Shifting the Bioinformatics Computing Paradigm: A Case Study using Genome Annotation

Andrew Thrasher, Douglas Thain, and Scott Emrich
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN

Zachary Musgrave
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI

***Abstract*—Next generation sequencing technologies have enabled various entities, ranging from large sequencing centers to individual laboratories, to sequence organisms of choice and analyze them on demand. Sequencing and analysis, however, is only part of the equation: to learn about a certain organism, scientists need to annotate it. Each of these problems is highly parallel at a basic level of computation; however, only a few applications support even a single parallelization framework such as MPI. Ideally, because of overall increasing demand for computational analysis and the inherent parallelism available in these problems, applications should utilize a generic parallel framework to take advantage of a large variety of computing systems; this would enable labs of various sizes to harness the computing power available to them without forcing them to invest in a particular type of batch system. Here we describe modifications made to one particular tool, MAKER. MAKER is a tool for genome annotation that is provided as both a serial application and as an MPI application. We make modifications to enable it to run without MPI and to utilize a wide variety of distributed computing platforms. Furthermore, our proposed parallel framework allows for easy explicit data transfer, which helps overcome a major limitation of bioinformatics tools that generally rely on a shared filesystem. The distributed computing framework we choose to utilize can be used, even during early stages of development, to run bioinformatics tools on clusters, grids, and clouds.**

***Keywords*-Distributed computing; Bioinformatics**

I. Introduction

Bioinformatics has emerged over the past decade as a major producer of digital data and consequently a large consumer of computing resources. The tremendous explosion in sequencing capability with greatly reduced costs has enabled scientists to now sequence non-model organisms and investigate their genomes. Moreover, they are also (re)sequencing larger numbers of genomes from a population including humans [1]. To help facilitate their scientific investigations, many genome projects also perform additional sequencing such as expressed sequence tags (ESTs), RNAseq, etc. These factors all contribute to the large quantity and variety of data available in a modern genome project.

Parallel to the increase in sequencing power, algorithms for genome analysis and assembly have improved. Annotation, however, remains a bottleneck to genome analysis because accurately determining which regions of a genome are "useful" usually is a highly human/curator intensive process. As the number of genomes sequenced outpace available curation resources, there have been ongoing attempts to automate and provide reasonable annotations from mostly computational approaches. These methods, however, are often highly computationally intensive because they integrate a variety of datasources to increase annotation quality for a target genome.

Many of these bioinformatics problems have natural parallelism. We previously demonstrated that a clever choice in distributed computing frameworks can allow developers to leverage a variety of heterogeneous computational resources in a light-weight and relatively easy to program manner [2]–[5]. There are a variety of similar existing frameworks. A common feature of a good framework is an easy to utilize API, and we use Work Queue and Makeflow provided by the Cooperative Computing Lab at Notre Dame (http://ccl.cse.nd.edu/). The main contribution of this work is a case study that utilizes a diverse collection of computing resources for data-intensive annotation and resulting distributed bioinformatics tools for the research community.

II. Related Work

Gene finding in novel genomes can follow an evidence-based or *ab initio* framework. In the former approach, well-characterized protein data from related organisms along with expression data (including newer generation sequences) are used to build gene models based on these experimentally observed intron-exon boundaries. Because these data are expensive to generate and do not cover everything, *ab initio* approaches are used to predict intro-exon boundaries using only a sequence of a new genome. Popular *ab initio* approaches include GLIMMER [6], FGENESH [7], GenScan [8], SNAP [9] and Augustus [10].

Each *ab initio* approach uses complex probabilistic models trained on either a small subset of known genes or a subset of likely genes. For example, GLIMMer uses long open reading frames found in bacterial genomes for training because they are unlikely to occur by chance [6]. The commercial gene finder FGENESH relied on high confidence

EST alignments for training and has been reported to work well for plant genomes such as maize [11].

Here, we use Maker that incorporates both Augustus and SNAP into the gene prediction pipeline. Both Augustus and SNAP rely on Hidden Markov Models (HMMs) as their underlying probalistic model. The key innovation in Augustus is modeling of intron lengths on top of the HMMs, which can improve results [10]. SNAP uses similar intron length modeling and is very similar to GenScan [8] The most important SNAP features for large-scale gene prediction in novel genomes are flexible state diagrams to allow changing the underlying HMM and flexible inputs such as the order or Markov model and weight matrix used. This allows for both customization and "boostrapping" in which models from related species can be used to iteratively improve SNAP training sets [9]. This iteration improves predictions, and is a key component in the Maker pipeline.

One additional component of the SNAP/MAKER pipeline is use of the Core Eukaryotic Gene Mapping Approach (CEGMA) [12]. CEGMA, which is provided by the author of SNAP, contains a set of 458 proposed core genes that are present in many species. Specifically, these highly conserved genes provide a highly reliable set of training models because their conservation makes it easier to derive exon-intron structure in a novel genome [12]. This method utilizes profile-HMMs to produce gene structures.

III. The Structure of MAKER

Maker is a genome annotation utility produced at the University of Utah. It is a pipeline for reconciling the output of many different annotation tools. To do this, it first runs independent tools on the input data, usually a genome and supporting information such as proteins and expressed sequence tags (ESTs). It then combines the output of these tools and produces a consensus annotation for the genomic regions. The standard Maker instance is configured to run in either serial mode or in parallel using MPI [13] while relying on a network file system. It is specifically aimed at smaller genome projects. The authors describe the accumulation of unannotated genomes that continues to occur as sequencing costs drop and assembly algorithms improve. They assume that the annotation step has become the bottleneck in the genomics project pipeline.

Maker operates on data in "tiers". Each tier is comprised of FASTA-formatted nucleotide sequences with potential comparisons to supporting data such as proteins, ESTs, RNAseq, and other relevant information. Tiers also include parameters specified by the user for use with the various tools that provide output to Maker such as Basic Local Alignment Search Tool (BLAST) [14], the ab-intio gene predictor SNAP [9], Exonerate [15], and cross_match from the Phrap suite [16]. These tools are often tied together using the BioPerl [17] library, which allows users with a basic knowledge of a scripting language to complete complex tasks using functions provided in this library. This also allows users to perform common tasks, such as parsing BLAST output . This variety of applications and libraries makes parallelization a complicated task, which we discuss here.

Tiers are computed in order. If a tier fails to complete, Maker re-runs the tier by default up to a specified threshold of attempts. Each tier, comprised of pieces of the input data, utilizes the various tools of the pipeline. Maker then takes the output of each tier and consolidates them into a final annotation. (See Figure 1)

Maker tracks the progress of the annotation using a log file. This file is maintained in a shared filesystem and, in the case of MPI Maker, each worker node attempts to gain a lock of the file and then write to this file what the worker is currently doing. This allows the "master" process to track the current state of the annotation and also to restart it in the event of failure or termination. This setup, however, makes distributed computing difficult as all workers are all bound by this file. Scaling up is also difficult as each worker from a cluster, grid or cloud must wait to obtain the lock of the file before it can begin. If it fails to obtain the lock for a certain length of time, the worker has to fail. Recovery of this worker depends on the nature of the underlying distributed framework but may add latency in systems such as Condor where jobs do not start instantly.

Even so, the tiered nature of Maker is a prime candidate for using distributed resources. Many of the annotation tools are standalone executables and require a couple of input data chunks. If there was explicit data management the application would be easily portable and practical in various heterogeneous distributed computing environments. By relying on a network file system for implicit data transfer and management, on the other hand, the user base is locked into distributed computing resources with attached networked file systems. This makes utilization of popular resources like Amazon's EC2 platform very difficult. Note the use of a common log file on a networked file system compounds this difficulty. Ideally, the master process should be capable of tracking tasks that it has dispatched and of determining success or failure from worker responses. This would also eliminate the bottle-neck of multiple workers trying to synchronously access the log file and as the problem scales up resulting in many being forced to wait.

In the following section, we provide a description of work we have undertaken to address some potential limitations of the current Maker implementation that make it difficult to utilize a distributed computing environment. We provide this description to show the relative simplicity of using a distributed computing framework to assist an application in being useful across a wide variety of execution environments. The resulting tool is available to the broad scientific community and has been integrated into a popular workflow (Galaxy) to maximize its application.

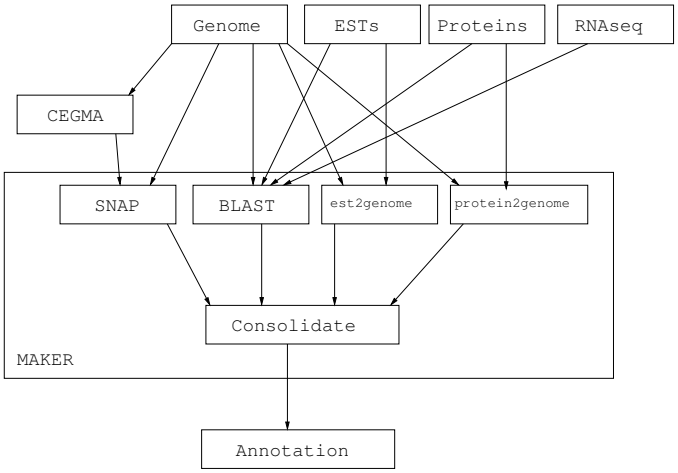

Figure 1. MAKER overview. Maker uses a number of external executables to produce an annotated genome from various evidence.

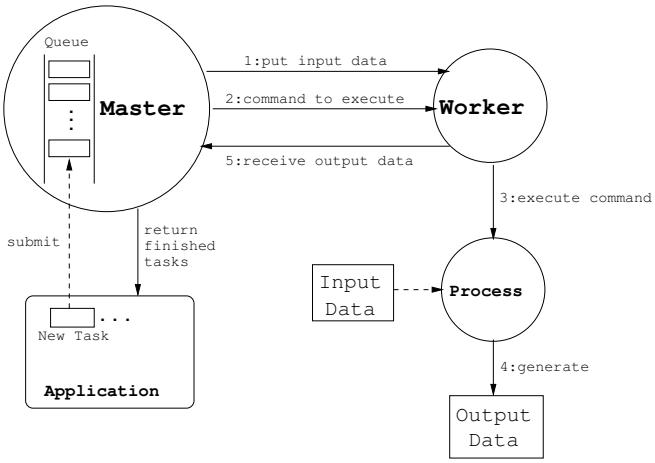

Figure 2. The general framework of a Work Queue application. An application creates a queue using the Work Queue API and submits tasks to the queue specifying the data to be processed, the executable and a command string. The queue then sends the information to a worker that is connected for processing. The worker returns the output to the queue that either marks the task completed and returns the output or marks the task failed and resubmits to another worker.

IV. Modifications and Evaluation of MAKER

In addition to the communities targeted by the authors of Maker, there are many others with growing genome annotation needs and constraints on resources. There are large genome project initiatives such as the 1000 genomes project [1] and the Anophelene cluster project [18]. These genome projects focus on comparative genomics, but may still require annotation services based on the large amounts of supporting data and related annotations generated, which Maker is well-suited to handle. The bottleneck is in the implementation. Users of Maker are confined to either serial mode or MPI mode on a grid. Many users and institutions have access to a variety of resources such as Condor [19], LSF [20], and SGE [21]. We believe bioinformatics applications should be capable of utilizing diverse computational resources to maximize availability to the biological user community.

To accomplish this, we have modified the Maker toolset to utilize a variety of distributed resources. We decided to rely on our prior knowledge and work with the Work Queue framework [2]. Work Queue is a light-weight master/worker implementation that uses worker processes which are submitted to a batch system. The worker count may fluctuate throughout the computational cycle and workers will process tasks until the completion of the computational workload. Workers can also be submitted to multiple batch systems. Additionally workers can be run as standalone processes on any machine so it is not necessary to have a batch system available to utilize individual computing nodes. This enables labs of various sizes from very small to large to leverage the computational resources to attack their annotation problems. The general architecture of a Work Queue application is presented in Figure 2.

The first major issue with combining Maker and Work Queue is that Maker is implemented in Perl and Work Queue in C. Therefore to simplify the combination of these tools we created a Perl module to interface with the Work Queue implementation. This was simplified by using SWIG [22] and creating an interface file that is input to SWIG and defines the functions and variables present in the C program. Once we had a Perl module that was a working Work Queue, we needed to modify Maker to utilize it.

As is often the case, this was a fairly straightforward but time consuming task. We needed to modify the Maker source code along with creation of an executable with some logic to be run on the worker nodes. The modification of the Maker source was primarily the addition of logic to call the Work Queue module for distribution of the computation. This involved setting up the Work Queue to accept tasks and then submitting the tiers as tasks to the Work Queue. We could potentially handle errors in the tier computation on the workers by signaling failure to the Work Queue that would automatically put the task back into the queue for recomputation. However, we instead return a status to the master node which then can determine which course of action to take in the case of failure, including resubmission of the tier to the queue. The stock Maker source code is 300 lines of Perl (including whitespace) and our modified version is 555 lines of Perl (including whitespace).

In addition to the benefits produced by distributing the computation across various resources, we also utilize other features the the Work Queue package such as generating a logfile. This allows us to generate visual representations of the path of computation and observe what occurred during computation (e.g., long tail effects). Also also provided valuable debugging information for our distributed environment as it provided a detailed description of what

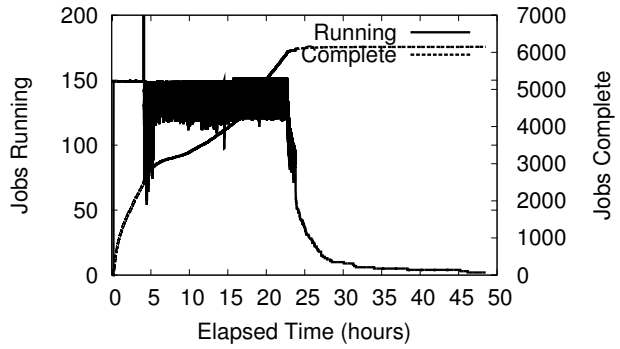

Figure 3. A run of the distributed maker platform using a 207Mb genome with additional supporting data.

occurred and when. Maker automatically splits the non-genomic inputs into chunks no larger than 100kb and then combines the output results. However this is not the case for genomic input. Instead it simply splits the genomic data into individual contigs. This means that we have an uneven computational problem. The computation time of chunks is directly related to the size of the input. This makes for a difficult distributed computation problem. We utilize the fast abort functionality of the Work Queue to enable us to abort abnormally long computation on problematic nodes in the distributed environment.

The authors of Maker provide a few compute times in [23]. They provide their analysis on a 2.236-Mb sequence on a 32 GB-RAM machine using a single processor on a machine with eight dual-core 2-GHz processors. They measure the compute time for the full Maker pipeline at 549 minutes on one processor and 299 on two processors. With our parallel modifications to run on Work Queue across distributed resources, we are able to scale up to the complete annotation of a 207Mb genome with 2Mb of EST and 1893 Mb of RNAseq supporting evidence. This required 2911 min. in a distributed environment with a worker pool of 150 nodes. The computation was able to use the full 150 nodes for 1363 min., but then a long-tail effect comes in (Figure 3).

The authors of Maker also report their findings on the *S. mediterranea* genome with a rate of 4.1 h/Mb. The genome is 850Mb which gives a total processing time of 3485 hr. If we assume our 207Mb genome with 1895Mb of supporting evidence will process at the same rate, this gives a single-core processing time of 848.7 hr. Using this we can compute our speedup to be 17.5x for this genome.

We ran the Maker serial tool along with our Work Queue on the *Caenorhabditis japonica* genome. We retrieved the genome from Wormbase [24] along with protein sequences. Additionally we retrieved approximately 30000 EST sequences and assembled them with CAP3 [25]. The genome is composed of 18817 contigs with a total length of over 166MB and the protein sequences total 36105 with a total length of over 10MB. The assembled EST data is 10000 contigs with a total length of over 5MB. These were run through Maker using the same control files. The control files were setup to use SNAP and est2genome as predictors. All other Maker options were held at their default settings. The serial version of Maker was run on a 12 core AMD Opteron machine with 64GB RAM. The master for the Work Queue version was run on the same machine with workers being submitted to our campus Condor [19] pool. We ran the parallel version of Maker using a constant set of 50 workers. As of submission time both are still running but we approximate the runtimes here based on current progress and speed. The serial version will require over 600 hours to complete while the parallel version will complete in approximately 15 hours across 50 nodes for a total compute time of 750 hours. This gives a speedup of 40x. This is less than the ideal linear speedup of 50x but is still a good result.

Table I
MAKER RUNTIME FOR VARIOUS ORGANISMS

Organism	Genome	EST	RNAseq
Anopheles stevensi	207MB	2.2MB	1893MB
Glossina morsitans	366MB	48MB	0MB
Rhodnius	699MB	6MB	372 MB

We have performed additional work to enhance our WQ-Maker's usefulness for the community at large: we added our tool to the popular Galaxy framework [26]–[28] for bioinformatics. The tool runs our Work Queue version of Maker by default, but could be made to run either the serial version of Maker or the MPI version of Maker by changing the executable that gets called. As Maker relies on a shared filesystem for distribution of work, Galaxy needs to be installed on a shared filesystem or at least have its job working directory on a shared filesystem for the Work Queue or MPI versions to function properly. We are working to eliminate this restriction in future versions of WQ-Maker.

V. RECOMMENDATIONS

For the bioinformatics community to be fully capable of utilizing the wide variety of computing resources at their disposal, there are several key issues that need to be addressed by developers. It is well established given the increasing popularity of cloud-based solutions many bioinformatics applications have highly parallel components. Based on our experience developers should consider changes to several key areas of the current bioinformatics computing paradigm.

First, many bioinformatics applications rely on a shared filesystem when explicit transfers may help run on more heterogenous resources. This also helps with debugging, both for the developer and the user, by allowing them to log and identify problems with the application that are caused by network failure and other issues which have

prevented a worker node from accessing key files. Explicit transfer must be tightly controlled in environments like Amazon where transfer into and out of a cloud incurs cost but running a master process in the cloud overcomes this concern. More importantly, a framework with build in fault tolerance like work queue or make flow alleviate the need for file-based checkpointing, which often limits the scalability of the application regardless of the amount of parallelism inherently available.

Second, common bioinformatics operations are often abstracted into libraries such as BioPerl. While convenient for developers and users, in more heterogenous environments like ad hoc clouds the library must be available across all compute nodes by either being installed on each node or on a shared filesystem. Neither of these are desirable situations large scale commodity-driven computation. We have found a paradigm similar to the functional programming revived in Google's map reduce is preferred as compiled and statically linked C executables can be more easily transferred around to multiple systems in our experience compared with Perl or Python.

Third, a distributed computing framework such as Work Queue enables the utilization of an application across a variety of execution environments (e.g., a SGE closer, Amazon EC2). In fact, we have used this framework effectively to harness Condor, SGE, individual workstations, and EC2 compute cycles for large bioinformatics workflows like MAKER. The clear advantage of a system such as this is that it allows the user to utilize any and all computing resources that are currently available to them (or can be purchased from a third party) offering elastic scaling for these applications without refactoring the underlying code.

VI. Conclusions

We present a modified Maker annotation tool capable of being run on a variety of distributed computing resources and commodity resources. This modification will enable even small labs to leverage additional computing power to complete the annotation of their genomes. Additionally this technique is generally applicable across bioinformatics tools. Tool developers can use the Work Queue platform to enable their application to be run across a variety of computational resources without having to know the details of all of the possible distributed environments.

The use a of framework for distributing bioinformatics computational problems across distributed computing resources needs to become commonplace. The framework must also be portable across a variety of systems, not tied to one particular system. Additionally this implies that tool developers need to consider the data transfer of their applications. Rather than simply relying on the implicit data transfer of a shared filesystem, tool developers should utilize frameworks that have explicit data transfer syntax. This will allow portable applications that can be of use to people in a variety of environments without being restricted to a particular implementation. Further, we believe tool developers need to be conscientious in their library selection. Utilizing libraries such as BioPerl inside of an application speeds implementation but greatly limits scaling in a heterogeneous distributed computing environment. Having these libraries utilized within an application will require them to be available on the computation nodes which makes distributed computing efforts much more difficult.

References

[1] R. Durbin, D. Altshuler, G. Abecasis, D. Bentley, A. Chakravarti, A. Clark, F. Collins, F. De La Vega, P. Donnelly, M. Egholm *et al.*, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, 2010.

[2] L. Yu, C. Moretti, A. Thrasher, S. Emrich, K. Judd, and D. Thain, "Harnessing parallelism in multicore clusters with the all-pairs, wavefront, and makeflow abstractions," *Cluster Computing*, vol. 13, pp. 243–256, 2010, 10.1007/s10586-010-0134-7. [Online]. Available: http://dx.doi.org/10.1007/s10586-010-0134-7

[3] A. Thrasher, R. Carmichael, P. Bui, L. Yu, D. Thain, and S. Emrich, "Taming complex bioinformatics workflows with weaver, makeflow, and starch," in *Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on*. IEEE, 2010, pp. 1–6.

[4] I. Lanc, P. Bui, D. Thain, and S. Emrich, "Adapting bioinformatics applications for heterogeneous systems: a case study," in *Proceedings of the second international workshop on Emerging computational methods for the life sciences*. ACM, 2011, pp. 7–14.

[5] C. Moretti, M. Olson, S. Emrich, and D. Thain, "Highly scalable genome assembly on campus grids," in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. ACM, 2009, p. 12.

[6] A. Delcher, D. Harmon, S. Kasif, O. White, and S. Salzberg, "Improved microbial gene identification with glimmer," *Nucleic acids research*, vol. 27, no. 23, p. 4636, 1999.

[7] A. Salamov and V. Solovyev, "Ab initio gene finding in drosophila genomic dna," *Genome Research*, vol. 10, no. 4, p. 516, 2000.

[8] C. Burge and S. Karlin, "Prediction of complete gene structures in human genomic dna1," *Journal of molecular biology*, vol. 268, no. 1, pp. 78–94, 1997.

[9] I. Korf, "SNAP: Semi-HMM-based Nucleic Acid Parser," *Ian Korf homepage: http://homepage.mac.com/iankorf/*.

[10] M. Stanke and S. Waack, "Gene prediction with a hidden markov model and a new intron submodel," *Bioinformatics-Oxford*, vol. 19, no. 2, pp. 215–225, 2003.

[11] S. Emrich, W. Barbazuk, L. Li, and P. Schnable, "Gene discovery and annotation using lcm-454 transcriptome sequencing," *Genome research*, vol. 17, no. 1, p. 69, 2007.

[12] G. Parra, K. Bradnam, and I. Korf, "Cegma: a pipeline to accurately annotate core genes in eukaryotic genomes," *Bioinformatics*, vol. 23, no. 9, p. 1061, 2007.

[13] I. Foster, *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley, 1995.

[14] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[15] G. Slater and E. Birney, "Automated generation of heuristics for biological sequence comparison," *BMC Bioinformatics*, vol. 6, no. 1, p. 31, 2005. [Online]. Available: http://www.biomedcentral.com/1471-2105/6/31

[16] P. Green, "Phrap," *http://www.phrap.org/phredphrap/phrap.html*.

[17] J. E. Stajich, D. Block, K. Boulez, S. E. Brenner, S. A. Chervitz, C. Dagdigian, G. Fuellen, J. G. Gilbert, I. Korf, H. Lapp, H. Lehvslaiho, C. Matsalla, C. J. Mungall, B. I. Osborne, M. R. Pocock, P. Schattner, M. Senger, L. D. Stein, E. Stupka, M. D. Wilkinson, and E. Birney, "The Bioperl Toolkit: Perl Modules for the Life Sciences," *Genome Research*, vol. 12, no. 10, pp. 1611–1618, 2002. [Online]. Available: http://genome.cshlp.org/content/12/10/1611.abstract

[18] N. Besansky, "Genome Analysis Of Vectorial Capacity In Major Anopheles Vectors Of Malaria Parasites," *White Paper*, 2008. [Online]. Available: http://www.vectorbase.org/Help/Anopheles_species_cluster_white_paper

[19] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing*. Wiley Online Library, 2002, pp. 299–335.

[20] http://www.platform.com, "LSF Home Page." [Online]. Available: http://www.platform.com

[21] W. Gentzsch, "Sun Grid Engine: Towards Creating a Compute Power Grid," in *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, 2001.

[22] http://www.swig.org/index.php, "Simplified Wrapper and Interface Generator (SWIG)." [Online]. Available: http://www.swig.org/index.php

[23] B. Cantarel, I. Korf, S. Robb, G. Parra, E. Ross, B. Moore, C. Holt, A. Sánchez Alvarado, and M. Yandell, "MAKER: an easy-to-use annotation pipeline designed for emerging model organism genomes," *Genome research*, vol. 18, no. 1, p. 188, 2008.

[24] L. Stein, P. Sternberg, R. Durbin, J. Thierry-Mieg, and J. Spieth, "Wormbase: network access to the genome and biology of caenorhabditis elegans," *Nucleic Acids Research*, vol. 29, no. 1, pp. 82–86, 2001.

[25] X. Huang and A. Madan, "Cap3: A dna sequence assembly program," *Genome research*, vol. 9, no. 9, pp. 868–877, 1999.

[26] B. Giardine, C. Riemer, R. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor *et al.*, "Galaxy: a platform for interactive large-scale genome analysis," *Genome research*, vol. 15, no. 10, p. 1451, 2005.

[27] D. Blankenberg, G. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor, "Galaxy: A web-based genome analysis tool for experimentalists," *Current Protocols in Molecular Biology*, 2010.

[28] J. Goecks, A. Nekrutenko, J. Taylor, and T. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. R86, 2010.