Universidad Nacional de San Agustín de Arequipa

Computación de Alto Desempeño

Centro de datos de Telefónica

Alumnos:
Vicente Machaca Arceda
Enzo Velásquez Lobatón
Carlo Corrales Delgado
Oscar Ramirez Valdez

Docente:
PhD. Alvaro Mamani Aliaga

19 de noviembre de 2021

Índice

1. Introducción

2. Marco teórico

2.1. Secuenciamiento de DNA

2.2. Secuenciemiento *Next-Genration*

2.3. Secuenciamiento *Single-cell RNA*

2.4. NCBI y SRA toolkit

3. Propuesta

En esta sección detallaremos el objetivo de la propuesta y los detalles técnicos para el desarrollo de la misma.

3.1. Plataforma implementada

En este trabajo se propone el desarrollo de una plataforma para el análisis de secuencias scRNA. Por ejemplo en la Figura 1, se presente las fases tradicionales de una análisis de secuencias de ADN (Next-generation). En este caso, vemos como se realiza el secuenciamiento de ADN/ARN, obteniendo miles y millones de lecturas cortas de la cadena de ADN. Entonces la idea es analizar estas secuencias con el fin de saber que genes estan activos y como influye esto en el fenotipo de la especie. Por ejemplo, este análisis se suele realizar sobre celulas de tejido tumoral y se compara con otro experimiento que tomo muestras de celulas sanas, luego de una comparación se puede determinar cuales son los genes activos o inactivos según el tipo de tumor y enfermedad, esto nos ayuda a comprender mejor las enfermedades y mas aún detectarlas en fases tempranas.

Como se menciono antes, la propuesta se base en el desarrollo de una plataforma que permita un análisis de secuencias de ADN sobre un sistema distribuido. Realizar todo el análisis es costoso, debido a eso **en esta etapa se desarrollo en la verificación de calidad de las secuencias** de entrada. En proyectos futuros se completará la plataforma con mas funcionalidades.

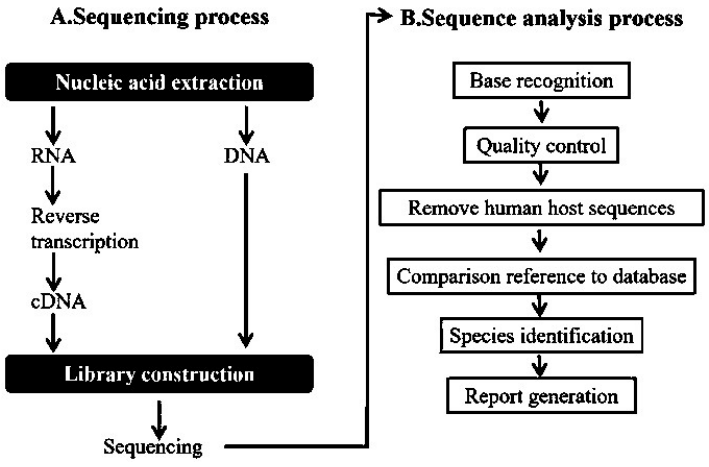

Figura 1: Phases comunes realizadas en un análisis de secuencias de ADN (Next-generation).

3.2. Herramientas utilizadas

Para el desarrollo de la propuesta se ha utilizado las herramientas de la Tabla 1. Se escogio, Spark, debido a su versatilidad y la gestión de RDDs que permiten un desarrollo distribuido facil de implementar. Luego, se opto por utilizar Pyspark, porque la evaluación de los *scripts* pueden hacerse desde el interprete de Python y incluso en Google Colab, esto es una ventaja porque reduce el tiempo de desarrollo.

Tabla 1: Herramientas utilizadas para el proyecto

Herramienta	**Version**
Spark	3.2.0
Python	3.8.10
Pyspark	3.2.0

3.3. Hardware

En cuanto al hardware utilizado en el sistema distribuido, este es detallado en la Tabla 2. Debido a la falta de recursos, solo se utilizo dos computadoras, una es un minicomputador Asus y una laptop Asus Taichi (ver Figura 2)

Tabla 2: Computadoras utilizadas en el sistema distribuído

Nombre de PC	**Especificaciones**
Desktop Asus	Procesador i7 de séptima generación y 8GB de memoria RAM. Sistema operativo Linux.
Laptop Asus	Procesador i5 de quinta generación y 4GB de memoria RAM. Sistema operativo Linux.

Figura 2: Computadoras utilizadas en el sistema distribuído.

3.4. Funcionalidades

Las funcionalidades de la propuesta son:

- Conteo de la cantidad de secuencias.
- Conteo total de las bases nitrogenadas.

- Computo de la longitud de todas las secuencias.
- Computo del promedio de las longitudes de las secuenias.
- Computo de la ocurrencia de cada base nitrogenada.
- Análisis de contenido por base.

3.5. Implementación

El código fuente de la propuesta está en esté repositorio. Lineas abajo, presentamos un extracto del archivo principal.

```
# este codigo, realiza un pequenio analisis a vcarias secuencias scRNA,
# se considero a ERR3014700. Se usa SRA toolkit para genera el archivo fasta.

from __future__ import print_function
from functools import wraps
import pyspark as spark
from pyspark import SparkConf
import time
from operator import add
import os
from subprocess import STDOUT, check_call, check_output


class Fastq:
   def __init__(self, path:str) -> str:
      self.path = path
      self.stop_context()
      self.sc = spark.SparkContext.getOrCreate(conf=self.set_conf())
      self.data = self.sc.textFile(self.path)

   def stop_context(self):
      try:
         self.sc.stop()
         except:
      pass

   def set_conf(self):
      conf = SparkConf().setAppName("App")
      conf = (conf.setMaster('local[*]')
      .set('spark.executor.memory', '4G')
      .set('spark.driver.memory', '16G')
      .set('spark.driver.maxResultSize', '8G'))
      return conf

   def _logging(func):
   @wraps(func)
   def log_print(instance, *args, **kwargs):
      start = time.time()
      res = func(instance, *args, **kwargs)
      print("Finished Executing {} in {}s!".format(func.__name__, time.time() - start))
      return res
      return log_print
```

```
   @_logging
   def get_data(self):
   return self.data

   @_logging
   def count_bases(self):
      seqs = self.extract_seq()
      seqs = seqs.flatMap(lambda line: list(line))
      seqs = seqs.map(lambda c: (c, 1))
      return seqs.reduceByKey(lambda a, b: a+b)#\


   @_logging
   def per_base_seq_content(self):
      seqs = self.extract_seq()
      seqs = seqs.flatMap(lambda line: list(line))
      seqs = seqs.map(lambda c: (c, 1))
      return seqs.reduceByKey(lambda a, b: a+b)#\

   @_logging
   def extract_seq(self):
      return self.data.filter(lambda x: x.isalpha())

   @_logging
   def get_lengths(self):
      seqs = self.extract_seq()
      return seqs.map(lambda x: len(x))

   def extract_qual(self):
      pass

   def extract_meta(self):
      pass


fasta = Fastq('/home/vicente/Documents/sratoolkit.2.11.3-ubuntu64/samples/ERR3014700.fastq')

# show first read
print("fasta head:", fasta.data.take(4))

# show read count
print("Read count:", fasta.data.count())

# extract sequences alone from the fastq file
seqs = fasta.extract_seq()

print("total sequences:", seqs.count())
print("sequences:", seqs.take(4))

# compute read lengths
lens = fasta.get_lengths()

# show the lengths of the first 10 reads
print("sequence lenghts:", lens.take(10))

# get the average read length
```

```
len_sum = lens.reduce(lambda x, y: x+y)
print("sequence mean lenght:", len_sum//lens.count())

# count base occurance
bases = fasta.count_bases()
print("Bases ocurrence:", bases.take(10))

import json
file = open("results.txt", "w")
results = {
'bases': fasta.data.count(),
'total_seqs': seqs.count(),
'seqs_len': lens.take(10),
'seqs_len_mean': len_sum//lens.count(),
'bases_ocurrence': bases.take(10)
}
file.write( json.dumps(results, indent=4) )

#########################################################
# procesamos per base sequence content
#########################################################
seqs = fasta.extract_seq()

# get max, min, mean lenght
lens = fasta.get_lengths()
len_max = lens.reduce(lambda x, y: max(x, y))
len_min = lens.reduce(lambda x, y: min(x, y))
len_sum = lens.reduce(lambda x, y: x+y)
len_mean = len_sum//lens.count()

print(len_max, len_min, len_mean)

import numpy as np

# vectores de las 4 bases, para el conteo
#list_a = fasta.sc.parallelize(np.zeros( len_max ))
#list_c = fasta.sc.parallelize(np.zeros( len_max ))
#list_g = fasta.sc.parallelize(np.zeros( len_max ))
#ist_t = fasta.sc.parallelize(np.zeros( len_max ))

list_a = np.zeros( len_max )
list_c = np.zeros( len_max )
list_g = np.zeros( len_max )
list_t = np.zeros( len_max )

lens = fasta.get_lengths()
lens_np = np.array(lens.collect())

#for i in range( 3 ): # por cada base
for i in range( len_max ):
if i < lens_np[i]:
acc_a = seqs.map(lambda x: x[i] if i < len(x) else 'X' ).filter( lambda x: x=='A').count()
acc_c = seqs.map(lambda x: x[i] if i < len(x) else 'X' ).filter( lambda x: x=='C').count()
acc_g = seqs.map(lambda x: x[i] if i < len(x) else 'X' ).filter( lambda x: x=='G').count()
acc_t = seqs.map(lambda x: x[i] if i < len(x) else 'X' ).filter( lambda x: x=='T').count()
```

```
list_a[i] = acc_a
list_c[i] = acc_c
list_g[i] = acc_g
list_t[i] = acc_t


n_seqs = seqs.count()
list_a = list_a/n_seqs
list_c = list_c/n_seqs
list_g = list_g/n_seqs
list_t = list_t/n_seqs

import matplotlib.pyplot as plt

plt.plot(range(len_max), list_a)
plt.plot(range(len_max), list_c)
plt.plot(range(len_max), list_g)
plt.plot(range(len_max), list_t)
#plt.show()
plt.savefig('per_base_content.png', bbox_inches='tight')
```

4. Experimentos y Resultados

Para evaluar el desempeño de la propuesta se evaluó las secuencias con el código **ERR3014700**, estas fueron descargadas de NCBI. Estás representan secuencias de *Human betaherpesvirus*. El tamaño de estas secuencis es de 260.3 MB, en formato SRA. Luego fueron descomprimidos con la herramienta SRA de NCBI, como resultado se obtuvo las secuencias en formato FastaQ, en este caso el archivo llego a ocupar 590.8 MB.

Luego se levanto el sistema distribuído, se subio el *script* a *spark-submit* y se realizo el análisis. La plataforma retorna como resultado un archivo *json* con detalle del análisis y un gráfico que detalla el análisis de contenido por base. Por ejemplo, para el experimiento mencionado anteriormente, se obtuvo el siguiente *json*.


```
{
   "bases": 1843156,
   "total_seqs": 462393,
   "seqs_len": [
      523,
      600,
      599,
      600,
      599,
      600,
      600,
      529,
      600,
      538
   ],
   "seqs_len_mean": 564,
   "bases_ocurrence": [
      [
         "C",
```

```
        71397200
      ],
      [
        "N",
        4054
      ],
      [
        "D",
        3004
      ],
      [
        "G",
        70502420
      ],
      [
        "A",
        59888213
      ],
      [
        "F",
        16160
      ],
      [
        "T",
        59010438
      ],
      [
        "B",
        203
      ],
      [
        "E",
        4144
      ]
    ]
}
```


Para el caso del análisis de contenido por base, la plataforma generá el grafico de la Figura 3. Según este análisis, comprobamos que las primeras bases tiene un margen de error (las curvas caen y decaen), luego tambien verificamos que hay posiciones donde la Adenina tiene baja presencia (curva de color rojo),

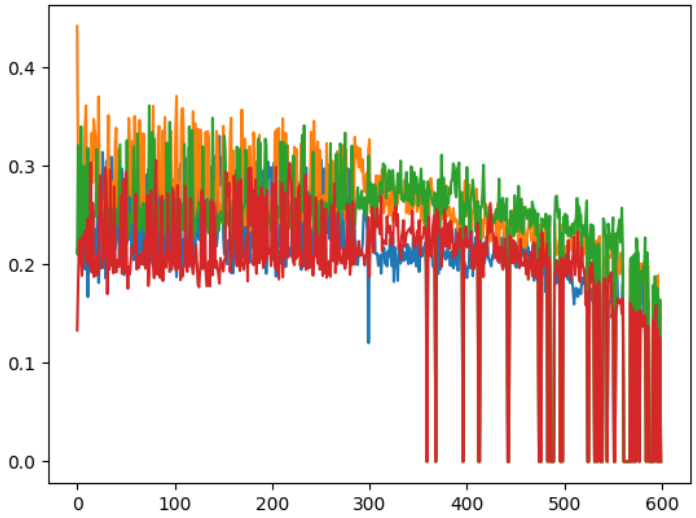

Figura 3: Anĺisis de contenido por base de las secuencias ERR3014700.

5. Conclusiones

Referencias