

Informe Proyecto Final PFC 2024-I

Solución al problema:

Para encontrar los mejores itinerarios de vuelo desde un aeropuerto de origen a un aeropuerto de destino, considerando múltiples criterios de optimización:

1. **Tiempo total de viaje:** Incluye el tiempo de vuelo y el tiempo de espera entre conexiones.
2. **Número de escalas:** Minimizar el número de conexiones entre vuelos.
3. **Tiempo de vuelo:** Solo el tiempo en el aire, excluyendo el tiempo de espera en las escalas.
4. **Horario de llegada:** Asegurar que el itinerario llegue antes de una hora específica.

La solución se implementa en la clase `FlightSchedulesPar`, que utiliza técnicas de paralelización para mejorar la eficiencia en la búsqueda de itinerarios de vuelo.

La solución planteada se basa en la paralelización de tareas y en aplicar límites de profundidad como criterio para hallar cuando es beneficioso realizar la paralelización y cuando la secuencial. Es decir, si la profundidad en el método de `schedule` es menor que `maxDepth`, entonces la búsqueda se realiza de manera paralela. En el caso paralelo, para cada vuelo que sale del aeropuerto de origen, se crea una tarea paralela que busca itinerarios desde el aeropuerto de destino de ese vuelo hasta el destino final. Estas tareas se crean utilizando la función `task` y se ejecutan en paralelo. Los resultados de estas tareas se combinan utilizando la función `flatMap` y la función `join`. Además, la creación de los itinerarios a partir de las listas de vuelos encontradas también se realiza en paralelo. Para cada lista de vuelos, se crea una tarea que calcula el tiempo total de vuelo, el número de escalas y el tiempo de vuelo, y crea un objeto `Itinerario`. Estas tareas se ejecutan en paralelo y los resultados se combinan utilizando la función `map` y la función `join`. Por lo tanto, la paralelización en `FlightSchedulesPar` se realiza a nivel de búsqueda de itinerarios y a nivel de creación de itinerarios.

Jean Carlos Lerma Rojas 2259305

Juan Camilo Garcia Saenz 2259416

Jhojan Serna Henao 2259504

```
TestSchedulesOutputTimePar.scala FlightSchedulesPar.scala ScheduleBenchmark.scala
class FlightSchedulesPar {
  private val maxDepthGlobal = 4 // es el máximo de escalas que se pueden hacer

  def schedules(flights: List[Vuelo], airports: List[Aeropuerto], maxDepth: Int)(origen: String, destination: String): List[Itinerario] = {
    def findSchedules(origen: String, destination: String, visited: Set[String], depth: Int): List[List[Vuelo]] = {
      if (origen == destination) List(List())
      else {
        val outboundFlights = flights.filter(f => f.AirportOrigin == origen && !visited.contains(f.AirportDestination))
        if (depth >= maxDepth) {
          outboundFlights.flatMap(flight => {
            findSchedules(flight.AirportDestination, destination, visited + origen, depth + 1).map(flight :: _) // por cada vuelo que se
          })
        } else {
          val scheduleTasks = outboundFlights.map { flight =>
            task {
              findSchedules(flight.AirportDestination, destination, visited + origen, depth + 1).map(flight :: _)
            }
          }
          scheduleTasks.flatMap(_.join())
        }
      }
    }

    val foundSchedules = findSchedules(origen, destination, Set(), 0)

    foundSchedules.map { flights =>
      task {
        val flightTotal = getFlightTotal(airports)(flights)
        val scales = flights.map(flight => flight.Scales).sum + flights.size - 1
        val flightTime = getFlightTime(airports)(flights)

        Itinerario(flights, flightTotal, scales, flightTime)
      }
    }.map(_.join())
  }
}
```

Comparativas:

benchmark.run(benchmark.benchmarkSchedulesSeq) (benchmark.benchmarkSchedulesPar)

```
class ScheduleBenchmark {
  //val flights: List[Vuelo] = vuelosA1
  val flights: List[Vuelo] = vuelosB1
  //val flights: List[Vuelo] = vuelosC1
  //val flights: List[Vuelo] = vuelosD1

  def benchmarkSchedulesSeq(): Double = {
    val time = withWarmer(new Warmer.Default) measure {
      flightSchedules.schedules(flights, airports)(origen, destination)
    }
    println(s"Tiempo promedio secuencial: ${time.value}")
    time.value
  }

  def benchmarkSchedulesPar(): Double = {
    val time = withWarmer(new Warmer.Default) measure {
      flightSchedulesPar.schedules(flights, airports, 4)(origen, destination)
    }
    println(s"Tiempo promedio paralelo: ${time.value}")
    time.value
  }
}

class ScheduleBenchmark {
  private val flightSchedulesPar = new FlightSchedulesPar()

  private val origen = "DEW"
  private val destination = "ATL"

  val airports: List[Aeropuerto] = aeropuertosCurso
  //val flights: List[Vuelo] = vuelosA1
  val flights: List[Vuelo] = vuelosB1
  //val flights: List[Vuelo] = vuelosC1
  //val flights: List[Vuelo] = vuelosD1

  def benchmarkSchedulesSeq(): Double = {
    val time = withWarmer(new Warmer.Default) measure {
      flightSchedules.schedules(flights, airports)(origen, destination)
    }
    println(s"Tiempo promedio secuencial: ${time.value}")
    time.value
  }

  def benchmarkSchedulesPar(): Double = {
    val time = withWarmer(new Warmer.Default) measure {
      flightSchedulesPar.schedules(flights, airports, 4)(origen, destination)
    }
    println(s"Tiempo promedio paralelo: ${time.value}")
    time.value
  }
}
```

projecto-final-programacion-funcional-concurrente:app [app...]

run

✓ proyecto-final-programacion-funcional-concurrente:app UP-TO-DATE
✓ :app:compileJava
✓ :app:compileScala UP-TO-DATE 54 ms
✓ :app:processResources
✓ :app:classes UP-TO-DATE
✓ :app:App.main() 1 sec, 269 ms

Task :app:compileScala UP-TO-DATE
Task :app:processResources NO-SOURCE
Task :app:classes UP-TO-DATE
Task :app:App.main()
Unable to create a system terminal
Tiempo promedio secuencial: 0.7973
Tiempo promedio paralelo: 0.7337
Fin de la ejecución

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

Jean Carlos Lerma Rojas 2259305
Juan Camilo Garcia Saenz 2259416
Jhojan Serna Henao 2259504

Tamaño de los datos de Vuelos	Solución Secuencial (ms)	Solución Paralela (ms)
15	0.2685	0.2323
40	0.0383	0.0203
100	49.5613	19.4102

```
benchmark.run(benchmark.benchmarkSchedulesTimeSeq) (benchmark.benchmarkSchedulesTimePar)
```

The screenshot displays an IDE with two Scala files: `ScheduleBenchmark.scala` and `TestSchedulesOutput.scala`. The `ScheduleBenchmark` class contains two methods: `benchmarkSchedulesTimeSeq` and `benchmarkSchedulesTimePar`. The `benchmarkSchedulesTimeSeq` method uses `FlightSchedules.schedulesTime` to calculate the average sequential time, while `benchmarkSchedulesTimePar` uses `FlightSchedulesPar.schedulesTime` for parallel execution. The `TestSchedulesOutput` class contains a `main` method that runs the benchmarks. The output of the benchmarks is shown in the console, indicating the average sequential time is 0.9442 and the average parallel time is 2.6202. A warning message at the bottom states: "Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0. You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins."

Tamaño de los datos de Vuelos	Solución Secuencial (ms)	Solución Paralela (ms)
15	0.4559	0.2151
40	0.9537	1.0748
100	44.5055	29.827

```
benchmark.run(benchmark.benchmarkSchedulesScalesSeq) (benchmark.benchmarkSchedulesScalesPar)
```

Jean Carlos Lerma Rojas 2259305
Juan Camilo Garcia Saenz 2259416
Jhojan Serna Henao 2259504

The screenshot shows an IDE with two Scala files. The left file, `ScheduleBenchmark.scala`, contains the following code:

```
class ScheduleBenchmark {  
  def benchmarkSchedulesTimePar(): Double = {  
    time.value  
  }  
  
  def benchmarkSchedulesScalesSeq(): Double = {  
    val time = withWarmer(new Warmer.Default) measure {  
      flightSchedules.schedulesScales(flights, airports)(origen, destination)  
    }  
    println(s"Tiempo promedio secuencial: ${time.value}")  
    time.value  
  }  
  
  def benchmarkSchedulesScalesPar(): Double = {  
    val time = withWarmer(new Warmer.Default) measure {  
      flightSchedulesPar.schedulesScales(flights, airports)(origen, destination)  
    }  
    println(s"Tiempo promedio paralelo: ${time.value}")  
    time.value  
  }  
}
```

The right file, `ScheduleBenchmark.scala`, contains the following code:

```
class ScheduleBenchmark {  
  private val flightSchedulesPar = new FlightSchedulesPar()  
  
  private val origen = "HOU"  
  private val destination = "BNA"  
  
  val airports: List[Aeropuerto] = aeropuertosCurso  
  val flights: List[Vuelo] = vuelosA1  
  //val flights: List[Vuelo] = vuelosB1  
  //val flights: List[Vuelo] = vuelosC1  
  //val flights: List[Vuelo] = vuelosD1  
  
  def benchmarkSchedulesSeq(): Double = {  
    val time = withWarmer(new Warmer.Default) measure {  
      flightSchedules.schedules(flights, airports)(origen, destination)  
    }  
    println(s"Tiempo promedio secuencial: ${time.value}")  
    time.value  
  }  
  
  def benchmarkSchedulesPar(): Double = {  
    val time = withWarmer(new Warmer.Default) measure {  
      flightSchedulesPar.schedules(flights, airports)(origen, destination)  
    }  
    println(s"Tiempo promedio paralelo: ${time.value}")  
    time.value  
  }  
}
```

The bottom panel shows the execution output for the project `proyecto-final-programacion-funcional-concurrente`:

```
> Task :app:compileJava UP-TO-DATE  
> Task :app:compileScala UP-TO-DATE  
> Task :app:processResources UP-TO-DATE  
> Task :app:classes UP-TO-DATE  
> Task :app:App.main() UP-TO-DATE  
Unable to create a system terminal  
Tiempo promedio secuencial: 0.4483  
Tiempo promedio paralelo: 0.6417  
Fin de la ejecución  
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.  
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
```

Tamaño de los datos de Vuelos	Solución Secuencial (ms)	Solución Paralela (ms)
15	0.4483	0.6417
40	1.4903	1.1823
100	19.9988	9.6963

```
benchmark.run(benchmark.benchmarkSchedulesTimeLandSeq)(benchmark.benchmarkSchedulesTimeLandPar)
```

Jean Carlos Lerma Rojas 2259305
Juan Camilo Garcia Saenz 2259416
Jhojan Serna Henao 2259504

7class ScheduleBenchmark { Jean Carlos Lerma Rojas +1*
66}
67
68def benchmarkSchedulesTimeLandSeq(): Double = { Jean Carlos Lerma Rojas
69val time = withWarmer(new Warmer.Default) measure {
70flightSchedules.schedulesTimeLand(flights, airports)(origen, destination)
71}
72println(s"Tiempo promedio secuencial: \${time.value}")
73time.value
74}
75
76def benchmarkSchedulesTimeLandPar(): Double = { Jean Carlos Lerma Rojas
77val time = withWarmer(new Warmer.Default) measure {
78flightSchedulesPar.schedulesTimeLand(flights, airports)(origen, destination)
79}
80println(s"Tiempo promedio paralelo: \${time.value}")
81time.value
82}
83
84def benchmarkSchedulesOutputTimeSeq(): Double = { Jean Carlos Lerma Rojas
85val time = withWarmer(new Warmer.Default) measure {
86flightSchedules.schedulesOutputTimeSeq(flights, airports)(origen, destination)
87}
88println(s"Tiempo promedio secuencial: \${time.value}")
89time.value
90}
91
92def benchmarkSchedulesOutputTimePar(): Double = { Jean Carlos Lerma Rojas
93val time = withWarmer(new Warmer.Default) measure {
94flightSchedulesPar.schedulesOutputTimePar(flights, airports)(origen, destination)
95}
96println(s"Tiempo promedio paralelo: \${time.value}")
97time.value
98}
99}

7class ScheduleBenchmark { Jean Carlos Lerma Rojas +1*
9private val flightSchedulesPar = new FlightSchedulesPar()
10
11private val origen = "HOU"
12private val destination = "BNA"
13
14val airports: List[Aeropuerto] = aeropuertosCurso
15val flights: List[Vuelo] = vuelosA1
16//val flights: List[Vuelo] = vuelosB1
17//val flights: List[Vuelo] = vuelosC1
18//val flights: List[Vuelo] = vuelosD1
19
20def benchmarkSchedulesSeq(): Double = { Jean Carlos Lerma Rojas
21val time = withWarmer(new Warmer.Default) measure {
22flightSchedules.schedules(flights, airports)(origen, destination)
23}
24println(s"Tiempo promedio secuencial: \${time.value}")
25time.value
26}
27
28def benchmarkSchedulesPar(): Double = { Jean Carlos Lerma Rojas
29val time = withWarmer(new Warmer.Default) measure {
30flightSchedulesPar.schedules(flights, airports)(origen, destination)
31}
32println(s"Tiempo promedio paralelo: \${time.value}")
33time.value
34}
35}

Runprojecto-final-programacion-funcional-concurrente.app [app... x
ScheduleBenchmarkbenchmarkSchedulesTimePar()ScheduleBenchmark

projecto-final-programacion-funcional-concurrente1 sec, 518 ms
app.compileJava
app.compileScala UP-TO-DATE63 ms
app.processResources
app.classes UP-TO-DATE1 ms
app.App.main()1 sec, 152 ms

> Task :app:App.main()
Unable to create a system terminal
Tiempo promedio secuencial: 0.2457
Tiempo promedio paralelo: 0.5824
Fin de la ejecución

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

Tamaño de los datos de Vuelos	Solución Secuencial (ms)	Solución Paralela (ms)
15	0.2457	0.5824
40	0.043	0.0467
100	56.3106	28.5022

```
benchmark.run(benchmark.benchmarkSchedulesOutputTimeSeq) (benchmark.benchmarkSchedulesOutputTimePar)
```

The screenshot shows an IDE with two Scala files. The left file, `ScheduleBenchmark.scala`, contains benchmarking functions: `benchmarkSchedulesTimeLandPar()`, `benchmarkSchedulesOutputTimeSeq()`, and `benchmarkSchedulesOutputTimePar()`. The right file, `ScheduleBenchmark.scala`, contains the `ScheduleBenchmark` class with attributes like `flightSchedulesPar`, `origen`, `destination`, and `airports`, and methods `benchmarkSchedulesSeq()` and `benchmarkSchedulesPar()`. The bottom panel shows the execution output for `proyecto-final-programacion-funcional-concurrente:app`, displaying sequential and parallel execution times for different data sizes (15, 40, 100) and a deprecation warning for Gradle 9.0.

Tamaño de los datos de Vuelos	Solución Secuencial (ms)	Solución Paralela (ms)
15	0.4843	0.4615
40	0.4029	0.526
100	22.0562	14.0076

Conclusiones:

Al hacer las comparaciones anteriores podemos observar que la versión paralela tuvo una mejor eficiencia que la versión secuencial, podemos decir que la implementación de la versión paralela fue la más apropiada ya que para tamaños de datos muy grandes la eficiencia del paralelo cada vez se vuelve más notoria. Incluso con datos pequeños la versión paralela tuvo un rendimiento similar o mejor en comparación con la versión secuencial.

Al observar las comparaciones para datos pequeños nos damos cuenta de que el costo computacional del paralelo fue mínimo ya que los tiempos de ejecución de las dos versiones fueron muy similares. o la versión paralela fue más rápida

Podemos concluir que la versión paralela tuvo un mejor rendimiento en comparación con la secuencial siendo esta versión la más eficiente y se recomienda la paralela sobre todo para tamaños de datos muy grandes, aunque como consideración en ciertos casos o funciones los

Jean Carlos Lerma Rojas 2259305

Juan Camilo Garcia Saenz 2259416

Jhojan Serna Henao 2259504

resultados fueron negativos para la versión paralela y esto pudo deberse a la complejidad de sus operaciones internas o factores externos del sistema.