Using SWARM service for a GRID based Sequence Assembly

Karthik Narayan Muthuraman
Primary Advisor : Dr. Geoffrey Fox

May 23, 2010

Contents

1 Abstract

EST Sequence Assembly is the process of assembling Expressed Sequence tags to contigs and identifying the genes and predicting the gene functions in the organism. The process is highly time consuming and memory intensive and running the process on a desktop or a small server is not feasible. This entails a large scale computing environment to run the pipeline of programs involved and a mechanism to submit the programs to the large scale resource and track the status. In this project, we are using TeraGrid, a open scientific discovery infrastructure which integrates the several high performace clusters available across the country and a high level job scheduling web service framework SWARM, to submit and run jobs on the TeraGrid.

In this paper, I demonstrate the importance of the EST sequence assembly problem, the motivation for using grid computing resources and SWARM, and a performance evaluation of the system in comparison with one of the existing EST Sequence assembly programs, the EST Piper, which does not use Grid resources.

2 Introduction

EST Sequence assembly is the process of assembling Expressed Sequence Tags [1] of an organism to contigs and then predicting gene functions from them. At the beginning of the EST project the starting material for the construction of cDNA library [9] is selected. This can be cells, tissues or even whole organisms. From this, the messenger RNAs are isolated. mRNAs are highly unstabe and so they are *Reverse Transcribed* to relatively more stable forms called the complementary DNA or cDNA. The cDNA has to be amplified to form a cDNA library. This is accomplished by cloning the cDNA into plasmid vectors. The plasmids are amplified by transforming the bacterium E-coli to generate the cDNA library. The cDNA library forms the basis of generating EST sequences. Usually the cloning of cDNA is done directionally, that is, it is known at which end of the vector the 5 prime and 3 prime ends of the cDNA are located. The cloned sequence can thus be sequenced from both ends simultaneously. The identified nucleotide sequence can be exported to a computer and the raw data is then processed.

ESTs are randomly extracted sequences from the cDNA library. The ESTs have to be cleaned to remove repetetive regions and low complexity regions from them, which should not be considered for assembly. The cleaned ESTs are then clustered based on multiple sequence alignment and the are grouped to clusters based on overlapping regions or the consensus region. The consensus region in each cluster is then assembled to a contig. Genes can be predicted [5] from contigs using tools like ORF Predictor, Glimmer or Meme or the contigs can be blasted against a known protein database to identify the functions of the gebes predicted by the contigs.

3 Teragrid

TeraGrid [6] is an open scientific discovery infrastructure combining leadership class resources at eleven partner sites to create an integrated, persistent computational resource. Using high-performance network connections, the TeraGrid integrates high-performance computers, data resources and tools, and high-end experimental facilities around the country. Currently, TeraGrid resources include more than a petaflop of computing capability and more than 30 petabytes of online and archival data storage, with rapid access and retrieval over high-performance networks. Researchers can also access more than 100 discipline-specific databases. With this combination of resources, the TeraGrid is the world's largest, most comprehensive distributed cyberinfrastructure for open scientific research.

TeraGrid is coordinated through the Grid Infrastructure Group (GIG) at the University of Chicago, working in partnership with the Resource Provider sites: Indiana University, the Louisiana Optical Network Initiative, National Center for Supercomputing Applications, the National Institute for Computational Sciences, Oak Ridge National Laboratory, Pittsburgh Supercomputing Center, Purdue University, San Diego Supercomputer Center, Texas Advanced Computing Center, and University of Chicago/Argonne National Laboratory, and the National Center for Atmospheric Research.

TeraGrid resources are integrated through a service oriented architecture in that each resource provides a "service" that is defined in terms of interface and operation. Computational resources run a set of software packages called "Coordinated TeraGrid Software and Services" (CTSS). CTSS provides a familiar user environment on all TeraGrid systems, allowing scientists to more easily port code from one system to another. CTSS also provides integrative functions such as single-signon, remote job submission, workflow support, data movement tools, etc. CTSS includes the Globus Toolkit, Condor, distributed accounting and account management software, verification and validation software, and a set of compilers, programming tools, and environment variables.

TeraGrid uses a 10 Gigabits per second dedicated optical backbone network, with hubs in Chicago, Denver, and Los Angeles. All resource provider sites connect to a backbone node at 10 Gigabits per second. TeraGrid users access the facility through national research networks such as the Internet2 Abilene backbone and National LambdaRail.

4 Swarm

Swarm [8] is a high-level job scheduling Web service framework, developed for scientific applications that must submit massive number of high-throughput jobs or workflows to highly distributed computing clusters. The Swarm service itself is designed to be extensible, lightweight, and easily installable on a desktop or small server. As a Web service, derivative services based on Swarm can be straightforwardly integrated with Web portals and science gateways. The SWARM service is capable of scheduling millions of jobs over distributed clusters, monitoring framework for large scale jobs, a user based job scheduling service, ranking grid resources based on predicted wait times and provides a standard Web Service interface for web applications.

Users access the Swarm framework through a simple Web service client. This is useful for desktop users and Gateway style applications that need lightweight clients. This ease-of-use feature is also applied to file management. Swarm provides support for third party job submissions. Users or applications do not have to maintain input files allowing them to be lightweight. Thus, after installing the programs we would use on the grid resources, the programs can be launched by contacting the SWARM web service. We can specify the resources we need along with queue parameters and other parameters required by the program.

Once the user submits a large group of jobs, tracking the status of these jobs is critical. Swarm provides statistical status reports to the users. Each of the jobs maintains the status of *Requested*, for jobs that stay in the backend Database, *Queued*, for jobs in the Swarm user queue, *Submitted*, for the jobs with available resources, *Idle*, for the jobs waiting in batch queue system in the cluster, *Completed*, for jobs that have completed, *Held*, for jobs that been held, *Running*, for the jobs being executed in the cluster. A status check provides the summary of the job status.

5 Materials and Methods

Our Grid based EST Sequence assembly uses *Repeat Masker* for cleaning of ESTs, *PaCE* for parallel clustering, *CAP3* for assembly and *BLAST* for homology search. All of these programs are installed in *Bigred, Ranger, Cobalt* and *Abe.* Each of the steps in the pipeline are briefly describe below.

5.1 Repeat Masker

The repetetive and low complexity regions in the ESTs arise because of the errors in sequencing and have to be removed or masked. This is because, these regions may produce very high scores even though they are not a part of a gene. Thus, these regions are not only a burden for the assembler but affect the quality of the results by producing false hits. Thus, we have the cleaning module, Repeat Masker as the first step of our pipeline.

RepeatMasker [2] is a program that screens DNA sequences for interspersed repeats and low complexity DNA sequences. The output of the program is a detailed annotation of the repeats that are present in the query sequence as well as a modified version of the query sequence in which all the annotated repeats have been masked (default: replaced by Ns). On average, almost 50% of a human genomic DNA sequence currently will be masked by the program. Sequence comparisons in RepeatMasker are performed by the program cross_match, an efficient implementation of the Smith-Waterman-Gotoh algorithm developed by Phil Green.

As Repeat Masker is just comparison of the input sequence with database sequences, we can trivially paralellize it. That is, we can break the input into smaller chunks which can be handled more easily, and run them in parallel. As, we use four super computing resources, this would reduce the run time effectively.

There is a small probability that repeat masker might mask a few non-repetitive regions in the ESTs. So, we add a post-processing step after running repeat masker. After getting the output from Repeat Masker, for each sequence we calculate the length of the region masked and if is less that 10% the length of the original sequence, we take the original sequence in the place of the masked sequence. If the masked region is more than 10% of the original sequence, we take the masked sequence. The rationale for the post processing step is that, if the length of the masked region is too small, then the probability that it is a repetitive region is low and moreover the sequence will not burden the assembler.

5.2 PaCE

To enable fast clustering of large-scale EST data, we use PaCE [4] (for Parallel Clustering of ESTs), a software program for EST clustering on parallel computers. The organization of PaCE is as follows. It first builds a distributed representation of the GST data structure in parallel. This data structure is constructed for the input set of EST sequences and their WatsonCrick complements, and is used for on-demand generation of promising pairs of ESTs in decreasing order of maximal common substring length. The pair generation itself is done in parallel. Maintaining and updating of the EST clusters is handled by a single processor, which acts as a master processor directing the remaining processors to both generate batches of promising pairs and perform pairwise alignment on selected promising pairs. It is not mandatory to perform pairwise alignment of each generated pair because the current set of EST clusters may obviate the need to do so. Hence, the master processor is also responsible for the selection of pairs to be aligned and is a necessary intermediary between pair generation and alignment. In order to reduce communication overhead, the master processor dispatches the selected pairs in batches of size batchsize, a configurable parameter. To provide an added degree of flexibility in balancing the load, we do not require that a pair generated on a processor be allocated to the same processor if a pairwise alignment is needed.

Once the GST has been constructed, it can be used to generate promising pairs. The algorithm used for on-demand pair generation is a variant of the suffix tree algorithm for computing all maximal repeats of a sequence. A pair of EST sequences should be reported if they share a maximal common substring of length greater than or equal to a threshold value. Because a pair of sequences can have more than one such maximal common substring, our algorithm might generate the pair more than once. The number of such duplicates per pair generated by the algorithm is at most the number of distinct maximal common substrings of the pair. The key here is that we report the presence of a maximal common substring for a pair directly from the suffix tree, without having to look at all the smaller length non-maximal common substrings contained within it. This results in a significant reduction in the run-time as opposed to the methods deployed in generating promising pairs by existing software.

Once, PaCE is run, we get a set of clusters and a list of FASTA headers in each cluster. For each cluster formed by PaCE, we extract the FASTA sequences and form a seperate FASTA file. Each of these clusters would be fed to CAP3 for assembly. Table 1 gives the summary of clusters predicted by PaCE for various sizes of human ESTs, ranging from 10000 to 2 million ESTs.

Table 1: Clusters by PaCE

Case	No. of Sequences in input	No. of clusters by PaCE
1	10000	974
2	20000	2412
3	150000	12544
4	2000000	17465

5.3 CAP3

CAP3 [7] is the assembly program that follows PaCE in the pipeline. The assembly algorithm consists of three major phases. In the first phase, 5' and 3' poor regions of each read are identified and removed. Overlaps between reads are computed. False overlaps are identified and removed. In the second phase, reads are joined to form contigs in decreasing order of overlap scores. Then, forwardreverse constraints are used to make corrections to contigs. In the third phase, a multiple sequence alignment of reads is constructed and a consensus sequence along with a quality value for each base is computed for each contig. Base quality values are used in computation of overlaps and construction of multiple sequence alignments.

CAP3 is a the most memory intensive application in the pipeline and it's complexity increases exponentially with number of sequences in the input. As ESTs are extracted randomly from the cDNA library, CAP3 cannot be trivially parallelized. However, with PaCE, the clusters can be assembled in parallel. But, the number of clusters formed by PaCE increses with the size of input. From Table 1, we can see that the number of clusters formed for 2 million human ESTs is 17465. Hence, the total number of jobs for CAP3 is a bottle neck and has to be minimized.

As we can see from figure 1, majority of the clusters have less than 10 sequences in them. The run time for CAP3 on these sequences will be less than a minute, but these small jobs may spend even a few hours on the queue even before they start executing. So, we tried sorting the clusters based on number of sequences they carried, and run the smaller clusters , which is decided based on a set threshold, on the local machine and bigger clusters on the teragrid.

As the number of CAP3 jobs increases, the waiting time in the queue increases. This is because of the combined wait times in the job scheduler of

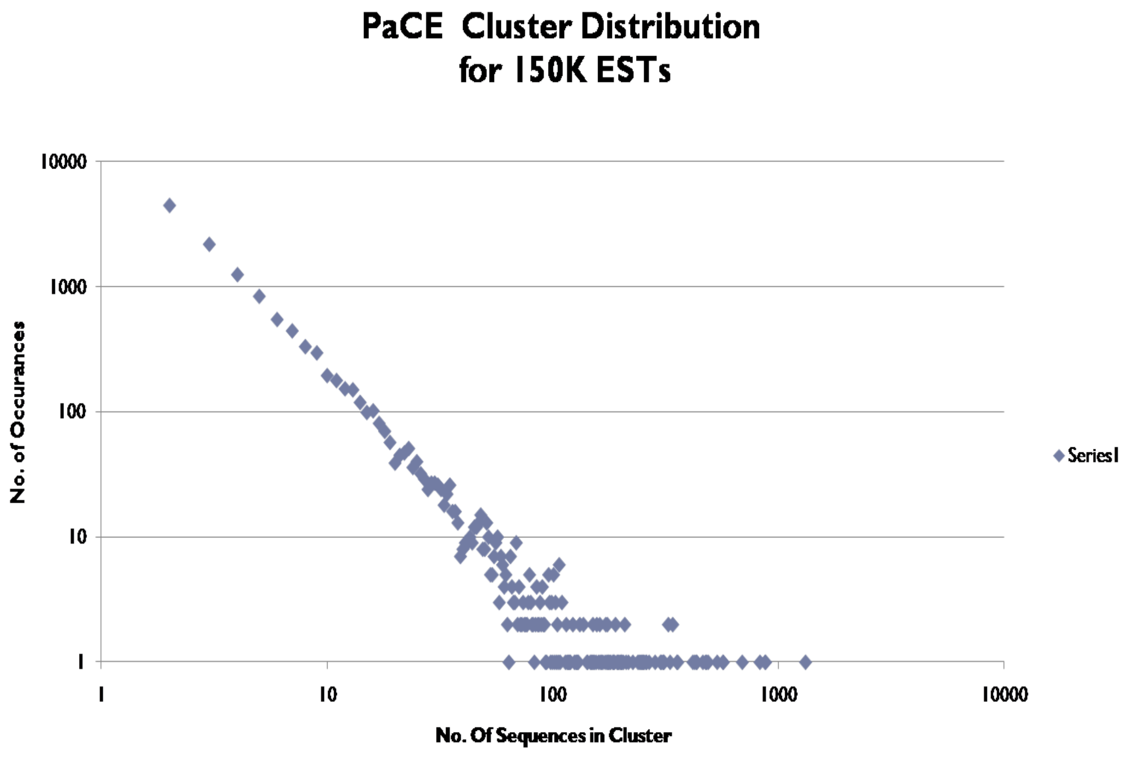

Figure 1: *Size distribution of clusters formed by PaCE for 150000 sequences.*

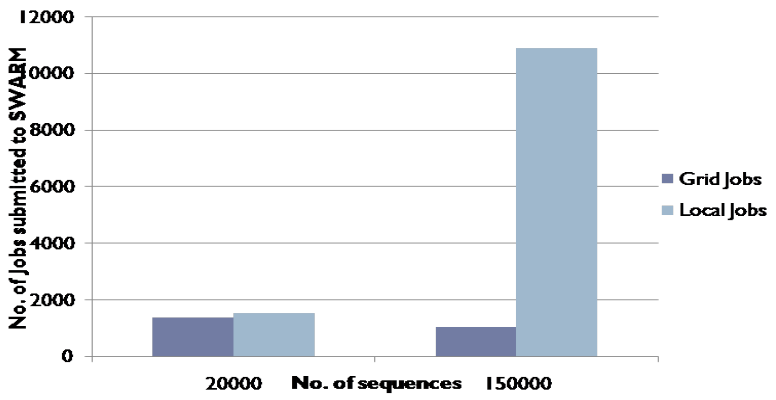

Figure 2: *Distribution of CAP3 jobs to the local and grid machines.*

Swarm and in the globus resource. To minimize the waiting time, we group the clusters into chunks and then run CAP3 on those chunks on the local and grid machines based on number of sequences in each chunk. This way, we do not break any cluster but reduce the number of CAP jobs considerably.

5.4 BLAST

BLAST [3] is the final program in the pipeline. We use *blastx* against *uniref100* database to predict the gene functions from the contigs predicted by cap3.

BLAST works through use of a heuristic algorithm. Using a heuristic method, BLAST finds homologous sequences, not by comparing either sequence in its entirety, but rather by locating short matches between the two sequences. This process of finding initial words is called seeding. It is after this first match that BLAST begins to make local alignments. While attempting to find homology in sequences, sets of common letters, known as words, are very important. For example, lets say that the sequence contains the following stretch of letters, GLKFA. If a BLASTp was being conducted under default conditions, the word size would be 3 letters. In this case, using the given stretch of letters, the searched words would be GLK, LKF, KFA. The heuristic algorithm of BLAST locates all common three-letter words between the sequence of interest and the hit sequence, or sequences, from the database. These results will then be used to build an alignment. After making words for the sequence of interest, neighborhood words are also assembled. These words must satisfy a requirement of having a score of at least the threshold, T, when compared by using a scoring matrix. Along the lines of terms stated above, if a BLASTp were being conducted, the scoring matrix that would be used would most likely be BLOSUM62. Once both words and neighborhood words are assembled and compiled, they are compared to the sequences in the database in order to find matches. The threshold score, T, determines whether a particular word will be included in the alignment or not. Once seeding has been conducted, the alignment, which is only 3 residues long, is extended in both directions by the algorithm used by BLAST. Each extension impacts the score of the alignment by either increasing or decreasing it. Should this score be higher than a pre-determined T, the alignment will be included in the results given by BLAST. However, should this score be lower than this pre-determined T, the alignment will cease to extend, preventing areas of poor alignment to be included in the BLAST results. Note, that increasing the T score limits the amount of space available to search, decreasing the number of neighborhood words, while at the same time speeding up the process of BLAST.

As BLAST is a database comparison done sequence by sequence, it can also be trivially paralellized. That is, we can break the input file into smaller chunks and run the smaller chunks in paralell on the four super computers we use.

6 Results

The Grid based EST sequence assembly pipeline was used to assemble 2 million human ESTs and the results are summarized below.

For an input of 2 million human ESTs PaCE took 1 hour and 22 minutes including waiting time in the queue. PaCE produced 75460 clusters and after grouping the clussters, the number of jobs for CAP3 was only 4073. CAP3 was run in 25 hours and 44 minutes which includes the waiting time in the queue. Thus, the total time for assembly was 27 hours and 6 minutes.

Table 2: Assembly results for 2 million human ESTs

Program	No. of jobs submitted to Swarm	Wait time + run time
Repeat Masker	1000	11:56 hours
PaCE	1	01:52 hours
CAP3	4073	25:44 hours
BLAST	893	49:00 hours

To validate the quality of results in our Grid based EST Assembly pipeline, we bench mark our tool against EST Piper [10] . We took 151000 ESTs of Daphnia pulex. The ESTs were then independently assembled using EST piper and our GRID based EST sequence assembly tool. The number of contigs predicted by EST piper was 17465 and the number of contigs predicted by Grid based EST sequence assembly was 20803. The contigs were then blasted against Uniref100 database with a e-value of 2. Grid based EST Assembly predicted, 90.8% of the genes predicted by EST piper. From the BLAST results, I compared the number unique genes predicted when considering only the best hit for each contig. This way Grid based EST Sequence assembly predicted 76.4% of the genes predicted by EST piper. The results are summarized in table 3.

While considering only the top BLAST hit for each contig, Grid based EST sequence assembly predicted 3284 genes not predicted by EST piper,

Table 3: Grid based EST assembly vs EST piper

No.	Name	EST piper	Grid based EST assembly
1	NO. of contigs	17465	20803
2	Average length of contigs	935	846
3	No. of hits	13216	15747
4	No. of unique top hit genes	9221	10329
5	Wait time + run time	52:00 hours	08:42 hours

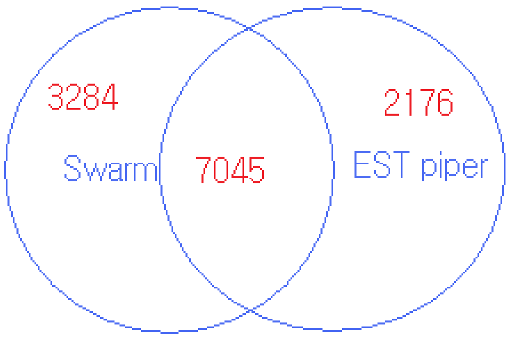

Figure 3: *BLAST results of Grid based EST Assembly vs EST piper for best hit.*

but missed 2176 genes predicted by EST piper. The results are summarized in the venn diagram.

References

[1] Gocayne JD Dubnick M Polymeropoulos MH Xiao H Merril CR Wu A Olde B Moreno RF et al. Adams MD, Kelley JM. Complementary dna sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651-1656, 1991.

[2] P Green AFA Smit, R Hubley. Repeat masker. 1996.

[3] Miller W Myers EW Lipman DJ. Altschul SF, Gish W. Basic local alignment search tool. *JOURNAL OF MOLECULAR BIOLOGY*, No 215::403–410, 1990.

[4] Suresh Kothari Anantharaman Kalyanaraman, Srinivas Aluru and Volker Brendel. Efficient clustering of large est data sets on parallel computers. *Nucleic Acids research*, No 11, 2003.

[5] Zhu W: Brendel V, Xing L. Gene structure prediction from consensus spliced alignment of multiple ests matching the same genomic locus. *Bioinformatics*, 20(7):1157-1169, 2004.

[6] C. et al. Catlett. Teragrid: Analysis of organization, system architecture, and middleware enabling new types of applications. *HPC and Grids in Action*, 2007.

[7] Xiaoqiu Huang and Anup Madan3. Cap3: A dna sequence assembly program. *Genome research*, No 9::868–877, 1999.

[8] M.Pierce Pallickara, SL. Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters. *IEEE Fourth International Conference on eScience*, pages 285–292, 12/2008 2008. December 7-12.

[9] Andreas Rohwer Paul M. Selzer, Richard J. Marhfer. *A Applied bioinformatics: an introduction*. Springer, 2008.

[10] Chris Hemmerich Ankita Sarangi John K Colbourne Zuojian Tang, Jeong-Hyeon Choi and Qunfeng Dong. Estpiper a web-based analysis pipeline for expressed sequence tags. *BMC Genomics*, 10: 174, 2009.