; i++ {
2:              i := i //shadow
3:              go func(num int) {
4:                      defer wg.Done()
5:                      sleepDuration := 1 + rand.Intn(3) // Sleep for 5 to 10 seconds
6:                      time.Sleep(time.Duration(sleepDuration) * time.Second)
7:                      fmt.Printf("Goroutine %d: Loop number %d\n", num, i)
8:              }(i)
9:      }
```

```
Goroutine 0: Loop number 0
Goroutine 1: Loop number 1
Goroutine 4: Loop number 4
Goroutine 3: Loop number 3
Goroutine 2: Loop number 2
All goroutines completed
```

optimize/gouroutines/main.go

Wait for goroutines to finish

```
1:      const numGoroutines = 5
2:      wg := sync.WaitGroup{}
3:      wg.Add(numGoroutines)
4:      // run go func
5:      // wg.Done()
6:      wg.Wait()
```

Let's create 30.000 objects

1 Bash

2 GO, single goroutine

3 GO, 100 goroutines

4 GO, 100 goroutines on AWS cloudshell

Alternative approach: S3 Batch Operations

- Service to call Lambda with a list of objects
- S3 Batch takes care of the concurrency
- List of objects can be created with S3 Inventory
- More on Lambda later

1: Bash

The inner loop:

```
 1:          for l in 0 1 2 3 4 5 6 7 8 9
 2:          do
 3:              for m in 0 1 2 3 4 5 6 7 8 9
 4:              do
 5:                  for o in 0 1 2 3 4 5 6 7 8 9
 6:                  do
 7:                      aws s3 cp readme.md s3://${BUCKET}/bash/readme-${i}-${k}-${l}-${m}-${o}.txt
 8:                  done
 9:              done
10:          done
```

Timing for **single** object:

```
time aws s3 cp readme.md s3://dateneimer/bash/readme.md
upload: ./readme.md to s3://dateneimer/bash/readme.md
aws s3 cp readme.md s3://dateneimer/bash/readme.md
0,54s user 0,11s system 80% cpu 0,809 total
```

1 Bash 30k

```
Start 00.000
Do 18 Mai 2023 11:32:22 CEST
...
Start 31.000
Do 18 Mai 2023 17:56:58 CEST
```

time > 6 hours for 30k objects

- Slow python startup each time
- App waits for response

2 GO, single goroutine

```
1:      for i := 1; i < 30000; i++ {
2:              key := fmt.Sprintf("single/test-go-%d.md", i)
3:
4:              Put(&key, file, &BUCKET)
5:
6:              log.Println("File: ", key)
7:
8:      }
```

```
go run main.go
12,13s user 8,22s system 1% cpu 22:45,49 total
```

3 GO, 100 goroutines

```
go run main.go
28,26s user 13,02s system 45% cpu 1:31,49 total
```

compare to single goroutine

```
go run main.go
12,13s user 8,22s system 1% cpu 22:45,49 total
```

`aws-go-sdk-v2/s3/manyfiles/main.go`

4 GO, 100 goroutines on EC2

- Create linux x86 executable

```
GOOS=linux GOARCH=amd64 go build -o dist/linux/create30k
```

Upload to cloudshell

```
[cloudshell-user@ip-10-4-19-214 ~]$ chmod u+x create30k
[cloudshell-user@ip-10-4-19-214 ~]$ export BUCKET=dateneimer
[cloudshell-user@ip-10-4-19-214 ~]$ echo "lorem ipso" >readme.md
[cloudshell-user@ip-10-4-19-214 ~]$ time ./create30k
...
real    1m22.495s
user    0m34.314s
sys     0m6.834s
```

Compare single goroutine to 100 goroutines

local: 8 core i9 macbook pro

Local/Bash/AWS cli

- time > 6 hours

Local/single goroutine

- `12,13s user 8,22s system 1% cpu 22:45,49 total`

Cloudshell/100 goroutines

- 1 core

```
real    1m22.495s
user    0m34.314s
sys     0m6.834s
```

Wrap up

- A bash loop with `aws s3 cp` : 6 hours
- A GO program with a single goroutine : 22 minutes
- A GO program with 100 goroutines : 1,5 minute
- A GO program with 100 goroutines on AWS cloudshell: 1,5 minutes