On the role of distributed computing in Symbolic Computation for Algebraic Topology *

Andrés, M. Pascual, V. Romero, A.† Rubio, J.

Abstract

In this work, we report on the preliminary analysis carried out in order to undertake the task of designing and building distributed systems for Symbolic Computation in Algebraic Topology. We try to make explicit how the peculiarities of the programs computing homology and homotopy groups can influence on the development of distributed systems to perform the same calculations. We stress not only on the (well-known) benefits of distributed computing, but also in the challenges and difficulties that appear due to the distinctive characteristics of our domain of computation.

Introduction

Nowadays, Internet appears as a suitable tool for performing scientific computations in a distributed collaborative way. One of the fields where this idea can be applied is that of Symbolic Computation. Computer Algebra packages are being extended to interconnect them. In fact, some Computer Algebra systems are already capable of performing distributed computing. An example is Distributed Maple [13], which is an environment for executing parallel computer algebra programs on multiprocessors and heterogeneous clusters.

With this example in mind, it is clear that this trend could be explored in any other Symbolic Computation system, in particular in systems devoted to algorithmic Algebraic Topology. The leader systems in this field are EAT [6] and Kenzo [7]. These systems, created by Sergeraert, can be used to compute homology groups of infinite topological spaces, namely loop spaces. Both systems are written in the Common Lisp programming language and have obtained some results (specifically, homology groups) which had never been determined before using neither theoretical nor computational methods. Since the programs are very time (and space) consuming, it is obvious that they are specially well placed to try making them distributed.

The organization of this paper is as follows. The next section presents our previous works and in section 2 we focus on the specific problems of distributed computing. The paper ends with a conclusions section and the bibliography.

*Partially supported by Ministerio de Ciencia y Tecnología, project TIC2002-01626 and by Comunidad Autónoma de La Rioja, project ACPI-2002/06.

†Supported by a FPU research grant, Ministerio de Educación, Cultura y Deporte.

1 Previous work

Motivated by the fact that there are some results of EAT and Kenzo that, up to now, can not be corroborated (nor refuted!) by any other mean (automatic or theoretical), we undertook some years ago the formal analysis of these systems, with the aim of increasing their reliability. Our first efforts were devoted to the algebraic specification of some data structures appearing in EAT and Kenzo (see, for instance, [8]). From this (still in progress) work, two new lines of research were born. In the first one, the Isabelle theorem prover [10] is used to approach the problem of constructing a computer-aided proof of the correctness of some fragments of Kenzo [2]. In the second line, our experience in algebraic specification has been also used to define an XML (eXtensive Markup Language) [14] format to encode the data occurring in the running of Kenzo [4].

This second line is a necessary step in order to build a distributed version of the systems. Hence, let us explain briefly that work. As it has been already mentioned, EAT and Kenzo are written in the Common Lisp programming language, but Java is the core language in the net-centric mainstream technology. Therefore we first thought of reconstructing the systems in the Java language. However Java is not a functional programming language, and functional programming features of Common Lisp are essential in EAT and Kenzo (namely, to encode, in a functional way, infinite data structures). In addition, the typing disciplines in Java and Common Lisp are very different. Hence, we introduced an intermediate step: we rebuilt (some fragments of) EAT in the statically typed functional programming language ML [11] (another reason to choose ML was that this is the programming language of the Isabelle theorem prover).

Thus, we have implemented two prototypes to perform computations in Algebraic Topology in ML and Java respectively. The following step we undertook was to interoperate between these systems through the net. For this, we needed to exchange complicated mathematical data between them. Our first attempt was to use MathML (Mathematical Markup Language) [9] (an XML dialect which was the first standard to represent mathematics) as exchange language, but it turned out to be not adequate as the elements of the algebraic structures that EAT works with are more complicated than those of MathML. Thus, we were forced to define our own DTD (see [14]), and we defined it as a proper extension of the MathML DTD for further compatibility. For maintenance purposes, each algebraic structure was endowed with the tree of calls that generated it.

For the moment, we have implemented a prototype that allows the three systems previously mentioned (the Common Lisp, ML and Java versions) to interchange XML documents (among them and also with themselves), as a first step towards collaborative computing. More precisely, one of the programs can stop temporarily its current computation and transfer the intermediate results in XML format (including also the necessary information on the "ambient" infinite algebraic structure where the computation is being carried out). Then the program itself or another program can recover these data and continue the computation. For the moment, the prototype works in a local, non-distributed way.

2 Distributed computing

One of the main barriers to increasing the usability of Kenzo is the fact that it does not provide a friendly front end: Common Lisp acts itself as a user interface. In view of the comments in the previous section, someone could wonder why not to make a publically accesible complete version in Java, a language much better known in any context. There are two reasons why this approach is not suitable. The first one is the time Kenzo needs to carry out the computations. Usually interesting groups need several days of CPU time over very powerful platforms, so it would be not feasible to work in a slow computer.

The second reason is that the computer-aided proof of the correctness of the system, when finished, would only certify the validity of the results in a determined environment, in an specific machine, with very concrete tools, but it would not be transferred easily to any other situation.

A solution which avoids this two problems consists on enabling a remote access to Kenzo. We would have the Kenzo program in a server machine, and several clients could access it from different computers. This task would allow clients to profit from the capabilities of Kenzo without the necessity of knowing the programming language Lisp, in which Kenzo is implemented. Using the (based in XML) interoperability between Lisp and Java mentioned in section 1, the client could be, for instance, an applet inserted in a HTML document, which would use the language Java to invoke some calculations in Algebraic Topology. These calculations would be translated in an XML document, which Kenzo would recapture and then process the computation. Once finished, the appropiate results would be again translated into XML, and come back to the client machine which would interprete them.

Nevertheless, this style of remote access has also two inconvenients. The first one is, once again, the time of response: it is not sensible that clients wait for several days before obtaining some answer. Hence, some kind of asynchronicity is necessary (perhaps by means of a subscription mechanism incorporated in a reactive way). The second problem is that the interoperability features presented in section 1 are almost unused. We could take profit of the two-layers organization of data in EAT and Kenzo. The first layer constructs in running time the algebraic structures (differential groups, algebras, coalgebras, and so on), several hundred in interesting cases, needed for computations. This stage, even if sophisticated, is finished in very few seconds. Only when the construction of first layer data is done, the very time consuming calculations with the second layer data structures start (these second layer data structures are elements of algebraic structures of the first one).

Hence, a clever design could put more work on the clients side, preparing the ambient data structures, checking the coherence of the input data, even doing some elementary computations, and only requiring the server when the really hard parts are encountered.

To this more collaborative client-server application, it is clear that the model previously presented is too poor, and therefore we need to try other designs. Some of the tools that we aim to work with are Corba [5] and RMI [12], which are middlewares that allow distributed and client/server programming, especially used with computers connected by a local net. They are based in objects, which have an interface that represents a contract between the client and the server. This interface is written as a Java interface for Java RMI, and in

IDL (a new interface description language) for CORBA. This intermediate language can then be used to translate to/from a variety of different languages, such as C, C++, Lisp, and also Java. In our particular case, it seems more appropiate to use CORBA, since there exist implementations for the Lisp language. But taking advantage of the Java-Lisp interoperability, we could also experiment with RMI.

As it is well known, the problems that traditional middlewares like Corba or RMI present to use them in the Internet has led to the increasing development of Web Services [3]. This is the third technology that we aim to try to work with to make possible the remote use of Kenzo, and to make it accessible by the Internet. Our goal is to build a Web Service which offers (some of) the capabilities of Kenzo and EAT to calculate through the net.

But, once a Web Service is enabled, it is tempting (and likely feasible) to use the same technology to make different computers on the net collaborate in a same computation, surpassing the client-server structure. However, this possibility opens new difficulties that will be addressed elsewhere.

3 Conclusions and future work

In this paper, the features of some Symbolic Computation Systems for Algebraic Topology, which are relevant to the objective of rebuilding them as distributed systems, have been detected. These characteristics are:

1) They are very time and space consuming.
2) They are written in Common Lisp.
3) They make use of higher order functional programming (to deal with infinite data structures).
4) They are organised in two layers of data structures.
5) They have been capable of computing results up to now unknown (stressing the importance of high reliability and of partial correctness proofs).

The influence of these points has been explored, leading us to a situation in which an experimental study of some prototypes (in Corba, RMI or Web Services) can be safely undertaken.

The next challenge would be to use Web Services in order to make a distributed version of Kenzo in which several computers could collaborate in a same computation (trying to decrease the time of response with regard to current version). The main difficulties here seem to be the need to coordinate several web services (some insights, in a different field of application, can be found in [1]) and to give a mechanised certification of correctness in this new net-based environment.

References

[1] P. Álvarez, J. A. Bañares, E. Mata, P. R. Muro-Medrano, J. Rubio. *Generative communication with semantic matching in distributed heterogeneous environments*, in Lecture Notes in Computer Science 2809 (2003) 555-569.

[2] J. Aransay, C. Ballarin, J. Rubio. *Towards a higher reasoning level in formalized Homological Algebra*, in Calculemus 2003, Aracne Editrice (2003) 84-88.

[3] G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[4] M. Andrés, F. J. García, V. Pascual, J. Rubio. *XML-Based interoperability among symbolic computation systems*, in Proceedings WWW/Internet 2003, Vol. II, Iadis Press (2003) 925-929.

[5] Object Management Group. *Common Object Request Broker Architecture (CORBA)*. `http://www.omg.org`.

[6] J. Rubio, F. Sergeraert, Y. Siret. *EAT: Symbolic Software for Effective Homology Computation*. Institut Fourier, Grenoble, 1997. `ftp://ftp-fourier.ujf-grenoble.fr/pub/EAT`.

[7] X. Dousson, F. Sergeraert, Y. Siret. *The Kenzo program*, Institut Fourier, Grenoble, 1999. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.

[8] L. Lambán, V. Pascual, J. Rubio. *An object-oriented interpretation of the EAT system*, in Applicable Algebra in Engineering, Communication and Computing 14 (2003) 187-215.

[9] Ausbrooks, R. et al. *Mathematical Markup Language (MathML)* Version 2.0, 2003. `http://www.w3.org/TR/2001/REC-MathML2-20010221/`.

[10] T. Nipkow, L. C. Paulson, M. Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, in Lecture Notes in Computer Science 2283 (2002).

[11] L. C. Paulson. *ML for the working programmer*, 2^{nd} edition. Cambridge University Press, 2000.

[12] Sun Microsystems, Inc. *Remote Method Invocation (RMI)*. `http://java.sun.com/products/jdk/rmi/`.

[13] Schreiner, W. et al. *Distributed Maple: Parallel Computer Algebra in Networked Environments*, in Journal of Symbolic Computation, Vol. 35, n 3 (2003) 205-347.

[14] T. Bray et al (eds). Extensible Markup Language (XML) 1.0 (Second edition), 2003. `http://www.w3.org/TR/REC-xml/`.

Departamento de Matemáticas y Computación.
Universidad de La Rioja. Edificio Vives. Calle Luis de Ulloa s/n.
26004 Logroño (La Rioja, Spain).
(email:{miriam.andres, mvico, ana.romero, julio.rubio}@dmc.unirioja.es)