

Initial experiences with GeneRecon on MiG

Thomas Mailund and Christian N.S. Pedersen
Bioinformatics Research Center (BiRC),
Dept. of Computer Science,
University of Aarhus, Denmark,
Email: {mailund,cstorm}@birc.dk

Jonas Bardino, Brian Vinter,
and Henrik H. Karlsen
Dept. of Mathematics & Computer Science,
University of Southern Denmark, Denmark,
Email: {jones,vinter,karlsen}@imada.sdu.dk

Abstract— We report on very early experiences with running a bioinformatics simulation study on a newly developed grid architecture. We briefly describe the bioinformatics application—an association mapping algorithm for both locating disease loci and separating cases into those diseased due to genetic factors and those diseased solely due to environment factors—and describe the grid architecture and how the application is set up to run on the grid.

1. Introduction

Locating the genetic factors behind common inheritable diseases is a highly relevant problem in medicine and bioinformatics [3]. The complexity of the problem, however, implies that finding biologically sound solutions requires algorithms of high time complexity, which makes extensive experimentation difficult. This paper describes the work done, and results from, utilising Grid computing [2] for large scale experiments with an algorithm, GeneRecon, for detecting genetic mutation clusters.

GeneRecon is a tool for analysis of population genetic data from case/control studies. The analysis works on genetic data which is collected from patients with a specific disease and a control group without this disease, and tries to locate genes that are increasing the risk of disease. The analysis, which is based on a Markov-chain Monte-Carlo method, is extremely CPU-demanding, since it requires searching through a very large parameter-space. The presented work seeks to evaluate a method that both significantly reduces the running time of the method and additionally can, more precisely, determine which patients have a given disease because of a genetic markup, rather than another cause, e.g. environmental effects. The work includes thousands of GeneRecon runs, which in practice is impossible without access to a very large number of processors.

Minimum intrusion Grid, MiG, is a new platform for Grid computing which is driven by a stand-alone approach to Grid, rather than integration with existing systems. The goal of the MiG project is to provide a

Grid infrastructure where the requirements on users and resources alike, to join Grid, is as small as possible. While striving for minimum intrusion, MiG will still seek to provide a feature rich and dependable Grid solution. In the remainder of this paper we describe the problem of isolating mutation clusters in section 2, introduce the Minimum intrusion Grid in section 3, and describe the setup that enables us to run large scale experiments with GeneRecon using multiple processors that are geographically distributed in section 4. Finally, we outline future work and conclusions are reached.

2. Association Mapping and Mutation Clusters

For diseases with a genetic component, carriers of high risk alleles are often descendants of one or a few founders, in whose genome the allele appeared by mutation. Recent shared ancestry of the carriers in the disease position results in a higher homogeneity, among the carriers, in the genomic region around the disease affecting gene, than in the population at large. This, in turn, causes markers—points of genetic variation—around the disease gene to be more associated with each other—in a statistical sense, a phenomenon known as *linkage disequilibrium*, or LD. The use of LD for disease gene mapping has in recent years received much attention, and is believed by many to hold great promise in the mapping of common complex diseases [3].

The *Common Disease, Common Variant* (CD/CV) hypothesis [7], fundamental to association mapping and based on the history of the human population, states that most common diseases with a genetic component are affected by a few alleles with a (relatively) high frequency in the population but with a (relatively) low penetrance. If it holds true, we would expect that, in a case/control study, several unaffected individuals would be carriers of the increased risk mutation—but not diseased, due to the low penetrance—and perhaps, at the same time, only few of the diseased individuals would carry the increased risk mutation while the remaining affected

individuals have the disease due to environmental causes. In such a study, identifying the disease carriers among the affected individuals would greatly benefit the search for the disease gene once a candidate genome region has been found.

To approach this problem, we have developed a *Markov-chain Monte Carlo* (MCMC) algorithm¹ based on Morris *et al.*'s shattered coalescent method [6], but incorporating ideas from Liu *et al.* [4] and Molitor *et al.* [5], where we separated the affected individuals into *mutation clusters*—where affected individuals in the same cluster are assumed to be descendants of a common founder—and a *null cluster*—for individuals affected due to environmental factors and not genetic factors. By sampling, during the run of the MCMC, the distribution of affected individuals among the mutation and null clusters, we hope to be able to infer which individuals carry an increased risk mutation and which are affected solely due to environmental factors, while at the same time locating the locus of the disease affecting gene. Introducing the mutation and null cluster can potentially also greatly reduce the running time of the algorithm, as we only explicitly model the genealogy of the cases in the mutation clusters, not of all cases, thus reducing the search space significantly.

While the methods we have based our new technique upon have proved themselves highly efficient in locating the disease locus on case/control data, they are unfortunately also very CPU demanding and requires several hours—or days—for a successful computation with a few hundreds of cases and controls, with a few tens of markers. This makes computer clusters or computer grid architectures essential for validating new mapping methods such as ours, since such validation requires analysis of a large number of (simulated) datasets to get useful statistics about the performance of the method.

3. Minimum Intrusion Grid (MiG)

MiG² is a Grid middleware model and implementation designed with previous Grid middleware experiences in mind. In MiG central issues such as security, scalability, privacy, strong scheduling and fault tolerance are in-

¹The new MCMC algorithm has been implemented in the association mapping tool *GeneRecon*, developed in cooperation with the Danish company Bioinformatics ApS (<http://www.bioinformatics.dk>). *GeneRecon* is commercially available, but with discounts for academic users.

²MiG is a non-profit project mainly developed by a group of computer scientists from University of Southern Denmark, Odense (SDU).

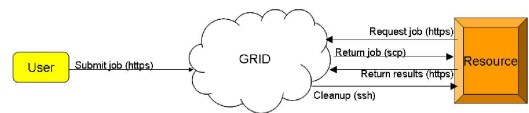


Fig. 1. MiG: Architectural overview.

cluded by design. Other Grid middlewares tend to suffer from problems with at least one of those issues.

The MiG model seeks to be non-intrusive in the sense that both users and resources should be able to join the Grid with a minimal initial effort and with little or no maintenance required. One way to obtain these features is keeping the required software installation to a functional minimum, e.g. the software that is required to run MiG includes only ‘need to have’ features, while any ‘nice to have’ features are completely optional. This design philosophy has been used, and reiterated, so stringently that in fact neither users nor resources are required to install *any* software that is MiG specific.

Another area where MiG strives to be non-intrusive is the communication with users and resources. Users in general and resources in particular can not be expected to have unrestricted network access in either direction. Therefore the MiG design enforces that all communication with resources and users should use only the most common protocols known to be allowed even with severely restricted networking configurations. Furthermore resources should not be forced to run any additional network-listening daemons.

Fig. 1 depicts the way MiG separates the users and resources with a Grid layer, which users and resources securely access through one of a number of MiG-servers. The MiG model resembles a classic client server model where clients are represented by either users or resources. The servers are represented by the Grid itself, which in the case of MiG is a set of actual computers, not simply a protocol for communicating between computers. Upon contacting Grid any client can request to either upload or download a file. Users in turn can additionally submit a file to the job-queue while resources can request a job.

Most of the actual functionality is located at the MiG servers, where it can be fully maintained and controlled by the MiG developers. Thus, in addition to minimising the user and resource requirements, the Grid layer simplifies consistent deployment of new versions of the software.

The security infrastructure relies on all entities: users, MiG-servers and resources, being identified by a signed

certificate and a private key. The security model is based on sessions and as such requires no insecure transfers or storage of certificates or proxy-certificates as it is seen with some Grid middlewares.

Users communicate securely with the server by means of the HTTPS protocol using certificates for two-sided authentication and authorisation. Server communication with the resources is slightly more complicated as it combines SSH and HTTPS communication to provide secure communication and the ability to remotely clean up after job executions.

MiG jobs are described with mRSL, which is an acronym for *minimal Resource Specification Language*. mRSL is similar to other Resource Specification Languages, but keeps the philosophy of minimum intrusion, thus mRSL tries to hide as many aspects of Grid computing as possible from the user. To further hide the complexities of Grid computing from the user, MiG supplies every user with a Grid home-directory where input and output files are stored. When a job makes a reference to a file, input or output, the location is simply given relative to the users home-directory and thus all aspects of storage-elements and transfer-protocols are completely hidden from the user. The user can access her home-directory through a web-interface or through a set of simple MiG-executables for use with scripting.

Job management and monitoring is very similar to file access so it is also done either through the web-interface or with the MiG-executables. Users simply submit jobs to the MiG-server, which in turn handles everything from scheduling and job hand out to input and output file management. An important aspect of this is that a job is not scheduled to a resource before the resource is ready to execute the job. Resources request jobs from the MiG-server when they become ready. The MiG-server then seeks to schedule a suitable job for execution at the resource. If one is found, the job with input files is immediately handed out to the resource. Otherwise the resource is told to wait and request a job again later. Upon completion of a job, the resource hands the result back to the MiG-server which then makes the result available to the user through her home directory.

Even though MiG is a new model, we have already implemented a stable single server version. It relies on the Apache web server (<http://httpd.apache.org/>) as a basis for the web interface and further functionality is handled by a number of cgi-scripts communicating with a local MiG server process. We have decided to implement as much of the project as possible in Python (<http://www.python.org/>) since it provides a very clear

syntax, a high level of abstraction and it allows rapid development.

The test MiG-server used for the Mutation Clusters experiment has three resources associated: Two clusters and a Network of Workstations. All in all those three resources deliver the computational power of about 128 Pentium IV CPUs at all times.

4. Mutation Clusters on MiG

Since the Mutation Clusters experiment is basically a large parameter sweep, the entire experiment can easily be partitioned into a lot of similar, independent jobs, where each job can be done in a reasonable amount of time—ranging from a single day to about a week—on a standard single-CPU computer. This makes the experiment setup very simple, as each job can be sent to almost any available resource, and any failed job can just be re-executed at a later time or on another resource. Parallelism comes from different jobs being executed concurrently on separate resources rather than parallelism within each job. Thus the parallel execution of the Mutation Clusters experiment can be expected to be highly efficient.

Initially, we only expected to use a small local cluster computer for the Mutation Clusters experiment. However, as the experiment proceeded it soon became clear that more computing resources were required for the experiment to finish within a reasonable time frame, so we inspected the alternatives. A Grid solution was an obvious candidate due to the high level of parallelism possible even with stand-alone computers. We were offered access to two such Grid solutions, that could each potentially deliver the extra computing power, NorduGrid [1] and MiG. In the end we chose to mainly use MiG since it consistently delivered enough resources even at this early stage of the development process.

Grid execution of an application generally requires some preparations. Apart from creating the Grid-specific job description most of the preparations are of a general kind that would apply to any Grid system. To maximise the number of suitable resources for the Grid jobs, it is necessary to minimise the number of external dependencies. In this case the original version of the application depended on the availability of Guile Scheme (<http://www.gnu.org/software/guile/>). Since Grid resources in general cannot be expected to have Guile Scheme available, we chose to statically link the software with Guile. Then the executable only depended on having a suitable C library available at the resources.

```

::RUNTIMEENVIRONMENT::
GENERECON-2.1.3
::EXECUTE::
$GENERECON_HOME/generecon run-1304.scm
::INPUTFILES::
run-1304.scm
::OUTPUTFILES::
locus.txt +JOBID+/locus.txt
cluster.txt +JOBID+/cluster.txt
connectedness.txt +JOBID+/connectedness.txt
likelihood.txt +JOBID+/likelihood.txt
::CPUTIME::
345600

```

Fig. 2. An mRSL specification for a Mutation Cluster run.

The next step was to create job descriptions in mRSL. Since we already knew how to manually run the jobs from the command line, this was mostly a matter of filling in the command line information in an mRSL template. Output files that were explicitly fetched from the cluster in the initial setup were specified as output files in the mRSL to make them automatically available in the MiG home directory after job execution.

Unfortunately, the three types of CPU resources did not share the same version of the C libraries, so although the software could be compiled on all the platforms, we needed different versions for each. Thus we created and deployed a *runtime environment* to be able to select the right version for job execution on a particular resource. A runtime environment is a way of describing the execution environment required by a job in the resource specification language. When a particular runtime environment is specified in the mRSL the resource will automatically set up that environment before executing the job. The correct version of the compiled tool was then simply selected by specifying the environment variable `GENERECON_HOME`, as shown in Figure 2.

The mRSL script should hopefully be quite easily understood. The `::RUNTIMEENVIRONMENT::` specifies that we wish to use a resource that offers the GeneRecon runtime-environment. The `::EXECUTE::` block holds the commands to be run in the job, which in this case is only `generecon` in our case, the path to `generecon` is guaranteed to be `$GENERECON_HOME` since we have required the GeneRecon runtime-environment. The `generecon` should be run with the parameter `run-1304.scm` which is also listed under `::INPUTFILES::`, meaning that this file should be copied from the users home-directory to the resource before the execution should begin. Under `::OUTPUTFILES::` a number of result files are listed,

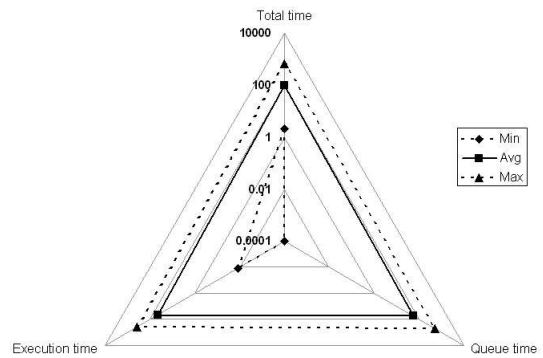


Fig. 3. MiG: Job statistics.

meaning that they should be copied to the users home-directory after job-completion. The term `+JOBID+` is a generic reference to the unique id of the job, which in this case is used to create a unique directory for each execution so that the result-files do not overwrite each other. Finally `::CPUTIME::` is set to 345600 seconds, or 96 hours, to ensure that the job will have sufficient time to complete.

When we introduced the runtime environment it also greatly simplified the job specification file in another way. The Mutation Cluster jobs are set up and controlled in the tool using a number of Scheme files, some of which are shared for all jobs and some of which are specific to the individual jobs. In the initial setup, all those files were specified as input files so that MiG implicitly copied each Scheme file to the job environment before execution. By using the runtime environment we could refactor the set of Scheme files into a library of modules stored with the runtime environment, and a single job-specific file; after this, only the job-specific file is specified and thus copied for the job execution.

The experiment included too many jobs to manually dispatch and monitor through the web interface. Instead we downloaded the MiG executables and created simple scripts to automate some level of job management using those executables. The scripts are connected to a MySQL (<http://www.mysql.com>) database for keeping track of job parameters and job results. Since MiG automatically hands out jobs to resources when they are idle, large batches of jobs were submitted to the MiG-server without worrying about load balancing and congestion.

All jobs were submitted at the same time and thus the first job started running almost immediately while the last jobs were queued for a long time. The resources that were made available to the GeneRecon experiments

differed from 1.7 GHz Intel Celeron CPUs to, 3.2 GHz Intel Xeon-64 CPUs. This combined with the varying execution times of the individual jobs, meant that execution times varied from a minimum 42 seconds to a maximum of more than 16 days! The minimum, maximum and average times for total turnover time, time in queue and time execution can be seen in Fig. 3

For the first part of the experiment we have validated that introducing the mutation and null clusters gives a runtime speedup, but does not reduce the methods ability to locate the disease locus significantly. For this study we have, for datasets simulated under various parameters, run several chains of the algorithm both with and without clustering, and then calculated the distance from the true location to the inferred location. This part of the experiment is now almost completed and the results are promising; when running with a mutation cluster of 50% of the cases, the running time is reduced to a third of running without clusters, but with only a 10%–20% reduction in precision, as shown on Fig. 4.

For the next phase of the experiment we will extract information about how long each case has been included in the mutation cluster, as opposed to the null cluster, and see if we can infer from this, which cases are mutants and which cases are diseased from environmental effects.

5. Future Work

The initial experiments with GeneRecon on MiG has been rewarding and we plan to continue this work in several directions. From a bioinformatics perspective there are many topics that become possible to explore with the availability of large computational resources, such as those provided by MiG. The efficiency of the disease loci inferences depends very much on the dataset which in turn depends on various parameters, such as recombination and mutation rates, population structure, and penetrance. Without access to grid-like CPU resources, this parameter space cannot be explored to fully figure out which parameters significantly affects the inference results and which have little effect on the algorithms efficiency.

An interesting problem, from the grid point of view, that will be further investigated, is whether it is possible to use check-pointing techniques to support migration of GeneRecon processes. This becomes interesting once the MiG project starts working on utilising ‘Screen Saver Science’ as part of its resource model. If a large number of computers join MiG when they are in screen-saver mode, the amount of available compute-resources increases significantly, but it is unlikely that a computer will remain in screen-saver mode for 48–96 hours, which is the time required to complete a GeneRecon job.

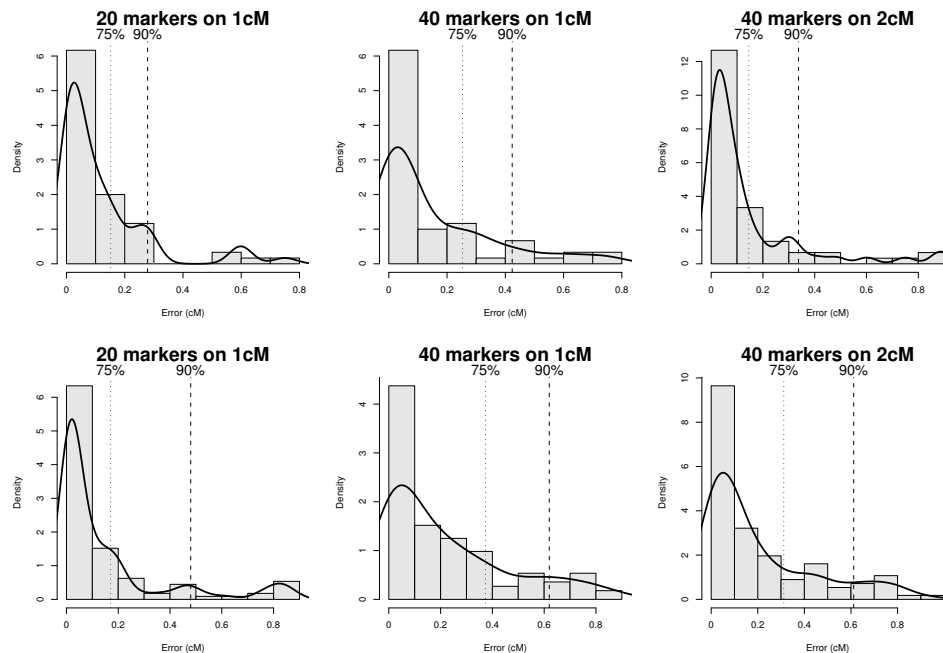


Fig. 4. Distances between inferred and true gene loci. The top row shows a summary the errors in locating the disease loci without using clusters; the bottom row shows the errors for running the mutation cluster method. The dashed lines show the 75% and 90% quantile, respectively.

6. Conclusions

The work presented in this paper represents very early experiences with both Minimum intrusion Grid and using Grid for GeneRecon experiments. Though Minimum intrusion Grid is in its very early stages we have successfully run a large scale Grid experiment with GeneRecon. In a period of two weeks GeneRecon has executed just under four CPU-years. At the time of writing we still need approximately one more CPU-year before the results are ready for analyses, this should be in an additional three days. To develop and test an application such as GeneRecon, such CPU power is an absolute requirement, and obtaining the processing power through MiG has proved very easy indeed. To be able to tap-into even more processing power, we are planning a screen-saver-science component for MiG and a checkpoint-restart feature for GeneRecon so that it may utilise such resources.

References

- [1] Paula Eerola, Balázs Kónya, Oxana Smirnova, Tord Ekelöf, Mattias Ellert, John Renner Hansen, Jakob Langgaard Nielsen, Anders Wäänänen, Aleksandr Konstantinov, and Farid Ould-Saada. Building a production grid in scandinavia. *IEEE Internet Computing*, 7(04):27–35, 2003.
- [2] Ian Foster. A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.
- [3] L.C. Lazzeroni. A chronology of fine-scale gene mapping by linkage disequilibrium. *Statistical Methods in Medical Research*, 10:57–76, 2001.
- [4] J.S. Liu, C. Sabatti, J. Teng, B.J.B. Keats, and N. Risch. Bayesian analysis of haplotypes for linkage disequilibrium mapping. *Genome Research*, 11:1716–1724, 2001.
- [5] J. Molitor, P. Marjoram, and D. Thomas. Fine-scale mapping of disease genes with multiple mutations via spatial clustering techniques. *Am. J. Hum. Genet.*, 73:1368–1384, 2003.
- [6] A.P. Morris, J.C. Whittaker, and D.J. Balding. Fine-scale mapping of disease loci via shattered coalescent modeling of genealogies. *Am. J. Hum. Genet.*, 70:686–707, 2002.
- [7] D.E. Reich and E.S. Lander. On the allelic spectrum of human disease. *Trends in Genetics*, 17(9):502–510, 2001.