International Journal of Computational Bioinformatics and In Silico Modeling
Vol. 2, No. 3 (2013): 106-109
Research Article
Open Access

ISSN: 2320-0634

Algorithm for Optimal Storage of a Distributed Bioinformatics System for Analysis of DNA Sequences

Chotan Sheel*, Mohammad Ibrahim Khan, Md. Iqbal Hasan Sarker and Tauhidul Alam

Department of CSE, Chittagong University of Engineering & Technology (CUET), Chittagong, Bangladesh.

*Corresponding author: Chotan Sheel, e-mail: chotan_cuetcse03@yahoo.com

Received: 03 March 2013 Accepted: 26 March 2013 Online: 01 May 2013

ABSTRACT

This paper provides an optimal storage algorithm of an effective design and implementation of a distributed bioinformatics computing system for analysis of DNA sequences (OPTSDNA). This system could be used for storing various sizes of DNA sequences into database. DNA sequences of different lengths were stored by using this algorithm. These sequences varied in size from very small to very large. The performance of this storage system is compared with sequential approach.

Keywords: Distributed Bioinformatics System, DNA Sequence, Optimal Storage, Sequential Approach

INTRODUCTION

Distributed computing (DC) provides a cost effective framework with efficient execution of a solution on multiple computers connected by a network. For Distributed Computing (DC), large tasks are divided into smaller problems which can then be executed on multiple computers at the same time independent of each other. The task must be broken up into independent problems to minimize inter-computers communication; otherwise distributed computing will not be effective. Over the past few years, the intermixing of computer science and the complexity of biology has lead to the prosperous field of bioinformatics [1]. Advances in molecular biology and technology for research have facilitated the process of sequencing of large portions of genomes in various species. Today computers have made medical research more efficient and accurate, by using parallel and distributed computers and complex biological modeling. Bioinformatics, is one of the newer areas, and has opened our eyes to a whole new world of biology.

The fusion of computers and biology has helped scientists learn more about species, especially humans. With the aid of the computers, we have learned a great deal about genetics, but there still stand many unanswered questions, that are being researched today. DNA sequence analysis can be a lengthy process ranging from several hours to many days. This paper builds a distributed system that provides the solution for many bioinformatics related applications.

The overall goal of this paper is to build an optimal storage of Distributed Bioinformatics Computing System for DNA (OPTSDNA) sequence analysis. This algorithm is capable of storing various length of DNA sequence in a Database by compressing the DNA sequence. We observed this algorithm by using a single computer and multiple computers. Different lengths of DNA sequences are stored in database to compare its response time.

Related Work

Different methods had been used to store DNA sequence in Database. To obtaining an image of a mass-storage device [2] the sequence of Genome is used Reverse Engineering code. Reverse engineering files on the mass – storage device is equivalent to design and maintenance specification. Obtaining one full human sequence will be technical challenges. Computers will play a crucial role in the entire process, from robotics to control experimental equipment to complex analytical methods for assembling sequence fragments.

Indexing for large sequence Database uses the n-gram wavelet transformation [3] upon one field and multi-fields index structure under the relational DBMS environment. Results show the need to consider index size and search time while using indexing carefully. Increasing window size decreases the amount of I/O reference and complexity is O (mn).

Indexing and Retrieval for Genomic Database uses CAFÉ indexed scheme[4] and it shows that the indexed approached results in significant, saving in computationally intensive local alignment, and that index-based searching is as accurate as existing exhaustive search scheme and it is better than BLAST.

Dynamic Programming [5,6] has time and space complexity of O(nm) for two strings S and Q of lengths n and m, for database comparisons it will needs matrix of size n * m. Hence for long sequence and large database this method will be not practical in term of both space and time.

Dictionary based indexing [5] for a database of sequence Si (I; 1,2,….n), creates index structure of size n corresponding to database size, predefining query lower bound length (L) to be equal to log(n) assumed. Query with larger length will be partitioned into smaller parts. All substrings of length L mapped to integers using hasing function and for queries larger than L split it into sub-queries, then search each sub-query alone and combine the results. This method indexes all possible strings of a pre-specified length L. Dictionary based index size is larger than the database.

BLAST technique [7] used to find local similarity and not global similarity. It is a string matching tool that has two phases: search all database sequences for a fixed substring length w for exact matching (at i). And using a threshold 't', continue searching after the exact match at both direction, left and right, for distance more than 'i' and before 'i-w' till exceed 't'. It stores pointer for location 'i'. So, space needed is more than the database size.

Suffix array [8] scans database strings using a window (window size w, overlapping amount Δ) and count repetition of all possible k-tuples. It stores result at vector of size σk (σ referred to alphabet chars A,C,G,T). Then it indexes those vectors at hierarchical binary tree and to compare new query with those vectors it uses Edit distance method. It runs 25 to 50 times faster than BLAST. Disadvantage of this method is the allowing of false drops and index size increase linearly with k value.

SST [9,10] scans the database by window w and map results to vector of size 4w. Then hierarchical clusters, non overlapping, built using k-means algorithm, as any new query need to be processed against the database, using cluster mean and neglect clusters that are far away from the new query. Disadvantages of SST are the complexity of calculations, and false clustering.

The specific objective of the proposed distributed algorithm for analysis of DNA sequences are:

1. Develop an optimal storage algorithm (OPTSDNA) and implementation of a distributed bioinformatics computing system for analysis of DNA sequence.
2. Implement them on loosely couple distributed network such as regular local area network.
3. The performance of storage system is compared with sequential approach.

This paper is organized in six sections. Section 3 discusses the Potential applications of the distributed algorithm. The overall distributed architecture and algorithm description is discussed in section 4. Section 5 discusses the Complexity Analysis and Results and Conclusion included in section 6.

Potential Application of the Proposed Distributed Algorithm

This distributed Bioinformatics system developed in this paper could be used for disease detection, criminal forensics analysis and protein analysis. Tetra-let, Pentad-let, Hexed-let repeats formally known as a Tetra-nucleotide, Pent nucleotide, Hex nucleotide. Repeat occurs when four, five and six consecutive nucleotides are repeated within a specific region of DNA sequence. These repeats can occur within or between genes. These consecutive repeats are frequently located in genes that encode transcription factors and which are active in the organism development process. Extensive Tetra-let, Panta-lets, Hex-let repeats are found when a mutation occurs in a gene. This mutation increases the number of occurrences of a particular nucleotide which can lead to a number of neurodegenerative diseases. These diseases include, Huntington's Disease (HD), Fragile X Syndrome, Kennedy's Disease, Myotonic Dystrophy, Spinocerebellar Ataxia Type 1 (SCA1), Dentatorubral Pallidoluysian atrophy (DRPLA),and Fragile X E mental retardation (FRAXE). In Kennedy's Disease, Huntington's disease, Spinocerebellar Ataxia Type 1, and Dentatorubral Pallidoluysian atrophy, the number of triplet repeats is quite small, in contrast to Fragile X Syndrome, Myotonic Dystrophy, and FRAXE, where the number of consecutive repeats may be very large, producing alleles that consist of thousands of repeats. These algorithms can help to detect Tetra-let, Panta-led and Hexa-let repeats in gene sequence, and can also search through DNA sequences to identify most frequently occurring repeats.

Design of Distributed Algorithms for DNA Sequences Analysis

The proposed distributed algorithm (OPTSDNA) is based on client server model. For distributed system, the proposed framework avoids duplicate computations on server machines. The input of our algorithm is DNA sequence. Input is devided into multiple segments. Then the segment generated code

which is we called encode. Then we store in database only encode od DNA segments. The process of storing DNA sequence in Database is divided into following process:

i. Input of our algorithm is a large number of DNA sequence.
ii. DNA segments are broken into N number of parts.
iii. N number of segments are stored in database.
iv. First check N number segments in Database.
v. If not found in database then first segment stored in database and generate code and continue to other segments.
vi. Then only generated code is stored in original database for size calculation.

If found in database then continue to check other segment in database and then only link with code stored in original database table for calculating size.

In our proposed approach, the DNA sequence is broken up in X segments where X = m * p. Here m = number of storage DNA and p = length of storage nucleotide base.

For storing DNA sequence in Database we use three tables. We store put sequence of DNA sequence CAGTCAGTTCCAGAG and CAGTTCCAGGCCTAGCTCAG by the following method. Firstly, input sequence are divided by 4, 5, 6 consecutive alphabets. First consecutive alphabet store in Table 2 and generate code, and then next consecutive alphabets are store in Table 2 for no. of reveals (4, 5, 6). If consecutive alphabets are previously stored in database then next time we can't store the same consecutive alphabets. Then only generated link of input DNA sequence are stored in Table 3. Table1 represents the input DNA sequence.

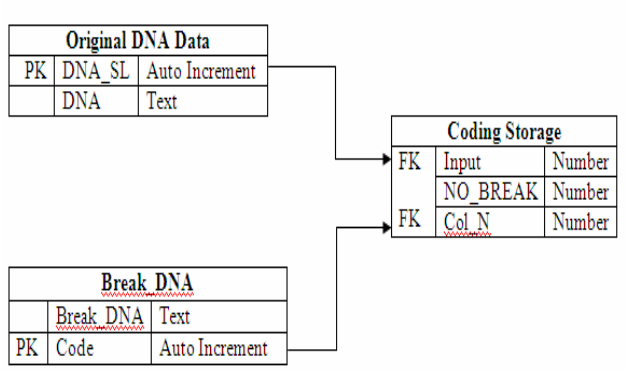

Figure 1. Database Normalization

Table 1. Original DNA Data

DNA_SL	DNA
1	CAGTCAGTTCCAGAG
2	CAGTTCCAGGCCTAGCTCAG

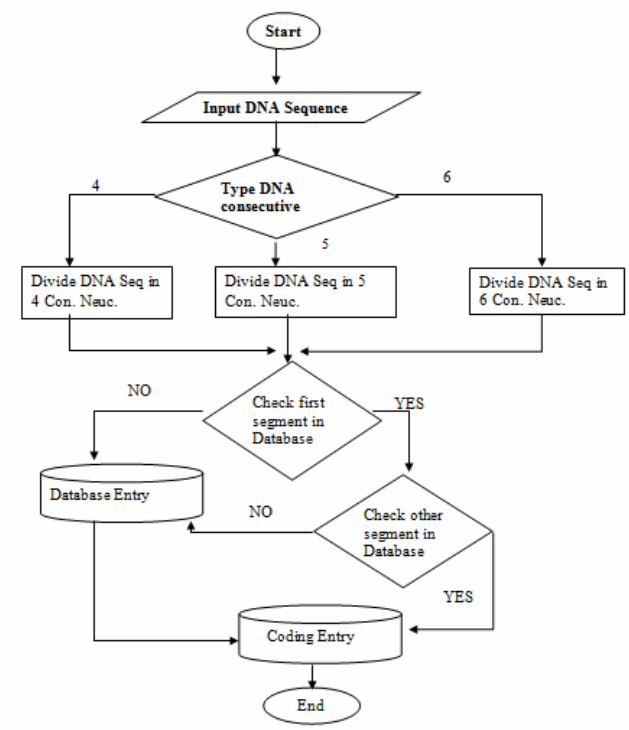

Figure 2. Flow Diagram of Proposed Algorithm

Table 2. Breaking DNA

Break_DNA	Code
CAGT	1
TCCA	2
GAG	3
CAGTC	4
AGTTC	5
CAGAG	6
CAGTCA	7
GTTCCA	8
AGAG	9
GGCC	10
TAGC	11
TCAG	12
CAGTT	13
CCAGG	14
CCTAG	15
CTCAG	16
CAGTTC	17
CAGGCC	18
TAGCTC	19
AG	20

Table 3. Coding Storage

Input	NO_ BREAK	Col_ 1	Col_ 2	Col_ 3	Col_ 4	Col_ 5	Col_ N
1	4	1	1	2	3		
1	5	4	5	6			
1	6	7	8	9			
2	4	1	2	10	11	12	
2	5	13	14	15	16		
2	6	17	18	19	20		

ALGORITHM

OPTSDNA (DNA, X, d, L, T)

1. X = Input of DNA
   DNA = sequence of DNA segment
   d = Division no. of DNA where value is 4, 5, 6
   L = link
2. X/d for create X1, X2, ……Xn
3. if d = 4 then X is divided in X1, X2, …..Xn with four consecutives nucleotide
   3.1 First X1 check in Database
   if X1 in Database then only link with generated code store in Coding Entry Table
   3.2. otherwise X1 store in Database and link with generating code store in Coding
   Entry Table
   3.3 Continue to other segment
4. if d = 5 then X is divided in X1, X2, …..Xn with five consecutives nucleotide
   4.1 First X1 check in Database
   if X1 in Database then only link with generated code store in Coding Entry Table
   4.2. otherwise X1 store in Database and link with generating code store in Coding
   Entry Table
   4.3 Continue to other segment
5. if d = 6 then X is divided in X1, X2, …..Xn with six consecutives nucleotide
   5.1 First X1 check in Database
   if X1 in Database then only link with generated code store in Coding Entry Table
   5.2. otherwise X1 store in Database and link with generating code store in Coding
   Entry Table
   5.3 Continue to other segment
6. End

Complexity Analysis and Result

Our DNA sequence is divided into n parts. So in the first iteration n comparisons take place. In second n-1, then n-2, n-3 do. So 1 + 2 + 3 + ……..+ n comparison are performed. According to the well known Gaussian sum formula these are exactly ½.(n-1).n comparisons. The sorting method has running time O (n^2). The expression O's called Landau's symbol.

Space complexity

$$= m \sum_{i=1}^{t} Dt + m \sum_{j=1}^{t} Ct + n \sum_{k=1}^{s} Ss$$

$$= m \left(\sum_{i=1}^{t} Dt + \sum_{j=1}^{t} Ct \right) + n \sum_{k=1}^{s} Ss$$

m = No. of row in Original DNA Data Table
D = Width of Data per row in Original DNA Data Table
t = No. of data in Original DNA Data Table
C = Width of data of generated code in Original DNA Data Table
n = No. of row in Coding Storage Table
S = Width of per field entry in Coding Storage Table
s = No. of Data field per row in Coding Storage Table

CONCLUSION

This system could be used for storing various sizes of DNA sequences into database. DNA sequences of different lengths were stored by using this algorithm. These sequences varied in size from very small to very large. Our proposed system gives the low CPU cost which is important factor of query performance, even considering today's hardware trend. Advantage of our proposed system is that 1) remove the duplication of DNA data entry in database, 2) require fewer bytes than original data to represent in database, 3) save I/O bandwidth and disk size. For future we will measure our algorithm performance.

REFERENCES

1. Rajita Kumar, Arooshi Kumar, and Sanjuli Agarwal, (2007) A Distributed bioinformatics Computing System for Analysis of DNA Sequences, IEEE.
2. Samer Mahmoud Wohoush, Mahmoud Hasan Saheb (2011) Indexing for Large Database Sequences, International Journal of Biometrics, 5(4) 202-215
3. Robert J. Robbbins, DNA as a Mass – Storage Device, 1994, 1995
4. Hugh E. Williams, Justin Zobel (2002) Indexing and Retrieval for Genomic Databases pp 1-25
5. An Efficient Index Structure for String Databases. Tamer Kahveci Ambuj K. Singh Department of Computer Science, University of California Santa Barbara, CA 93106 , 2001.
6. Fast Dynamic Programming Based Sequence Alignment Algorithm. Nur'Aini Abdul Rashid', Rosni Abdullah, Abdullah Zawawi Haji Talib, Zalila Ali, IEEE, 2006.
7. MAP: Searching Large Genome Databases. T. Kahveci, A. Singh (2003) Pacific Symposium on Biocomputing 8:303-314
8. S. Muthukrishnan and S. C. Sahinalp. (2000) Approximate nearest neighbor and sequence comparison with block operations. STOC '00 Proceedings of the thirty-second annual ACM symposium on Theory of computing, 416-424
9. E. Giladi et al., (2002) SST: An Algorithm for Finding Near-Exact Sequence Matches in Time Proportional to the Logarithm of the Database Size. Bioinformatics 18, 873–877.
10. Jun Liang, Lin Xiao, Di Zhang (2009) An Efficient Approach for Building Compressed Full-text Index for structured Data IEEE, 59-63

*****