Towards Context-Aware DNA Sequence Compression for Efficient Data Exchange

Wajeeta Lohana, Jawwad A. Shamsi, Tahir Q. Syed, Farrukh Hasan
Systems Research Laboratory, Computer Science, National University of Computer and Emerging Sciences, Pakistan
wajeeta@gmail.com,{ jawwad.shamsi, tahir.syed, farrukh.hassan}@nu.edu.pk

Abstract— **DNA sequencing has emerged as one of the principal research directions in systems biology because of its usefulness in predicting the provenance of disease but also has profound impact in other fields like biotechnology, biological systematic and forensic medicine. The experiments in high throughput DNA sequencing technology are notorious for generating DNA sequences in huge quantities, and this poses a challenge in the computation, storage and exchange of sequence data. Computing on the Cloud helps mitigate the first two challenges because it gives on-demand machines through which we are able to save cost and it gives flexibility to balance the load, both computation- and storage- wise. The problem with data exchange could be mitigated to an extent through the use of data compression. This work proposes a context-aware framework that decides the compression algorithm which can minimize the time-to-completion and efficiently utilize the resources by performing experiments on different Cloud and algorithm combinations and configurations. The results obtained from this framework and experimental setup shows that DNAX is better than rest of the algorithms in any context, but if the file size is less than 50kb then one can go for CTW or Gencompress. The Gzip algorithm which is used in the NCBI repository to store the sequences has the worst compression ratio and time.**

Keywords—Bioinformatics; DNA; Gene Compression; Context-aware Compression.

I. INTRODUCTION

Understanding the basic variation within species and between species is a fundamental question bioinformatics addresses. It requires encoding of the nucleotides sequences, collectively called the genome, and encoding genetic information in DNA/RNA (Human_genome n.d.). This sequence can help us diagnose diseases, identify individuals, and develop such tools which can be used to detect if the function of a cell is normal etc. (DNA_sequencing n.d.)

High throughput sequencing techniques are responsible for generating large amount of data. It has been observed that the first genome took roughly 12 years to encode and cost up to a billion USD. The second generation sequencing took a week with 1000 USD cost. If we talk about 3G, it has reduced the cost down to 100 USD and time to days instead of months and years [1].The old techniques like Sanger-based Capillary Sequencing which was used in Human genome project generated data up to 4GB, but now the emerging technologies with massive parallel sequencing are generating data in Terabytes (TB) [2]. The labs where sequencing techniques are being used submit these sequences to archival institutes like INSDC, NCBI, EMBL and DDBJ [3]. Due to this exponentially growing data, the problem of storing and performing analysis on this data is a major concern for the bioinformatics research community.

Efficient disk array are required to store this data, while very powerful machines are required to perform the analysis. We pose ourselves the following research questions, and then investigate their answers through this work:

- While uploading DNA sequences for analysis on cloud, which algorithm is good and can minimize the overall time in given context?
- Can general purpose algorithms like Gzip which are based on LZ and Huffman encoding techniques save overall time better than DNA-based algorithm? Are they also beneficial in terms of memory?

To answer these questions, a framework is proposed which decides

1. Whether it is crucial to compress DNA sequences?
2. Which algorithm should be used?

The algorithms selected for the experiments include: CTW, DNAX, Gencompress, and Gzip

These algorithms are freely available with guidelines on how to use them. The framework utilizes the rules (model) generated from the training data. This training data was generated by performing experiments over different machines by varying the context. The context basically comprises of available RAM, CPU speed, bandwidth, file size, and algorithm. The label is then assigned to training data based on the equation in which four important parameters are included with equal weights. These parameters are time to compress, time to decompress, time to upload, and download. The algorithm which minimizes the overall time is the winner and labels will be assigned accordingly. Then that model is applied on testing data which comprises 25% of the overall experiments performed.

Our research makes the following contributions:

- Hypothesis of compressing DNA sequences based on context (cloud architecture, algorithm, data size, etc.)
- Study of the change of context on the behavior of the DNA sequences' compression.
- Overall time variation with respect to context

/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPSW.2015.89

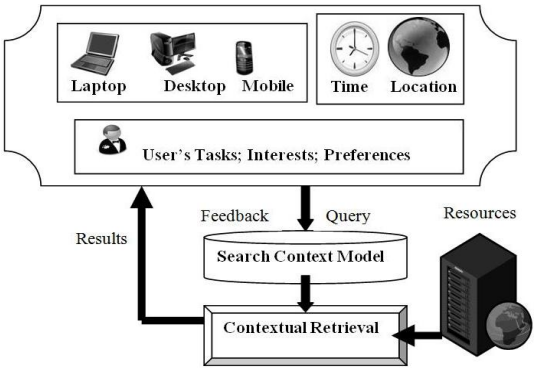

Figure 1 - General Structure of Context aware Compression

- Identifying the fact that uploading on a storage account not only depends on bandwidth but RAM and CPU is also significant.

- Impact of using a general-purpose algorithm like GZip on overall time, RAM usage, and compression ratio.

Context-aware compression is an unexplored area of bioinformatics research. One straightforward challenge it poses is that the behavior of these algorithms is different for a given file size. The file with a small size can take more time than a larger file. This anomaly varies with algorithm to algorithm and the contents in the files because some algorithms look for the exact repeats while others for approximate repeats.

In the related work the compression ratio and compression time has been the main focus while we additionally include the infrastructural cost of analyzing overall compression, decompression, uploading and downloading time on Azure Cloud.

II. BACKGROUND

A. Context Aware Compression

Context Aware compression or Context based compression is basically the technique to compress data based on the context provided. The context can be the available RAM, CPU speed, CPU utilization, bandwidth (in case of uploading and downloading) etc. Using context-aware compression we can determine which algorithm works better compared to others in a given context. This procedure involves the inquiry of current resources available and analysis on historical data to choose the better algorithm.

Figure 1 shows the general process of distributed sequencing computation using a compressed query. The context aware compression framework looks at the current resources and the rules available. These rules can be obtained by analyzing historical data. Using the rules, the framework will come up with the optimum solution. In [4], the authors provide the information regarding the context awareness for handheld devices, where the location and other things are the context. In our case the location of compression is the VM's specification like RAM, CPU and Bandwidth which are simulated by VMware workstation.

B. Bioinformatics Sequences

The DNA (Deoxyribonucleic acid) structure consists of Nucleotides. The arrangement of Nucleotides within the DNA structure has impact on how the cell can function properly. Besides this, it also constitutes the genetic information (DNA_Sequence n.d.). Using DNA sequencing technology we get this information in the form of a string, the particular arrangement of which helps identify how it impacts the cell functioning and which leads towards the cure or identification of a disease[2]. These sequences consist of four bases: adenine, cytosine, guanine, and thymine, usually abbreviated using the symbols A, C, G and T respectively [5]. It has been observed that there is only 0.1% variation among DNA sequences of same species [3][6] .There are three kinds of repetitions found in these biological sequences. The first are the repeats in Long sequence itself, the second is the repetition based on reverse complement like 'A' always having a pair with 'T', and 'C' with 'G'. By using this characteristic we can identify such repeats. The third kind of repeat is based on mutation because sequences belonging to same species are 99.9% the same [6].

III. RELATED WORK

The compression of DNA sequences due to its specific characteristics is found to be the critical task in the data compression field [7]. There are many algorithms for text compression but they do not give good compression ratio for DNA sequences. Among the tools which compress text, such as bzip2 which uses burrows-Wheeler transform+Huffman+Move-to-Front for compression, gzip which utilizes huffman+ Lz and others have failed to give good compression ratio [3].The Huffman which is based on calculating the frequency of symbols does not give good results because the four symbols (A, G, T, C) do not have different probabilities [8]. CTW is well known for compression of DNA sequence but it has been observed through experiments that although it has good compression ratio compared to few other DNA sequence specific algorithms like DNAX, it consumes more time in decompression procedure than other algorithms [9] like DNAX and GenCompress.

Thus in the field of bio-informatics different algorithms have been proposed, based on both greedy and dynamic approaches [10].

The compression technique refers to the method of transferring the data load from network to CPU and memory. Using compression techniques one can save the memory but these techniques require computational cost which involves RAM too. This space saving procedure follows the basic concept of how possibly to write the next substring in such a way that it refers to the existing substring with minimum bytes. With this basic concept many categories of compression were introduced, both for general purpose and DNA-Sequence based data.

Broadly speaking about the DNA-Sequence based compression techniques there are basically two modes;

Table 1 - Algorithms: Encoding techniques and Methodology

Algo Name	Methodology	Encoding (Repeats)	Encoding(NonRepeats)
BioCompress	Detects exact and reverse complement repeats	Fibonacci coding to encode the length and position of it is previous location	2bpc
Bio-compress2		Same as BioCompress	Order-2 arithmetic coding
Cfact	Searches longest exact repeats in two passes. First pass suffix tree second pass encoding	Lz	2bpc
Gencompress	For Approximate repeats uses edit distance operation and 2 integer	Hamming distance in gencompress1 and edit distance in gencompress2	Edit distance operation for mutation and pair or integers
DNAcompress	Two pass algo, uses Pattern hunter approximate Repeats including complement palindrome	(l,i,j) (e,i,j) For Approximate repeat encoding	Lz
DNAC	Four phase algorithm	Fibonacci encoding	
DNAPack	Dynamic programming to search repeats	Hamming distance	Order-2 arithmetic coding or context tree weighting or naïve 2-bits
CTW+LZ	Context tree weighting		
DNAX	Exact Repeats and Reverse Complement	Uses information in Approximate repeats	Arithmetic coding
XM	Statistics		

horizontal mode and vertical mode [11]. In Horizontal mode there is only one sequence which is compressed by referring to substring within the same sequence. In vertical mode, another sequence is used for the reference to compress.

A. Overview of Horizontal mode algorithms

Horizontal mode is further divided into categories based on substitution, statistics, grammar and table comparisons [3] [12].

The first algorithm to compress the DNA sequence which was introduced was "BioCompress" [11]. It first finds the exact repeating sequence and then reverse complement of the sequence and then stores them in a tree. Afterwards it encodes these repeats by their frequency and preposition of the previous occurrence. It then uses 2 bits to encode the remaining region. A revised version of Biocompresss "BioCompress2" uses the same technique except it encodes the remaining string using order-2 arithmetic coding [11]. [13], another algorithm which searches for the long repeats in first pass and builds suffix tree while in second pass it performs encoding using lz.

Gencompress [14], is also one of the well-known substitution algorithms. It searches the optimal prefix of unprocessed substring which has approximate match in processed substring to encode it efficiently. It limits the search by putting constraint at the edit operation using a threshold value. It uses edit operation to refer the approximate repeats. These edit operations are basically insert, delete and replace. Gencompress has also two versions. Gencomrpess-1 uses hamming distance to encode while in Gencompress-2 the edition distance is being used [3]. DNA Compress [15] achieves average 13.7% compression which is faster than other algorithms [16]. It finds all approximate repeats by using Software Pattern Hunter. To encode both approximate and exact repeats it uses LZ.

We have also used DNAX [17] algorithm in our work. DNAX unlike Gencompess works on the exact repeats. The RAM usage of the Gencomrpess is high due to the fact that it looks for the approximate repeats and eventually has good compression ratio than others. If we talk about the time taken for compression, DNAX performs better than Gencompress. It follows the strategy of encoding the exact repeats only, but to encode them it uses the information from approximate repeats. When no match is found, arithmetic coding is utilized.

In 2004, a revised algorithm based on DNAX was published by the name of DNAC [3]. It is four phases based algorithm. It constructs suffix tree in first phase to find exact repeats, in second phase, using dynamic programming, exact repeats are approximated to partial repeats. In third phase the optimal non-overlapping repeats are extracted. In fourth phase it uses Fibonacci ending to encode repeats. DNAPACK [18] gives better results than Gencompress, Ctw and DNACompress. It uses hamming distance for repeating substrings while for non-repeats it uses one of three methods (order-2 arithmetic, context tree weighting, and naïve 2bits per symbol). Li et al.'s DNA-COMPACT [2], which can compress DNA with or without reference, improves the compression ratio. It is a two pass algorithm. In the first pass it finds the repeats and complementary palindromes while in the second pass the remaining sequence is coded. In that the contextual model is improved over the XM [19] as XM uses Bayesian averaging which can generate biased results. Instead they use logistic regression.

The other category of horizontal compression is statistics based, where encoding is based on predicting the probability distribution of the symbol to be encoded. The model of sequence is generated based on this distribution. Good compression can be obtained if the model provides high probability of the next symbol's actual value. The algorithms in these categories are XM [19], CDNA [20] and ARM [21]. Among these three, XM is the popular one and it has competitive compression ratio. These three techniques require more computation due to the models that need to be generated, therefore, practically these are usable for small sequences only [3].

In the substitution-statistics based category, algorithms such as Ctw+lz [22], offline [7] have been developed. Grammar-based algorithms construct context free grammar to represent input data. That CFG is then encoded to binary after converting into streams. One algorithm in this category is DNASequitur [23]. Table 1 contains the summary of algorithms and includes the encoding techniques used by them in case there are repeats within the sequence and if no repeats are found. It also shows how each algorithm works.

Unfortunately, the source codes of all algorithms are not available. Therefore, many researchers have used standard benchmark files to relate their work. We were able to get code of DNAX and Gencompress and started our work with that. Besides, we incorporated Gzip Deutsch et al. [24] and CTW willems et al. [25] to give comparative analysis over both general text and DNA specific algorithms.

While work regarding context aware compression of DNA sequences is not forthcoming in literature, it exists for simple text algorithms. For DNA compression, the related work that analyzes the trade-off between compression time and memory does exist. It is summarized as follows:

In Wandelt et al. [1], a reference genome is used for compression with fine-tuning the trade-off between compression time and memory. The idea is to find out longest prefix-suffix match by mapping with reference genome and place entries for reference in the file to be compressed. Following are the three methods used by them to achieve better compression.

- Block-change entry BC(i): next entries are encoded with respect to reference block i.
- Relative match entry RM(i,j): The input matches the reference block at position i for j characters.
- Raw entry R(s): A string s is encoded raw (for instance if there is no good matching block).

The 1000 genome project is compressed using this approach. It has been observed that compression ratio is 1:400 and by increasing block size more efficient results are achieved.

In Krintz et al. [26], the authors have proposed an Adaptive Compression Environment (ACE), which automatically and transparently applies compression on stream at TCP/IP level to improve transfer performance. ACE uses two technologies, Open Runtime platform (ORP) [27] from Intel Microprocessor Research Lab (MRL) and Network Weather Service (NWS) [28]. ORP is used for the decision purpose while NWS for monitoring resources and measurement in timely manner. Network sensors are the light weight processes which execute on client device to forecast the impact of last decision made for compression. When user wants to know the future impact of compression then ACE uses these values to decide whether to compress by using Bzip, LZO and Zlib or not to compress. ACE decides on last samples of compression ratios and if those are unavailable because compression was not applied due to some reason (like CPU load is not enough and Bandwidth is high) ACE will consider CPU load and bandwidth for its estimation.

In Wiseman et al. [29], the authors have considered the network transmission rate and processor resources to assess compression effectiveness. The algorithm and techniques have been incorporated in a middleware ECho (Eisenhauer n.d.). This middleware is based on Message Passing Interface (MPI).

"Reducing speed" is measured when blocks are compressed while "receiving speed" is measured when compressed blocks are received. The middleware has been tested for commercial and scientific dataset. A dataset is transferred within 29.138 seconds without compression. On the contrary compressed database takes 10 second (Compression takes 60% of total time).

B. Overview of Vertical mode algorithms

The algorithm/tool included in this category is; DNAZip [30].The goal of this tool is to compress database to 4MB so that it can be attached in email. It needs reference genome (~3GB) and reference SNP map (~1.2GB). Another Approach G-SQZ Tembe et al. [31] uses Huffman-coding to compress data without altering the sequence. In Daily et al. [32], a data-structure is purposed to efficiently compress data by mapping on reference sequence. In kuruppu et al. [6] COMRAD is modified with iterative dictionary. Coil [21] is another algorithm based on idea of edit-tree coding, it has high compression ratio but not good timings.

IV. Experimental Setup and Framework Design

A. Experimental Setup

Framework is based on deciding which algorithm can be chosen based on the current resources available. The decision is taken from the model learnt from the experiments performed. The experiments have been conducted at different machines. There were mainly three machines; an i5 with 6 GB RAM and 2.4GHz processor, a core 2 duo machine with 2.0GHz Processor and 3GB RAM, a VM at Windows Azure cloud with 2.1GHz AMD processor with 3.5GB RAM. The first two machines were used for experiments by installing VMware workstation in order to create controlled environment. The parameters for context such as RAM and Bandwidth were simulated on these machines. Besides this, a storage account (SAAS) was used to store the uploaded files in the form of Blobs (Binary large object). A container is created and these files are uploaded as BLOBs.

The compression along with uploading is performed at VMs on these two machines while downloading from storage account and decompression is performed at cloud. By doing this, we were able to find out which algorithm was good for a given scenario. From experiments, it was clear that uploading data at cloud was not only dependent on bandwidth but the processor speed and RAM also mattered, while the size of the compressed file remains unchanged.

For experiments, we downloaded sequences from NCBI, and uploaded to cloud via ftp [33]. These are compressed with gz and most of the sequences are of bacteria. After decompression, the file contains multiple sequences along with text. We separated the sequences and removed the extra text so that single sequence experiments can be carried out smoothly. The seven files from benchmark standard dataset are used by

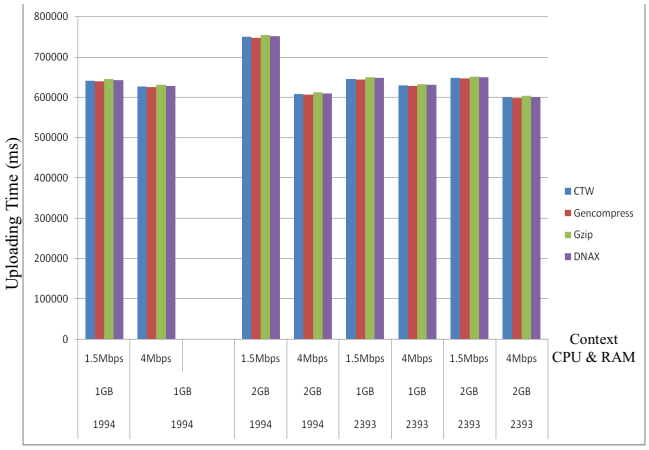

Figure 2 - ***Graphical Representation of Uploading Time in different Context***

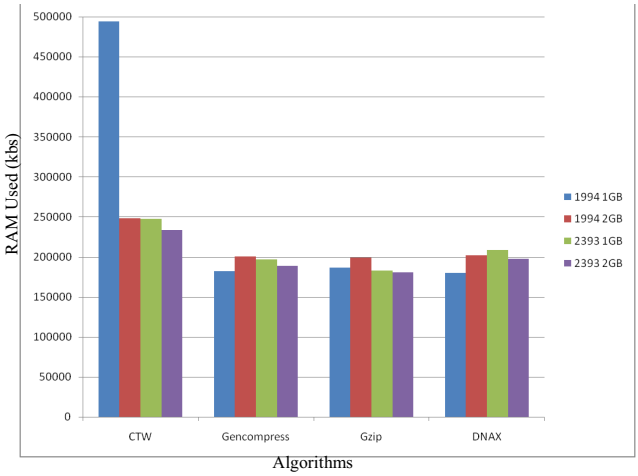

Figure 3- ***Graphical Representation of RAM used***

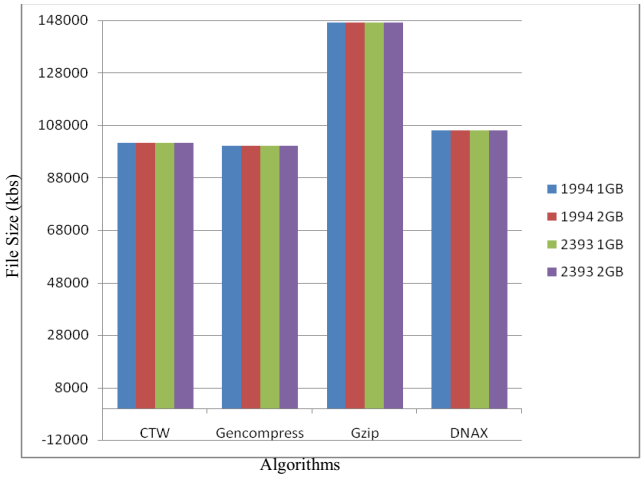

Figure 4 - Graphical Representation of Compressed File Size

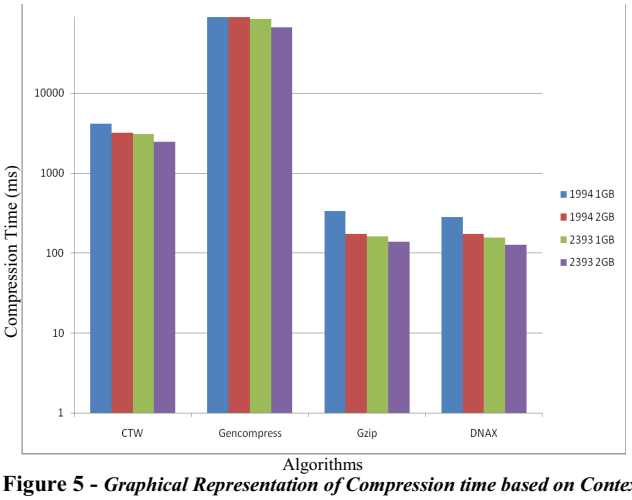

Figure 5 - ***Graphical Representation of Compression time based on Context***

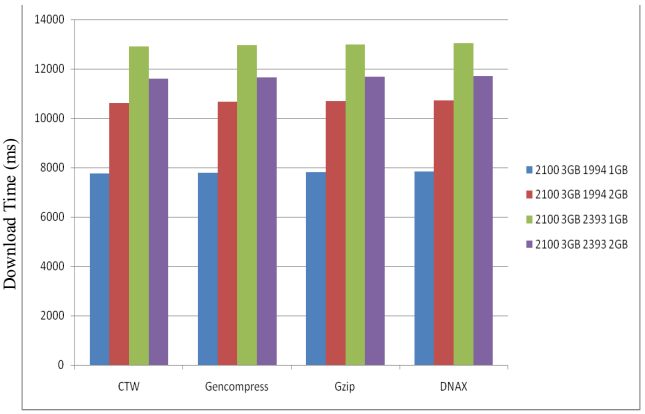

Figure 6- ***Graphical representation of Download Time***

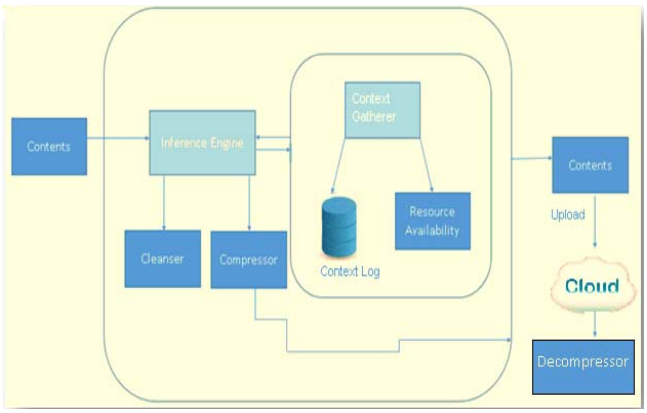

Figure 7 - Bioinformatics framework

most of the authors in their work [18]. A total of 132 files are used in the experiments with different file sizes.

B. Experiments

In this section we going to discuss about the experiments performed on different machines for dependent variables (Compressed file Size (bytes), RAM used (In bytes), and Time (Uploading, downloading, Compress, Decompress) (Milliseconds)) with different context is given which shows how algorithms are behaving when context changes. In Figure 2, the uploading time is depicted. On average, it has been

observed that by increasing all the three parameters of the contexts i.e. RAM, Bandwidth and CPU speed, the uploading time can be improved in general.

Figure 3 shows that when DNAX and Gencompress are compared, then DNAX is good when RAM and CPU are low, while for the rest of cases Gencompress is better. Slight variation in these results exists, as RAM usage cannot be predicted easily based on the context. Figure 4, we can observe that DNAX is fine in compression ratio after Gencompress and CTW with benefit over the time variable. The context doesn't change the compression ratio because the algorithms work according to their logics which are based on certain thresholds. The threshold is what changes the RAM consumption and time of compression.

In Figure 5, we can analyze that compression time for Gencompress is bad due to its edit distance operation to find the approximate repeats in order to minimize the compression ratio. While looking at the given context it can be observed that the change in RAM only does not change the compression time for Gencompress while change in CPU brings a little change. For CTW, Gzip and DNAX CPU are important to some extent. DNAX is taking less time than others.

Figure 6 shows the download time for each algorithm. There is a slight difference between these algorithms based on CPU usage and RAM availability of Cloud. These differences are nearly 27 Ms to 45Ms between algorithms. The decompression time were also observed it was noted that DNAX has foremost least decompression time than rest of the algorithms.

C. *Labeling of the Training data*

In this step the training data is which has been gathered from experiments is going to be used for labeling. In table 2, different variables with weights are listed. Labeling is actually deciding which algorithm is good in a given context. Let say there are in same context there were four algorithm which is best if we consider Time only (100% weight, or RAM only or any combination of TIME and RAM listed in Table 2. Below is the equation that were used for this labeling.

$$E = w * (Compresion_{time}) + w * (Decompression_{time}) + w * (Upload_{time}) + w * (Download_{time}) + w * (RAM_{USED})$$

Using above equation, label were assigned based on which algorithm is giving less value for this equation. Given a context the algorithm which is utilizing the less resources is selected to label. After this labeling, rules are generated so that create model that learn such pattern.

D. *Bio-informatics compression Framework*

The proposed framework in figure 7, is composed of different components. These components work together to provide an optimal solution based on different contexts. The parameters which build the context for compressing data include: Size of file, Algorithm, Bandwidth, CPU Speed, and Memory Available.

The above mentioned factors affect the RAM utilization of an algorithm and the time taken by it for compression, decompression, uploading and downloading. Different components of this framework are: Inference engine, which decides which algorithm should be chosen for compression, the Compressor, which uses that algorithm to compress the file, the Context gatherer that collects the information regarding the resources available and applies the rules. These rules are generated through Decision tree induction using methods CHAID (Chi-squared Automatic Interaction Detector) and CART (Classification and Regression Trees). These rules are supposed to be model which generated by learning algorithm applied on training data. Above information will be used by the inference engine to decide the algorithm. Extra information is cleansed by the Cleanser. At the cloud VM that file is then downloaded and decompressed.

V. RESULTS

In this section the main results regarding both CHAID and CART are going to be discussed. It has been observed that methods give approximately the same results but CART was found to be more effective as the problem to predict the algorithm is basically that of the predication of category based on continuous or categorical variables. During the learning phase it was observed that there were no records where Gzip was used as label, which means that Gzip is not good while considering the overall time.

Once the rules are generated, they are incorporated in framework for testing. As mentioned earlier, this testing data was separated in the starting. These are 1056 rows on which these rules have been applied. Which includes 33 files so 33*32 (with different context) =1056 rows.

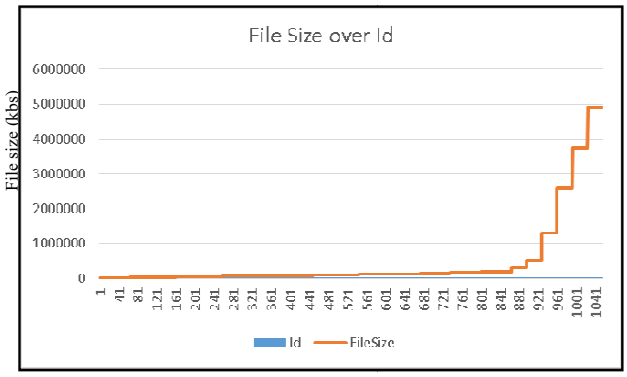

Figure 8 – File Size w.r.t Row Id / Number of Records

Figure 8, showing the relationship between the file size and number of rows. The below relationship will be used in next sections to show the time consumption and RAM usages in different contexts.

A. *Result with rules generated by CHAID for time*

The training data which is trained by equation with equal weights was assigned to CHAID for rules generation. When these rules were applied to the testing data, following results in the form of chart were observed.

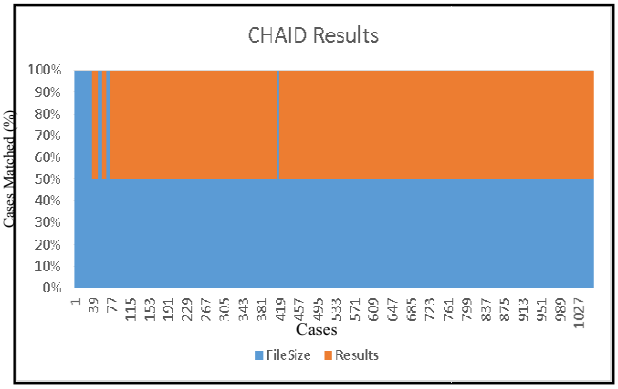

Figure 9 – CHAID Results for time (100% weight) after applying rules (Validation)

Figure 9 shows the results obtain by CHAID. By doing this the chart with gaps were found which shows when CHAID rules fail on a given file size. In above results it is indicated that when the file is less than 50kb and RAM is less than 2GB with CPU speed less than or equal to 2393, the rules could not be validated. As per training, the gaps actually indicate that the label of Gencompress is missing. As CHAID uses the methodology based on the variable which splits more so the majority of splits labeled the rows with DNAX algorithm with a fraction of CTW. Here is the accuracy given for CHAID;

Accuracy= Cases Matched/TotalCases

Accuracy=0.946

The detail analysis over the RAM and CPU with file size is given which shows how the results are varying based on the rules generated.

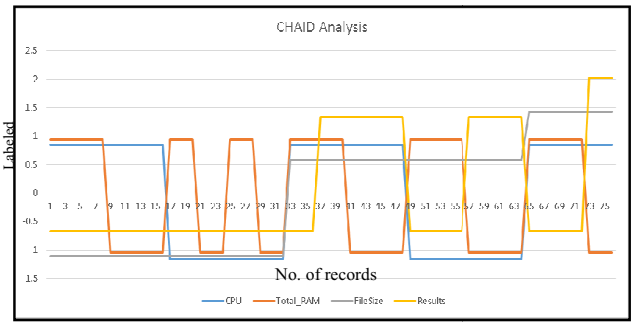

Figure 10 – CHAID Analysis based on Context

In Figure 10, by using the normalized values for CPU, TotalRAM, file size and results are being depicted. The observation shown is for the file size less than 50kb with different RAM and CPU. The yellow line represents the result. If it is less than 0, it specifies that the case was mismatch while above zero indicates it was labeled as per training. Further it is clear from results that in the starting when the size of file is very small then CHAID method failed to predict the correct algorithm. Gradually when the file size increases CHAID methodology found DNAX is the best for compression.

B. Result with rules generated by CART for time

In this section the results regarding the rules generated by CART are going to be described. Like CHAID method CART method gives priority to the DNAX on the whole. But as discussed, CART is used when there is predication involved because it identifies the resemblance within the class and generates binary tree accordingly. By doing so good results were obtained because the cases related with Gencompress algorithms were also identified.

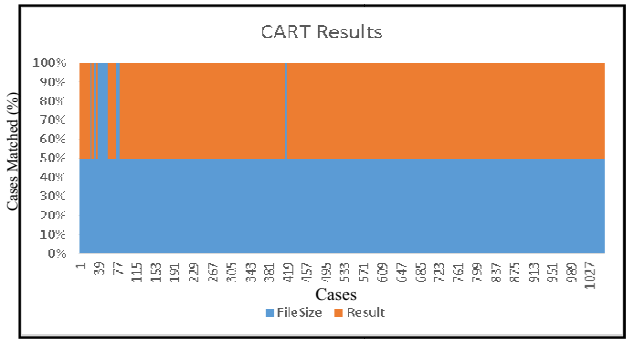

Figure 11 - CART Results for total time (100% weight) after applying rules (Validation)

In Figure 11 the rules are identified for files with file size less than 50kb. These were missing in the CHAID results. Still gaps are there because of the slight difference of overall timing between CTW and Gencompress. Here is the accuracy given for CART;

Accuracy= Cases Matched/TotalCases

Accuracy= 0.962

The detailed analysis over the RAM and CPU with file size is given for CART.

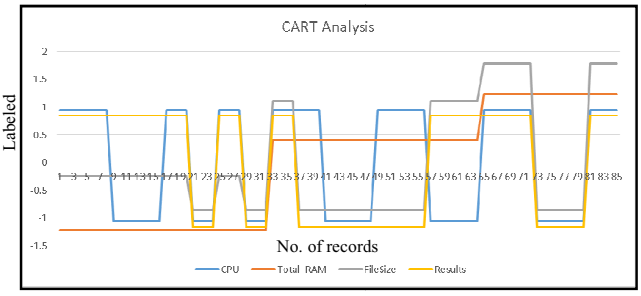

Figure 12 – CART Analysis based on context

In Figure 12 the normalized results are shown for first 86 records. The results line in yellow indicates that for first 17 files it was fine but when the suddenly CPU and RAM both get low it preferred Gencompress as label while originally it was CTW. The model overall indicates that when the file size is less than 30kb the Gencompress should be preferred. Overall again results shows the DNAX is good in timing with little compromise at compression ratio. Gzip because of overall bad timing is not included in the training data so eventually is it is not considered in results also.

C. Result with rules generated by CHAID for RAM

In this section results are given based on the rules generated for RAM using CHAID. It has been observed that CHAID method is found to be better than CART for RAM usage. But unfortunately the results are not good.

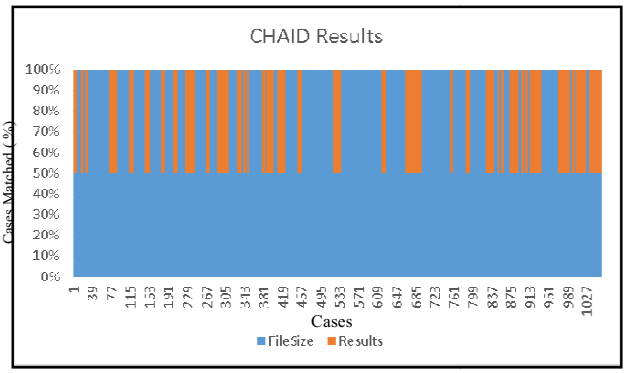

Figure 13 - CHAID Results for RAM (100% weight) after applying rules (Validation)

Figure 13, the gaps show that the files didn't get the correct labels that were assigned using rules generated on training data for RAM usages. These gaps are more than those found in compression time may be due to the fact that the RAM consumption also depends on CPU usage which is not consistent.

The Accuracy for this model is given as under:

Accuracy= Cases Matched/TotalCases

Accuracy= 0.3614

The detail analysis over the RAM and CPU with file size is given for CHAID.

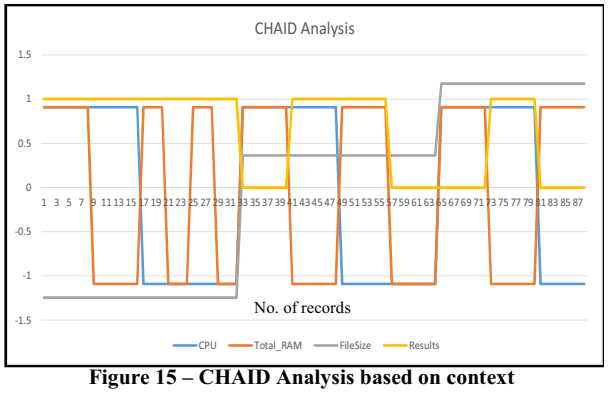

Figure 15 – CHAID Analysis based on context

In Figure 14, the analysis on CHAID based on the context is graphed. For the first 87 records the graph shows that when files size along with CPU and RAM is also increases then again the results gets poor. The change in results doesn't seem as such logical as in case of compression time. As mentioned the accuracy is 36% when the rules are applied on testing data.

D. Result with rules generated by CART for RAM

In this section the results on the rules generated using CART are analyzed. As discussed, CART doesn't give good results same as CHAID and there is only difference of 3%. Overall the classification for RAM usage is not found to be good that is the reason why rules that were applied to training

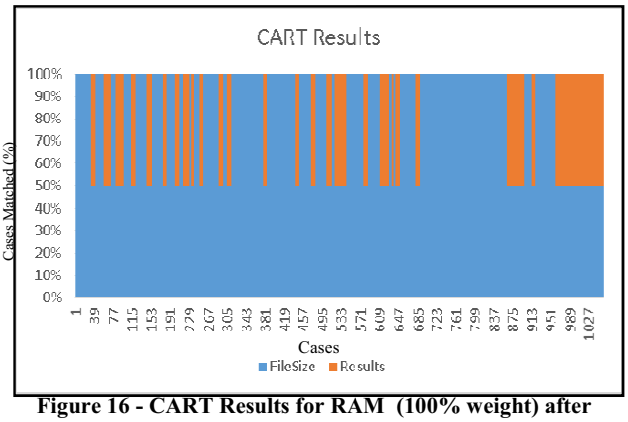

Figure 16 - CART Results for RAM (100% weight) after applying rules (Validation)

data doesn't produced effective results for testing data.

In Figure 15, the gaps indicate that rules could not identify the label at the start which was done by CHAID method. The gaps get large when size is greater than 200Kb. Accuracy for this model is given below;

Accuracy= Cases Matched/TotalCases

Accuracy= 0.3342

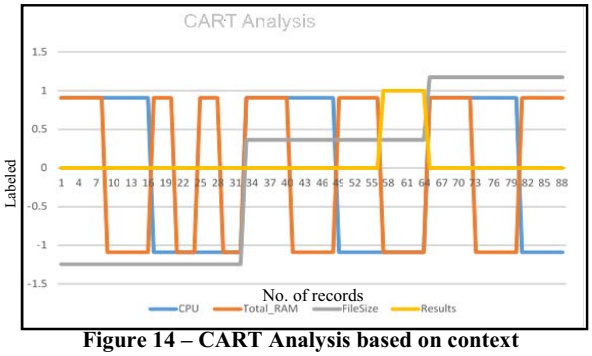

Figure 14 – CART Analysis based on context

In Figure 16, the analysis is given for first 88 records which shows that at the start CART method couldn't find the label and suddenly when CPU and RAM get low, somehow it gives poor results. On average it is noticed from experiments also when CPU speed is increased and RAM is increased then RAM utilization is high.

E. Overall Results with accuracy

In this section the overall results with accuracy are given in Table 2. It gives the summarized accuracy based on the

combinations tried for dependent variables like RAM, and Time with different weights.

Table 2 – Accuracy of generated Rules

Method	*Weight*	*Var1*	*Var 2*	*Var 3*	*Accuracy*
CRT	100	RAM	N/A	N/A	33.50
CHAID		RAM	N/A	N/A	36.14
CRT	100	TIME	N/A	N/A	96.20
CHAID		TIME	N/A	N/A	94.60
CRT	100	Compression Time	N/A	N/A	98.48
CHAID	100	Compression Time	N/A	N/A	98.48
CART	60:40	RAM	TIME	N/A	35.23
CHAID		RAM	TIME	N/A	35.42
CART	40:60	RAM	TIME	N/A	44.32
CHAID		RAM	TIME	N/A	39.77
CRT	70:30	RAM	TIME	N/A	35.23
CHAID		RAM	TIME	N/A	35.42
CRT	30:70	RAM	TIME	N/A	42.80
CHAID		RAM	TIME	N/A	41.29
CRT	80:20	RAM	TIME	N/A	30.11
CHAID		RAM	TIME	N/A	35.42
CRT	20:80	RAM	TIME	N/A	42.80
CHAID		RAM	TIME	N/A	38.64
CRT	90:10	RAM	TIME	N/A	33.90
CHAID		RAM	TIME	N/A	33.90
CRT	10:90	RAM	TIME	N/A	45.83
CHAID		RAM	TIME	N/A	36.55
CRT	50:50	RAM	Compression Time	N/A	38.64
CHAID	50:50	RAM	Compression Time	N/A	35.23
CRT	33:33:33	RAM	Compression Time	Upload Time	22.54
CHAID		RAM	Compression Time	Upload Time	27.65
CRT	20:40:40	RAM	Compression Time	Upload Time	43.94
CHAID		RAM	Compression Time	Upload Time	37.50
CRT	40:40:20	RAM	Compression Time	Upload Time	45.45
CHAID		RAM	Compression Time	Upload Time	38.26
CRT	40:50:10	RAM	Compression Time	Upload Time	42.61
CHAID		RAM	Compression Time	Upload Time	39.77

In Table 2, the different combination were tried to find out the relationship between RAM usage and Time. Overall the results are not good when both RAM usage and Time are considered because overall if the accuracy of RAM usage is considered it goes up to 36% only, which eventually means that RAM used cannot be predicted based on given context. The learning model should consider other parameters such as CPU usage. It has been observed too that when CPU usage is high then RAM usage also gets high.

We observe that on the whole DNAX is the winner with respect to RAM usage and compression time. The strategy it uses is to find the exact repeats instead of approximate due to this fact it consumes less time than Gencompress which looks for the approximate repeats and find edit operation. On the other hand Gzip uses the dictionary based approach while CTW uses the statistics based approach. The statistics based approach on Markov models is also utilized in the XM algorithm for DNA. But we found that although CTW is good for compression ratio and its compression takes less time than Gencompress but when it comes to decompressing the sequence, on average CTW performs the worst.

Regarding context aware compression, RAM usage for GZip is low on average and CTW consumes more memory. The RAM used by the algorithms also depends on CPU usage which we have observed through experiments. We observed that in multiple cases when CPU usage is greater than 30% the RAM usage got double. The classification for RAM usage is not so good due to the fact that it is nearly same for all algorithms. Ideally the general purpose algorithms take less memory when RAM and CPU get increased but for DNA specific algorithms the RAM usage is dependent upon CPU speed. So their behavior in terms of RAM usage is slightly different which generates poor classification.

The compression time is although predictable and discriminative as a classification variable. On the whole it suggests DNAX but the overall time for DNAX is low whether it is Decompression or Download or Compression. For upload Gencompress on average is good with nearly difference of 5000 ms as compared to DNAX because of the compression ratio of DNAX. It has been observed also that when RAM get increased for same CPU, all algorithms are providing good upload and compression time but increase in CPU yields better results.

Using classification, the generated rules indicate that for small size Gencompress or CTW can be used but not otherwise because when file size increases the impact of overall time is also increased. It has been also analyzed that with compromise on RAM, time can be saved using compression. For large files up to MBs DNAX provides good results in terms of timing while Gencompress for small files can provide significant results. If file size is large then Gencompress and CTW are not good because CTW has poor decompression time. Gzip can be used by compromising on space saving.

VI. CONCLUSION

A DNA sequence is different from general text and therefore requires different compression strategies than text. Due to the limitations of code and the software development practices followed for these algorithms only the algorithms listed in Table 1 were found which suited to DNA sequences. The other two algorithms are general purpose text algorithm, i.e. CTW and Gzip. It has been observed in DNA compression

research that CTW is giving good compression ratio. So we also included it in our research.

From the results we can conclude that context-aware compression of DNA sequences is not deterministic because of several reasons. If we train data over individual dependent variables (TIME, and RAM_USED) separately and test over the testing data then we get results up to 95%. On the contrary, training by assigning different weights like 40 /60 or 70/30 or 90/10 and so on provides results up to max 45%. The reason both dependent variable yields poor results is, the RAM used for algorithms CTW and GenCompress is approximately same and RAM usage varies based on the CPU Usage also which is not deterministic because of sudden background processes. Besides the time to upload also depends upon the CPU because to upload the file at Azure storage account it first requires the file to be converted into a continuous stream and then uploaded as BLOB.

Directions for future work could be to improve the Eq. 1, to identify the impact with CPU usage also with different combination as mentioned in Table 8. In this research File size were restricted to 10MB but for future work even more long sequences will be targeted. Along with we will work on how vertical sequences can be compress using horizontal algorithms by measuring their tradeoffs. The context at cloud could be changed to analyze the impact at decompression and download time as in current research only client context was changed. Besides the compression of multiple sequences, that is, vertical sequences using horizontal algorithm vs. the vertical algorithms can also be considered in future research.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] Wandelt, Sebastian , and ulf Leser. "Adaptive efficient compression of genomes." Algorithms for Molecular Biology, 2012.

[3] Pinghao Li, Shuang Wang,Jihoon Kim,Hongkai Xiong,Lucila Ohno-Machado,Xiaoqian Jiang. "DNA-COMPACT: DNA COMpression Based on a Pattern-Aware Contextual Modeling Technique." PLoS ONE , 2013.

[4] S. Bakr, Nour, and Amr A. Sharawi. "DNA Lossless Compression Algorithms: Review." American Journal of Bioinformatics Research, 2013.

[5] Abowd, Gregory D., Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. "Towards a better understanding of context and context-awareness." In Handheld and ubiquitous computing Springer Berlin Heidelberg, 1999: 304-307.

[6] Adjeroh, Don , Yong Zhang, Amar Mukherjee, Matt Powell, and Tim Bell. "DNA Sequence Compression using the Burrows-Wheeler Transform." Bioinformatics Conference, 2002. Proceedings. IEEE Computer Society . 2002. 303 - 313.

[7] kuruppu, Shanika, Bryan Beresford-smith, Thomas Conway , and Justin Zobel. "Iterative Dictionary Construction for compression of large DNA Data Sets." Computational Biology and Bioinformatics, IEEE/ACM Transactions on, 2011

[8] A. Apostolico, S. Lonardi. "Compression of Biological Sequences by Greedy Off-line Textual Substitution." Proceedings Data Compression Conference, 2000: 143–152.

[9] Toshiko Matsum, Kihiko Sadakane,Hiroshi Imai,Takumi Okazaki. "Can General-Purpose Compression Schemes Really Compress DNA Sequences?" Currents in Computational Molecular Biology, 2000: 76-77

[10] Sato, Hisahiko, Takashi Yoshioka, Akihiko Konagaya, Tetsuro Toyoda. "DNA data compression in the post genome era." Genome Informatics Series, 2001: 512-514.

[11] Minh Duc Cao, Trevor I. Dix,loyd Allison,Chris Mears. "A Simple Statistical Algorithm for Biological Sequence Compression." Data Compression Conference, 2007. DCC '07 , 2007: 43 - 52.

[12] Grumbach , Stéphane , and Fariza Tahi. "Compression of DNA sequences." CiteSeer, 1994.

[13] Giancarlo, Raffaele, Davide Scaturro, Filippo Utro. "Textual data compression in computational biology: a synopsis." Bioinformatics 25, no. 13, 2009: 1575-1586.

[14] Rivals, Eric, Max Dauchet, Jean-Paul Delahaye, Olivier Delgrange. "Fast discerning repeats in DNA sequences with a compression algorithm." In Proc. Genome Informatics Workshop, 1997: 215-226.

[15] Xin, Chen, Sam Kwong, and Ming Li. "A compression algorithm for DNA sequences." Engineering in Medicine and Biology Magazine, IEEE, 2001.

[16] Chen, Xin, Mang Li, Bin Ma, and John Tromp. "DNACompress: fast and effective DNA sequence compression." Bioinformatics, 2002.

[17] Timothy , W, J White, and Michael D Hendy. "Compressing DNA sequence databases with coil." BMC Bioinformatics, 2008.

[18] Rastero, Giovanni Manzini and Marcella. "A simple and fast DNA compressor." Software: Practice and Experience, 2004: 1397–1411.

[19] Behzadi, Behshad, and Fabrice Le Fessant. "DNA Compression Challenge Revisited: A Dynamic Programming Approach." In Combinatorial Pattern Matching, 190-200. Springer Berlin Heidelberg, 2005.

[20] Cao, Minh Duc, Trevor I. Dix, Lloyd Allison, and Chris Mears. " simple statistical algorithm for biological sequence compression." In Data Compression Conference, 2007: 43-52.

[21] Lewicki, Thomas Hill and Paul. Statistics: Methods and Applications. 2006.

[22] Allison, Lloyd, Timothy Edgoose, and Trevor I. Dix. "Compression of strings with approximate repeats." In ISMB, 1998: 8-16.

[23] Matsumoto, Toshiko, Kunihiko Sadakane, and Hiroshi Imai. "Biological sequence compression algorithms." GENOME INFORMATICS SERIES (2000), 2000: 43-52.

[24] Cherniavsky, Neva, and Richard Ladner. "Grammar-based compression of DNA sequences." DIMACS Working Group on The Burrows-Wheeler Transform 21, 2004.

[25] Deutsch, L. Peter. "GZIP file format specification version 4.3." 1993.

[26] Willems, Frans MJ, Yuri M. Shtarkov, and Tjalling J. Tjalkens. "The context-tree weighting method: Basic properties." Information Theory, IEEE Transactions , 1995: 653-664.

[27] Krintz, Chandra, and Sezgin Sucu. "Adaptive on-the-fly compression ." Parallel and Distributed Systems, IEEE Transactions on , 2006.

[28] Cierniak, Michał, Guei-Yuan Lueh, and James M Stichnoth. "Practicing JUDO: Java under dynamic optimizations." PLDI '00 Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation . 2000. 13 - 26.

[29] Wolski, Rich, Neil T Spring, and Jim Hayes. "The network weather service: a distributed resource performance forecasting service for metacomputing." Future Generation Computer Systems, 1999: 757–768.

[30] Wiseman, Yair , Karsten Schwan, and Patrick Widener. "Efficient end to end data exchange using configurable compression." ACM SIGOPS Operating Systems Review, 2005.

[31] Christley, Scott, Yiming Lu, Chen Li, and Xiaohui Xie. "Human genomes as email attachments." Bioinformatics, 2009.

[32] Tembe , Waibhav , James Lowey, and Edward Suth. "G-SQZ: compact encoding of genomic sequence and quality data." Bioinformatics, 2010.

[33] Daily , Kenny, Paul Rigor, Scott Chirstley, Xiaohui Xie, and Pierre Baldi. "Data structures and compression algorithms for high-throughput sequencing technologies." BMC Bioinformatics, 2010.

[34] NCBI. n.d. ftp://ftp.ncbi.nlm.nih.gov/genbank/ (accessed 6 2014).