Using a Tuple Space Web Service for Parallel Processing in Bioinformatics

BARBARA MUELLER and LOÏC SCHULÉ
Swiss Institute of Technology (EPFL)
and
GEORGE C. WELLS
Rhodes University

This paper discusses the use of a Tuple Space Web Service for the construction of parallel processing applications in the field of bioinformatics. This system is based on Linda, which is a coordination language for parallel and distributed processing. The use of a coordination language greatly simplifies the parallelization of bioinformatics applications. Since this system is conceived as a web service, it provides language portability, has a simple and easily understandable architecture and does not depend on a dedicated network of workstations.

To illustrate the utility of the tuple space web service, a DNA string searching application has been developed. The results of this application show very favorable performance benefits arising from the use of the web service on a network of commodity workstations. The speedup measured for the application on 30 processors was 29.1 times the one measured for one processor, which means that the execution time was reduced by 97.1%.

General Terms: Novel Computational Tools and Techniques; Pattern Recognition; Genomics and Proteomics;

Additional Key Words and Phrases: Tuple Space; Web Service; Linda; Parallel Processing; Bioinformatics Applications;

1. INTRODUCTION

The project described in this paper is an in-depth investigation of the suitability of a tuple space web service for the development of parallel bioinformatics applications and continues the work done by Wells et al on Java and Linda for Parallel Processing in Bioinformatics (full details can be found in [Wells and Akhurst 2005]). Our web service is modeled closely on the standard Linda model, which has a highly desirable simplicity for writing parallel or distributed applications. As a coordination language the Linda model is responsible solely for the coordination and communication requirements of an application, relying on a host language for expressing the computational requirements of the application.

Much of the early software development in the field of bioinformatics has been done using scripting languages such as Perl. Since the implementation of the web service uses XML to communicate between the client and the server, it allows language portability as long as the client language supports XML and HTTP connections. In other words, the language used to implement bioinformatics applications can be freely chosen by the user and does not have to be the same as the implementation language used by the web service itself.

The facts that the web service allows language portability and that it is very suitable for parallel or distributed applications are the main advantages of this project and make it highly appealing for bioinformatics applications.

1.1 Roadmap

Section 2 briefly introduces the programming language Java and gives an overview of the Java tools used in this project. Section 3 introduces the field of bioinformatics and gives an overview of some basic molecular biology. Section 4 summarizes the main characteristics of the Linda programming model. This is followed by a description of the implementation of the tuple space web service in section 5. Finally, section 6 describes the parallel motif searching application which illustrates the performance of the tuple space web service. To sum up, the last two sections present and discuss the results and provide final conclusions.

Author Addresses:
Barbara Mueller, Department of Computer Science, P. O. Box 94, Grahamstown 6140, South Africa; barbara.mueller@epfl.ch
Loïc Schulé, Department of Computer Science, P. O. Box 94, Grahamstown 6140, South Africa; loic.schule@epfl.ch
George C. Wells, Department of Computer Science, P. O. Box 94, Grahamstown 6140, South Africa; G.Wells@ru.ac.za

2. JAVA

Java is a widely-used, general-purpose programming language, developed by Sun Microsystems in the mid-1990s. It has rapidly become the programming language of choice in many different areas, and has been the subject of extensive language and library development. In particular, it has strong support for string-handling and pattern-matching, provided by specialized libraries. Moreover, it simplifies web service development by providing tools to create HTTP clients as well as reading and writing XML data.

During the early 1990's, developments have led to increasing interest in Java as a language for scientific high-performance computing (HPC) [Pancake and Lengauer 2001], [Bull et al. 2001], [Villacis 1999]. One of the major strengths of Java is its portability, allowing Java programs to be executed on different hardware and operating system platforms without the need for recompilation. All that is required is a Java Virtual Machine (JVM), or interpreter, for the specific combination of hardware and operating system being used.

We used the beta version of Java 1.6 to implement the web service, since the newly added package `javax.tools` provides interfaces to compile code at run time, which is required for some of the advanced features of the web service. However, Java version 1.5 is sufficient to implement a client application. The large number of libraries that are provided in Java, e.g. the support for string-handling and pattern-matching, as well its portability make Java a highly appealing language for bioinformatics applications. But as mentioned before, Java is not the only option for client applications since the web service is langage independent and hence applications written in other languages can easily be adapted to this web service.

3. BIOINFORMATICS

Bioinformatics is a field arising from the application of computers to various biological problems, particularly with regard to research on genes and proteins. A recent paper by Cohen provides a concise overview of the current state of bioinformatics, and its relationship to both biology and computer science as the foundational disciplines [Cohen 2005].

The following definition for the term *bioinformatics* has been proposed:

> Bioinformatics is conceptualising biology in terms of molecules and applying 'informatics techniques' to understand and organise the information associated with these molecules, on a large scale. In short, bioinformatics is a management information system for molecular biology [Luscombe et al. 2001].

Many bioinformatic applications deal with DNA or protein sequences, hence string comparisons and string searching algorithms are essential in bioinformatics. There is, of course, much more than this to bioinformatics, which includes many other categories of problems, such as deriving three-dimensional protein structures, etc. Further information may be found in the many references on this subject, such as [Gibas and Jambeck 2001].

While Java has been used in some bioinformatics applications, and is supported by the development of the BioJava libraries[1] [2], to date much of the software development in this area has used scripting languages such as Perl[3]. While such scripting languages are relatively simple to use and have strong support for string-matching operations and for combining other applications, they suffer from limitations such as poor performance due to their interpreted nature and a lack of support for parallel and distributed programming. This has led to growing interest in Java for biological applications.

Notable among these is the work of Sheil [Sheil 2003], who developed a system to predict the secondary structure of selected proteins using artificial neural networks in parallel. This system utilized Java and the JavaSpaces implementation of Linda, and hence has much in common with our investigation. Wells [Wells and Akhurst 2005] investigated the identification and location of specific patterns in DNA that correspond to proteins of interest, which is also the focus of the research conducted in this project. This involves the application of pattern matching techniques to strings representing both the DNA sequences and the *motifs* corresponding to the protein sequences of interest. Given the large volumes of data involved (the complete human genome comprises about 4GB of data when represented as a text string), parallel programming becomes a desirable, even necessary, technique for improving the performance of such bioinformatics applications.

4. THE LINDA PROGRAMMING MODEL

The Linda programming model has a highly desirable simplicity for writing parallel or distributed applications. As a *coordination language* it is responsible solely for the coordination and communication requirements of an application, relying on a *host language* for expressing the computational requirements of the application. The

[1]http://www.biojava.org/
[2]http://java.sun.com/developer/technicalArticles/javaopensource/biojava/
[3]http://www.bioperl.org/

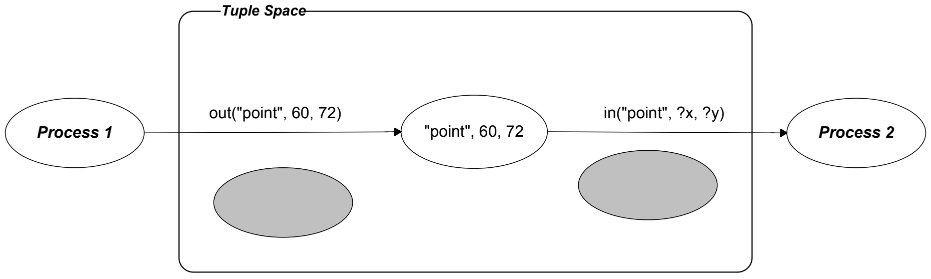

Figure 1. *A Simple One-to-One Communication Pattern.*

Linda model comprises a conceptually shared memory store (called *tuple space*) in which data is stored as records with typed fields (called *tuples*). The tuple space is accessed using five simple operations:

`out`	Outputs a tuple from a process into tuple space
`in`	Removes a tuple from the tuple space and returns it to a process, blocking if a suitable tuple cannot be found
`rd`	Returns a copy of a tuple from the tuple space to a process, blocking if a suitable tuple cannot be found
`inp`	Non-blocking form of in, returns an indication of failure, rather than blocking if no suitable tuple can be found
`rdp`	Non-blocking form of rd

Note that the names used for these operations here are the original names used in Yale's Linda research.

Input operations specify the tuple to be retrieved from the tuple space using a form of *associative addressing* in which some of the fields in the tuple (called an *antituple*, or *template*, in this context) have their values defined. These are used to find a tuple with matching values for those fields. The remainder of the fields in the antituple are variables which are bound to the values in the retrieved tuple by the input operation (these fields are sometimes referred to as *wildcards*). In this way, information is transferred between two (or more) processes. A simple one-to-one message communication between two processes can be expressed using a combination of out and in as shown in Figure 1.

In this case `('point', 60, 72)` is the tuple being deposited in the tuple space by Process 1. The antituple, `('point', ?x, ?y)`, consists of one defined field (i.e. `'point'`), which will be used to locate a matching tuple, and two wildcard fields, denoted by a leading `?`. The variables `x` and `y` will be bound to the values 60 and 72 respectively, when the input operation succeeds, as shown in the diagram. If more than one tuple in the tuple space is a match for an antituple, then any one of the matching tuples may be returned by the input operations.

Other forms of communication (such as one-to-many broadcast operations, many-to-one aggregation operations, etc.) and synchronization (e.g. semaphores, barrier synchronization operations, etc.) are easily synthesized from the five basic operations of the Linda model. Further details of the Linda programming model can be found in [Carriero and Gelernter 1990].

5. EXTENDED LINDA WEB SERVICE

The extended Linda web service is based on the extended Linda defined by Wells [Wells 2001] and is modeled closely on the standard Linda model, supporting the five basic Linda tuple space operations. The web service is also referred to as *eLindaWS* for brevity. A first version of the eLindaWS has been defined by Wells [Wells 2006]. This first version was only performing the basic Linda operations, whereas the actual version supports more operations and is much more mature. The current implementation of the eLindaWS has been developed to allow experimentation with the concepts that it embodies.

The eLindaWS consists of two separate parts: a server-side component, providing the web service itself, and a small client library, abstracting some of the details of the use of the web service. The server side of the web service has been developed as a Java servlet, using the Apache Tomcat servlet container[4]. Servlets are a Java technology for providing dynamic web content, and are thus ideally suited for the implementation of web services.

Web services have been proposed as a mechanism for utilizing the ubiquity of the World Wide Web to provide a simple, convenient distributed service architecture. In other words, a web service is a way of implementing a distributed system using the internet to communicate. In order to keep our web service as simple and convenient

[4]http://tomcat.apache.org/

```
<?xml version='1.0' encoding='utf-8'?>
<OutputTuple>
      <TupleData>
            <Field type='string'>'Point'</Field>
            <Field type='int'>12</Field>
            <Field type='int'>67</Field>
      </TupleData>
</OutputTuple>
```

```
<?xml version='1.0' encoding='utf-8'?>
<InputTuple blocking='false' destructive='true'>
      <TupleData>
            <Field type='string'>'Point'</Field>
            <Field type='int'/>
            <Field type='int'/>
      </TupleData>
</InputTuple >
```

Figure 2. *Examples of the XML Format of eLindaWS Tuples.*

as possible, we decided to adopt the *REST* (Representational State Transfer) approach proposed by Roy Fielding as a reaction to the proliferation of standards [Fielding and Taylor 2002]. The REST approach essentially calls for the transmission of simple XML messages across the Internet, using the standard HTTP protocol. This approach is distinguished by its simplicity, in comparison to more conventional approaches such as the use of SOAP, WSDL, etc.

XML stands for *EXtensible Markup Language* and is a standard designed to describe data[5], in our case tuples and antituples. Using XML to transport data has the advantage that the client and the server can be written in any language (Java, Perl, etc.) as long as they can communicate together and deal with XML. Figure 2 illustrates the XML format of an input and an output tuple sent from the client to the server. The XML received by the server is translated into a Java data structure, which is used with a Linda-style tuple space to carry out the relevant input or output operation.

The eLindaWS has three extensions to the original Linda programming model:

(1) Multiple, named tuple spaces
(2) Multi-tuple input and output operations
(3) A Programmable Matching Mechanism

These extensions are briefly explained hereafter. Multiple, named tuple spaces allow unrelated tuples to be stored separately, and thus provide greater efficiency. The multi-tuple output operation, which is called `outMany`, is used to minimize network traffic and to accelerate the execution time of applications. If a client application wants to add several tuples in a tuple space at the same time, the use of `outMany` allows a single XML document to be sent, containing all the tuples.

The multiple input operations, which have corresponding names and functionalities to the single input operations (`inMany, inpMany, rdMany, rdpMany`) return all tuples that match the given antituple. Similar to the `outMany` operation, this diminishes the network traffic.

The programmable matching mechanism allows the use of more flexible criteria for locating tuples. This is useful in situations where a global view of the tuples in a tuple space is required. For example, consider a scenario where a DNA sequence has to be retrieved which is the closest to another sequence (i.e. not necessarily equal to the given sequence). In situations such as these, searching for a sequence may involve retrieving all tuples before choosing the appropriate one. The programmable matching mechanism allows moving the necessary computation out of the application into a special form of matcher, which is executed on the server, minimizing the network traffic. This process is completely transparent to the application, which simply inputs a tuple, using a specialized matcher.

6. THE PARALLEL MOTIF SEARCHING APPLICATION

In order to investigate the benefits of using the eLindaWS for bioinformatics, an example application has been developped. It is not meant to be a revolutionary application and the pure performance of the application are not meant to be great. The goal of this application is to illustrate the usability and the easiness of the eLindaWS to parallelise applications. This example also shows that the eLindaWS is an efficient yet easily understandable tool and can therefore be used by biologists without a strong background in computer science.

The example application searches DNA sequences for subsequences corresponding to specific proteins (or *motifs*). This required the reverse-translation of protein sequences to obtain regular expressions describing the equivalent DNA sequences. Once these had been obtained, the regular expressions could be used with the Java `java.util.regex` package (part of the standard Java libraries since version 1.4 was released) in order to locate the motifs within DNA sequences.

[5]http://www.w3.org/XML/

The reverse-translation process involves several steps and is described in more detail below, followed by a description of the design of our application.

6.1 Obtaining Regular Expressions from Protein Sequences

The first step was to obtain suitable protein sequences. The consensus patterns for a number of protein motifs were downloaded from the PROSITE database [Hulo et al. 2004]. The Sequence Manipulation Suite, developed by the Center for Computational Genomics at Pennsylvania State University[6], was then used for the actual reverse translation, producing a DNA sequence. This was then expressed as a regular expression, suitable for use in a searching algorithm. For example, the consensus pattern for the protein sequence for the 'Homeobox' engrailed-type protein is `L-M-A-[EQ]-G-L-Y-N`. This is then stripped of all characters other than the amino acid specifiers, yielding `LMAEQGLYN`, and reverse-translated using the Sequence Manipulation Suite. This process results in considerable redundancy as a particular amino acid may be produced by a number of codons. For example, the amino acid represented by `L` may have either a `T` or `C` in the first position of the codon, a `T` in the second position and any one of the four DNA bases in the third position — this particular combination can be represented by the regular expression `[TC]T[GATC]`. The final result is the regular expression corresponding to the 'Homeobox' engrailed-type protein:

```
[TC]T[GATC]ATGGC[GATC][GC]A[GA]GG[GATC][TC]T[GATC]TA[TC]AA[TC]
```

Of particular interest in this expression is the subsequence `[GC]A[GA]`. This is the translation of the `[EQ]` protein sequence, formed from the combination of the regular expressions for the E and Q amino acids (`E = GA[GA]` and `Q = CA[GA]`).

6.2 The Design of the Parallel Motif Searching Application

The motif searching application was developed using the classic replicated-worker pattern [Freeman et al. 1999]. In this case, the master/controller process deposits tuples into the tuple space that specify the work to be done by the remainder of the processes (i.e. the 'workers'). Each tuple contains the name of the file containing the DNA sequence to be scanned, a unique task number and the total number of tasks generated (usually, but not necessarily, equal to the number of worker processes). The task number and the total number of tasks are used by the worker processes to determine which segment of the DNA sequence is to be searched. The calculation of the start and end points of the segment also has to include a degree of overlap between adjoining sections to allow for motifs that may span two segments.

After retrieving a tuple with the task parameters, a worker process reads in the calculated segment of the specified DNA sequence from the given file (using the Java random access file facilities). The regular expressions for the motifs of interest are then read in, and the `java.util.regex` library is used to perform the searching of the DNA segment. The workers take the results found (start- and end-points of identified motifs, and the actual DNA sequences for the motifs) and return these to the master process as tuples.

The master process retrieves the result tuples, and is then responsible for collating them as well as detecting and removing any duplicates found (due to the overlapping of adjoining segments, described above). Finally, the results are summarized and reported (together with processing time taken) to the user of the program.

Figure 3 illustrates the communication of the Parallel Motif Searching Application. While not wishing to go in depth into the implementation details, we would like to point out the different operations used for this application (`in, inp, inMany, out, outMany`). When comparing our current test results to former results conducted using the same application but using only the operations *in* and *out*, the current eLindaWS leads to a remarkable speedup.

7. EXPERIMENTAL PERFORMANCE RESULTS AND DISCUSSION

The performance of the Parallel Motif Searching application was measured for a 1GB file containing the first five human chromosomes (full details are available in [Akhurst 2004]) and for various numbers of processes. The volume of data in this file was too large to be handled by a single process and had therefore to be divided into a number of subtasks. This fact underscores the need for the parallelization of this application.

7.1 Experimental Environment

The network of workstations that was used for the experimental work consisted of a number of commodity PCs, such as might typically be found in any university or research center. These machines were all identical, equipped with Intel Pentium 4 processors, running at 3.2Ghz, and with 1024MB of RAM. A fast (100MBps), switched

[6]http://www.cbio.psu.edu/

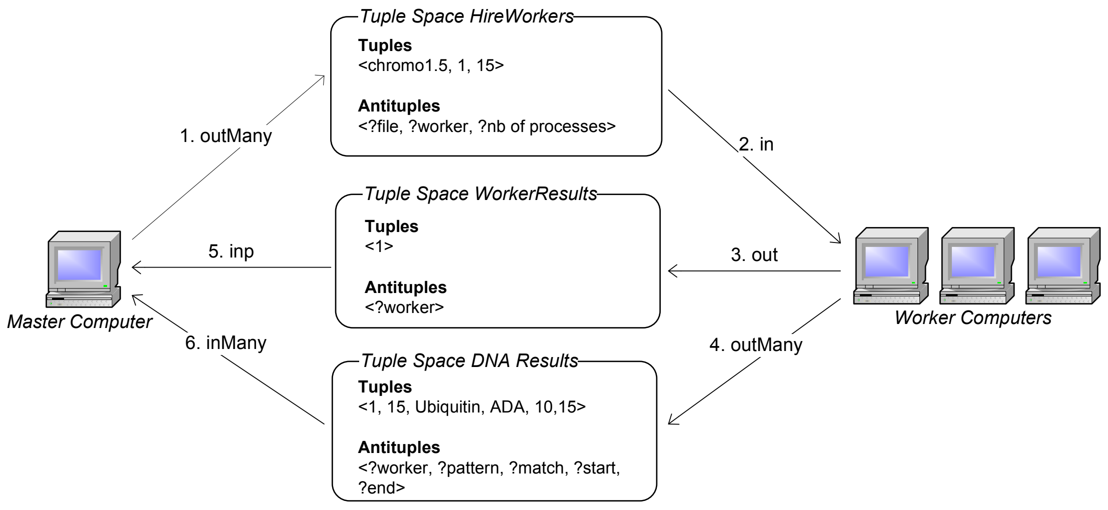

Figure 3. *The Communication in the Parallel Motif Searching Application.*

Ethernet network was used for the communication. Each machine ran the Windows XP operating system, and the Java Development Kit used was version 1.5.0 for the workers and version 1.6.0 for the server. The DNA sequences were stored locally on each workstation.

This configuration is ideal but not necesssary to perform the tests. We could also have stored the DNA sequences on a central computer or we could have made the master distribute the sequences to the workers. Moreover, we could have used computers in different locations and make use of an internet connection and the eLindaWS would still have been functional. This fact is very important since it is one of the core concepts of using a web service.

7.2 Results

We report in this section in detail on the results found for the 1GB file. Figure 4 shows the time needed to extract all matching patterns for various number of workers, where one worker corresponds to one workstation. The number of tasks varies between 15 and 30 to ensure that each worker processes the same number of DNA sequences (details can be found in Table I).

Each experiment was repeated eleven times and the average of the results was calculated in order to minimize the effect of any variations due to other network traffic or other factors.

As can be seen clearly, the time needed to process the file decreases with an increasing number of workers. In absolute terms, the time taken by a single worker was about 7 minutes (this required using 15 separate DNA sequences to accommodate the volume of data). This was reduced to a minimum of 14 seconds, when using 30 workers (in this setting each worker had to process only one DNA sequence) — a significant reduction in overall processing time.

7.3 Discussion

The speedup[7] measured for our application on 30 processors was 29.1 (an efficiency of 97.1%). However, we have to bear in mind that we cannot calculate the exact speedup at 30 processors with our datapoints, since we divided the 1GB datafile in 15 tasks for one worker, whereas we divided it into 30 tasks for 30 workers. The only points which allows us to calculate the exact speedup are at 5 processors and at 15 processors, which gives a speedup of 4.98 (an efficiency of 99.7%) and of 11.96 (an efficiency of 79.73%), respectively. This result points out that the Parallel Motif Searching Application can be highly parallelized with the eLindaWS, and underlines the suitability of our web service for parallelizable applications.

However, our main goal is not to obtain the best efficiency rate possible, but is to provide a tool which allows to parallelize applications easily, which provides language portability and which is understandable for researchers without a strong background in computer science. The Parallel Motif Searching Application is meant to illustrate the simplicity of designing and implementing an efficient solution using the eLindaWS. With regard to the various implementations of the Linda model in Java, Sheil makes the observation that 'Jini and JavaSpaces are not easy to use in a rapid development environment' [Sheil 2003]. In this regard, the eLindaWS is a much simpler system

[7]The speedup is calculated as the ratio of the single-worker time to that of the n-worker time.

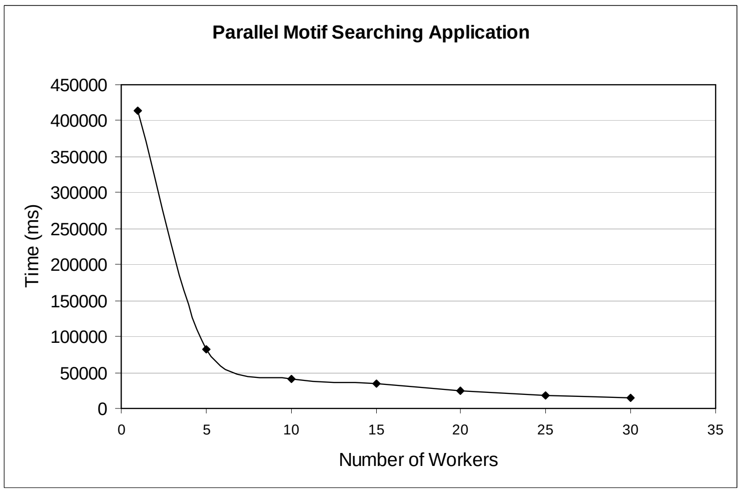

Figure 4. *Time needed to scan a 1GB file.*

Number of Workers	Number of Tasks	Average Time m	s	ms	Standard Deviation m	s	ms
1	15	6	53	629.3	0	43	493.8
5	15	1	22	894.8	0	6	391.7
10	20	0	40	470.8	0	0	642.9
15	15	0	34	573.2	0	9	758.1
20	20	0	24	765.5	0	2	504.0
25	25	0	18	315.6	0	1	315.5
30	30	0	14	205.6	0	0	185.3

Table I. *Average Execution Time and Standard Deviation for a 1GB file.*

and can be easily understood and used by scientists without a strong background in computer science. In order to use the eLindaWS, it is sufficient to know how to call the different output and input functions.

The speedup obtained depends on both the architecture of the eLindaWS and the implementation of the client application. Our focus was to provide an optimal architecture of the web service. It is likely that better results can be obtained with a better implementation of the Parallel Motif Searching Application, e.g. using indexing or another programming language. However, these issues depend on the futur users of our system and are not critical for this paper.

8. CONCLUSIONS

This study has demonstrated that our tuple space web service is a powerful and efficient tool for parallel or distributed applications, with good support for bioinformatics applications. The eLindaWS provides both language independency and a highly desirable simplicity for writing distributed applications.

Moreover, this study shows that Java provides a powerful programming and runtime environment, and provides helpful tools for both the client and the server side. However, applications can be written in other programming languages since the eLindaWS supports language portability. Since many bioinformatics applications are coded in scripting languages like Perl and since such environments usually do not have an extensive support for aspects such as parallel and distributed execution of algorithms, using the eLindaWS for those purposes is highly suitable. In order to demonstrate the interoperability of the web service, we implemented the master process of the parallel motif searching application in Python[8]. We cannot provide implementation details in this paper, but the results show that the master process can communicate perfectly with both client processes and the server, even if their implementation languages differ.

Results of the DNA motif searching application show that very good performance benefits can be derived from the use of the eLindaWS, using standard workstations, such as those found in most universities or research centers.

[8]http://www.python.org/

While this study used a dedicated network of workstations for the parallel execution of the application, this is unlikely to be available at all times in a typical research situation. Since the eLindaWS is a web service, workstations at different locations could be used to run a distributed applications. With this architecture, it is possible to share the work between hundreds of computers present on the internet and thus have enough resources to perform computationally-intensive processes. With the eLindaWS, it could be possible to create projects such as the well-known SETI screensaver[9]. Such techniques could be useful in order to extend the practical viability of our work.

As the field of bioinformatics continues to develop and mature, it is our belief that increasing use will have to be made of efficient development tools for parallel and distributed applications. We are convinced that the language portability together with the simplicity of the eLindaWS is a powerful tool for various bioinformatics applications, meeting the increasing demand for flexible, language independent tools.

ACKNOWLEDGMENTS

This work was supported by the Distributed Multimedia Centre of Excellence in the Department of Computer Science at Rhodes University, South Africa, with funding from Telkom SA, Business Connexion, Comverse, Verso Technologies and THRIP. Financial support was also received from the National Research Foundation (NRF) of South Africa.

REFERENCES

Bull, J. M., Smith, L. A., Pottage, L., and Freeman, R. 2001. Benchmarking Java against C and Fortran for scientific applications. In *JGI '01: Proc. 2001 Joint ACM-ISCOPE Conference on Java Grande*. ACM Press, 97–105.

Carriero, N. and Gelernter, D. 1990. *How to Write Parallel Programs: A First Course*. The MIT Press.

Cohen, J. 2005. Computer science and bioinformatics. *Comm. ACM 48,* 3 (Mar.), 72–78.

Fielding, R. and Taylor, R. 2002. Principled design of the modern web architecture. *ACM Trans. Internet Technology 2,* 2 (May), 115–150.

Freeman, E., Hupfer, S., and Arnold, K. 1999. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley.

Gibas, C. and Jambeck, P. 2001. *Developing Bioinformatics Computer Skills*. O'Reilly.

Hulo, N., Sigrist, C. J. A., Le Saux, V., Langendijk-Genevaux, P. S., Bordoli, L., Gattiker, A., De Castro, E., Bucher, P., and Bairoch, A. 2004. Recent improvements to the PROSITE database. *Nucleic Acids Research 32*, D134–D137. Database issue.

Luscombe, N., Greenbaum, D., and Gerstein, M. 2001. What is bioinformatics? A proposed definition and overview of the field. *Method Inform Med 40*, 346–358.

Pancake, C. and Lengauer, C. 2001. High-performance Java. *Comm. ACM 44,* 10 (Oct.), 98–101.

Sheil, H. 2003. Distributed scientific computing in Java: observations and recommendations. In *PPPJ '03: Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*. Computer Science Press, Inc., 219–222.

Villacis, J. 1999. A note on the use of Java in scientific computing. *SIGAPP Appl. Comput. Rev. 7,* 1, 14–17. http://doi.acm.org/10.1145/570150.570155.

Wells, G. 2001. A programmable matching engine for application development in Linda. Ph.D. thesis, University of Bristol, U.K.

Wells, G. 2006. A tuple space web service for distributed programming. In *PDPTA*. 444–450.

Wells, G. and Akhurst, T. 2005. Using Java and Linda for parallel processing in bioinformatics for simplicity, power and portability. In *Proc. IPS-USA-2005*. Cambridge, MA, USA.

[9]http://setiweb.ssl.berkeley.edu/