A Lightweight, Scalable Grid Computing Framework for Parallel Bioinformatics Applications

Hans De Sterck
Department of Applied Mathematics
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
Email: hdesterck@uwaterloo.ca

Rob Markel
Scientific Computing Division
National Center for
Atmospheric Research
Boulder, Colorado 80307, USA
Email: markelrs@ucar.edu

Rob Knight
Department of Chemistry and Biochemistry
University of Colorado at Boulder
Boulder, Colorado 80309, USA
Email: rob@spot.colorado.edu

Abstract— **In recent years our society has witnessed an unprecedented growth in computing power available to tackle important problems in science, engineering and medicine. For example, the SHARCNET network links large computing resources in 11 leading academic institutions in South Central Ontario, thus providing access to thousands of compute processors. It is a continuous challenge to develop efficient and scalable algorithms and methods for solving large scientific and engineering problems on such parallel and distributed computers. If the computing power available in such computational grids can be unleashed effectively in a scalable way, large scientific problems can be solved that would otherwise be hard to solve using the machines available in a stand-alone way. This paper describes techniques and software developed that allow to apply the power of computational grids to large-scale, loosely coupled parallel bioinformatics problems. Our approach is based on decentralization and implemented in Java, leading to a flexible, portable and scalable software solution for parallel bioinformatics. We discuss advantages and disadvantages of this approach, and demonstrate seamless performance on an ad-hoc grid composed of a wide variety of hardware for a real-life parallel bioinformatics problem. The bioinformatics problem described consists of virtual experiments in RNA folding executed on hundreds of compute processors concurrently, which may establish one of the missing links in the chain of events that led to the origin of life.**

I. Introduction

'Computational grids' [1] (for example, Ontario's SHARCNET, see http://www.sharcnet.ca/about.php) are spatially distributed heterogeneous collections of computing resources. If the computing power available in such grids can be unleashed effectively in a scalable way, large scientific problems can be solved that would otherwise be hard to solve using the machines available in a stand-alone way. Loosely coupled problems are especially suitable for grid computing. In particular, many loosely coupled bioinformatics problems pose a dire computational challenge, both because the amount of data to be processed can be huge, and because the processing algorithms can be extremely complex. In this paper, we describe the TaskSpaces [2] software framework for scientific computing on computational grids, and its application to parallel bioinformatics problems. Some of the technical aspects of this work are described in more detail in [2], [3], and more detailed biology results are presented in [4].

Since a few years, computational grids [1] are being built in many places in the world, including the US (e.g., the TeraGrid, see http://www.teragrid.org), Europe (e.g. the Enabling Grids for E-science in Europe project, see http://egee-intranet.web.cern.ch/egee-intranet/gateway.html), and Canada. SHARCNET, at present, links 11 leading academic institutions in South Central Ontario. Loosely coupled problems can be solved on grids in a cross-platform manner, distributed in space and time. Many efforts to realize the concepts of grid computing at the software level are now underway [1], both in academic and industry settings (e.g., the Globus project, see http://www.globus.org).

In recent years, parallel and distributed computing techniques have steadily been gaining popularity for tackling difficult bioinformatics problems. Two important reasons for the use of parallel techniques can be identified easily. First, bioinformatics problems involve increasingly large datasets. Second, the algorithms used in many bioinformatics applications can be computationally prohibitive. For many bioinformatics problems it is therefore clear that computations limited to a single CPU cannot deliver the required computing power, and that parallel and distributed computing approaches are therefore necessary. A large class of bioinformatics problems can be parallelized easily, with minimal or no interprocess communication. These types of problems are called loosely coupled, and they are especially suitable for distributed processing. More tightly coupled problems require intensive interprocess-communication. Efficient parallel and distributed computing is typically more challenging for this type of problems. In the present paper we discuss both types of problems.

Consider, for example, the case of a university researcher who is confronted with a complex bioinformatics problem. The researcher typically has access to a wide range of computational resources on different scales. These resources include desktop machines that may be available in the researcher's own lab (of the order of 10 CPUs or so), PC clusters that may be available at the department level (order 100 CPUs), parallel computers that may be available in the university's computer center (order 100-1000 CPUs), and pools of large clusters and parallel supercomputers (up to several thousand CPUs) that can be accessed through networks like SHARCNET, or at national supercomputer centers such as the US National Center for Supercomputing Applications (NCSA) and the San Diego Supercomputer Center (SDSC). In this paper, we propose an approach to parallel bioinformatics that, ideally, allows the researcher to develop the bioinformatics software locally on a single PC. Then, depending on the size of the problem at hand, the task can be distributed seamlessly over any or all of the wide variety of machines available.

This 'universal computing dream' is hard to realize for several reasons. The hardware, operating systems (and versions of operating systems), supporting software, and queueing systems may all vary among available machines. The researcher will wonder how to install and maintain code on all these machines, how to distribute tasks and data, how the results will be collected and centralized, and so forth. Scripts that automate some of these tasks will be brittle when software is upgraded, or machines are added or removed. In the light of these obstacles, the 'universal computing dream' seems little more than an ever-receding mirage.

However, in this paper we describe TaskSpaces, a system we developed that demonstrates that many components of the 'universal computing dream' can be realized on today's infrastructure using grid computing. The grid computing concept can be easily understood by considering the analogy with a power grid. A power grid user accesses the grid in order to obtain electrical power, which is an interchangeable commodity. Indeed, the user's machines do not care where or how the power they use is produced (the user may have ethical concerns that affect the desirability of particular power sources, but, to the hardware, all electricity is equivalent).

Two crucial properties make the power grid work:

1) The grid can be accessed through a standard interface.
   In the case of a power grid, the standard interface is simply the electrical plug, which gives access to the power grid that operates at standard voltages and frequencies.
2) The grid is scalable.
   This scalability works both from the user's side (the user can access more power as needed), and from the power producer's side (the grid operator can switch in additional power generators as demand rises).

Ideally, grid computing would work in exactly the same way: a user accesses the geographically distributed grid in order to obtain CPU cycles, which are considered an interchangeable commodity (the user does not care where the computing cycles are produced) (Fig. 1). Unfortunately, accessibility through a standard interface (the first of the two essential properties of a grid) can be difficult to achieve with computers. In TaskSpaces, the standard interface is provided by the Java virtual machine, which is almost universally available. Java behaves almost exactly in the same way on all those machines, and Java's 'executable byte-code' is, in theory, fully interchangeable between machines. In TaskSpaces, the second essential grid property, scalability, is realized through the concepts of 'bag-of-tasks' computing and tuple spaces. Consequently, each user can submit many tasks concurrently, and the grid operator can switch in additional compute farms when demand is high.

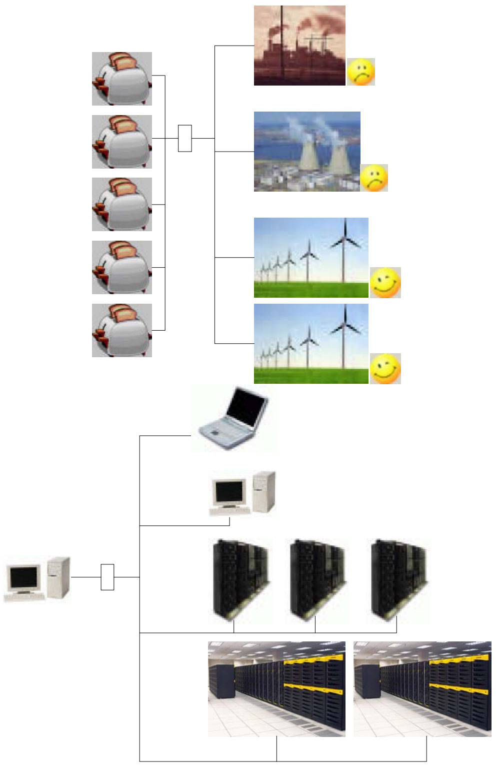
Fig. 1. Analogy between a power grid (top) and a computational grid (bottom). Both exhibit scalability from the user's and the producer's sides, and need to be accessible through a standard interface.

The analogy between computational and power grids is not perfect: computing cycles and data are more complex than electrical power units. We can identify the following additional requirements for computing grids, some essential and others pragmatic.

3) Information is not interchangeable, and must often be kept confidential (unlike electrical power). The grid must allow secure resource sharing.
4) Information is not easily replaceable (unlike electrical power). The grid must provide fault-tolerance mechanisms such as transactions.
5) Parallel computers use many different queueing systems. The grid must provide resource allocation and scheduling.
6) Large problems may require deployment on heterogeneous hardware and software. The grid must provide a mechanism for distributing the application code transparently to the machines on which calculations are ultimately performed.
7) Many problems require interprocess communication. The grid must allow efficient communication between processes.
8) Computing resources are expensive. The grid must allow users to be billed according to cycle usage.
9) Some problems require specific turnaround time, data

transfer bandwidth, fault-tolerance, etc. The grid may need to provide quality-of-service guarantees.

10) Problems must be connected with computing resources. Either the grid must allow the user to discover resources, or the grid must be able to discover tasks as they are presented (TaskSpaces uses the latter approach, resembling a real power grid).

Many efforts to realize the concepts of grid computing are now underway. Some projects, such as Globus, try to define standards for what eventually may become a worldwide, unified, computational grid ('The Grid'), very much along the lines of 'The Internet' and 'The World Wide Web'. However, many of the difficulties summarized above are still far from being resolved in a general, satisfactory way, and it is not clear that generally usable standards for grid computing will become available and accepted soon. Therefore, we have developed TaskSpaces, a software framework for a smaller-scale computational grid. TaskSpaces is based on the design criteria of decentralization, provided by an underlying tuple space concept, and platform independence, provided by implementation in Java. Our goal was to produce a lightweight grid environment that is easy to install and operate, and to demonstrate that it can be used efficiently for real-world parallel bioinformatics problems. In this effort, we have attempted to deal with some, but not all, of the challenges listed above. Besides providing an environment for solving real bioinformatics problems on small, 'privately operated' grids, we hope that our experiences may reveal some methods of overcoming the challenges mentioned above, and that these methods may become more generally useful in guiding standards adopted for larger grids. At present, many different approaches are being tested on small-scale, privately operated grids, both in research and commercial settings. The successful approaches will survive, and, driven by demand and cost savings through efficiency gains, these privately run grids may eventually become connected to form a World Wide Grid, very much like national power grids are presently connected to neighboring grids throughout most of the world.

The rest of this paper is organized as follows. The next Section describes TaskSpaces, our prototype software framework for grid computing, which we based on tuple space concepts and implemented in Java. Section III describes a loosely coupled parallel bioinformatics application that we investigated on a computational grid, namely the problem of finding correctly folded RNA motifs in sequence space. Section IV describes our experience with operating the software framework on a computational grid composed of local workstations and parallel clusters at supercomputer centers. Brief results for the RNA motif problem are presented in Section V. The paper concludes with a Section on future work, and a summary.

II. The TaskSpaces Framework

TaskSpaces is a prototype lightweight grid computing framework for scientific computing characterized by two major design choices: decentralization, provided by an underlying tuple space concept, and object-orientation and platform-independence, provided by implementation in Java. The TaskSpaces framework has been described in full detail in [2]; in this Section we summarize its main properties.

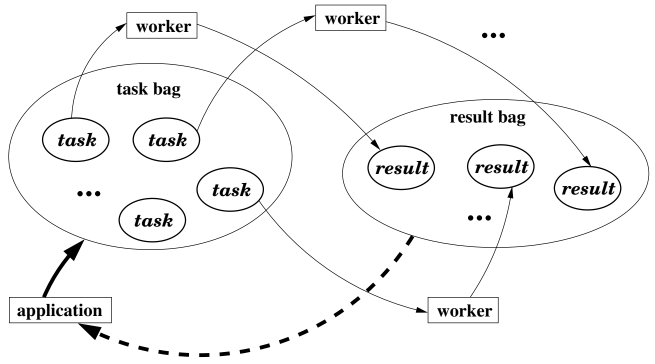

Fig. 2. The tuple space paradigm for distributed computing.

Tuple spaces were pioneered in the late 1970s, and were first realized in the Linda system and language [5]. In a tuple space distributed computing environment, processes communicate solely by adding tuples to and taking them from a tuple space, a form of independent associative memory. A tuple is a sequence of fields, each of which has a type and contains a value. Fig. 2 shows conceptually how distributed computation works in a tuple space environment. An application program places subtasks resulting from the partitioning of a large computational problem into a tuple space (which in the Bag-of-Tasks paradigm is called a 'task bag' [6]), in which each subtask is represented as a tuple. 'Worker processes' take the task objects from the task bag, execute the tasks, and place the result in a 'result bag' as another tuple. The tuple space concept allows tasks to be decoupled both in space and time. The distributed computing process is decoupled in space, as the application, task and results bags, and the various worker processes may reside on a heterogeneous collection of machines that are connected by a network but that are otherwise widely geographically distributed. This decoupling allows flexible topology for the computation, permitting automatic configuration based on the availability of worker processes. The distributed computing process is also decoupled in time: since spaces are persistent, tuples are persistent while resident in the space, and processes can access tuples long after the depositing process has completed execution.

Fig. 3 shows a conceptual deployment diagram of the TaskSpaces framework. TaskSpaces uses an event-driven model. On startup, worker processes register with a task bag. The application process sends subtask objects to the task bag, and the task bag sends those task objects to available workers. The task bag acts as a 'superqueue', and thus alleviates the problem of scheduling when multiple supercomputers with different unsynchronized queueing systems are used. Scalability is inherent because users may put several applications in the task bag at the same time, and the grid operator can add 'worker farms' when needed. After a task is processed, the worker puts a result object in the result bag, from which the result objects are collected for final assembly by the application. TaskSpaces is implemented in Java, providing a standard, platform-independent interface to the grid system and exploiting Java's built-in networking and security features.

The TaskSpaces code consists of several classes. All classes, except for the Runner class, are served to participant machines via HTTP servers. Configurable properties files which contain system information and parameters are also served by HTTP servers. For a more detailed description of the class structure of the TaskSpaces framework, see [3] and [2]. Application code need not be installed and maintained on workers, because it is downloaded from a central server when task objects arrive at each worker. Installing and executing a Java bytecode executable of size < 2kB allows any worker host to participate in the grid. Thus, installation and maintenance of TaskSpaces is extremely lightweight and easy. In fact, the complete TaskSpaces codebase is extremely small and compact, due to the simplicity of the design, and the availability of Java's built-in networking and object manipulation capabilities.

TaskSpaces can be used in taskfarming mode for problems that do not require interprocess communication, such as independent folding of many RNA sequences (see below). It can also be used for other applications that do require interprocess communication, handling such communication in a scalable way by transmitting serialized Java objects over sockets. TaskSpaces scales well on large grids composed of supercomputers at NCSA, SDSC, and other supercomputer centers, connected over the internet, for a parallel computing problem in numerical linear algebra [2]. This problem requires neighbor-neighbor interprocess communication, and it is thus surprising that the scalability for this problem in the heterogeneous grid environment is so good.

Looking back at the prerequisites we set out in the previous Section for the 'universal computing dream' we pursue, it is instructive to consider how our prototype grid implementation

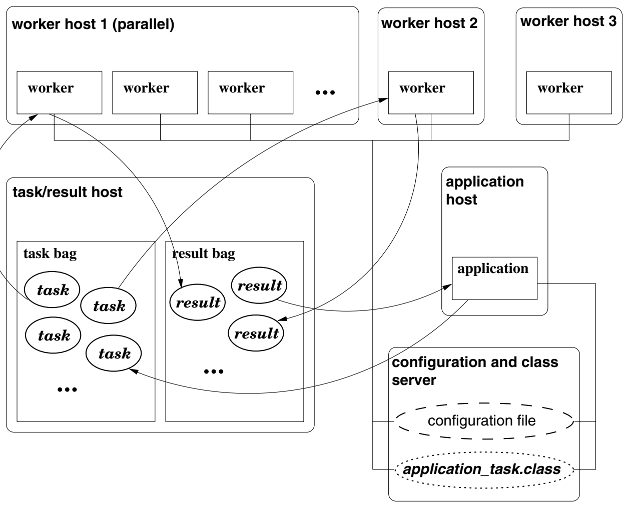

Fig. 3. TaskSpaces framework deployment diagram.

performs with respect to our aspirations. Some of the functionality is only present in a rudimentary way in our prototype implementation, but more sophisticated versions based on the general concepts presented can easily be imagined.

1) Standard interface: YES.
Through implementation in Java. In the strict sense this limits the applications to code written in Java, but, with limited sacrifices in generality, application code in other languages can be used as well (see below).
2) Scalable: YES.
Through the tuple space concept. Scalability from the producer side is currently performed 'by hand', but automated strategies can easily be imagined. Also, bags can in principle be replicated when access loads become high and bottlenecks arise, and automatic strategies to this end can be considered as well.
3) Secure resource sharing: not implemented yet in TaskSpaces.
But definitely feasible using Java's built-in mechanisms of digital signatures and public-private key cryptography.
4) Fault-tolerance: not implemented yet in TaskSpaces.
But, for instance, automatic duplication of bags for backup reasons could easily be achieved via simple cloning of Java objects.
5) Resource allocation and scheduling: YES.
The task bag acts as a 'superqueue'.
6) Automatic distribution of application code to worker machines: YES.
By downloading Java objects from the task bags. The objects contain both the data and references to the application code, which is downloaded automatically from the class server upon first use by a worker.
7) Scalable interprocess communication: YES.
Through direct exchange of serialized Java objects over sockets between workers, see also [2]. Efficient collective communications would require additional features such as multi-level communication schemes (see below).
8) User charging algorithms: not implemented yet in TaskSpaces.
Simple charging strategies are straightforward to implement.
9) Quality-of-service: not implemented yet in TaskSpaces.
This may require thorough study of the particular grid environments considered, and instrumentation of objects and worker machines with performance measures and priority mechanisms.
10) Resource discovery: YES.
Computing resources discover tasks by making themselves available to the task bags, rather than the other way around. Compute farms are presently assigned to task bags by hand, but automatic, multi-level assignment strategies are feasible.

The overview above shows that the TaskSpaces design, despite its simplicity, is quite effective in realizing many of the conceptual aspirations of the 'universal computing dream'.

In the following Sections, we illustrate how the framework, with minimal effort, can be used for a practical, real-life parallel bioinformatics application on ad-hoc computational grids composed of a variety of widely available hardware types.

III. Application: Finding Correctly Folded RNA Motifs in Sequence Space

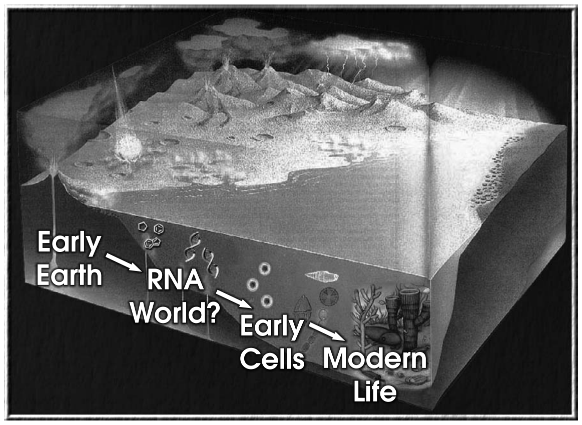

Fig. 4. The RNA world hypothesis: if a small number of random RNA molecules, say a pool of 10^6 to 10^9 sequences, has a reasonable probability of containing molecules with various chemical functions, then primitive metabolisms would be expected to have arisen many times on the early Earth.

We have applied the TaskSpaces framework to the following problem, which is relevant both to natural evolution and to a process of artificial evolution called SELEX that has been widely used to select new molecular functions from random pools of RNA. Given a pool of random RNA molecules of a specified length (typically 50-200 bases), what is the probability that the random pool contains molecules that have the right sequence and are folded into the right structure needed for a particular chemical function? This question is critical for the RNA world hypothesis: if molecules that can catalyze a particular reaction are especially common, the idea that the tiny amounts of RNA that would be produced by prebiotic synthesis could produce an RNA metabolism becomes more plausible (Fig. 4) [7]. Chemically active RNA molecules are also routinely synthesized, in SELEX laboratory experiments, from intially random pools of RNA molecules [4]. Specifically, we have focussed on determining the abundance of isoleucine and hammerhead RNA motifs in random molecules [4]. The isoleucine motif is the shortest RNA motif capable of binding specifically to the amino acid isoleucine, while the hammerhead motif cuts RNA at specific locations and has been found both in cells and through SELEX. It has been determined experimentally that chemical function of a certain type appears when the random RNA molecule contains a prescribed motif, which is composed of several modules with partially specified sequence, and has a prescribed folding structure (see Fig. 5). The probability that a random molecule matches both the prescribed sequence and the structure, $P(seq, struct)$, is calculated in two steps as $P(seq, struct) = P(seq)\ P(struct|seq)$. The sequence probability $P(seq)$ can be approximated accurately by combinatorial formulas [7], [4]. The conditional probability of obtaining the right folding structure, given a partially random molecule that contains the right sequence, cannot be approximated analytically. In stead, we approximate this probability by computational folding of large samples of RNA molecules (a sample size of 10,000 is typically used): the probability is approximated by the number of partially random molecules that fold into the correct structure, divided by the total number of molecules in the sample. One important question of interest is the variation of the probability $P(seq, struct)$ as the composition of the random pool changes, since the composition of RNA pools may have varied widely on the primitive earth and since modern genomes vary widely in composition, possibly affecting the evolution of specific functions. We set out to investigate whether specific kinds of chemical function arise more often in pools with overall composition biases in particular directions. This required the computational folding of many samples in $\{A, C, G, U\}$ composition space. We used 5% intervals in composition space, leading to 969 different compositions to be tested. Varying the length of the random molecules (we have considered lengths of 50, 100, and 150 nucleotides), further increased the number of foldings required. For the results to be discussed briefly below (see [4] for a more detailed discussion), we performed about hundred million computational foldings. This constitutes a computational problem of moderately large size, which would require weeks to months on a single fast workstation. We decided to use a grid computing approach, mainly for flexibility, portability and scalability reasons.

We used the Vienna RNA folding package [8], which is written in C, for folding individual sequences. The RNAfold executable is called by the Java application on each worker node as needed. Non-Java executables must be compiled in advance for each worker architecture, and can be downloaded from the code server by the workers upon first use. Thus, although reliance on code written in other languages increases the effort required for cross-platform operation, it is still feasible.

IV. Case Study: Operating the Framework on a Computational Grid

We simulated the RNA function probability problem on a grid composed of the NCSA IA32 Linux Platinum Supercluster and various P4 Linux workstations at CU Boulder, Colorado. The Platinum machine features 968 P3 compute processors (1GHz). For code development and execution of some smaller subproblems, only the local workstations were used, while for larger problems the local workstations were combined with up to 200 Platinum processors concurrently. The total computing time used for this project so far, including extensive initial runs for exploring the problem and determining the right approach and questions to be answered, amounts to approximately 10,000 Platinum processor hours.

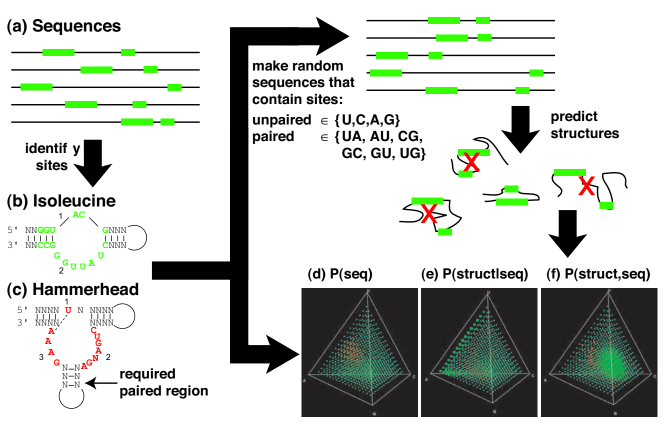

Fig. 5. Procedure for determining the effects of folding and sequence composition on motif abundance. (a) the motifs are identified by comparing sequences with the same function. The isoleucine aptamer (b) and the hammerhead ribozyme (c) both consist of modules that must have an exact sequence, and flanking helices that must base pair but need meet no other constraints. These diagrams show the exact sequence and structure requirements that were used in the calculations: base pairs are indicated by connecting lines. We calculate Pr(sequence) (d) from the sequence requirements, and Pr(structure|sequence) (e) by constructing large samples of random sequences that contain the motif and computationally predicting their structures. The overall probability of finding a correctly folded sequence (f) is obtained by multiplying the probabilities from (d) and (e).

The framework was easy to install on candidate worker machines. Even though Java is normally not thought of very much as a language for supercomputing, it is actually available on all machines we obtained access to, even the largest parallel supercomputers. In fact, Java is catching up fast in execution speed with other languages, and the advantages in ease of use and portability may actually give it a good future in scientific computing. Locating the Java executable (which is typically not included in the standard path), copying the Java worker bytecode to the worker machine, and starting the workers, was typically very fast: for most machines it did not take more than 15 minutes to make them participate in the grid. On parallel computes, the standard queueing systems were used. Varying queue delays on concurrently participating machines did not cause a problem, because the taskbag (typically located on a workstation in Boulder) acts as a superqueue, and the RNA folding tasks are loosely coupled and do not require any interprocess communication and synchronization. The Internet was used as network connection between the grid machines, and network performance was adequate at all times.

A major obstacle in constructing ad-hoc grids like this is security, which will become increasingly important as research networks and institutions are increasingly targeted by malicious intruders. Under pressure from malicious attacks, potential worker machines will often be protected by firewalls. Participation in a grid then requires additional firewall configuration, as our framework requires at present worker nodes with externally accessible IP addresses. Security is another reason why we expect, as argued before, that grids will develop as 'islands' for the foreseeable future, further delaying the concept of a 'World Wide Grid'. Another inconvenience in operating a grid is the variety of queueing systems operating on parallel computers and clusters. If machines were available where TaskSpaces workers would be the only, continuously running processes, then much of the queueing considerations could be dealt with in more efficient ways that decrease turnaround times, for instance by extending the use and functionality of task bags as superqueues.

We can summarize our experiences with operating the grid framework for a real problem on a real moderately sized grid, by saying that the framework mostly delivered the promised flexibility, portability and scalability. TaskSpaces is currently being deployed on SHARCNET, applied to the bioinformatics problem described here, and to additional bioinformatics applications.

V. Results for the RNA Motif Problem

We estimated the abundance of two motifs, the hammerhead ribozyme and the isoleucine aptamer, in random-sequence pools of many compositions and several lengths. These two well-studied motifs provide test cases for our code on the TaskSpaces framework, with which we plan to analyze dozens to hundreds of motifs. Knowing where particular kinds of RNA sequences are most likely to be found in the space of possible compositions, and where these sequences are most likely to fold into the correct structure if they are found, will provide striking new insight into the conditions under which particular RNA activities can evolve.

To test the effects of nucleotide composition on the probability of meeting the sequence requirements and the probability of correct folding, we generated 10,000 random sequences at each of the 969 possible 5% intervals of sequence composition. The sequences were of total length 50, 100, and 150 nucleotides, meeting the sequence requirements for each of the hammerhead and isoleucine motifs. We repeated the analysis for sequence length 50 allowing G-U base pairs (a weaker type of pairing than the more familiar 'Watson-Crick' G-C and A-U base pairs, which are found at a small but not negligible frequency in biological RNA structures). Thus we folded a total of 77,520,000 sequences for this experiment.

We found that the composition of the randomized sequences had a striking effect on both the probability of finding each motif and the probability of correct folding. Table I compares the estimates, including folding, from this work with our previous estimates of the abundance of the same motifs [7] based on a simplified, not computationally intensive approach. For a more detailed description of our results, see [4].

Our results demonstrate striking relationships between nucleotide composition and the probability of finding specific sequences, and suggest that we may be able to predict which kinds of random-sequence pools (for SELEX or in organisms) might be most able to evolve particular functions. The probability of finding the specific functions we searched for (10^{-8} to 10^{-12}) are rather lower than we had predicted from previous work, demonstrating that the effects of folding are important

TABLE I
SINGLE-SEQUENCE MATCH PROBABILITIES AND POOL SIZES REQUIRED FOR 50% ABUNDANCE COMPARED BETWEEN THIS ANALYSIS AND OUR PREVIOUS CALCULATIONS [7].

Motif		New	Old
Ile	Probability	1.7×10^{-10}	3.3×10^{-6}
	Num Seqs	4.5×10^{9}	2.1×10^{5}
HH	Probability	3.4×10^{-11}	6.2×10^{-11}
	Num Seqs	2.1×10^{10}	1.1×10^{10}

and cannot be ignored. These figures are consistent with the observation that new RNA activities are routinely isolated in the laboratory from random-sequence pools of 10^{12} to 10^{15} molecules, although they do not provide support for the idea that an RNA metabolism could have arisen from only a few hundred thousand random RNA molecules as might have been present on the prebiotic Earth. Due to the chemical problems in synthesizing large amounts of RNA without enzymes, it has often been suggested that a simpler self-reproducing system preceded the RNA World. However, once RNA was first synthesized (perhaps for an entirely different reason), our results show that catalytic activity would soon be likely to emerge: 10^{15} 100-nucleotide RNA molecules is about 50 micrograms of RNA, less than the amount of RNA found in a single gram of modern tissue.

VI. Future Work

There are many interesting ways in which the framework can be extended. First of all, we are planning to build full Python language functionality into the framework to allow researchers familiar with that language to scale their single-CPU tasks easily to the grid. Python is becoming increasingly popular as a language for bioinformatics, mirroring its success for other scientific computing tasks. Second, as indicated in the enumeration in Section II, the framework implementation needs to be extended with regards to scalability, fault-tolerance, security, charging algorithms, and quality-of-service. For example, fault-tolerance may be enhanced by cloning of objects and bags, and transaction-type communication. Third, we plan to add more extensive functionality, in terms of support for complex parallel workflows, connection with databases for data furnishing and result collection, and multi-level tree-based collective communication for tightly-coupled parallel applications.

On the parallel bioinformatics application side, additional loosely coupled parallel bioinformatics applications will be studied, including variants of the previously considered RNA folding statistics problem (for instance, investigation of the effect of the length of the molecules on correct folding), and an examination of whether certain compositional features of ribosomal RNA are universal across organisms or across RNA molecules. We are also considering more challenging applications, including proteomics workflows and tightly coupled problems such as building large phylogenies.

VII. Conclusion

We have described a software framework for scientific computing on computational grids that is based on tuple-space principles and implemented in Java, and we have demonstrated that seamless simulation on an ad-hoc grid composed of a wide variety of hardware is feasible for real-life parallel bioinformatics problems. The language and general approach we used is most appropriate in cases in which flexibility and ease of configuration outweigh concerns about extracting maximal performance on a given architecture for a given, fixed, application with fixed, large problem size that must be executed repeatedly. In this latter situation, it is often a good investment to develop specific optimized software solutions of 'high-performance computing' type. Also, when a single, large computing platform is available that fulfills all computing resource needs and remains in a stable configuration over a long time, a project may not need simulation software that can be deployed on a heterogeneous, distributed network. In many situations, however, research is dynamic, available computing resources may be heterogeneous and spatially distributed, and research goals and directions may change continuously. In such a rapid-prototyping environment with wide variations in problem sizes, with complex changing workflows, and with fast variations in application code, a platform-independent 'high-throughput computing' grid solution of the type proposed in this paper may be most appropriate, because of the gains in flexibility, portability and cross-platform scalability. The performance and applicability of the TaskSpaces framework within the SHARCNET environment is currently being evaluated, and results will be reported in future work.

Acknowledgment

This work was partially supported by the US National Computational Science Alliance under grant MCB020011 and utilized the NCSA IA32 Linux Supercluster Platinum.

References

[1] I. Foster and C. K. (eds.), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[2] H. De Sterck, R. Markel, T. Pohl, and U. Rüde, "A lightweight Java Taskspaces framework for scientific computing on computational grids," in *Proceedings of the ACM Symposium on Applied Computing, Track on Parallel and Distributed Systems and Networking*, 2003, pp. 1024–1030.

[3] H. De Sterck, R. Markel, and R. Knight, "Taskspaces: A software framework for parallel bioinformatics on computational grids," in *Parallel Computing in Bioinformatics and Computational Biology*, A. Zomaya, Ed. John Wiley and Sons, 2005.

[4] R. D. Knight, H. De Sterck, R. S. Markel, S. Smit, A. Oshmyansky, and M. Yarus, "Abundance of correctly folded RNA motifs in sequence space, calculated on computational grids," *Nucleic Acids Research*. Submitted, 2004.

[5] D. Gelertner, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[6] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*. Boston: Addison Wesley, 2000.

[7] R. Knight and M. Yarus, "Finding specific RNA motifs: function in a zeptomole world?" *RNA*, vol. 9, no. 2, pp. 218–30, 2003.

[8] I. Hofacker, W. Fontana, P. Stadler, L. Bonhoeffer, M. Tacker, and P. Schuster, "Fast folding and comparison of RNA secondary structures," *Monatshefte Fur Chemie*, vol. 125, no. 2, pp. 167–188, 1994.