2016-01-01

# Assessing Accuracies And Improving Efficiency For Segmentation-Based Rna Secondary Structure Prediction Methods

Gerardo A. Cardenas
*University of Texas at El Paso*, gacardenas@utep.edu

# ASSESSING ACCURACIES AND IMPROVING EFFICIENCY FOR SEGMENTATION-BASED RNA SECONDARY STRUCTURE PREDICTION METHODS

GERARDO ALBERTO CARDENAS JAMES

Doctoral Program in Computational Science

APPROVED:

Ming-Ying Leung, Ph.D., Chair

Kyle L. Johnson, Ph.D.

Shirley Moore, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

# Dedication

This dissertation is dedicated to my beloved parents Ofelia James and Jose Luis Cardenas for all their continued support and counsel during my life and professional career. In the same way, I would like to dedicate this dissertation to my wife Edith Castrellon, for all her support, guidance, tolerance, and care during the realization of this project and to our two children Gerardo and Edith, whose sacrifice will be rewarded with the realization of this dissertation.

This project is also dedicated to all my family members who made this dissertation possible in many ways. My deepest gratitude to my brother Jose Luis Cardenas for giving me his example and perseverance to achieve any objective; my sister Ofelia Cardenas for teaching me that everything in this life can be overcome if one proposes it; and my brother Carlos Cardenas for all his help and support during my life. To my nephews Ana Cristina, Ximena, Sofia, Natalia, Juan Carlos, y Jose Luis Cardenas, my deepest and sincere thanks to each of you because you have been the best motivation ever to become myself a better human being every day. I hope one day you read this project and serve as an example for you to achieve all proposed goals during your life.

A special dedication to my grandfather Francisco James and grandmother Ofelia Martinez, who unfortunately passed away time ago but they are always present in my mind.

# ASSESSING ACCURACIES AND IMPROVING EFFICIENCY FOR SEGMENTATION-BASED RNA SECONDARY STRUCTURE PREDICTION METHODS

by

GERARDO ALBERTO CARDENAS JAMES, M. Sc.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Computational Science Program

THE UNIVERSITY OF TEXAS AT EL PASO

Fall 2016

## Acknowledgements

I would like to express my sincere gratitude to Dr. Ming-Ying Leung for giving me all her guidance, trust, and support for almost 10 years, which has contributed towards my research work and professional career. To my committee, Dr. Kyle L. Johnson and Dr. Shirley Moore, whose work in the biology and computational science areas, respectively, encourage me to achieve the objective of my dissertation. I am extremely grateful to members of my committee for giving me their valuable assistance and support throughout the development of this project.

In addition, I would like to thank every person who has contributed emotionally during the development of this research, has participated somehow directly or indirectly in the realization of this dissertation, and has supported me in the achievement of this personal goal. Professionally speaking, I would like to thank all administrative people who formed the application support team in the Information Technology department at the City of El Paso in 2010, specially to Monica Castillo, for giving me the opportunity of working full-time and at the same time attending to all core and elective computational science courses. Furthermore, I would like to express my deepest appreciation to the Asset Management Technologies group from El Paso Electric Company, and a special thank you to Frank Esparza, for all his support and advice provided during the second half of the PhD, which have had a positive impact in my personal and professional development.

I also want to thank the many people who helped me directly and were part of this adventure during six years. To my doctoral classmates, Octavio Lerma and Javier Polanco, for all their help and collaboration during my PhD. To my nearest friends Arlen Martinez, Enrique Rivas, Karina Ramos, Felipe Velez, Juan Carlos Rivera, Ana Betancourt, and Juan Carlos Calderon, for their motivational support that helped me to complete this adventure. My gratitude also goes out to all those people who indirectly supported me during my carrier., specially my coworkers from the University of Texas at El Paso (UTEP), the City of El Paso, Dyonyx LLC, and El Paso Electric Company.

# Abstract

RNA secondary structure prediction has become an important area of interest in biology and medicine because it helps in understanding the mechanisms of many biological processes such as gene regulation and viral replication, and in designing RNA-based therapies to treat various diseases such as cancers and AIDS. Different thermodynamics-based computational algorithms for RNA structure prediction exist, and have been used to help understand the disease mechanisms and design treatments. However, most of these computational tools that can predict complex pseudoknot structures have a sequence length limitation of few hundred nucleotide bases due to their high demands of computer resources. Yet, many RNA molecules, such as those making up viral genomes, are thousands of bases long. To overcome the sequence length limitation, a segmentation approach called chunk concatenation method was previously proposed to cut a long RNA sequence into shorter chunks at strategic positions that conserve inversion patterns in the nucleotide sequence, predict each single chunk independently by existing software like pknotsRG or RNAstructure, and then combine the results to build the final prediction of the entire RNA. In the present study, we investigated whether the prediction accuracy over 136 sequences with known structures obtained from a collection of Rfam non-coding RNA families sequences could be improved by capturing possible structures formed between two neighboring chunks that would be missed by the usage of the chunk concatenation method. We compared the overall prediction accuracies of regular, chunk concatenation, and two-chunk elimination prediction methods to determine whether they are statistically different in prediction accuracy or not. In addition, a high-throughput distributed batch computing system called HTCondor has been used to reduce the waiting time for the RNA secondary structure prediction of 14 Nodavirus sequences in comparison to their sequential prediction, overcoming the need of high demands of computer resources when processing long RNA sequences.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

## 1.1 Biological Significance of RNA

Ribonucleic acid (RNA) is one of the three most important biological macromolecules that are of special importance for all living organisms, which is composed of four nucleic acids: C (cytosine), A (adenine), G (guanine), and U (Uracil) [43]. The interaction of these nucleotides through hydrogen bonds allow to form either canonical base-pairing A with U and G with C or non-canonical base-paring G with U known as Wobble (Figure 1.1a) as established by Watson and Crick in reference to the Central Dogma "where DNA was the storehouse of genetic information and RNA was the bridge that transferred this information from the nucleus to the cytoplasm where proteins were made" [11].

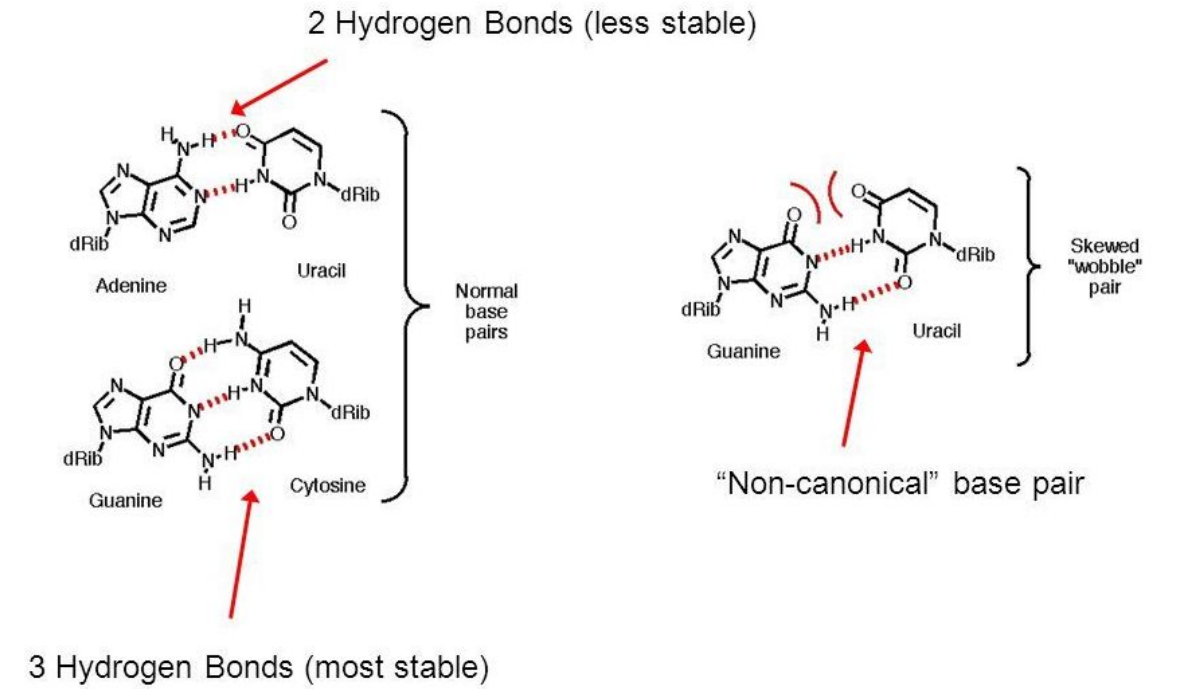



Image Source: "BC 5254/GCS 719, Computer Applications in Biomedical Research" http://www.finchcms.edu/cms/biochem/Walters/rna_folding.html

Figure 1.1a: Example of base-paring in RNA.

RNA is found in several organisms including bacteria, virus, plants, and animals [26] and plays important roles in many biological processes, including participation in protein synthesis, as carriers of genetic information in the cell, catalysts in cellular processes, and mediator in determining the expression level of genes by carrying the information from the DNA to the protein-building system [3].

There are three primary types of RNA involved in the protein synthesis: messenger RNA (mRNA), transfer RNA (tRNA), and ribosomal RNA (rRNA). The mRNA is responsible for making copies of the DNA information for protein synthesis, the tRNA is in charge of translating mRNA sequences into amino acid sequences (polypeptides), and the rRNA is in control of making up to 55% of ribosomes and the rest of proteins. Understanding the roles for the different types of RNA in the biological cellular processes can lead to develop and understand new methods of diagnosis and treating diseases [21].

Moreover, RNA represents the genomic molecules in many viruses, some of which cause fatal diseases such as AIDS, human influenza, or several forms of hepatitis [8]. In addition, there are many types of viruses like polioviruses, togaviruses, flaviviruses, and rotaviruses where the RNA is its main genetic material is injected [8]. Usually, viruses infect people, plants, and animals and since viruses do not replicate themselves, the virus infects a cell by injecting its genetic information into cell nucleus to take advantage of the replication process and infect other cells [26]. Understanding RNA in viruses is important for developing treatments and epidemiological control strategies to counteract their development and propagation of any disease they might cause in people, animals, and plants [8].

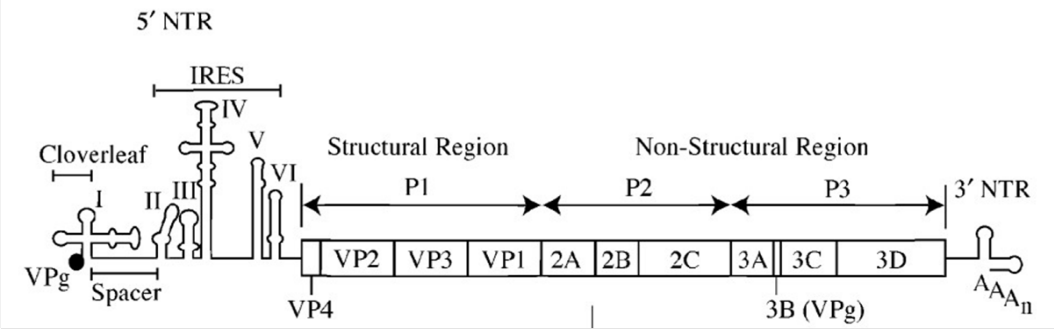



Figure 1.1b: Graphical representation of a genomic structure of poliovirus type 1.

## 1.2 Secondary Structure Analysis

The secondary structure of RNA is the collection of hydrogen-bonded base pairs, which is constructed by folding back to itself forming base-paired segments that can yield structures. The secondary structures are made of two categories of structural elements: stem-loops and pseudoknots (Figure 1.2a). The stem loops are structures where two regions of the sequence base pair each other forming a stem with some unpaired bases in the center forming a loop. The pseudoknots are complex structures that occur when stretches of the same sequence interact near each other. The stability of such structures resides, among other factors, in how the nucleotides are organized within the sequence forming base-stacking interactions, the base pairs interactions of A-U and G-C with 3 hydrogen bonds, and G-U with 2 hydrogen bonds, gap's length in loops, and the minimum free energy of the regions in RNA secondary structures.

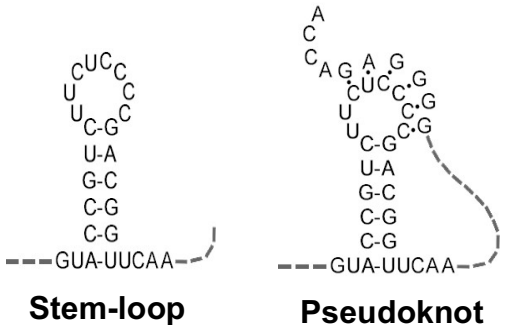



Figure 1.2a: Example of two categories of structural elements.

The function of an RNA molecule is determined by its structure, therefore, the significance of the secondary structure analysis lies in that it can help in determining the functionality of the RNA molecule as well as of many regulatory, catalytic, and structural processes of the cell. Besides informing about the functionality of the RNA molecule, the secondary structure of RNA gives information about the domain structure of the molecule and location of important sites within the structure. The RNA structure prediction has two important roles. First, it helps in the interpretation of experiments related to the RNA function. Second, it helps to establish new experiments to probe function.

One of the problems with the RNA secondary structure prediction is the heavy handling and processing of long RNA sequences due to the limitation of computer hardware and software. Most of the existing software based on the RNA secondary structure prediction, with or without pseudoknots consideration, processes all submitted RNA sequences in a sequential manner, leading to a decrease in the speed up of the RNA molecule prediction because each long chain of RNA sequence is processed one at a time and has thousands of internal operations established by the prediction algorithms. Furthermore, the prediction of long RNA sequences such as the RNA viruses, whose genomes are composed of RNA encoding a number of proteins, is usually impossible due to its limitation in sequence length imposed either by the authors at the application level or the computer resources available where RNA prediction programs are executed. In order to reduce the waiting time of secondary structure prediction of long RNA sequences, they can be broken up into several chunks, predicted individually using a throughput computing tool, and assembled back to their original positions to form the final predicted structure [46]. A distributed computing cluster can provide resources to manage the prediction of each RNA sequence chunk in parallel by distributing chunks to idle machines available in the cluster, allowing processing long length RNA sequences and improving the waiting time.

## 1.3 HTCondor Historical Perspective and Current Status

In the decade of the 80's, there was a high uncertainty on distributed computing but at the same time computer engineers and programmers had a clear understanding that it would be a feasible topic to address [9].

Since 1988, Miron Livny, a professor and senior researcher from the Computer Science department at the University of Wisconsin–Madison, used his doctoral thesis in combination with the novel Remote Unix and Crystal Multicomputer developed by DeWitt, Finkel, and Solomon to come up with a new idea of throughput computing by utilizing inactive processors called Condor [9]. In 2012, to bypass a lawsuit challenging the University of Winsconsin-Madison for the use of Condor trademark, the Computer Science department decided to rename its project to HTCondor, emphasizing the high-throughput computing achieved by using Condor [9].

The HTCondor software project was born at the University of Wisconsin-Madison (UW-Madison) with the sole purpose of having a new system for distributed computing capable of running computationally intensive jobs. It is classified as a high-throughput distributed batch computing system meaning that HTCondor has shared utilization of autonomous resources toward a common goal by exploiting all available computing resources efficiently [39]. Given its batch computing system classification, the HTCondor system inherits all the batch system features such as job management, scheduling policy, priority scheme, resource monitoring, and resource management. One of the problems that most of the distributed systems face is the lack of scalability. The features of HTCondor include flexibility, allowing scalability, easy of installation [39].

As of August 2, 2016, the center for high throughput computing at the University of Wisconsin-Madison released a stable version of HTCondor 8.5.6, where different issues and bugs detected in previous versions were fixed and several enhancements were incorporated to

increase its usefulness [15]. In addition, people from the center for high throughput computing have identified six challenge areas that will dictate the roadmap of HTCondor for the next five years. These areas include “evolving resource acquisition models, hardware complexity, widely disparate use cases, data intensive computing, black-box applications, and scalability” [36].

HTCondor job management system has been used intensively in the Computer Sciences department at University of Madison-Wisconsin for many years., Since the release of HTCondor to the public, many institutions, industries, and government have been using this tool as an efficient and reliable way to process multiple computational intensive jobs faster by using current and idle computer assets [14]. In fact, HTCondor has been used to develop “grid-style computing environments”, where different HTCondor installations from different departments within the company or other external entities have interacted and collaborated each other by interconnecting their HTCondor systems to enhance their user experience and at the same time, have more resources to work more jobs and reduce the waiting time when processing big data [14].

In this study, the HTCondor throughput computing tool has been used to contribute in reducing the waiting time not only when long RNA sequences are submitted for their prediction but also to overcome the issue introduced by having hundreds or even thousands of chunks from long RNA sequences generated when either chunk concatenation or two-chunk elimination methods are used on fourteen Nodavirus sequences with unknown real structures.

# Chapter 2: Literature Review

## 2.1 Computational prediction algorithms for RNA secondary structures

Due to the great amount of RNA sequences available and their variability in sequence length, computational algorithms for the prediction of RNA secondary structures have been developed to improve their processing time and increase their accuracy prediction. One old dynamic programming-based algorithm developed by Zuker and Stiegler used stacking and destabilizing energies (thermodynamics) to find the most favorable secondary structures of RNA based on the minimum free energy in contrast to other existing prediction methods based on topological concepts (see Table 2.1a) [50]. The applied technique improved the execution time of the RNA structure prediction using RNA sequences with N number of nucleotides in length by $N^3$ over other pre-existing RNA prediction algorithms such as Pipas and McMahon's prediction method using thermodynamics with an execution time of $2^N$, and Studnicka's prediction method with an execution time of $N^5$ [50].

Table 2.1a: Stacking and destabilizing energies

| $WX\backslash YZ$ | CG | GC | AU | UA | GU | UG |
|---|---|---|---|---|---|---|
| CG | -3.26 | -2.36 | -2.11 | -2.08 | -1.41 | -2.11 |
| GC | -3.42 | -3.26 | -2.35 | -2.24 | -1.53 | -2.51 |
| AU | -2.24 | -2.08 | -0.93 | -1.10 | -0.55 | -1.36 |
| UA | -2.35 | -2.11 | -1.33 | -0.93 | -1.00 | -1.27 |
| GU | -2.51 | -2.11 | -1.27 | -1.36 | +0.47 | +1.29 |
| UG | -1.53 | -1.41 | -1.00 | -0.55 | +0.30 | +0.47 |

- Energies ($\Delta G$ in kcals/mol) of $\begin{smallmatrix}5'W\,Y\,3'\\3'X\,Z\,5'\end{smallmatrix}$ stacked basepairs.
  - Note that $\Delta G$ of $\begin{smallmatrix}5'W\,Y\,3'\\3'X\,Z\,5'\end{smallmatrix}$ stacks is the same as $\begin{smallmatrix}5'Z\,X\,3'\\3'Y\,W\,5'\end{smallmatrix}$ stacks.

Mathews *et al.* (1999) Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *JMB*.

Several other RNA secondary structure prediction tools have been developed and different research results have been produced based on Zuker' stacking and destabilizing energies [34]. For instance, Yongmei Ji developed a computer algorithm called comRNA to predict common RNA secondary structure motifs including pseudoknots in unaligned sequences, which allowed the prediction of mRNA structure motifs that play an important role in regulating gene expression [18]. Depending on the technique used, its complexity was between O(m) and $O(m^n)$, where m is the maximum number of stems studied in one sequence and n the total number of sequences [18].

In addition, Rivas and Eddy developed a dynamic programming algorithm (later implemented by Elena Rivas under the name of PKNOTS) based on thermodynamics to predict the optimal RNA secondary structure including pseudoknots, which had a worst case complexity of $O(n^6)$ in time and $O(n^4)$ in storage [32]. However, Jens Reeder created another practical pseudoknot folding algorithm based on thermodynamics and claimed to have an improved execution time in comparison to the Rivas and Eddy's approach. This recursive pseudoknots algorithm required $O(n^4)$ in time and $O(n^2)$ in space to predict the energetically RNA optimal structure usually containing pseudoknots [29].

Another algorithm, called NUPACK, was proposed by Dirks and Pierce to allow the design and analysis of secondary structures for nucleotide-based systems [47]. This algorithm was based on a partition function, thermodynamic models, base-pairing probabilities including pseudoknots or not, and recursive techniques to predict the RNA secondary structure. The partition function algorithm for an analysis that does not include pseudoknots took $O(n^4)$ and when some types of pseudoknots were considered, it took $O(n^5)$, where n is the length of the sequence [7].

Several methods using different RNA prediction techniques have been introduced. Jihong Ren et al. proposed HotKnots in both desktop and web-based methods. This approach was based on constructing stable stems iteratively, which allowed considering a wide range of multiple secondary structures using thermodynamics (minimum free energy) [31]. In comparison to other prediction software, HotKnots outperformed in terms of sensitivity and specificity in contrast to other dynamic programming-based algorithms such as the Rivas and Eddy's approach and the NUPACK algorithm of Dirks and Pierce, and also all other heuristic iterated loop matching-based algorithms proposed by Ruan and genetic-based algorithms such as the STAR package by van Batenburg [31].

Another heuristic algorithm for the prediction of RNA secondary structures called IPknots developed by Kengo Sato et al. to help in the prediction of RNA secondary structure considered pseudoknots by maximizing the expected accuracy of the resulting structures using an optimization technique called integer programming [33]. This method improved accuracy and prediction speed when compared to other competitive RNA prediction methods such as HotKnots, NUPACK, and PKNOTS [33]. However, it is capable of retaining its improved performance only when long sequences less than one thousands of nucleotides in length are used [33].

In addition, pknotsRG proposed by Kens Reeder et al. is a method that uses a dynamic programming algorithm to find potential secondary RNA structures based on minimum free energis, which are estimated using the Turner energy rules [30]. Moreover, RNAstructure prediction software is based on a method that allows the prediction of the RNA secondary structure based on either the minimum free energy or maximum expected accuracy [23]. In this study, both methods are used to assess the accuracy of the new proposed two-chunk elimination method.

Current prediction algorithms that are not based in computational thermodynamics or heuristics methods have shown great improvements in comparison to old-fashion algorithms. Iterative HFold is an algorithm that assists in the prediction of the RNA secondary structures considering pseudoknots by taking in consideration "pseudoknot-free structures" as input and generating potential pseudoknotted structures with at least the same energy as the input pseudoknotted structure [17]. This method improves both accuracy and performance in comparison to most of the current methods of RNA secondary structures prediction using pseudoknots such as IPknots, HotNnots, or ShapeKnots [17].

A recent approach proposed by Daniel Yehdego et al., to overcome the sequence length limitation established for the majority of the RNA prediction software due to computer resources dependencies is called chunk concatenation [45]. This approach improves the previous dynamic programming algorithm proposed by Abel Licon et al. for finding the optimal segmentations of RNA sequences for the prediction of RNA secondary structures [20]. This improvement took into consideration statistical information on inversions for cutting purposes instead of finding all possible ways of cutting RNA sequences [46].

The chunk concatenation method consists of five main steps including the inversions detection, exclusion plot creation, generation of chunks, chunks prediction, and concatenation of predicted chunks. The five main steps are described as follows:

1. Identify regions in each sequence with high concentrations of inversions and avoid cutting such regions (Figure 2.1a),

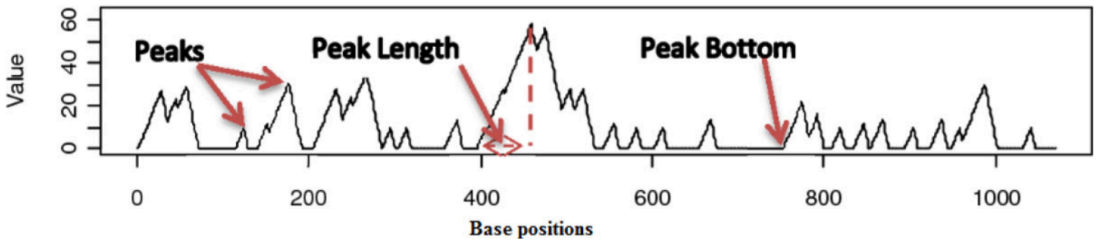



Figure 2.1a: Rising stretches in the plot indicate the presence of inversions.

2. Create excursion plots to identify where to cut in the sequence and determine how many chunks will be generated (Figure 2.1b).

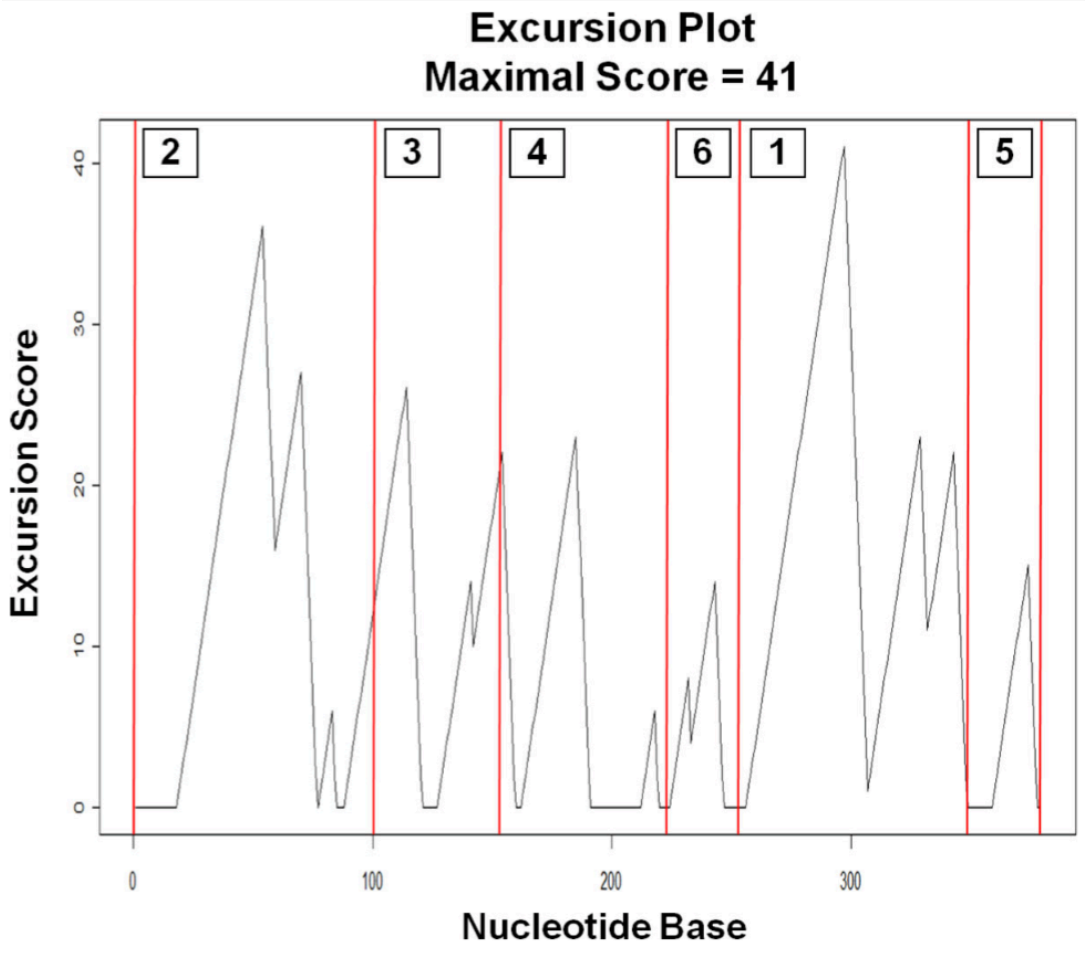



Figure 2.1b: Excursion Plot with Center Method.

3. Generate single chunks after cutting method is applied (figure 2.1c)

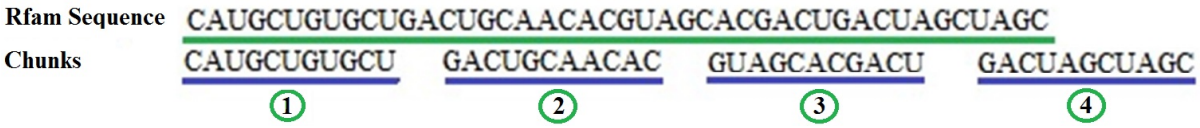



Figure 2.1c: Resulting chunks after chunk concatenation method is applied.

4. Once the chunking process is completed, RNA secondary structures for each chunk along with their minimum free energy (MFE) are predicted (Figure 2.1d).

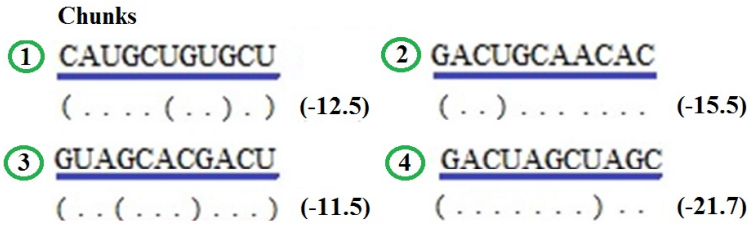



Figure 2.1d: Example of two chunks and their predicted structures

5. After the prediction process, chunks of sequences and structures are put together in the same order they were chopped to form the resulting RNA secondary structure prediction (Figure 2.1e)

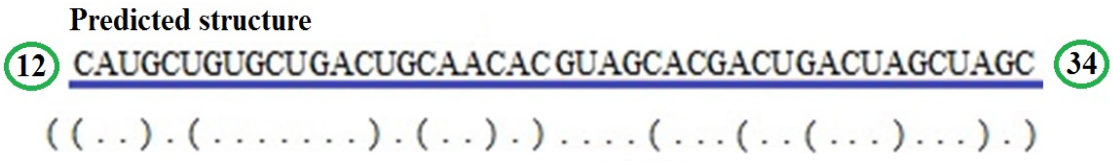



Figure 2.1e: Final predicted structure

The idea of cutting RNA sequences using the chunk concatenation method might lead to the assumption that accuracies obtained from the comparison between RNA real structures versus predicted RNA secondary structures could be impacted negatively. In addition, the integrity of the predicted RNA secondary structures could be altered due to the prediction of all chunks individually and concatenation of predicted chunks contiguously to form the final predicted structures. However, results obtained from previous work show that the chunk concatenation method outperformed other RNA secondary structure prediction software significantly [27]. For instance, the chunk concatenation method improved RNA secondary structures accuracies, when pknotsRG prediction software was used, in 75% of the cases in comparison to the RNA secondary structure predictions obtained when the whole sequences were used [27].

## 2.2 Other Bioinformatics applications using HTCondor

Due to the exponential growth of datasets in the Bioinformatics field area, some studies have relied on HTCondor functionalities to process the humongous amount of information available to analyze and to acquire performance benefits. HTCondor has been applied in different science areas such the Bioinformatics area, which deals with a great amount of biological data available to address biological questions. For instance, a project called BioCompute has employed the HTCondor techniques for executing batch bioinformatics jobs. The use of HTCondor was to distribute input queries across the HTCondor pool with the sole objective of conducting large BLAST searches by improving the runtimes. In this project, a daily load of 2310 sequences were processed by the HTCondor cluster of 32 nodes in 20 hours in comparison to a single node that would have taken more than three weeks to process the same amount of sequences [4].

Proteomics is another field where HTCondor has been utilized for distributing the execution of the heuristic BLASTP algorithm together with a sliding window approach. A protein segment of n amino acids is used with the BLASTP algorithm to find its similarities. Once this segment has been processed by BLASTP, a window fraction is moved one amino acid further down the protein sequence. The process is repeated until all possible regions of the protein sequence have been covered. This process ends up with an increasing amount of execution time because it is done sequentially. Thus, the execution time can be reduced by distributing different window sizes to different idle processors using the HTCondor grid computing [2].

In the same area, the automatic annotation of Glycosylphosphatidylinositol (GPI) structures is another study that has applied the HTCondor techniques to distribute chunks of information to idle machines to reduce execution time. The automatic annotation tool is a library-search algorithm that helps in identifying GPIs by matching fragment peaks in the spectra with molecular masses obtained from theoretical GPI structures. This algorithm was tested using the protozoan parasite Trypanosoma cruzi, agent associated with the Chagas disease. The algorithm for the prediction of the T.cruzi dataset was executed sequentially using a Dell PowerEdge T300 with four 64-bit processors, 8GB RAM memory and CenOS Linux, taking up to 10 days of continuous work for its prediction. The same T.cruzi dataset, along with the prediction algorithm has been submitted to the HTCondor pool, consisting of 31 processors in 64-bit machines running CentOS Linux with a speed ranging from 1 to 3.3 GHz and from 1 to 8 GB RAM, having as a result an improvement to 4 days in total to complete its prediction [1].

More recent studies outside the biology area include the assistance in different technological areas including the Clemson Center for Geospatial Technologies (CCGT) in the analysis of large amounts of geospatial data that takes 4.11 days to process sequentially by splitting it out into several small chunks to facilitate its processing and manipulation in 3 hours [5] and the parallelization of intensive stochastic computations with a hydrological model called Gridded Surface Subsurface Hydrologic Analyst (GSSHA) that allowed to minimize costs caused by using expensive cloud computer or supercomputers resources [38].

# Chapter 3: Methodology

An approach called the chunk concatenation method has been recently proposed in order to improve the efficiency of the prediction of RNA secondary structures of long sequences [21-24]. The method consists in cutting a long RNA sequence into shorter segments called chunks, carrying out RNA secondary structure prediction of chunks individually, and assembling resulting predicted chunk structures to form an overall predicted structure of the original sequence. In the present study, we investigated whether the prediction accuracy of the chunk concatenation method could be improved by applying a two-chunk elimination method that would capture possible structures formed between two neighboring chunks that would be missed by the usage of the chunk concatenation method. Furthermore, we have measured the waiting time for the RNA secondary structure prediction of 14 Nodavirus sequences using both a sequential approach and a high-throughput distributed batch computing system approach called HTCondor to determine whether waiting time for prediction could be reduced or not. Finally, we compared the overall prediction accuracies of regular, chunk concatenation, and two-chunk elimination prediction methods to determine for which chunk size would all methods be not statistically different in prediction accuracy.

## 3.1 Datasets

For further performance testing with real RNA data using distributed computing on the established HTCondor pool, we used two datasets as inputs to both RNAstructure and pknotsRG RNA prediction software to measure the prediction accuracy using the regular, chunk concatenation, and two-chunk elimination prediction methods and measurement of the execution time using HTCondor throughput computing tool when predicting long RNA sequences.

The first is a collection of Rfam sequences with known secondary structures containing pseudoknots. Rfam is a secondary structure database containing information about non-coding RNA families obtained from the use of sequence alignments and covariance models. Each family contained in the Rfam database represents a group of RNA sequences that function at the RNA level such as tRNA, microRNAs, and spliceosomal RNAs [12]. The second dataset is the 14 RNA genome sequences from the family of nodaviruses that represent non-enveloped, icosahedral, RNA viruses with diameters in the range of 25-34 nm considered the causative agents of viral nervous necrosis of marine fish [21, 22].

### 3.1.1 Rfam

Rfam is a group of non-coding RNA families based on multiple sequence alignments and covariance models [13]. Non-coding RNA molecules are functional molecules that are transcribed from RNA but never translated into a protein [24]. The main purpose is to regulate gene expression at both transcriptional and post-transcriptional levels [24]. There exist several RNA sequence databases that provide protein annotations for several types of RNAs such as Pfam, SMART, and Prosite. However, a proposed Rfam database has been implemented to group existing structural RNA alignments into a common structure annotation format, facilitate RNA database searches using covariance model software, and provide a way to facilitate the analysis and annotation of sequences using a web portal [24]. Each family contained in the Rfam database represents a group of RNA sequences that function at the RNA level such as tRNA, microRNAs, and spliceosomal RNAs [12]. In this study, the Rfam dataset utilized is comprised of 136 non-coding RNA sequences with real structures from the Rfam library (see Table 3.1.1a).

Table 3.1.1a: RFam dataset

| Sequence Name | Length | Sequence Name | Length | Sequence Name | Length | Sequence Name | Length | Sequence Name | Length | Sequence Name | Length |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RF00001_A | 117 | RF00026_B | 108 | RF00162_A | 103 | RF00215_B | 70 | RF00383_A | 117 | RF00483_B | 121 |
| RF00002_B | 150 | RF00028_B | 344 | RF00168_B | 179 | RF00220_A | 86 | RF00384_B | 62 | RF00484_A | 149 |
| RF00003_B | 161 | RF00030_A | 297 | RF00169_B | 102 | RF00225_B | 151 | RF00386_A | 90 | RF00485_A | 114 |
| RF00004_A | 145 | RF00031_A | 66 | RF00171_A | 168 | RF00230_B | 246 | RF00387_A | 168 | RF00487_B | 211 |
| RF00005_A | 72 | RF00035_B | 110 | RF00172_A | 81 | RF00231_A | 275 | RF00388_B | 102 | RF00488_B | 568 |
| RF00007_B | 129 | RF00036_A | 337 | RF00175_B | 114 | RF00232_A | 170 | RF00389_B | 158 | RF00489_A | 80 |
| RF00009_A | 320 | RF00040_B | 338 | RF00176_A | 91 | RF00234_B | 160 | RF00391_A | 171 | RF00490_A | 68 |
| RF00011_B | 374 | RF00045_B | 213 | RF00177_A | 482 | RF00290_A | 140 | RF00433_B | 152 | RF00492_B | 144 |
| RF00012_B | 219 | RF00048_A | 61 | RF00179_B | 71 | RF00362_B | 79 | RF00434_B | 122 | RF00493_A | 70 |
| RF00013_A | 183 | RF00050_A | 157 | RF00181_A | 86 | RF00363_B | 70 | RF00435_A | 109 | RF00494_A | 80 |
| RF00014_A | 85 | RF00059_B | 105 | RF00182_A | 129 | RF00364_A | 62 | RF00437_B | 131 | RF00497_B | 93 |
| RF00015_B | 141 | RF00100_A | 330 | RF00193_A | 273 | RF00365_A | 66 | RF00444_B | 147 | RF00503_A | 293 |
| RF00017_B | 305 | RF00102_B | 159 | RF00194_B | 89 | RF00366_B | 65 | RF00460_B | 74 | RF00506_B | 144 |
| RF00019_A | 84 | RF00106_B | 105 | RF00198_A | 104 | RF00367_B | 65 | RF00461_B | 309 | | |
| RF00020_A | 115 | RF00107_A | 76 | RF00199_B | 110 | RF00373_A | 133 | RF00463_A | 127 | | |
| RF00021_B | 116 | RF00109_A | 79 | RF00209_A | 379 | RF00374_A | 101 | RF00466_B | 93 | | |
| RF00023_B | 363 | RF00114_B | 115 | RF00210_A | 462 | RF00378_B | 107 | RF00467_A | 75 | | |
| RF00025_A | 152 | RF00161_A | 64 | RF00214_B | 93 | RF00382_A | 64 | RF00481_B | 100 | | |

### 3.1.2 Nodavirus

Nodavirus is a microscopic group of single stranded RNA viruses from the Nodaviridae family, which causes a disease called VNN (Viral Nervous Necrosis) that affects newborn baby fish. Nodaviruses are non-enveloped, icosahedral, RNA viruses with diameters in the range of 25-34 nm considered the causative agents of viral nervous necrosis of marine fish. The diseases produced by the Nodaviruses involve the central nervous system and the retina where they usually produce vacoulation and cell necrosis [45, 46]. These viruses are composed of two segments labeled RNA1 and RNA2 but sometimes they include an additional RNA3 segment. In addition, such viruses are found via microscopy, sensitive or molecular methods, or through immunological assays [35]. In this study, a collection of 7 groups consisting of 14 long RNA sequence segments with no real secondary structures from the Nodavirus family has been used. This collection includes black beetle virus (BBV), Boolarra virus (BoV), flock house virus (FHV), nodamura virus (NoV), pariacoto virus (PaV), and striped jack virus (SJV), ETNN virus [27] (see Table 3.1.2a).

Table 3.1.2a: Nodavirus dataset

| Specie | Sequence Name | Length |
|---|---|---|
| **BBV** | BBV.RNA1 | 3106 |
| | BBV.RNA2 | 1399 |
| **BoV** | BoV.RNA1 | 3097 |
| | BoV.RNA2 | 1322 |
| **ETNN** | ETNN.RNA1 | 3103 |
| | ETNN.RNA2 | 1433 |
| **FHV** | FHV.RNA1 | 3107 |
| | FHV.RNA2 | 1400 |
| **NoV** | NoV.RNA1 | 3204 |
| | NoV.RNA2 | 1336 |
| **PaV** | PAV.RNA1 | 3011 |
| | PAV.RNA2 | 1311 |
| **SJV** | SJV.RNA1 | 3107 |
| | SJV.RNA2 | 1421 |

## 3.2 Two-chunk elimination method

A new approach called two-chunk elimination is proposed to capture potential structures formed between two neighboring chunks that might be previously discarded by the chunk concatenation method. Finding these missing structures is crucial in the improvement of accuracy in the prediction of the RNA secondary structures. The two-chunk elimination method consists in six steps illustrated in Figure 3.2a.

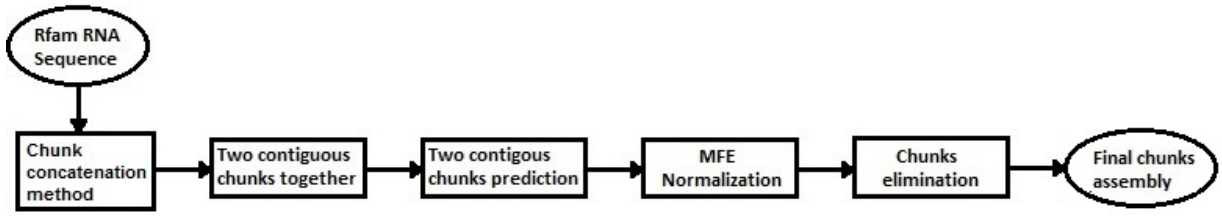



Figure 3.2a: Two-chunk elimination method workflow.

Two-chunk elimination method consists in:

a. Processing the Rfam dataset using the chunk concatenation method through the desktop application version called Segmenta [46] with chunk sizes between 60 and 120 and gap size between 0 and 8 as parameters.


Figure 3.2a: Example of four chunks using Segmenta software.

b. Once the chunk concatenation method is completed, resulting chunks are put together to form two-contiguous chunks

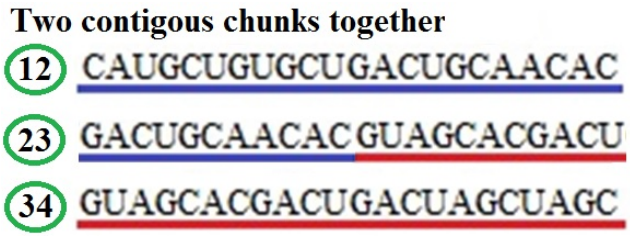



Figure 3.2b: Example of two-contiguous chunks.

c. Thereafter, the RNA secondary structure of each two-contiguous chunk is predicted using both RNAstructure and pknotsRG software (Figure 3.1c).

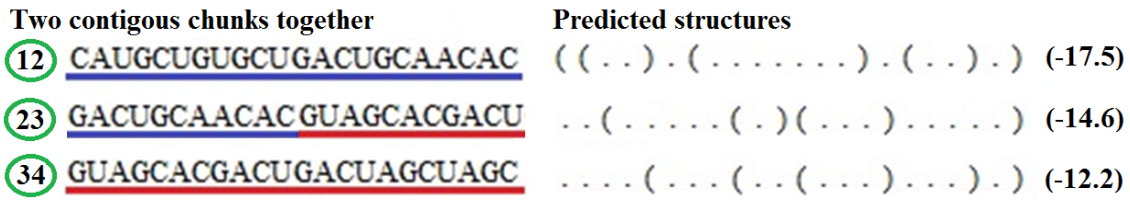



Figure 3.2c: Example of two-contiguous chunks and their predicted structures using RNAstructure.

d. For each two-contiguous predicted chunk structure, a normalized free energy (NFE) is calculated based on the MFE of the predicted chunk and the structure length using the following formula:

$$NFE = \frac{MFE}{Structure\ Length}$$

Figure 3.2d: NFE formula using MFE and Structure Length.

e. All structures are labeled with their respective individual or combined chunk number. Once enumerated, all structures pass through an iteration process where some of them are either kept or sacrificed based on their calculated NFE. In the first iteration, all structures are considered in the elimination process. The sacrificed structure is the one with less calculated NFE. When a structure is discarded, all others structures containing the sacrificed chunk are removed as well. The resulting structures participate in the next iteration until all structures have taken into consideration.

In Figure 3.2g, the structure labeled with number 2 is sacrificed in the first iteration for having the smallest NFE in comparison to all other structures. Therefore, all structures containing structure 2, such as structures 12 and 23, are sacrificed as well. Subsequently, structure 1, 3, 4, and 34 participate on the next iteration. In the second iteration, structure 34 is sacrificed for having the smallest NFE and as a consequence, structures 3 and 4 are sacrificed as

well. In the third iteration, structure 1 is the only one left so the resulting selected structures are 1, 2, and 34.

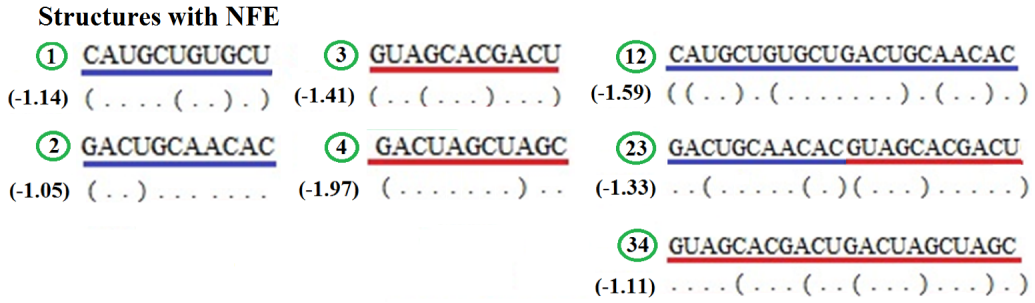



Figure 3.2g: Predicted structures and their NFE.

f. After the elimination process, structures 1, 2, and 34 are assembled to form the resulting RNA secondary prediction structure (Figure 3.2h).

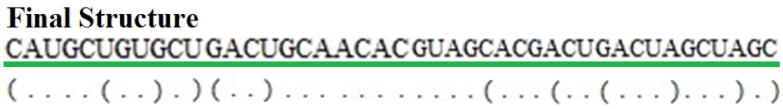



Figure 3.2h: Resulting predicted structure.

## 3.3 Overall Prediction Accuracies

The resulting RNA predicted structures using both pknotsRG and RNAstructure RNA secondary structure prediction software were compared to corresponding real Rfam RNA structures for accuracy assessment. The accuracy process consisted in comparing every resulting RNA predicted structure versus its corresponding RFam real structure. First, both RNA structures in dot-bracket notation were aligned with each other. Second, every position within the sequences was sequentially traversed to determine the number of matched structures found.

Finally, using Formula 3.3a, the accuracy percentage is calculated, which states how similar the resulting RNA structure was in comparison to the RNA Rfam real structure.

$$Accuracy\ Percentage = \frac{((2 * Total\ Parentheses\ Matched) + (1 * Total\ Dots\ Matched))}{Structure\ Length}\ x\ 100$$

Figure 3.3a: Accuracy percentage formula.

Furthermore, a Perl script called Accuracy.pl has been developed to automate the accuracy process described in this section (Appendix F). The Perl script navigates through both real and predicted Rfam structures and compares each position in bracket-notation format to find gaps or non-pairing bases represented by a dot or exact sub-structures in the structure. Sub-structures are represented by parentheses. If a sub-structure is found, the parenthesis is counted as a match; otherwise, as a mismatch. Once all matched dots and parenthesis are counted, the formula in figure 3.3a is applied to obtain the accuracy percentage. Because a parenthesis forms a structure, it is counted twice in the formula whereas the dot is counted only once. In addition, repeated measures ANOVA and T-Test were used to analyze the accuracy between regular, chunk concatenation, and two-chunk prediction methods under different chunk sizes (see accuracy results in Chapter 4).

## 3.4 General Description of the HTCondor pool

HTCondor is a high-throughput distributed batch computing system that allows submitting computationally intensive jobs to participant idle machines for their processing, based on a pre-established policy that dictates when, how, and in which node to run jobs [14]. It does not require having any special hardware or software installed prior to its configuration, however, it is recommended to have sufficient number of CPU's per computer participating in the pool,

approximately 25 Mbytes of memory for central manager's tasks for a pool of 100 machines in size, and a good HTCondor architecture design to manage network traffic efficiently through the collector, negotiator, and central manager daemons [15].

A configuration file, known as condor_config, is needed for the pool to work properly. Without this file, HTCondor has no way to understand what, where, when, and how to run submitted jobs. This configuration file needs to be setup for all nodes participating in the pool. This file includes the specification of the central manager, general information about the machine being configured, name of the pool, read-write privileges for each machine in the pool, ports where the communication takes place, and daemons list to enable central manager, submitters, and executers nodes. This configuration file needs to be consistent with the HTCondor architecture design and compiled using the install HTCondor command to start master, schedd, collector, and negotiator processes, depending on the configuration deamon list specified in the condor_config file.

### 3.4.1 HTCondor Features

HTCondor offers several features that would help in the processing of jobs. These features include [16]:

a. Checkpoint that allows pausing the execution of jobs and continuing their execution later were they left off and migration that allows migrating the execution of jobs from one machine to another without sacrificing the integrity of the jobs. This is extremely beneficial for big jobs that are computationally intensive because the network traffic might suffer problems that might interrupt the execution of jobs. Hence, they can be easily run after restoring the network or transferred to another pool or machines to complete their execution.

b. Remote system calls where the user is able to control the job submissions without the need of having special access to the pool. Hence, the user only needs to have access to any submitter computer belonging to the pool and start the execution of jobs without granting any special access to all people using the grid.

c. No changes to the HTCondor are needed when trying to use the pool for job submissions. The user needs to create just a submit file and use the commands without having to recompile any component.

d. The ability of running jobs submitted from one pool to another called Flocking, which makes the job submission process very flexible based on pre-established policies that dictate how and where to run jobs.

e. Job dependency that allows setting certain order to all jobs so they are executed according to configured rules.

f. The way HTCondor handles the computer resources usage by allowing the owners to take control of their computers transparently. This is possible because HTCondor is capable of using those computer resources for job submissions when they are really idle. Therefore, when HTCondor detects owners' activity in a specific computer resource, HTCondor releases such computer for its use.

### 3.4.2 General steps for running jobs in HTCondor

HTCondor requires certain steps to be done prior to the job submission in order to operate correctly. The general steps for running jobs in HTCondor are as follows:

1. The main executable programs used in the pool have to be able to run in the background and with no manual entries. HTCondor handles standard inputs and outputs internal and automatically so all parameters have to be sent in a submit file for their processing.

2. HTCondor requires a runtime environment, also known as universe; so all jobs can run and interact each other. In this study, the vanilla universe is the environment used which require specifying inputs and outputs in a config file to a shared drive, since this environment does not allow checkpoint and migrate features.
3. An HTCondor submit description file has to be created for the central manger to dictate all nodes participating in the pool that is time to execute jobs. This file contains information to allow executer nodes to run a specified executable program in the vanilla universe with one or more arguments as parameters, specify the location of the files to be read as arguments, specify the location of where to save resulting output, and policy rules that dictate under what circumstances a job must be executed (see Appendix B for an example of a submit description file).

4. The HTCondor command "condor_submit <submit description file>" has to be executed to tell the central manager to start the pool activity by calling the negotiator daemon to start analyzing which node is available in the pool, based on its policy established in the configuration file. After negotiation, a node is selected, the job is run, and the collector daemon gets the result and based on the HTCondor submit description file, the result is placed in the specified path, and the control is send back to the central manager to have another negotiation for processing jobs in the queue.

### 3.4.3 Bioinformatics HTCondor pool configuration

The bioinformatics program at the University of Texas at El Paso has a fully configured HTCondor pool composed of 26 computers from three bioinformatics computing laboratories (BCLs). The HTCondor pool is made up of 12-core processor node with 64 GB of RAM running a submitter daemon, one Intel quad-core processor with 16 GB of RAM acting as a central manager with both collector and negotiator daemons, 7 Intel quad-core processors with 8 GB of RAM, 15 Intel quad-core processors with 16 GB of RAM, and 2 Intel 8-core processors nodes with 8 GB of RAM, totaling 104 nodes running executer daemons (Figure 3.4.2a).

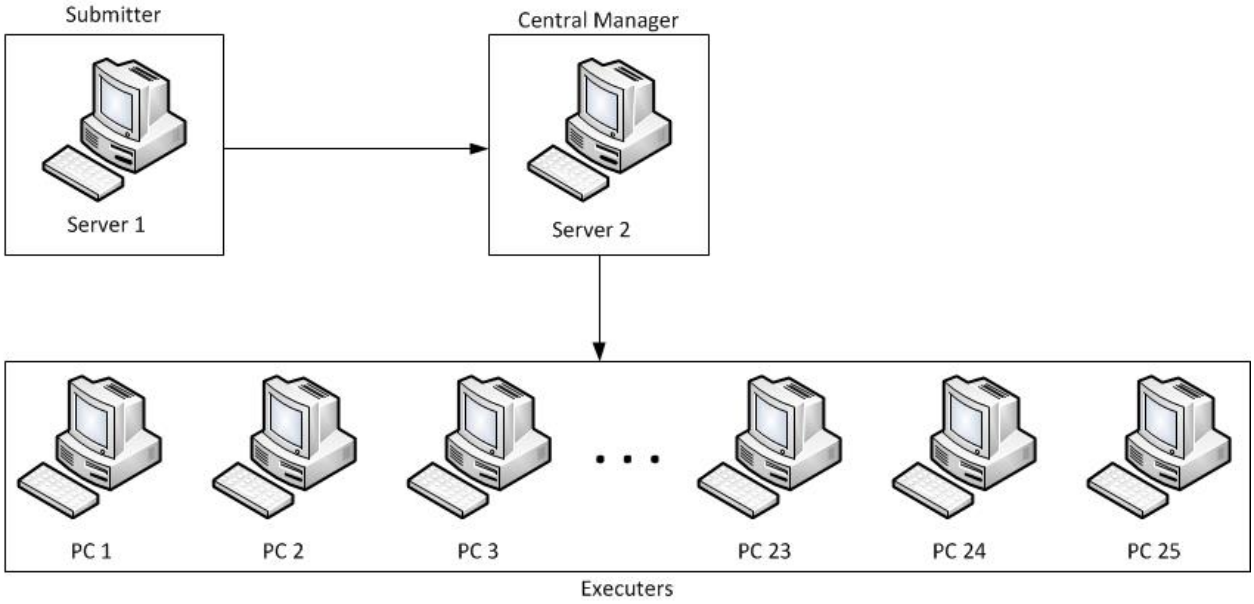



Figure 3.4.2a: BCLs HTCondor Pool Design Architecture.

RNA secondary structure prediction is a time-consuming and labor-intensive process, which is dependent of either computer architecture used or the sequence size to be predicted. The waiting time for the secondary structure prediction of RNA sequences using the pknotsRG program [6] increases when running sequentially as the number and length of the sequences becomes larger. For small datasets, the waiting time is not a significant problem since it can be

done in minutes or even hours; however, for larger datasets sizes, the waiting time increases rapidly due to the extensive manipulation that needs to be done for their prediction. The objective of using the HTCondor pool is to reduce the waiting time for completion of the secondary structure prediction of a dataset of RNA sequences. We have measured the waiting time for the RNA secondary structure prediction of 14 sequences obtained from Nodavirus dataset using both sequential approach and HTCondor throughput computing to determine whether waiting time prediction would be reduced or not.

### 3.4.3 Bioinformatics HTCondor pool submission process

A Perl script called 1_FormatGeneratorBinfoserv02.pl has developed to generate submit description files for each Nodavirus sequence chunk automatically using pknotsRG (Appendix C). In the same way, a Perl script called 2_FormatGeneratorBinfoserv02.pl has been developed to generate submit description files using RNAstructure (Appendix H). These scripts generate two text files called directories.txt and files.txt. The former contains a list of 14 directories where each one represents a group of Nodavirus dataset. The latter contains a list of file names where each one represents a Nodavirus RNA sequence chunk.

HTCondor requires a formatted submit description file containing information about the job to be executed. Therefore, the scripts create temporal submit files based on previous text files generated, containing the location of a Perl script called RNAPredExec_1.pl (Appendix D) and RNAPredExec_2.pl (Appendix I), which call pknotsRG and RNAstructure prediction software, respectively, a vanilla runtime environment, the linux platform type, the e-mail notification deactivated, the chunk sequence name with the ".out" extension as the output filename, and a RNA sequence of each Nodavirus sequence chunk as argument (Appendix D). The temporal submit file is sent to the HTCondor pool via condor_submit command through a command-line call, which generates output files containing RNA secondary structure prediction of each

Nodavirus sequence chunk.

### 3.4.5 HTCondor measurement of waiting time

HTCondor does not have any function that allows measuring the overall waiting time of the execution of a job from start to end. However, HTCondor offers different commands that can be manipulated to estimate the waiting time of jobs. Two commands that can be used are "condor_history" and "condor_q". The former gives information of each executed job along with a timestamp whereas the latter allows querying the uncompleted jobs' queue.

Therefore, a Perl script called CheckSubmitCompletion.pl was created to measure the waiting time of the RNA secondary structure prediction using HTCondor. At the beginning of the program, the "date" Perl function was used to set the initial execution time. Moreover, the Perl script ran the "condor_q" HTCondor command every 5 seconds to check whether the HTCondor queue was empty or not. Once the HTCondor jobs queue was empty, the "date" Perl function was executed after the HTCondor pool completed the job execution. At the end, a subtraction between the final date and start date was done to come up with the non-sequential waiting execution time.

Table 3.4.5: HTCondor Waiting Time Measurements

| Nodes | Run | Time (min) |
|---|---|---|
| 4 | 1 | 90 |
| | 2 | 90 |
| | 3 | 90 |
| 8 | 1 | 10 |
| | 2 | 10 |
| | 3 | 9 |
| 12 | 1 | 7 |
| | 2 | 6 |
| | 3 | 6 |

| | | |
|---|---|---|
| 16 | 1 | 5 |
| | 2 | 5 |
| | 3 | 5 |
| 32 | 1 | 4 |
| | 2 | 3 |
| | 3 | 3 |
| 64 | 1 | 3 |
| | 2 | 3 |
| | 3 | 4 |
| 104 | 1 | 4 |
| | 2 | 4 |
| | 3 | 4 |

### 3.4.6 Bioinformatics sequential submission process

A Perl script called 2_FormatGeneratorBinfoserv02.pl has been developed to predict the RNA secondary structure for each Nodavirus sequence chunk sequentially (Appendix E). This script generates two text files called directories.txt and files.txt. The former contains a list of 14 directories where each one represents a group of Nodavirus dataset. The latter contains a list of file names where each one represents a Nodavirus RNA sequence chunk. This script reads each Nodavirus RNA sequence chunk and predicts the RNA secondary structure using pknotsRG prediction software. The resulting RNA structure predictions are saved into a shared drive.

### 3.4.7 Sequential measurement of waiting time

The sequential execution time measurements of the RNA secondary structure prediction were performed as part of the Perl script called 2_FormatGeneratorBinfoserv02.pl. The measure is obtained by using an internal Perl function called date, which is specified just before the code where the prediction of each RNA sequence chunk using pknotsRG program starts and just after the last chunk of RNA sequence has been predicted. This measurement was performed three times and the times were averaged to state the final waiting time.

Table 3.4.5a: Sequential Waiting Time Measurements

| Run | Time (min) |
|---|---|
| **1** | 90 |
| **2** | 90 |
| **3** | 90 |

### 3.4.8 Methods for Nodavirus RNA predicted structures assessment

The Nodavirus RNA prediction structures from the fourteen (14) families of Nodavirus genomes were obtained in only four (4) minutes on average using HTCondor. Unfortunately, the real RNA secondary structures from the fourteen families of Nodavirus genomes have not been obtained yet. As a consequence, a true comparison for accuracy assessment could not be done because there is not any real structure to compare the predicted structures obtained by pknotsRG and RNAstructure. However, based on the predicted RNA structures, two different methods to compare the resulting predicted structures were proposed to analyze and find general patterns or differences across the Nodavirus family.

The first proposed method was to predict all possible RNA secondary structures for each of the fourteen Nodavirus sequences using the chunk concatenation method via Segmenta. Once all RNA sequence chunks were generated, pknotsRG and RNAstructure prediction software was used to predict RNA structures considering pseudoknots with MFE. Thereafter, a Perl script called 5_Assembly_with_MFE_and_extra_file.pl was executed to export all RNA structure names with their respective MFE in a file called SummaryFinalMFE.txt. Due to the Segmenta software, chunks from different gaps and stem lengths were generated for each Nodavirus RNA sequence. This file was imported to Excel and modified manually to obtain the sequence representative from each family, which had the MFE.

The second proposed method was to use a pseudoknot visualization tool called PseudoViewer3 to visualize the bracket notation format that each RNA predicted structure has in a human readable manner. PseudoViewer3 is a XML web-based tool that allows visualizing

large-scale RNA secondary structures with different types of pseudoknots [44]. A basic interaction of the PseudoViewer3 web-based tool and pknotsRG is depicted in Figure 3.4.8a.

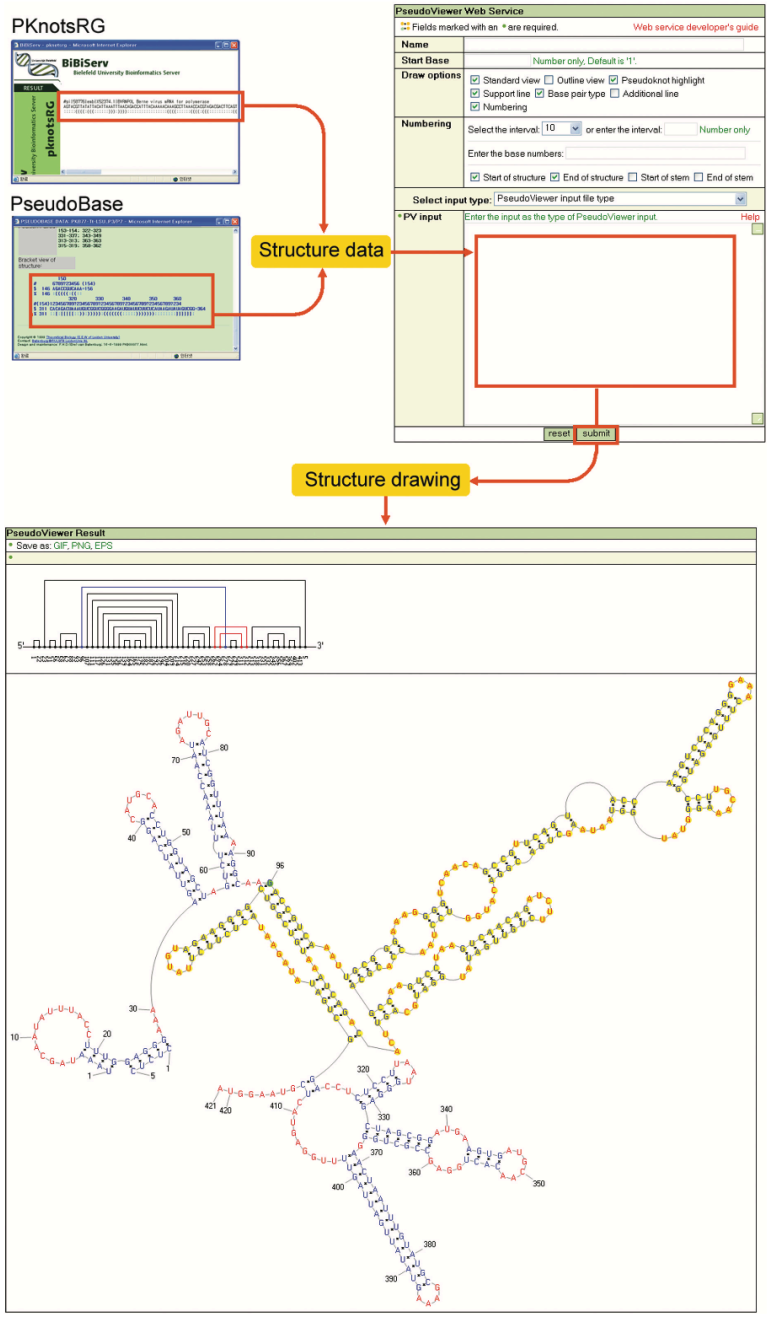



Figure 3.4.8a: PseudoViewer3 with pknotsRG.

# Chapter 4: Results

## 4.1 Accuracy comparison between regular, chunk concatenation, and two-chunk elimination methods

The pknotsRG and RNAstructure prediction programs were used to predict the RNA secondary structure and free energy for the chunks of Rfam family sequences generated by the chunk concatenation and two-chunk elimination methods. For each chunking method, chunk size between 60 and 120 in increments of 10 where used. In addition, both prediction programs were used to predict the RNA secondary structure using the regular prediction method; that is, submitting the whole RNA sequence and letting both programs deal with all calculations and not considering any chunking process.

The resulting Rfam predicted secondary structures from all three methods were compared to the Rfam real secondary structures for accuracy assessment for each chunk size combination (see Appendix F for a detailed information on accuracy assessment). Finally, statistical methods such as t-test [42] and repeated measures ANOVA [10], which is basically a one-way ANOVA for related but not independent groups, were used to determine whether the related accuracy means, obtained from two different methods, were statistically different or not. Conclusions about the significance of accuracy means were based on the p-values obtained from each method with a significance level of 0.05 (see Table 4.1a).

Table 4.1a: P-values summary from t-test and ANOVA using both pknotsRG and RNAstructure

| p-values | RNA Prediction | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | pknotsRG | | | | RNAstructure | | | |
| | t-test | | | ANOVA | t-test | | | ANOVA |
| | R vs O | R vs O | O vs T | All | R vs O | R vs O | O vs T | All |
| C60 | 0.02015 | 0.000519 | 0.03012 | 0.000248 | 3.28E-06 | 5.16E-09 | 1.18E-06 | 5.66E-13 |
| C70 | 0.03851 | 0.01446 | 0.4597 | 0.0121 | 0.00208 | 3.65E-06 | 9.37E-05 | 1.49E-07 |
| C80 | 0.2109 | 1.42E-01 | 0.6305 | 0.237 | 0.001449 | 1.25E-04 | 0.0001608 | 3.20E-06 |
| C90 | 0.2055 | 2.07E-01 | 0.8266 | 0.266 | 0.005661 | 2.58E-03 | 0.01011 | 0.000199 |
| C100 | 0.4131 | 4.99E-01 | 0.9692 | 0.619 | 0.1703 | 8.93E-02 | 0.03293 | 0.0831 |
| C110 | 0.1755 | 1.74E-01 | 0.6232 | 0.132 | 0.06142 | 2.38E-02 | 0.04254 | 0.012 |
| C120 | 0.9254 | 5.44E-01 | 0.4126 | 0.727 | 0.2342 | 1.66E-01 | 0.02739 | 0.175 |

Furthermore, seven box-plots representing chunks between 60 and 120 in increments of 10 were created for both pknotsRG and RNAstructure to visualize graphically how the distribution of accuracy means among all three methods was through their quartiles. In Figure 4.1a, box-plots are illustrated where chunk concatenation and two-chunk elimination methods using pknotsRG were not statistically different starting at chunk size of 70 and above.

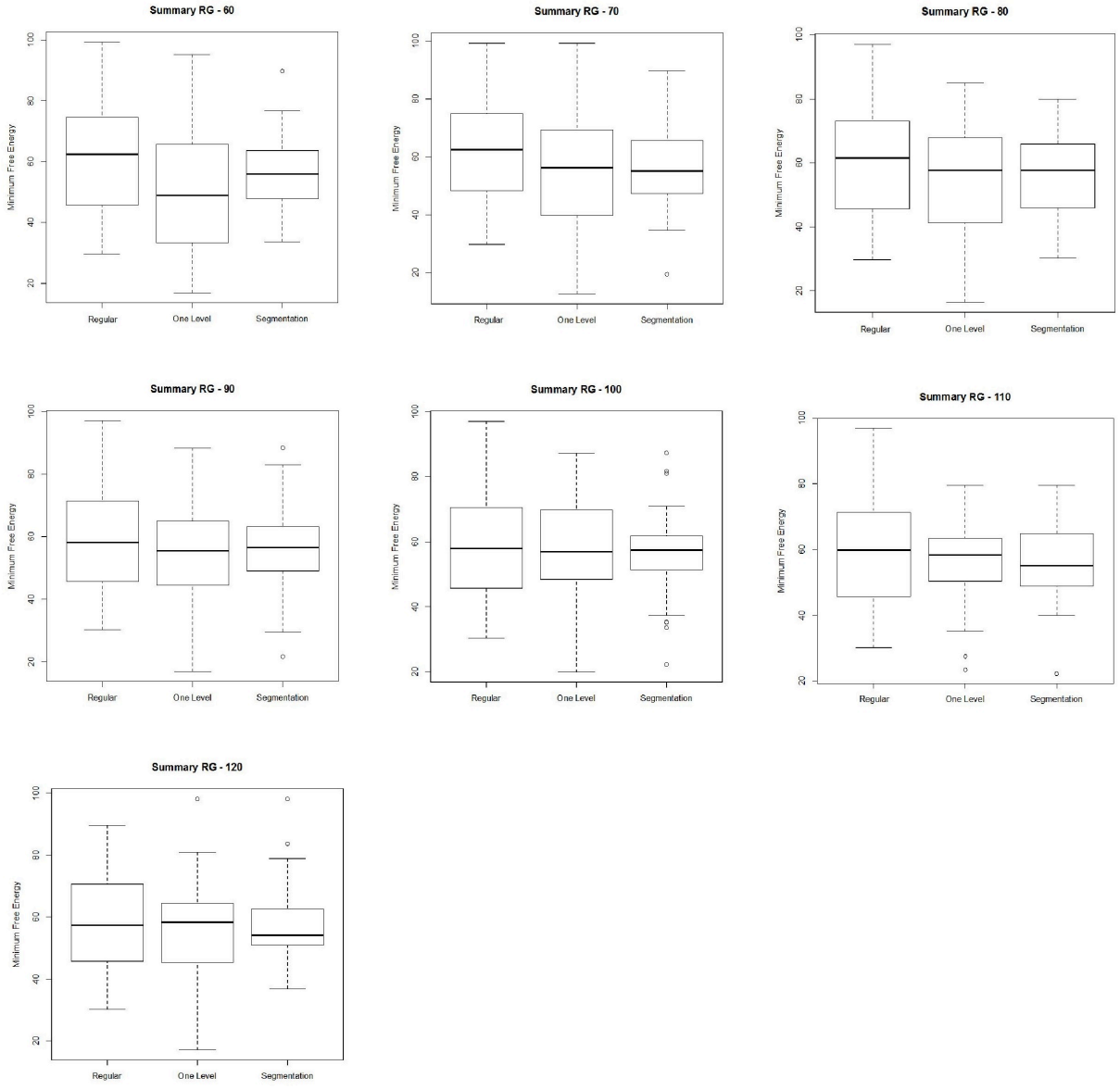



Figure 4.1a: Box-plots with chunk size between 60 and 120 using pknotsRG.

In Figure 4.1b, box-plots are illustrated where chunk concatenation and two-chunk elimination methods using RNAstructure were not statistically different starting at chunk size 90 and above.

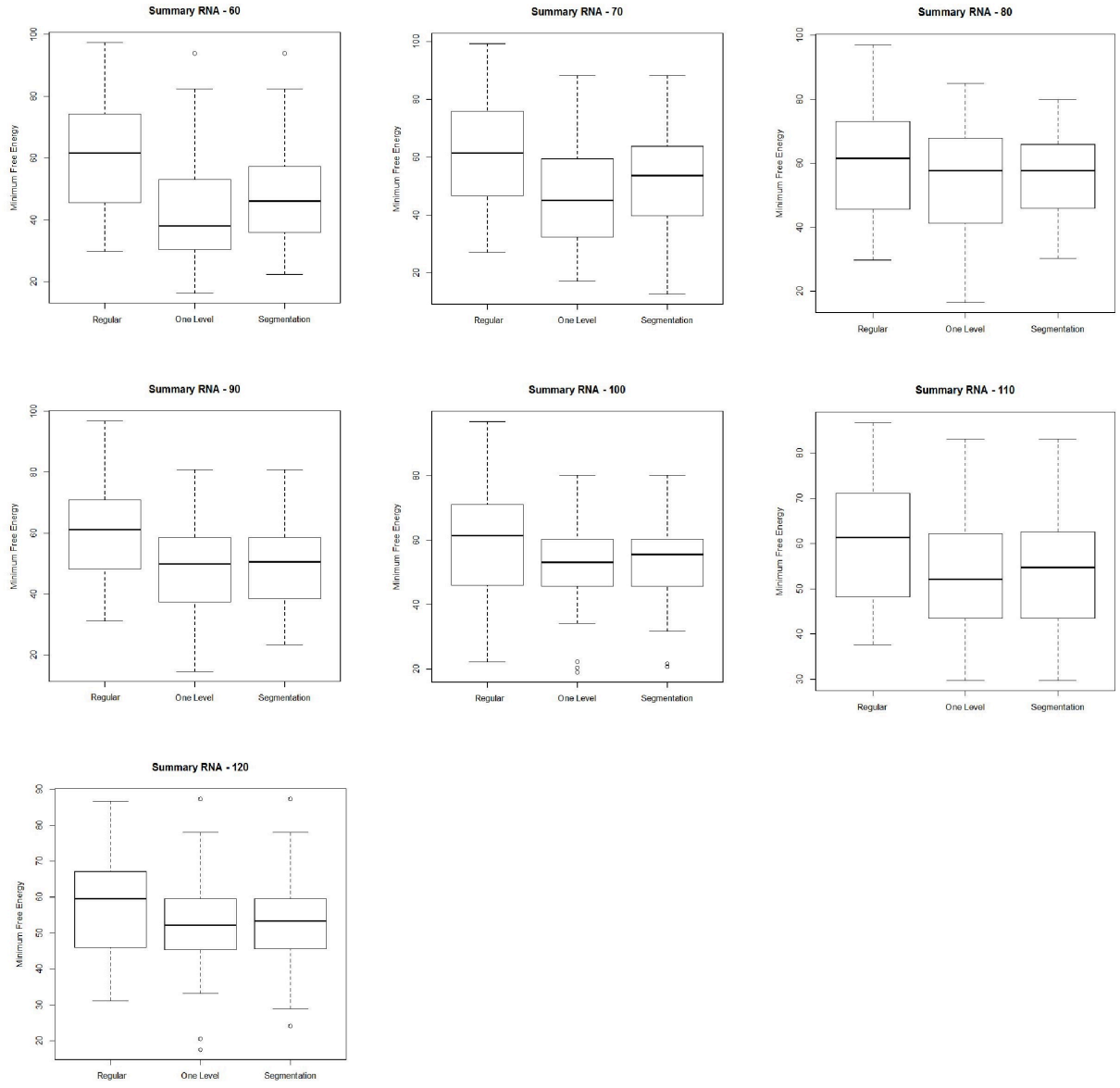



Figure 4.1b: Box-plots with chunk size between 60 and 120 using RNAstructure.

Based on Table 4.1a, repeated measures ANOVA analysis using pknotsRG prediction software states a p-value of 0.237 when chunk size is 80, leading to the conclusion that both chunk concatenation and two-chunk elimination methods are not statistically different when chunk size is greater than 70 bases long. On the other hand, t-test using pknotsRG states a p-value of 0.4597 when chunk size is 70, leading to the same conclusion that both chunk concatenation and two-chunk elimination methods are not statistically different when chunk size is greater than 60.

In the same way, RNAstructure prediction software stated a p-value of 0.0831 when chunk size was 100 based on a repeated measures ANOVA analysis, leading to the conclusion that both methods are not statistically different when chunk size is greater than 90 bases. In addition, t-test using pknotsRG stated a p-value less than 0.05 for any chunk size, leading to the conclusion that both methods are statistically different no matter what the chunk size is used.

In summary, two-chunk elimination method does not contribute to increased accuracy prediction in comparison with the existing chunk concatenation method when chunk size is greater than 70 bases long using pknotsRG and greater than 90 using RNAstructure prediction software.

## 4.2 Nodavirus Prediction Results using HTCondor

Thanks to the HTCondor high-throughput computing, long sequences of RNA, such as the ones pertaining to the Nodavirus dataset, could be predicted and obtained in less waiting time (see Section 3.4.5 for time measurements using different CPU's) than the waiting time either specified in the pknotsRG and RNAstructure documentation or measured using a sequential RNA secondary structure prediction process (see Section 3.4.7 for time measurements using a sequential execution).

The Nodavirus RNA prediction structures from the fourteen (14) families of Nodavirus genomes were obtained in only four (4) minutes on average using HTCondor. Unfortunately, the real RNA secondary structures from the fourteen families of Nodavirus genomes have not been obtained yet. As a consequence, a true comparison for accuracy assessment could not be done because there is not any real structure to compare the predicted structures obtained by either pknotsRG or RNAstructure. However, based on the predicted RNA structures, two different methods to compare resulting predicted structures were proposed to try to find general patterns or differences across the Nodavirus family (see Section 3.4.8 for an explanation of these proposed methods).

The first proposed method was to predict all possible RNA secondary structures for each of the fourteen Nodavirus sequences using the chunk concatenation method via Segmenta and using pknotsRG. As a result, Table 4.2a shows the list of fourteen Nodavirus RNA sequence names with the predicted MFE.

Table 4.2a: RNA secondary structures with MFE using HTCondor

| Sequence | MFE |
|---|---|
| BBV.RNA1_L3G0M0C400 | -936.99 |
| BBV.RNA2_L4G1M0C400 | -450.95 |
| BoV.RNA1_L4G5M0C400 | -995.65 |
| BoV.RNA2_L4G5M0C400 | -374.39 |
| ETNN.RNA1_L3G0M0C400 | -1057.8 |
| ETNN.RNA2_L3G5M0C400 | -487.58 |
| FHV.RNA1_L3G0M0C400 | -936.85 |
| FHV.RNA2_L5G1M0C400 | -446.1 |
| Nodamura_RNA11_L3G0M0C400 | -1280.38 |
| Nodamura_RNA22_L3G5M0C400 | -409.98 |
| PaV.RNA1_L3G4M0C100 | -756.33 |
| PaV.RNA2_L4G1M0C400 | -457.4 |
| SJV.RNA1_L3G2M0C400 | -1110.3 |
| SJV.RNA2_L3G0M0C400 | -484.6 |

Based on these results, an identified general pattern resides in the gap and stem length parameters. It seems that the best RNA predicted structures based on the MFE had stem length between 3 and 5 whereas the gap size was between 0 and 5 on most sequences and only two had a gap size of 2 and 4.

In addition, there is no available tool that estimates the MFE of RNA structures considering pseudoknots. There is a program called RNAeval from the Vienna package [22] that estimates the MFE of a given structure but it does not consider pseudoknots. Hence, the method used in this study to estimate the MFE was to add up all MFE from all predicted RNA chunks of each sequence during the assembly process. Therefore, another conclusion could be based on the estimated MFE depicted in table 4.2a. It is noticeable that the MFE of each sequence is relatively high in comparison to the Rfam. This behavior can be justified based on the thermodynamics MFE concept because pseudoknots would contribute more to the stabilization of the RNA structures in comparison to non-pseudoknots RNA structures. Based on the MFE, the more negative the energy, the more stable the RNA secondary structure is.

The second proposed method was to visualize all RNA predicted structures described in table 4.2a using the PseudoViewer3 web-application. Given the structure length limitation of PseudoViewer3, we were able to generate images for the first four sequences in table 4.2a. For the rest of the RNA predicted structures, the web application threw a timeout error. The visualization of predicted RNA structures are depicted in Appendix G. In general terms and just by looking at the illustrations, there is no any relationship between these RNA predicted structures. However, there might be some regions that can match if a structure alignment were performed.

The same analysis could not be possible using RNAstructure prediction software for the way the prediction software handles the structure prediction. This process consists in running

three programs sequentially called Fold, RemovePseudoknots (if required), and ct2dot, where each program depends on the antecessor. HTCondor does not control the dependency workflow of those programs when jobs are distributed. Therefore, the waiting time could not be measured efficiently due to the fact that HTCondor would only act as a distributor of jobs, rather than as a negotiator and collector of jobs.

### 4.3 Comparison of waiting times using sequential method and HTCondor

Prediction of RNA secondary structures for Nodavirus sequences are considered computationally intensive because each viral genome consists of two pieces of RNA, called RNA1 and RNA2, about 1300 and 3200 bases long, respectively. The same problem occurs with many RNA molecules, such as those making up viral genomes, which are thousands of bases long. Yet, most of the computational tools for the prediction of RNA secondary structures, including pknotsRG and RNAstructure, have a sequence length limitation of few thousands of nucleotide bases due to their high demands on computer resources.

For example, prior versions of pknotsRG were able to predict a sequence of length 200 in 80 seconds whereas a recent version of pknotsRG was able to predict structures with the same sequence length in only 1 second and up to 1000 nucleotides in 10 minutes with 31 MB of memory consumption [30]. In addition, the RNAstructure webserver has established a sequence length limitation of no more than 2500 nucleotides per sequence because there is a limitation in terms of server demand and hardware in the webserver [23]. However, the downloadable software version of RNAstructure does not have any sequence length restriction due to the fact that the only restriction established is based on the hard disk and memory RAM used on the machine executing the RNAstructure program [23].

Due to these hardware-dependent limitations, a high-throughput computing technology called HTCondor has been used to reduce the waiting time pknotsRG and RNAstructure prediction software take to predict RNA structures when dealing with sequences of thousands of nucleotides in length and considering pseudoknots. The predictions of special RNA structures, called pseudoknots, involve many computationally intensive operations that potentially would cause a delay when the RNA structures are being predicted.

The Bioinformatics HTCondor pool has been adapted to run on 1, 4, 8, 12, 16, 32, 64, and 104 nodes. The waiting time has been measured when using both the traditional sequential execution of pknotsRG and the distributed computing design. Furthermore, the speed-up has been calculated based on the serial and parallel execution times to determine how fast the RNA secondary structure is predicted by distributing chunks versus predicting it sequentially. Speed-up is calculated by dividing the elapsed time needed by one processor (sequential execution) divided by the time needed on $p$ processors [19]. Finally, the efficiency has been calculated to determine how effectively the CPU's are being used versus how much time is spent on synchronization and communication [19]. Efficiency is calculated by dividing the speed-up using $p$ processors divided by $p$ processors.

The sequences from the Rfam dataset used in this study are not more than 600 nucleotides in length. For this reason, all sequences from the Nodavirus dataset have been used because they have thousands of nucleotides in length. Given that the RNA secondary structure prediction limitation resides in the hardware that is utilized during the prediction process and that both chunk concatenation and two-chunk elimination methods are not statistically different when chunk size is greater than 70 nucleotides in length for pknotsRG and 90 nucleotides in length for RNAstructure, the two-chunk elimination method was used to chomp all Nodavirus sequences using a chunk sizes between 100 and 500. Moreover, both pknotsRG and RNAstructture prediction software were set up to allow pseudoknots prediction with the sole purpose of

increasing the number of calculations done when the prediction of the RNA secondary structure was performed.

The resulting waiting times were generated along with speed-up and efficiency calculations to assess how beneficial the RNA secondary structure prediction using pknotsRG and RNAstructure is when HTCondor was involved. Different chunk sizes between 100 and 500 nucleotides were used to determine how valuable the use of HTCondor was when trying to predict RNA structures from multiple chunks obtained from a sequence with thousands of nucleotides using both pknotsRG and RNAstructure programs (see Appendix AA).

Based on these attempts, HTCondor did not improve the waiting time when chunk size was less than 400. In some cases, it had less performance in comparison to a sequential (one single node) execution. However, a considerable improvement was noticed when chunk size was 400 (see Table 4.3.1a).

Table 4.3.1a: Waiting time, speed-up, and efficiency measurements of pseudoknots sequences of chunk size of 400 using pknotsRG

| Nodes | Time (min) | Speed-up | Efficiency |
|---|---|---|---|
| 1 | 110 | 1 | 1 |
| 4 | 19 | 5.79 | 1.44 |
| 8 | 10 | 11 | 1.37 |
| 12 | 7 | 15.71 | 1.3 |
| 16 | 5 | 22 | 1.37 |
| 32 | 4 | 27.5 | 0.86 |
| 64 | 4 | 27.5 | 0.43 |
| 104 | 4 | 27.5 | 0.26 |

Clearly, a comparison between CPU nodes used in the HTCondor pool at UTEP and waiting time elapsed for the RNA secondary structure prediction suggests that with a sequential execution (regular submission process), the RNA structure prediction took 110 minutes in comparison to 4 minutes when 32 and above nodes are available (Figure 4.2a).

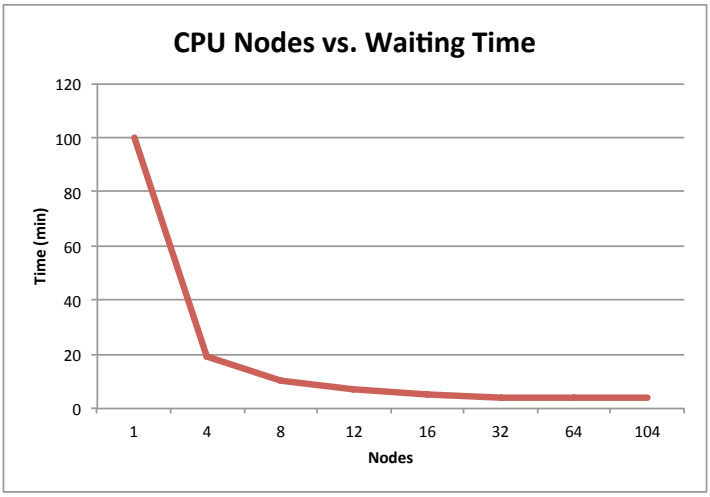



Figure 4.2a: Graph of CPU Nodes vs. Waiting Time

There is also a great improvement in terms of speed-up when HTCondor is used. The speed-up of a sequential run has been improved up to 32 nodes, where the graph becomes steady.

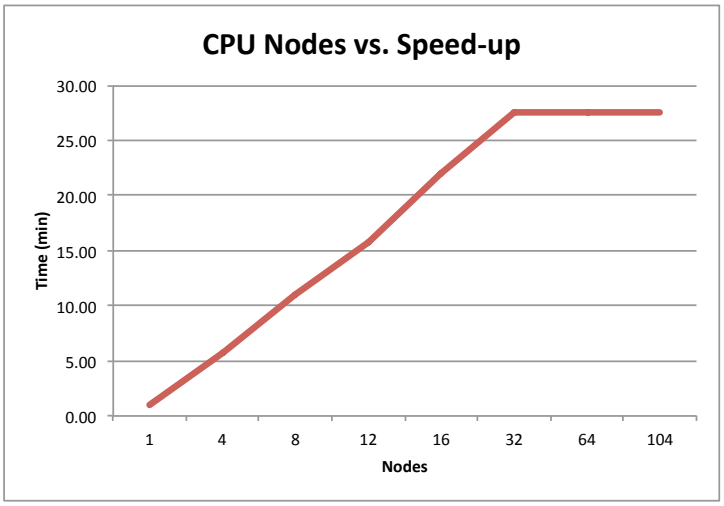



Figure 4.2b: Graph of CPU Nodes vs. Speed-up

Finally, there were some improvements in efficiency when HTCondor is used. The efficiency of a sequential run was improved at a certain point when 16 nodes were used. The efficiency was impacted negatively when more than 16 nodes were used in the prediction process.

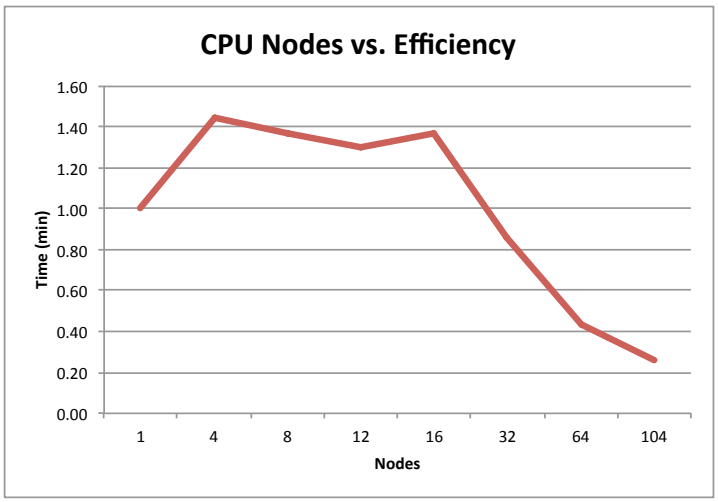



Figure 4.2c: Graph of CPU Nodes vs. Efficiency

# Chapter 5: Conclusions and Future Development

## 5.1 Conclusions

One of the problems with the RNA Secondary Structure prediction is the heavy handling and processing of long RNA sequences due to the limitation of computer hardware and software restrictions. The pknotsRG and RNAstructure software were able to process long RNA sequences faster and with high accuracy when the two-chunk elimination method was applied by cutting long RNA sequences into chunks in precise locations to reduce the loss of the original RNA structure. In the end, it was determined that these methods were not statistically different when chunk size increased more than 70 nucleotides in length for pknotsRG and 90 nucleotides in length for RNAstructure prediction software.

Furthermore, HTCondor throughput computing software was used to alleviate the waiting time when processing not only long RNA sequences but also the chunks generated by the chunk concatenation and two-chunk elimination methods. The current HTCondor architecture design did not allow measuring the CPU execution time, only the real waiting time. This is because the central manager is not able to monitor every node activity when a job is being executed or finished, a factor that would contribute to have a humongous overhead in communication. Furthermore, the waiting time using distributed computing was not measured with high accuracy because HTCondor does not offer the capability of monitoring the status queue in a smaller time scale. Hence, a linux crontab job was used to check the HTCondor queue every five seconds, introducing some noise in the waiting time measurement results.

Therefore, considering just the waiting time, the Bioinformatics HTCondor pool was able to reduce drastically the waiting time of the RNA secondary structure prediction in comparison to the sequential prediction execution, from 110 minutes with one single CPU to 4 minutes with 32 CPU nodes and above.

Due to the bottleneck caused in the central manager by the HTCondor communication process, the constant queue status check every five seconds, and the number of chunks generated after the chunk concatenation and two-chunk method was applied, the waiting time increased at some degree. However, it could be controlled by modifying the maximum chunk-length in the centered cutting method and the maximum jobs (shadows) running on each node, the scheduler queue size, and how and when to run jobs in the HTCondor configuration file.

## 5.2 Future Development

For future work, the current HTCondor hardware has to be upgraded with more efficient and powerful resources to improve the speed-up and efficiency of job distribution and output gathering process since it is expected to have thousands of chunks generated and to be predicted. Moreover, the current HTCondor pool configuration must be adjusted to accommodate the addition of new computer resources.

A first hardware change would be to set in place a powerful server as a central manager to have efficient management in both the collection of information about every node participating in the pool and the negotiation (match-making) between the nodes participating in the pool and the number of jobs, based on a policy, to be processed.

A second hardware configuration would be to set in place a powerful computer acting as a submitter and executer machine in order to have efficient management in both the submission of jobs accumulated in a local queue to the central manager and the execution of queued jobs sent back from central manager for processing.

A third configuration change would be at the HTCondor root level to adjust the number of nodes acting as submitter machines to allow a better control of the local jobs queue and the

maximum jobs that can be running simultaneously. As part of this configuration change, security roles have to be created so authorized people can only use the HTCondor pool and submit jobs with priority for unknown people.

In addition, there is a vision of establishing a collaboration plan between different institutions using HTCondor computing. For instance, Purdue University runs the largest condor pool spanning multiple institutions in a DiaGrid project (http://diagrid.org) with more than 10,000 CPUs available for processing jobs. There would be a tremendous benefit for high computing demand projects to use an interconnected grid computing not only in the Bioinformatics program HTCondor pool but also to use other nodes including Purdue University resources with the sole purpose of achieve desired goals faster and efficiently.

There is also a plan to create a HTCondor web-based version for both chunk concatenation and two-chunk elimination methods so many applications hosted at the University of Texas at El Paso such as Oncominer, ISOGlyP, GPI-Mass Spectrometry Data Digest can be used by external researchers to get faster results. Currently, these programs run in a sequential fashion so the introduction of HTCondor would help in alleviating the computationally intensive jobs by distributing them into the nodes participating in the HTCondor pool to have reliable and faster results.

It would also be convenient to develop an algorithm for predicted RNA secondary structure alignment. Since there is no real RNA secondary structure to compare against, the accuracy measurement would be almost impossible, unless there would be a mechanism of comparing RNA secondary structures, with or without considering pseudoknots, among themselves. This algorithm would facilitate the accuracy measurement for the predicted RNA secondary structures that do not have any real RNA secondary structures associated. In addition, it would be possible to align predicted RNA secondary structures with all other well-known real

RNA secondary structures in such a way that we could infer structure similarities along with potential information about the function of the unknown RNA secondary sructures.

# References


[1] Aguilar-Bonavides, Clemente, Gerardo A. Cardenas, E. S. Nakayasu, F. G. Lopes, Igor C. Almeida, Ming-Ying Leung, “Automatic Annotation of GPI Structures Using Grid Computing”. 5th International Conference on Bioinformatics and Computational Biology (BICoB).

[2] Andrade, Jorge, "Grid and High-Performance Computing for Applied Bioinformatics", Royal Institute of Technology, 2007.

[3] Arnold, P. “The importance of nucleic acid”. 10 Feb. 2010. http://www.brighthub.com/science/genetics/articles/63611.aspx. Accessed 13 Sep. 2016.

[4] Braga-Henebry, Patrick, "BioCompute: Harnessing Distributed Systems for Bioinformatics Applications", May 2009.

[5] Carbajal, P. “When a lot of little things add up to a lot of time; HTCondor to the rescue”. https://aciref.org/when-a-lot-of-little-things-add-up-to-a-lot-of-time-htcondor-to-the-rescue/. Accessed 7 September 2016.

[6] Dirks, Robert M., Justin M. Bois, Joseph M. Shaeffer, Erik Winfree, Niles A. Pierce, "Thermodynamic Analysis of Interacting Nucleic Acid Strand". SIAM Review. Society for Industrial and Applied Mathematics, 2007, p.65-88.

[7] Dirks, R. M. and Pierce, N. A. (2014). An algorithm for computing nucleic acid base=pairing probabilities including pseudoknots. Wiley Periodicals Inc. doi:10.1002/jcc.20057

[8] Domingo, Esteban (Dec 2008) RNA Virus Genomes. In: eLS. John Wiley & Sons Ltd, Chichester. http://www.els.net [doi: 10.1002/9780470015902.a0001488.pub2]

[9] Doublas Thain, Tood Tannenbaum, and Miron Livny. “Condor and the Grid”, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. ISBN: 0-470-85319-0.

[10] Field, A. P. (2013). *Discovering statistics using SPSS: and sex and drugs and rock ‘n’ roll* (4th Edition). London: Sage.

[11] “Francis Crick (1916-2004)”. The DNA molecule is shaped like a twisted ladder. DNA Learning Center, Cold Spring Harbor Laboratory. 2011. 8 Oct. 2016.

[12] Gardner, Paul P., Jennifer Daub., John Tate, Benjamin L. Moore, Isabelle H. Osuch, Sam Griffiths-Jones, Robert D. Finn, Eric P. Nawrocki, Diana L. Kolbe, Sean R. Eddy, Alex Bateman., “Rfam: Wikipedia, clans and the decimal release”. Nucleic Acids Research. Vol. 39. D141-D145.

[13] Griffiths-Jones, Sam et al. “Rfam: An RNA Family Database.” *Nucleic Acids Research* 31.1 (2003): 439–441. Print.

[14] HTCondor High Throughput Computing. “What is HTCondor?”. 26 Aug. 2016. http://research.cs.wisc.edu/htcondor/description.html. Accessed 6 Sep. 2016.

[15] HTCondor High Throughput Computing. “What’s new with HTCondor”. 24 Aug. 2016. http://research.cs.wisc.edu/htcondor/new.html. Accessed 6 Sep. 2016.

[16] Condor High Throughput Computing. “HTCondor Version 8.5.7 Manual”. n.d. http://research.cs.wisc.edu/htcondor/manual/v8.5/index.html. Accessed 2 Nov. 2016.

[17] Jabbari, Hosna, and Anne Condon. “A Fast and Robust Iterative Algorithm for Prediction of RNA Pseudoknotted Secondary Structures.” *BMC Bioinformatics* 15 (2014): 147. *PMC*. Web. 3 Nov. 2016.

[18] Ji, Yongmei, Xu, Xing, and Stormo, Gary D. "A graph theoretical approach to predict common RNA secondary structure motifs including pseudoknots in unaligned sequences", Oxford University Press, February 2004.

[19] Karp, Alan H., Horace P. Flatt. "Measuring Parallel Processor Performance", Communications of the ACM, 1990. Vol. 33. p.539-543.

[20] Licon, A., M. Taufer, M.-Y. Leung, and K. Johnson, “A dynamic programming algorithm for finding the optimal segmentation of an RNA secondary structure prediction,” in Proc. of the International Conference on Bioinformatics and Computational Biology, March 2010.

[21] Lodish H, Berk A, Zipursky SL, et al. Molecular Cell Biology. 4th edition. New York: W. H. Freeman; 2000. Section 4.4, The Three Roles of RNA in Protein Synthesis. Available from: http://www.ncbi.nlm.nih.gov/books/NBK21603/.

[22] Lorenz, Ronny et al. "ViennaRNA Package 2.0." *Algorithms for Molecular Biology : AMB* 6 (2011): 26. *PMC*. Web. 9 Nov. 2016.

[23] Mathews, David H. "RNA Secondary Structure Analysis Using RNAstructure." *Current protocols in bioinformatics / editorial board*, Andreas D. Baxevanis … [et al.] 0 12 (2006): Unit-12.6. PMC. Web. 13 Sept. 2016

[24] Mercer TR, Mattick JS. Structure and function of long noncoding RNAs in epigenetic regulation. Nature Structural & Molecular Biology 20, 300-307 (2013)

[25] Mubnday, B.L. and T. Nakai. "Special topic review: Nodaviruses as pathogens in larval and juvenile marine finfish." World Journal of Microbiology & Biotechnology, 1997. Vol 13. p. 375-381.

[26] Net Industries. "RNA Function". 2016. http://science.jrank.org/pages/5892/RNA-Function.html. Accessed 2 Oct. 2016.

[27] Nishizawa, T., M. Furuhasi, T. Nagai, T. Nakai, K. Muroga. "Genomic Classification of Fish Nodaviruses by Molecular Phylogenetic Analysis of the Coat Protein Gene." Appl. Environ. Microbiol, 1997. p. 1633-1636.

[28] "Nodavirus." *Aquablue Sea*foods, http://www.aquablueseafoods.com.au/nodavirus.shtml. Accessed 9 August 2016.

[29] Reeder, J. and Giegerich, R. "Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics." BMC Bioinformatics, 2004. p. 1-12.

[30] Reeder, Jens, Peter Steffen and Robert Giegerich. "pknotsRG: RNA pseudoknot folding including near-optimal structures and sliding windows" Nucleic Acids Res., 35, W320-W324.

[31] Ren, Jihong. et al. “HotKnots: Heuristic Prediction of RNA Secondary Structures Including Pseudoknots.” *RNA* 11.10 (2005): 1494–1504. *PMC*. Web. 9 Oct. 2016.

[32] Rivas E., Eddy S.R., "A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots", Academic Press, 1999.

[33] Sato, Kengo et al. “IPknot: Fast and Accurate Prediction of RNA Secondary Structures with Pseudoknots Using Integer Programming.” *Bioinformatics* 27.13 (2011): i85–i93. *PMC*. Web. 1 Nov. 2016.

[34] Seetin, Matthew G., David H. Mathews, "RNA Structure Prediction: An Overview of Methods", Methods in Molecular Biology, Volume 905, 2012, p. 99-122.

[35] Shetty, Mahesh et al. “Betanodavirus of Marine and Freshwater Fish: Distribution, Genomic Organization, Diagnosis and Control Measures.” *Indian journal of virology : an official organ of Indian Virological Society* 23.2 (2012): 114–123. *PMC*. Web. 17 Aug. 2016.

[36] Tannenbaum, Todd. “What’s new in HTCondor? What’s coming?”. Center for High Throughput Computing, University of Wisconsin-Madison. 18 May 2016. Web. 11 Sep 2016.

[37] Taufer, Michela., Abel Licon, Roberto Araiza, David Mireles, F. H. D. van Batenburg, Alexander P. Gultyaev, Ming-Ying Leung, “PseudoBase++: an extension of PseudoBase for easy searching, formatting and visualization of pseudoknots”, Nucleic Acids Research, 2009. D127-D135.

[38] Taylor, Spencer. “High Performance Computing of Hydrologic Models Using HTCondor”. Deparment of Civil and Environmental Engineering, Brigham Young University. 2013.

[39] Thain, Douglas, Todd Tannenbaum, Miron Livny. “Distributed Computing in Practice: The Condor Experience”, Vol. 17, No. 2-4, 323-356.

[40] van Batenburg, F. H. D., A. P. Gultyaev, C. W. A. Pleij, J. Ng, J. Oliehoek. “PseudoBase: a database with RNA pseudoknots”. Nucleic Acids Research, 2000. p. 201–204.

[41] Watson, James D., Tania A. Baker, Stephen P. Bell, Alexander Gann, Michael Levine, Richard Losick, "Molecular Biology of the Gene", Pearson Education, 2003, p.1-33.

[42] “What is the one-sample t-test?”. Conduct and interpret a one-sample t-test. London: Sage. Statistics Solutions. 2016. 5 Oct. 2016.

[43] “What is RNA?”. The RNA Society. n.p., n.d. 8 Oct. 2016.

[44] Y. Byun and K. Han, PseudoViewer: web application and web service for visualizing RNA pseudoknots and secondary structures, Nucleic Acids Research, Vol.34, W416-W422, 2006.

[45] Yehdego, D. T., Zhang, B., Kodimala, V. K., Vegesna, R., Johnson, K. L., Taufer, M., Leung, M.-Y., (to appear 2013). Secondary structure predictions for long RNA sequences based on inversion excursions and MapReduce. Proceedings of the 12th IEEE International Workshop on High Performance Computational Biology (HiCOMB), Boston, May 20 2013.

[46] Yehdego, D. T., Kodimala, V. K., Viswakula, S., Zhang, B., Vegesna, R., Johnson, K.L., Taufer, M., Leung, M.-Y. (2012). Secondary Structure Predictions for Long RNA Sequences. Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine. pp. 545-547. New York, NY.

[47] Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M. and Pierce, N. A. (2011). NUPACK: Analysis and design of nucleic acid systems. J. Comput. Chem., 32: 170–173. doi:10.1002/jcc.21596

[48] Zhang, B., Yehdego, D. T., Johnson, K. L., Leung, M.-Y., Taufer, M., (2012). A Modularized MapReduce Framework to Support RNA Secondary Structure. (pp. 1-8). IEEE: 2012 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).

[49] Zhang, B., Yehdego, D. T., Johnson, K. L., Leung, M.-Y., and Taufer, M., (to appear 2013) Enhancement of Accuracy and Efficiency for RNA Secondary Structure Prediction by Sequence Segmentation and MapReduce, BMC Structural Biology.

[50] Zuker M., Stiegler P., "Optimal Computer folding of large RNA sequences using thermodynamics and auxiliary information", Nucleic Acids Research 9, 1981, p.133-148.

# Glossary

| TERM | DEFINITION |
|---|---|
| **ACCURACY** | It defines how close a measured value is with respect to the known value. |
| **CHUNK** | A segment of a long RNA sequence obtained from either chunk concatenation or two-chunk elimination method. |
| **REGULAR METHOD** | RNA secondary structure prediction method that uses the whole sequence as an input in the prediction of RNA secondary structures. |
| **HTCONDOR** | High-throughput computing tool that allows to execute jobs on idle machines and therefore minimizes the waiting time for the final output. |
| **NFE** | Normalized Free Energy obtained by dividing the minimum free energy of a given sequences by its length. |
| **NODAVIRUS** | A microscopic group of single stranded RNA viruses from the Nodaviridae family, which causes a disease called VNN (Viral Nervous Necrosis) that affects newborn baby fish. |
| **CHUNK CONCATENATION METHOD** | Method that allows to cut long RNA sequences into chunks of size (S) and gap size (G). |
| **MFE** | Minimum Free Energy value that allows to establish the stabilization of a RNA structure based on Thermodynamics. |
| **PKNOTSRG** | RNA secondary structure prediction algorithm including pseudoknots that is based on the Turner energy rules to find the structure with MFE. |
| **RNA** | A single-stranded macromolecules composed of four nucleic acids: C (cytosine), A (adenine), G (guanine), and U (Uracil). |
| **RNASTRUCTURE** | RNA secondary structure prediction algorithm including pseudoknots that is based on the Thermodynamics to find the structure with MFE. |
| **RFAM** | A group of non-coding RNA families based on multiple sequence |

| | |
|---|---|
| | alignments and covariance models. |
| **WAITING TIME** | The total time elapsed for the prediction of RNA secondary sequences dataset. |

# Appendix A. Condor Config File

```
##########################################################################
##
##  condor_config
##
##  This is the global configuration file for condor.  Any settings
##  made here may potentially be overridden in the local configuration
##  file.  KEEP THAT IN MIND!  To double-check that a variable is
##  getting set from the configuration file that you expect, use
##  condor_config_val -v <variable name>
##
##  The file is divided into four main parts:
##  Part 1:  Settings you likely want to customize
##  Part 2:  Settings you may want to customize
##  Part 3:  Settings that control the policy of when condor will
##           start and stop jobs on your machines
##  Part 4:  Settings you should probably leave alone (unless you
##  know what you're doing)
##
##  Please read the INSTALL file (or the Install chapter in the
##  Condor Administrator's Manual) for detailed explanations of the
##  various settings in here and possible ways to configure your
##  pool.
##
##  Unless otherwise specified, settings that are commented out show
##  the defaults that are used if you don't define a value.  Settings
##  that are defined here MUST BE DEFINED since they have no default
##  value.
##
##  Unless otherwise indicated, all settings which specify a time are
##  defined in seconds.
##
##########################################################################

##########################################################################
##########################################################################
##
## ######                    #
## #   #  ##  #####  #####      ##
## #   # # # #  #   #      ##
## ###### #  # #  #   #       #
## #    ###### #####   #       #
## #    #  # # #   #       #
## #    #  # #  #   #      #####
```

```
##
##  Part 1:  Settings you likely want to customize:
##########################################################################
##########################################################################

##  What machine is your central manager?
CONDOR_HOST       = server1.utep.edu

##--------------------------------------------------------------------
##  Pathnames:
##--------------------------------------------------------------------
##  Where have you installed the bin, sbin and lib condor directories?
RELEASE_DIR                 = /usr/local/condor-8.4.6

##  Where is the local condor directory for each host?
##  This is where the local config file(s), logs and
##  spool/execute directories are located
LOCAL_DIR             = $(RELEASE_DIR)/local.PC1

##  Where is the machine-specific local config file for each host?
LOCAL_CONFIG_FILE       = $(LOCAL_DIR)/condor_config.local

##  Where are optional machine-specific local config files located?
##  Config files are included in lexicographic order.
LOCAL_CONFIG_DIR        = $(LOCAL_DIR)/config

## Blacklist for file processing in the LOCAL_CONFIG_DIR
## LOCAL_CONFIG_DIR_EXCLUDE_REGEXP =
^((\..*)|(.*~)|(#.*)|(.*\.rpmsave)|(.*\.rpmnew))$

## If the local config file is not present, is it an error?
## WARNING: This is a potential security issue.
## If not specified, the default is True
#REQUIRE_LOCAL_CONFIG_FILE = TRUE

##--------------------------------------------------------------------
##  Mail parameters:
##--------------------------------------------------------------------
##  When something goes wrong with condor at your site, who should get
##  the email?
#CONDOR_ADMIN           = root@$(FULL_HOSTNAME)

##  Full path to a mail delivery program that understands that "-s"
##  means you want to specify a subject:
#MAIL                        = /bin/mail
```

```
##--------------------------------------------------------------------
##  Network domain parameters:
##--------------------------------------------------------------------
##  Internet domain of machines sharing a common UID space.  If your
##  machines don't share a common UID space, set it to
##  UID_DOMAIN = $(FULL_HOSTNAME)
##  to specify that each machine has its own UID space.
UID_DOMAIN                  = utep.edu

##  Internet domain of machines sharing a common file system.
##  If your machines don't use a network file system, set it to
##  FILESYSTEM_DOMAIN = $(FULL_HOSTNAME)
##  to specify that each machine has its own file system.
FILESYSTEM_DOMAIN     = utep.edu

##  This macro is used to specify a short description of your pool.
##  It should be about 20 characters long. For example, the name of
##  the UW-Madison Computer Science Condor Pool is ``UW-Madison CS".
COLLECTOR_NAME              = Bioinformatics HTCondor Pool

##  Part 2:  Settings you may want to customize:
##  (it is generally safe to leave these untouched)

##  The user/group ID <uid>.<gid> of the "Condor" user.
##  (this can also be specified in the environment)
##  Note: the CONDOR_IDS setting is ignored on Win32 platforms
#CONDOR_IDS=x.x
CONDOR_IDS=1027.2002

##--------------------------------------------------------------------
##  Flocking: Submitting jobs to more than one pool
##--------------------------------------------------------------------
```

```
##  Flocking allows you to run your jobs in other pools, or lets
##  others run jobs in your pool.
##
##  To let others flock to you, define FLOCK_FROM.
##
##  To flock to others, define FLOCK_TO.

##  FLOCK_FROM defines the machines where you would like to grant
##  people access to your pool via flocking. (i.e. you are granting
##  access to these machines to join your pool).
#FLOCK_FROM =
##  An example of this is:
#FLOCK_FROM = somehost.friendly.domain, anotherhost.friendly.domain

##  FLOCK_TO defines the central managers of the pools that you want
##  to flock to. (i.e. you are specifying the machines that you
##  want your jobs to be negotiated at -- thereby specifying the
##  pools they will run in.)
#FLOCK_TO =
##  An example of this is:
#FLOCK_TO = central_manager.friendly.domain, condor.cs.wisc.edu

##  FLOCK_COLLECTOR_HOSTS should almost always be the same as
##  FLOCK_NEGOTIATOR_HOSTS (as shown below).  The only reason it would be
##  different is if the collector and negotiator in the pool that you are
##  flocking too are running on different machines (not recommended).
##  The collectors must be specified in the same corresponding order as
##  the FLOCK_NEGOTIATOR_HOSTS list.
FLOCK_NEGOTIATOR_HOSTS = $(FLOCK_TO)
FLOCK_COLLECTOR_HOSTS = $(FLOCK_TO)
## An example of having the negotiator and the collector on different
## machines is:
#FLOCK_NEGOTIATOR_HOSTS = condor.cs.wisc.edu, condor-negotiator.friendly.domain
#FLOCK_COLLECTOR_HOSTS =  condor.cs.wisc.edu, condor-collector.friendly.domain

##--------------------------------------------------------------------
##  Host/IP access levels
##--------------------------------------------------------------------
##  Please see the administrator's manual for details on these
##  settings, what they're for, and how to use them.

##  What machines have administrative rights for your pool?  This
##  defaults to your central manager.  You should set it to the
##  machine(s) where whoever is the condor administrator(s) works
##  (assuming you trust all the users who log into that/those
##  machine(s), since this is machine-wide access you're granting).
```

```
ALLOW_ADMINISTRATOR = $(CONDOR_HOST), $(IP_ADDRESS)

##  If there are no machines that should have administrative access
##  to your pool (for example, there's no machine where only trusted
##  users have accounts), you can uncomment this setting.
##  Unfortunately, this will mean that administering your pool will
##  be more difficult.
#DENY_ADMINISTRATOR = *

##  What machines should have "owner" access to your machines, meaning
##  they can issue commands that a machine owner should be able to
##  issue to their own machine (like condor_vacate).  This defaults to
##  machines with administrator access, and the local machine.  This
##  is probably what you want.
ALLOW_OWNER = $(FULL_HOSTNAME), $(ALLOW_ADMINISTRATOR)

##  Read access.  Machines listed as allow (and/or not listed as deny)
##  can view the status of your pool, but cannot join your pool
##  or run jobs.
##  NOTE: By default, without these entries customized, you
##  are granting read access to the whole world.  You may want to
##  restrict that to hosts in your domain.  If possible, please also
##  grant read access to "*.cs.wisc.edu", so the Condor developers
##  will be able to view the status of your pool and more easily help
##  you install, configure or debug your Condor installation.
##  It is important to have this defined.
ALLOW_READ = *
#ALLOW_READ = *.your.domain, *.cs.wisc.edu
#DENY_READ = *.bad.subnet, bad-machine.your.domain, 144.77.88.*

##  Write access.  Machines listed here can join your pool, submit
##  jobs, etc.  Note: Any machine which has WRITE access must
##  also be granted READ access.  Granting WRITE access below does
##  not also automatically grant READ access; you must change
##  ALLOW_READ above as well.
##
##  You must set this to something else before Condor will run.
##  This most simple option is:
##    ALLOW_WRITE = *
##  but note that this will allow anyone to submit jobs or add
##  machines to your pool and is a serious security risk.

ALLOW_WRITE = *
#ALLOW_WRITE = *.your.domain, your-friend's-machine.other.domain
#DENY_WRITE = bad-machine.your.domain
```

```
##  Are you upgrading to a new version of Condor and confused about
##  why the above ALLOW_WRITE setting is causing Condor to refuse to
##  start up?  If you are upgrading from a configuration that uses
##  HOSTALLOW/HOSTDENY instead of ALLOW/DENY we recommend that you
##  convert all uses of the former to the latter.  The syntax of the
##  authorization settings is identical.  They both support
##  unauthenticated IP-based authorization as well as authenticated
##  user-based authorization.  To avoid confusion, the use of
##  HOSTALLOW/HOSTDENY is discouraged.  Support for it may be removed
##  in the future.

##  Negotiator access.  Machines listed here are trusted central
##  managers.  You should normally not have to change this.
ALLOW_NEGOTIATOR = $(CONDOR_HOST), $(IP_ADDRESS)
##  Now, with flocking we need to let the SCHEDD trust the other
##  negotiators we are flocking with as well.  You should normally
##  not have to change this either.
ALLOW_NEGOTIATOR_SCHEDD = $(CONDOR_HOST),
$(FLOCK_NEGOTIATOR_HOSTS), $(IP_ADDRESS)

##  Config access.  Machines listed here can use the condor_config_val
##  tool to modify all daemon configurations.  This level of host-wide
##  access should only be granted with extreme caution.  By default,
##  config access is denied from all hosts.
#ALLOW_CONFIG = trusted-host.your.domain

##  Flocking Configs.  These are the real things that Condor looks at,
##  but we set them from the FLOCK_FROM/TO macros above.  It is safe
##  to leave these unchanged.
ALLOW_WRITE_COLLECTOR = $(ALLOW_WRITE), $(FLOCK_FROM)
ALLOW_WRITE_STARTD    = $(ALLOW_WRITE), $(FLOCK_FROM)
ALLOW_READ_COLLECTOR  = $(ALLOW_READ), $(FLOCK_FROM)
ALLOW_READ_STARTD     = $(ALLOW_READ), $(FLOCK_FROM)


##--------------------------------------------------------------------
##  Security parameters for setting configuration values remotely:
##--------------------------------------------------------------------
##  These parameters define the list of attributes that can be set
##  remotely with condor_config_val for the security access levels
##  defined above (for example, WRITE, ADMINISTRATOR, CONFIG, etc).
##  Please see the administrator's manual for further details on these
##  settings, what they're for, and how to use them.  There are no
##  default values for any of these settings.  If they are not
##  defined, no attributes can be set with condor_config_val.
```

```
## Do you want to allow condor_config_val -rset to work at all?
## This feature is disabled by default, so to enable, you must
## uncomment the following setting and change the value to "True".
## Note: changing this requires a restart not just a reconfig.
#ENABLE_RUNTIME_CONFIG = False

## Do you want to allow condor_config_val -set to work at all?
## This feature is disabled by default, so to enable, you must
## uncomment the following setting and change the value to "True".
## Note: changing this requires a restart not just a reconfig.
#ENABLE_PERSISTENT_CONFIG = False

## Directory where daemons should write persistent config files (used
## to support condor_config_val -set).  This directory should *ONLY*
## be writable by root (or the user the Condor daemons are running as
## if non-root).  There is no default, administrators must define this.
## Note: changing this requires a restart not just a reconfig.
#PERSISTENT_CONFIG_DIR = /full/path/to/root-only/local/directory

##  Attributes that can be set by hosts with "CONFIG" permission (as
##  defined with ALLOW_CONFIG and DENY_CONFIG above).
##  The commented-out value here was the default behavior of Condor
##  prior to version 6.3.3.  If you don't need this behavior, you
##  should leave this commented out.
#SETTABLE_ATTRS_CONFIG = *

##  Attributes that can be set by hosts with "ADMINISTRATOR"
##  permission (as defined above)
#SETTABLE_ATTRS_ADMINISTRATOR = *_DEBUG, MAX_*_LOG

##  Attributes that can be set by hosts with "OWNER" permission (as
##  defined above) NOTE: any Condor job running on a given host will
##  have OWNER permission on that host by default.  If you grant this
##  kind of access, Condor jobs will be able to modify any attributes
##  you list below on the machine where they are running.  This has
##  obvious security implications, so only grant this kind of
##  permission for custom attributes that you define for your own use
##  at your pool (custom attributes about your machines that are
##  published with the STARTD_ATTRS setting, for example).
#SETTABLE_ATTRS_OWNER = your_custom_attribute, another_custom_attr

##  You can also define daemon-specific versions of each of these
##  settings.  For example, to define settings that can only be
##  changed in the condor_startd's configuration by hosts with OWNER
##  permission, you would use:
#STARTD_SETTABLE_ATTRS_OWNER = your_custom_attribute_name
```

```
##--------------------------------------------------------------------
##  Network filesystem parameters:
##--------------------------------------------------------------------
##  Do you want to use NFS for file access instead of remote system
##  calls?
#USE_NFS              = False

##  Do you want to use AFS for file access instead of remote system
##  calls?
#USE_AFS              = False

##--------------------------------------------------------------------
##  Checkpoint server:
##--------------------------------------------------------------------
##  Do you want to use a checkpoint server if one is available?  If a
##  checkpoint server is not available or USE_CKPT_SERVER is set to
##  False, checkpoints will be written to the local SPOOL directory on
##  the submission machine.
#USE_CKPT_SERVER        = True

##  What's the hostname of this machine's nearest checkpoint server?
#CKPT_SERVER_HOST      = checkpoint-server-hostname.your.domain

##  Do you want the starter on the execute machine to choose the
##  checkpoint server?  If False, the CKPT_SERVER_HOST set on
##  the submit machine is used.  Otherwise, the CKPT_SERVER_HOST set
##  on the execute machine is used.  The default is true.
#STARTER_CHOOSES_CKPT_SERVER = True

##--------------------------------------------------------------------
##  Miscellaneous:
##--------------------------------------------------------------------
##  Try to save this much swap space by not starting new shadows.
##  Specified in megabytes.
#RESERVED_SWAP          = 0

##  What's the maximum number of jobs you want a single submit machine
##  to spawn shadows for?  The default is a function of $(DETECTED_MEMORY)
##  and a guess at the number of ephemeral ports available.

## Example 1:
#MAX_JOBS_RUNNING     = 10000

## Example 2:
```

```
## This is more complicated, but it produces the same limit as the default.
## First define some expressions to use in our calculation.
## Assume we can use up to 80% of memory and estimate shadow private data
## size of 800k.
#MAX_SHADOWS_MEM   = ceiling($(DETECTED_MEMORY)*0.8*1024/800)
## Assume we can use ~21,000 ephemeral ports (avg ~2.1 per shadow).
## Under Linux, the range is set in /proc/sys/net/ipv4/ip_local_port_range.
#MAX_SHADOWS_PORTS = 10000
## Under windows, things are much less scalable, currently.
## Note that this can probably be safely increased a bit under 64-bit windows.
#MAX_SHADOWS_OPSYS = ifThenElse(regexp("WIN.*","$(OPSYS)"),200,100000)
## Now build up the expression for MAX_JOBS_RUNNING.  This is complicated
## due to lack of a min() function.
#MAX_JOBS_RUNNING    = $(MAX_SHADOWS_MEM)
#MAX_JOBS_RUNNING    = \
#  ifThenElse( $(MAX_SHADOWS_PORTS) < $(MAX_JOBS_RUNNING), \
#            $(MAX_SHADOWS_PORTS), \
#            $(MAX_JOBS_RUNNING) )
#MAX_JOBS_RUNNING    = \
#  ifThenElse( $(MAX_SHADOWS_OPSYS) < $(MAX_JOBS_RUNNING), \
#            $(MAX_SHADOWS_OPSYS), \
#            $(MAX_JOBS_RUNNING) )


##  Maximum number of simultaneous downloads of output files from
##  execute machines to the submit machine (limit applied per schedd).
##  The value 0 means unlimited.
#MAX_CONCURRENT_DOWNLOADS = 10

##  Maximum number of simultaneous uploads of input files from the
##  submit machine to execute machines (limit applied per schedd).
##  The value 0 means unlimited.
#MAX_CONCURRENT_UPLOADS = 10

##  Condor needs to create a few lock files to synchronize access to
##  various log files.  Because of problems we had with network
##  filesystems and file locking over the years, we HIGHLY recommend
##  that you put these lock files on a local partition on each
##  machine.  If you don't have your LOCAL_DIR on a local partition,
##  be sure to change this entry.  Whatever user (or group) condor is
##  running as needs to have write access to this directory.  If
##  you're not running as root, this is whatever user you started up
##  the condor_master as.  If you are running as root, and there is a
##  condor account, it is probably condor.  Otherwise, it is whatever
##  you've set in the CONDOR_IDS environment variable.  See the Admin
##  manual for details on this.
```

```
LOCK            = $(LOG)

##  If you don't use a fully qualified name in your /etc/hosts file
##  (or NIS, etc.) for either your official hostname or as an alias,
##  Condor would not normally be able to use fully qualified names in
##  places that it'd like to.  You can set this parameter to the
##  domain you'd like appended to your hostname, if changing your host
##  information is not a good option.  This parameter must be set in
##  the global config file (not the LOCAL_CONFIG_FILE from above).
#DEFAULT_DOMAIN_NAME = your.domain.name

##  If you don't have DNS set up, Condor will normally fail in many
##  places because it can't resolve hostnames to IP addresses and
##  vice-versa. If you enable this option, Condor will use
##  pseudo-hostnames constructed from a machine's IP address and the
##  DEFAULT_DOMAIN_NAME. Both NO_DNS and DEFAULT_DOMAIN must be set in
##  your top-level config file for this mode of operation to work
##  properly.
#NO_DNS = True

##  Condor can be told whether or not you want the Condor daemons to
##  create a core file if something really bad happens.  This just
##  sets the resource limit for the size of a core file.  By default,
##  we don't do anything, and leave in place whatever limit was in
##  effect when you started the Condor daemons.  If this parameter is
##  set and "True", we increase the limit to as large as it gets.  If
##  it is set to "False", we set the limit at 0 (which means that no
##  core files are even created).  Core files greatly help the Condor
##  developers debug any problems you might be having.
#CREATE_CORE_FILES     = True

##  When Condor daemons detect a fatal internal exception, they
##  normally log an error message and exit.  If you have turned on
##  CREATE_CORE_FILES, in some cases you may also want to turn on
##  ABORT_ON_EXCEPTION so that core files are generated when an
##  exception occurs.  Set the following to True if that is what you
##  want.
#ABORT_ON_EXCEPTION = False

##  If your site needs to use UID_DOMAIN settings (defined above) that
##  are not real Internet domains that match the hostnames, you can
##  tell Condor to trust whatever UID_DOMAIN a submit machine gives to
##  the execute machine and just make sure the two strings match.  The
##  default for this setting is False, since it is more secure this
##  way.
#TRUST_UID_DOMAIN = False
```

```
## If you would like to be informed in near real-time via condor_q when
## a vanilla/standard/java job is in a suspension state, set this attribute to
## TRUE. However, this real-time update of the condor_schedd by the shadows
## could cause performance issues if there are thousands of concurrently
## running vanilla/standard/java jobs under a single condor_schedd and they
## are allowed to suspend and resume.
#REAL_TIME_JOB_SUSPEND_UPDATES = False

## A standard universe job can perform arbitrary shell calls via the
## libc 'system()' function. This function call is routed back to the shadow
## which performs the actual system() invocation in the initial directory of the
## running program and as the user who submitted the job. However, since the
## user job can request ARBITRARY shell commands to be run by the shadow, this
## is a generally unsafe practice. This should only be made available if it is
## actually needed. If this attribute is not defined, then it is the same as
## it being defined to False. Set it to True to allow the shadow to execute
## arbitrary shell code from the user job.
#SHADOW_ALLOW_UNSAFE_REMOTE_EXEC = False

## KEEP_OUTPUT_SANDBOX is an optional feature to tell Condor-G to not
## remove the job spool when the job leaves the queue.  To use, just
## set to TRUE.  Since you will be operating Condor-G in this manner,
## you may want to put leave_in_queue = false in your job submit
## description files, to tell Condor-G to simply remove the job from
## the queue immediately when the job completes (since the output files
## will stick around no matter what).
#KEEP_OUTPUT_SANDBOX = False

## This setting tells the negotiator to ignore user priorities.  This
## avoids problems where jobs from different users won't run when using
## condor_advertise instead of a full-blown startd (some of the user
## priority system in Condor relies on information from the startd --
## we will remove this reliance when we support the user priority
## system for grid sites in the negotiator; for now, this setting will
## just disable it).
#NEGOTIATOR_IGNORE_USER_PRIORITIES = False

## This is a list of libraries containing ClassAd plug-in functions.
#CLASSAD_USER_LIBS =

## This setting tells Condor whether to delegate or copy GSI X509
## credentials when sending them over the wire between daemons.
## Delegation can take up to a second, which is very slow when
## submitting a large number of jobs. Copying exposes the credential
## to third parties if Condor is not set to encrypt communications.
```

```
## By default, Condor will delegate rather than copy.
#DELEGATE_JOB_GSI_CREDENTIALS = True

## This setting controls whether Condor delegates a full or limited
## X509 credential for jobs. Currently, this only affects grid-type
## gt2 grid universe jobs. The default is False.
#DELEGATE_FULL_JOB_GSI_CREDENTIALS = False

## This setting controls the default behavior for the spooling of files
## into, or out of, the Condor system by such tools as condor_submit
## and condor_transfer_data. Here is the list of valid settings for this
## parameter and what they mean:
##
##   stm_use_schedd_only
##      Ask the condor_schedd to solely store/retrieve the sandbox
##
##   stm_use_transferd
##      Ask the condor_schedd for a location of a condor_transferd, then
##      store/retrieve the sandbox from the transferd itself.
##
## The allowed values are case insensitive.
## The default of this parameter if not specified is: stm_use_schedd_only
#SANDBOX_TRANSFER_METHOD = stm_use_schedd_only

## This setting specifies an IP address that depends on the setting of
## BIND_ALL_INTERFACES. If BIND_ALL_INTERFACES  is True (the default), then
## this variable controls what IP address will be advertised as the public
## address of the daemon. If BIND_ALL_INTERFACES is False, then this variable
## specifies which IP address to bind network sockets to. If
## BIND_ALL_INTERFACES is False  and NETWORK_INTERFACE is not defined,
Condor
## chooses a network interface automatically. It tries to choose a public
## interface if one is available. If it cannot decide which of two interfaces
## to choose from, it will pick the first one.
#NETWORK_INTERFACE =

##--------------------------------------------------------------------
##  Settings that control the daemon's debugging output:
##--------------------------------------------------------------------

##
## The flags given in ALL_DEBUG are shared between all daemons.
##

#ALL_DEBUG              =
#MAX_DEFAULT_LOG                = 10 Mb
```

```
#MAX_COLLECTOR_LOG = $(MAX_DEFAULT_LOG)
#COLLECTOR_DEBUG              =

#MAX_KBDD_LOG          = $(MAX_DEFAULT_LOG)
#KBDD_DEBUG            =

#MAX_NEGOTIATOR_LOG        = $(MAX_DEFAULT_LOG)
#NEGOTIATOR_DEBUG    = D_MATCH
#MAX_NEGOTIATOR_MATCH_LOG = $(MAX_DEFAULT_LOG)

#MAX_SCHEDD_LOG            = $(MAX_DEFAULT_LOG)
#SCHEDD_DEBUG          = D_PID

#MAX_SHADOW_LOG            = $(MAX_DEFAULT_LOG)
#SHADOW_DEBUG          =

#MAX_STARTD_LOG            = $(MAX_DEFAULT_LOG)
#STARTD_DEBUG          =

#MAX_STARTER_LOG           = $(MAX_DEFAULT_LOG)

#MAX_MASTER_LOG            = $(MAX_DEFAULT_LOG)
#MASTER_DEBUG          =
##  Truncates master log start up?
#TRUNC_MASTER_LOG_ON_OPEN       = False

#MAX_JOB_ROUTER_LOG     = $(MAX_DEFAULT_LOG)
#JOB_ROUTER_DEBUG       =

#MAX_ROOSTER_LOG        = $(MAX_DEFAULT_LOG)
#ROOSTER_DEBUG          =

#MAX_SHARED_PORT_LOG    = $(MAX_DEFAULT_LOG)
#SHARED_PORT_DEBUG      =

#MAX_HDFS_LOG           = $(MAX_DEFAULT_LOG)
#HDFS_DEBUG             =

# High Availability Logs
#MAX_HAD_LOG            = $(MAX_DEFAULT_LOG)
#HAD_DEBUG              =
#MAX_REPLICATION_LOG        = $(MAX_DEFAULT_LOG)
#REPLICATION_DEBUG   =
#MAX_TRANSFERER_LOG         = $(MAX_DEFAULT_LOG)
#TRANSFERER_DEBUG    =
```

```
## The daemons touch their log file periodically, even when they have
## nothing to write. When a daemon starts up, it prints the last time
## the log file was modified. This lets you estimate when a previous
## instance of a daemon stopped running. This parameter controls how often
## the daemons touch the file (in seconds).
#TOUCH_LOG_INTERVAL = 60

##########################################################################
##########################################################################
##
## ######                    #####
## #   #  ##  #####  #####     #   #
## #   # # # #   #   #         #
## ######  #  # #   #   #       #####
## #     ###### #####   #         #
## #     #  # # #   #       #   #
## #     #  # #  #   #       #####
##
##  Part 3:  Settings control the policy for running, stopping, and
##  periodically check-pointing condor jobs:
##########################################################################
##########################################################################

##  This section contains macros are here to help write legible
##  expressions:
MINUTE              = 60
HOUR                = (60 * $(MINUTE))
StateTimer      = (time() - EnteredCurrentState)
ActivityTimer = (time() - EnteredCurrentActivity)
ActivationTimer = ifThenElse(JobStart =!= UNDEFINED, (time() - JobStart), 0)
LastCkpt        = (time() - LastPeriodicCheckpoint)

##  The JobUniverse attribute is just an int.  These macros can be
##  used to specify the universe in a human-readable way:
STANDARD  = 1
VANILLA             = 5
MPI                 = 8
VM                  = 13
IsMPI           = (TARGET.JobUniverse == $(MPI))
IsVanilla       = (TARGET.JobUniverse == $(VANILLA))
IsStandard      = (TARGET.JobUniverse == $(STANDARD))
IsVM            = (TARGET.JobUniverse == $(VM))

NonCondorLoadAvg  = (LoadAvg - CondorLoadAvg)
```

```
BackgroundLoad              = 0.3
HighLoad                    = 0.5
StartIdleTime               = 15 * $(MINUTE)
ContinueIdleTime            =  5 * $(MINUTE)
MaxSuspendTime              = 10 * $(MINUTE)
MaxVacateTime               = 10 * $(MINUTE)

KeyboardBusy                = (KeyboardIdle < $(MINUTE))
ConsoleBusy                 = (ConsoleIdle  < $(MINUTE))
CPUIdle                     = ($(NonCondorLoadAvg) <= $(BackgroundLoad))
CPUBusy                     = ($(NonCondorLoadAvg) >= $(HighLoad))
KeyboardNotBusy             = ($(KeyboardBusy) == False)

BigJob          = (TARGET.ImageSize >= (50 * 1024))
MediumJob       = (TARGET.ImageSize >= (15 * 1024) && TARGET.ImageSize < (50 *
1024))
SmallJob        = (TARGET.ImageSize <  (15 * 1024))

JustCPU                     = ($(CPUBusy) && ($(KeyboardBusy) == False))
MachineBusy                 = ($(CPUBusy) || $(KeyboardBusy))

##  The RANK expression controls which jobs this machine prefers to
##  run over others.  Some examples from the manual include:
##    RANK = TARGET.ImageSize
##    RANK = (Owner == "coltrane") + (Owner == "tyner") \
##                + ((Owner == "garrison") * 10) + (Owner == "jones")
##  By default, RANK is always 0, meaning that all jobs have an equal
##  ranking.
#RANK                       = 0


##########################################################################
##  This where you choose the configuration that you would like to
##  use. If you dont specify, the default policy is a no-preemption
##  policy. This is what older config files called TESTINGMODE_*
##########################################################################

# When should we only consider SUSPEND instead of PREEMPT?
#WANT_SUSPEND = False

# When should we preempt gracefully instead of hard-killing?
#WANT_VACATE = False

##  When is this machine willing to start a job?
START = True
```

```
##  When to suspend a job?
SUSPEND = False

##  When to resume a suspended job?
CONTINUE = True

##  When to nicely stop a job?
##  (as opposed to killing it instantaneously)
PREEMPT = False

##  When to instantaneously kill a preempting job
##  (e.g. if a job is in the pre-empting stage for too long)
KILL = False

##  A non-zero value here is another way to have a no-preemption policy
##  a job is given this many seconds to exit even when it is preempted.
#MaxJobRetirementTime    = 0

#PERIODIC_CHECKPOINT = ((time() - LastPeriodicCheckpoint)/60.0) > (180.0 +
$RANDOM_INTEGER(-30,30,1))
#PREEMPTION_REQUIREMENTS = False
#PREEMPTION_RANK = $(UWCS_PREEMPTION_RANK)
#NEGOTIATOR_PRE_JOB_RANK = RemoteOwner =?= UNDEFINED
#NEGOTIATOR_POST_JOB_RANK = (RemoteOwner =?= UNDEFINED) *
(ifthenElse(isUndefined(KFlops), 1000, Kflops) - SlotID - 1.0e10*(Offline=?=True))
#CLAIM_WORKLIFE          = 1200

##  When should a local universe job be allowed to start?
#START_LOCAL_UNIVERSE = TotalLocalJobsRunning < 200

##  When should a scheduler universe job be allowed to start?
#START_SCHEDULER_UNIVERSE = TotalSchedulerJobsRunning < 200

##########################################################################
## This is the UWisc - CS Department Configuration.
##########################################################################

# When should we only consider SUSPEND instead of PREEMPT?
# Only when SUSPEND is True and one of the following is also true:
#   - the job is small
#   - the keyboard is idle
#   - it is a vanilla universe job
UWCS_WANT_SUSPEND  = ( $(SmallJob) || $(KeyboardNotBusy) || $(IsVanilla) ) && \
                ( $(SUSPEND) )

# When should we preempt gracefully instead of hard-killing?
```

```
UWCS_WANT_VACATE   = ( $(ActivationTimer) > 10 * $(MINUTE) || $(IsVanilla) )

# Only start jobs if:
# 1) the keyboard has been idle long enough, AND
# 2) the load average is low enough OR the machine is currently
#    running a Condor job
# (NOTE: Condor will only run 1 job at a time on a given resource.
# The reasons Condor might consider running a different job while
# already running one are machine Rank (defined above), and user
# priorities.)
UWCS_START         = ( (KeyboardIdle > $(StartIdleTime)) \
                 && ( $(CPUIdle) || \
                     (State != "Unclaimed" && State != "Owner")) )

# Suspend jobs if:
# 1) the keyboard has been touched, OR
# 2a) The CPU has been busy for more than 2 minutes, AND
# 2b) the job has been running for more than 90 seconds
UWCS_SUSPEND = ( $(KeyboardBusy) || \
               ( (CpuBusyTime > 2 * $(MINUTE)) \
                 && $(ActivationTimer) > 90 ) )

# Continue jobs if:
# 1) the cpu is idle, AND
# 2) we've been suspended more than 10 seconds, AND
# 3) the keyboard hasn't been touched in a while
UWCS_CONTINUE = ( $(CPUIdle) && ($(ActivityTimer) > 10) \
                && (KeyboardIdle > $(ContinueIdleTime)) )

# Preempt jobs if:
# 1) The job is suspended and has been suspended longer than we want
# 2) OR, we don't want to suspend this job, but the conditions to
#    suspend jobs have been met (someone is using the machine)
UWCS_PREEMPT = ( ((Activity == "Suspended") && \
               ($(ActivityTimer) > $(MaxSuspendTime))) \
                    || (SUSPEND && (WANT_SUSPEND == False)) )

# Maximum time (in seconds) to wait for a job to finish before kicking
# it off (due to PREEMPT, a higher priority claim, or the startd
# gracefully shutting down).  This is computed from the time the job
# was started, minus any suspension time.  Once the retirement time runs
# out, the usual preemption process will take place.  The job may
# self-limit the retirement time to _less_ than what is given here.
# By default, nice user jobs and standard universe jobs set their
# MaxJobRetirementTime to 0, so they will not wait in retirement.
```

```
UWCS_MaxJobRetirementTime = 0

##  If you completely disable preemption of claims to machines, you
##  should consider limiting the time span over which new jobs will be
##  accepted on the same claim.  See the manual section on disabling
##  preemption for a comprehensive discussion.  Since this example
##  configuration does not disable preemption of claims, we leave
##  CLAIM_WORKLIFE undefined (infinite).
#UWCS_CLAIM_WORKLIFE = 1200

# How long to allow a job to vacate gracefully.  After this time,
# the job is killed.
MachineMaxVacateTime = $(MaxVacateTime)

# Abort graceful eviction of a job, even though it has not
# yet used all the time allotted by MachineMaxVacateTime.
UWCS_KILL = false

##  Only define vanilla versions of these if you want to make them
##  different from the above settings.
#SUSPEND_VANILLA  = ( $(KeyboardBusy) || \
#       ((CpuBusyTime > 2 * $(MINUTE)) && $(ActivationTimer) > 90) )
#CONTINUE_VANILLA = ( $(CPUIdle) && ($(ActivityTimer) > 10) \
#                    && (KeyboardIdle > $(ContinueIdleTime)) )
#PREEMPT_VANILLA  = ( ((Activity == "Suspended") && \
#                    ($(ActivityTimer) > $(MaxSuspendTime))) \
#                    || (SUSPEND_VANILLA && (WANT_SUSPEND == False)) )
#KILL_VANILLA    = false

##  Checkpoint every 3 hours on average, with a +-30 minute random
##  factor to avoid having many jobs hit the checkpoint server at
##  the same time.
UWCS_PERIODIC_CHECKPOINT = $(LastCkpt) > (3 * $(HOUR) + \
                          $RANDOM_INTEGER(-30,30,1) * $(MINUTE) )

##  You might want to checkpoint a little less often.  A good
##  example of this is below.  For jobs smaller than 60 megabytes, we
##  periodic checkpoint every 6 hours.  For larger jobs, we only
##  checkpoint every 12 hours.
#UWCS_PERIODIC_CHECKPOINT          = \
#         ( (TARGET.ImageSize < 60000) && \
#           ($(LastCkpt) > (6  * $(HOUR) + $RANDOM_INTEGER(-30,30,1))) ) || \
#         (  $(LastCkpt) > (12 * $(HOUR) + $RANDOM_INTEGER(-30,30,1)) )

##  The rank expressions used by the negotiator are configured below.
##  This is the order in which ranks are applied by the negotiator:
```

```
##    1. NEGOTIATOR_PRE_JOB_RANK
##    2. rank in job ClassAd
##    3. NEGOTIATOR_POST_JOB_RANK
##    4. cause of preemption (0=user priority,1=startd rank,2=no preemption)
##    5. PREEMPTION_RANK

##  The NEGOTIATOR_PRE_JOB_RANK expression overrides all other ranks
##  that are used to pick a match from the set of possibilities.
##  The following expression matches jobs to unclaimed resources
##  whenever possible, regardless of the job-supplied rank.
UWCS_NEGOTIATOR_PRE_JOB_RANK = RemoteOwner =?= UNDEFINED

##  The NEGOTIATOR_POST_JOB_RANK expression chooses between
##  resources that are equally preferred by the job.
##  The following example expression steers jobs toward
##  faster machines and tends to fill a cluster of multiprocessors
##  breadth-first instead of depth-first.  It also prefers online
##  machines over offline (hibernating) ones.  In this example,
##  the expression is chosen to have no effect when preemption
##  would take place, allowing control to pass on to
##  PREEMPTION_RANK.
UWCS_NEGOTIATOR_POST_JOB_RANK = \
 (RemoteOwner =?= UNDEFINED) * (ifthenElse(isUndefined(KFlops), 1000, Kflops) -
SlotID - 1.0e10*(Offline=?=True))

##  The negotiator will not preempt a job running on a given machine
##  unless the PREEMPTION_REQUIREMENTS expression evaluates to true
##  and the owner of the idle job has a better priority than the owner
##  of the running job.  This expression defaults to true.
UWCS_PREEMPTION_REQUIREMENTS = ((SubmitterGroup =?= RemoteGroup) \
      && ($(StateTimer) > (1 * $(HOUR))) \
         && (RemoteUserPrio > TARGET.SubmitterUserPrio * 1.2)) \
      || (MY.NiceUser == True)

##  The PREEMPTION_RANK expression is used in a case where preemption
##  is the only option and all other negotiation ranks are equal.  For
##  example, if the job has no preference, it is usually preferable to
##  preempt a job that has just started instead of a job with a longer
##  runtime, to cause less badput.  The default is to rank all
##  preemptable matches the same.  However, the negotiator will always
##  prefer to match the job with an idle machine over a preemptable
##  machine, if all other negotiation ranks are equal.

UWCS_PREEMPTION_RANK = (RemoteUserPrio * 1000000) -
ifThenElse(isUndefined(TotalJobRuntime), 0, TotalJobRuntime)
```

```
##########################################################################
##  This is a Configuration that will cause your Condor jobs to
##  always run.  This is intended for testing only.
##########################################################################

##  This mode will cause your jobs to start on a machine an will let
##  them run to completion.  Condor will ignore all of what is going
##  on in the machine (load average, keyboard activity, etc.)

TESTINGMODE_WANT_SUSPEND        = False
TESTINGMODE_WANT_VACATE                 = False
TESTINGMODE_START               = True
TESTINGMODE_SUSPEND                     = False
TESTINGMODE_CONTINUE            = True
TESTINGMODE_PREEMPT                     = False
TESTINGMODE_KILL                = False
TESTINGMODE_PERIODIC_CHECKPOINT       = False
TESTINGMODE_PREEMPTION_REQUIREMENTS = False
TESTINGMODE_PREEMPTION_RANK = 0

# Prevent machine claims from being reused indefinitely, since
# preemption of claims is disabled in the TESTINGMODE configuration.
TESTINGMODE_CLAIM_WORKLIFE = 1200


##########################################################################
##########################################################################
##
## ######                  #
## #   #  ##  #####  #####    #  #
## #   # # # #  #   #     #  #
## ###### #  # #  #   #     #  #
## #    ###### #####   #     #######
## #    #  # # #   #       #
## #    #  # #  #   #       #
##
##  Part 4:  Settings you should probably leave alone:
##  (unless you know what you're doing)
##########################################################################
##########################################################################

##########################################################################
##  Daemon-wide settings:
##########################################################################
```

```
##  Pathnames
LOG             = $(LOCAL_DIR)/log
SPOOL                   = $(LOCAL_DIR)/spool
EXECUTE                 = $(LOCAL_DIR)/execute
BIN             = $(RELEASE_DIR)/bin
LIB             = $(RELEASE_DIR)/lib
INCLUDE                 = $(RELEASE_DIR)/include
SBIN            = $(RELEASE_DIR)/sbin
LIBEXEC                 = $(RELEASE_DIR)/libexec

## If you leave HISTORY undefined (comment it out), no history file
## will be created.
HISTORY                 = $(SPOOL)/history

##  Log files
COLLECTOR_LOG  = $(LOG)/CollectorLog
KBDD_LOG  = $(LOG)/KbdLog
MASTER_LOG        = $(LOG)/MasterLog
NEGOTIATOR_LOG= $(LOG)/NegotiatorLog
NEGOTIATOR_MATCH_LOG = $(LOG)/MatchLog
SCHEDD_LOG        = $(LOG)/SchedLog
SHADOW_LOG        = $(LOG)/ShadowLog
STARTD_LOG        = $(LOG)/StartLog
STARTER_LOG       = $(LOG)/StarterLog
JOB_ROUTER_LOG  = $(LOG)/JobRouterLog
ROOSTER_LOG     = $(LOG)/RoosterLog
SHARED_PORT_LOG = $(LOG)/SharedPortLog
# High Availability Logs
HAD_LOG                 = $(LOG)/HADLog
REPLICATION_LOG           = $(LOG)/ReplicationLog
TRANSFERER_LOG= $(LOG)/TransfererLog
HDFS_LOG    = $(LOG)/HDFSLog

##  Lock files
SHADOW_LOCK     = $(LOCK)/ShadowLock

## This setting controls how often any lock files currently in use have their
## time stamp updated. Updating the time stamp prevents administrative programs
## like 'tmpwatch' from deleting long lived lock files. The parameter is
## an integer in seconds with a minimum of 60 seconds. The default if not
## specified is 28800 seconds, or 8 hours.
## This attribute only takes effect on restart of the daemons or at the next
## update time.
# LOCK_FILE_UPDATE_INTERVAL = 28800

##  This setting primarily allows you to change the port that the
```

```
##  collector is listening on.  By default, the collector uses port
##  9618, but you can set the port with a ":port", such as:
##  COLLECTOR_HOST = $(CONDOR_HOST):1234
COLLECTOR_HOST  = $(CONDOR_HOST)

## The NEGOTIATOR_HOST parameter has been deprecated.  The port where
## the negotiator is listening is now dynamically allocated and the IP
## and port are now obtained from the collector, just like all the
## other daemons.  However, if your pool contains any machines that
## are running version 6.7.3 or earlier, you can uncomment this
## setting to go back to the old fixed-port (9614) for the negotiator.
#NEGOTIATOR_HOST = $(CONDOR_HOST)

##  How long are you willing to let daemons try their graceful
##  shutdown methods before they do a hard shutdown? (30 minutes)
#SHUTDOWN_GRACEFUL_TIMEOUT    = 1800

##  How much disk space would you like reserved from Condor?  In
##  places where Condor is computing the free disk space on various
##  partitions, it subtracts the amount it really finds by this
##  many megabytes.  (If undefined, defaults to 0).
RESERVED_DISK               = 5

##  If your machine is running AFS and the AFS cache lives on the same
##  partition as the other Condor directories, and you want Condor to
##  reserve the space that your AFS cache is configured to use, set
##  this to true.
#RESERVE_AFS_CACHE   = False

##  By default, if a user does not specify "notify_user" in the submit
##  description file, any email Condor sends about that job will go to
##  "username@UID_DOMAIN".  If your machines all share a common UID
##  domain (so that you would set UID_DOMAIN to be the same across all
##  machines in your pool), *BUT* email to user@UID_DOMAIN is *NOT*
##  the right place for Condor to send email for your site, you can
##  define the default domain to use for email.  A common example
##  would be to set EMAIL_DOMAIN to the fully qualified hostname of
##  each machine in your pool, so users submitting jobs from a
##  specific machine would get email sent to user@machine.your.domain,
##  instead of user@your.domain.  In general, you should leave this
##  setting commented out unless two things are true: 1) UID_DOMAIN is
##  set to your domain, not $(FULL_HOSTNAME), and 2) email to
##  user@UID_DOMAIN won't work.
#EMAIL_DOMAIN = $(FULL_HOSTNAME)

##  Should Condor daemons create a UDP command socket (for incoming
```

```
##  UDP-based commands) in addition to the TCP command socket?  By
##  default, classified ad updates sent to the collector use UDP, in
##  addition to some keep alive messages and other non-essential
##  communication.  However, in certain situations, it might be
##  desirable to disable the UDP command port (for example, to reduce
##  the number of ports represented by a CCB broker, etc).  If not
##  defined, the UDP command socket is enabled by default, and to
##  modify this, you must restart your Condor daemons. Also, this
##  setting must be defined machine-wide.  For example, setting
##  "STARTD.WANT_UDP_COMMAND_SOCKET = False" while the global setting
##  is "True" will still result in the startd creating a UDP socket.
#WANT_UDP_COMMAND_SOCKET = True

##  If your site needs to use TCP updates to the collector, instead of
##  UDP, you can enable this feature.  HOWEVER, WE DO NOT RECOMMEND
##  THIS FOR MOST SITES!  In general, the only sites that might want
##  this feature are pools made up of machines connected via a
##  wide-area network where UDP packets are frequently or always
##  dropped.  If you enable this feature, you *MUST* turn on the
##  COLLECTOR_SOCKET_CACHE_SIZE setting at your collector, and each
##  entry in the socket cache uses another file descriptor.  If not
##  defined, this feature is disabled by default.
#UPDATE_COLLECTOR_WITH_TCP = True

## HIGHPORT and LOWPORT let you set the range of ports that Condor
## will use. This may be useful if you are behind a firewall. By
## default, Condor uses port 9618 for the collector, 9614 for the
## negotiator, and system-assigned (apparently random) ports for
## everything else. HIGHPORT and LOWPORT only affect these
## system-assigned ports, but will restrict them to the range you
## specify here. If you want to change the well-known ports for the
## collector or negotiator, see COLLECTOR_HOST or NEGOTIATOR_HOST.
## Note that both LOWPORT and HIGHPORT must be at least 1024 if you
## are not starting your daemons as root.  You may also specify
## different port ranges for incoming and outgoing connections by
## using IN_HIGHPORT/IN_LOWPORT and OUT_HIGHPORT/OUT_LOWPORT.
#HIGHPORT = 9700
#LOWPORT = 9600

##  If a daemon does not respond for too long, do you want go generate
##  a core file?  This basically controls the type of the signal
##  sent to the child process, and mostly affects the Condor Master
#NOT_RESPONDING_WANT_CORE       = False


##############################################################################
```

```
##  Daemon-specific settings:
##--------------------------------------------------------------------

##--------------------------------------------------------------------
##  condor_master
##--------------------------------------------------------------------
##  Daemons you want the master to keep running for you:
##MASTER
DAEMON_LIST                 = MASTER, COLLECTOR, NEGOTIATOR
##SUBMITTER AND          EXECUTER
#DAEMON_LIST = MASTER, STARTD, SCHEDD
##EXECUTER
##DAEMON_LIST = MASTER, STARTD

##  Which daemons use the Condor DaemonCore library (i.e., not the
##  checkpoint server or custom user daemons)?
#DC_DAEMON_LIST = \
#MASTER, STARTD, SCHEDD, KBDD, COLLECTOR, NEGOTIATOR, EVENTD, \
#VIEW_SERVER, CONDOR_VIEW, VIEW_COLLECTOR, HAWKEYE, CREDD, HAD, \
#DBMSD, QUILL, JOB_ROUTER, ROOSTER, LEASEMANAGER, HDFS,
SHARED_PORT, \
#DEFRAG, GANGLIAD


##  Where are the binaries for these daemons?
MASTER                              = $(SBIN)/condor_master
STARTD                              = $(SBIN)/condor_startd
SCHEDD                              = $(SBIN)/condor_schedd
KBDD                          = $(SBIN)/condor_kbdd
NEGOTIATOR                          = $(SBIN)/condor_negotiator
COLLECTOR                     = $(SBIN)/condor_collector
CKPT_SERVER                         = $(SBIN)/condor_ckpt_server
STARTER_LOCAL                       = $(SBIN)/condor_starter
JOB_ROUTER                  = $(LIBEXEC)/condor_job_router
ROOSTER                   = $(LIBEXEC)/condor_rooster
HDFS                          = $(SBIN)/condor_hdfs
SHARED_PORT                         = $(LIBEXEC)/condor_shared_port
TRANSFERER                          = $(LIBEXEC)/condor_transferer
DEFRAG                    = $(LIBEXEC)/condor_defrag
GANGLIAD                    = $(LIBEXEC)/condor_gangliad

##  When the master starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need to query the central manager to find the master.  This
##  feature can be turned off by commenting out this setting.
MASTER_ADDRESS_FILE = $(LOG)/.master_address
```

```
##  Where should the master find the condor_preen binary? If you don't
##  want preen to run at all, set it to nothing.
PREEN                           = $(SBIN)/condor_preen

##  How do you want preen to behave?  The "-m" means you want email
##  about files preen finds that it thinks it should remove.  The "-r"
##  means you want preen to actually remove these files.  If you don't
##  want either of those things to happen, just remove the appropriate
##  one from this setting.
PREEN_ARGS                      = -m -r

##  How often should the master start up condor_preen? (once a day)
#PREEN_INTERVAL                 = 86400

##  If a daemon dies an unnatural death, do you want email about it?
#PUBLISH_OBITUARIES             = True

##  If you're getting obituaries, how many lines of the end of that
##  daemon's log file do you want included in the obituary?
#OBITUARY_LOG_LENGTH            = 20

##  Should the master run?
#START_MASTER                   = True

##  Should the master start up the daemons you want it to?
#START_DAEMONS                  = True

##  How often do you want the master to send an update to the central
##  manager?
#MASTER_UPDATE_INTERVAL         = 300

##  How often do you want the master to check the time stamps of the
##  running daemons?  If any daemons have been modified, the
##  master restarts them.
#MASTER_CHECK_NEW_EXEC_INTERVAL  = 300

##  Once you notice new binaries, how long should you wait before you
##  try to execute them?
#MASTER_NEW_BINARY_DELAY        = 120

##  What's the maximum amount of time you're willing to give the
##  daemons to quickly shutdown before you just kill them outright?
#SHUTDOWN_FAST_TIMEOUT          = 120

######
```

```
##  Exponential back off settings:
######
##  When a daemon keeps crashing, we use "exponential back off" so we
##  wait longer and longer before restarting it.  This is the base of
##  the exponent used to determine how long to wait before starting
##  the daemon again:
#MASTER_BACKOFF_FACTOR              = 2.0

##  What's the maximum amount of time you want the master to wait
##  between attempts to start a given daemon?  (With 2.0 as the
##  MASTER_BACKOFF_FACTOR, you'd hit 1 hour in 12 restarts...)
#MASTER_BACKOFF_CEILING             = 3600

##  How long should a daemon run without crashing before we consider
##  it "recovered".  Once a daemon has recovered, we reset the number
##  of restarts so the exponential back off stuff goes back to normal.
#MASTER_RECOVER_FACTOR              = 300


##--------------------------------------------------------------------
##  condor_collector
##--------------------------------------------------------------------
## Address to which Condor will send a weekly e-mail with basic
## non-specific information about your pool. See
## http://htcondor.org/privacy.html for more information.
## To disable this behavior, uncomment the below line to
## explicitly set CONDOR_DEVELOPERS to NONE.
#CONDOR_DEVELOPERS = condor-admin@cs.wisc.edu
#CONDOR_DEVELOPERS = NONE

## Global Collector to periodically advertise basic
## non-specific information about your pool. See
## http://htcondor.org/privacy.html for more information.
## To disable this behavior, uncomment the below line to
## explicitly set CONDOR_DEVELOPERS_COLLECTOR to NONE.
#CONDOR_DEVELOPERS_COLLECTOR = condor.cs.wisc.edu
#CONDOR_DEVELOPERS_COLLECTOR = NONE

##  When the collector starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need to query the central manager to find the collector.  This
##  feature can be turned off by commenting out this setting.
##  This is essential when using a port of "0" (automatic) for the
##  COLLECTOR_HOST, a useful technique for personal Condor installs.
COLLECTOR_ADDRESS_FILE = $(LOG)/.collector_address
```

```
##
##  Conventional HTCondor installations start up as root, and can thus
##  set their own file descriptor limit.  Upping the collector's doesn't
##  hurt anything and ameliorates a common scalability problem.
##
# COLLECTOR_MAX_FILE_DESCRIPTORS = 10240


##--------------------------------------------------------------------
##  condor_negotiator
##--------------------------------------------------------------------
## Determine if the Negotiator will honor SlotWeight attributes, which
## may be used to give a slot greater weight when calculating usage.
#NEGOTIATOR_USE_SLOT_WEIGHTS = True


## How often the Negotaitor starts a negotiation cycle, defined in
## seconds.
#NEGOTIATOR_INTERVAL = 60

## Should the Negotiator publish an update to the Collector after
## every negotiation cycle. It is useful to have this set to True
## to get immediate updates on LastNegotiationCycle statistics.
#NEGOTIATOR_UPDATE_AFTER_CYCLE = False


##--------------------------------------------------------------------
##  condor_startd
##--------------------------------------------------------------------
##  Where are the various condor_starter binaries installed?
STARTER_LIST = STARTER, STARTER_STANDARD
STARTER                 = $(SBIN)/condor_starter
STARTER_STANDARD        = $(SBIN)/condor_starter.std
STARTER_LOCAL           = $(SBIN)/condor_starter

##  When the startd starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need to query the central manager to find the startd.  This
##  feature can be turned off by commenting out this setting.
STARTD_ADDRESS_FILE  = $(LOG)/.startd_address

##  When a machine is claimed, how often should we poll the state of
##  the machine to see if we need to evict/suspend the job, etc?
#POLLING_INTERVAL        = 5

##  How often should the startd send updates to the central manager?
```

```
#UPDATE_INTERVAL        = 300

##  How long is the startd willing to stay in the "matched" state?
#MATCH_TIMEOUT                  = 300

##  How long is the startd willing to stay in the preempting/killing
##  state before it just kills the starter directly?
#KILLING_TIMEOUT         = 30

##  When a machine unclaimed, when should it run benchmarks?
##  LastBenchmark is initialized to 0, so this expression says as soon
##  as we're unclaimed, run the benchmarks.  Thereafter, if we are
##  unclaimed and it has been at least 4 hours since we ran the last
##  benchmarks, run them again.  The startd keeps a weighted average
##  of the benchmark results to provide more accurate values.
##  Note, if you don't want any benchmarks run at all, set RunBenchmarks to "False".
#RunBenchmarks = False
#RunBenchmarks = (LastBenchmark == 0 ) || ((time() - LastBenchmark) >= (4 * $(HOUR)))

##  When the startd does benchmarks, which set of benchmarks should we
##  run?  The default is the same as pre-7.5.6: MIPS and KFLOPS.
benchmarks_joblist = mips kflops

##  What's the max "load" of all running benchmarks?  With the default
##  (1.01), the startd will run the benchmarks serially.
benchmarks_max_job_load = 1.0

#  MIPS (Dhrystone 2.1) benchmark: load 1.0
benchmarks_mips_executable = $(LIBEXEC)/condor_mips
benchmarks_mips_job_load = 1.0

#  KFLOPS (clinpack) benchmark: load 1.0
benchmarks_kflops_executable = $(LIBEXEC)/condor_kflops
benchmarks_kflops_job_load = 1.0


##  Normally, when the startd is computing the idle time of all the
##  users of the machine (both local and remote), it checks the utmp
##  file to find all the currently active ttys, and only checks access
##  time of the devices associated with active logins.  Unfortunately,
##  on some systems, utmp is unreliable, and the startd might miss
##  keyboard activity by doing this.  So, if your utmp is unreliable,
##  set this setting to True and the startd will check the access time
##  on all tty and pty devices.
#STARTD_HAS_BAD_UTMP = False
```

```
##  This entry allows the startd to monitor console (keyboard and
##  mouse) activity by checking the access times on special files in
##  /dev.  Activity on these files shows up as "ConsoleIdle" time in
##  the startd's ClassAd.  Just give a comma-separated list of the
##  names of devices you want considered the console, without the
##  "/dev/" portion of the pathname.
#CONSOLE_DEVICES = mouse, console


##  The STARTD_ATTRS (formerly STARTD_EXPRS) entry allows you to
##  have the startd advertise arbitrary attributes from the config
##  file in its ClassAd.  Give the comma-separated list of entries
##  from the config file you want in the startd ClassAd.
##  NOTE: because of the different syntax of the config file and
##  ClassAds, you might have to do a little extra work to get a given
##  entry into the ClassAd.  In particular, ClassAds require double
##  quotes (") around your strings.  Numeric values can go in
##  directly, as can boolean expressions.  For example, if you wanted
##  the startd to advertise its list of console devices, when it is
##  configured to run benchmarks, and how often it sends updates to
##  the central manager, you'd have to define the following helper
##  macro:
#MY_CONSOLE_DEVICES = "$(CONSOLE_DEVICES)"
##  Note: this must come before you define STARTD_ATTRS because macros
##  must be defined before you use them in other macros or
##  expressions.
##  Then, you'd set the STARTD_ATTRS setting to this:
#STARTD_ATTRS = MY_CONSOLE_DEVICES, RunBenchmarks, UPDATE_INTERVAL
##
##  STARTD_ATTRS can also be defined on a per-slot basis.  The startd
##  builds the list of attributes to advertise by combining the lists
##  in this order: STARTD_ATTRS, SLOTx_STARTD_ATTRS.  In the below
##  example, the startd ad for slot1 will have the value for
##  favorite_color, favorite_season, and favorite_movie, and slot2
##  will have favorite_color, favorite_season, and favorite_song.
##
#STARTD_ATTRS = favorite_color, favorite_season
#SLOT1_STARTD_ATTRS = favorite_movie
#SLOT2_STARTD_ATTRS = favorite_song
##
##  Attributes in the STARTD_ATTRS list can also be on a per-slot basis.
##  For example, the following configuration:
##
#favorite_color = "blue"
#favorite_season = "spring"
#SLOT2_favorite_color = "green"
```

```
#SLOT3_favorite_season = "summer"
#STARTD_ATTRS = favorite_color, favorite_season
##
##  will result in the following attributes in the slot classified
##  ads:
##
## slot1 - favorite_color = "blue"; favorite_season = "spring"
## slot2 - favorite_color = "green"; favorite_season = "spring"
## slot3 - favorite_color = "blue"; favorite_season = "summer"
##
##  Finally, there is an automatic mechanism to
##  publish the COLLECTOR_HOST setting as a string.  This can be
##  useful using the "$$(COLLECTOR_HOST_STRING)" syntax in the submit file
##  for jobs to know (for example, via their environment) what pool
##  they're running in.  You can change the value published in the
##  COLLECTOR_HOST_STRING attribute by changing this value,
##  (this must be a value classad string value, so don't forget the quotes!)
#COLLECTOR_HOST_STRING = "$(COLLECTOR_HOST)"

##  When the startd is claimed by a remote user, it can also advertise
##  arbitrary attributes from the ClassAd of the job it is working on.
##  Just list the attribute names you want advertised.
##  Note: since this is already a ClassAd, you don't have to do quoting.
##  Note2: the job attributes ImageSize, ExecutableSize, JobUniverse, NiceUser are
automatically advertised.
#STARTD_JOB_ATTRS =

##  If you want to "lie" to Condor about how many CPUs your machine
##  has, you can use this setting to override Condor's automatic
##  computation.  If you modify this, you must restart the startd for
##  the change to take effect (a simple condor_reconfig will not do).
##  Please read the section on "condor_startd Configuration File
##  Macros" in the Condor Administrators Manual for a further
##  discussion of this setting.  Its use is not recommended.  This
##  must be an integer ("N" is not a valid setting, that's just used to
##  represent the default).
#NUM_CPUS = N

##  If you never want Condor to detect more the "N" CPUs, uncomment this
##  line out. You must restart the startd for this setting to take
##  effect. If set to 0 or a negative number, it is ignored.
##  By default, it is ignored. Otherwise, it must be a positive
##  integer ("N" is not a valid setting, that's just used to
##  represent the default).
#MAX_NUM_CPUS = N
```

```
##  Normally, Condor will automatically detect the amount of physical
##  memory available on your machine.  Define MEMORY to tell Condor
##  how much physical memory (in MB) your machine has, overriding the
##  value Condor computes automatically.  For example:
#MEMORY = 128

##  How much memory would you like reserved from Condor?  By default,
##  Condor considers all the physical memory of your machine as
##  available to be used by Condor jobs.  If RESERVED_MEMORY is
##  defined, Condor subtracts it from the amount of memory it
##  advertises as available.
#RESERVED_MEMORY = 0

######
##  SMP startd settings
##
##  By default, Condor will evenly divide the resources in an SMP
##  machine (such as RAM, swap space and disk space) among all the
##  CPUs, and advertise each CPU as its own slot with an even share of
##  the system resources.  If you want something other than this,
##  there are a few options available to you.  Please read the section
##  on "Configuring The Startd for SMP Machines" in the Condor
##  Administrator's Manual for full details.  The various settings are
##  only briefly listed and described here.
######

##  The maximum number of different slot types.
#MAX_SLOT_TYPES = 10

##  Use this setting to define your own slot types.  This
##  allows you to divide system resources unevenly among your CPUs.
##  You must use a different setting for each different type you
##  define.  The "<N>" in the name of the macro listed below must be
##  an integer from 1 to MAX_SLOT_TYPES (defined above),
##  and you use this number to refer to your type.  There are many
##  different formats these settings can take, so be sure to refer to
##  the section on "Configuring The Startd for SMP Machines" in the
##  Condor Administrator's Manual for full details.  In particular,
##  read the section titled "Defining Slot Types" to help
##  understand this setting.  If you modify any of these settings, you
##  must restart the condor_start for the change to take effect.
#SLOT_TYPE_<N> = 1/4
#SLOT_TYPE_<N> = cpus=1, ram=25%, swap=1/4, disk=1/4
#  For example:
#SLOT_TYPE_1 = 1/8
#SLOT_TYPE_2 = 1/4
```

```
##  If you define your own slot types, you must specify how
##  many slots of each type you wish to advertise.  You do
##  this with the setting below, replacing the "<N>" with the
##  corresponding integer you used to define the type above.  You can
##  change the number of a given type being advertised at run-time,
##  with a simple condor_reconfig.
#NUM_SLOTS_TYPE_<N> = M
#  For example:
#NUM_SLOTS_TYPE_1 = 6
#NUM_SLOTS_TYPE_2 = 1

##  The number of evenly-divided slots you want Condor to
##  report to your pool (if less than the total number of CPUs).  This
##  setting is only considered if the "type" settings described above
##  are not in use.  By default, all CPUs are reported.  This setting
##  must be an integer ("N" is not a valid setting, that's just used to
##  represent the default).
#NUM_SLOTS = N

##  How many of the slots the startd is representing should
##  be "connected" to the console (in other words, notice when there's
##  console activity)?  This defaults to all slots
#SLOTS_CONNECTED_TO_CONSOLE = $(NUM_CPUS)

##  How many of the slots the startd is representing should
##  be "connected" to the keyboard (for remote tty activity, as well
##  as console activity).  Defaults to all slots
#SLOTS_CONNECTED_TO_KEYBOARD = $(NUM_CPUS)

##  If there are slots that are not connected to the
##  keyboard or the console (see the above two settings), the
##  corresponding idle time reported will be the time since the startd
##  was spawned, plus the value of this parameter.  It defaults to 20
##  minutes.  We do this because, if the slot is configured
##  not to care about keyboard activity, we want it to be available to
##  Condor jobs as soon as the startd starts up, instead of having to
##  wait for 15 minutes or more (which is the default time a machine
##  must be idle before Condor will start a job).  If you don't want
##  this boost, just set the value to 0.  If you change your START
##  expression to require more than 15 minutes before a job starts,
##  but you still want jobs to start right away on some of your SMP
##  nodes, just increase this parameter.
#DISCONNECTED_KEYBOARD_IDLE_BOOST = 1200

######
```

```
##  Settings for computing optional resource availability statistics:
######
##  If STARTD_COMPUTE_AVAIL_STATS = True, the startd will compute
##  statistics about resource availability to be included in the
##  classad(s) sent to the collector describing the resource(s) the
##  startd manages.  The following attributes will always be included
##  in the resource classad(s) if STARTD_COMPUTE_AVAIL_STATS = True:
##    AvailTime = What proportion of the time (between 0.0 and 1.0)
##      has this resource been in a state other than "Owner"?
##    LastAvailInterval = What was the duration (in seconds) of the
##      last period between "Owner" states?
##  The following attributes will also be included if the resource is
##  not in the "Owner" state:
##    AvailSince = At what time did the resource last leave the
##      "Owner" state?  Measured in the number of seconds since the
##      epoch (00:00:00 UTC, Jan 1, 1970).
##    AvailTimeEstimate = Based on past history, this is an estimate
##      of how long the current period between "Owner" states will
##      last.
#STARTD_COMPUTE_AVAIL_STATS = False

##  If STARTD_COMPUTE_AVAIL_STATS = True, STARTD_AVAIL_CONFIDENCE
sets
##  the confidence level of the AvailTimeEstimate.  By default, the
##  estimate is based on the 80th percentile of past values.
#STARTD_AVAIL_CONFIDENCE = 0.8

##  STARTD_MAX_AVAIL_PERIOD_SAMPLES limits the number of samples of
##  past available intervals stored by the startd to limit memory and
##  disk consumption.  Each sample requires 4 bytes of memory and
##  approximately 10 bytes of disk space.
#STARTD_MAX_AVAIL_PERIOD_SAMPLES = 100

##       CKPT_PROBE is the location of a program which computes aspects of the
##       CheckpointPlatform classad attribute. By default the location of this
##       executable will be here: $(LIBEXEC)/condor_ckpt_probe
CKPT_PROBE = $(LIBEXEC)/condor_ckpt_probe

##--------------------------------------------------------------------
##  condor_schedd
##--------------------------------------------------------------------
##  Where are the various shadow binaries installed?
SHADOW_LIST = SHADOW, SHADOW_STANDARD
SHADOW                  = $(SBIN)/condor_shadow
SHADOW_STANDARD         = $(SBIN)/condor_shadow.std
```

```
##  When the schedd starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need to query the central manager to find the schedd.  This
##  feature can be turned off by commenting out this setting.
SCHEDD_ADDRESS_FILE = $(SPOOL)/.schedd_address

##  Additionally, a daemon may store its ClassAd on the local filesystem
##  as well as sending it to the collector. This way, tools that need
##  information about a daemon do not have to contact the central manager
##  to get information about a daemon on the same machine.
##  This feature is necessary for Quill to work.
SCHEDD_DAEMON_AD_FILE = $(SPOOL)/.schedd_classad

##  How often should the schedd send an update to the central manager?
#SCHEDD_INTERVAL        = 300

##  How long should the schedd wait between spawning each shadow?
#JOB_START_DELAY        = 2

##  How many concurrent sub-processes should the schedd spawn to handle
##  queries?  (UNIX only)
#SCHEDD_QUERY_WORKERS   = 3

##  How often should the schedd send a keep alive message to any
##  startds it has claimed?  (5 minutes)
#ALIVE_INTERVAL         = 300

##  This setting controls the maximum number of times that a
##  condor_shadow processes can have a fatal error (exception) before
##  the condor_schedd will simply relinquish the match associated with
##  the dying shadow.
#MAX_SHADOW_EXCEPTIONS  = 5

##  Estimated virtual memory size of each condor_shadow process.
##  Specified in kilobytes.
# SHADOW_SIZE_ESTIMATE      = 800

##  The condor_schedd can renice the condor_shadow processes on your
##  submit machines.  How "nice" do you want the shadows? (1-19).
##  The higher the number, the lower priority the shadows have.
# SHADOW_RENICE_INCREMENT         = 0

## The condor_schedd can renice scheduler universe processes
## (e.g. DAGMan) on your submit machines.  How "nice" do you want the
## scheduler universe processes? (1-19).  The higher the number, the
## lower priority the processes have.
```

```
# SCHED_UNIV_RENICE_INCREMENT = 0

##  By default, when the schedd fails to start an idle job, it will
##  not try to start any other idle jobs in the same cluster during
##  that negotiation cycle.  This makes negotiation much more
##  efficient for large job clusters.  However, in some cases other
##  jobs in the cluster can be started even though an earlier job
##  can't.  For example, the jobs' requirements may differ, because of
##  different disk space, memory, or operating system requirements.
##  Or, machines may be willing to run only some jobs in the cluster,
##  because their requirements reference the jobs' virtual memory size
##  or other attribute.  Setting NEGOTIATE_ALL_JOBS_IN_CLUSTER to True
##  will force the schedd to try to start all idle jobs in each
##  negotiation cycle.  This will make negotiation cycles last longer,
##  but it will ensure that all jobs that can be started will be
##  started.
#NEGOTIATE_ALL_JOBS_IN_CLUSTER = False

## This setting controls how often, in seconds, the schedd considers
## periodic job actions given by the user in the submit file.
## (Currently, these are periodic_hold, periodic_release, and periodic_remove.)
#PERIODIC_EXPR_INTERVAL = 60

##
##  Conventional HTCondor installations start up as root, and can thus
##  set their own file descriptor limit.  Upping the schedd's limit has
##  some minor downsides (larger buffers passed to select() and the like),
##  but upping it makes a class of difficult-to-debug problems much rarer.
##
# SCHEDD_MAX_FILE_DESCRIPTORS = 4096

######
## Queue management settings:
######
##  How often should the schedd truncate it's job queue transaction
##  log?  (Specified in seconds, once a day is the default.)
#QUEUE_CLEAN_INTERVAL       = 86400

##  How often should the schedd commit "wall clock" run time for jobs
##  to the queue, so run time statistics remain accurate when the
##  schedd crashes?  (Specified in seconds, once per hour is the
##  default.  Set to 0 to disable.)
#WALL_CLOCK_CKPT_INTERVAL = 3600

##  What users do you want to grant super user access to this job
##  queue?  (These users will be able to remove other user's jobs).
```

```
##  By default, this only includes root.
QUEUE_SUPER_USERS      = root, condor


##--------------------------------------------------------------------
##  condor_shadow
##--------------------------------------------------------------------
##  If the shadow is unable to read a checkpoint file from the
##  checkpoint server, it keeps trying only if the job has accumulated
##  more than MAX_DISCARDED_RUN_TIME seconds of CPU usage.  Otherwise,
##  the job is started from scratch.  Defaults to 1 hour.  This
##  setting is only used if USE_CKPT_SERVER (from above) is True.
#MAX_DISCARDED_RUN_TIME = 3600

##  Should periodic checkpoints be compressed?
#COMPRESS_PERIODIC_CKPT = False

##  Should vacate checkpoints be compressed?
#COMPRESS_VACATE_CKPT = False

##  Should we commit the application's dirty memory pages to swap
##  space during a periodic checkpoint?
#PERIODIC_MEMORY_SYNC = False

##  Should we write vacate checkpoints slowly?  If nonzero, this
##  parameter specifies the speed at which vacate checkpoints should
##  be written, in kilobytes per second.
#SLOW_CKPT_SPEED = 0

##  How often should the shadow update the job queue with job
##  attributes that periodically change?  Specified in seconds.
#SHADOW_QUEUE_UPDATE_INTERVAL = 15 * 60

##  Should the shadow wait to update certain job attributes for the
##  next periodic update, or should it immediately these update
##  attributes as they change?  Due to performance concerns of
##  aggressive updates to a busy condor_schedd, the default is True.
#SHADOW_LAZY_QUEUE_UPDATE = TRUE


##--------------------------------------------------------------------
##  condor_starter
##--------------------------------------------------------------------
##  The condor_starter can renice the processes of Condor
##  jobs on your execute machines.  If you want this, uncomment the
##  following entry and set it to how "nice" you want the user
```

```
##  jobs. (1-19)  The larger the number, the lower priority the
##  process gets on your machines.
##  Note on Win32 platforms, this number needs to be greater than
##  zero (i.e. the job must be reniced) or the mechanism that
##  monitors CPU load on Win32 systems will give erratic results.
#JOB_RENICE_INCREMENT         = 10

##  Should the starter do local logging to its own log file, or send
##  debug information back to the condor_shadow where it will end up
##  in the ShadowLog?
#STARTER_LOCAL_LOGGING     = TRUE

##  If the UID_DOMAIN settings match on both the execute and submit
##  machines, but the UID of the user who submitted the job is not in
##  the passwd file of the execute machine, the starter will normally
##  exit with an error.  Do you want the starter to just start up the
##  job with the specified UID, even if it is not in the passwd file?
#SOFT_UID_DOMAIN         = FALSE

##  honor the run_as_owner option from the condor submit file.
##
#STARTER_ALLOW_RUNAS_OWNER = TRUE

##  Tell the Starter/Startd what program to use to remove a directory
##  condor_rmdir.exe is a windows-only command that does a better job
##  than the built-in rmdir command when it is run with elevated privileges
##  Such as when when Condor is running as a service.
##   /s is delete sub-directories
##   /c is continue on error
WINDOWS_RMDIR = $(SBIN)\condor_rmdir.exe
#WINDOWS_RMDIR_OPTIONS = /s /c

##--------------------------------------------------------------------
##  condor_procd
##--------------------------------------------------------------------
##
# the path to the procd binary
#
PROCD = $(SBIN)/condor_procd

# the path to the procd "address"
#   - on UNIX this will be a named pipe; we'll put it in the
#     $(LOCK) directory by default (note that multiple named pipes
#     will be created in this directory for when the procd responds
#     to its clients)
#   - on Windows, this will be a named pipe as well (but named pipes on
```

```
#     Windows are not even close to the same thing as named pipes on
#     UNIX); the name will be something like:
#         \\.\pipe\condor_procd
#
PROCD_ADDRESS = $(LOCK)/procd_pipe

# Note that in other Condor daemons, turning on D_PROCFAMILY will
# result in that daemon logging all of its interactions with the
# ProcD.
#
PROCD_LOG = $(LOG)/ProcLog

# This is the maximum period that the procd will use for taking
# snapshots (the actual period may be lower if a condor daemon registers
# a family for which it wants more frequent snapshots)
#
PROCD_MAX_SNAPSHOT_INTERVAL = 60

# On Windows, we send a process a "soft kill" via a WM_CLOSE message.
# This binary is used by the ProcD (and other Condor daemons if PRIVSEP
# is not enabled) to help when sending soft kills.
WINDOWS_SOFTKILL = $(SBIN)/condor_softkill

##--------------------------------------------------------------------
##  condor_submit
##--------------------------------------------------------------------
##  If you want condor_submit to automatically append an expression to
##  the Requirements expression or Rank expression of jobs at your
##  site, uncomment these entries.
#APPEND_REQUIREMENTS        = (expression to append job requirements)
#APPEND_RANK                = (expression to append job rank)

##  If you want expressions only appended for either standard or
##  vanilla universe jobs, you can uncomment these entries.  If any of
##  them are defined, they are used for the given universe, instead of
##  the generic entries above.
#APPEND_REQ_VANILLA = (expression to append to vanilla job requirements)
#APPEND_REQ_STANDARD        = (expression to append to standard job requirements)
#APPEND_RANK_STANDARD     = (expression to append to vanilla job rank)
#APPEND_RANK_VANILLA        = (expression to append to standard job rank)

##  This can be used to define a default value for the rank expression
##  if one is not specified in the submit file.
#DEFAULT_RANK          = (default rank expression for all jobs)

##  If you want universe-specific defaults, you can use the following
```

```
##  entries:
#DEFAULT_RANK_VANILLA      = (default rank expression for vanilla jobs)
#DEFAULT_RANK_STANDARD   = (default rank expression for standard jobs)

##  If you want condor_submit to automatically append expressions to
##  the job ClassAds it creates, you can uncomment and define the
##  SUBMIT_ATTRS (formerly SUBMIT_EXPRS) setting.  It works just like the
STARTD_ATTRS
##  described above with respect to ClassAd vs. config file syntax,
##  strings, etc.  One common use would be to have the full hostname
##  of the machine where a job was submitted placed in the job
##  ClassAd.  You would do this by uncommenting the following lines:
#Machine = "$(FULL_HOSTNAME)"
#SUBMIT_ATTRS = Machine

## Condor keeps a buffer of recently-used data for each file an
## application opens.  This macro specifies the default maximum number
## of bytes to be buffered for each open file at the executing
## machine.
#DEFAULT_IO_BUFFER_SIZE = 524288

## Condor will attempt to consolidate small read and write operations
## into large blocks.  This macro specifies the default block size
## Condor will use.
#DEFAULT_IO_BUFFER_BLOCK_SIZE = 32768

##--------------------------------------------------------------------
##  condor_preen
##--------------------------------------------------------------------
##  Who should condor_preen send email to?
#PREEN_ADMIN                = $(CONDOR_ADMIN)


##--------------------------------------------------------------------
##  Java parameters:
##--------------------------------------------------------------------
##  If you would like this machine to be able to run Java jobs,
##  then set JAVA to the path of your JVM binary.  If you are not
##  interested in Java, there is no harm in leaving this entry
##  empty or incorrect.

JAVA = /usr/bin/java

## JAVA_CLASSPATH_DEFAULT gives the default set of paths in which
## Java classes are to be found.  Each path is separated by spaces.
## If your JVM needs to be informed of additional directories, add
```

```
## them here.  However, do not remove the existing entries, as Condor
## needs them.

JAVA_CLASSPATH_DEFAULT = $(LIB) $(LIB)/scimark2lib.jar .

##  JAVA_CLASSPATH_ARGUMENT describes the command-line parameter
##  used to introduce a new classpath:

JAVA_CLASSPATH_ARGUMENT = -classpath

##  JAVA_CLASSPATH_SEPARATOR describes the character used to mark
##  one path element from another:

JAVA_CLASSPATH_SEPARATOR = :

##  JAVA_BENCHMARK_TIME describes the number of seconds for which
##  to run Java benchmarks.  A longer time yields a more accurate
##  benchmark, but consumes more otherwise useful CPU time.
##  If this time is zero or undefined, no Java benchmarks will be run.

JAVA_BENCHMARK_TIME = 2

##  If your JVM requires any special arguments not mentioned in
##  the options above, then give them here.

JAVA_EXTRA_ARGUMENTS =

##
##--------------------------------------------------------------------
##  Condor-G settings
##--------------------------------------------------------------------
##  Where is the GridManager binary installed?

GRIDMANAGER                     = $(SBIN)/condor_gridmanager
GT2_GAHP                = $(SBIN)/gahp_server
GRID_MONITOR                    = $(SBIN)/grid_monitor

##--------------------------------------------------------------------
##  Settings that control the daemon's debugging output:
##--------------------------------------------------------------------
##
## Note that the Gridmanager runs as the User, not a Condor daemon, so
## all users must have write permission to the directory that the
## Gridmanager will use for it's logfile. Our suggestion is to create a
## directory called GridLogs in $(LOG) with UNIX permissions 1777
## (just like /tmp )
```

```
##  Another option is to use /tmp as the location of the GridManager log.
##

#MAX_GRIDMANAGER_LOG      = $(MAX_DEFAULT_LOG)
#GRIDMANAGER_DEBUG=

GRIDMANAGER_LOG = $(LOG)/GridmanagerLog.$(USERNAME)
GRIDMANAGER_LOCK = $(LOCK)/GridmanagerLock.$(USERNAME)

##--------------------------------------------------------------------
##  Various other settings that the Condor-G can use.
##--------------------------------------------------------------------

## Adjust how frequently the gridmanager checks the status of grid jobs.
## GRIDMANAGER_JOB_PROBE_INTERVAL sets the minimum time between checks
## for each job's status in seconds. For grid-types where each job's
## status must be checked individually (gt2, gt5, nordugrid, batch),
## GRIDMANAGER_JOB_PROBE_RATE sets the maximum number of status check
## requests the gridmanager will send to each remote resource per
## second. If enough jobs are submitted to a remote resource, the
## interval between checks for each job will increase so as not to
## exceed the set rate.
#GRIDMANAGER_JOB_PROBE_INTERVAL = 60
#GRIDMANAGER_JOB_PROBE_RATE = 5

## For grid-type gt2 jobs (pre-WS GRAM), limit the number of jobmanager
## processes the gridmanager will let run on the headnode. Letting too
## many jobmanagers run causes severe load on the headnode.
GRIDMANAGER_MAX_JOBMANAGERS_PER_RESOURCE = 10

## If we're talking to a Globus 2.0 resource, Condor-G will use the new
## version of the GRAM protocol. The first option is how often to check the
## proxy on the submit site of things. If the GridManager discovers a new
## proxy, it will restart itself and use the new proxy for all future
## jobs launched. In seconds,  and defaults to 10 minutes
#GRIDMANAGER_CHECKPROXY_INTERVAL = 600

## The GridManager will shut things down 3 minutes before loosing Contact
## because of an expired proxy.
## In seconds, and defaults to 3 minutes
#GRDIMANAGER_MINIMUM_PROXY_TIME  = 180

## Condor requires that each submitted job be designated to run under a
## particular "universe".
##
## If no universe is specified in the submit file, Condor must pick one
```

```
## for the job to use. By default, it chooses the "vanilla" universe.
## The default can be overridden in the config file with the DEFAULT_UNIVERSE
## setting, which is a string to insert into a job submit description if the
## job does not try and define it's own universe
##
#DEFAULT_UNIVERSE = vanilla

#
# The Cred_min_time_left is the first-pass at making sure that Condor-G
# does not submit your job without it having enough time left for the
# job to finish. For example, if you have a job that runs for 20 minutes, and
# you might spend 40 minutes in the queue, it's a bad idea to submit with less
# than an hour left before your proxy expires.
# 2 hours seemed like a reasonable default.
#
CRED_MIN_TIME_LEFT              = 120


##
## The GridMonitor allows you to submit many more jobs to a GT2 GRAM server
## than is normally possible.
#ENABLE_GRID_MONITOR = TRUE

##
## When an error occurs with the GridMonitor, how long should the
## gridmanager wait before trying to submit a new GridMonitor job?
## The default is 1 hour (3600 seconds).
#GRID_MONITOR_DISABLE_TIME = 3600

##
## The location of the wrapper for invoking
## Condor GAHP server
##
CONDOR_GAHP = $(SBIN)/condor_c-gahp
CONDOR_GAHP_WORKER = $(SBIN)/condor_c-gahp_worker_thread

##
## The Condor GAHP server has its own log.  Like the Gridmanager, the
## GAHP server is run as the User, not a Condor daemon, so all users must
## have write permission to the directory used for the logfile. Our
## suggestion is to create a directory called GridLogs in $(LOG) with
## UNIX permissions 1777 (just like /tmp )
## Another option is to use /tmp as the location of the CGAHP log.
##
#MAX_C_GAHP_LOG        = $(MAX_DEFAULT_LOG)
```

```
#C_GAHP_LOG = $(LOG)/GridLogs/CGAHPLog.$(USERNAME)
C_GAHP_LOG = /tmp/CGAHPLog.$(USERNAME)
C_GAHP_LOCK = /tmp/CGAHPLock.$(USERNAME)
C_GAHP_WORKER_THREAD_LOG = /tmp/CGAHPWorkerLog.$(USERNAME)
C_GAHP_WORKER_THREAD_LOCK = /tmp/CGAHPWorkerLock.$(USERNAME)

##
## Location of the PBS/LSF gahp and its associated binaries
##
GLITE_LOCATION = $(LIBEXEC)/glite
BATCH_GAHP = $(GLITE_LOCATION)/bin/batch_gahp

##
## The location of the wrapper for invoking the Unicore GAHP server
##
UNICORE_GAHP = $(SBIN)/unicore_gahp

##
## The location of the wrapper for invoking the NorduGrid GAHP server
##
NORDUGRID_GAHP = $(SBIN)/nordugrid_gahp

## The location of the CREAM GAHP server
CREAM_GAHP = $(SBIN)/cream_gahp

## Condor-G and CredD can use MyProxy to refresh GSI proxies which are
## about to expire.
#MYPROXY_GET_DELEGATION = /path/to/myproxy-get-delegation

## The location of the Deltacloud GAHP server
DELTACLOUD_GAHP = $(SBIN)/deltacloud_gahp

##
## EC2 (REST): Universe = Grid, Grid_Resource = ec2
##

## The location of the ec2_gahp program, required
EC2_GAHP = $(SBIN)/ec2_gahp

## Location of log files, useful for debugging, must be in
## a directory writable by any user, such as /tmp
#EC2_GAHP_DEBUG = D_FULLDEBUG
EC2_GAHP_LOG = /tmp/EC2GahpLog.$(USERNAME)

## As of this writing EC2 has a hard limit of 20 concurrently
## running instances, so a limit of 20 is imposed so the GridManager
```

```
## does not waste its time sending requests that will be rejected.
GRIDMANAGER_MAX_SUBMITTED_JOBS_PER_RESOURCE_EC2 = 20

##
##--------------------------------------------------------------------
##  condor_credd credential management daemon
##--------------------------------------------------------------------
##  Where is the CredD binary installed?
CREDD                           = $(SBIN)/condor_credd

##  When the credd starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need an additional "-n host:port" command line option.  This
##  feature can be turned off by commenting out this setting.
CREDD_ADDRESS_FILE    = $(LOG)/.credd_address

##  Specify a remote credd server here,
#CREDD_HOST  = $(CONDOR_HOST):$(CREDD_PORT)

## CredD startup arguments
## Start the CredD on a well-known port.  Uncomment to to simplify
## connecting to a remote CredD.  Note: that this interface may change
## in a future release.
CREDD_PORT                      = 9620
CREDD_ARGS                      = -p $(CREDD_PORT) -f

## CredD daemon debugging log
CREDD_LOG                   = $(LOG)/CredLog
CREDD_DEBUG                     = D_FULLDEBUG
#MAX_CREDD_LOG                  = $(MAX_DEFAULT_LOG)

## The credential owner submits the credential.  This list specifies
## other user who are also permitted to see all credentials.  Defaults
## to root on UNIX systems, and Administrator on Windows systems.
#CRED_SUPER_USERS =

## Credential storage location.  This directory must exist
## prior to starting condor_credd.  It is highly recommended to
## restrict access permissions to _only_ the directory owner.
CRED_STORE_DIR = $(LOCAL_DIR)/cred_dir

## Index file path of saved credentials.
## This file will be automatically created if it does not exist.
#CRED_INDEX_FILE = $(CRED_STORE_DIR/cred-index

## condor_credd  will attempt to refresh credentials when their
```

```
## remaining lifespan is less than this value.  Units = seconds.
#DEFAULT_CRED_EXPIRE_THRESHOLD = 3600

## condor-credd periodically checks remaining lifespan of stored
## credentials, at this interval.
#CRED_CHECK_INTERVAL = 60


##
##--------------------------------------------------------------------
##  VM Universe Parameters
##--------------------------------------------------------------------
## Where is the Condor VM-GAHP installed? (Required)
VM_GAHP_SERVER = $(SBIN)/condor_vm-gahp

## If the VM-GAHP is to have its own log, define
## the location of log file.
##
## Optionally, if you do NOT define VM_GAHP_LOG, logs of VM-GAHP will
## be stored in the starter's log file.
## However, on Windows machine you must always define VM_GAHP_LOG.
#
VM_GAHP_LOG      = $(LOG)/VMGahpLog
#MAX_VM_GAHP_LOG     = $(MAX_DEFAULT_LOG)
#VM_GAHP_DEBUG = D_FULLDEBUG

## What kind of virtual machine program will be used for
## the VM universe?
## The three primary options are KVM, Xen and VMware.  (Required: no default)
#VM_TYPE = kvm

## How much memory can be used for the VM universe? (Required)
## This value is the maximum amount of memory that can be used by the
## virtual machine program.
#VM_MEMORY = 128

## Want to support networking for VM universe?
## Default value is FALSE
#VM_NETWORKING = FALSE

## What kind of networking types are supported?
##
## If you set VM_NETWORKING to TRUE, you must define this parameter.
## VM_NETWORKING_TYPE = nat
## VM_NETWORKING_TYPE = bridge
## VM_NETWORKING_TYPE = nat, bridge
```

```
##
## If multiple networking types are defined, you may define
## VM_NETWORKING_DEFAULT_TYPE for default networking type.
## Otherwise, nat is used for default networking type.
## VM_NETWORKING_DEFAULT_TYPE = nat
#VM_NETWORKING_DEFAULT_TYPE = nat
#VM_NETWORKING_TYPE = nat

## In default, the number of possible virtual machines is same as
## NUM_CPUS.
## Since too many virtual machines can cause the system to be too slow
## and lead to unexpected problems, limit the number of running
## virtual machines on this machine with
#VM_MAX_NUMBER = 2

## When a VM universe job is started, a status command is sent
## to the VM-GAHP to see if the job is finished.
## If the interval between checks is too short, it will consume
## too much of the CPU. If the VM-GAHP fails to get status 5 times in a row,
## an error will be reported to startd, and then startd will check
## the availability of VM universe.
## Default value is 60 seconds and minimum value is 30 seconds
#VM_STATUS_INTERVAL = 60

## How long will we wait for a request sent to the VM-GAHP to be completed?
## If a request is not completed within the timeout, an error will be reported
## to the startd, and then the startd will check
## the availability of vm universe.  Default value is 5 minutes.
#VM_GAHP_REQ_TIMEOUT = 300

## When VMware or Xen causes an error, the startd will disable the
## VM universe.  However, because some errors are just transient,
## we will test one more
## whether vm universe is still unavailable after some time.
## In default, startd will recheck vm universe after 10 minutes.
## If the test also fails, vm universe will be disabled.
#VM_RECHECK_INTERVAL = 600

## Usually, when we suspend a VM, the memory being used by the VM
## will be saved into a file and then freed.
## However, when we use soft suspend, neither saving nor memory freeing
## will occur.
## For VMware, we send SIGSTOP to a process for VM in order to
## stop the VM temporarily and send SIGCONT to resume the VM.
## For Xen, we pause CPU. Pausing CPU does not save the memory of VM
## into a file. It only stops the execution of a VM temporarily.
```

```
#VM_SOFT_SUSPEND = TRUE

## If Condor runs as root and a job comes from a different UID domain,
## Condor generally uses "nobody", unless SLOTx_USER is defined.
## If "VM_UNIV_NOBODY_USER" is defined, a VM universe job will run
## as the user defined in "VM_UNIV_NOBODY_USER" instead of "nobody".
##
## Notice: In VMware VM universe, "nobody" can not create a VMware VM.
## So we need to define "VM_UNIV_NOBODY_USER" with a regular user.
## For VMware, the user defined in "VM_UNIV_NOBODY_USER" must have a
## home directory.  So SOFT_UID_DOMAIN does not work for VMware VM universe job.
## If neither "VM_UNIV_NOBODY_USER" nor "SLOTx_VMUSER"/"SLOTx_USER" is
defined,
## VMware VM universe job will run as "condor" instead of "nobody".
## As a result, the preference of local users for a VMware VM universe job
## which comes from the different UID domain is
## "VM_UNIV_NOBODY_USER" -> "SLOTx_VMUSER" -> "SLOTx_USER" -> "condor".
#VM_UNIV_NOBODY_USER = login name of a user who has home directory

## If Condor runs as root and "ALWAYS_VM_UNIV_USE_NOBODY" is set to TRUE,
## all VM universe jobs will run as a user defined in "VM_UNIV_NOBODY_USER".
#ALWAYS_VM_UNIV_USE_NOBODY = FALSE

##--------------------------------------------------------------------
##  VM Universe Parameters Specific to VMware
##--------------------------------------------------------------------

## Where is perl program? (Required)
VMWARE_PERL = perl

## Where is the Condor script program to control VMware? (Required)
VMWARE_SCRIPT = $(SBIN)/condor_vm_vmware

## Networking parameters for VMware
##
## What kind of VMware networking is used?
##
## If multiple networking types are defined, you may specify different
## parameters for each networking type.
##
## Examples
## (e.g.) VMWARE_NAT_NETWORKING_TYPE = nat
## (e.g.) VMWARE_BRIDGE_NETWORKING_TYPE = bridged
##
##  If there is no parameter for specific networking type, VMWARE_NETWORKING_TYPE
is used.
```

```
##
#VMWARE_NAT_NETWORKING_TYPE = nat
#VMWARE_BRIDGE_NETWORKING_TYPE = bridged
VMWARE_NETWORKING_TYPE = nat

## The contents of this file will be inserted into the .vmx file of
## the VMware virtual machine before Condor starts it.
#VMWARE_LOCAL_SETTINGS_FILE = /path/to/file

##--------------------------------------------------------------------
##  VM Universe Parameters common to libvirt controlled vm's (xen & kvm)
##--------------------------------------------------------------------

## Networking parameters for Xen & KVM
##
## This is the path to the XML helper command; the libvirt_simple_script.awk
## script just reproduces what Condor already does for the kvm/xen VM
## universe
LIBVIRT_XML_SCRIPT = $(LIBEXEC)/libvirt_simple_script.awk

## This is the optional debugging output file for the xml helper
## script.  Scripts that need to output debugging messages should
## write them to the file specified by this argument, which will be
## passed as the second command line argument when the script is
## executed

#LIBVRT_XML_SCRIPT_ARGS = /dev/stderr

##--------------------------------------------------------------------
##  VM Universe Parameters Specific to Xen
##--------------------------------------------------------------------

##  Where is bootloader for Xen domainU? (Required)
##
##  The bootloader will be used in the case that a kernel image includes
##  a disk image
#XEN_BOOTLOADER = /usr/bin/pygrub


##
##--------------------------------------------------------------------
##  KBDD - keyboard activity detection daemon
##--------------------------------------------------------------------
##  When the KBDD starts up, it can place it's address (IP and port)
##  into a file.  This way, tools running on the local machine don't
##  need an additional "-n host:port" command line option.  This
```

```
##  feature can be turned off by commenting out this setting.
KBDD_ADDRESS_FILE = $(LOG)/.kbdd_address

##
##--------------------------------------------------------------------
##  condor_ssh_to_job
##--------------------------------------------------------------------
# NOTE: condor_ssh_to_job is not supported under Windows.

# Tell the starter (execute side) whether to allow the job owner or
# queue super user on the schedd from which the job was submitted to
# use condor_ssh_to_job to access the job interactively (e.g. for
# debugging).  TARGET is the job; MY is the machine.
#ENABLE_SSH_TO_JOB = true

# Tell the schedd (submit side) whether to allow the job owner or
# queue super user to use condor_ssh_to_job to access the job
# interactively (e.g. for debugging).  MY is the job; TARGET is not
# defined.
#SCHEDD_ENABLE_SSH_TO_JOB = true

# Command condor_ssh_to_job should use to invoke the ssh client.
# %h --> remote host
# %i --> ssh key file
# %k --> known hosts file
# %u --> remote user
# %x --> proxy command
# %% --> %
#SSH_TO_JOB_SSH_CMD = "ssh -oUser=%u -oIdentityFile=%i -
oStrictHostKeyChecking=yes -oUserKnownHostsFile=%k -oGlobalKnownHostsFile=%k -
oProxyCommand=%x %h"

# Additional ssh clients may be configured.  They all have the same
# default as ssh, except for scp, which omits the %h:
#SSH_TO_JOB_SCP_CMD = "scp -oUser=%u -oIdentityFile=%i -
oStrictHostKeyChecking=yes -oUserKnownHostsFile=%k -oGlobalKnownHostsFile=%k -
oProxyCommand=%x"

# Path to sshd
#SSH_TO_JOB_SSHD = /usr/sbin/sshd

# Arguments the starter should use to invoke sshd in inetd mode.
# %f --> sshd config file
# %% --> %
#SSH_TO_JOB_SSHD_ARGS = "-i -e -f %f"
```

```
# sshd configuration template used by condor_ssh_to_job_sshd_setup.
#SSH_TO_JOB_SSHD_CONFIG_TEMPLATE =
$(LIB)/condor_ssh_to_job_sshd_config_template

# Path to ssh-keygen
#SSH_TO_JOB_SSH_KEYGEN = /usr/bin/ssh-keygen

# Arguments to ssh-keygen
# %f --> key file to generate
# %% --> %
#SSH_TO_JOB_SSH_KEYGEN_ARGS = "-N " -C " -q -f %f -t rsa"

##########################################################################
##
##  Condor HDFS
##
##  This is the default local configuration file for configuring Condor
##  daemon responsible for running services related to hadoop
##  distributed storage system.  You should copy this file to the
##  appropriate location and customize it for your needs.
##
##  Unless otherwise specified, settings that are commented out show
##  the defaults that are used if you don't define a value.  Settings
##  that are defined here MUST BE DEFINED since they have no default
##  value.
##
##########################################################################

##########################################################################
## FOLLOWING MUST BE CHANGED
##########################################################################

## The location for hadoop installation directory. The default location
## is under 'libexec' directory. The directory pointed by HDFS_HOME
## should contain a lib folder that contains all the required Jars necessary
## to run HDFS name and data nodes.
#HDFS_HOME = $(RELEASE_DIR)/libexec/hdfs

## The host and port for hadoop's name node. If this machine is the
## name node (see HDFS_SERVICES) then the specified port will be used
## to run name node.
#HDFS_NAMENODE = hdfs://example.com:9000
#HDFS_NAMENODE_WEB = example.com:8000

#HDFS_BACKUPNODE = hdfs://example.com:50100
#HDFS_BACKUPNODE_WEB = example.com:50105
```

```
## You need to pick one machine as name node by setting this parameter
## to HDFS_NAMENODE. The remaining machines in a storage cluster will
## act as data nodes (HDFS_DATANODE).
#HDFS_NODETYPE = HDFS_DATANODE

## If machine is selected to be NameNode then by a role should defined.
## If it selected to be DataNode then this parameter is ignored.
## Available options:
## ACTIVE: Active NameNode role (default value)
## BACKUP: Always synchronized with the active NameNode state, thus
##         creating a backup of the namespace. Currently the NameNode
##         supports one Backup node at a time.
## CHECKPOINT: Periodically creates checkpoints of the namespace.
#HDFS_NAMENODE_ROLE = ACTIVE

## The two set of directories that are required by HDFS are for name
## node (HDFS_NAMENODE_DIR) and data node (HDFS_DATANODE_DIR). The
## directory for name node is only required for a machine running
## name node service and  is used to store critical meta data for
## files. The data node needs its directory to store file blocks and
## their replicas.
#HDFS_NAMENODE_DIR = /tmp/hadoop_name
#HDFS_DATANODE_DIR = /scratch/tmp/hadoop_data

## Unlike name node address settings (HDFS_NAMENODE), that needs to be
## well known across the storage cluster, data node can run on any
## arbitrary port of given host.
#HDFS_DATANODE_ADDRESS = 0.0.0.0:0

##########################################################################
## OPTIONAL
##########################################################################

## Sets the log4j debug level. All the emitted debug output from HDFS
## will go in 'hdfs.log' under $(LOG) directory.
#HDFS_LOG4J=DEBUG

## The access to HDFS services both name node and data node can be
## restricted by specifying IP/host based filters. By default settings
## from ALLOW_READ/ALLOW_WRITE and DENY_READ/DENY_WRITE
## are used to specify allow and deny list. The below two parameters can
## be used to override these settings. Read the Condor manual for
## specification of these filters.
## WARN: HDFS doesn't make any distinction between read or write based connection.
#HDFS_ALLOW=*
```

```
#HDFS_DENY=*

#Fully qualified name for Name node and Datanode class.
#HDFS_NAMENODE_CLASS=org.apache.hadoop.hdfs.server.namenode.NameNode
#HDFS_DATANODE_CLASS=org.apache.hadoop.hdfs.server.datanode.DataNode
#HDFS_DFSADMIN_CLASS=org.apache.hadoop.hdfs.tools.DFSAdmin

## In case an old name for hdfs configuration files is required.
#HDFS_SITE_FILE = hdfs-site.xml


##
##-------------------------------------------------------------------
##  file transfer plugin defaults
##-------------------------------------------------------------------
      FILETRANSFER_PLUGINS = $(LIBEXEC)/curl_plugin, $(LIBEXEC)/data_plugin
```

## Appendix B. Condor Submit File

```
# RNA Secondary Structure Prediction - HTCondor Submit File
Executable = /export/HTCondorScripts/Sequence_Chunks/RNAPredExec.pl
Universe = vanilla
Output =
/export/HTCondorScripts/Sequence_Chunks/001[105,165]BBV.RNA1_L4G1M0C120Centered.
out

requirements = ( Arch=="X86_64") && ( OpSys=="LINUX" )
should_transfer_files = NO
transfer_executable = FALSE
notification = never
arguments = ACACCGACGAGCCGACGACGTUTUTUTUACU

Queue
```

## Appendix C. Submit File Generator Script based on pknotsRG

```
#!/usr/bin/perl
use strict;

#Generate file containing the filenames
system("ls -l ../../HTCondor/ | egrep '^d' | awk '{ print \$9 }' > directories.txt");

my ($fileLocation) = "";
my($directoryList) = "directories.txt";

open (FILE,$directoryList) or die $!;
my @directoryContent = <FILE>;
close(FILE);

for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
{
  my ($directoryFileName) = $directoryContent[$i];
  chomp($directoryFileName);
  system("rm -rf ".$directoryFileName);
}

for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
```

```
{
  my ($directoryFileName) = $directoryContent[$i];

  chomp($directoryFileName);

  # Recreate folders
  $fileLocation =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_F
asta";

  #Generate file containing the filenames
  my ($file) = "filenames.txt";

  if ( !-d $fileLocation)
  {
    system("mkdir -p ".$fileLocation);
  }

  #system("cp -r ../../HTCondor/".$directoryFileName."/*Regular* ".$fileLocation);
  system("cp -r ../../HTCondor/".$directoryFileName."/*Centered* ".$fileLocation);
  my $dir = qx{ls $fileLocation};

  open(my $OUT, ">", 'filenames.txt') or die $!;
  print $OUT $dir;
  close $OUT;

  my $seqnum = 0;
  my @seq;
  my @seqheader;

  open(FILE, $file) or die $!;
  my (@filenames) = <FILE>;
  close(FILE);

  my($counter)=1;

  my ($dirToCreate) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_O
utput";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }
```

```
  my($dirToCreate2) = "/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_OutReference";

  system("rm -rf $dirToCreate2");

  if ( !-d $dirToCreate2)
  {
    system("mkdir -p ".$dirToCreate2);
  }

  $dirToCreate = "/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_SubmitFiles";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }

  for(my $i=0;$i<=scalar(@filenames)-1;$i++)
  {
    my ($filename) = $filenames[$i];
    my $seqs = &get_fasta("/export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Fasta/".$filename);

    foreach my $key (sort keys %$seqs)
    {
      open (OUTREFERENCE, ">> $dirToCreate2/".remove_extension($filename));
        print OUTREFERENCE $key."\n";
      close (OUTREFERENCE);

      my ($submitfile) = $key.".submit";
      my ($description) = "Submit file for RNA Prediction using Segmentation";
        my ($executable) = "/export/HTCondorScripts/Sequence_Chunks/RNAPredExec_1.pl";

      open (CONDORFILE, "> $dirToCreate/$submitfile");
        print CONDORFILE "# ".$description."\n";
        print CONDORFILE "Executable = ".$executable."\n";
        print CONDORFILE "Universe   = vanilla"."\n";
        print CONDORFILE "Output = /export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Output
```

```
/"."$key".".out\n\n";
          print CONDORFILE "should_transfer_files = NO\n";
          print CONDORFILE "transfer_executable = FALSE\n\n";
          print CONDORFILE "arguments  = ".$$seqs{$key}."\n";
          print CONDORFILE "Queue\n\n";
        close (CONDORFILE);

        system("condor_submit $dirToCreate/".$submitfile);
     }
   }
}

# Remove extension from the filename
sub remove_extension {
    my $filename = shift @_;

    $filename =~ s/
              (.)            # matches any character
              \.             # the literal dot starting an extension
              [^.]+            # one or more NON-dots
              $              # end of the string
        /$1/x;

    return $filename;
}

# Remove title and get just the sequence
sub get_fasta{
  my $filename = $_[0];

  open(FILE, $filename) or die("Cannot open FASTA file.\n");
  my %seqs;
  my $header;
  my $first = 0;
  my @lines= <FILE>;
  foreach my $line(@lines){
  chomp($line);

  if ($line =~ /^>/){
    $header = $line;
    $header =~ s/^>//;
    $header =~ s/\s.*//;
    if ($first == 0){
      $first = 1;
    }
    next;
```

```
  }
  if ($first == 0){ die("Not FASTA file.\n"); }
    $seqs{$header} = $seqs{$header}.$line;
}
  close(FILE);
  return \%seqs;

          }
```

## Appendix D. RNA Prediction Script using pknotsRG

```
#!/usr/bin/perl -w
#use strict;
#use warnings;

$sequence=$ARGV[0];

##HTCondor with Non-Pseudoknots using pknotsRG
#system("/applications/pknotsRG-1.3/src/pknotsRG -k 0 ".$sequence);

##HTCondor with Pseudoknots using pknotsRG
system("/applications/pknotsRG-1.3/src/pknotsRG ".$sequence);
```

## Appendix E. Sequential RNA Prediction Script

```
#!/usr/bin/perl
use strict;

#Generate file containing the filenames
system("ls -l ../../HTCondor/ | egrep '^d' | awk '{ print \$9 }' > directories.txt");

my ($fileLocation) = "";
my($directoryList) = "directories.txt";

open (FILE,$directoryList) or die $!;
my @directoryContent = <FILE>;
close(FILE);

system("date >> seqtime.txt");
system("\n\n >> seqtime.txt");
```

```
for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
{
  my ($directoryFileName) = $directoryContent[$i];
  chomp($directoryFileName);
  system("rm -rf ".$directoryFileName);
}

for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
{
  my ($directoryFileName) = $directoryContent[$i];

  chomp($directoryFileName);

  # Recreate folders
  $fileLocation                                                                                                    =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_F
asta";

  #Generate file containing the filenames
  my ($file) = "filenames.txt";

  if ( !-d $fileLocation)
  {
    system("mkdir -p ".$fileLocation);
  }

  #system("cp -r ../../HTCondor/".$directoryFileName."/*Regular* ".$fileLocation);
  system("cp -r ../../HTCondor/".$directoryFileName."/*Centered* ".$fileLocation);
  my $dir = qx{ls $fileLocation};

  open(my $OUT, ">", 'filenames.txt') or die $!;
  print $OUT $dir;
  close $OUT;

  my $seqnum = 0;
  my @seq;
  my @seqheader;

  open(FILE, $file) or die $!;
  my (@filenames) = <FILE>;
  close(FILE);

  my($counter)=1;
```

```
  my ($dirToCreate) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_
Output";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }

  my($dirToCreate2) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_
OutReference";

  system("rm -rf $dirToCreate2");

  if ( !-d $dirToCreate2)
  {
    system("mkdir -p ".$dirToCreate2);
  }

  $dirToCreate =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_S
ubmitFiles";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }

  for(my $i=0;$i<=scalar(@filenames)-1;$i++)
  {
    my ($filename) = $filenames[$i];
    my $seqs =
&get_fasta("/export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileNa
me"."_Fasta/".$filename);

   foreach my $key (sort keys %$seqs)
   {
      open (OUTREFERENCE, ">> $dirToCreate2/".remove_extension($filename));
```

```
        print OUTREFERENCE $key."\n";
      close (OUTREFERENCE);

      ##pknotsRG WITH NO PSEUDOKNOTS
      #system("/applications/pknotsRG-1.3/src/pknotsRG -k 0 ".$$seqs{$key}."> /export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Output/"."$key".".out.seq");
      ##pknotsRG WITH PSEUDOKNOTS
      system("/applications/pknotsRG-1.3/src/pknotsRG ".$$seqs{$key}."> /export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Output/"."$key".".out.seq");
    }
  }
}

system("date >> seqtime.txt");


# Remove extension from the filename
sub remove_extension {
   my $filename = shift @_;

   $filename =~ s/
            (.)         # matches any character
            \.          # the literal dot starting an extension
            [^.]+         # one or more NON-dots
            $           # end of the string
      /$1/x;

   return $filename;
}

# Remove title and get just the sequence
sub get_fasta{
  my $filename = $_[0];

  open(FILE, $filename) or die("Cannot open FASTA file.\n");
  my %seqs;
  my $header;
  my $first = 0;
  my @lines= <FILE>;
  foreach my $line(@lines){
  chomp($line);
```

```
  if ($line =~ /^>/){
    $header = $line;
    $header =~ s/^>//;
    $header =~ s/\s.*//;

    if ($first == 0){
      $first = 1;
    }
    next;
  }
  if ($first == 0){ die("Not FASTA file.\n"); }
    $seqs{$header} = $seqs{$header}.$line;}
  close(FILE);
  return \%seqs;
}
```

## Appendix F. Accuracy Perl Script

```
#!/usr/bin/perl -w
#use strict;
#use warnings;

sub calculateAccuracy{

#my @structOrig=split('',$Original);
#my @structPred=split('',$Predicted);
my ($Original, $Predicted) = @_;

my @structOrig=split('',$Original);
my @structPred=split('',$Predicted);

#print "Original: $Original\n";
#print "Predicted: $Predicted\n";

#TOTAL LENGTH OF THE STRUCTURE
my $totalLength= scalar(@structOrig);

#ARRAY CONTAINING PARENTHESIS AND DOTS POSITIONS
my @closeRightOrig=();
my @openLeftOrig=();
my @closeRightPred=();
my @openLeftPred=();
```

```
my @dotOrig=();
my @dotPred=();

my $countLeftOrig=0;
my $countLeftPred=0;
my $countRightOrig=0;
my $countRightPred=0;
my $countDotOrig=0;
my $countDotPred=0;

#CONSTRUCT OPEN LEFT ORIGINAL AND PREDICTED
for($i=0;$i<$totalLength;$i++)
{
  if($structOrig[$i] eq '(')
  {
    $openLeftOrig[$countLeftOrig] = $i+1;
    $countLeftOrig = $countLeftOrig + 1;
  }

  if($structPred[$i] eq '(')
  {
    $openLeftPred[$countLeftPred] = $i+1;
    $countLeftPred = $countLeftPred + 1;
  }
}

my $totalOpenLeftOrigLength = scalar(@openLeftOrig);
my $totalOpenLeftPredLength = scalar(@openLeftPred);

#CONSTRUCT CLOSE RIGHT ORIGINAL
my @structCPOrig=split('',$Original);
my $flag;
for($i=$totalOpenLeftOrigLength-1;$i>=0;$i--)
{
  $flag=1;
  $j=$openLeftOrig[$i];
  while(($j<$totalLength) && ($flag==1))
  {
    if($structCPOrig[$j] eq ')')
    {
      #print "J VALUE: $j\n";
      my $test=join('',@structCPOrig);
      substr($test,$j,1) ='.';
      @structCPOrig=split('',$test);
```

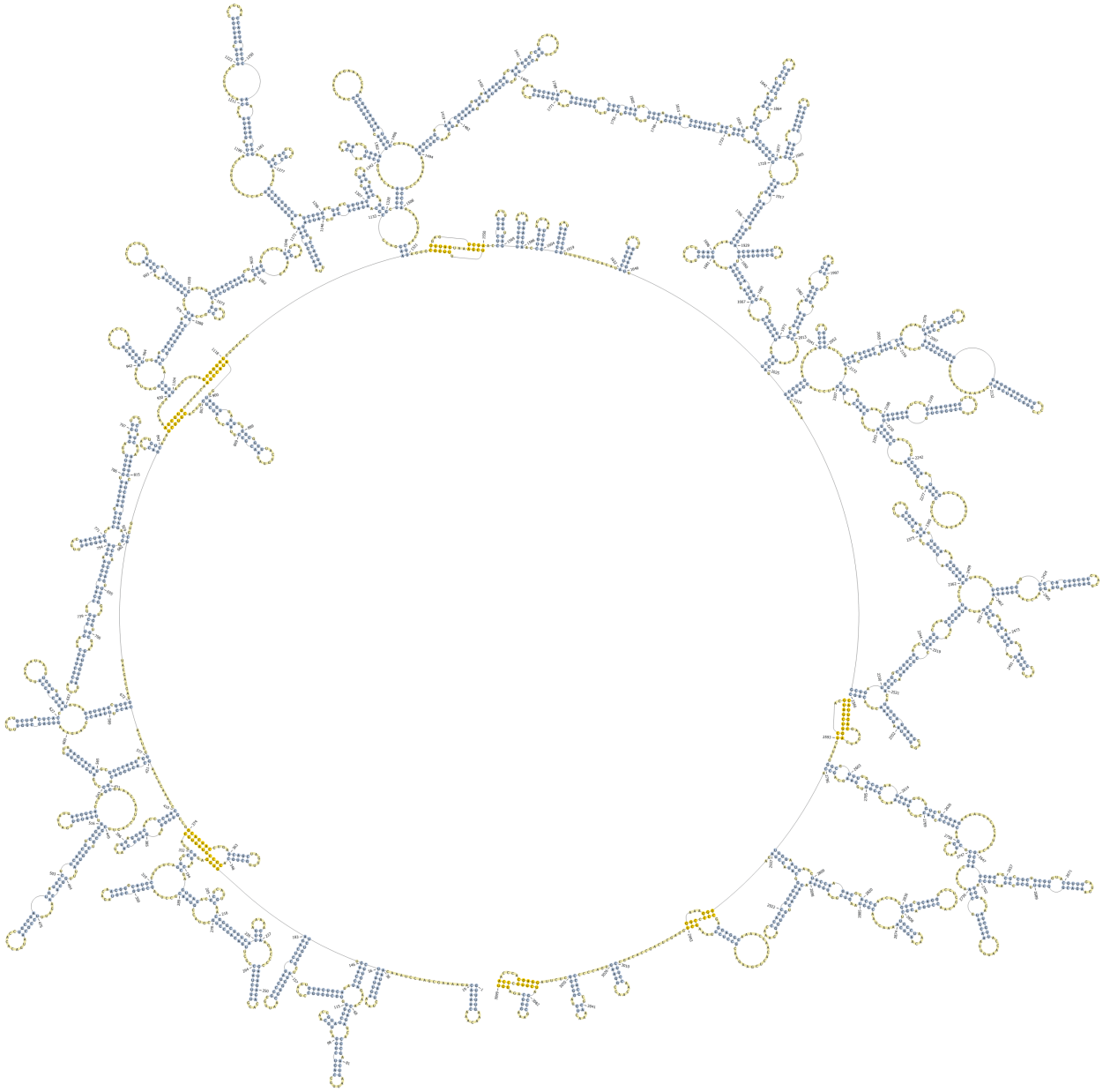
```
      #print "TEST: $test\n";
      $closeRightOrig[$i] = $j+1;
      $flag=0;
    }
    $j=$j+1;
  }
}

#for(my $i=0;$i<$totalOpenLeftOrigLength;$i++)
#{
#   print "Content Original ( $i: ".$openLeftOrig[$i]."\n";
#   print "Content Original ) $i: ".$closeRightOrig[$i]."\n";
#}

#CONSTRUCT CLOSE RIGHT PREDICTED
my @structCPPred=split('',$Predicted);

for($i=$totalOpenLeftPredLength-1;$i>=0;$i--)
{
  $flag=1;
  $j=$openLeftPred[$i];
  while(($j<$totalLength) && ($flag==1))
  {
    if($structCPPred[$j] eq ')')
    {
      #print "J VALUE: $j\n";
      my $test=join('',@structCPPred);
      substr($test,$j,1) ='.';
      @structCPPred=split('',$test);
      #print "TEST: $test\n";
      $closeRightPred[$i] = $j+1;
      $flag=0;
    }
    $j=$j+1;
  }
}

#for(my $i=0;$i<$totalOpenLeftPredLength;$i++)
#{
#   print "Content Predicted ( $i: ".$openLeftPred[$i]."\n";
#   print "Content Original ) $i: ".$closeRightPred[$i]."\n";
#}

#COUNT MATCH PARENTHESIS STRUCTURES
```

```
my $countParenthesisStructure = 0;

for(my $i=0;$i<$totalOpenLeftOrigLength;$i++)
{
  for(my $j=0;$j<$totalOpenLeftPredLength;$j++)
  {
     #print "[$i]".$openLeftOrig[$i]." - [$j] ".$closeRightOrig[$i]."\n";
     #print "[$i]".$openLeftPred[$j]." - [$j] ".$closeRightPred[$j]."\n\n";

     if ($openLeftOrig[$i] eq $openLeftPred[$j])
     {
       if ($closeRightOrig[$i] eq $closeRightPred[$j])
       {
         #print "Original: ( at ".$openLeftOrig[$i]." closes at ".$closeRightOrig[$i]." ) \n";
         #print "Predicted ( at ".$openLeftPred[$j]." closes at ".$closeRightPred[$j]." ) \n";
             $countParenthesisStructure = $countParenthesisStructure + 1;
       }
     }
  }
}

#COUNT MATCH DOT STRUCTURES
my $countDotStructure = 0;
for(my $i=0;$i<$totalLength;$i++)
{
   if (($structOrig[$i] eq '.') && ($structPred[$i] eq '.'))
   {
     #print "Original: . at ".$structOrig[$i]." and Predicted: ".$structPred[$i]." ) \n";
     $countDotStructure = $countDotStructure + 1;
   }
}

#CALCULATE ACCURACY
my $Accuracy = ((2 * $countParenthesisStructure) + (1 * $countDotStructure))/$totalLength;
$AccuracyPercentage = $Accuracy*100;
$AccuracyPercentage = sprintf("%.2f", $AccuracyPercentage);
#print "Accuracy is: $Accuracy\n";
#print "Percentage is: $AccuracyPercentage\n\n";
return $AccuracyPercentage;
}

#MAIN
my ($Orig) = $ARGV[0];
chomp ($Orig);
```

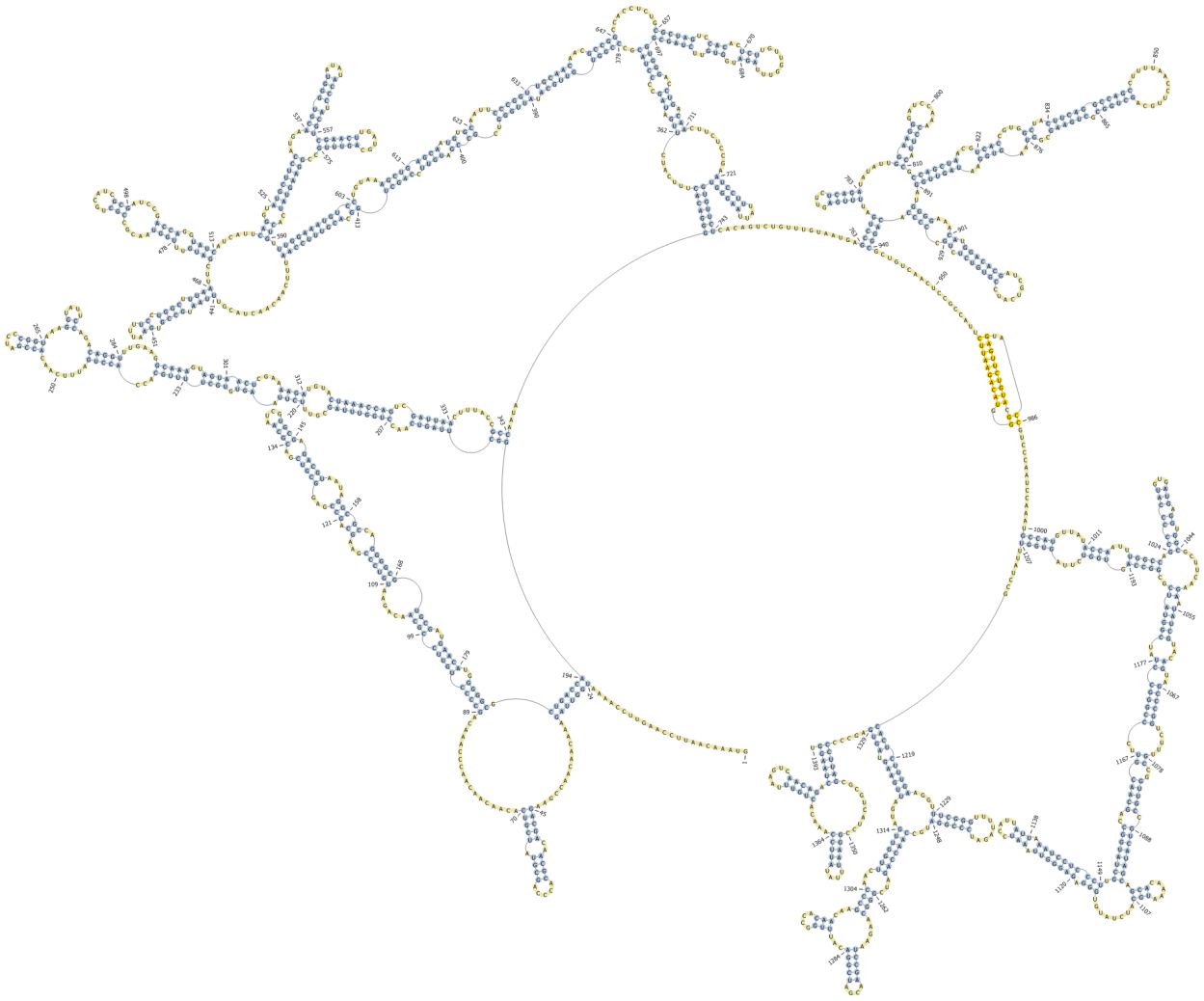
```
my ($Pred) = $ARGV[1];
chomp ($Pred);

print calculateAccuracy($Orig, $Pred)."\n";
```

# Appendix G. Visualization of Predicted Nodavirus RNA Secondary Structures.

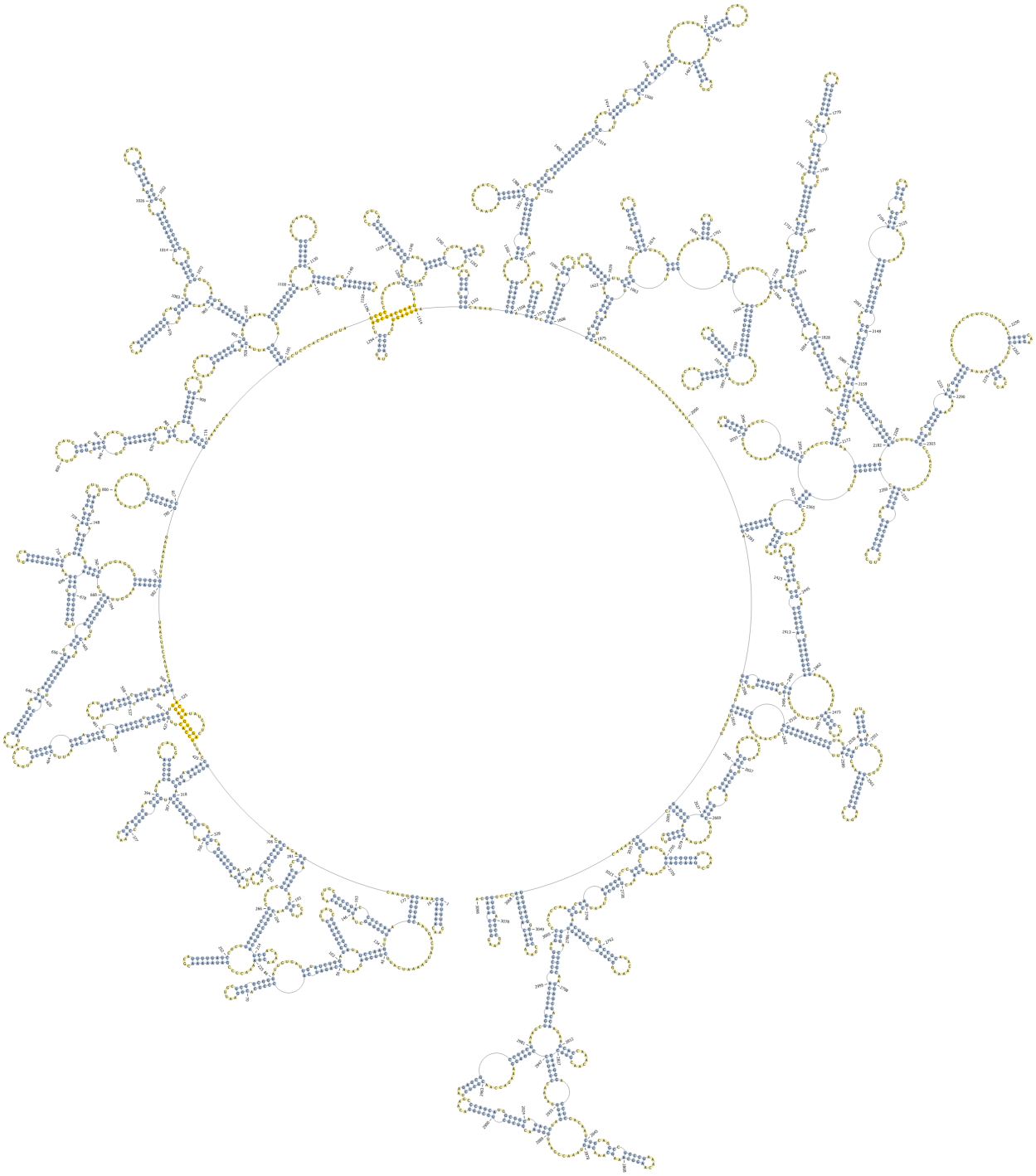
## a. BBV.RNA1_L3G0M0C400Centered



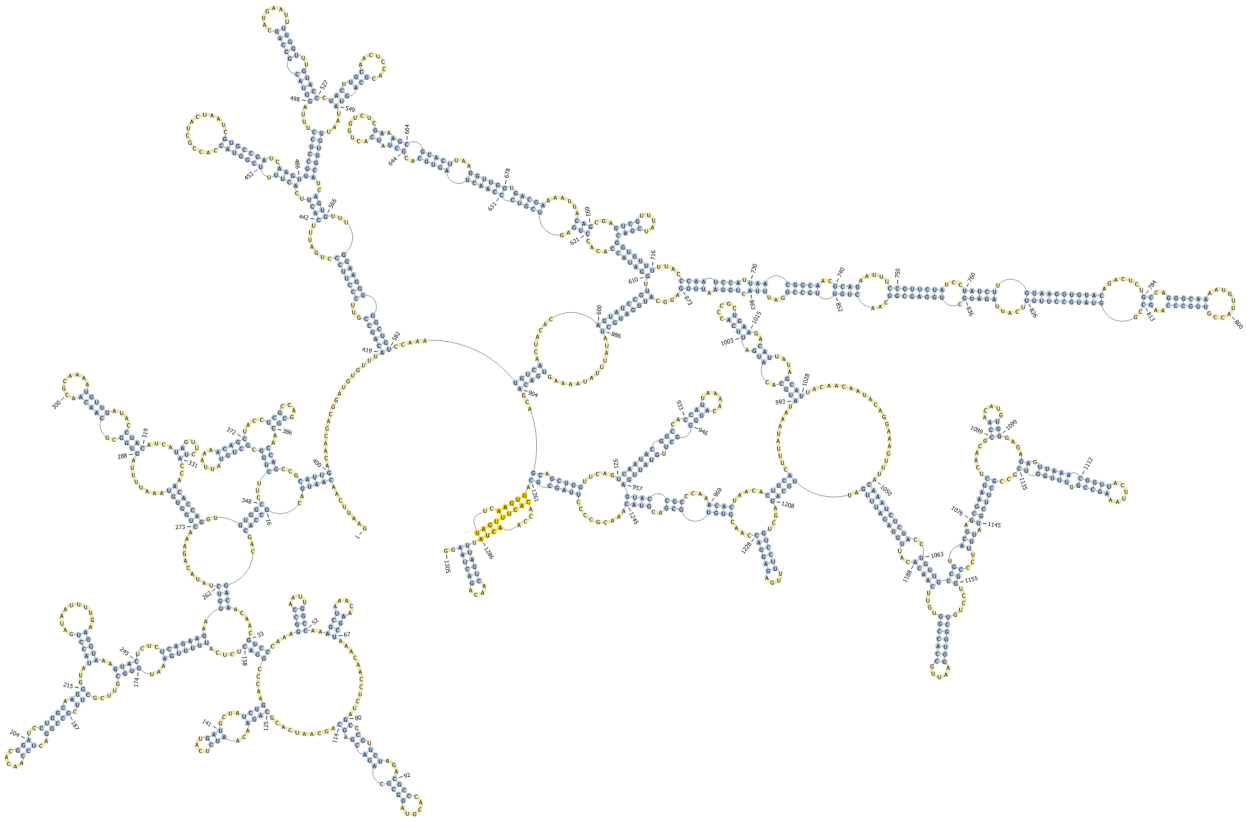
**b. BBV.RNA2_L4G1M0C400Centered**



**c. BoV.RNA1_L4G5M0C400Centered**



**d. BoV.RNA2_L4G5M0C400Centered**



## Appendix H. Submit File Generator Script based on RNAstructure

```
#!/usr/bin/perl
use strict;

#Generate file containing the filenames
system("ls -l ../../HTCondor/ | egrep '^d' | awk '{ print \$9 }' > directories.txt");

my ($fileLocation) = "";
my($directoryList) = "directories.txt";

open (FILE,$directoryList) or die $!;
my @directoryContent = <FILE>;
```

```
close(FILE);

system("date >> seqtime.txt");
system("\n\n >> seqtime.txt");

for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
{
  my ($directoryFileName) = $directoryContent[$i];
  chomp($directoryFileName);
  system("rm -rf ".$directoryFileName);
}

for(my $i=0;$i<=scalar(@directoryContent)-1;$i++)
{
  my ($directoryFileName) = $directoryContent[$i];

  chomp($directoryFileName);

  # Recreate folders
  $fileLocation                                                                                                    =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_F
asta";
  ###system("rm -rf ".$fileLocation);

  #Generate file containing the filenames
  my ($file) = "filenames.txt";

  if ( !-d $fileLocation)
  {
    system("mkdir -p ".$fileLocation);
  }

  #system("cp -r ../../HTCondor/".$directoryFileName."/*Regular* ".$fileLocation);
  system("cp -r ../../HTCondor/".$directoryFileName."/*Centered* ".$fileLocation);
  my $dir = qx{ls $fileLocation};

  open(my $OUT, ">", 'filenames.txt') or die $!;
  print $OUT $dir;
  close $OUT;

  my $seqnum = 0;
  my @seq;
  my @seqheader;
```

```
  open(FILE, $file) or die $!;
  my (@filenames) = <FILE>;
  close(FILE);

  my($counter)=1;

  my ($dirToCreate) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_
Output";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }

  my($dirToCreate2) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_
OutReference";

  system("rm -rf $dirToCreate2");

  if ( !-d $dirToCreate2)
  {
    system("mkdir -p ".$dirToCreate2);
  }

  $dirToCreate =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_S
ubmitFiles";

  system("rm -rf $dirToCreate");

  if ( !-d $dirToCreate)
  {
    system("mkdir -p ".$dirToCreate);
  }

  for(my $i=0;$i<=scalar(@filenames)-1;$i++)
  {
    #print $counter++."\n";
    my ($filename) = $filenames[$i];
    my $seqs =
```

```
&get_fasta("/export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Fasta/".$filename);

    foreach my $key (sort keys %$seqs)
    {
          ####print "$dirToCreate2/".remove_extension($filename)."\n";
          ####print "KEY: $key\n";
       open (OUTREFERENCE, ">> $dirToCreate2/".remove_extension($filename));
         print OUTREFERENCE $key."\n";
       close (OUTREFERENCE);

       ##pknotsRG WITH NO PSEUDOKNOTS
       #system("/applications/pknotsRG-1.3/src/pknotsRG -k 0 ".$$seqs{$key}."> /export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Output/"."$key".".out.seq");
       ##pknotsRG WITH PSEUDOKNOTS
       system("/applications/pknotsRG-1.3/src/pknotsRG ".$$seqs{$key}."> /export/HTCondorScripts/Sequence_Chunks/$directoryFileName/$directoryFileName"."_Output/"."$key".".out.seq");
    }

    #Remove the filenames.txt file
    #system("rm -rf ".$file);
  }
}

system("date >> seqtime.txt");


# Remove extension from the filename
sub remove_extension {
   my $filename = shift @_;

   $filename =~ s/
            (.)          # matches any character
            \.           # the literal dot starting an extension
            [^.]+         # one or more NON-dots
            $            # end of the string
      /$1/x;

   return $filename;
}

# Remove title and get just the sequence
```

```
sub get_fasta{
  my $filename = $_[0];

  open(FILE, $filename) or die("Cannot open FASTA file.\n");
  my %seqs;
  my $header;
  my $first = 0;
  my @lines= <FILE>;
  foreach my $line(@lines){
  chomp($line);

  if ($line =~ /^>/){
    $header = $line;
#print "HEADER1: $header\n\n";
    $header =~ s/^>//;
#print "HEADER2: $header\n\n";
    $header =~ s/\s.*//;
#print "HEADER3: $header\n\n";
    if ($first == 0){
      $first = 1;
    }
    next;
  }
  if ($first == 0){ die("Not FASTA file.\n"); }
    $seqs{$header} = $seqs{$header}.$line;
#print "HASH: $seqs{$header}\n\n";
  }
  close(FILE);
  return \%seqs;
}
```

## Appendix I. RNA Prediction Script using RNAstructure

```
#!/usr/bin/perl -w
#use strict;
#use warnings;

my($key) = $ARGV[0];
my($directoryFileName) = $ARGV[1];
```

```
my($dirToCreate3) =
"/export/HTCondorScripts/Sequence_Chunks/".$directoryFileName."/".$directoryFileName."_C
hunks";

#HTCondor with Non-Pseudoknots using RNAstructure
system("tr '\\r' '\\n' < $dirToCreate3/".$key.".chunk > ".$dirToCreate3."/".$key.".out.proc");
```

# Biography

Gerardo Cardenas received his bachelor degree in Computer Science from the University of Texas at El Paso (UTEP) in El Paso, Texas. After he graduated, he worked in the Information Technology area in different industries including media, manufacture, healthcare, and government.

In 2009, he completed his M. Sc. Degree in Bioinformatics from the University of Texas at El Paso (UTEP). During his Masters, his research focus was on developing a tool to standardize the semantic descriptions of data and transformations commonly found in the domain of phylogenetic analysis. In fall 2010, he joined the Computational Science PhD program at UTEP.

While pursuing his degree, Gerardo Cardenas worked full-time as application support specialist for the City of El Paso. He is currently full-time employed at The El Paso Electric Company.

Gerardo Cardenas presented his research at International Society for Computational Biology (ISCB) 2016 conference and workshops including the 15th Joint UTEP/NMSU Workshop on Mathematics, Computer Science, and Computational Sciences.

He is conducting his dissertation under the supervision of Dr. Ming-Ying Leung. The focus on his research is to improve computational accuracy of RNA secondary structure prediction using a proposed two-chunk elimination method, reduce waiting time in the prediction of RNA secondary structures, and increase efficiency in the secondary structure prediction of long RNA sequences using distributed computing.

Contact Information: gacardenas@utep.edu

This dissertation was typed by the author