**Отчет об увеличении производительности функции парсинга .txt файла**

1. Исходные данные:

1.1 Неоптимизированная функция SlowSearch парсинга .txt файла имеет следующие параметры производительности:

goos: windows

goarch: amd64

pkg: hw3

cpu: Intel(R) Core(TM) i7-**8550**U CPU @ **1.80**GHz

BenchmarkSlow-**8** **42** **29373357** ns/op **20208563** B/op **189846** allocs/op

PASS

ok hw3 **2.763**s

1.2 Параметры производительности некой оптимизированной функции Solution:

BenchmarkSolution-**8** **500** **2782432** ns/op **559910** B/op **10422** allocs/op

2. Задача – оптимизировать\* функцию таким образом, чтобы один из параметров (B/op, allocs/op) был лучше параметра функции Solution и еще один не хуже параметра указанной функции с учетом 20%, например,

allocs/op < allocs/op\*1,2 (Solution) -> 10422 \* 1,2 = 12506.

\* Без использования горутин и sync.Pool.

3. Анализ параметров производительности.

3.1 В начале осуществлен вывод информации о длительности операций исходной функции профилировщиком cpu.out. Наиболее ресурсоемкие операции:

. **10**ms **27**: r := regexp.MustCompile("@")

. **70**ms **36**: user := make(**map**[**string**]**interface**{})

. **1.47**s **38**: err := json.Unmarshal([]byte(line), &user)

. **10**ms **50**: browsers, ok := user["browsers"].([]**interface**{})

. **1.65**s **62**: **if** ok, err := regexp.MatchString("Android", browser); ok && err == **nil** { ... }

. **1.31**s **84**: **if** ok, err := regexp.MatchString("MSIE", browser); ok && err == **nil** { ... }

3.2 Следующим шагом осуществлен вывод информации о затрачиваемой памяти операциями с помощью профилировщика mem.out. Наиболее ресурсоемкие операции:

. **28.04**MB **22**: fileContents, err := ioutil.ReadAll(file)

**4.92**MB **5.05**MB **32**: lines := strings.Split(string(fileContents), "\n")

**4.53**MB **16.83**MB **38**: err := json.Unmarshal([]byte(line), &user)

. **63.54**MB **62**: **if** ok, err := regexp.MatchString("Android", browser); ok && err == **nil** { .. }

. **42.62**MB **84**: **if** ok, err := regexp.MatchString("MSIE", browser); ok && err == **nil** { ... }

3.3 Далее была выведена информация по наибольшему количеству аллокаций по операциям:

**16000** **16000** **36**: user := make(**map**[**string**]**interface**{})

**8001** **362811** **38**: err := json.Unmarshal([]byte(line), &user)

. **544248** **62**: **if** ok, err := regexp.MatchString("Android", browser); ok && err == **nil** { ... }

. **480170** **84**: **if** ok, err := regexp.MatchString("MSIE", browser); ok && err == **nil** { ... }

4. При анализе данных из п. 3 были приняты и реализованы следующие решения по улучшению функции:

* Функции пакета регулярных выражений regexp заменены на функции стандартного пакета strings для поиска подстроки в строке и замены элемента в строке;
* Для того чтобы не загружать сразу все данные из текстового файла был использован Scanner из пакета bufio, что позволило загружать данные из файла поэтапно;
* Был изменен анмаршелинг данных в структуру вместо мапы;
* Функция анмаршелинга данных была применена из пакета easyjson;
* Было уменьшено количество циклов в процессе выполнения функции, все операции осуществлены в одном цикле Scanner`а, который проходит по всему документу;
* У созданной структуры User поля имеют тип string, что позволило уйти от конвертации типов (interface -> string);
* Информация о пользователе (который использует Android и MSIE) сразу выводится в конце итерации цикла.

4. Результаты реализованных улучшений:

Производительность исходной функции SlowSearch:

goos: windows

goarch: amd64

pkg: hw3

cpu: Intel(R) Core(TM) i7-**8550**U CPU @ **1.80**GHz

BenchmarkSlow-**8** **42** **29373357** ns/op **20208563** B/op **189846** allocs/op

PASS

ok hw3 **2.763**s

Производительность модернизированной функции FastSearch:

goos: windows

goarch: amd64

pkg: hw3

cpu: Intel(R) Core(TM) i7-**8550**U CPU @ **1.80**GHz

BenchmarkFast-**8** **632** **1780900** ns/op **482893** B/op **6483** allocs/op

PASS

ok hw3 **2.896**s

Производительность функции Solution:

BenchmarkSolution-**8** **500** **2782432** ns/op **559910** B/op **10422** allocs/op

Из приведенных значений видно, что число аллокаций на операцию:

**6484 allocs/op (FastSearch) < 10422 allocs/op (Solution)**

**482893 B/op (FastSearch) < 559910 B/op (Solution)**

Таким образом, задача оптимизации функции SlowSearch считается выполненной. Листинги функций SlowSearch и FastSearch приведены в Приложении 1 и Приложении 2 соответственно.

Приложение 1. Листинг функции SlowSearch.

**func** SlowSearch(out io.Writer) {

file, err := os.Open(filePath)

**if** err != **nil** {

panic(err)

}

fileContents, err := ioutil.ReadAll(file)

**if** err != **nil** {

panic(err)

}

r := regexp.MustCompile("@")

seenBrowsers := []**string**{}

uniqueBrowsers := **0**

foundUsers := ""

lines := strings.Split(string(fileContents), "\n")

users := make([]**map**[**string**]**interface**{}, **0**)

**for** \_, line := **range** lines {

user := make(**map**[**string**]**interface**{})

err := json.Unmarshal([]byte(line), &user)

**if** err != **nil** {

panic(err)

}

users = append(users, user)

}

**for** i, user := **range** users {

isAndroid := **false**

isMSIE := **false**

browsers, ok := user["browsers"].([]**interface**{})

**if** !ok {

**continue**

}

**for** \_, browserRaw := **range** browsers {

browser, ok := browserRaw.(**string**)

**if** !ok {

**continue**

}

**if** ok, err := regexp.MatchString("Android", browser); ok && err == **nil** {

isAndroid = **true**

notSeenBefore := **true**

**for** \_, item := **range** seenBrowsers {

**if** item == browser {

notSeenBefore = **false**

}

}

**if** notSeenBefore {

seenBrowsers = append(seenBrowsers, browser)

uniqueBrowsers++

}

}

}

**for** \_, browserRaw := **range** browsers {

browser, ok := browserRaw.(**string**)

**if** !ok {

**continue**

}

**if** ok, err := regexp.MatchString("MSIE", browser); ok && err == **nil** {

isMSIE = **true**

notSeenBefore := **true**

**for** \_, item := **range** seenBrowsers {

**if** item == browser {

notSeenBefore = **false**

}

}

**if** notSeenBefore {

seenBrowsers = append(seenBrowsers, browser)

uniqueBrowsers++

}

}

}

**if** !(isAndroid && isMSIE) {

**continue**

}

email := r.ReplaceAllString(user["email"].(**string**), " [at] ")

foundUsers += fmt.Sprintf("[%d] %s <%s>\n", i, user["name"], email)

}

fmt.Fprintln(out, "found users:\n"+foundUsers)

fmt.Fprintln(out, "Total unique browsers", len(seenBrowsers))

}

Приложение 2. Листинг функции FastSearch.

**func** FastSearch(out io.Writer) {

file, err := os.Open(filePath)

**if** err != **nil** {

panic(err)

}

**defer** file.Close()

seenBrowsers := []**string**{}

uniqueBrowsers := **0**

i := -**1**

scanner := bufio.NewScanner(file)

user := User{}

fmt.Fprintln(out, "found users:")

**for** scanner.Scan() {

i++

err := easyjson.Unmarshal(scanner.Bytes(), &user)

**if** err != **nil** {

panic(err)

}

isAndroid := **false**

isMSIE := **false**

browsers := user.Browsers

**for** \_, browser := **range** browsers {

**if** ok := strings.Contains(browser, "Android"); ok {

isAndroid = **true**

notSeenBefore := **true**

**for** \_, item := **range** seenBrowsers {

**if** item == browser {

notSeenBefore = **false**

}

}

**if** notSeenBefore {

seenBrowsers = append(seenBrowsers, browser)

uniqueBrowsers++

}

}

**if** ok := strings.Contains(browser, "MSIE"); ok {

isMSIE = **true**

notSeenBefore := **true**

**for** \_, item := **range** seenBrowsers {

**if** item == browser {

notSeenBefore = **false**

}

}

**if** notSeenBefore {

seenBrowsers = append(seenBrowsers, browser)

uniqueBrowsers++

}

}

}

**if** !(isAndroid && isMSIE) {

**continue**

}

email := strings.ReplaceAll(user.Email, "@", " [at] ")

foundUsers := fmt.Sprintf("[%d] %s <%s>", i, user.Name, email)

fmt.Fprintln(out, foundUsers)

}

fmt.Fprintln(out, "\nTotal unique browsers", len(seenBrowsers))

}