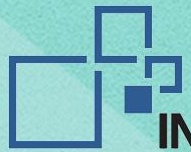
Concurrent Optimization Methods Using GO

PRESENTED BY



INDEX EXCHANGE®

June 5 (5:30-8:00pm)

STC 0010

Gotta GO Fast

Concurrent Optimization Methods Using Go

Brodie Roberts

June 5, 2018

Get The Code

github.com/indexexchange/GoOptimizationMethods

Intro to Go

Intro to Go

- Compiled, strongly-typed language (C style)
- Concurrency primitives
- Run-time garbage collection
- Small instruction set
- Easy to learn:
  - tour.golang.org

Intro to Go: Basics

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello Hack The 6ix!")
    var a int = 0
    var b int = 1

    for i := 0; i < 20; i++ {
        b, a = a+b, b
        fmt.Printf("fib(%d) = %d\n", i+1, a)
    }
}
```

Intro to Go: Concurrency

- Go concurrency primitives:
  - Goroutines
  - Channels

Intro to Go: Goroutines

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    // Goroutine
    go work()

    // Regular function call
    work()
}

func work() {
    time.Sleep(time.Second)
    fmt.Println("Did some work!")
}
```

Intro to Go: Channels

```go
package main

import (
    "fmt"
    "time"
)

func main() {
    // Make a channel
    var done chan bool = make(chan bool)

    // Launch slow goroutine
    go slow(done)

    // Wait for it to finish
    var result bool = <- done
}

func slow(done chan bool) {
    time.Sleep(time.Second)
    done <- true
}
```

Intro to Go: Channels

```go
package main

import "fmt"

func main() {
    // Make a channel
    var data chan int = make(chan int)

    // Launch producer goroutine
    go func() {
        for i := 0; i < 10; i++ {
            data <- i
        }
        close(data) // Close the channel
    }()

    // Loop over channel output
    for i := range data {
        fmt.Printf("Got data: %d\n", i)
    }
}
```

Sample Problem:

Counting Unique Views

Sample Problem

- Millions of users visit our site every day
- We want to count unique users and visits per unique user
- Sample input:

Timestamp	User ID	IP	OS	Browser
2018-06-05 06:00:00	331	192.168.0.12	Windows	Chrome
2018-06-05 06:00:00	438	192.168.0.20	Mac OS X	Safari
2018-06-05 06:00:00	0	192.168.0.135	Android	Chrome
2018-06-05 06:00:00	0	192.168.0.51	Windows	IE

Sample Problem

- Logged in users are grouped by User ID
- Users with no account are grouped by device info
- Sample output:

Count of views	User ID	IP	OS	Browser
25	331			
168	438			
19	0	192.168.0.135	Android	Chrome
824	0	192.168.0.51	Windows	IE

Benchmarking 101

Benchmarking 101

- Repeatable benchmark test
  - Big enough to not give artifacts
- Consistent Environment
- Have a baseline
  - Reasonable upper and lower bounds
- Understand the code

Repeatable Test

Demo

Baseline: Bash

Demo

Baseline

Input	Output	Bash (lower)	Bash (upper)
10M	4783	0.2s	51s
300M	7399	5.5s	26m 10s

Go Solution

Go Solution: Pipeline Model

```
/*
    Structure:
        list() -filePaths-> read() -rawData-> parse() -parsedData-> aggregate() -processedData-> write()
*/
func processDataFolder(folder string) {
    timer := time.Now()

    var filePaths = make(chan string)     // Carries paths of input files
    var rawData = make(chan string)       // Carries raw input lines
    var parsedData = make(chan *Visit)    // Carries parsed Visit structs
    var processedData = make(chan *Visit) // Carries aggregated Visit structs

    go list(folder, filePaths)

    go read(filePaths, rawData)

    go parse(rawData, parsedData)

    go aggregate(parsedData, processedData)

    write(processedData)

    fmt.Printf("Processing folder %s took %dms (%v)\n", folder, MillisecondsSince(timer), time.Since(timer))
}
```

Go Solution: Read Step

```
func read(filePaths chan string, out chan string) {
    defer close(out)
    counter, timer := 0, time.Now()

    for filePath := range filePaths {
        file, err := os.Open(filePath)
        fail(err)

        scanner := bufio.NewScanner(file)
        for scanner.Scan() {
            out <- scanner.Text() // Send data on the output channel
        }
        fail(scanner.Err())

        file.Close()
        counter++
    }
    fmt.Printf("\tRead %d files in %dms\n", counter, MillisecondsSince(timer))
}
```

Go Solution: Parse Step

```
const INPUT_SEP = "\t"
const INPUT_FIELDS = 5
func parse(in chan string, out chan *Visit) {
    defer close(out)
    counter, timer := 0, time.Now()

    for line := range in {
        parts := strings.Split(line, INPUT_SEP)
        if len(parts) != INPUT_FIELDS {
            fail(fmt.Errorf("Wrong number of fields in line: %s", line))
        }
        userID, err := strconv.ParseUint(parts[1], 10, 32)
        fail(err)

        doBusyWork() // Real parsing would take more computing than this

        out <- NewVisit(uint32(userID), parts[2], parts[3], parts[4])
        counter++
    }
    fmt.Printf("\tParsed %d lines in %dms\n", counter, MillisecondsSince(timer))
}
```

```
type Visit struct {
    UserID  uint32
    IP      string
    OS      string
    Browser string
    Count   uint32
}
```

Go Solution: Aggregate Step

```
func aggregate(in chan *Visit, out chan *Visit) {
    defer close(out)
    var hash = make(map[string]*Visit)

    for visit := range in {
        var key string = visit.GetKey()

        cachedVisit, exists := hash[key]
        if exists {
            cachedVisit.Count += visit.Count
        } else {
            hash[key] = visit
        }
    }

    counter, timer := 0, time.Now()
    for _, visit := range hash {
        out <- visit
        counter++
    }
    fmt.Printf("\tWrote %d lines in %dms\n", counter, MillisecondsSince(timer))
}
```

Go Solution: Write Step

```
func write(in chan *Visit) {
    file, err := os.OpenFile(OUTPUT_FILE, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0755)
    fail(err)

    for visit := range in {
        fmt.Fprintf(file, "%s\n", visit)
    }
}
```

Go Solution: Pipeline Model

```
/*
    Structure:
        list() -filePaths-> read() -rawData-> parse() -parsedData-> aggregate() -processedData-> write()
*/
func processDataFolder(folder string) {
    timer := time.Now()

    var filePaths = make(chan string)     // Carries paths of input files
    var rawData = make(chan string)       // Carries raw input lines
    var parsedData = make(chan *Visit)    // Carries parsed Visit structs
    var processedData = make(chan *Visit) // Carries aggregated Visit structs

    go list(folder, filePaths)

    go read(filePaths, rawData)

    go parse(rawData, parsedData)

    go aggregate(parsedData, processedData)

    write(processedData)

    fmt.Printf("Processing folder %s took %dms (%v)\n", folder, MillisecondsSince(timer), time.Since(timer))
}
```

Go vs Baseline

Demo

Go vs Baseline

Input	Output	Bash (lower)	Bash (upper)	Go (quick)	Go V1
10M	4783	0.2s	51s	29s	46s
300M	7399	5.5s	26m 10s	14m 40s	24m 10s

Strip Down Testing

Demo

Version 2

Input	V1	V2
10M	46s	26s
300M	24m 10s	12m 40s

Version 3

Input	V1	V2	V3
10M	46s	26s	23s
300M	24m 10s	12m 40s	10m 10s

Version 4

Input	V1	V2	V3	V4
10M	46s	26s	23s	6.4s
300M	24m 10s	12m 40s	10m 10s	3m 2s

Version 5

Input	V1	V2	V3	V4	V5
10M	46s	26s	23s	6.4s	3.1s
300M	24m 10s	12m 40s	10m 10s	3m 2s	1m 51s

Next Steps...

FIN

Questions?