



HIGH BUSINESS-TECHNICAL SCHOOL OF UZICE, SERBIA

4th International Conference
"SCIENCE AND HIGHER EDUCATION IN FUNCTION OF
SUSTAINABLE DEVELOPMENT"
SED 2011
7th and 8th of October 2011, Uzice, Serbia

WCF PLATFORM FOR DISTRIBUTED EVALUATION IN EVOLUTIONARY ALGORITHMS

B. Stojanovic¹, V. Simic¹, M. Ivanovic¹, A. Kaplarevic-Malisic¹, A. Stanojevic¹

¹ Department of Mathematics and Informatics, Faculty of Science, Kragujevac, SERBIA, bobi@kg.ac.rs

Summary: *Evolutionary algorithms are powerful techniques for optimization of complex systems. These methods require large number of evaluations of candidate solutions which take huge processor time. In this paper, service-oriented platform for distributed evaluation using WCF (Windows Communication Foundation) web services is presented. The platform developed can be easily incorporated into any evolutionary algorithm, giving mechanism for distribution of individuals and collection of evaluation results. This concept provides parallelization of evolutionary algorithms independently of geographic location and platform where evaluation is performed. Performed tests have shown that proposed platform gives significant speedup, especially in evaluation of large-scale problems.*

Keywords: *evolutionary, genetic, algorithm, distributed, optimization*

1. INTRODUCTION

Evolutionary algorithms (EAs) are stochastic search methods that simulate the process of natural evolution. These algorithms have proven themselves as a robust and powerful mechanism when it comes to solving challenging optimization problems.

Evolutionary algorithm mimics 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 best candidates for reproduction, one has to determine the fitness of each individual in the population by evaluating it with respect to each objective. Each evaluation of the solution for the real-world problem usually requires running a complex, time consuming computer simulation, and so the use of distributed computing is a necessity.

Different approaches for the use of parallelism in evolutionary algorithms for optimization have been proposed in the literature and surveys have been written [4], [5]. Three major parallel models of EAs exist: the master-slave model, the island model and the diffusion model.

In the master-slave model the objective functions evaluations are distributed among several slave processors, while a master processor executes the rest of EA. In the island model, the population is divided into several sub-populations (islands) and serial EA 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 many parameters and design decisions. In the diffusion model the population is spatially distributed onto a neighborhood structure which is usually a two-dimensional rectangular grid. There is one individual per grid point and ideally, one processor per individual. Therefore this model is called fine grained. The selection and mating is confined to a small neighborhood around each individual, and since individuals who take part in the selection are distributed among several processors the communication costs tend to be high.

The purpose of this paper is to introduce platform for distributed evaluation of individuals in evolutionary algorithms based on the master-slave model. Evaluations of individuals are performed by Windows Communication Foundation (WCF) web services. WCF is a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications that communicate across the web.

The rest of the paper is organized as follows: in section 2 we review related work. Then, the proposed platform is described in Section 3, followed by the presentation of experiment results and discussion in Section 4. Some concluding remarks are presented in the last section.

2. RELATED WORK

There are several protocols and libraries which provide aid in the development of parallel systems, by hiding some of the network connection and transmission details. The most important are MPI and OpenMP. MPI, the Message Passing Interface, is the standard for development of parallel systems on distributed memory systems, whereas OpenMP (Open Multi-Processing) is the standard in shared memory systems [3], [4]. The parallelization and distribution functionality in evolutionary software packages is often built on top of libraries implementing MPI, like in Simdist [5] and ParadisEO [8].

Grid technologies support the sharing and coordinated use of diverse resources in dynamic virtual organizations. The Globus toolkit is an open-source, community-based set of software tools to enable the aggregation of compute, data, and other resources to form computational grids. Multi population algorithm using master-slave parallel model in a Globus toolkit based grid environment was implemented in [6]. Implementation of the framework requires selection of certain parameters: the number of subpopulations, the size of the subpopulations, the number of migrants in an optimization step, and the number of generations that are computed in one optimization step. ParadisEO-CMW [7], which is re-designed ParadisEO, provides a rich set of parallelization strategies. But it is only intended for grids consisting of multiple Condor pools combined via flocking. This form of grid is not as popular as Globus based grids and it is questionable if it can be seen as a grid in the commonly accepted meaning at all. JG2A [9] was created as an extension of JGA [10] to take advantage of grid technologies, allowing instances parallelization and population evaluation parallelization. JG2A uses the Globus Toolkit 4 grid middleware, but requires Condor as underlying scheduler on the different sites. This makes it inflexible because in general other local resource managers than Condor are used at different computing sites of a grid and these sites may be under different administrative control (which makes it hard to enforce the deployment of Condor on all those sites). In [11] authors implement grid-enabled framework MOGA-G for multi-objective optimization in a Service-Oriented Architecture using the Globus Toolkit. In the implementation of the MOGA-G framework there are two different services. One service exposes the operations of the multi-objective genetic algorithm to the client, and the other provides operations for running evaluations of the objective function on the computational grid. Another SOA approach to the implementation of parallel evolutionary algorithms can be found in [12]. Two-level hierarchical parallel genetic algorithm has been implemented in a distributed computational Grid, with multiple subpopulations distributed across the Grid resources. The evaluation of candidate solutions in these subpopulations is then performed using the Master-Slave paradigm. In [13] Globus was used to develop gPAES, a Grid extension of the PAES algorithm. In the gPAES, the model of search consists in remotely executing a number of sequential PAES algorithms (separately exploring the whole search space) on machines of the Grid and locating the best solutions according some defined metrics. In [14] authors propose hybrid method using Multi-objective Particle Swarm optimization and Binary search methods in order overcome the problem of long waiting time of the master processor in a heterogeneous environment. However the implementation of the method requires selecting certain parameter, which depends on the heterogeneous resources and the estimated load on them.

To the best of our knowledge, there exists no previous framework for distributed evaluation in EAs that uses WCF web services.

3. PROPOSED PLATFORM

The service-oriented distribution of evaluation imposed the use of master-slave model as the most convenient way to parallelize evolutionary algorithm. This model aims at distributing the (objective function) evaluation of the individuals on several slave computing resources while the master node executes the rest of the algorithm in sequential fashion. In terms of web services, the client acts as the master node (by generating and varying the population), and the service acts as the worker (by evaluating the individuals). One drawback of the master-slave model is the communication overload between the processors. Here, we ignore this as we aim to solve very expensive optimization problems.

We developed a distribution subsystem to be inserted as an intermediate layer between the main evolutionary loop on the master, and the evaluation of individuals which takes place on slaves. The master is unaware of the number of slaves evaluating individuals, and which slave evaluates which individual. Each time the generation has to be evaluated, the distribution subsystem is started in a separate thread.

The main part of distribution subsystem is the evaluation pool. When master sends individuals to evaluation, they are being queued in the evaluation pool, and its job is to distribute them to available web services and to assign evaluation result to the corresponding individual.

EvaluationPool is an abstract class that serves as the base for deriving specific classes. Those derived classes use the thread pool provided by the .NET Framework through the ThreadPool class [15]. A *thread pool* is a collection of threads that can be used to perform a number of tasks in the background. This leaves the primary thread free to perform other tasks asynchronously. As many individuals as there is available web services are assigned to those web services in the separate threads from the thread pool, Figure 1. Evaluations can be processed asynchronously, without tying up the primary thread of evaluation pool or delaying the processing of subsequent requests. 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 completed the evaluation. This reuse enables applications to avoid the cost of creating a new thread for each task. If the evaluation on one web service fails for some reason, individual is reassigned to another web service.

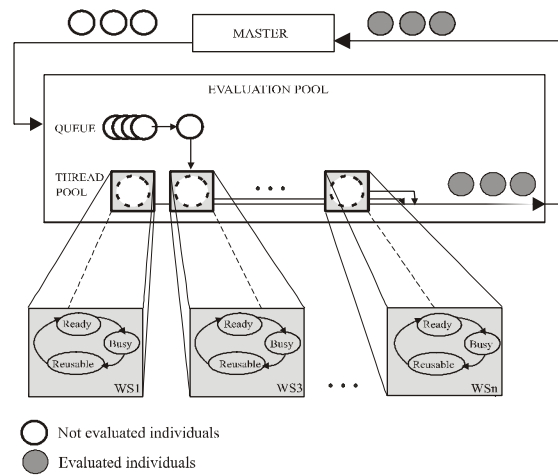


Figure 1: Proposed platform

The developer of the evolutionary system has to separate the evaluation step from the rest of the evolutionary algorithm loop, but has no concern with the details of distributing individuals and retrieving evaluations results. It is the task of the distribution subsystem to distribute individuals as efficiently as possible among the slave nodes, and gather the results.

Web services we use for evaluation are set up using Windows Communication Foundation (WCF). WCF is a framework for building service-oriented applications [16], [17]. It is designed to support distributed computing, where remote services can be consumed by multiple clients. Services expose one or more endpoints where clients send messages to request work. Each endpoint consists of an address specifying where to send messages, a binding describing how to send messages, and a contract describing what the messages contain. WCF includes predefined bindings for most common communication protocols such as SOAP over HTTP, SOAP over TCP, and SOAP over Message Queues, etc. Interaction between WCF endpoint and client is performed using a SOAP envelope. SOAP envelopes comply to a simple XML form that makes WCF platform independent.

Services use Web Services Description Language (WSDL) to share endpoint descriptions with clients, so any WCF client can consume the service, regardless of which platform the service is hosted on. WCF supports interoperability with WCF applications running on the same Windows machine or WCF running on a various Windows machines or standard Web services built on platforms such as Java running on Windows or other operating systems.

We defined a .NET interface `IWcfEvaluationService` to serve as the service contract and implemented the service contract in a .NET class `WcfEvaluationService`, known as the service type, to configure its behavior. The task our WS is to be able to perform evaluation of an individual it receives from client. As illustrated by the code in Figure 2 the operation `Evaluate` that we want our web service to expose is annotated with `OperationContract`. Method `Evaluate` accepts `JobParameters` – individual to evaluate, and returns `EvaluationResult`. Complex types that we used for the method parameters and return values must have a data contract defined for them to be serializable, but due to the limited number of pages, we omitted that part of code in Figure 2.

```

namespace WcfEvaluationServiceLibrary
{
    [ServiceContract]
    public interface IWcfEvaluationService
    {
        [OperationContract]
        EvaluationResult Evaluate(JobParameters par);
    }
    ...
}

```

Figure 2: Interface for WCF web service

4. RESULTS AND DISCUSION

Described distribution subsystem was primarily developed for use on distributed platforms, such as computing cluster. The same or slightly modified mechanism can be used on shared memory and even grid platforms.

Besides portability of clients, the system is robust to a client failure. The algorithm efficiently distributes work to all currently available resources. Even if the evaluation of any individual fails on any node, the simulation will continue running on the remaining healthy nodes.

Accordingly to the way our algorithm work, the only part of the entire simulation that can be improved, i.e. speeded up, is the evaluation of the whole generation. Depending on how long evaluation of an individual lasts and a size of master-slave communication overhead, parallelization is more or less a good choice. Hence, we have performed tests (speed-up analysis) aiming assessing scalability of the proposed system.

Benchmark tests were done on a Beowulf type cluster consisting of 14 nodes, each with a single 2.4GHz Intel Core2Quad Q6600 nodes with 8GB RAM memory, totaling to 56 CPUs with 112GB RAM memory. All nodes run Scientific Linux 5.6. in x86_64 architecture. Since the original MS .Net Framework is not available for UNIX platforms, we employed the open source implementation of .NET Framework – Mono v2.10.5 which almost completely complies to .NET 3.5 standard. PBS Torque has been employed for resource management tasks.

Scalability was analyzed taking into account the average time required for evaluation of a single individual. We measured time needed for evaluation of whole generation and took average value in order to calculate the speed-up coefficient. Size of the population was fixed to the values common to the problems taken into consideration [18], [19]. In order to calculate a couple of fitness parameters needed for multiple criteria optimization, each service takes 10 double variables.

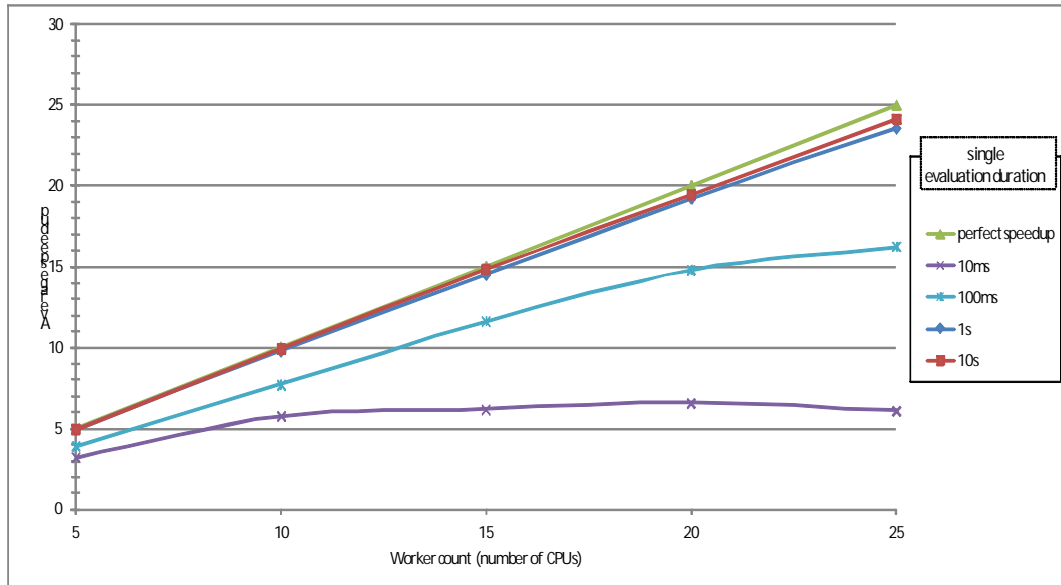


Figure 3: Scalability analysis of the distributed EA run. Speed-up is calculated as T_m/T_f , where T_m is experimentally measured time and T_f is theoretically assumed time for sequential process of evaluation. Perfect speed-up is the theoretical limit assuming that all communications are neglected.

Figure 3 shows average speed-up of evaluation of one generation as a function of a number of workers involved. Speed-up is calculated as T_m/T_t , where T_m is experimentally measured time and T_t is theoretically assumed time for sequential process of evaluation, i.e. product of number of individuals and time needed for evaluating a single individual. Since distributed processing inevitably has to have some communication overhead, we performed testing with different problems complexity determined by average duration of a single individual evaluation.

Scalability analysis shows significant difference in speed-up values among evaluations of different problem complexity quantified by single evaluation duration. As can be seen from Figure 3 for more complex evaluations the speed-up lines converge to the *perfect scalability line*. On the other hand, considering less complex evaluations, speed-up reaches its 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 workers. At some point, the communications overweight computations, so the entire system becomes too expensive due to its complexity.

5. CONCLUSION

This paper presents service-oriented platform for distributed evaluation using WCF web services. The platform can be easily integrated into any existing evolutionary algorithm. It will provide mechanism for distribution of individuals and collection of evaluation results. Performed tests have shown that the platform gives significant speedup and is particularly suitable for running EAs with computationally expensive evaluations which is the common case in real-world applications.

Future work includes implementation of the proposed platform in a Globus toolkit based grid environment. We may be challenged to face with firewall issues, since most of grid sites are protected with the firewall. One possible solution would be to swap the roles of master and worker in communication sense. This would imply involving of job manager. In that case, the job manager is a WCF service which plays a role of intermediary between the evaluation pool and workers. Master, as a WCF client, sends jobs and to the manager and receives results. The workers on the remote nodes invoke the job manager to provide them with evaluation jobs. Since, the only callable Web service is the job manager, the firewall issues are solved.

Acknowledgments

The part of this research is supported by Ministry of Science in Serbia, Grants III41007, OI174028, TR37013, III44010, and TR 14005, and FP7 ICT-2007-2-5.3 (224297) ARTreat project.

REFERENCES

1. **Jaimes, A.; Coello, C.:** *Applications of Parallel Platforms and Models in Evolutionary Multi-Objective Optimization*, Biologically-Inspired Optimisation Methods, Studies in Computational Intelligence. s.l. : Springer Berlin / Heidelberg, Vol. 210, pp. 23-49, 2009.
2. **Talbi, E-G. et al.:** *Parallel Approaches for Multiobjective Optimization*, Multiobjective Optimization, Lecture Notes in Computer Science. Heidelberg : Springer, pp. 349-372, 2008.
3. **Dagum, L.; Menon, R.:** *OpenMP: an industry standard api for shared-memory programming*, Comput. Sci. Eng. IEEE , Vol. 5, pp. 46-55, 1998.
4. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard Version 2.2. [Online] September 2009. <http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf>.
5. **Hoverstad, B. A.:** *Simdist: a distribution system for easy parallelization of evolutionary computation*, Genet Program Evolvable Mach, Vol. 11, pp. 185-203, 2010.
6. **Cahon, S.; Melab, N.; Talbi, E-G.:** *ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics*, Journal of Heuristics, Vol. 10, pp. 353-376, 2004.
7. **Limmer, S.; Fey, D.:** *Framework for Distributed Evolutionary Algorithms in Computational Grids*, Advances in Computation and Intelligence, Lecture Notes in Computer Science. Heidelberg : Springer, Vol. 6382, pp. 170-180, 2010.
8. **Melab, N.; Talbi, E-G.; Cahon, S.:** *An enabling framework for parallel optimization on the computational grid*. Cardiff, UK, Proc. of the 5th IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGRID'2005), 2005.
9. **Ramírez, M. A., et al.:** *JG2A: A Grid-Enabled Object-Oriented Framework for Developing Genetic Algorithms*, COPA, 2009.

10. **Medaglia, A. L.; Gutiérrez, E.:** *JGA: An Object-Oriented Framework for Rapid Development of Genetic Algorithms*, in Rennard, J. P.: *Handbook of Research on Nature Inspired Computing for Economics and Management*. Hershey, PA, USA, 2006.
11. **Shenfield, A.; Fleming, P.:** *A Service Oriented Architecture for Decision Making in Engineering Design*, *Advances in Grid Computing - EGC 2005*, *Lecture Notes in Computer Science*, Vol. 3470, pp. 334–343, 2005.
12. **Lim, D, et al.:** *Efficient hierarchical parallel genetic algorithms using grid computing*, *Future Generation Computer Systems*, Vol. 23, pp. 658-670, 2007.
13. **Luna, F.; Nebro, A. J.; Alba, E.:** *Observations in using Grid-enabled technologies for solving multi-objective optimization problems*, *Parallel Computing*, Vol. 32, pp. 377-393, 2006.
14. **Mostaghim S.; Branke J.; Lewis A.; Schmeck H.:** *Parallel Multi-objective Optimization using Master-Slave Model on Heterogeneous Resource*, *IEEE, Congress on Evolutionary Computation (CEC)*. pp. 1981 - 1987, 2008.
15. MSDN Library. *How to: Use a Thread Pool (C# Programming Guide)*. [Online] Microsoft.
[http://msdn.microsoft.com/en-us/library/3dasc8as\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/3dasc8as(v=VS.90).aspx).
16. **Chappell, D.:** MSDN Library. *Introducing Windows Communication Foundation in .NET Framework 4*, 2010. <http://msdn.microsoft.com/en-us/library/ee958158.aspx>.
17. **Skonnard, A.:** *Learn The ABCs Of Programming Windows Communication Foundation*, MSDN Magazine, February 2006.
18. **Coello, C. C.; Lamont, A.; Gary, B.; Veldhuizen, D. A. van:** *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. 2nd Edition.: Springer-Verlag New York, Inc., 2007.
19. **Deb, K.:** *Multi-objective optimization using evolutionary algorithms*. Chichester, UK : Wiley, 2001.
20. **Deb, K.:** *Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction*. 2011. KanGAL Report No.2011003.
21. **J., Dongarra et al.:** *High-performance computing: clusters, constellations, MPPs, and future directions*, *Comput. Sci. Eng.*, Vol. 7, pp. 51–59, 2005.