FACTA UNIVERSITATIS (NIŠ)
SER. MATH. INFORM. Vol. 29, No 2 (2014), 155–171

DESIGN AND COMPARISON OF TWO WEB SERVICE BASED FRAMEWORKS FOR PARALLEL EVALUATION OF THE POPULATION IN GENETIC ALGORITHMS *

Miloš Ivanović, Ana Kaplarević-Mališić, Višnja Simić and Boban Stojanović

Abstract. Genetic algorithms are powerful techniques for optimization of complex systems. These methods require a large number of evaluations of candidate solutions which take huge CPU time. This paper introduces two web service based frameworks for parallel evaluation of the population in genetic algorithm using the master-slave model. Developed frameworks can be easily incorporated into any genetic algorithm, giving a universal mechanism for distribution of individuals and collection of the evaluation results. This concept provides parallelization of genetic algorithms on various distributed architectures, including multiprocessors and computing clusters. Performed tests have shown that proposed frameworks achieve significant speedup, especially when evaluating large-scale problems. In addition, a case study from the field of hydrology is presented.
Key words: Parallel computing, Genetic algorithm, Optimization, Web service, Hydrology

1. Introduction

Genetic algorithms (GAs), as a major class of Evolutionary algorithms (EAs), have proven themselves as a robust and powerful mechanism when it comes to solving challenging optimization problems [1, 2]. They mimic the process of natural evolution, by modifying the set of potential solutions, called population, through selection, crossover and mutation of individuals. In order to select the best candidates for reproduction, one has to evaluate the fitness of each individual in the population. Solving the optimization problem using a genetic algorithm requires evaluation of hundreds of individuals through several tens of generations. Since each evaluation in real-world problems usually requires running a complex, time consuming computer simulation, the whole optimization process can last for hours or even days.

Received March 12, 2014.; Accepted Jun 03, 2014.
2010 *Mathematics Subject Classification.* C.2.4; G.1.6; J.2
*The part of this research is supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia (Grants III41007, OI174028, TR37013, III44010, and TR 14005, and FP7 ICT-2007-2-5.3 (224297) ARTreat project.

In order to speed up the optimization task, various approaches for the use of parallelism in genetic algorithms have been proposed in the literature and surveys have been written [3, 4, 5, 6]. Three major parallel models can be distinguished [5]: self-contained parallel cooperation (between different algorithms), problem independent intra-algorithm parallelization, and problem dependent intra-algorithm parallelization.

Self-contained parallel cooperation algorithms, known as island model algorithms, are suitable for large scale search spaces and can be used on parallel systems with very limited communication. The population is divided into several sub-populations (islands) and serial GA is executed in each of these islands for a number of generations called an epoch. At the end of each epoch, individuals migrate between neighboring islands along migration paths. Inter-processor communication frequency in this model is low, but modeling requires a lot of parameters and design decisions. Intra-algorithm parallelization does not affect applied genetic algorithm, and is therefore used to speed up the search. Algorithms with problem independent intra-algorithm parallelization, called master-slave models or global parallelization algorithms, consider parallelization of a single iteration of a genetic algorithm. This type of parallelization model is suitable for problems where fitness evaluation of an individual is time-consuming. In problem dependent intra-algorithm parallelization, problem-dependent operations are parallelized, i.e. evaluation of a single individual is parallelized (different objectives and/or constraints). This model is useful when dealing with time and/or memory intensive objectives and constraints. For example, when several solvers have to be used, or when multiple runs for one fitness evaluation have to be done.

In the field of large scale optimization, both shared memory and distributed memory parallel systems have been used. In a shared-memory multiprocessor environment, the operating system is helped with multi-threaded APIs such as POSIX Threads, OpenMP and inherent thread models in platforms such as Java and .NET dominate. On the other hand, considering a distributed memory environment, previous work usually employs involved MPI (Message Passing Interface), sockets and Java RMI. More recently, global Internet trends induced the use of grid middleware such as those based on Globus toolkit and common Internet based technologies like web services.

Many frameworks for the parallel execution of the EAs have been proposed and implemented. Distributed BEAGLE [7] uses the master-slave model for distributing evaluations on several processors of Beowulf clusters or LANs of workstations. ParadisEO [8] is a general framework for parallel and distributed metaheuristics. It provides the most common parallel and distributed models, portable on distributed-memory machines and shared-memory multiprocessors, since they are implemented using standard libraries such as MPI, PVM and PThreads. MALLBA [9] is a C++ library of algorithms for optimization that can deal with parallelism (using MPI) on LAN or WAN platforms. All algorithms in the MALLBA library are implemented as software skeletons that users customize depending on the specific problem. In the Simdist framework [10], distribution subsystem and evolution-

ary system are run as separate processes, communicating only through standard I/O streams, so the evolutionary system can be implemented in any programming language. The framework uses MPI as the underlying protocol for parallel programming.

This paper introduces two frameworks for parallel evaluation of the population in GA using the master-slave model. Both frameworks distribute unevaluated individuals to slave processes, collect and return the evaluation results to the master. What distinguishes our systems from the ones previously developed is that they employ web services, which provide a means of interoperating between framework elements [11]. The frameworks are easy to implement, independent of the chosen GA, and can be deployed on different parallel architectures, such as multiprocessor and computing cluster. Windows Communication Foundation (WCF) [12] was used for web service implementation in proposed frameworks. The frameworks are presented from an architectural perspective and pros and cons of each are analyzed in detail.

The rest of the paper is organized as follows: the proposed frameworks are described in Section 2. In Section 3 experimental results and discussion are presented, followed by a case study in Section 4. Section 5 concludes and outlines some future work.

2. Description of the Frameworks

The proposed frameworks consist of one dispatcher and a number of workers. Such architecture requires a sequential GA to be split into two separate programs: the master part, containing the main evolutionary loop, and the slave part, containing the fitness evaluation routine which is executed by the workers. The dispatcher acts as the intermediate layer between the master part of GA and the workers which evaluate individuals. The frameworks provide asynchronous parallel evaluation of individuals from a generation. The master executes the evolutionary loop in the main thread (Figure 2.1) to the point when a generation needs to be evaluated. Then, the dispatcher is started in a separate thread. It receives individuals from the master, enqueues them, distributes individuals to available workers, and returns the evaluations results from the workers to the master. The main thread remains stopped until all individuals from a generation are evaluated and afterwards continues with the rest of the genetic algorithm. The developer of the optimization system has to separate the evaluation step from the rest of the genetic algorithm loop, but has no concern with the details of distributing individuals and retrieving evaluations results. It is up to the dispatcher to distribute individuals to the workers as efficiently as possible, and gather the results.

Although both proposed frameworks follow the intra-algorithm parallelization model, they differ regarding communication protocol. The first framework is based on the push distribution model, where the dispatcher initiates communication with workers and requests evaluation of individuals. In contrast, the second framework uses a pull model, where workers request evaluation tasks from the dispatcher.

The motivation for using WCF comes from the fact that a certain number of existing commercial scientific applications are developed in .NET platform. These frameworks enable GA based applications developed in .NET platform to seamlessly distribute evaluation of individuals to the platforms that support WCF (Windows, Mono/Linux, etc.). Furthermore, when it comes to the fitness evaluation, the proposed frameworks allow both: usage of .NET based evaluators, as well as evaluators developed in other platforms supported by underlying OS.

Detailed descriptions of both frameworks are given in the following text.

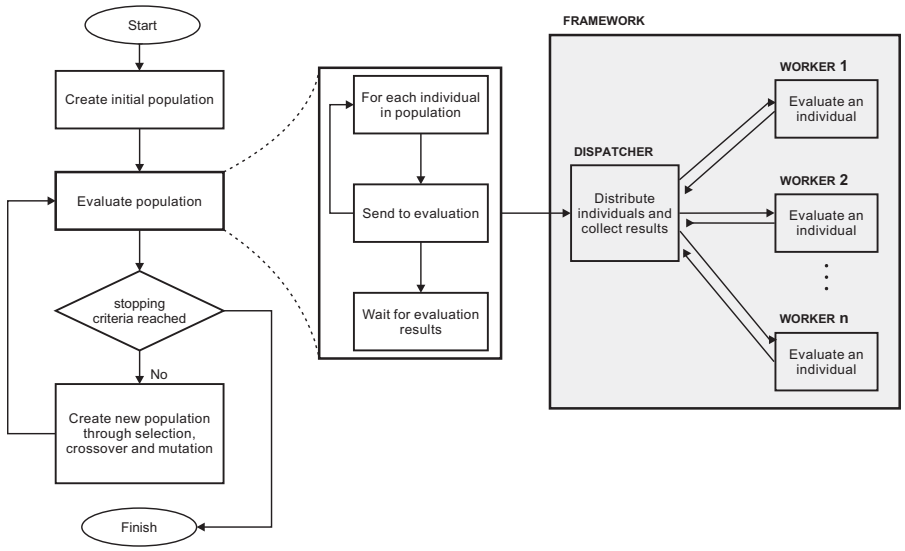

FIG. 2.1: Schematic view of master-slave parallel GA with the dispatcher as the intermediate layer between the master part of GA and the workers which evaluate individuals

2.1. Push-model framework

The push-model framework was introduced in [13]. The main part of the dispatcher is the evaluation pool. It represents an intermediate step between genetic algorithm running on the master and routines that evaluate individuals on the worker nodes. When the master sends individuals for evaluation, they are being queued in the evaluation pool. The evaluation pool acts as the client and dispatches individuals to the workers which are implemented as WCF web services.

Figure 2.2 illustrates the push-model framework. The evaluation pool is initialized by the master with information regarding available web services. Each time the generation has to be evaluated, the master starts the evaluation pool in a separate thread and sends it individuals for evaluation, where they are being

queued. The main thread of the algorithm remains blocked until all individuals sent to evaluation are evaluated.

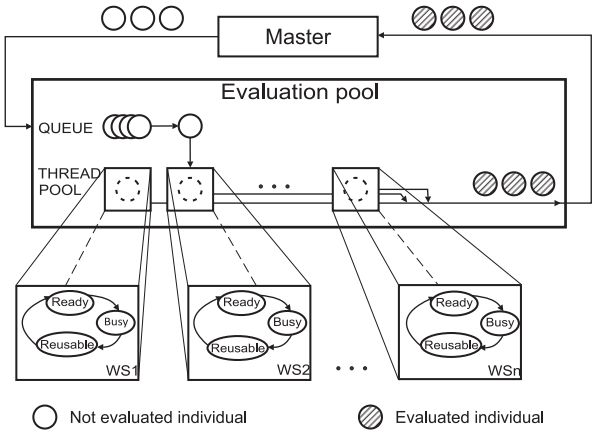

FIG. 2.2: Push-model framework

The evaluation pool uses the thread pool provided by the .NET Framework through the ThreadPool class [14, 15]. A thread pool is a collection of threads that can be used to perform a number of tasks in the background. In this framework, every single thread is used to send an individual to the web service running on the slave node. This way, evaluations can be processed asynchronously, without tying up the primary thread of the evaluation pool or delaying the processing of subsequent requests. Each thread calls the web service on a network computer and sends it an individual that has to be evaluated. When evaluation of the individual is done, the result is returned to the client application and assigned to the individual. Once a thread in the pool completes its task, it is returned to a queue of waiting threads. Then it can be reused for the rest of queued individuals, the same as the corresponding web service which has completed the evaluation. This reuse enables applications to avoid the cost of creating a new thread for each task.

Once all the web services return results and there are no more individuals in the queue, the evaluation pool sends a signal to the main thread that all evaluations are over and that the master thread can continue with the rest of the genetic algorithm.

If any web service raises an exception and the evaluation of the individual fails, the same individual is sent to the first free web service. To avoid repetitive errors on the faulty web service, the evaluation pool does not send individuals to that service for some time. This way, the faulty web service is not fully eliminated from the evaluation process, but has a chance to be recovered.

It should be noted that push-model has limitations in large computing clusters and computing grid environments, where worker nodes usually reside behind firewalls, and often within NAT (Network Address Translation) domain. Firewalls and NAT prevent dispatcher to directly invoke web services residing on workers,

which is resolved by the pull communication model.

2.2. Pull-model framework

In a pull model, the dispatcher receives requests from both the master and workers. The master sends individuals to the dispatcher where they wait to be evaluated. Worker nodes request individuals from the dispatcher, execute the evaluation and return obtained fitness values to the dispatcher. On the request of the master, the dispatcher fetches evaluated individuals back to the master.

The dispatcher in the pull-model is divided into two components: the evaluation pool and the manager. In contrast to the push-model framework, the evaluation pool in the pull-model does not communicate with workers directly, but rather over the manager which is the only web service in the entire framework. The manager is being called by both the evaluation pool and a number of workers, acting as intermediate WCF service between job requests and available computing resources. All the activities performed by the manager are reactions to the client invocations.

The master starts the evaluation pool in a separate thread and sends it individuals for evaluation, where they are being queued (Figure 2.3). The evaluation pool promptly reacts on arrival of each new individual, and sends it to the manager. When a whole generation is sent, the master waits for all individuals to be evaluated and returned from the evaluation pool.

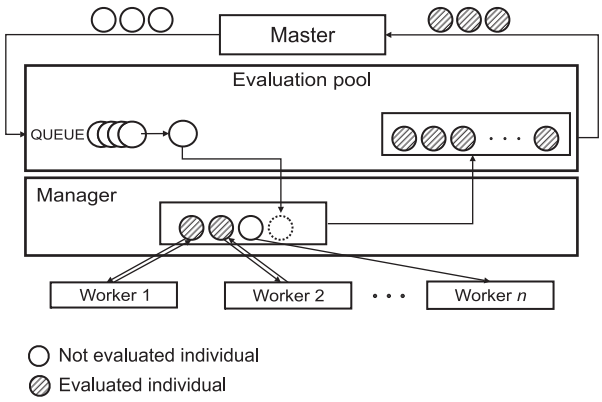

FIG. 2.3: Pull-model framework

A worker registers with the manager and requests one individual to evaluate. Upon a manager's response, the worker executes evaluation of the supplied individual, returns the result and afterwards is able to request a new unevaluated individual. The workers communicate with the manager in asynchronous fashion. This implies that they act as clients accessing the manager through two different types of web service calls. The first type of call pulls a job from the manager, and the

second type returns evaluation results to the manager. On each client invocation the manager starts a separate service thread in order to enable handling multiple clients in a parallel manner.

The frequency of worker requests plays a major role in overall pull-model framework performance. Due to the nature of genetic algorithms, unevaluated individuals are sent to the manager intermittently. Consequently, there are periods between generations when workers are not occupied and therefore request new jobs frequently. Too frequent invocations decrease the manager's responsiveness, since it has to employ resources for each call, send a reply back and finally release occupied resources. In order to avoid this performance bottleneck, in case the worker receives the response that there are no unassigned individuals at the moment, it remains idle for a specified time (*workerIdleInterval*) before sending out a new job request. It should be noted that too many requests overload the manager, but too long of an interval delays commencement of the distributed evaluation process.

3. Performance analysis - results and discussion

The main objectives of the analysis were:

- *To evaluate the parallelization validity* of both proposed frameworks, measuring speedup of GA
- *To evaluate the overhead influence rate in various parts of the problem spectrum*. More precisely, the aim was to experimentally measure the extent to which the unit size of transferred data, as well as the manager communication frequency with worker processes, affects overall slowdown.

Using a simple GA with real-encoded chromosomes we solved the artificial test problem. The fitness function was a dummy function that does nothing except receive individuals, sleep for a certain number of seconds, and return random fitness value to the evaluation pool or to the manager, depending on whether push-model or pull-model framework was used. Since one of the most significant goals of this study was to rate inherent communication overhead within distributed GA, the intention was to mask the influence of various CPU speeds and loads in the real computing environment, such as a computing cluster, and isolate networking effects. Therefore, it did not matter whether the workers were performing calculations simply sleeping, it only mattered how long it took them to return their response. The following set of parameters for the simple GA algorithm was adopted: a simulated binary crossover with a probability of 0.9 and a distribution index of 20, and polynomial mutation with a distribution index of 20 and probability of $1/l$, where l is the chromosome length.

Benchmark setup for performance examination considered the following variable parameters: **fitness evaluation time for an individual** (from 10 milliseconds to 10 seconds), **population size** (100, 250, 500, and 1000 individuals), **length of**

chromosomes (represented by 10, 100 and 1000 variables of type *double*), **number of workers** (5 to 24). The *workerIdleInterval* was set to 1 second. This value turned out to be a well balanced choice, since managers' CPU load never rose above 25%.

The benchmarks were carried out using two parallel architectures; *shared memory architecture* represented by **symmetric multiprocessor**, and *distributed memory architecture* represented with a Beowulf type **computing cluster**.

For completeness, a brief description the hardware/software environment follows. Multiprocessor architecture was represented by SMP machine equipped with 2 AMD Opteron 6174 CPUs (2.2GHz, 12-core each, totaling to 24 cores) and 64GB of RAM. Computing cluster architecture was represented with a Beowulf type cluster consisting of 14 nodes, each with a single 2.4GHz Intel Core2Quad Q6600 CPUs with 8GB RAM memory, totaling to 56 CPUs with 112GB RAM memory. All nodes run Scientific Linux 5.8 x86_64 with cluster jobs running inside PBS Torque batching environment. Since the original MS .Net Framework is not available on POSIX compatible platforms, the authors employed the substitution; the open source implementation of .NET Framework Mono v2.10.5 which almost completely complies to .NET 3.5 standard.

3.1. Speedup analysis

Having in mind the adopted parallelization strategy (distributed evaluations of the individuals), it turns out that the most valuable information reflecting general speedup is the time needed to evaluate the entire generation. The speedup values in the diagrams below are averages taken from 100 separate runs. Depending on the fitness evaluation time of an individual, parallelization is more or less a good choice. Precisely speaking, we measured the time needed to evaluate the whole generation as a function of number of workers involved, while population size (500), as well as length of the chromosome (10 doubles) were fixed. The maximum number of workers was limited to 24, due to the fact that this is the number of CPU cores within representative multiprocessor architecture taken for benchmarking purposes.

In Figure 3.1, the speedup analysis graphs of the push and pull-model framework in multiprocessor and computing cluster environments are presented. Speedup is calculated as T_t/T_m where T_t is the theoretically assumed duration of a fully sequential process of evaluation (number of individuals times individual evaluation duration) and T_m is the experimentally measured time. Perfect speedup is the theoretical limit assuming that all communications are neglected. The most obvious observation considering these diagrams is that two proposed frameworks behave more or less similarly in a scalability sense, regardless of architecture (either multiprocessor or computing cluster). In the case of more time consuming fitness evaluations, speedup quickly converges to theoretical ideal. On the other hand, with less time consuming evaluations, speedup reaches maximum at some point and then starts decreasing. The reason for such behavior lies in web service communication overhead, which becomes more significant with larger number of

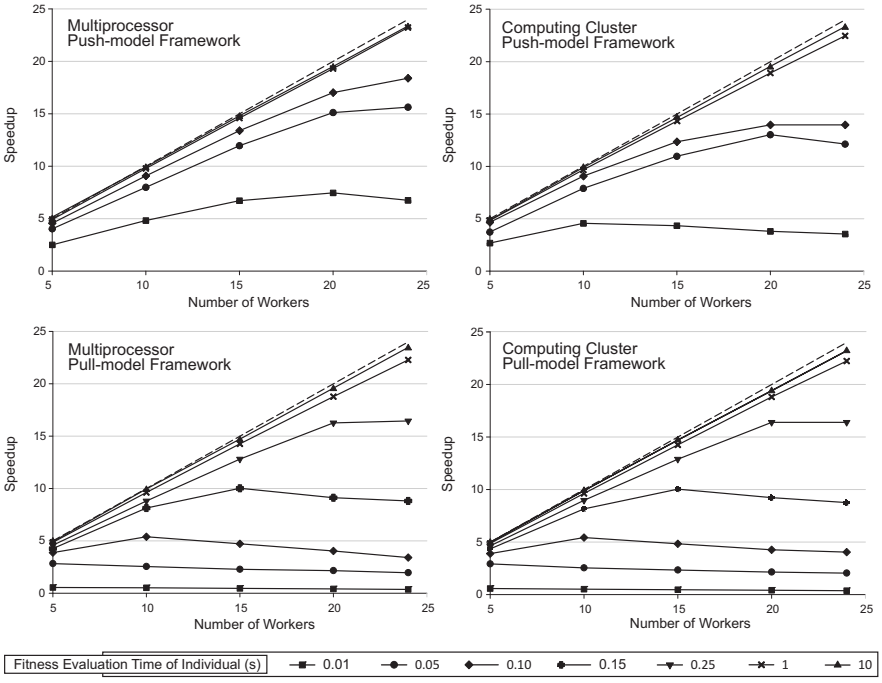

FIG. 3.1: Speedup analysis of the distributed GA run

workers. At some point, the communication overweighs the computation, making the entire system too expensive due to its inherent complexity. The movement of the saddle point to the right hand side with increasing fitness evaluation time is also an expected behaviour. Moreover, the pull-model framework has a slower pace of efficiency improvement with increasing fitness evaluation time. This is the effect of using pull, instead of push, method for acquiring jobs, as explained above. If one compares the left hand side diagrams with their right counterparts (architectural difference), it can be seen that push-model framework behaves better on a computing cluster than on a many core multiprocessor. The explanation of this result can be found in the communication overhead, which will be explained in more detail in the next section. In contrast, pull-model framework behaves almost identically on both architectures.

3.2. Rating the influence of communication overhead

In order to assess the influence of the communication overhead, we have analyzed the functional dependencies between:

- *Communication overhead and number of individuals in one generation*, with varying fitness evaluation time of an individual and fixed chromosome length;

- *Communication overhead and the size of total data transfer* determined by chromosome length, with varying fitness evaluation time of an individual and fixed number of individuals in one generation.

The most significant factor that influences the overhead turns out to be the fitness evaluation time of an individual. Since the master has to wait for all individuals of a generation to be evaluated in order to carry on with the rest of the algorithm, the overhead influence increases with decreasing fitness evaluation time of an individual. Having this in mind, all tests were performed with varying fitness evaluation time of an individual.

3.2.1. How size of a generation influences the communication overhead

In this analysis, the chromosome length was fixed to a relatively low value of 10 parameters of type *double*. The majority of the real-world problems the authors dealt with in the past had approximately this chromosome length, therefore the motivation to quantify their inherent overhead. In order to simplify the comparison between different architectures (multiprocessor or computing cluster) the number of engaged workers was fixed to 24, as explained above.

Figure 3.2 contains four diagrams, one for each order of magnitude of the fitness evaluation time of an individual. The overhead is quantified as a relative value $(T_p - T_t)/T_t$, where T_p is an average time needed to evaluate a generation taken from 100 consecutive runs, and T_t denotes theoretical ideal time value obtained by dividing the time needed for a sequential run with 24 (fixed number of workers).

The first obvious observation from the diagrams shown in Figure 3.2 is that the number of individuals per generation shows significant influence on the overhead only in the cases of more time consuming fitness evaluations. The result makes sense if one considers the size of a generation to be directly proportional to the workers' synchronization period. Larger generation means less frequent synchronizations, which leads to lower relative overhead values. For fitness evaluation time that exceeds 1s, the relative overhead falls below 10% for an averagely sized generation of 500 individuals.

Moreover, if one compares measured performance of two proposed frameworks with varying generation size, it is noticeable that pull-model framework performs considerably worse with less time consuming evaluations and the overhead falls at a slower rate with increase of evaluation duration of an individual. On the other hand, considering long enough fitness evaluations, the lines that indicate two frameworks converge toward each other, as expected.

3.2.2. How the size of a chromosome influences the communication overhead

The focus of this section is to estimate how the amount of data being sent to the workers influences overall performance of two frameworks on two different hardware architectures. The benchmarks have been performed for various chromosome

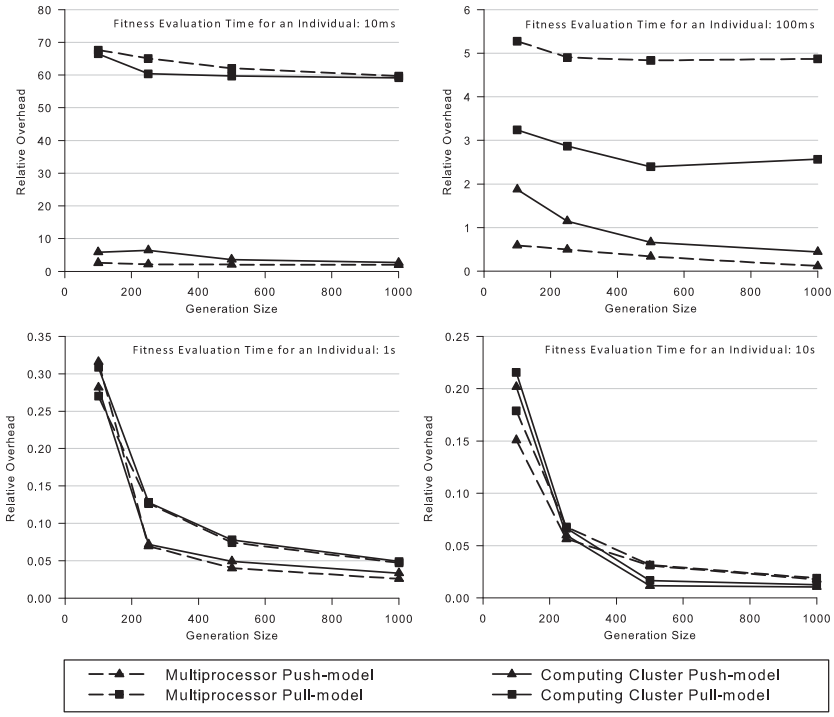

FIG. 3.2: Relative overhead as a function of number of individuals within a generation

sizes represented by 10, 10^2 and 10^3 parameters of type *double*, 500 individuals per generation and 24 active workers. Therefore, overall useful application data being transferred over the network ranges from 40KB up to 4MB. In Figure 3.3, the relative overhead defined above is shown as a function of the amount of all the useful data being transferred through the network. Values were averaged using 100 consecutive generations. The results for fitness evaluation time of an individual that takes less than 100ms are not shown, since their overhead is too high to be considered for any kind of practical use. The first obvious observation is that with increasing complexity, as overhead value falls, the data exchange matters less and less. The push-model framework keeps the leading position in the whole range of various fitness evaluation times of an individual. The performance difference between push-model and pull-model framework decreases quickly as fitness evaluation time of an individual grows.

It is even more interesting how the performance of the proposed frameworks depends on running within computing cluster or multiprocessor environments. If one focuses only on ribbons representing measurements of a push-model framework,

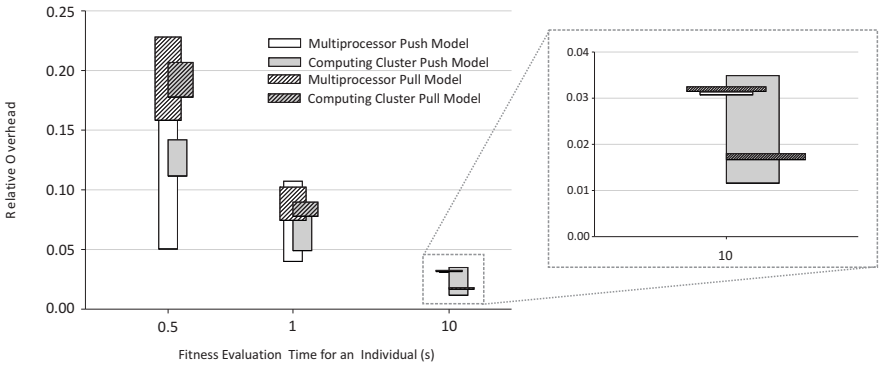

FIG. 3.3: Relative overhead as a function of total data transferred through the network

it can be seen that the multiprocessor ribbon is completely covering the computing cluster ribbon. The same trend is also present for pull-model framework and the explanation of this behavior is simple. While the amount of transferred data is low, the relative overhead is lower for multiprocessor, since the entire data traffic is handled by a single network controller without reaching its full capacity. On the other side, when the amount of data is large, the transfer can be better handled with multiple network controllers in the computing cluster environment.

4. Case study

An automatic calibration of distributed hydrologic model has been chosen to demonstrate the ability of the proposed frameworks to speed up calculations in real-world environmental engineering problems. The hydrologic model calibration is formulated as the following optimization problem: for a given hydrologic model, find the values of parameters (which cannot be measured) to achieve the smallest possible error of the model [16, 17]. The hydrologic model used in this study was based on SWAT model [18] and requires a large number of input parameters: meteorological, topographic, pedological, parameters regarding vegetation, etc.

Hydrologic model of the Drina river basin (Figure 4.1) has been divided into 98 watersheds consisting of approximately 25000 hydrologic response units (HRUs). Climatic inputs used in the model included historical daily precipitation, maximum and minimum temperature, solar radiation data, relative humidity, and wind speed for a period of 3 years. For the given period, measured outflows at several junctions are known. In order to fit model results to the measurements, estimation of the following 11 parameters was performed: maximum water volume to remain on plants in a given day, soil porosity, filtration coefficient, the maximum water volume

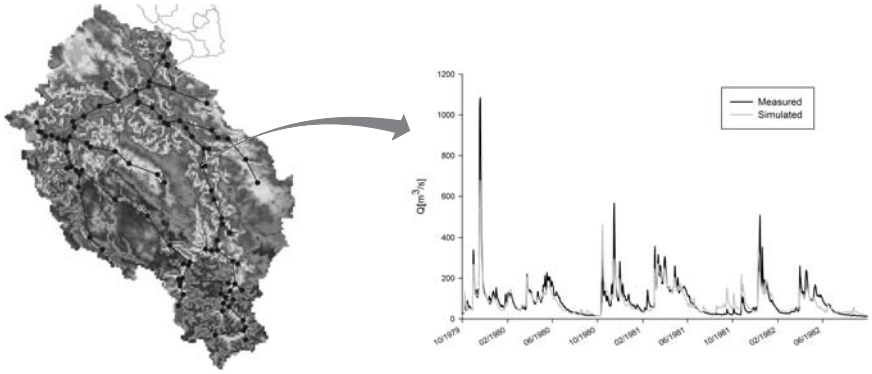

FIG. 4.1: Calibration of the Drina river basin hydrologic model. Comparison between measured and calculated outflows at Prijepolje hydroprofile

to remain on fully grown plants, surface runoff linear reservoir time constant, ground water discharge linear reservoir time constant, time constant of linear reservoir representing rivers, base temperature for the start of formation of the snow pack, rainfall gradient, gradient of temperature drop with an increase in altitude and snow melting factor (meanings of parameters are given in [18, 19]). Each individual was encoded to represent a set of possible values for the model parameters.

The problem of calibration of the Drina river basin hydrologic model is the multiobjective optimization problem with two objective functions to minimize: (1) the root mean square error (RMSE) of the observed *vs.* calculated outflows, which tend to better fit higher (peak) outflows; (2) the logarithmic error (LOGE) that uses the logarithms of the observed and calculated outflows, rather than their original values, and favours lower (basic) outflows. For solving the model calibration problem we used NSGA-II algorithm [20]. The following set of algorithm parameters was adopted: a population size of 500 individuals, a maximum of 100 generations, a simulated binary crossover with a probability of 0.9 and a distribution index of 20, and polynomial mutation with a distribution index of 20 and probability of $1/l$, where l is the chromosome length. All results are taken from 10 independent runs.

Evaluation of a single individual requires one run of model simulation for a given parameter set, which on average takes about 30 seconds for this model. Consequently, sequential execution of GA with 500 individuals in 100 generations would last for more than 17 days. GA based automatic calibration of the distributed hydrologic model of the Drina river basin has been parallelized using pull model framework and executed on computing cluster consisted of 5 servers, each equipped with two 16-core AMD Opteron 6272 and 48GB RAM and 10k rpm hard disk drive. The pull-model framework has been chosen for its inherent flexibility and acceptable relative overhead, which is less than 3% for evaluation lengths ex-

ceeding 10s, according to performed benchmarks. In order to examine the benefits of parallel execution we have measured how increasing the number of processors affects total runtime of calibration.

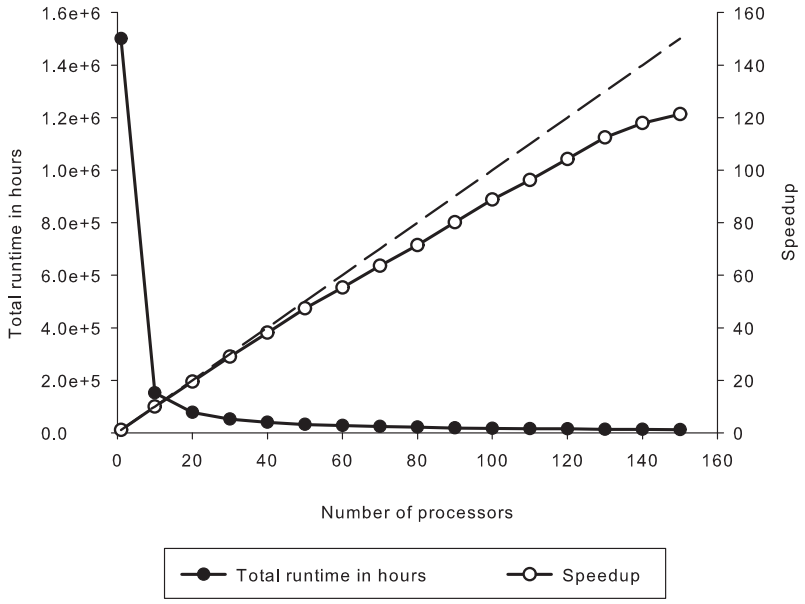

FIG. 4.2: The total runtime in hours and speedup of automatic calibration using pull-model framework on the computing cluster

Figure 4.2 shows total runtime in hours and speedup of automatic calibration using pull-model framework on the described computing cluster. As it can be seen in Figure 4.2, the speedup is almost linear, and runtime graph steeply declines at the beginning since 10 processors finish evaluation almost 10 times faster than one processor.

5. Conclusions and future work

This paper presents two service-oriented frameworks for parallel evaluation of the individuals in GA using WCF web services. The frameworks follow the master-slave parallelization model, which does not require any change of the sequential GA, since the fitness evaluation of an individual is separated from the rest of the algorithm. Applied web-service oriented architecture enables the frameworks to be used on various distributed platforms ranging from multicomputer to computing

clusters and grids. The frameworks provide suitable mechanisms intended to distribute individuals and collect the evaluation results, independently of applied selection, crossover and mutation operators. The first framework is based on the push task distribution model and the second framework uses the pull task distribution model, which is more firewall and grid friendly.

Performed benchmarks have shown that the fitness evaluation time of an individual is the major influence on performance of the frameworks. The frameworks exhibited significant communicational overhead for short evaluations. The overhead has been influenced not only by fitness evaluation time, but also by chromosome length and generation size. In addition, the pull model has shown even lower performance, due to the more complex communication protocol. In contrast, for time consuming evaluations, both frameworks have shown equal performance, achieving almost ideal speedups. The case study has verified that the frameworks are suitable for running GAs with computationally expensive evaluations which are common in real-world applications.

However, considerable work remains to be done. There is a major drawback in the pull model framework as the manager becomes a single point of failure, which can be addressed through redundancy. The second drawback of both frameworks is their inherent inelasticity in occupying computing resources. Even when there are no pending evaluation tasks, the frameworks keep resources reserved, preventing other users from employing them.

The authors are currently working on integrating grid job allocator Work Binder [21] in order to fulfill the elasticity request in the European Grid Initiative (EGI) grid environment. The next step might be to modify Work Binder in order to use standard EC2 interface to independently manage cloud instances according to the system load.

REFERENCES

1. C. A. Coello Coello, G. B. Lamont and D. A. Van Veldhuizen: *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2nd ed. Springer, New York, 2007.
2. K. Deb: *Multi-objective optimization using evolutionary algorithms*. Wiley, UK, 2001.
3. E. Alba, G. Luque and S. Nesmachnow: *Parallel metaheuristics: recent advances and new trends*. Int. T. Oper. Res. **20**, no. 1 (2013), 1–48.
4. A. Jaimes and C. A. Coello Coello: *Applications of Parallel Platforms and Models in Evolutionary Multi-Objective Optimization*. In: Biologically-Inspired Optimisation Methods (A. Lewis, S. Mostaghim, and M. Randal, eds.), Springer, Berlin, 2009, 23–49.
5. E. Talbi, S. Mostaghim, T. Okabe, H. Ishibuchi, G. Rudolph and C. A. Coello Coello: *Parallel Approaches for Multiobjective Optimization*. In: Multiobjective Optimization. Interactive and Evolutionary Approaches (J. Branke, K. Deb, K. Miettinen, and R. Slowinski, eds.), Springer, Berlin, 2008, 349–372.
6. M. Tomassini: *Parallel and distributed evolutionary algorithms*. In: Evolutionary Algorithms in Engineering and Computer Science (K. Miettinen, M. Mkel, P. Neittaanmki, and J. Periaux eds.), Wiley, New York, 1999, 113–133.

7. C. Gagne, M. Parizeau and M. Dubreuil: *Distributed beagle: an environment for parallel and distributed evolutionary computations*. In: Proc. of the 17th Annual Int. Symp. on High Performance Computing Systems and Applications (HPCS), 2003, 201–208.
8. S. Cahon, N. Melab and E. Talbi: *ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics*. J. Heuristics **10**, no. 3 (2004), 353–376.
9. E. Alba, G. Luque, J. Garcia-Nieto, G. Ordonezand G. Leguizamon:*MALLBA: a software library to design efficient optimisation algorithms*. Int. J. Innov. Comput. Appl. **1**, no. 1 (2007), 74–85.
10. B. A. Hoverstad: *Simdist: a distribution system for easy parallelization of evolutionary computation*. Genet. Program. Evol. M., **11**, no. 2 (2010), 185–203.
11. D. Booth, H. Haas, F. Mccabe, E. Newcomer, M. Champion, C. Ferris, C. et al.: *Web services architecture*. W3C Web Services Architecture Working Group Note. Available at: `http://www.w3.org/TR/ws-arch/`, 2004.
12. D. Chappell: *Introducing Windows Communication Foundation*, MSDN Library. Available at: `http://download.microsoft.com/download/C/2/5/C2549372-D37D-4F55-939A-74F1790D4963/Introducing_WCF_in_NET_Framework_4.pdf`, 2010.
13. B. Stojanovic, V. Simic, M. Ivanovic, A. Kaplarevic-Malisic and A. Stanojevic: *WCF Platform for Distributed Evaluation in Evolutionary Algorithms*. In: Proc. of the 4th Int. Conf. Science and Higher Education in Function of Sustainable Development (SED), Uzice, Serbia (October 2011), 8–13.
14. J. Albahari and B. Albahari: *C# 3.0 in a Nutshell*. 3rd ed. O'Reilly Media, 2007.
15. A. Jones: *C# Programmer's Cookbook*. Microsoft Press, 2003.
16. D. P. Solomatine, Y. Dibike and N. Kukuric: *Automatic calibration of groundwater models using global optimization techniques*. Hydrolog. Sci. J., **44**, no. 6 (1999), 879–894.
17. N. Milivojevic, Z. Simic, A. Orlic , V. Milivojevic and B. Stojanovic: *Parameter estimation and validation of the proposed SWAT based rainfall-runoff model: Methods and outcomes*. Journal of Serbian Society for Computational Mechanics, **3**, no. 1 (2009), 86–110.
18. J. G. Arnold and N. Fohrer: *SWAT2000: Current capabilities and research opportunities in applied watershed modeling*. Hydrol. Process., **19**, no. 3 (2005), 563–572.
19. Z. Simic, N. Milivojevic, D. Prodanovic, V. Milivojevic and N. Perovic: *SWAT-Based Runoff Modeling in Complex Catchment Areas Theoretical Background and Numerical Procedures*. Journal of the Serbian Society for Computational Mechanics, **3**, no. 1 (2009), 38–63.
20. K. Deb, A. Pratap, S. Agarwal and T. Meyarivan: *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation **6** (2002), 182–197.
21. B. Marovic, M. Potocnik and B. Cukanovic, B: *Multi-application bag of jobs for interactive and on-demand computing*. Scalable Computing: Practice and Experience, **10**, no. 4 (2009), 413–418.

Miloš Ivanović, Ana Kaplarević-Mališić, Višnja Simić and Boban Stojanović
Faculty of Science
Department of Mathematics and Informatics
P. O. Box 60
34000 Kragujevac, Serbia
mivanovic@kg.ac.rs (M. Ivanović)
ana@kg.ac.rs (A. Kaplarević-Mališić)
visnja@kg.ac.rs (V. Simić)
bobi@kg.ac.rs (B. Stojanović)