Jurnal Tugas Akhir Fakultas Informatika Desember-2020

Implementasi Sequence Alignment pada Lingkungan Spark - Hadoop berbasis Single Node

Bernadheta Ayu Putri Mardika[1], Setyorini[2], Siti Amatullah K[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]bernadhetaapm@students.telkomuniversity.ac.id, [2] setyorini@telkomuniversity.ac.id, [3] karimahsiti@telkomuniversity.ac.id

Abstrak

BLAST *(Basic Local Alignment Search Tool)* adalah algoritma yang paling umum dipakai untuk pencocokan sekuensi rantai DNA dari specimen organik, NCBI merupakan salah satu badan yang menangani penyimpanan dataset dan mempergunakan algoritma tersbut, akan tetapi pada umumnya pencacahan akan memakan energi dan waktu komputasi yang cukup besar jika diaplikasikan dalam dataset dengan jumlah besar. melakukan optimasi algoritma yang bersifat universal dapat mengubah standarisasi dalam bioinformatika, oleh karena itu optimasi dilakukan di ranah perangkat keras, pemrosesan dan manajemen database. Pada penelitian ini dikaji penggunaan NCBI BLAST yang dijalankan di platform *Apache Spark* dengan manajemen pemrosesan dataset *Apache Hadoop* sebagai langkah optimasi pemrosesan pencacahan pencarian sekuensi data protein nukleutida dari *specimen organic* dari *query*. Dari hasil kajian ini menunjukkan waktu pemrosesan yang tidak terpengaruh dari banyaknya karakter pencarian, kemudian data sekuens yang acak dan rusak masih tetap dapat dicocokkan dengan baik.

Kata Kunci: *BLAST, Apache spark,Hadoop*

Abstract

BLAST (Basic Local Alignment Search Tool) is the most commonly used algorithm for matching DNA chain sequences of organic specimens, NCBI is one of the institution that handle dataset storage and use the algorithm, but in general the enumeration will take a considerable amount of energy and computational time if applied in a large number of datasets. doing algorithm optimization that is universal can change the standardization in bio informatics, therefore optimization is carried out in hardware, processing and database management. In this study, NCBI BLAST was reviewed by Apache Spark platform with Apache Hadoop dataset processing management as an optimization step to optimize the processing of nucleutide protein data sequential search from organic specimens from queries. From the results of this study shows the unaffected processing time of the number of search characters, then the random and damaged sequence data can still be matched well.

***Keyword*: BLAST, Apache spark,Hadoop**

1. Pendahuluan

1.1 Latar Belakang

Bioinformatika merupakan kajian yang memadukan disiplin biologi molekul, matematika dan teknik informasi (TI). Ilmu in didefinisikan sebagai aplikasi dari alat komputasi dan analisa untuk menangkap dan menginterpretasikan data-data biologi molekul[1]. Kajian baru bioinformatika ini tak lepas dari perkembangan biologi molekul modern yang ditandai dengan kemampuan manusia untuk memahami genom, yaitu cetak biru informasi genetik yang menentukan sifat setiap makhluk hidup yang disandi dalam bentuk pita molekul DNA (asam deoksiribonukleat). Kemampuan untuk memahami dan memanipulasi kode genetik DNA ini sangat didukung oleh TI melalui perangkat keras maupun lunak. Contoh topik utama bidang ini meliputi basis data untuk mengelola informasi biologis, penyejajaran sekuens *(sequence alignment)*, prediksi struktur untuk meramalkan bentuk struktur protein maupun struktur sekunder RNA, analisis filogenetik, dan analisis ekspresi gen[2].

Algoritma Smith-Waterman diperkenalkan oleh Temple F. Smith dan Michael S.Waterman pada tahun 1981 di sebuah Jurnal Molekular Biologi dalam papernya yang berjudul Identification of Common Molecular Subsequences[3]. Algoritma ini digunakan untuk mencari kesamaan yang paling sigifikan (local alignment)dari dua buah rangkaian sekuens gen, dimana setiap sekuens gen dalam rangkaian yang satu dibandingkan dengan sekuens gen rangkaian yang lain berdasarkan kesamaan strukturnya, Sekuens gen dapat

berupa Deoxyribo Nucleic Acid (DNA) atau RiboNucleic Acid (RNA).Sekuens gen tersebut disusun berdasarkan kesamaaan rantai basa hidrogen pembentuknya yaitu Adenin (A), Guanin (G), Cytosin (C), Timin (T) untuk DNA atau A, G, C, Urasil (U) untuk RNA[3], [4] .

Dalam 30 tahun terakhir algoritma Smith-Waterman telah digunakan dalam bidang bioinformatika, termasuk juga *sequence alignment*, *read mapping*, *biological sequence database searching* dan deteksi variasi[5]. Algoritma Smith-Waterman adalah pendekatan berbasis dynamic programming yang digunakan untuk mencari database *sequence* untuk mengidentifikasi kemiripan antara query dan *subject sequence*, algoritma Smith-Waterman juga merupakan metode yang paling akurat untuk local *sequence alignment*, dynamic programming, yaitu metode penyelesaian masalah dari suatu permasalahan dengan menguraikan solusi menjadi sekumpulan langkah-langkah sedemikian sehingga solusi dari sebuah persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan[6]. Prinsip yang digunakan dalam dynamic programming di algoritma ini adalah prinsip optimasi, yaitu setiap langkah yang diambil merupakan langkah yang paling maksimum, dalam hal ini berarti memberikan skor yang terbesar dari segala kemungkinan langkah pembentuknya. Dalam bioinformatika *Sequence alignment* adalah suatu proses penyusunan atau penjajaran antara dua atau lebih *sequence* agar ditemukan kecocokan atau kemiripan[8].

Pesatnya peningkatan banyak *sequence* dalam database DNA atau protein menjadi salah satu tantangan dalam proses penjajaran di masa yang akan datang sehingga dibutuhkan langkah-langkah yang efisien untuk menyelesaikan masalah tersebut. Melakukan optimasi algoritma *Smith Waterman* dalam BLAST *(Basic local search Alignment Tool)* untuk mempercepat pencacahan dan pencocokan dengan jumlah dataset yang besar akan membutuhkan sumber daya dan waktu yang lebih lama, maka usulan optimasi dilakukan di ranah pemrosesan komputasi dan penyimpanan basis data dari dataset.

Apache spark digunakan untuk menganalisis data secara *real-time* dan juga merupakan perangkat lunak yang efektif untuk memproses data dengan skala yang besar[7]. *Apache spark* digunakan untuk manajemen komputasi dan penanganan aplikasi BLAST yang berjalan, kemudian *Apache* Hadoop digunakan untuk menangani penyimpanan dataset protein nukleutida. Riset optimasi algoritma pencacahan dengan menggunakan BLAST di atas *apache spark* telah dilakukan oleh G.Zhao, dkk [8] Hui Wang dkk[19], C.Christoper Craig[20],dari referensi tersebut menghasilkan kecepatan pencacahan yang lebih baik dibanding dengan NCBI BLAST yang dipasang secara *standalone*. Penelitian ini dilakukan dengan melakukan simulasi dengan 1 mesin komputer untuk melihat perbandingan kinerja antara program NCBI BLAST *standalone* yang terpasang dengan penyimpanan lokal terhadap program NCBI BLAST yang berjalan diatas *Apache Spark* dan 1 node Hadoop dipakai sebagai manajer penyimpanan dataset *nukleutida* NCBI didalam HDFS. Sehingga dapat dibuktikan nilai optimasi kinerja pencacahan yang dibahas.

1.2 Topik dan Batasan

Topik yang dibahas pada tugas akhir ini yaitu merupakan proses alignment untuk mencari kemiripan query data dalam suatu database dengan menggunakan
Batasan-batasan dari tugas akhir ini yaitu:

1. Menggunakan dataset protein nukleotida diambil dari database NCBI .
2. Simulasi dijalankan dalam konfigurasi 1 mesin komputer dengan 1 node penyimpanan hadoop.

1.3 Perumusan Masalah

1. Bagaimana ukuran performansi kecepatan pencacahan sekuensi dari kedua sistem yang dipasang yakni NCBI BLAST *standalone* dan NCBI BLAST dengan bantuan *Apache Spark* dan *Apache Hadoop*.
2. Bagaimana hasil pencarian sekuensi dan akurasi setelah diterapkan NCBI BLAST dengan *Apache Spark* dan *Apache Hadoop*.

1.4 Tujuan

Tujuan pada penelitian tugas akhir ini yaitu:

1. Melakukan implementasi *Basic Local Search tool* menjadi aplikasi yang berjalan diatas *Apache Spark* beserta dengan penyimpanan dataset nukleutida didalam 1 node penyimpanan *Apache Hadoop*.
2. Mengukur aspek optimasi dari hasil implementasi dari segi kecepatan pemrosesan dan pencacahan dibandingkan dengan NCBI BLAST dengan konfigurasi *standalone*.

2. Studi Terkait

2.1 Sequence Alignment

Sequence alignment adalah proses penyusunan dua atau lebih *sequence* sehingga persamaan *sequence-sequence* tersebut tampak nyata. *Sequence alignment* merupakan metode dasar dalam analisis *sequence*,

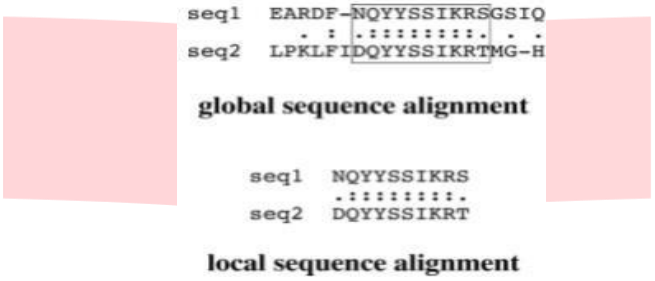
metode ini melakukan proses dimana urutan dibandingkan dengan mencari pola karakter umum dan membangun koresponden residu-residu antara urutan yang terkait. *Pairwise sequence alignment* adalah proses penyelarasan antara dua urutan dan merupakan dasar pencarian database similarity. Tujuan dari *pairwise sequence* adalah untuk menemukan pasangan terbaik dari dua urutan, sehingga menghasilkan korespondensi maksimum di antara *residu-residu*. Untuk mencapai tujuan tersebut ditemukan dua strategi alignment berbeda yang dapat digunakan yaitu *global alignment* dan *local alignment*[5], [8].

1. ***Global Alignment***

   Global Alignment berfungsi untuk menentukan tingkat kesamaan atau kemiripan antara *sequence-sequence* yang disejajarkan. *alignment* dilakukan dari awal hingga akhir dari kedua *sequence*, kedua metode ini diterapkan untuk menyejajarkan secara bersamaan berdasarkan dari panjang yang kira-kira sama.[9]

2. ***Local Alignment***

   Local alignment berfungsi untuk menemukan bagian yang paling mirip antara *sequence-sequence* yang disejajarkan, dua *sequence* yang akan disejajarkan dapat memiliki panjang yang berbeda [9]



Gambar 2.1 Contoh Perbandingan Pairwise yang Menunjukan Perbedaan Antara Global Alignment dan Local Alignment [9]

2.2 Dynammic Programming

Dynamic programing adalah suatu metode pemecahan masalah, *dynamic programming* juga mengatur perhitungan untuk menghindari perhitungan ulang nilai yang sudah diketahui agar dapat menghemat banyak waktu[10]. *Dynamic programming* juga dapat dikatakan sebagai metode pemecahan masalah dengan cara menguraikan solusi menjadi *step* atau tahapan sedemikian rupa sehingga solusi dari persoalan tersebut dapat dipandang dari serangkaian keputusan yang saling berkaitan. *Dynamic programming* pada dasarnya mirip dengan metode dot matrix dimana hal tersebut juga menciptakan garis *alignment* dua dimensi. *Dynamic programming* menemukan cara dengan mengubah dot matriks menjadi *scoring matrix* untuk memperhitungkan match dan mismatch, dengan cara mencari skor tertinggi dalam matriks ini[6].

2.3 Algoritma Smith-Waterman

Algoritma *Smith-Waterman* adalah algoritma yang digunakan khusus untuk meneliti *local alignment* dengan cara membandingkan *segment-segment* dari dua buah *sequence*. Algoritma ini merupakan salah satu yang termasuk *pairwise alignment*, dalam penjajaran protein, algoritma *Smith-Waterman* membandingkan setiap *segment* dari *sequence* protein [5]. Algoritma Smith-Waterman mampu menyelaraskan DNA *sequence alignment* tanpa harus menggunakan ujung-ujung DNA yang terkait. Algoritma *Smith-Waterman* digunakan juga untuk mengidentifikasi *local alignment* yang optimal antara *query* dan *subject sequence* dengan cara menghitung skor kesamaan dengan menggunakan metode *dynamic programming*. Algoritma *Smith-Waterman* dapat dibagi menjadi dua tahap yaitu : yang pertama *similarity matrix* atau biasa disebut *matrix alignment* dapat digunakan untuk mendapatkan *skor alignment* yang optimal dan yang kedua yaitu *traceback* yang digunakan untuk mendapatkan *alignment path* yang optimal.[11]

1. *Similarity Matrix Filling*

   Query sequence didefinisikan dengan $Q = q1, q2, q3 \ldots qm$ dan Panjang (Q) = m. subjek *sequence* didefinisikan sebagai $S = s1, s2, s3 \ldots sn$ dan panjang (S) = n. SM adalah subtitusi matriks yang mendefinisikan skor substitusi untuk semua pasangan residu. Biasanya bobot SM yaitu $(qi, sj) \leq 0$ saat $qi \neq sj$ dan $SM\ (qi, sj) > 0$ saat $qi = sj$. Substitusi matriks umumnya digunakan untuk penjajaran protein contohnya seperti BLOSUM atau PAM. *Penalty* digunakan

untuk membuka *gap* dan memperluas *gap*, masing-masing didefinisikan dengan *Gopen* dan *Gext*. *Ei,j* dan *Fi,j* merupakan skor yang melibatkan symbol *I* pertama dari *Q* dan symbol *j* pertama dari *S*, dan diakhiri dengan *gap* dalam urutan *Q* atau *S*. *Hi,j* adalah skor *alignment* yang ditunjukan sebagai berikut :

$$E_{i,j} = \max\{E_{i,j-1} - G_{ext}, H_{i,j-1} - G_{open}\}$$
$$F_{i,j} = \max\{F_{i-1,j} - G_{ext}, H_{i-1,j} - G_{open}\}$$
$$H_{i,j} = \max\{0, E_{i,j}, F_{i,j}, H_{i-1,j-1} - SM(q_i, s_j)\}$$

Nilai *Ei,j*,*Fi,j* dan *Hi,j* diinisialisasi dengan 0 ketika *I* = 0 atau *j* = 0. Skor *alignment* yang paling maksimal dalam matriks H adalah skor *local alignment* yang optimal[12].

2. *Traceback*

Traceback digunakan untuk analisis atau pengoptimalan, seperti mapping dan variasi. Setelah skor *local alignment* optimal telah diperoleh, maka *traceback* dilakukan untuk mendapatkan pasangan segmen dengan *similarity* maksimum berdasarkan matriks H, hingga posisi yang nilainya nol tercapai. Algoritma *Smith-Waterma*n bias dibagi menjadi dua mode berdasarkan apakah ada *traceback* atau tidak yaitu : mode *alignment scores (ASM)* dan mode lain yang mendukung yaitu *alignment traceback (ATM)*[12].

Berikut adalah contoh ilustrasi proses perhitungan algoritma *Smith-Waterman* antara *query* PNHIGD dan PTHIKWGD, yang menggunakan BLOSUM50 sebagai matriks substitusi, 12 *penalty* sebagai open *gap* penalty dan 2 sebagai *penalty extension gap penalty*.

	-	P	T	H	I	K	W	G	D
-	0	0	0	0	0	0	0	0	0
P	0	10	0	0	0	0	0	0	0
N	0	0	10	1	0	0	0	0	2
H	0	0	0	20	8	6	4	2	0
I	0	0	0	8	25	13	11	9	7
G	0	0	0	6	13	23	11	19	8
D	0	0	0	4	11	12	18	10	27

Gambar 2.2 Ilustrasi Algoritma Smith-Waterman [1]

2.4 Apache Spark

Apache spark adalah *open-source software* yang digunakan untuk menganalisis data secara *real-time*. *Apache spark* menggunakan model pemrograman baru yang mendukung algoritma *iterative machine learning* serta analisis data interaktif, yang mempertahankan skalabilitas toleransi kesalahan *MapReduce*[13]. Dalam pemrograman parallel, spark memasok dua abstraksi utama yaitu: *Resilient Distributed Datasets* (RDDs) dan operasi parallel pada dataset. RDD adalah fitur paling penting di *apache spark*, yang memungkinkan pemrograman melakukan perhitungan dalam memori pada *cluster* yang besar. Dalam spark setiap RDD diekspresikan sebagai objek *scala* dan *spark provider*. RDD dapat dibangun kembali apabila partisi di seluruh *cluster* hilang, dalam spark setiap RDD diekspresikan sebagai objek scala dan terdapat empat cara untuk membangun spark yaitu :

1. Dari file system bersama, seperti HDFS.

2. Dengan cara memparalelkan scala dan membaginya menjadi irisan yang berbeda kemudian mengirimkan setiap irisan ke pekerja di *cluster*.
3. Dengan cara mengubah RDD yang ada menjadi RDD yang baru, spark juga menyediakan metode untuk mengubah RDD, contohnya seperti *flatMap*, *SortByKey*, filter dan yang lainnya.
4. Dengan cara mengubah RDD yang ada kemudian disimpan dalam memori atau HDFS.

Selain itu *spark* juga menyediakan dua jenis *variable* bersama yaitu *broadcast variables* dan *accumulators variables*[13].

2.5 BLAST

BLAST adalah program yang digunakan untuk membandingkan sekuens biologis primer contohnya seperti sekuens asam amino protein, sekuens RNA dan sekuens nukleotida. Adapun beberapa jenis BLAST yaitu BLASTN, BLASTP, BLASTX TBLASTN, dan TBLASTX. Dimana BLASTN menggunakan untuk *sequence* nukleotida dengan database *sequence* nukleotida, BLASTP menggunakan *sequence* protein sebagai *query* untuk mencocokan berdasarkan database sequence protein, kemudian BLASTX menggunakan sequence nukleotida sebagai sequence, yang selanjutnya yaitu TBLASTN menggunakan sequence protein dan yang terakhir yaitu TBLASTX menggunakan sequence nukleotida.[1].

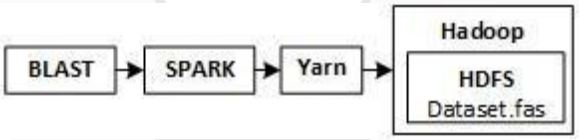
3. Sistem yang Dibangun

3.1 Arsitektur Rancangan

Penelitian ini menggunakan model arsitektur yang akan dijelaskan pada subbab berikut:

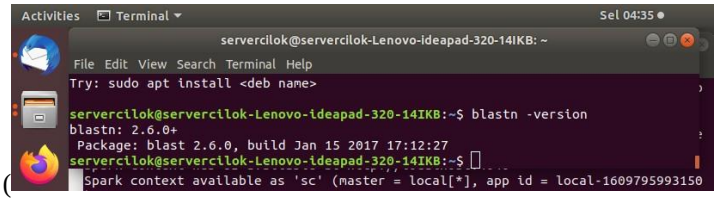
Gambar 3.1 skema arsitektur rancangan BLAST

BLAST API dari NCBI akan dipakai sebagai pencarian dan pencocokan rantai protein sesuai seukensi yang dicari, script BLAST dipasang di sisi Spark untuk penanganan job, komunikasi dan memori. Hadoop digunakan untuk penanganan penyimpanan blok klaster dataset.



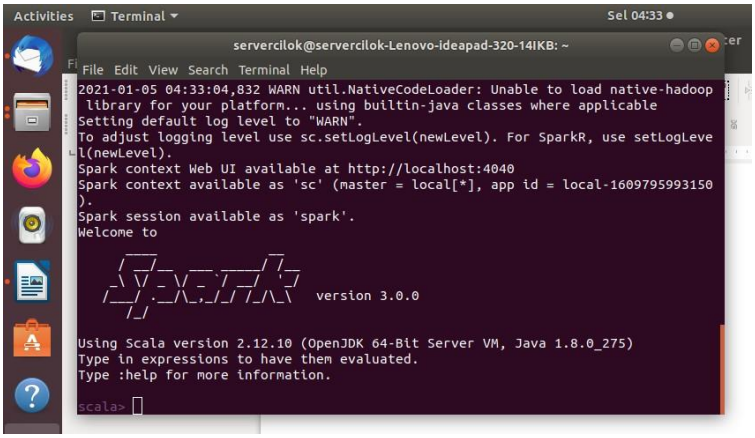
Gambar 3.1 Data Blok Distribusi job pada sistem

Seperti terlihat pada Data Blok, komponen dependensi yang dibutuhkan adalah BLAST, *Apache Spark* dan *Apache Hadoop*. Komponen *Apache BLAST* yang dipasang adalah versi 2.6.



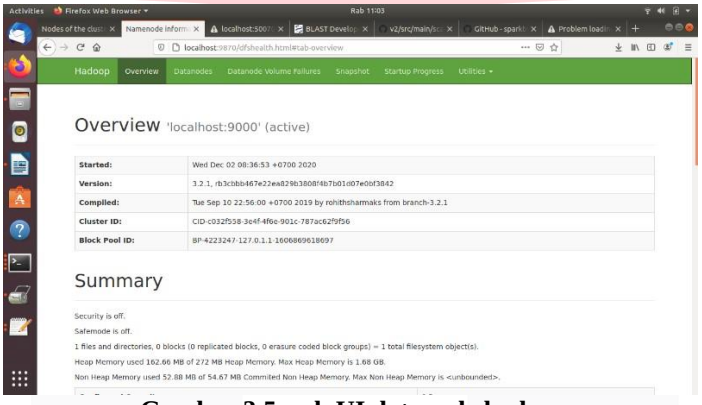
Gambar 3.3 Output Versi BLASTn

Tool BLAST yang akan digunakan ialah BLASTn untuk pencacahan sekuensi protein nucleotide, sedangkan *Apache Spark* yang digunakan ialah *Apache Spark* versi 3.0 bersama scala versi 2.12



Gambar 3.4 Output Versi Spark

Hadoop yang digunakan untuk node penyimpanan sistem ialah versi 3, berebeda dengan versi 2 sebelumnya Hadoop 3 memiliki skema akses penyimpanan HDFS *(Hadoop File System)* yang sudah dikonfigurasi secara *default*. Penanganan *map reducer* pun menggunakan *Yarn*.



Gambar 3.5 web UI datanode hadoop

Komponen datanode hadoop akan mencatat metrix penyimpanan seperti, besar berkas, jumlah berkas, kegagalan penyimpanan, proses akses dan input output berkas.

Kumpulan data yang terdiri dari dataset dan query masing masing disimpan dalam file DATASET1.fas dan tesquery.fas. script menggunakan API SparkBLAST[20] untuk membuat BLAST bekerja diatas Spark. S pertama-tama dilakukan cloning berkas API [20] dengan perintah

```
git clone https://github.com/Ufscar-Fiocruz-Ifsul/spark-blast2.0.git
```

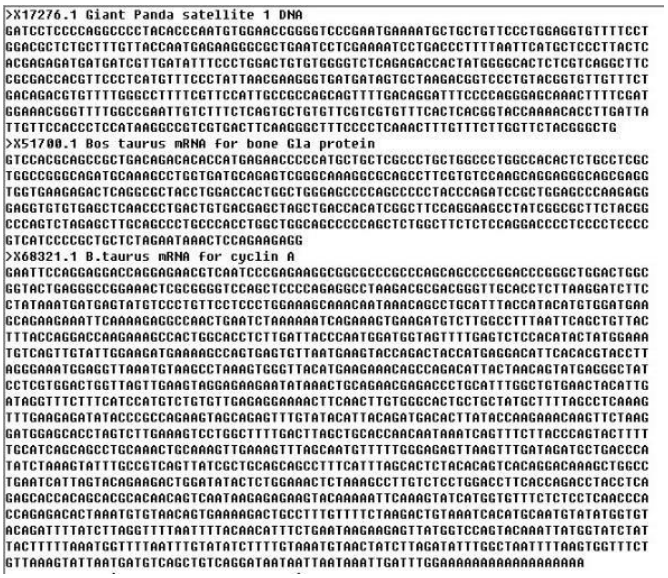
Selanjutnya meng-compile paket dependensi NCBI BLAST 2.6 *standalone* yang sudah terpasang dengan API SparkBLAST. Untuk menjalankan aplikasi menggunakan script berikut:

```
spark-submit --executor-memory qtd-memory --driver-memory qtd-memory --num-executors num-executor --executor-cores 1 --driver-cores num-cores --class SparkBlast target/scala-2.12/simple-project_2.10-1.0.jar num-cores "/home/hadoop/blastall/bin/blastall -p blastn -d usr/local/hadoop/tempdata/database.fa -e 1E-05 -v 1000 -b 1000 -m 8" input output; hadoop hdfs-getmerge output output.local
```

aplikasi di submit kedalam *Apache Spark*, versi scala diatur mengikuti versi yang diinstal, dalam komputer ini menggunakan scala versi 2.10 lokasi penyimpanan dataset diatur sesuai dengan penyimpanan folder database node hadoop, dalam sistem ini dataset disimpan di direktori instalasi hadoop, dalam komputer simulasi ini pada alamat usr/local

3.2 Dataset

Dataset yang digunakan dalam penelitian ini yaitu di peroleh dari *cluster* referensi NCBI *(National Center for Biotechnology)* yang merupakan *server* yang memuat *database* tentang informasi kesehatan dan bioteknologi [15]. Dataset yang digunakan yaitu protein nucleotide dengan format FASTA.



Gambar 3.6 Bentukan berkas dataset nukleutida

Struktur dataset terdiri dari *header* dan konten, konten berupa rantai *nukleotida* yang akan digunakan dalam *query* pencacahan, tiap spesies maupun protein memiliki rantai yang unik namun memiliki komposisi kemiripan sekuens dengan *family* atau indukannya. *Header* terdiri dari nomor id penyimpanan dilanjutkan dengan nama representasi pemilik sekuensi protein tersebut. Dalam penelitian ini digunakan partisi kecil dari dataset NCBI nt sebesar 33mb dari total 320 GB.

Sedangkan berkas *query* adalah berkas berformat fasta berisi karakter sekuens yang ingin dicocokkan dan dicari kedalam dataset.

Karakter query tidak perlu harus memiliki *header* identitas karena tujuan sistem pencocokan ini ialah untuk mencari identitas yang cocok terhadap karakter yang dicari. dan Berkas *query* harus memiliki minimal 30 karakter protein yang valid, apabila dibawah 30 karakter maka pencarian gagal

3.3 Skenario Pengujian

Skenario pengujian untuk mengetahui kinerja sistem alignment rancangan, apakah sistem alignment dapat melakukan pencacahan terhadap input sampel.

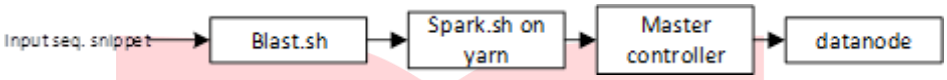
- Skenario 1
  menggunakan query dengan jumlah sekuensi DNA 81,162,243,324,405 kemudian dilihat kesesuaian dan bobot kecocokannya. Tujuan pengujian scenario 1 ini adalah untuk menguji kecepatan pemrosesan terhadap jumlah query yang dimasukkan.
- Skenario 2
  Menggunakan query campuran yang terdiri dari data yang berasal dari 3 sequensial yang berbeda.
- Skenario 3
  Menggunakan query yang barisannya dimanipulasi di beberapa titik data, bersifat acak 3-20 titik. Tujuan pengujian ini untuk melihat kinerja system apabila berasal dari specimen yang memiliki data sekuensi cacat atau telah bermutasi.
- Skenario 4
  Menggunakan query specimen asing yang berasal dari luar dataset kemudian dilakukan pencocokan apakah memiliki kemiripan dengan id yang ada, hal ini untuk menunjukkan pola kelengkapan data dan sensitifitas sistem pencocokan terhadap query dengan kemungkinan kecocokan yang sangat kecil.

Parameter yang dilihat dari masing-masing scenario adalah nilai akurasi kecocokan dan kecepatan pencacahan.Nilai kecocokan berdasarkan bobot *scoring* dan *E-value*, *E-value* adalah nilai probabilitas ekspektasi ditemukannya karakter dalam 1 sekuens, semakin kecil nilai *E-value* maka semakin baik kecocokannya. Bit score adalah nilai banyaknya karakter kecocokan query terhadap semua id yang dianggap cocok didalam database.

Kecepatan pencacahan diambil dari *command time* terminal linux. Nilai yang ditampilkan adalah nilai *real* yang berarti waktu yang digunakan untuk memproses perintah, *user* ialah jumlah total proses yang digunakan secara langsung, *sys* ialah jumlah total waktu pemrosesan oleh kernel. Semakin kecil waktu maka semakin cepat perintah dan pemrosesan dilakukan.

3.3.1 Fungsionalitas system

Pengujian dilakukan dengan melakukan perbandingan antara *alignment* dari BLAST NCBI terdedikasi, spark blast hadoop dengan *single node* dan sparkblast hadoop multi data node.



Gambar 3.7 tiga skema pengujian pencacahan BLAST

Pengujian menggunakan personal komputer dengan spesifikasi Core i5 2.5 ghz physical memory 8gb dengan system operasi Ubuntu 14 dan dependensi NCBI BLAST 2.2.28. Pengujian menggunakan parameter pengujian sesuai scenario pengujian.

3.3.2 Performansi system

Pengujian indikasi performansi sistem dilakukan dari komponen sistem dilihat dari jalannya dependensi *Apache Hadoop*, *Apache Spark* dan BLAST, apabila semua dependensi tidak menunjukkan error maka fungsi dasar dari integrasi sistem tercapai. Performansi sistem yang dirancang dinilai dari aspek kecepatan dan akurasi, aspek kecepatan diukur dengan memanggil perintah *time* di dalam terminal setiap kali perintah

parameter akurasi dinilai dari tingkat kecocokan dan kesesuaian identitas karakter sebenarnya terhadap identitas hasil pencocokan yang ditemukan dari menjalankan perintah pencocokan, sedangkan nilai parameter kecocokan dari aspek BLAST dapat dilihat pada skenario pengujian.

4. Hasil dan Evaluasi

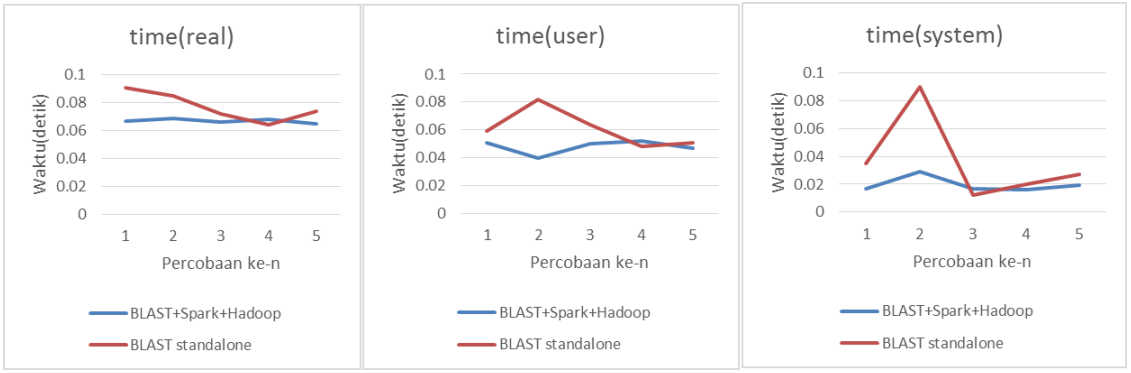
Bagian evaluasi ini membahas hasil pengujian dari skenario yang telah diusulkan pada bagian 3, Berikut ini adalah hasil pengujian dari 4 skenario usulan.

Skenario 1 ialah pengujian dengan memakai jumlah karakter query berbeda-beda pada tiap aktivitas pencarian. Informasi pengujian ditunjukkan pada lampiran 1. Tabel 4.1 menunjukkan rekapitulasi parameter waktu per aktivitas query dengan 5 jumlah karakter berbeda.

Tabel 4.1 rekapitulasi parameter waktu

Uji ke	jumlah karakter	BLAST+Spark+Hadoop(detik)			BLAST standalone(detik)		
		real	user	system	real	user	system
1	81	0.067	0.051	0.017	0.091	0.059	0.035
2	162	0.069	0.04	0.029	0.085	0.082	0.09
3	243	0.066	0.05	0.017	0.072	0.064	0.012
4	324	0.068	0.052	0.016	0.064	0.048	0.02
5	405	0.065	0.047	0.019	0.074	0.051	0.027

Masing-masing pengujian terdiri dari 5 jumlah karakter berbeda yakni 81,162,243,324 dan 405 karakter diambil dari 5 identitas sekuensi yang sama. Pola data dapat terlihat pada grafis 4.1 berikut.



Gambar 4.1 Grafik perbandingan waktu proses dari sistem BLAST+Spark+Hadoop dan BLAST standalone

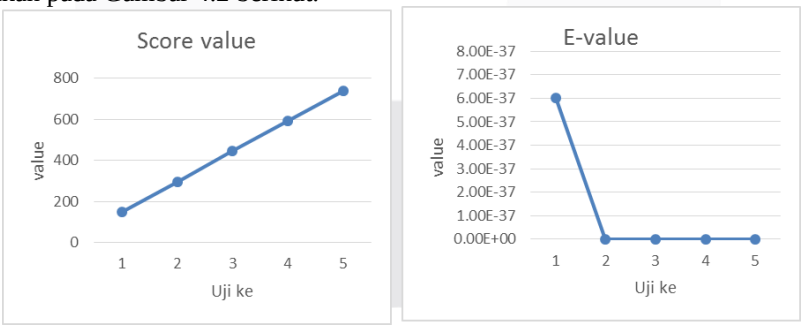
Pada grafik pada gambar 4.1 terlihat dari 3 parameter real,user dan system bahwa konfigurasi BLAST+Spark+Hadoop memiliki waktu siklus rata-rata dari 5 pengujian yang lebih rendah dibanding BLAST standalone dengan penyimpanan lokal, optimasi waktu ini karena aplikasi pencacahan yang berjalan di atas Apache Spark dibanding berjalan di kernel dasar. Dari 5 jumlah karakter berbeda menunjukkan jumlah karakter tidak mempengaruhi waktu siklus pencacahan.

Sedangkan hasil akurasi pencarian ditunjukkan pada tabel 4.2, parameter uji ialah Score Bit dan E-value seperti yang dijabarkan pada bagian uji skenario.

Tabel 4.2 rekapitulasi akurasi pencarian skenario 1

Uji ke	jumlah karakter	BLAST+Spark+Hadoop			BLAST standalone		
		Score	E-value	Status	Score	E-value	Status
1	81	148	6.00E-37	tepat	148	6.00E-37	tepat
2	162	296	5.00E-81	tepat	296	5.00E-81	tepat
3	243	444	3.00E-125	tepat	444	3.00E-125	tepat
4	324	592	1.00E-169	tepat	592	1.00E-169	tepat
5	405	739	0	tepat	739	0	tepat

Dari 5 pengujian dengan jumlah karakter berbeda menunjukkan bahwa 5 identitas pencarian tepat sasaran sesuai target yang diinginkan, pengujian ditunjukkan pada lampiran 4 sedangkan tren hubungan jumlah karakter terhadap kecocokan ditunjukkan pada Gambar 4.2 berikut.



Gambar 4.2 Grafik peforma akurasi dari 5 jumlah karakter query berbeda.

Dari pengujian skenario 1 ini menunjukkan bahwa perbandingan BLAST+Spark+Hadoop dan Blast Standalone tidak menunjukkan perbedaan, artinya konfigurasi BLAST+Spark+Hadoop tidak mempengaruhi akurasi proses pencocokan, dari pengujian menunjukkan makin banyak karakter membuat Score Bit makin naik dan menurunkan E-value hal ini menunjukkan makin banyak karakter maka menaikkan akurasi pencocokan sekuensi.

Skenario 2 menunjukkan pengujian dengan menggunakan query dengan mencampurkan karakter dari bermacam-macam id sekuens. Informasi pengujian ditunjukkan pada lampiran 2. Hasil uji waktu ditunjukkan pada Tabel 4.3 berikut:

Tabel 4.3 rekapitulasi waktu uji skenario 2

Uji ke	BLAST+Spark+Hadoop(detik)			BLAST standalone(detik)		
	real	user	system	real	user	system

1	0.069	0.055	0.013	0.068	0.067	0.009
2	0.192	0.073	0.02	0.042	0.04	0.012
3	0.073	0.061	0.012	0.045	0.049	0.008
4	0.077	0.057	0.02	0.049	0.05	0.015

Pada pencarian dengan pencocokan dengan dataset yang tercampur antar sekuens menunjukkan bahwa kinerja pencocokan menggunakan NCBI standalone memiliki waktu siklus lebih ringkas. Hal ini disebabkan pencarian beulang yang dilakukan untuk menemukan tiap kemungkinan karakter yang muncul, dengan pemrosesan bawaan dan kernel linux menunjukkan siklus yang lebih optimal dibanding dengan menjalankan pencarian skenario 2 menggunakan Apache Spark.Sedangkan hasil pencocokan ditunjukkan oleh Tabel 4.4.

Tabel 4.4 rekapitulasi kecocokan temuan uji skenario 2

Uji ke	BLAST+Spark+Hadoop(jumlah)		BLAST standalone(jumlah)	
	id query	id found	id query	id found
1	6	5	6	5
2	6	5	6	5
3	6	3	6	3
4	6	5	6	5

Hasil yang ditunjukkan oleh tabel 4.4 menunjukkan hasil pencocokan dari 6 id query yang diinginkan memiliki akurasi 3/6 dan 5/6 temuan teratas dari seharusnya. Hal ini menunjukkan peforma dan akurasi yang sama antara kedua pengaturan.

Informasi pengujian ditunjukkan pada lampiran 3. Hasil tabulasi rekapitulasi waktu uji skenario 3 ditunjukkan pada tabel 4.5

Tabel 4.5 rekapitulasi waktu uji skenario 3

Uji ke	BLAST+Spark+Hadoop(detik)			BLAST standalone(detik)		
	real	user	system	real	user	system
1	0.077	0.066	0.016	0.047	0.056	0.005
2	0.049	0.044	0.005	0.046	0.056	0.004
3	0.077	0.052	0.025	0.077	0.067	0.023
4	0.078	0.057	0.02	0.048	0.048	0.012

Tabel hasil uji akhir menunjukkan bahwa rata-rata waktu dari masing-masing parameter menunjukkan nilai yang seragam. Menunjukkan skenario pencacahan dengan data yang dirusak memiliki beban kerja yang sama terhadap kedua sistem.

Tabel 4.6 rekapitulasi akurasi pencarian skenario 3

Uji ke	BLAST+Spark+Hadoop(detik)			BLAST standalone(detik)		
	score	E-value	kecocokan id	score	E-value	kecocokan id
1	3129	0	2 id	3129	0	2 id
2	2987	0	2 id	2987	0	2 id
3	2124	0	3 id	2124	0	3 id
4	3482	0	1 id	3482	0	1 id

Hasil uji Skenario 4 ditunjukkan oleh Lampiran 4, skenario 4 diuji dengan mengambil data query diluar dataset, query berasal dari sekuens milik *Helicobacter pylori gene for 23S rRNA, complete sequence*, dengan jumlah karakter berbeda. Hasil uji baik dari parameter waktu dan kecocokan ditunjukkan pada Tabel 4.7 berikut.

Tabel 4.7 rekapitulasi pengujian skenario 4

Uji ke	jumlah karakter	BLAST+Spark+Hadoop(detik)			hasil pencocokan	BLAST standalone(detik)			hasil pencocokan
		real	user	system		real	user	system	
1	81	0.071	0.055	0.017	no hits found	0.07	0.052	0.025	no hits found
2	162	0.072	0.059	0.013	no hits found	0.05	0.041	0.017	no hits found
3	243	0.241	0.075	0.013	found	0.074	0.062	0.023	found
4	324	0.183	0.056	0.029	found	0.043	0.052	0.005	found
5	405	0.046	0.028	0.017	found	0.044	0.045	0.013	found
6	mixed	0.044	0.039	0.005	no hits found	0.042	0.037	0.013	no hits found

Dari hasil pengujian pada skenario 4 menunjukkan bahwa semakin banyak rantai protein karakter yang ditemukan dari sampel maka makin tinggi kemungkinan ditemukannya kecocokan terhadap identitas yang lain, kecocokan dengan identitas rantai lain menunjukkan ada hubungan rantai protein baik secara turunan spesies maupun *family* yang sama. Pada jumlah karakter 243 menunjukkan jumlah karakter minimum terhadap pencarian yang optimal untuk sebuah rantai protein baru yang ditemukan.

5. Kesimpulan

Sistem NCBI BLAST dapat dipasang diplatform *apache spark* bersama dengan manajemen dataset Hadoop dengan konfigurasi single node. Dari 4 tipe skenario yang dilakukan terhadap pencacahan dengan BLAST tidak terganggu dengan konfigurasi NCBI BLAST+Apache Spark +Apache Hadoop ini, baik saat kondisi ekstrim saat data query rusak maupun tercampur. Akan tetapi terdapat perbedaan pada waktu pemrosesan. Pada saat jumlah query yang tinggi namun dengan data lengkap ditunjukkan pada pengujian 1 menunjukkan waktu komputasi yang lebih cepat, akan tetapi saat data tercampur menunjukkan bahwa NCBI standalone dengan menggunakan sistem operasi linux Ubuntu dan mesin bawaan mampu menghasilkan kecepatan siklus lebih ringkas, hal ini karena didukung oleh kapabilitas pemrosesan *multicore* yang dimiliki komputer simulasi. Kemudian waktu siklus dari kedua konfigurasi menunjukkan waktu yang sama.

Referensi

[1] W. Miller, *An Introduction to Bioinformatics Algorithms*, vol. 101, no. 474. 2006.
[2] T. L. Wargasetia, “Peran Bioinformatika dalam Bidang Kedokteran,” *Jurnal Kedokteran Maranatha*, 2010.
[3] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, 1981, doi: 10.1016/0022-2836(81)90087-5.
[4] E. Ernawati, D. Puspitaningrum, and A. Pravitasari, “Implementasi Algoritma Smith–Waterman Pada Local Alignment Dalam Pencarian Kesamaan Pensejajaran Barisan DNA (Studi Kasus: DNA Tumor Wilms),” *Pseudocode*, 2015, doi: 10.33369/pseudocode.1.2.170-177.
[5] F. N. Muhamad, R. B. Ahmad, S. M. Asi, and M. N. Murad, “Reducing the search space and time complexity of Needleman-Wunsch algorithm (Global alignment) and smith-waterman algorithm (local alignment) for DNA sequence alignment,” *Jurnal Teknologi*, vol. 77, no. 20, pp. 137–146, 2015, doi: 10.11113/jt.v77.6564.
[6] Fikri Akbar L and Rosnani Ginting, “Dynamic Programming dalam Penyelesaian Masalah Penjadwalan,” *Talenta Conference Series: Energy and Engineering (EE)*, 2019, doi: 10.32734/ee.v2i3.751.
[7] R. Guo, Y. Zhao, Q. Zou, X. Fang, and S. Peng, “Bioinformatics applications on Apache Spark,” *GigaScience*, vol. 7, no. 8, 2018, doi: 10.1093/gigascience/giy098.
[8] G. Zhao, C. Ling, and D. Sun, “SparkSW: Scalable distributed computing system for large-scale biological sequence alignment,” in *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*, 2015, doi: 10.1109/CCGrid.2015.55.
[9] A. A. Sunarto, “Perbandingan Program Sequence Alignment,” *Jurnal Rekayasa Nusaputra*, vol. 1, no. 1, pp. 1–5, 2015.
[10] G. Aristi, “Perbandingan Algoritma Greedy, Algoritma Cheapest Insertion Heuristics Dan Dynamic Programming Dalam Penyelesaian Travelling Salesman Problem,” *Paradigma*, vol. XVI, no. 2, pp. 52–58, 2014, [Online]. Available: https://ejournal.bsi.ac.id/ejurnal/index.php/paradigma/article/view/779.
[11] B. Xu, C. Li, H. Zhuang, J. Wang, Q. Wang, and X. Zhou, “Efficient Distributed Smith-Waterman Algorithm Based on Apache Spark,” *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2017-June, pp. 608–615, 2017, doi: 10.1109/CLOUD.2017.83.
[12] R. V. Imbar *et al.*, “Implementasi Cosine Similarity dan Algoritma Smith-Waterman untuk Mendeteksi Kemiripan Teks,” *Jurnal Informatika*, 2014.

[13] C. Z. Tumbel, H. Sitepu, and M. Hutagalung, "Analisis Big Data Berbasis Stream Processing Menggunakan Apache Spark," *Jurnal Telematika*, vol. 11, no. 1, p. 6, 2017, [Online]. Available: http://journal.ithb.ac.id/telematika/article/view/145.

[14] R. J. (LinkedIn) Chansler, "Data Availability and Durability with the Hadoop Distributed File System,"
;*Login*, 2012.

[15] T. Barrett *et al.*, "NCBI GEO: Archive for functional genomics data sets - Update," *Nucleic Acids Research*, 2013, doi: 10.1093/nar/gks1193.

[16] A. Novanta, "PENDETEKSIAN PLAGIARISME PADA DOKUMEN TEKS DENGAN MENGGUNAKAN ALGORITMA SIMTH-WATERMAN," 2009.

[17] J. Oliver, "APLIKASI UNTUK MEMBANDINGKAN KEMIRIPAN DUA BUAH TEKS/DOKUMEN BERBASIS WEB DENGAN MENGGUNAKAN ALGORITMA SMITH-WATERMAN," *Journal of Chemical Information and Modeling*, 2013.

[18] C. B. Do, M. S. P. Mahabhashyam, M. Brudno, and S. Batzoglou, "ProbCons: Probabilistic consistency-based multiple sequence alignment," pp. 330–340, 2005, doi: 10.1101/gr.2821705.1994.

[19] H.Wang,L.Li,C.Zhou,H.Lin,D.Deng, "Spark-based Parallelization of Basic Local Alignment Search Tool" College of Data Science and Application Inner Mongolia University of Technology,2020

[20] C.C.Chambers, "An Implementation of NCBI BLAST in the Apache Spark Framework," CALIFORNIA STATE UNIVERSITY, NORTHRIDGE,2016