Service-Oriented Volunteer Computing for Massively Parallel Constraint Solving Using Portfolios

Zeynep Kiziltan and Jacopo Mauro

Department of Computer Science, University of Bologna, Italy
{zeynep,jmauro}@cs.unibo.it

1 Introduction

Recent years have witnessed growing interest in parallelising constraint solving based on tree search (see [1] for a brief overview). One approach is search-space splitting in which different parts of the tree are explored in parallel (e.g. [2]). Another approach is the use of algorithm portfolios. This technique exploits the significant variety in performance observed between different algorithms and combines them in a portfolio [3]. In constraint solving, an algorithm can be a solver or a tuning of a solver. Portfolios have often been run in an interleaving fashion (e.g. [4]). Their use in a parallel context is more recent ([5], [1]).

Considering the complexity of the constraint problems and thus the computational power needed to tackle them, it is appealing to benefit from large-scale parallelism and push for a massive number of CPUs. Bordeaux et. al have investigated this in [1] . By using the portfolio and search-space splitting strategies, they have conducted experiments on constraint problems using a parallel computer with the number of processors up to 128. They reported that the parallel portfolio approach scales very well in SAT, in the sense that utilizing more processors consistently helps solving more instances in a fixed amount of time.

As done also in [1], most of the prior work in parallel constraint solving assumes a parallel computer with multiple CPUs. This architecture is fairly reliable and has low communication overhead. However, such a computer is costly and is not always at our disposal, especially if we want to push for massive parallelism. Jaffar et al addressed this problem in [2] by using a bunch of computers in a network (called "volunteer computing" in what follows). They employed 61 computers in a search-space splitting approach and showed that such a method is effectively scalable in ILP.

In this paper, we combine the benefits of [1] and [2] when solving constraint satisfaction problems (CSPs). We present an architecture in which massive number of volunteer computers can run several (tunings of) constraint solvers in parallel in a portfolio approach so as to solve many CSPs in a fixed amount of time. The architecture is implemented using the service-oriented computing paradigm and is thus modular, flexible for useful extensions and allows to utilise even the computers behind a firewall. We report experiments up until 100 computers. As the results confirm, the architecture is effectively scalable.

A. Lodi, M. Milano, and P. Toth (Eds.): CPAIOR 2010, LNCS 6140, pp. 246–251, 2010.

2 Service-Oriented Volunteer Computing

Volunteer computing is a type of distributed computing which brings together the computational resources that are often idle and available in a network (e.g. `distributed.net`). As it offers a cost effective and large computing capability, volunteer computing seems to be a good candidate for the basis of a massive parallel constraint solving architecture using portfolios.

Service-oriented computing (SoC) is an emerging paradigm in which services are autonomous computational entities that can be composed to obtain more complex services for developing massively distributed applications (see e.g. the Sensoria Project `http://www.sensoria-ist.eu/`). In the context of volunteer computing, a service can be for instance the functionality which distributes jobs to the computers. Our architecture is designed and implemented in Jolie (`http://www.jolie-lang.org/`) which is the first full-fledged programming language based on SoC paradigm. The reasons behind the choice of SoC and thus Jolie can be summarised as follows. First, it is scalable; massive number of communications with different computers can easily be handled. Second, it is modular; new services can easily be integrated and organised in a hierarcy. This is particulary important in an architecture like ours which has several sub services. Third, it allows us to deploy the framework in a number of different ways. Jolie provides interaction between heterogenous services, like in the case of web services (e.g integrating a google map application in a hotel-search application). We can therefore easily interact with other services (even graphical ones) in the future and make our architecture be part of a more complex system.

3 Our Architecture

Fig. 1 depicts our architecture using a notation similar to UML communication diagrams. When services are used, we can have two kinds of messages: one way message denoted by the string ⟨ message name ⟩(⟨ data sent ⟩) and a request response message denoted by ⟨ message name ⟩ (⟨ data sent ⟩)(⟨ data received ⟩). The figure is read as follows. The user utilises the *redirecting service* to get the location of the *preprocessing service* and then sends to the preprocessing service a problem instance i_k to be solved. Once i_k is sent, the preprocessing service contacts the *CBR service* which runs a case-based reasoning system to provide the expected solving time t_k of i_k. The preprocessing server then sends t_k and i_k to the *instance distributor service*. This service is responsible for scheduling the instances for different (tunings of) solvers and assigning the related jobs to the volunteer computers. This can be done in a more intelligent way thanks to t_k provided by the CBR service. This value can be used for instance to minimize the average solving time. Finally, the *volunteer service* asks the redirecting service the location of the instance distributor service and then requests a job from it using a request response message. This is the only way a job can be sent to the volunteer service. Note that the use of the redirecting service makes it possible to have multiple preprocessing and instance distributor services in the future.

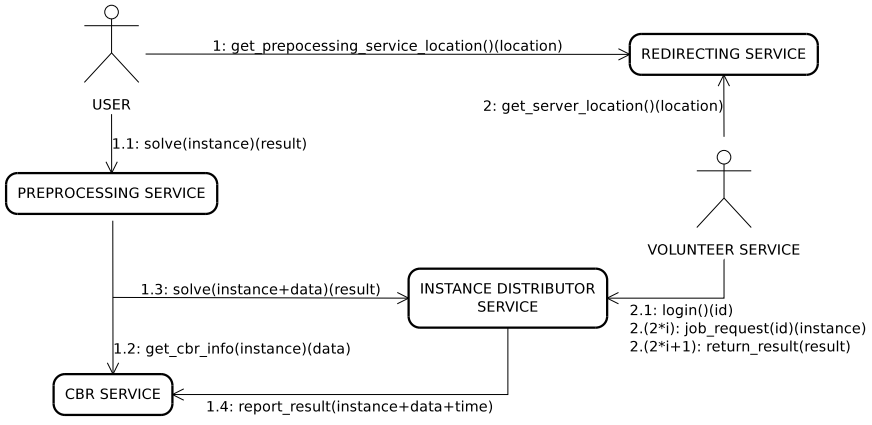

Fig. 1. Architecture

An input instance of the architecture is specified in XCSP which is a relatively new format to represent constraint networks using XML (`http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf`). The reason of this choice is that (i) XCSP format has been used in the last constraint solver competitions and thus many solvers have started to support it; (ii) such a low level representation is useful to extract the feature vectors needed by a CBR algorithm.

4 Preliminary Experimental Results

In these preliminary experiments, our concern is the scalability. We thus currently exclude the CBR service and observe how the architecture scales as the number of computers increases. In the experiments, Dell Optiplex computers of our labs running Linux with Intel core 2 duo and Pentium 4 processors are used. Up to 100 of them are employed for the volunteer service and only one for the remaining services. We consider the instances of the 2009 CSP Solver Competition (`http://www.cril.univ-artois.fr/CPAI09/`), six of its participating solvers (Abscon 112v4 AC, Abscon 112v4 ESAC, Choco2.1.1 2009-06-10, Choco2.1.1b 2009-07-16, Mistral 1.545, SAT4J CSP 2.1.1) and one solver (bpsolver 2008-06-27) from the 2008 competition (`http://www.cril.univ-artois.fr/CPAI08/`). These solvers are provided as black-box, their tunings is not possible. Hence, an instance is solved by 7 solvers on 7 different computers. The experiments focus on the following instances: (i) Easy SAT: 1607 satisfiable instances solved in less than 1 minute; (ii) Easy UNSAT: 1048 unsatisfiable instances solved in less than 1 minute; (iii) Hard SAT: 207 satisfiable instances solved in between 1 and 30 minutes; (iv) Hard UNSAT: 106 unsatisfiable instances solved in between 1 and 30 minutes. Such times refer to the best solving times of the competition.

In Table 1, we present the number of instances solved in 30 minutes for the easy instances and in 1 hour for the hard instances. As an experiment is affected by the current work load of the computers, we perform and report three

Table 1. Experimental results

n°	Easy SAT (30 min)			Easy UNSAT (30 min)			Hard SAT (1h)			Hard UNSAT (1h)		
20	15	14	15	17	18	18	3	3	6	7	7	9
40	132	128	135	150	150	150	8	8	7	16	17	13
60	141	140	140	320	318	322	19	15	14	23	23	22
80	144	145	151	335	323	328	25	21	25	29	30	30
100	179	179	192	336	345	334	25	25	25	44	33	36

runs. The results are promising. Even without the CBR service and the different tunings of solvers, the number of the instances solved in a fixed amount of time increases as the number of computers increases, no matter how busy the volunteer computers are. Note that only one computer is used to run the preprocessing and the instance distributor services, and yet the system can handle 100 computers without any problems. The main reason for not always obtaining a linear speed up is that some solvers cannot solve even the easy instances in less then 30 minutes. Hence, many computers are spending more than 30 minutes for solving an already solved instance. This has happened 104 times in the tests of easy SAT instances with 100 volunteer computers. In the same tests, we as well encountered 35 solver failures. These observations suggest we shall allow the interruption of a computation if the related instance is already solved.

5 Related Work

There is considerable amount of prior work on parallel constraint solving. We here discuss only those that use massive parallelism. Our work is similar to the one described in [1] in the sense that we too use the portfolio approach. However, there are a number of differences. First, we consider CSP instances and several different constraint solvers (including SAT and CP solvers), as opposed to SAT instances and one SAT solver. Second, we create portfolios by running each instance on several computers and several (tunings of) solvers at the same time. The solver-independent approach of [1] instead uses only different tunings of the same solver. Third, whilst we assume a group of independent computers available in a network, [1] assumes a dedicated cluster of computers.

Jaffar et al [2] as well propose an architecture based on volunteer computing. Unlike ours, this architecture uses the search-space splitting strategy and the experiments confirm scalability on ILP instances using 61 computers. There are however other substantial differences. In many environments like laboratories and home networks, computers stay behind a firewall or network address translation (NAT) which limit their access from outside. We are able to access such computers by using only the request response messages instead of using direct messages as done in [2]. This choice brings further advantages over [2] like smaller number of messages sent, the applicability to the majority of networks, and having only the server as a possible bottleneck. The price to pay is the impossibility of using certain protocols to create a tree of volunteer computers. This is however not needed in our architecture. Our volunteer computers never

communicate with each other so as to avoid potential scalability problems. This is not the case in [2]. As the number of computers increases, the overhead of distribution becomes too high which can lead to significant slowdown.

Our architecture owes a lot to CPHydra [4] , the winner of 2008 CSP Solver Competition . It combines many CP solvers in a portfolio. CPHydra determines via CBR the subset of the solvers to use in an interleaved fashion and the time to allocate for each solver, given a CSP instance. Our work can thus be seen as the parallel version of CPHydra which eliminates the need of interleaving, giving the possibility of running several (tunings of) solvers at the same time. This brings the chance of minimising the expected solving time as there is no order among the solvers. Moreover, parallelism gives the opportunity of updating the base case of CBR even in a competition environment.

6 Conclusions and Future Work

We have presented an architecture in which massive number of volunteer computers can run several (tunings of) constraint solvers in parallel in a portfolio approach. The architecture is implemented in SoC which is becoming the choice of paradigm for the development of scalable and massively distributed systems. The initial experimental results confirm the scalibility. Our plans for future are to make the architecture more efficient, useful and realiable. As for efficiency, we are currently working on the CBR service and investigating how to best benefit from similar cases in a parallel context. As for usability, we aim at tackling two limitations. First, our architecture gets XCSP instance format which is too low level for a CP user. The good news is that the architecture can easily be integrated to a high level modelling & solving platform such as Numberjack (`http://4c110.ucc.ie/numberjack`) which will soon output to XCSP. In this way, we can obtain a system in which the user states her problem easily at a high-level of abstraction, then the problem gets converted to XCSP and then (CBR-based) parallel solver is invoked. Second, our architecture is focused on the portfolio approach. We intend to investigate how to exploit massive number of computers in search-space splitting when solving optimisation problems in CP. Finally, should the computers go off or malfunction, we might want to replicate the jobs assigned to the computers or redirect them to other computers. We will study methods to make the architecture more reliable from this perspective.

References

1. Bordeaux, L., Hamadi, Y., Samulowitz, H.: Experiments with massively parallel constraint solving. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 443–448 (2009)
2. Jaffar, J., Santosa, A.E., Yap, R.H.C., Zhu, K.Q.: Scalable distributed depth-first search with greedy work stealing. In: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), pp. 98–103 (2004)

3. Gomes, C., Selman, B.: Algorithm portfolios. Artificial Intelligence 1-2, 43–62 (2001)
4. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Proceedings of the 19th Irish Conference on Artificial Intelligence, AICS 2008 (2008)
5. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: Solver description. In: SAT-Race 2008 (2008)