PES: A system for parallelized fitness evaluation of evolutionary methods

Onur Soysal, Erkin Bahçeci, and Erol Şahin

Kovan research group
Department of Computer Engineering
Middle East Technical University
06531 Ankara, Turkey
{soysal, erkinb, erol}@ceng.metu.edu.tr
http://kovan.ceng.metu.edu.tr

Abstract. The paper reports the development of a software platform, named PES (Parallelized Evolution System), that parallelizes the fitness evaluations of evolutionary methods over multiple computers connected via a network. The platform creates an infrastructure that allows the dispatching of fitness evaluations onto a group of computers, running both Windows or Linux operating systems, parallelizing the evolutionary process. PES is based on the PVM (Parallel Virtual Machine) library and consists of two components; (1) a server component, named PES-Server, that executes the basic evolutionary method, the management of the communication with the client computers, and (2) a client component, named PES-Client, that executes programs to evaluate a single individual and return the fitness back to the server. Performance of PES is tested for the problem of evolving behaviors for a swarm of mobile robots simulated as physics-based models, and the speed-up characteristics are analyzed.

1 Introduction

Evolutionary Robotics[6] is a new approach for designing autonomous robots. It considers autonomous robots as artificial organisms that can be selectively "bred" based on their fitness for the tasks that they are being designed for. An active research track in Evolutionary Robotics is the development of behaviors for simulated robot systems. In these studies, the goal is to evolve controllers that use the sensory information of the robot to control robot's actuators such that the robot accomplishes a desired task. Initially a population of genotypes that encode the controllers is given. Then the robots are simulated under the control of the controller specified by the genotype and its fitness is evaluated. The fittest controllers are then allowed to reproduce as defined by a set of genetic operators, and the process is repeated. Evolving behaviors in simulated robot systems require large amounts of computational power, mainly due to the evaluation of different individuals. Therefore the computational requirements of

using evolutionary methods tend to be proportional to the computational requirements of evaluating a single individual. This creates a major bottleneck for evolutionary methods.

Recently there have been projects[3, 5] that aim to create platforms that can use the idle processing power of computer systems that are connected over a network. SETI@home[3] is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). The downloaded program runs as a background task or a screen-saver getting chunks of radio telescope data from a server and performing a computation intensive processing of the data to seek signs of artificial signals and reporting the results back to the server. This platform made it possible for the SETI project to utilize a total computing power that equals or surpasses supercomputers.

This paper reports the development of a platform, named *PES* (Parallellized Evolution System), that parallelizes an evolutionary method on a group of computers connected via a network. In the next section, we describe PES and its specifications. Section 2 describes the implementation of PES, explaining server and client components of the system. Section 3 analyzes the speed-ups obtained using PES for the problem of evolving behaviors for a swarm of simulated mobile robots. Section 4 summarizes the results and discusses the shortcomings of the system and future development directions of the system.

2 PES

PES[1] is a platform to parallelize evolutionary methods on a group of computers connected via a network. It separates the fitness evaluation of genotypes from other tasks (such as selection and reproduction) and distributes these evaluations onto a group of computers to be processed in parallel. PES consists of two components; (1) a server component, named PES-Server, that executes the evolutionary method, the management of the communication with the client computers, and (2) a client component, named PES-Client, that executes programs to evaluate a single individual and return the fitness back to the server. Figure 1 shows the structure of a PES system.

PES provides the user with an easy interface that relieves him from dealing with the communication between server and client processes. PES-Client is developed for both Windows and Linux, enabling the PES system to harvest computational power from computers running either of these operating systems. An easy-to-use framework for implementing evolutionary methods, and the interoperability of the system distinguishes PES from other systems available and makes it a valuable tool for evolutionary methods with large computational requirements.

PES uses PVM (Parallel Virtual Machine)[4][2], a widely utilized message passing library in distributed and parallel programming studies, for communi-

[1] Documentation and source code of PES is freely available at http://kovan.ceng.metu.edu.tr/software/PES/.

[2] Available at http://www.csm.ornl.gov/pvm/pvm_home.html.

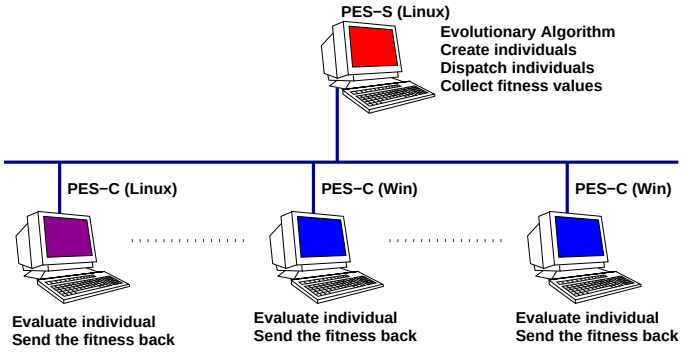


Fig. 1. Sketch of PES system is shown. The PES-Server runs on a Linux machine and handles the management of the evolutionary method. It executes the selection and reproduction of the individuals (genotypes) which are then dispatched to a group of PES-Clients (running both Windows and Linux systems). The individuals are then evaluated by the clients and their fitnesses are sent back to the server.

cation between the server and the clients. We have also considered MPI[2] as an alternative to PVM. MPI is a newer standard that is being developed by multiprocessor machine manufacturers and is more efficient. However PVM is more suitable for our purposes since (1) it is available in source code as free software and is ported to many computer systems ranging from laptops to CRAY supercomputers, (2) it is inter-operable, i.e. different architectures running PVM can be mixed in a single application, (3) it does not assume a static architecture of processors and is robust against failures of individual processors.

PES wraps and improves PVM functionality. It implements a time-out mechanism to detect processes that have crashed or have entered an infinite loop. It provides *ping*, *data* and *result* message facilities. Ping messages are used to check the state of client processes. Data messages are used to send task information to client processes and result packages are used to carry fitness information from clients.

Now we will describe the PES-Server and PES-Clients.

2.1 PES-Server

PES-Server provides a generic structure to implement evolutionary methods. This structure is based on Goldberg's basic Genetic Algorithm[1] and is designed to be easily modified and used by programmers. The structure assumes that fitness values are calculated externally. In its minimal form, it supports tournament selection, multi-point cross-over and multi-point mutation operators.

PES-Server maintains a list of potential clients (computers with PES-Client installed), as specified by their IP numbers. Using this list, the server executes an evolutionary method and dispatches the fitness evaluations of the individuals

to the available clients. The assignment passes the location of the executable to be run on the client as well as the parameters that represent that particular individual and the initial conditions for the evaluation. Then it waits for the clients to complete the fitness evaluation and get the computed fitness values back.

PES-Server contains fault detection and recovery facilities. Using the *ping* facility the server can detect clients that have crashed and assign the uncompleted tasks to other clients. In its current implementation, the server waits for the results of fitness evaluations from all the individuals in a generation before dispatching the individuals from the next generation.

2.2 PES-Client

PES-Client acts as a wrapper to handle the communication of the clients with the server. It fetches and runs a given executable (to evaluate a given individual) with a given set of parameters. It returns the fitness evaluations, and other data back to the server.

Client processes contain a loop that accepts, executes and sends results of tasks. Client processes reply to *ping* signals sent by the PES-Server to check their status. Crashed processes are detected through this mechanism.

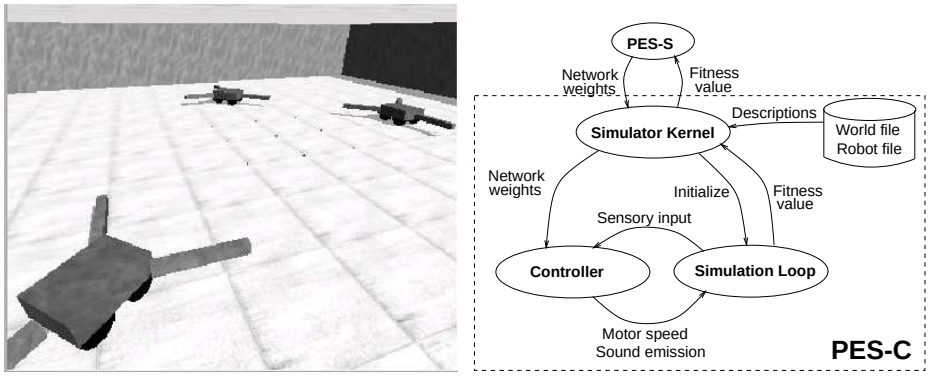
PES-Clients are developed for single processor PC platforms running Windows and Linux operating systems. Note that to use clients with both operating systems the fitness evaluating program should be compilable on both systems. In its current implementation, these clients have the fitness evaluation component embedded within them to simplify the communication. Yet, once the clients are installed, the fitness evaluation component of the system can be updated using *scp* (secure copy) utility from the server.

3 Experimental Results

The PES platform is developed as part of our work within the Swarm-bots project[3][7, 8], for evolving behaviors for a simulated swarm of mobile robots. We have conducted experiments to evolve behaviors for clustering a swarm of robots and analyzed the speed-up and efficiency of the PES system in this task.

The swarm of robots and their environment are modeled using ODE (Open Dynamics Engine), a physics-based simulation library, Figure 2(a). The parameters of this simulation and parameters of the controller (network weights) are passed to the PES-Client from the PES-Server. The simulator constructs the world and runs it, by solving differential dynamic equations. Movements of the robots are determined by their controller as specified by the genotype. This controller uses the sensors of the robots and moves the robots for 2000 time steps. Then a fitness value is computed based on a measure of clustering achieved. This fitness value is then returned to PES-Server, Figure 2(b).

[3] More information can be found at http://www.swarm-bots.org.



(a) **(b)**

Fig. 2. (a) A snapshot of the environment being simulated: Three mobile robots, distributed in an arena are enclosed by walls, can be seen. The rods emanating out of the robot bodies visualize the proximity sensing capabilities of the robots. **(b)** Architecture of PES-Client being used.

3.1 The experimental set-up

We have installed PES-Clients to 12 PC's of a student laboratory at the Computer Engineering Department of Middle East Technical University, Ankara, Turkey. During the experiments, these computers were being used by other users and each of them had different workloads that varied in time. The population size is set as 48, requiring that 48 fitness evaluations take place during each generation. 30 generations were evolved.

Figure 3 plots the load of the 12 processors in time, during the evaluation of five generations. PES-Server waits for fitness evaluations of all the individuals in a generation before the selection and reproduction of the individuals of the next generation. In the plot, the vertical lines separate the generations. Within each generation, 48 fitness evaluations are calculated, which are visible as dark diamonds or dark horizontally stretched hexagons. It can be seen that the fitness evaluation time varies between different cases. There are two major causes of this. First, each processor has a different and varying workload depending on the other users of that computer. Second, the physics-based simulation of the swarm of robots slows down dramatically as robots collide with each other and the walls in the environment. As a result, the computation required by different simulations vary drastically.

In order to analyze the speed-ups achieved through the use of the PES system and its efficiency, we have repeated the evolution experiment using 1, 2, 3, 6 and 12 processors. The data obtained is used to compute the following two measures:

$$S_p = \frac{\text{Time required for single machine}}{\text{Time required for p machines}}$$

$$E_p = \frac{\text{Speed-up with p processors}}{\text{p}}$$

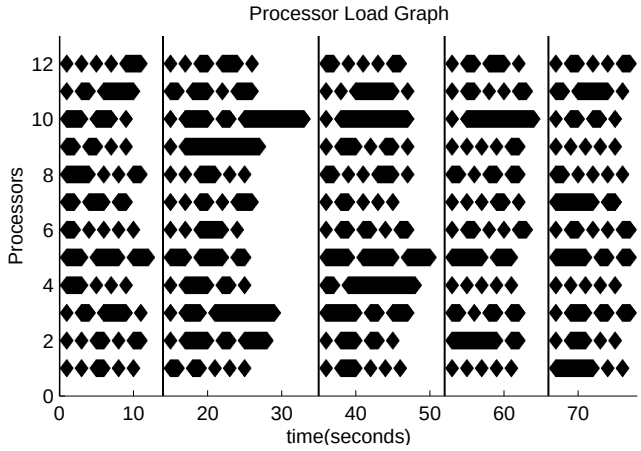
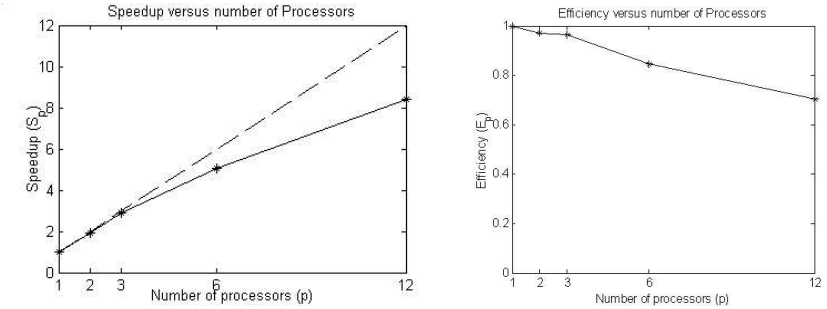


Fig. 3. The load of the 12 processors during 5 generations of evolution. See text for discussion.

The results are plotted in Figure 4(a,b). Ideally S_p should be linear to number of processors and E_p should be 1. The deviance is a result of the requirement that all individuals need to be evaluated before moving on to the next generation. As a consequence of this, after the dispatching of the last individual in a generation, all-but-one of the processors has to wait for the last processor to finish its evaluation. This causes a decrease in the speed-up and efficiency plots. Note that, apart from the number of processors, these measures also depend on two other factors: (1) the ratio of total number of fitness evaluations in a generation to the number of processors, (2) the average duration of fitness evaluations and their variance.



(a) **(b)**

Fig. 4. (a) Speed-up is plotted against number of processors. **(b)** Efficiency is plotted against number of processors.

In Section 2, we described a ping mechanism that is implemented to check whether a processor has crashed or not. This mechanism is crucial since we envision PES to harvest idle processing powers of existing computer systems and cannot make assumptions about the reliability of the clients. Figure 5 shows the ping mechanism at work during an evolution where we had 20 fitness evaluations in each generation that run on 4 processors. Similar to the plot in Figure 3, this plot shows the loads of the processors. The numbers in the hexagons are labels that shows the index of the individual being evaluated. The continuous vertical bars separate the generations. the dotted vertical lines that are drawn at 15, 30, 45, and 60 seconds mark the pings that check the status of the processors. In this experiment, processor 2 crashed while it was evaluating individual 1. PES-Server detected this crash at the first ping (at time 15) and assigned the evaluation of individual 1 to processor 1, and removed processor 2 from its client list.

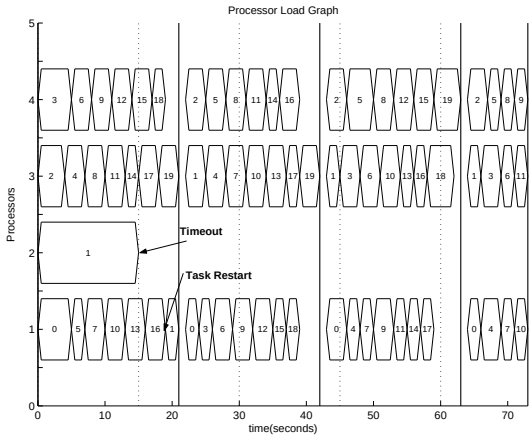


Fig. 5. Load of each processor during a run in which a processor fails. See text for discussion.

4 Conclusion

In this paper, we have reported a new parallel processing platform for running evolutionary methods. Unlike existing parallel processing platforms, PES provides a generic framework within which evolutionary methods can be easily implemented. PES provides an interface which relieves its user from the details of communication and the status of the clients. PES adds better fault tolerance mechanisms to PVM allowing crashed processes to be detected. The user only needs to program a separate client program with a few constraints and should

customize the server program to meet its requirements while leaving the rest to PES.

The analysis showed that although evolutionary methods lend themselves easily to parallelization, non-incremental selection and reproduction (that is the selection and reproduction occurring only after all the individuals are evaluated) is a major source of wasting processing time. The use of incremental selection and reproduction methods would eliminate this problem.

Acknowledgments This work was partially funded by the SWARM-BOTS project, a Future and Emerging Technologies project (IST-FET) of the European Community, under grant IST-2000-31010. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

This work is also funded by Middle East Technical University within the "Sanal Robot Kolonisi" project under grant BAP-2003-07-2-00-01.

References

1. Goldberg D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
2. Hempel, R., *The MPI Standard for Message Passing.* High–Performance Computing and Networking, International Conference and Exhibition, Proceedings, Volume II: Networking and Tools. Ed. Gentzsch, Wolfgang and Harms, Uwe. 247–252,1994.SV, LNCS vol797, 1994.
3. Korpela E., Werthimer D., Anderson D., Cobb J., Lebofsky M., *SETI@home: An Experiment in Public-Resource Computing* Communications of the ACM, Vol. 45 No. 11, pp. 56-61, November, 2002.
4. Kowalik J. (ed), *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press Scientific and Engineering Computation, 1994.
5. Mukherjee S., Mustafi J., Chaudhuri A., *Grid Computing: The Future of Distributed Computing for High Performance Scientific and Business Applications* Lecture Notes in Computer Science, Vol. 2571, pp 339-342, 2002.
6. Nolfi S., Floreano D., *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press/Bradford Books, 2000.
7. Şahin E., Labella T.H., Trianni V., Deneubourg J.-L., Rasse P., Floreano D., Gambardella L.M., Mondada F., Nolfi S., Dorigo M., *SWARM-BOT: Pattern Formation in a Swarm of Self-Assembling Mobile Robots.* In A. El Kamel, K. Mellouli, and P. Borne, editors, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia, October 6-9, 2002. Piscataway, NJ: IEEE Press.
8. Trianni V., Labella Th.H., Gross R., Sahin E., Rasse Ph., Deneubourg J.-L. and Dorigo M. *Evolving Aggregation in a Robotics Swarm.* Accepted to the European Conference on Artificial Life (ECAL'03), 2003.
9. Vinschen C., Faylor C., Delorie D. J., Humblet P., Noer G., *Cygwin User's Guide.*