Biological Sequence Alignment On The Computational Grid Using The Grads Framework [*]

Asim YarKhan (yarkhan@cs.utk.edu)
Computer Science Department, University of Tennessee

Jack J. Dongarra (dongarra@cs.utk.edu)
Computer Science Department, University of Tennessee
Computer Science and Mathematics Division, Oak Ridge National Laboratory

Abstract.
In spite of the existence of several Grid middleware projects, developing and executing programs on the computational Grid remains a user intensive process. The Grid Application Development Software (GrADS) project is working to make the Grid easy to use despite the dynamically changing status of Grid resources. Several software packages are being ported to the GrADS framework in order to guide its development. In this paper, we present the work done to Grid-enable a biological sequence alignment package (FastA) and to run it under the GrADS framework. Protein and genome sequence alignment is a basic operation in bioinformatics and tends to be very compute intensive. We discuss the advantages of using GrADS framework for FastA.

Keywords: biological sequence alignment, GrADS project, grid scheduling

1. Introduction

The Grid Application Development Software (GrADS) project [4] is developing a framework to make it simpler to prepare and execute programs on a computational Grid. In order to guide this development, an implementation of the GrADS framework known as GrADSoft [5, 6] is being developed simultaneously with a set of diverse software packages. This set of packages includes the numerical linear algebra library ScaLAPACK [19], the physics/astrophysics problem solving environment Cactus [1, 2, 13] and satisfiability solvers for circuit design.

This paper describes the work done to enable a biological sequence alignment application FastA [18] to run on the GrADSoft framework. The advantages and challenges of a Grid based implementation will be discussed.

[*] This work is supported in part by the National Science Foundation contract GRANT #E81-9975020, SC R36505-29200099, R011030-09, "Next Generation Software: Grid Application Development Software (GrADS)".

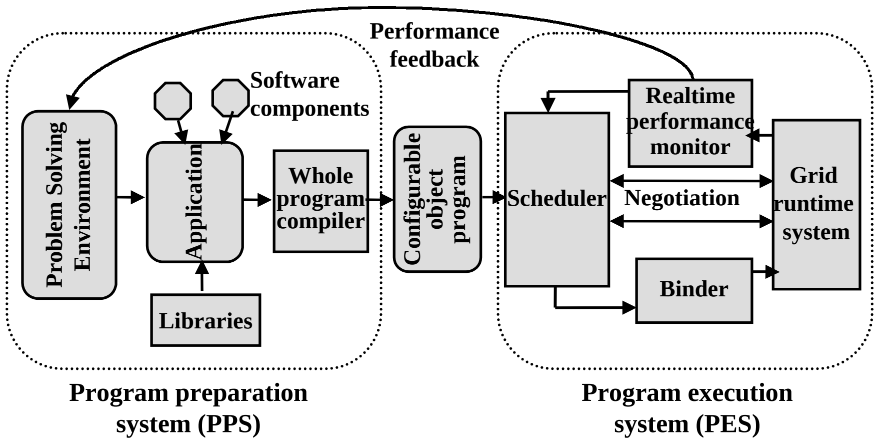
Performance feedback
Software components
Problem Solving Environment
Application
Whole program compiler
Libraries
Configurable object program
Scheduler
Realtime performance monitor
Negotiation
Grid runtime system
Binder
Program preparation system (PPS)
Program execution system (PES)

Figure 1. The Grid Application Development Software (GrADS) Architecture

2. The Computational Grid and the GrADS Project

Several Grid [10] computing infrastructure projects exist to enable the use of geographically and administratively distributed resources, such as Globus [9] and Legion [11]. However, developing and deploying application across these Grid resources remains a user intensive process. Among other tasks, the user is responsible for ensuring that all the supporting framework exists on the Grid resources, that the resources are available and not currently busy, that the connectivity between the resources is sufficient, etc. Additionally, the user needs to track the application to ensure that the Grid resources have not changed so as to cause the application to fail or be unacceptably delayed.

The Grid Application Development Software (GrADS) [4] project is a multi-university research project which works to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified information sharing. The GrADS project intends to provide tools and technologies for the development and execution of applications in a Grid environment. In the GrADS vision, the end user simply presents their parallel application to the framework for execution. The framework is responsible for scheduling the application on an appropriate set of resources, launching and monitoring the execution, and, if necessary, rescheduling the application on a different set of resources. A high-level view of the GrADS architecture [12] is shown in Figure 1.

In the GrADS vision, the **Program Preparation System** (PPS) handles application-development, composition, and compilation. The application source code is manipulated in a high level problem solving environment (PSE) to integrate software libraries and to prepare it for further processing. The compiler analyzes the application and generates a intermediate representation which is stored as the configurable object program (COP). The COP encapsulates all the results of the PPS for later usage, including application specific performance models and data mappers. The COP may used multiple times and can be stored for later use.

The **Program Execution System** (PES) handles resource discovery, scheduling, execution, performance monitoring and rescheduling. In order to execute an application, the user submits the application specific parameters to the GrADS system. The application COP is retrieved and the PES is invoked. The scheduler uses the grid run-time system which is built on top of Globus [9] Monitoring and Discovery Service (MDS) and Network Weather Service (NWS) [25] to determine the availability and status of the appropriate grid resources. The performance model and mapper from the COP are used by the scheduler to determine a good set of resources for the application. The binder compiles the intermediate code to the resource-specific format, and enables the performance monitoring to take place. The application is then launched on the scheduled resources. A real-time performance monitor tracks the application performance on the grid resources, and if the performance contract (i.e., expected performance behavior) [23] is violated, the rescheduler may migrate the application to alternate resources.

The previous section outlined the long-term GrADS vision; as components of this vision are implemented, they are incorporated in a prototype implementation of GrADS called GrADSoft.

3. Biological Sequence Matching and the Grid

Sequence matching is one of the most important primitive operations in computational biology, often forming the basis of more complicated and sophisticated operations. For example, projects that assemble DNA from shotgun sequencing use similarity searches to find overlapping fragments. The two tasks involved in matching sequences are similarity computation and alignment. In similarity computation, a metric is calculated that measures the syntactic difference between two sequences. In the alignment task, the costs of additions, deletions and substitutions required to match one sequence with the other are calculated. The matrix of costs associated with the additions, deletions and substitutions are determined by biologists. Sequence matching tasks often needing to be performed repeatedly over huge protein and genome databases.

The full pairwise sequence matching task is usually solved as using dynamic programming technique, similar to calculating the edit distance between two strings. Well known dynamic algorithms exist to compute the alignment (e.g., Needleman-Wunsch [17], Smith-Waterman [22]), however the computational costs of these algorithms high. With the size of the protein and genome databases growing rapidly, the methods tend to be too slow on traditional computing resources. Many heuristic approaches to speeding up the alignment problem exist, two of the better known are the fast BLAST algorithm [3] and the slightly slower, more accurate FastA algorithm [18]. FastA is approximately 10-50 times faster than the Smith-Waterman algorithm and is often used as a good compromise between speed and accuracy.

Given the size and growth of the protein and genome sequence databases, it is often undesirable to transport and replicate all databases over a wide area network. A Grid approach to sequence matching can keep all the databases at a small number of sites, while bringing the computation to the data. Additionally, since searches over different parts of the reference database can be carried out without any communication between nodes, this is a application is a good fit for a Grid implementation. In this paper we demonstrate one approach to adapting the sequence alignment package FastA [8] to run on a Grid using the GrADS framework.

4. FastA, GrADSoft and Data Locality

The FastA sequence alignment implementation developed and coded by William Pearson [18, 8] has been adapted by us to use remote, distributed databases. These databases may be partially replicated on some or all of the grid nodes. The original MPI master-worker implementation assumed that the reference databases were only available at the master node. In the original implementation, this reference data was communicated in equal sized messages from the master to the workers, so as to provide an approximately equal distribution of the reference data to each worker.

The GrADS version of FastA has the reference databases residing at the worker nodes; the master sends a message informing each worker what portion of which database it should match against. In a Grid scenario, the worker nodes need not be homogeneous, so we schedule varying work loads on the workers in order to minimize the time to completion for the entire query. The master node distributes queries to each worker and collects and collates the results.

Since databases may be located at more than one worker, workers may be assigned only a portion of a database in order to minimize the execution time. FastA provides an interesting scheduling challenge due to the database locality requirements and large computational requirements.

5. Performance Model, Mapper, Performance Contracts

The FastA Performance Model estimates the execution time for the application on a specific set of Grid nodes given measures of various machine and network characteristics (e.g., the free memory, CPU power/availability and network latency and bandwidth). These system characteristics are obtained from Globus MDS and the Network Weather System and may be estimated into the near future. The Performance Model was determined by running a variety of experiments over query sequences and databases. The observed execution times were fitted to a nonlinear model based on the length of the query strings and the databases. Since there is no worker-to-worker communication in this application, the only communication aspect that was the time to distribute the queries to the worker nodes and collect results. This model has been designed to extrapolate beyond the parameters of the original experiments, but it can only be expected to be accurate within those parameters.

A Mapper is an necessary complement for the Performance Modeler. The Mapper allocates different amounts of work to the grid nodes in order to minimize the overall execution time. The amount of work allocated depends on which databases are available on each node and the status of the node and the network. A simple linear approximation to the entire performance model is used to estimate the execution time for different data distributions. A freely available linear solver [15, 24] was used to choose the data distribution so as to minimize the execution time. Figure 2 shows the GrADSoft framework using data locality and system status information in order to make scheduling decisions.

Scheduling in GrADSoft works by presenting trial sets of eligible grid nodes to the application specific Performance Model in order to obtain an estimated execution time. The Performance Model calls the Mapper to define a data distribution over the nodes, and then returns the estimated time required for the problem over this data distribution and using the current system information from NWS. The set of nodes which have then lowest estimated execution time for the problem are used to run the problem. The Scheduler generates the trial sets of nodes using deterministic greedy orderings along with some prior

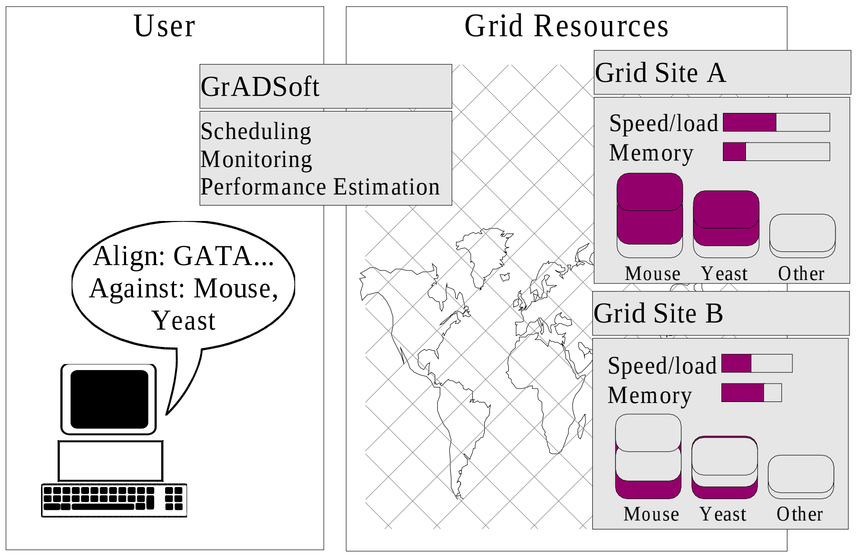

Figure 2. An overview of FastA running on the Grid; system status and database locality are used in making scheduling decisions

network knowledge so that nodes within a single cluster tend to be presented together [5, 7, 6]. Other schedulers can also be used in the GrADSoft framework, for example, experiments have been performed using a simulated annealing scheduler [26] and an exhaustive search scheduler exists.

A key feature of the GrADS architecture is the performance contract which specifies an expected execution performance to be obtained on a set of grid resources. A performance contract can be developed using the application specific performance model which takes into account the capabilities and current state of the grid resources. Since we are dealing with a changing grid environment, there are many circumstances that may cause an application to violate its performance contract. Some possible reasons for failing performance contracts could be that other processes have been launched on the hosts, or the communication links have become crowded. The user is informed via a graphical display of the performance violation, and can query the display to obtain more detailed information about which performance measures have caused a problem (e.g., CPU usage, communication bandwidth, etc.).

6. Executing FastA using GrADSoft

In the preparation phase, the GrADSoft binder component edits the FastA binary to enable performance monitoring. Calls to a performance measurement and monitoring tool AutoPilot [21, 20] are inserted into the binary. These calls spawn a parallel thread which reports the status of the execution to an external AutoPilot manager program. AutoPilot monitors performance information obtained by using the Performance API (PAPI) [14, 16] in addition to other message passing measures. Additionally, in the preparation phase, the Performance Model and Mapper specific to the FastA application are loaded for use by the GrADSoft scheduler.

GrADSoft uses the grid runtime system to obtain current information about available Grid resources. The Scheduler uses this information along with the Performance Model and Mapper to generate a near-optimal schedule for the application. The application is launched on the selected Grid resources, with a master process being run on the first host and worker processes being run on all the other hosts. While the application runs, the AutoPilot monitor thread reports the status to the AutoPilot manager. The external AutoPilot manager can be used to present the user with various views of the performance measures in order to determine if the execution is progressing as expected.

The master process informs each worker about which section of the reference database it is responsible based on the data distribution and schedule that is generated by the Scheduler. Each worker process loads up its section of the reference database into its memory. For each query sequence, the master sends the sequence to each worker, and collects the replies from the worker. The master summarizes and presents the results to the user.

7. Summary

This work was demonstrated at Supercomputing 2002, along with several other demonstrations of the GrADS framework. The GrADSoft architecture greatly reduces the burden on the end user of finding, selecting and using the appropriate grid resources for their project. The GrADS framework provides a plausible method for providing large amounts of computing power to applications on demand. The logistic tasks of moving the application to the site of the data and gathering the results are handled by the framework.

References

1. Gabrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chuang Liu, Thomas Radke, Ed Seidel, and John Shalf. The Cactus Worm: Experiments with dynamic resource discovery and allocation in a grid environment. *International Journal of High Performance Computing Applications*, 15(4):345–358, 2001.
2. Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Nick Karonis, Matei Ripeanu, Ed Seidel, and Brian Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of Supercomputing Conference*, November 2001.
3. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
4. Fran Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman, John Mellor-Crummey, Dan Reed, Linda Torczon, and Rich Wolski. The GrADS Project: Software support for high-level Grid application development. *International Journal of Supercomputer Applications*, 15(4):327–344, 2001.
5. Holly Dail. A modular framework for adaptive scheduling in grid application development environments. Master's thesis, University of California at San Diego, March 2002. Available as UCSD Tech. Report CS2002-0698.
6. Holly Dail, Fran Berman, and Henri Casanova. A decoupled scheduling approach for grid application development environments. *Journal of Parallel and Distributed Computing*, 2003. To appear.
7. Holly Dail, Henri Casanova, and Francine Berman. A Decoupled Scheduling Approach for the GrADS Program Development Environment. In *Proceedings of Supercomputing Conference*, November 2002.
8. FASTA package of sequence comparison programs at `ftp://ftp.virginia.edu/pub/fasta`.
9. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
10. Ian Foster and Carl Kesselman. The Globus Toolkit. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 259–278. Morgan Kaufmann, San Francisco, CA, 1999. Chap. 11.
11. Andrew S. Grimshaw, William A. Wulf, and the Legion team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.
12. K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, R. Aydt, D. Reed, D. Gannon, L. Johnson, C. Kesselman, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a framework for preparing and executing adaptive grid programs. In *16th International Parallel and Distributed Processing Symposium (IPDPS '02 (IPPS and SPDP))*, pages 171–171, Washington - Brussels - Tokyo, April 2002. IEEE.
13. Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and evaluation of a resource selection framework for Grid applications. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, July 2002.
14. K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis using hardware counters.

In *International Conference on Parallel and Distributed Computing Systems*, August 2001.
15. lp_solve FTP site at `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.
16. Philip Mucci. The performance API PAPI. White Paper of the University of Tennessee, http://icl.cs.utk.edu/projects/papi/, March 2001.
17. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.
18. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, 85:2444–2448, 1988.
19. Antoine Petitet, Susan Blackford, Jack Dongarra, Brett Ellis, Graham Fagg, Kenneth Roche, and Sathish Vadhiyar. Numerical libraries and the Grid. *The International Journal of High Performance Computing Applications*, 15(4):359–374, November 2001.
20. Randy L. Ribler, Huseyin Simitci, and Daniel A. Reed. The Autopilot performance-directed adaptive control system. *Future Generation Computer Systems*, 18(1):175–187, September 2001.
21. R.L. Ribler, J.S. Vetter, H. Simitci, and D.A. Reed. Autopilot: Adaptive Control of Distributed Applications. *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing*, July 1998.
22. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
23. Frederik Vraalsen, Ruth A. Aydt, Celso L. Mendes, and Daniel A. Reed. Performance Contracts: Predicting and monitoring grid application behavior. In *Proceedings of the 2nd International Workshop on Grid Computing*, Nov 2001.
24. H.P. Williams. *Model Building in Mathematical Programming.* Wiley, Chichester, New York, second edition, 1995.
25. Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, October 1999.
26. Asim YarKhan and Jack J. Dongarra. Experiments with scheduling using simulated annealing in a Grid environment. *Lecture Notes in Computer Science*, 2536:232–242, 2002.

Address for Offprints: Innovative Computing Laboratory, Computer Science Department, University of Tennessee, 1122 Volunteer Blvd Suite 413, Knoxville, TN 37996-3450, USA, Phone: USA 865-974-8295