1997

Collaborative Environments for Scientific Computing The Task of Algorithm/Software Selection

N. Ramakrishnan

Anupam Joshi

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
97-057

COLLABORATIVE ENVIRONMENTS FOR SCIENTIFIC COMPUTING: THE TASK OF ALGORITHM/SOFTWARE SELECTION

Naren Ramakrishnan
Anupam Joshi
Elias N. Houstis
John R. Rice

CSD-TR #97-057
December 1997

Collaborative Environments for Scientific Computing: The Task of Algorithm/Software Selection

N. Ramakrishnan
Department of Computer Sciences
Purdue University, IN 47907-1398
email: naren@cs.purdue.edu

Anupam Joshi
Department of Computer Engineering & Computer Science
University of Missouri, Columbia, MO 65211
email: joshi@csdeca.cs.missouri.edu

Elias N. Houstis and John R. Rice
Department of Computer Sciences
Purdue University, IN 47907-1398
email: {enh,jrr}@cs.purdue.edu

Abstract

The advent of High Performance Computing (HPC) and the Internet has brought forth a 'net–centric' scenario for scientific computation where important pieces of algorithms and software are spread across a(the) network. The current networked computing scenario assumes that the choice of method to solve a scientific problem is fixed a priori and that appropriate code, modules are downloaded, compiled and linked to form static programs. However, the selection of an appropriate method is by no means obvious and suitable methodologies do not exist to aid in the mapping of problem solution requirements to appropriate algorithms/software. Moreover, the network–based paradigm also allows for a scenario wherein multiple servers can house appropriate algorithms; this brings up the need for collaborative systems that dynamically select among competing servers/repositories and advise about algorithms/software. In this paper, we present a methodology that performs automatic algorithm selection for well–defined domains in scientific computing. Our approach selects among competing servers to determine the 'best' resources based on a notion of 'reasonableness'. This is implemented in the context of distributed KBs, i.e., the knowledge bases containing advisory information are themselves distributed across the network.

1 Introduction

Networked scientific computing representing the future computational paradigm signifies the changing landscape to support engineering and scientific computation. Computing in this paradigm assumes a service–oriented view wherein the user obtains vital pieces of software and information over the network and these are combined and linked together at run–time to perform the desired task. This shift from using software as a monolithic software product is motivated by (i) changes in the computing infrastructure (WANs, vBNS, wireless satellite–based networking), (ii) the increasing complexity of application software, (iii) the multidisciplinary nature of large scale application research, and (iv) the potential need to combine geographically disparate resources to obtain the MIPs required to solve grand challenge problems. While this scenario is yet to be realized in its grand totality, important pieces of its infrastructure are already in place. There exist systems that (i) provide access to software libraries over the network such as Netlib [6], (ii) enable full–service problem solving environments (PSEs) to be utilized over the web (WWW) - Net//ELLPACK [2], NetSolve [5] etc., (iii) index software modules from repositories according to functionality such as the Guide to Available Mathematical Software (GAMS) [4] and (iv) automate access to databases of test problems, data from performance evaluation such as NEOS [1] and the Matrix Market [3].

Such existing systems, however, assume that the choice of method (algorithm) to solve a given scientific problem is fixed *a priori* (static) and that appropriate code is located, downloaded, compiled and linked to yield static programs. Thus, scientists are very often faced with the onerous task of selecting suitable software for the problem at hand in the presence of practical constraints on accuracy, time and cost. In other words, it is required to 'adaptively' select software to conform to the performance requirements set by the user. We refer to this as the *algorithm/software selection problem.* This also encompasses issues of how a user specifies problem queries, extracts feature/content information and infers results. This problem becomes even more critical in a networked scenario where the computational resources increase several–fold and, in general, are non–homogeneous. As will be shown, in such a situation we face issues of learning and adaptation in collaborative systems and how one determines the most 'reasonable' resource to solve a given problem. The need for such automatic algorithm/software selection systems is becoming increasingly critical due to several reasons: (a) PSEs are becoming more ubiquitous and widely accepted in scientific communities; automatic algorithm/software systems would serve as advisory/recommender 'agents' for using PSEs. (b) A rapid increase in the number of online algorithms/methods made available to the application scientist has provided the impetus for developing such WWW based "Problem Solving Services" (PSSs). (c) Such systems also aid (indirectly) in the performance evaluation of scientific software. (d) If successful, they will serve as high level front–ends to software repositories of numerical algorithms; they can also complement the services of software 'indexing' systems like GAMS.

In this paper, we emphasize the need for collaborative multi–agent systems to achieve these goals and present details from the PYTHIA project at Purdue which concentrates on building advisory agents for scientific problem solving. The rest of the paper is organized as follows: Section 2 describes the intrinsic difficulties associated with solving the algorithm/software selection problem and in addition, stresses the issues associated with solving this in a networked setting. Section 3 details our effort in this direction, namely the collaborative PYTHIA system for automatic algorithm selection. Early empirical results with this system are provided in Section 4.

2 Algorithm/Software Selection

The problem of algorithm/software selection has its origins in an early paper by Rice [12]. The task is to decide on a good (enough) algorithm to achieve desired objectives, given a problem in scientific computation and performance criteria constraints on its solution (such as accuracy, time, cost, etc.). Ideally, one would like to build automated advisory systems that recommend strategies for problem solving. Even for 'routine' tasks in scientific computing, this can get quite complicated. Part of the difficulty stems from the unknown and ill–understood factors influencing the applicability (or lack thereof) of an algorithm in a certain context, the complex mapping from problem features to better algorithms, and the unstable way in which features could have an influence on algorithm selection. Further, the huge dimensionality of the problem (and feature) and algorithm spaces, lack of understanding of how the problem characteristics affect algorithm performance, and the inherent uncertainty in interpreting and assessing the performance measures of a particular algorithm for a particular problem compound to the difficulty. In this paper, we focus on this problem for the domain of elliptic partial differential equations (PDEs), in particular problems of the form

$$Lu = f \text{ on } \Omega, \quad \mathrm{B}u = \mathrm{g} \text{ on } \partial\Omega \tag{1}$$

where L is a second order linear operator (elliptic), B is a differential operator involving up to first order partial derivatives of u, and Ω is a bounded open region in 2– or 3–dimensional space. Given a problem as above and performance criteria constraints on accuracy and time, it is required to perform a selection such as: *"Use the 5-point star algorithm with a* 200 × 200 *grid on an NCube/2 using* 16 *processors: Confidence* - 0.85*"*. The efficacy of such predictions depends on the knowledge bases (KBs) utilized for such inference and how such KBs are developed. It is recognized that successful algorithm selection depends on (at least) three factors: (i) Performance evaluation of scientific software, (ii) the feature determination of scientific computing objects (SCOs) such as functions, domains, regions, operators, etc. and (iii) the "expert" methodology that provides domain specific inference to map from problem solution requirements to potential algorithms.

Even when such elaborate KBs are developed that take into account the diverse requirements for algorithm selection, the effectiveness of such schemes are contingent on their 'experience' as

brought out by the knowledge acquisition stage of the process. For example, PDEs come in different flavors and while one algorithm selection system might have been trained on PDEs arising from heat conduction problems, another could base its expertise on problems from computational fluid mechanics. In the networked scenario, when this causes a proliferation of such advisory systems, each with a specialized KB, one naturally thinks of strategies that could allow collaboration between these systems and capture the knowledge corpus of the network in an effective manner. At a different level, one needs to take into account the confidence that each system places on its recommendation, its experience, and how one determines the most reasonable system to solve a given problem. Ideally, thus one needs a scheme that maps from a given PDE problem to an appropriate advisory system on the network. Moreover, the situation can be more dynamic — the abilities of such individual systems can be expected to change over time (as they add more problems to their KB), more such systems might come into existence, etc. Thus any mapping technique utilized should have the ability to learn *on–line* and yet perform efficient selections.

3 The Collaborative PYTHIA System

The collaborative PYTHIA methodology [10] recognizes the fact that there are many different types of PDEs but that most scientists tend to use only from a limited subset of them encountered in their application domain. Hence, the approach taken is to create several different PYTHIA 'agents', each of which has information about a certain class of PDE problems and can select an appropriate solver for a given PDE of the class. Each PYTHIA agent ranks various solvers on test problems involving different constraints on accuracy and time. The strategy then, is to compare a given problem to the ones it has seen before, and then use its knowledge about the performance characteristics of prior problems to estimate those of the given one. Together with a good method to solve a given problem, a PYTHIA agent also provides a factor of confidence that it has in the recommended strategy. In its current preliminary implementation, a PYTHIA agent accepts as input the description of an elliptic PDE problem, and produces the method(s) appropriate to solve it.

The PYTHIA project web pages at *http://www.cs.purdue.edu/research/cse/pythia* provide information about this collaborative PYTHIA methodology and facilities to invoke it remotely. At the outset, there is a facility to provide feature information about a PDE problem. In particular, there are forms that enable the user to provide details about the operator, function, domain geometry and boundary conditions. Once these details are given, the information is submitted to a 'central' PYTHIA agent called 'C-PYTHIA' that performs further processing. It first classifies the given PDE problem into several categories of problems. These classes are defined based on the properties possessed by the solutions of the PDEs and are 'Solution–Singular', 'Solution–Analytic', 'Solution–Oscillatory', 'Solution–Boundary–Layer' and 'Mixed–Boundary–Conditions'. Having classified the problem into one or more of these classes, the PDE is submitted to an appropriate PYTHIA agent for this class of problems, which in turn selects a good (best?) strategy and reports back to the user. The important research issue is how to automatically learn this mapping from a PDE problem to an appropriate agent and update it suitably with time.

To determine an appropriate agent, we use a concept of 'reasonableness' to automatically generate exemplars for this purpose. This is needed because the user cannot be expected to have information about the most reasonable resource(s) for a given problem in such a dynamic environment. For example, in response to a query from the user about a particular PDE problem, each PYTHIA agent might suggest a different method with varying levels of confidence in the recommended selection. Moreover, each of these agents might have different levels of expertise (such as the kind of PDEs it knows about) and different 'training' history. The user, thus, cannot be expected to know which method is most suitable for the problem if all these responses are supplied. Hence, we propose a measure of reasonableness that allows the automatic (and unsupervised) 'ranking' of the PYTHIA agents for a particular problem (class). The trick is to combine two factors, one which denotes the probability of a proposition q being true, and the other which denotes its utility. Specifically, the reasonableness of a proposition q is defined as follows [9]:

$$r(q) = p(q)U_t(q) + p(\sim q)U_f(q),$$

where $U_t(q)$ denotes the positive utility of accepting q if it is true, $U_f(q)$ denotes the negative utility

of accepting q if it is false and $p(q)$ be the probability that q is true.

In the case of PYTHIA, each agent produces a number indicating its confidence in its recommendation, so $p(q)$ is readily available, and $p(\sim q)$ is simply $1 - p(q)$. For the utility, we use the following definition:

$$U_i(q) = -U_f(q) = f(N_e),$$

where f is some squashing function mapping the domain $(0, \infty)$ into $(0, 1]$, and N_e is the number of exemplars of a given type (that of the problem being considered) that the agent has seen. We chose $f(x) = \frac{2}{1+e^{-x}} - 1$. We chose this expression as the measure of the utility of an agent because it reflects the number of problems of the present type that it has seen. The value of an utility function is to measure the amount of knowledge that an agent appears to have. The more the problems of a certain kind in an agent's KB, the more appropriate will its prediction be for new problems of the same type. Hence, we have designed the utility to be a function of N_e.

We have worked on several techniques to determine a good mapping from the problem space to 'reasonable' agents using standard statistical methods, gradient descent methods, machine learning techniques and other classes of algorithms. Our experience has shown that specialized techniques developed for this domain perform better than conventional off–the–shelf approaches [8]. In particular, we have designed a neuro–fuzzy algorithm that infers efficient mappings, caters to mutually non–exclusive classes (i.e., a given PDE problem can belong to more than one class *simultaneously*) and learns the classifications in a dynamic manner. For more details about this algorithm, see [11].

4 Empirical Results

We have implemented our methodology in a case study that involves selecting suitable solvers for a population of 167 linear, second–order elliptic PDEs. From this population, the following six classes were defined.

(i) SINGULAR : PDE problems whose solutions have at least one singularity (6 exemplars).
(ii) ANALYTIC : PDE problems whose solutions are analytic (35 exemplars).
(iii) OSCILLATORY : PDE problems whose solutions oscillate (34 exemplars).
(iv) BOUNDARY-LAYER : Problems with a boundary layer in their solutions (32 exemplars).
(v) BOUNDARY-CONDITIONS-MIXED : Problems that have mixed boundary conditions in their solutions (74 exemplars).
(vi) SPECIAL : PDE Problems whose solutions do not fall into any of the classes (i)–(v) (10 problems).

The total number of exemplars is 191, 24 more than 167; this is due to the mutually non–exclusive classes. The total set of 167 problems in the PDE population was divided into two parts – the first part containing 111 exemplars (we refer to this as the larger training set), and the second comprising of the remaining 56 exemplars. We created five cases, with 2, 3, 4, 5 and 6 PYTHIA agents respectively. In each case, each PYTHIA agent knows about a certain class(es) of PDE problems. For example, in the case study with 6 PYTHIA agents, each agent knows about one class of problems. In the '3–agent' case, agent 1 knows about problem classes 1 and 2, agent 2 knows about classes 3 and 4 and agent 3 knows about classes 5 and 6. The agents communicate using the Knowledge Query and Manipulation Language (KQML) [7], using protocol defined performatives. All PYTHIA agents understand and utilize a private language (PYTHIA–Talk) that describes the meaning (content) of the KQML performatives. We conducted two sets of experiments: We first trained our technique on the larger training set of {problem, agent} pairs (using the notion of reasonableness defined earlier) and tested our learning on the smaller training set of 56 exemplars. In effect, the smaller training set forms the test set for this part of the experiment. In the second experiment, the roles of these two sets are reversed. We also compared our technique with two very popular gradient descent techniques for training feedforward neural networks, namely, Vanilla (Plain) Backpropagation (BProp) and Resilient Propagation (RProp). Fig. 1 summarizes the results.

It can be easily seen that our method consistently outperforms BProp and RProp on learning the mapping from problems to agents. Also, performance using the larger training set was expectedly better than that using the smaller training set. Moreover, our algorithm operates in an on–line

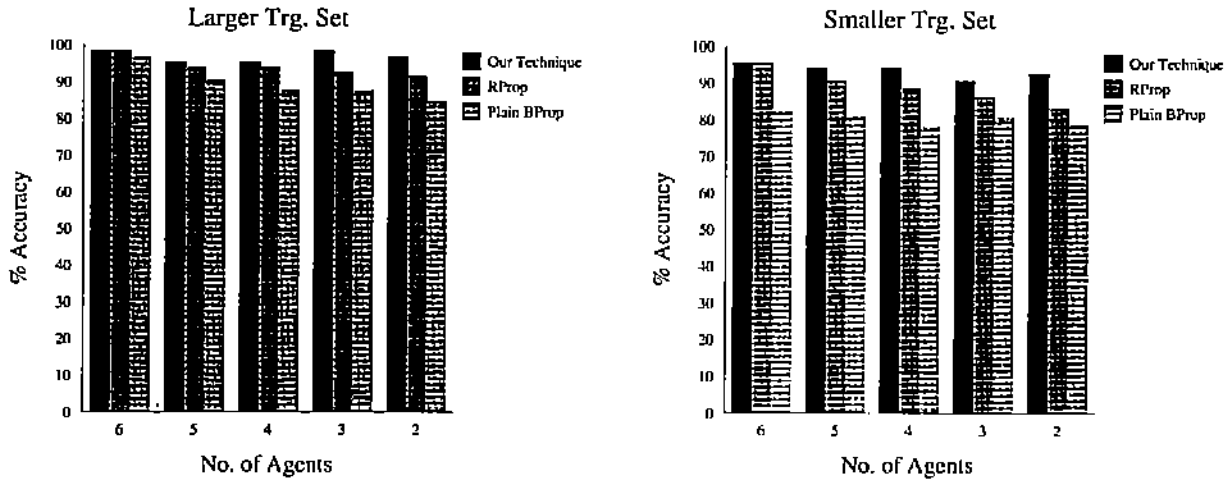

Figure 1: Results of Learning. The graph on the left depicts the results with the larger training set and the one on the right shows the values for the smaller training set. In each case, accuracy figures for the 5 scenarios (with 2, 3, 4, 5 and 6 agents) are presented with all the three algorithms considered in this paper.

mode; new data do not require retraining on the old. This enables us to automatically infer the capabilities of multiple PYTHIA agents. If the capabilities of agent 1 were to change, for example, in the 6–agent case, then our network could infer the new mappings without losing the information already learnt. This feature is absent in most other methods of classification such as BProp and RProp. In these the dimensionality of the network is fixed and it is imperative that the old data be kept around if these networks are to update their learning with new data. Our case study should thus be viewed in the incremental learning sense where the abilities of each of the six agents are being used and tracked continually in a dynamic setting.

References

[1] NEOS. http://www.mcs.anl.gov/home/etc/Guide/TestProblems.

[2] Net //ELLPACK. http://pellpack.cs.purdue.edu/netpp.

[3] The Matrix Market. http://math.nist.gov/MatrixMarket.

[4] R.F. Boisvert, S.E. Howe, and D.K. Kahaner. The Guide to Available Mathematical Software Problem Classification System. *Comm. Stat. - Simul. Comp.*, vol.20(4):pp.811–842, 1991.

[5] N. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. Technical Report CS-95-313, Dept. Computer Science, University of Tennessee, 1995.

[6] J. Dongarra, T. Rowan, and R. Wade. Software Distribution Using XNETLIB. *ACM Transactions on Mathematical Software*, vol.21(1):pp.79–88, 1995.

[7] R. Fritzson et. al. KQML- A Language and Protocol for Knowledge and Information Exchange. In *Proc. 13th Intl. Distributed Artificial Intelligence Workshop*, July 1994.

[8] A. Joshi, N. Ramakrishnan, E.N. Houstis, and J.R. Rice. On Neurobiological, Neuro–Fuzzy, Machine Learning and Statistical Pattern Recognition Techniques. *IEEE Transactions on Neural Networks: Special Issue on Neural Networks and Pattern Recognition*, vol.8(1):pp.18–31, 1997.

[9] K. Lehrer. *Theory of Knowledge.* Westview Press, Boulder, CO, USA, 1990.

[10] N. Ramakrishnan, A. Joshi, E.N. Houstis, and J.R. Rice. Neuro-Fuzzy Approaches to Collaborative Scientific Computing. In *Proc. IEEE International Conference on Neural Networks'97*, volume Vol.I, pages 473–478, 1997.

[11] N. Ramakrishnan, A. Joshi, S. Weerawarana, E.N. Houstis, and J.R. Rice. Neuro-Fuzzy Systems for Intelligent Scientific Computing. In *Intelligent Engineering through Artificial Neural Networks. Vol. 5: Fuzzy Logic and Evolutionary Programming (C.H. Dagli et.al., eds.*, pages 279–284. ASME Press, 1995.

[12] J.R. Rice. The Algorithm Selection Problem. *Advances in Computers*, vol.15:pp.65–118, 1976.