Cluster and Grid Based Classification of Transposable Elements in Eukaryotic Genomes

Nirmal Ranganathan, Cédric Feschotte*, David Levine
Department of Computer Science and Engineering, *Department of Biology
University of Texas at Arlington
nrangana@cse.uta.edu, cedric@uta.edu, levine@cse.uta.edu

Abstract — **In the last few years many computer and laboratory improvements in the production and analysis of DNA sequences have made possible the complete sequencing of whole genomes. This provides a wealth of raw genomes that needs to be processed and annotated. All eukaryotic genomes examined and published thus far contain repetitive DNA. The amount of repetitive DNA in any specific eukaryotic genome ranges from 5% to 80%. These repeats consist mainly of transposable elements and tandem repeats which need to be identified, classified and annotated in order to sequence and annotate an entire genome. This paper discusses the design and implementation of a distributed cluster and grid based workflow to classify transposable elements. We show experimental results for representative species genomes on a cluster and grid. The performance and results of the workflow with regard to turnaround time, scalability, load balancing, resource utilization and fault tolerance are shown and discussed.**

Index Terms — **cluster, distributed workflow, transposable elements, bioinformatics.**

I. Introduction

The increasing pace of genome sequencing for many species has created a need to analyze these genomes quickly and encouraged new computational methods to assist in the processing of these genome sequences. One area essential for genome sequencing, annotation and analysis is the identification, classification and annotation (so-called 'masking') of transposable elements (TEs). TEs form a major part of repetitive DNA and copies of these DNA sequences are dispersed at multiple locations across the entire genome.

A significant part of bioinformatics work is involved with the use of workflows and data parallel applications, which are likely candidates to be executed on a cluster, SMP or the Grid [1]. Recognizing this, there are many parallel implementations of bioinformatics applications currently available; for example HMMer [2], FASTA [3], mpiBLAST[4], ClustalW-MPI [5], BeoBLAST [6], fastDNAml [7] and GridBlast [8]. There are numerous grid workflow construction tools such as GridAnt [9] and ICENI[10], which provide tools for creating complex workflows for grids, and there is Taverna [11], BioPipe [12] and Wildfire [13] which are workflow generation tools specific to biology and bioinformatics workflows. While these tools provide significant benefit creating integrated application systems, our system dynamically adapts the workflow to respond to changing processing and data needs.

Our effort deals with the development of a unique, distributed, cluster and grid-based application for an automated classification of TEs. Currently the process of classifying TEs is very time-consuming as it essentially relies on a manual curation of each repeat family. In addition the classification is biologically complex and, therefore, the curation task needs to be accomplished by a few TE experts. Given the frenetic pace at which genome sequence information is released in the public databases, there is an urgent need to develop an application to automate this annotation step. This paper describes the design of the classification workflow and how it is integrated as a distributed workflow in a cluster environment and porting of the application to a computational grid. We then discuss performance results of the application on the C. elegans genome and the human X chromosome.

II. Classification of Transposable Elements

Transposable elements are sequences of DNA that may be copied or moved around to different positions within the genome by a process called transposition. TEs often comprise a significant fraction of the genome, e.g. about 22% of the fruit fly genome and nearly half of the human genome [14]. The movement of TEs can generate mutations and chromosomal rearrangements and thus affect the expression of 'host' genes and ultimately the phenotype of the organism. TEs [15] are generally 100 to 20,000 base pairs long. For many years TEs were merely considered to be 'junk' DNA, but with increasing knowledge and understanding of TEs it is becoming clear that TEs are involved in a broad variety of processes directly related to the functioning of the host cell such as telomere elongation, genetic and epigenetic regulation

This work is supported in part by the Distributed and Parallel Processing Cluster (DPCC) at the University of Texas at Arlington. The DPCC is funded by NSF MRI award EIA-0216500. This work is also sponsored in part by the State of Texas workforce commission funding for BioGrid Texas.

of gene expression and the birth of new proteins and functions during evolution [16-19].

Detection of TEs is a basic step in many biologically important analyses including, but not limited to, sequence assembly during genome sequencing, genome annotation, similarity searches and gene and coding sequence prediction. The study of transposable elements involves three steps: identification, classification and masking. *De novo* TE identification in complete genome sequences is a computationally intensive task and tools like RECON [20], RepeatScout [21], Piler [22] and ReAS [23] have been designed to automate and facilitate this step. Though none of these programs have been designed as a distributed application, RepeatScout may utilize multiple shared memory processors and all of these tools require a lot of memory. Until now the classification of repeats has been performed 'manually' one repeat family at a time and automation of this process is the main focus of this work. The masking (i.e. final annotation in a genome sequence) of TEs has been primarily done based on the similarity of previously known repeats. Tools such as Repeat Masker [24], Masker Aid [25] and Censor [26] are widely used to detect repeat sequences based on similarity, and then mask those sequences, usually by replacing detected repeats by special characters.

The levels of classification are based on certain properties of the TEs, which distinguish them from one another. *Class* distinctions are created based on the transposition intermediate and mode of transposition. Class 1 elements transpose by means of a RNA intermediate that is reverse transcribed into DNA before integration. The diagnostic coding sequence of class 1 elements is the enzyme reverse transcriptase. Class 2 elements transpose directly via a DNA intermediate using an element-encoded enzyme called transposase. In the next level, *subclass* is distinguished based on structural properties, integration mechanism and the coding capacity. The structural properties are terminal inverted repeats, long terminal repeats, tRNA-like secondary structure and terminal simple sequence repeats. The integration mechanism is reflected by the target site duplications (TSD), which the TE creates in the flanking host DNA upon insertion. Figure 1 illustrates the two major transposition mechanisms and the formation of TSD. The next level of classification is *superfamilies*, which are distinguished by integration mechanism (e.g. size and sequence of the TSD) as well as phylogenetic analysis of the element-encoded proteins (if present).

Classification of TEs is a multi-step process, which relies on the combination of multiple evidence, based on various properties (for example TSDs and the structure of the TEs). Several independent steps are involved in the process of classifying these sequences. Our approach integrates the results of three independent methods to classify TEs. The three approaches are:

- Homology search
- Target Site Duplication (TSD) search
- Structural search

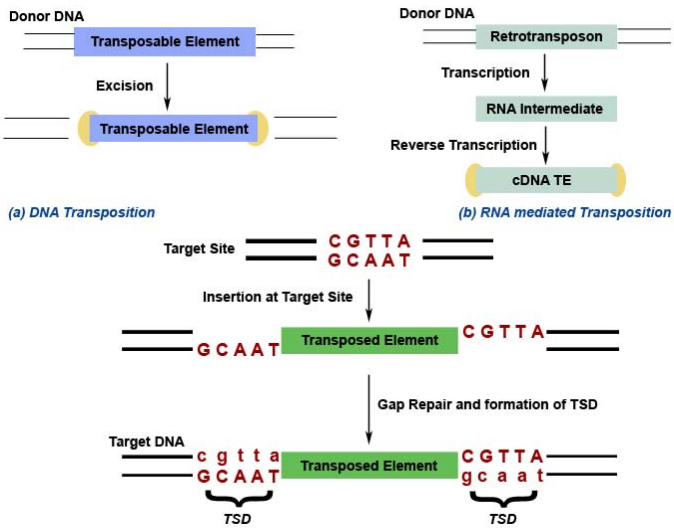

Figure 1: Transposition of Transposable element (TE)

The homology search compares the input TEs to recognize TE families. Certain TEs that exhibit similar characteristics are grouped together as a family. We search for similarity of the input TE to existing TE elements in the Repbase [27] database. The Repbase database is the major authoritative repository of known TE classifications and sequences.

Target site duplication is the process of creating duplicate sequences of a portion of the host genome at the ends of the transposable element when they are inserted into new locations within the genome. This process creates the same sequence on both ends of the transposable element (for example CGTTA<TE sequence>CGTTA, also see Fig. 1). Different families of TEs form different TSDs during insertion and this helps distinguish the TE family. This step is a computationally and memory intensive task as the entire genome needs to be searched for the various copies of the input TE sequence.

The final method is the structural search, which is designed to identify the aforementioned structural characteristics of TE subclasses and superfamilies. This is also a computationally intensive search since all the possibilities for such structures need to be searched. The last step integrates the outputs from all three steps and computes a tentative classification for the input TE sequence based on the overlap between the results of the three steps.

While the classification of a single TE requires one pass of the each process in the workflow, the classification of the entire genome requires several hundred or thousand iterations of the workflow, requiring hundreds to thousands of hours if run sequentially. The interactions between iterations are minimal, thus yielding an "embarrassingly parallel" implementation, which scales well to clusters and grids. Figure 2 shows the classification workflow.

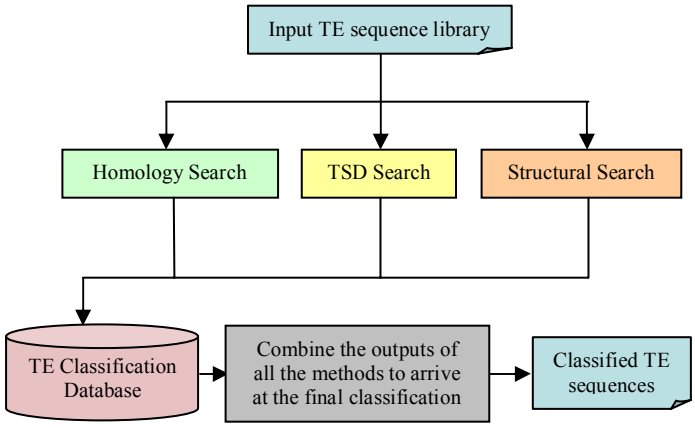

Figure 2: Classification workflow

Our application has several specific performance goals. Those goals are listed below.

A. *Turnaround time*

It is critical to achieve a fast turnaround time to assist biologists in performing their analysis quickly. Fast turnaround time can be achieved by utilizing more processors, by being able to process the data set in parallel. Providing partial intermediate results will help a biologist to immediately start working on their interests, increasing their productivity.

B. *Load balancing*

In a distributed environment, the available resources and the duration of their availability are very dynamic. A distributed workflow essentially needs to manage the load levels on the different nodes effectively, providing a balanced distribution of load.

C. *Scalability*

Scalability is another important issue to consider, given the "*embarrassingly parallel*" nature of the application. Such applications are very scalable because of the nature of a single program operating on different data sets. Considering scalability (S) and number of parallel tasks (N_p), our goal is:
$S \alpha N_p$

D. *Resource Utilization*

Resource utilization is a challenging task when an application is deployed on a grid. We show the maximizing resource utilization in a cluster and the extensions to effectively use the available resources in a grid.

E. *Fault Tolerance*

One important consideration in any distributed application is fault tolerance. We provide fault tolerance at the data level, meaning that errors generated during the processing of data are handled effectively. Additionally, any errors caused by the cluster and grid software also need to be handled providing the user with appropriate error-logging messages.

III. DESIGN

The workflow consists of several different steps which can be performed in parallel and construct TE classification into a data parallel task. We begin by designing a system using a cluster that allows us to efficiently execute the workflow on a cluster of any size and then show extensions to a grid. The following factors were taken into account while designing the computational workflow: turn around time, load balancing, scalability, resource utilization and fault tolerance.

The inputs to this workflow are a TE consensus sequence that will be annotated and the entire genome to which the TE belongs. In most cases, one is not interested in a single TE sequence but a library of TE sequences which can be 100 to several 1000 sequences. This TE sequence library is generated by one of the identification programs such as RECON, RepeatScout, Piler or ReAS.

The program consists of 3 different job types:

1. The executers are the jobs that perform the actual classification process defined in our system.
2. The data pool is a single job that consists of a pool of input TE sequences.
3. The gatherer is also a single job that performs the gathering of results from the executers and handles errors created by the executers.

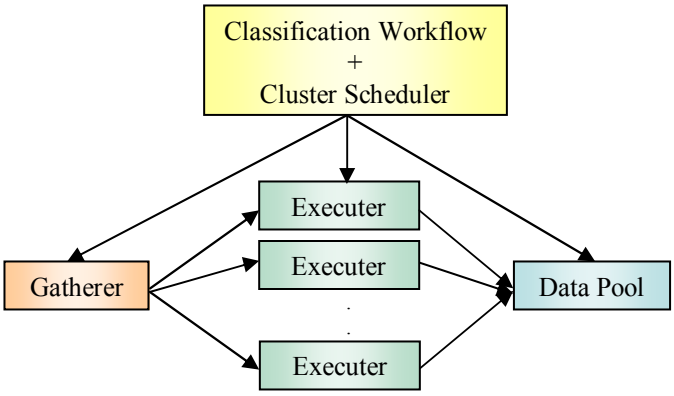

Figure 3: Execution in a cluster

The first step is obtaining an estimation of the number of free nodes in the cluster. In this work we support the batch schedulers PBS [28], Torque [29] and LSF [30], individual scheduler policies are not supported. The estimation process uses information from the batch scheduler giving the number of free nodes, the number of jobs currently in the queue and the number of TE sequences in the input library. The program dispatches one data pool job, one gatherer job and several executer jobs. The number of executer jobs depends on the number of free nodes and the number of queued jobs. When there are no queued jobs, the number of executer jobs is equal to the number of free nodes. When there are queued jobs, a target number is generated based on the size of the input library.

The data pool job provides exclusive access to the executer. Only one executer can access the data pool at any instant, thus avoiding a race condition by the executers. The executers receive one data unit, process it and then request new data. This proceeds until the pool is empty at which point the executer signals its exit status to the gatherer. The gatherer

checks for any errors in the results of the executer and collects the data. Errors occur when all the workflow steps have not been processed - this step is application specific. Data sets with errors are inserted back into the data pool for re-processing. If there are no errors the executer acceptance signal is sent to the executer, which then terminates. After all the executers are terminated, the gatherer groups all the output and creates a summary of the run along with the TE classification library for the genome.

When users query the gatherer for intermediate results, the gatherer looks up all the executers and gathers their completed data sets. Intermediate results allow a user to get started on the analysis of the output within a short time rather than wait for all the TE sequences to be processed.

A. TE classification on a Grid

Grid computing in addition to a means for researchers to do existing research faster provides a number of other capabilities. While the ability to carry out existing experiments in less time is definitely beneficial, grids also allow for more advanced research to be carried out at lesser costs and in a collaborative environment. BioGrids aim to provide biologists with tools and portal interfaces to perform their research in a collaborative environment utilizing the computing capabilities of a grid. BioGrid initiatives such as ^{my}Grid [31], BioGRID (Poland) [32], NC BioGrid [33], BioGrid (Japan) [34] and BioGrid NUS [35] concentrate on enabling bioinformatics applications to be executed on grids.

Our TE classification application is run on a home made version of a BioGrid, BGrid [36], which consists of custom middleware providing minimal facilities such as job submission, monitoring, and data proxy management. We omit the detailed description of BGrid in this paper, since it provides only limited functionality needed by the classifier and we touch upon only those aspects that are used for the classification of TEs. Our BGrid implementation consists of two parts: the BGrid Portal and the BGrid Middleware. The BGrid Portal is a front-end for applications through which biologists submit their tasks. The portal is a single entry point where standard and custom written biology applications are made accessible. These applications are scheduled and monitored through BGrid middleware. BGrid uses Globus[37] for data management, authentication, and resource discovery.

The job submission agent will use the monitoring information from the job monitoring agent to locate available processors and the availability of the genome sequences and the software at that particular site. The job submission agent first dispatches jobs to sites where resources and the software are available. This way only the data set need to be transferred, reducing the network load. If such resources are not available, job data along with the software are packaged and dispatched to the other free resources. The jobs are split up based on the number of TE sequences in the input library and the number of available resources. Within each site the jobs are dispatched to the local scheduler in the same way we would run the application on a cluster. Globus GRAM [38] is used for all the job submissions on the grid.

The monitoring agent monitors sites where the jobs are being executed to gather information on the progress of the job and communicates that status information to the results gatherer at regular intervals. It is also responsible for collecting information on jobs that can be rescheduled at a different site to achieve a faster turnaround time. This information is passed on to the job submission agent, which reschedules the job at an alternate site.

The results gathering agent gathers results from the various processing sites and stores it across replicated databases where it can be accessed by all collaborating users of the Biogrid portal. The software and data transfer and the results gathering use GridFTP [39] for the transfer of the files.

Figure 4 shows a simplified software stack of the BGrid consisting of five parts. The BGrid portal consists of all the portlets that allow biologists to submit jobs through a web interface. The BGrid has custom made middleware, which are more suitable for a grid responsible for all the job execution and resource location services for bioinformatics applications. The next layer is all the Globus middleware. The cluster management layer consists of cluster schedulers and monitoring software along with cluster management tools. The last layer is the system software consisting of the operating system, compliers, software libraries, biological software and genome databases and other third party software.

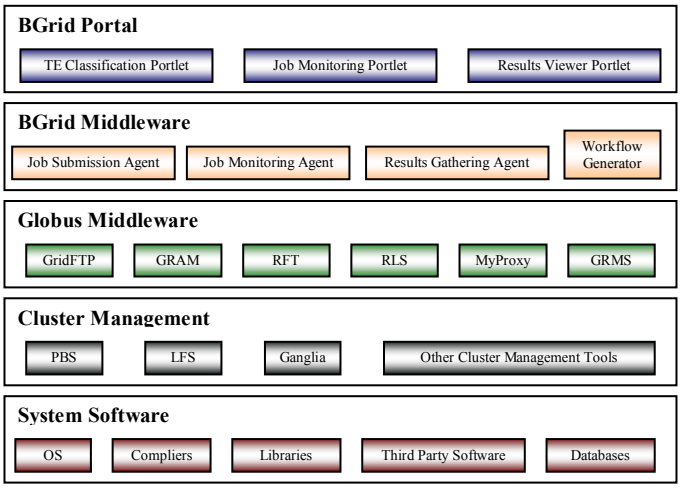

Figure 4: Simplified software stack of BGrid

IV. Results and Discussion

We have successfully deployed this system at the Distributed and Parallel Computing Cluster at UTA which consists of 81 dual processor 2.667 GHZ Xeon compute nodes with 2GB memory each. This system was run at various load levels on the cluster on different genomes. In addition, the grid implementation utilizes four distinct clusters, ranging from 10 to 200 nodes each and connected with a 622 Mbps campus network and gigabit Ethernet within the clusters. Each cluster is composed of several 1.5 Ghz and faster processors similar to the previously described DPCC cluster. All clusters are running Linux, jobs are scheduled via Torque or PBS and each

has Globus and bioinformatics software installed.

In this paper we present the performance of our workflow to classify TEs for the Caenorhabditis elegans (worm) genome and the human X chromosome. These two genomes were chosen because of the diversity of their transposable elements and the size of the genomes.

The program was run on two data sets at various loads, ranging from a single executer to 60 executers. This allowed us to measure the turnaround time, scalability and the load balancing achieved by the program on the cluster and the grid. Under normal conditions the program would spawn the number of executers based on the available number of free nodes, number of queued jobs and the size of the data set.

The data set for C. elegans consisted of 408 TE consensus sequences identified de novo using RepeatScout. The data set for the human X chromosome consisted of 751 TE consensus sequences which were also identified de novo using RepeatScout. Using our application we classified 145 (35.8%) of the TE sequences for C. elegans and classified 43 new elements which were not previously identified or classified. For the human X chromosome we were able to classify 500 (66.6%) of the elements.

As an additional step we classified the TE elements that were already classified in Repbase as a control for our classification system. We performed the classification of two previously well annotated species, C. elegans and Drosophila melanogaster. For C. elegans the number of repeats in Repbase are 124 and we classified 111 (89.52%) of those repeats with an accuracy of 92%. The D. melanogaster genome consists of 156 repeats and we were able to classify 141 (90.38%) repeats with an accuracy of 95%.

The use of the application in a grid introduces extra overhead in the form of software and job staging and the retrieval of results. This overhead in using a grid to process the repeats in C. elegans genomes accounts for 0.4% to 9% increase in the overall processing time when moving from 1 to 60 processors and the overhead for the human X chromosome ranges from 0.2% to a negligible percentage of the overall time. This overhead is caused by the time taken to stage the software, genome and the data set, in addition the retrieval of the results (up to 5 gigabytes) accounts for a major portion of the overhead. The experimental setup on the grid distributed jobs to each local cluster, job distribution depended on the number of available, unoccupied processors, but jobs were always run on at least three different clusters simultaneously.

A. *Turnaround time*

For the C. elegans genome it required 20.6 hours of CPU time using a single processor and only 25 minutes using 60 processors. In the case of the human X chromosome 250 hours of CPU time was required on a single processor and we were able to complete the processing in 10 hours using 60 processors. It would take nearly 7 months to complete the processing of the entire human genome which can be completed in slightly more than one week using our distributed workflow.

The time taken for the processing of the worm genome is very small compared to the human X chromosome. There was a speedup of 50 times in the C. elegans genome and 25 times in the human X chromosome from 1 to 60 processors. Figures 5 and 6 show the turnaround time using a cluster and grid for the C. elegans genome and the human X chromosome respectively.

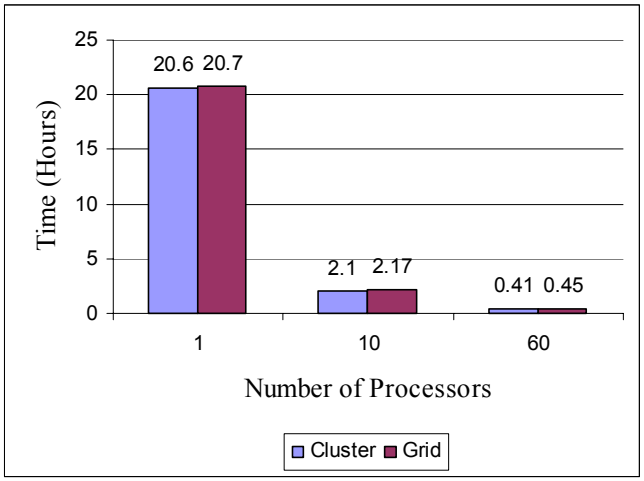

Figure 5: Turn around time for C. elegans (Note that a one-processor grid is a trivial case)

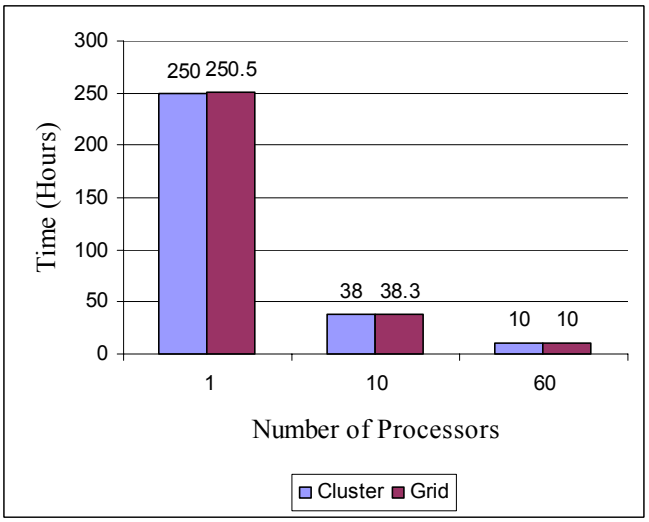

Figure 6: Turn around time for human X Chromosome (Note that a one-processor grid is a trivial case)

We note that the grid implementation is only slightly worse than the cluster version. While in many applications this would be surprising, in this case the grid version benefits from the high speed campus network, the ability to stage data concurrent with processing and the recognition that this processing on a cluster minimizes the network overhead. This type of result is also more common in this class of "embarrassingly parallel" systems. Depending on the characteristics of other clusters within grids, we would not expect such pleasant minimal impact results on a more typical grid.

B. *Scalability*

For this application the scalability depends on the number

of sequences in the input TE library. There is, however, a lower threshold limit for the scalability beyond which a constant amount of time is required irrespective of the increase in the number of processors. This limit is reached when the time to execute a single data set is more than the time taken by the other jobs to execute several data sets.

The threshold in the scalability for the worm genome was reached at 55 processors and for the human X chromosome at 60 processors, increasing the number of processors beyond that limit will still result in the same turn around time for the respective genomes. The scalability is not linear because the data set is the actual genomic biological data and not simulated data. Since the actual data sets vary tremendously in terms of the size of the genome, number of repeats and other factors, we see in smaller genomes that even a small number of CPUs yield a tremendous turnaround time benefit compared to a single CPU. Figures 7 and 8 depict the scalability on a cluster for the classification of C. elegans and human X chromosome respectively. The scalability is shown for a increase in steps of 10 processors.

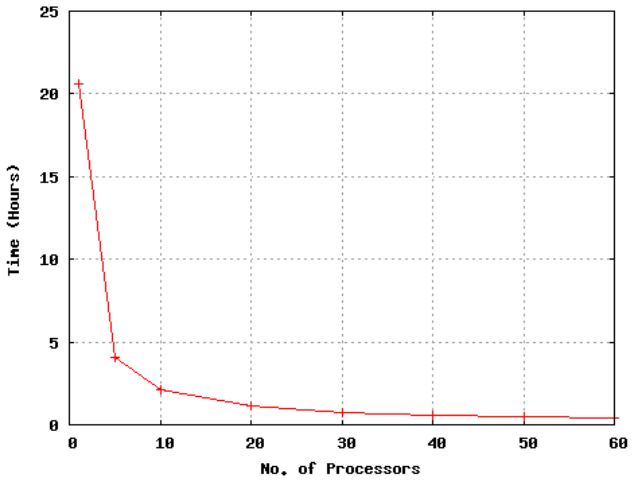

Figure 7: Scalability for C. elegans genome on a cluster

C. *Load balancing*

Our system does not introduce extra load on the cluster scheduler by queuing extra jobs. It creates the number of jobs based on the available number of free nodes. Each job, in turn, fetches a data set from the data pool and processes it. This way no single node gets substantially a higher load than others. A job fetches a new data set only after completion of the current data set. This leads to efficiency since there will be no idle node or another node with extra processing loads.

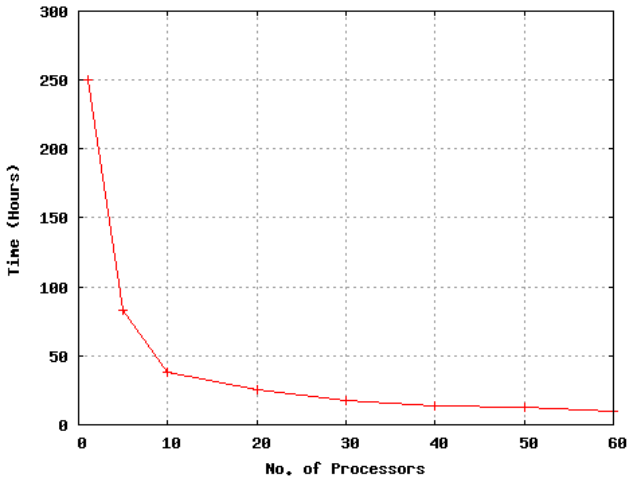

Figure 8: Scalability for human X chromosome on a cluster

In the case of C. elegans (Figure 9) we have considered the load balancing when 10, 20, 30, 40, 50 and 60 processors were used. There was an even distribution of load in the C. elegans genome. The load balancing in the human X chromosome (Figure 10) is shown only for 10 and 60 processors. There was a large variation in the load distribution in the human genome. We noted that the times taken for a few data sets are much longer than most other data sets. This was not the case with C. elegans where the times taken had smaller variances. This problem of one data set taking a long time can be improved by calculating a moving average over the completed jobs and ensuring that new jobs conform to the moving average. Jobs that take more time than the moving average can be discontinued and further be split up into the independent tasks of the workflow. This provides a faster turn around and balances the load more evenly. In the following graphs, each plot represents the time consumed on a single processor when the application was executed with a fixed number of processors.

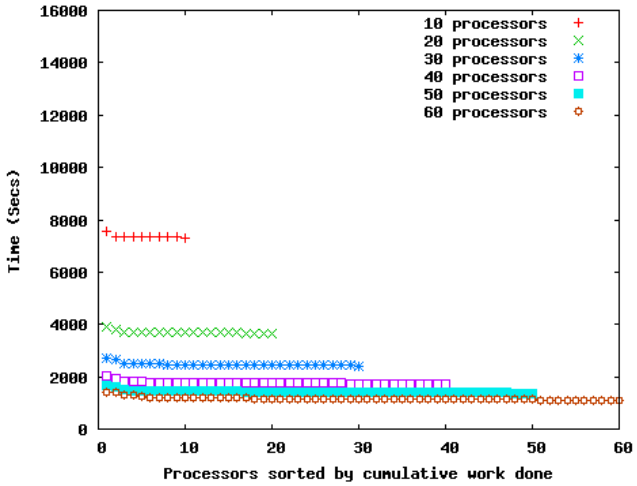

Figure 9: Load balancing for C. elegans genome on a cluster

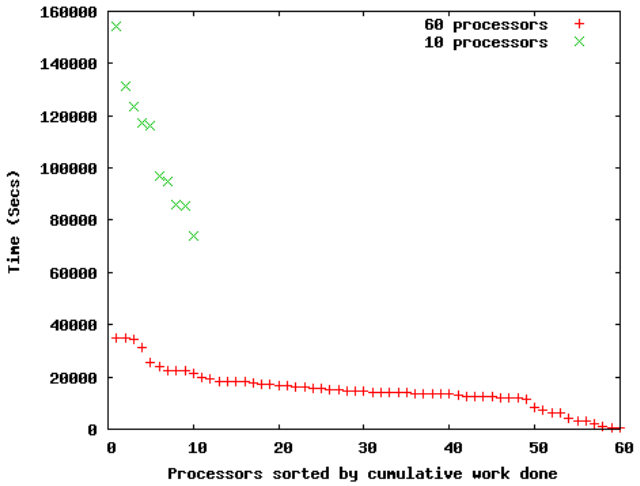

Figure 10: Load balancing for human X Chromosome a cluster

D. *Resource Utilization*

This program achieves maximum resource utilization by using all the free nodes available in a cluster or grid at the time of initialization. Further monitoring at regular intervals spawns more jobs when the queue becomes empty and nodes become freed while there are more data sets available in the data pool to be processed.

V. Conclusion

Bringing together computer science researchers with cluster and grid experience and biological researchers with requirements to access powerful computational resources will spur innovation in both areas. We were able to classify 35.8% of the TE input sequences for the C. elegans genome and 66.6% of the sequences for the human X chromosome. The classified sequences for C. elegans contained elements that were previously manually classified and 43 new elements which had never been previously classified. Our data set for C. elegans contained 82.25% of the TE sequences already found.

Classification of TE is computationally intensive and can take from 21 hours for the C. elegans genome to 7 months in the case of the human genome. Using clusters and grids provides faster turn around and high scalability. The speedup with the increase in processors varied from 50 for the C. elegans genome up to 25 for the human X chromosome. The speedup varies significantly due to the size of the genome, the complexity of the repeats and the number of repeats in the input TE library of consensus sequences. The load balancing also varies in cases where certain TE sequences take more time to process. The speed up can drastically be increased with the use of a larger computational grid. Finally, the performance of our system greatly depends on the quality, size and complexity of the genomes and the TE sequences.

References

[1] I. Foster, C. Kesselman, et al. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, vol. 99(15), 200-222, 2001.

[2] HMMER: Profile HMM's for protein sequence analysis. (Jan 2006) Available: http://hmmer.wustl.edu.

[3] FASTA: protein sequence comparison. (Jan 2006) Available: http://fasta.bioch.virginia.edu/.

[4] mpiBLAST: Parallel Blast. (Jan 2006) Available: http://mpiblast.lanl.gov/.

[5] Li, Kuo-Bin. ClustalW-MPI: ClustalW Analysis using distributed and parallel computing. *Bioinformatics*, vol. 19(12), 1585-1586, 2003.

[6] J. D. Grant, R. L. Dunbrack, F. J. Manion, M. F. Ochs. BeoBLAST: Distributed BLAST and PSI-BLAST on a Beowulf cluster. *Bioinformatics*, vol. 18(5), 765-766, 2002.

[7] G. J. Olsen, H. Matsuda, R. Hagstrom, R. Overbeek. fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Compu. Appl. Biosci.*, vol. 10, 41-48, 1994.

[8] A. Krishnan. GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework. *Concurrency and Computation: Practice and Experience*, vol. 17, 1607-1623, 2003.

[9] GridAnt: A Grid Workflow System. (Jan 2006) Available: http://www-unix.globus.org/cog/projects/gridant.

[10] A. Mayer, S. McGough et. al. ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time. UK e-Science All Hands Meeting, p. 627-634, 2003

[11] Taverna. (Jan 2006) Available: http://taverna.sourceforge.net/.

[12] S. Hoon, K. K. Ratnupa, J. Chia et. al. Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis. *Genome Research*, vol. 13, 1904-1915, 2003.

[13] F. Tang, C. L. Chua, L. Y. Ho, Y. P. Lim, P. Issac, A. Krishnan. Wildfire: distributed, Grid-enabled workflow construction and execution. *BMC Bioinformatics*, vol. 6, 69, 2005.

[14] V. V. Kapitonov, J. Jurka. Molecular paleontology of transposable elements in the Drosophila melanogaster genome. *Proc. Natl. Acad. Sci.,* vol. 100: 6569-6574, 2003.

[15] D. J. Finnegan. Eukaryotic transposable elements and genome evolution. *Trends in Genetics,* vol. 5, 103-107, 1989.

[16] M. G. Kidwell and D. R. Lisch. Perspective: Transposable Elements, Parasitic DNA, and Genome Evolution. *Evolution*, vol. 55(1), 1-24, 2001.

[17] N. J. Bowen and I. K. Jordan. Transposable Elements and evolution of eukaryotic complexity. *Mol. Biol.,* vol. 4(3), 65-76, 2002.

[18] C. Feschotte, N. Jiang, S. R. Wessler. Plant Transposable Elements: Where Genetics Meets Genomics. *Nature Reviews Genetics,* vol. 3, 329-341, 2002.

[19] P. L. Deininger, J. V. Moran, M. A. Batzer and H. H. Kazazian. Mobile elements and mammalian genome evolution. *Curr. Opin. Genet. Dev.,* vol. 13, 651-658, 2003.

[20] Z. Bao and S. R. Eddy. Automated de novo Identification of Repeat Sequence Families in Sequenced Genomes. *Genome Research*, vol. 12, 1269-1276, 2002.

[21] A. L. Price, N. C. Jones and P. A. Pevzner. De novo identification of repeat families in large genomes. *Bioinformatics*, vol. 21, i351-i358, 2005.

[22] R. C. Edgar and E. W. Myers. PILER: identification and classification of genomic repeats. *Bioinformatics*, vol. 21(Suppl 1), i152-i158, 2005.

[23] R. Li, J. Ye, S. Li, J. Wang, Y. Han, et. al. ReAS: Recovery of ancestral sequences for transposable elements from the unassembled reads of a whole genome shotgun. *PLoS Computational Biology,* vol. 1(4), E43, 2005.

[24] A. F. Smit. Repeat Masker. (Jan 2006) Available: http://www.repeatmasker.org.

[25] J. Bedell, I. Korf, W. Gish. Masker Aid: a performance enhancement to RepeatMasker. *Bioinformatics*, vol. 16, 1040–1041, 2000.

[26] J. Jurka, P. Klonowski, V. Dagman, P. Pelton. CENSOR - a program for identification and elimination of repetitive elements from DNA sequences. *Computers and Chemistry*, vol. 20, 119-122, 1996.

[27] J. Jurka, V. V. Kapitonov, A. Pavlicek, P. Klonowski, O. Kohany. Repbase Update, a database of eukaryotic repetitive elements. *Cytogenetic Genome Research*, vol. 110, 462-467, 2005.

[28] OpenPBS. (Jan 2006) Available: http://www.openpbs.org/

[29] Torque. (Jan 2006) Available: http://www.clusterresources.com/pages/products/torque-resource-manager.php.
[30] Load Sharing Facility. (Jan 2006) Available: http://accl.grc.nasa.gov/lsf/
[31] R. Stevens, A. Robinson, and C. A. Goble. myGrid: Personalised Bioinformatics on the Information Grid. *Intl. Conf. of Intelligent Systems in Molecular Biology*, vol. 19: i302-i304, 2003.
[32] BioGRID Poland. (Jan 2006) Available: http://biogrid.icm.edu.pl/index.php?section=main&subsection=info
[33] North Carolina BioGrid. (Jan 2006) Available: http://www.ncbiogrid.org/
[34] BioGrid Japan. (Jan 2006) Available: http://www.biogrid.jp/
[35] BioGrid in the National University of Singapore. (Jan 2006) Available: http://www.bic.nus.edu.sg/biogrid/
[36] N. Ranganathan, A. Vaidya, S. T. Reddy. BGrid: Component based architecture for Bioinformatics on the Grid, unpublished.
[37] I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. Supercomputer Applications*, vol. 11(2), 115-128, 1997.
[38] J. Nabrzyski, J. M. Schopf, J. Weglarz. Grid Resource Management. Kluwer Publishing, 2003.
[39] W. Allcock. GridFTP Protocol Specification. GGF Technical Report, 2003.