JavaSpaces - An Affordable Technology For The Simple Implementation Of Reusable Parallel Evolutionary Algorithms

Christian Setzkorn, Ray C. Paton

Department of Computer Science, University of Liverpool, UK

Abstract. Parallel evolutionary algorithms are often used to alleviate the large computational demands of standard evolutionary algorithms. Unfortunately, implementations of parallel evolutionary algorithms can be complicated and often require specific hardware and software settings. This frequently results in very problem-specific parallel evolutionary algorithms with little scope for reuse. This paper investigates the JavaSpaces technology to help to overcome these problems. This technology is free of charge, simplifies the implementation of parallel/distributed applications, and is independent of hardware and software specifications. Several approaches for the implementation of different parallel evolutionary algorithms using JavaSpaces are proposed and successfully tested.

1 Introduction

It is well known that evolutionary algorithms require large amounts of computational resources, especially when the problems to be tackled become complicated and/or when the evaluation of the candidate solutions is computationally expensive [3]. This can hamper their practicality. For these reasons, many researchers have proposed paradigms to execute evolutionary algorithms in parallel. This has given rise to Parallel Evolutionary Algorithms (PEAs). Several proposed PEAs are reviewed in [3, 17]. Unfortunately, implementations of PEAs can be complicated, and often require specific hardware and software specification. This frequently results in very problem-specific applications with little scope for reuse.

This paper investigates the utility of the JavaSpaces technology[1] for the implementation of a synchronous master-slave PEA [10] and a coarse-grained PEA. The proposed frameworks can be used to tackle different problems concurrently with evolutionary algorithms. The JavaSpaces technology has many advantages: it is free of charge, known to simplify the implementation of parallel/distributed applications and is independent of hardware and software settings. It can therefore be used to harvest the computational power of a number of different computers within institutions. The computers can have different hardware and software specifications. To our knowledge JavaSpaces itself has not been used to implement PEAs. Atienza *et al.* [2] has used the Jini technology to implement PEAs. JavaSpace builds upon Jini, but Jini is harder to use.

The scalability of the implemented approaches is evaluated on a particular machine-learning problem; the induction of classifiers from data. Evolutionary algorithms have

[1] JavaSpaces is a trademark of SUN Microsystems, Inc.

been used successfully for this task in the past. In fact, they sometimes produce better classifiers then deterministic approaches because they perform a global search, and can cope better with attribute interactions [5, 8]. Unfortunately, evolutionary approaches are often too slow to be used for large real-world data sets. This is due to the fact that each individual has to be evaluated on a large number of data samples to determine its fitness [1].

The implemented approach induces fuzzy classification rules systems from data using a multi-objective evolutionary approach, building upon the work of Zitzler *et al.* [19] and Ishibuchi *et al.* [12] . This type of classifier was chosen due to its potential high comprehensibility [15]. A general introduction to the problem of fuzzy classification rule induction can be found in [13]. Cordón *et al.* [4] provide an introduction to the construction of fuzzy classification rule systems utilising evolutionary algorithms.

This paper is structured as follows. Section 2 provides some details about the JavaSpaces technology. The implemented approaches are explained in section 3 and their scalability is investigated in section 4. Section 5 contains the conclusions, and suggests avenues for future research.

2 JavaSpaces - A Brief Introduction

SUN Microsystems Inc. proposed the JavaSpaces specification in 1999. It was inspired by the concept of Linda [9]. Linda's core idea is that an object storage can greatly simplify the implementation of parallel and distributed applications. This is because it can be accessed by several processes using a small number of operations. The object storage is referred to as ' tuple space' [9], hence the name JavaSpaces. SUN Microsystems Inc. offers an implementation of the JavaSpaces specification, which utilises technologies such as Jini, RMI [14, 18] and the programming language JAVA [11]. Therefore a JavaSpace can be executed on many different computer platforms due to JAVA's platform independence.

Processes that run within a network of computers, can access the JavaSpace in a concurrent manner. This allows them to communicate and coordinate their actions by writing and reading objects to and from the JavaSpace. Objects are entities that are defined in terms of data and behaviours. They can represent numbers, strings or arrays, and executable programs (evolutionary algorithm individuals). The JavaSpace itself handles the concurrent access, and hence frees the programmer from problems such as race conditions. Objects can be exchanged using the operations *write*, *read*, and *take*. Transactions [7] can be used to make the operations secure because there are many potential sources of failure within a network. For example, objects can get lost on the way between a writing/reading process and a JavaSpace.

Processes can register their interest for a particular type of object with the space. If the specific object is written into the JavaSpace, it notifies the processes via a RemoteEvent [16]. Therefore, many different applications (evolutionary algorithms) could use a JavaSpace at the same time.

In summary, the JavaSpaces technology provides a shared, persistent, and securely accessible opportunity to exchange objects for processes in a computer network. This allows the simple implementation of parallel and distributed applications. For an in-depth introduction to the JavaSpace technology see for example [7, 6].

3 The Implemented Approaches

3.1 The Synchronous Master-Slave Parallel Evolutionary Algorithm

Figure 1 depicts the structure of the JavaSpaces based synchronous master-slave PEA implementation. The synchronous master-slave PEA was first proposed by Grefenstette in 1981 [10]. The JavaSpaces implementation was inspired by the compute-server approach proposed by Freeman *et al.* [7].

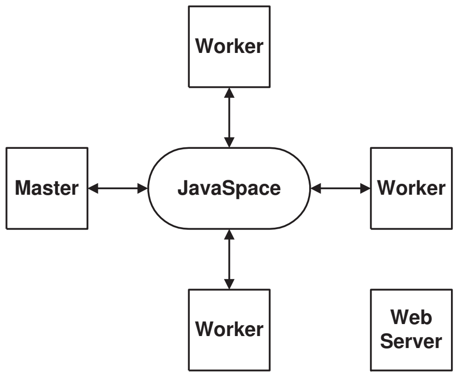

Fig. 1. Structure of the JavaSpaces based synchronous master-slave PEA implementation.

The master executes the evolutionary algorithm. During the fitness evaluation copies of individuals are written into the JavaSpace. When the master creates a copy of an individual, it attaches a unique identifier and a web server address (URL) to it. Workers (slave computers) take individuals from the space (one at the time) and evaluate them. If data is required during the evaluation, it is downloaded from the web server indicated by the URL attached to each individual. This is only done if the worker has not already downloaded the necessary data. A worker does not return the evaluated individual into the space but rather a ‘result object’ consisting of the individual’s objective values and its unique identifier. The master removes these result objects from the space, and uses the unique identifier to update the objective values of a particular individual. The master carries on with other evolutionary processes as soon as it has updated the objective values of all individuals. This approach has many advantages, which are listed and explained now.

- o It is easy to implement and does not change the dynamics of the underlying evolutionary process
- o It implements automatic load balancing. Workers that represent more powerful computers, or those that receive simpler individuals can evaluate more of them as they can access the space concurrently.
- o Workers can be added and removed whilst the evolutionary process is running.
- o Workers can be executed on many different hardware and software platforms because they are implemented in JAVA, which is a platform independent language.
- o Workers do not steal any computational resources as long as no individuals are written into the space. They could therefore run/sleep on all the computers of an institution at all times. A policy could be established that individuals are only written into the space outside normal working times. In this manner all the computer power of an institution could be harvested with virtually no cost, since the computers are often not switched off and JavaSpaces is free of charge.
- o The programming language JAVA also offers what is known as dynamic class downloading [18], which enables workers to evaluate individuals originating from different EAs that are tackling different problems. As long as the object (a particular individual) implements a specific interface (e.g. SpaceIndividualInterface), a slave computer is capable of downloading the necessary class files from a web server in order to evaluate (execute) this individual. This means that such a parallel evolutionary algorithm can be reused (without shut-down) or could even be used by several users, running different evolutionary algorithms at the same time.

However, the presented approach has one disadvantage. It does not fully exploit the power of the utilised computers. This is due to the idle times of the master and the workers. The master has to wait until all the workers have returned the evaluated individuals and the workers are idle until the master initiates the fitness evaluation again. Asynchronous master-slave and coarse-grained PEAs can be used to overcome these problems. The implemented of these PEAs using the JavaSpaces technology is explained in the next sections.

3.2 The Asynchronous Master-Slave Parallel Evolutionary Algorithm

The asynchronous master-slave PEA has the same structure as the synchronous master-slave PEA and was also put forward by Grefenstette in 1981 [10]. In order to tackle the afore-mentioned ‘idle time problem’ of the synchronous master-slave PEA, a non-generational EA is executed on the master. Individuals are written into the space as soon as they require evaluation (e.g. after mutation). Therefore workers are constantly supplied with individuals and the master can proceed with other evolutionary processes as soon as a sufficient number of individuals have been taken from the space [2]. This results in lower idle-times for both workers and the master.

[2] The sufficient number of individuals depends upon the number of individuals that are necessary for the selection process. In this case binary tournament was deployed which only requires two individuals.

3.3 The Coarse-Grained Parallel Evolutionary Algorithm

The coarse-grained PEA has the same structure as the synchronous master slave approach. However each worker runs an evolutionary algorithm instead of just evaluating individuals. The workers have to be started first. When a worker has started, it writes an object containing its IP-address into the space. After all workers have started, the master is started. It removes all objects from the space. The master then writes an object containing all the accumulated IP-addresses back in the space. Each worker reads this object from the space. This makes the workers aware of each other.

The master now initialises a population of individuals and writes them into the space. Each of these individuals contains an IP-address and, if necessary, the web server address of the data. An IP-address is chosen with a uniform probability from the previously accumulated IP-addresses. A worker reads those individuals from the space that contain its IP-address.

Each individual also contains a counter. Each time an individual passes through the evolutionary algorithm this counter is increased. If counter reaches a maximum value (number of generations), an indicator is added to the individual and it is written in the space. This indicator makes it impossible for a worker to take such an individual from the space again. The master can only take an individual from the space if it contains this indicator. This enables the master to accumulate the final population.

If an individual's counter has not yet reached its maximum value, a process within the worker decides whether or not a particular individual is sent (migrated) to another worker or whether it remains with the worker. An individual only migrates with a specific probability (migration probability). To migrate an individual the original IP-address is removed from it. After this, another worker's IP-address is attached to it before it is written into the space. This IP-Address is sampled with a uniform probability from the other workers IP-addresses.

An interesting property of this implementation of a coarse grained PEA approach is that its behaviour only depends on two parameters: the migration probability and the number of workers. The approach is also topologically independent because all workers are virtually interconnected via the space [3]. This makes this implementation much easier when compared to other approaches. The migration probability controls the communication costs. If a low migration probability is choosen, the approach can be expected to be faster than the synchronous master-slave approach.

[3] Different topologies can be implemented by only supplying each EA with particular IP-addresses.

4 Methodology And Results

To investigate the scalability of the synchronous master-slave PEA (parallel implementation), it is first compared against a serial implementation, which performs the fitness evaluation locally. After this, several parallel implementations are compared, which utilise different numbers of workers.

The master and the serial implementation were executed on a PC with the operating system Windows 2000, a single Intel Pentium 4 2.6 GHz processor, and 1GB memory. The workers were executed on PCs with the operating system Windows 2000, a single Intel Celeron 2 GHz processor, and 256MB memory. The JavaSpace and the web server ran on one PC running the operating system Linux (RedHat 9.0). This machine had a single AMD Athlon 1.3GHz processor and 512MB memory. The latest JavaSpaces version that comes with Jini 2.0 was used[4]. All machines were connected via a 100 Mbit switched network.

For the first experiments several synthetic data sets were created, each containing between 1000 to 10000 samples. A population of one hundred individuals was evaluated ten times on each data set. The average evaluation time and standard deviations were computed. Figure 2 depicts the results of the first experiment.

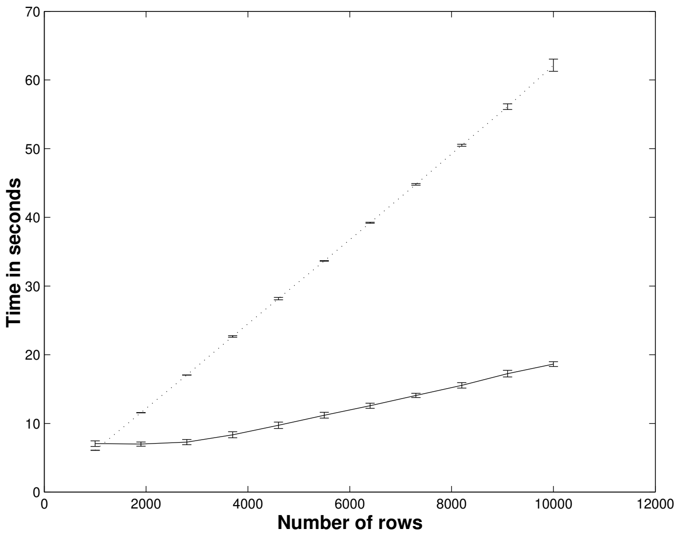

Fig. 2. Comparison between the serial implementation (dotted line) and the JavaSpace implementation (solid line) utilising five slave computers.

It can clearly be seen that the parallel implementation outperforms the serial implementation for data containing more than 1000 samples. Data sets of this size are

[4] It can be downloaded free of charge from http://wwws.sun.com/software/communitysource/jini/download.html

not uncommon nowadays. Figure 3 compares several synchronous master-slave PEAs, which utilise different numbers of workers.

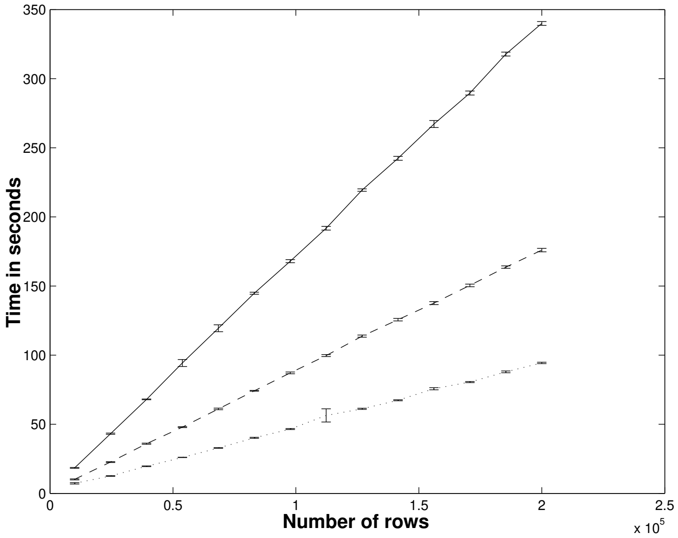

Fig. 3. Comparisons between several parallel implementation utilising different numbers of workers (4 workers - solid line, eight workers - dashed line, sixteen workers - dotted line).

A data set can contain between 10000 to 200000 samples. It can clearly be seen that the utilisation of more workers results in a faster evaluation time for these data sets. Figure 4 depicts the achieved ratios of the decrease in evaluation time. The 'four workers' parallel implementation is compared with the 'eight workers' and 'sixteen workers' parallel implementation, whereas the 'eight workers' parallel implementation is compared with the 'sixteen workers' implementation. Ratios of the decrease in evaluation time of two, four, and two are expected.

Figure 4 shows that the deployment of eight workers results in a decrease in evaluation time by a factor of two. One would expect the same when eight instead of sixteen workers are deployed. Unfortunately, this is not the case. The reason for this might be extra communication costs due to the existence of a larger number of workers. This is also emphasised by the fact that the deployment of sixteen workers instead of four workers only results in a speedup of about three and a half.

Three experiments were performed to compare the three PEAs. For each experiment a PEA was run one hundred times. For each run a data set containing N rows was produced. The value for N was determined randomly through uniform sampling from the interval $[6000 \ldots 200000]$. An EA run terminated after the selection process generated 1000 individuals. This termination criterion enabled a comparison of generational and non-generational approaches. This termination criterion also works for the

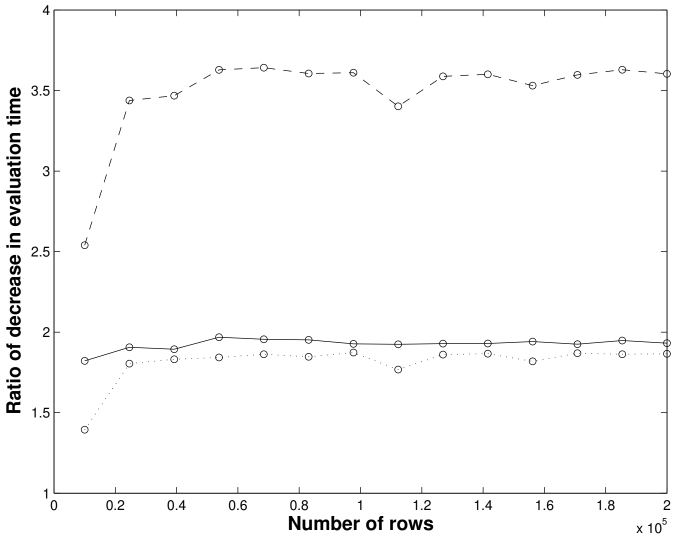

Fig. 4. Ratios of the decrease in evaluation time (8 vs 16 workers - dotted line, 4 vs 8 workers - solid line, 4 vs 16 workers - dashed line).

coarse-grained PEA, although it has several selection processes. As each individual is equipped with a counter, it enables one to measure how many individuals all selection processes have produced globally.

The individuals had a fixed size to keep the communication costs constant. A migration probability of 0.1 was used for the coarse-grained PEA. This means that (on average) every tenth individual is migrated to another EA.

The three experiments utilised the following hardware and software settings. All workers (EAs in the case of the coarse-grained PEA) were executed on PCs with the operating system Linux, a single AMD Athlon 1.3 GHz processor and 512 MB memory. The master implementations were executed on a PC with the operation system Windows 2000, and a single AMD Athlon 850 MHz processor and 256 MB memory. The JavaSpace was executed on a machine with the same specification as that of the workers. The JavaSpace implementation that comes with Jini 1.4 was used. The utilised web server was a HP RP 2400 machine, with one PA 8500 RISC 440 MHz processor, 630 MB memory, and Apache 1.3.6 and the HP-UX 11.00 operating system. All computers were connected via a 100 Mbit switched network. It is important to note that other users have used the machines, except the master, while the experiments were performed. Therefore the results show that the approach can be used in a multi-user environment.

As stated above, an experiment consisted of one hundred pairs of independent values (number of cases) and dependent values (execution time in seconds). In order to compare experiments visually the statistical package SAS[5] was used. For each experiment a regression line was fitted through the data. A ninety-nine percent confidence interval was computed for the mean predicted values for each experiment. Figure 5 shows the results when twenty workers are deployed.

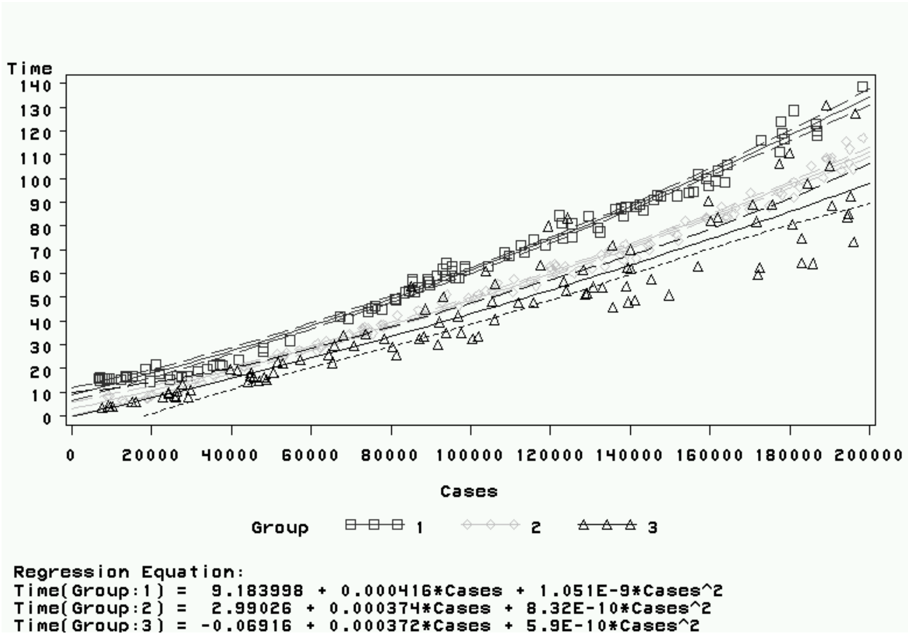

Fig. 5. Comparsion between different PEAs (Group 1: 20-worker synchronous master-slave PEA, Group 2: 20-worker asynchronous master-slave PEA, Group 3: 20-worker coarse-grained PEA).

It can clearly be seen that the coarse-grained PEA outperforms the synchronous master-slave PEA for more than three thousand rows. The coarse-grained PEA outperforms the asynchronous master-slave PEA for more than one hundred thousand rows. However, the spread of data points also indicates that the execution time of the coarse-grained PEA can vary.

5 Conclusions And Future Work

This paper has shown that the JavaSpaces technology can be used to improve the performance of evolutionary algorithms in a simple and affordable manner. Several approaches for the implementation of different parallel evolutionary algorithms were proposed and successfully tested using this technology. It was also shown that the PEA

[5] SAS is a trademark of SAS Institute Inc.

implementations perform well in multi-user heterogeneous computer networks. As expected, the coarse-grained PEA outperformed the synchronous and asynchronous master slave PEAs. This is most probably due to its low communication costs. Although the coarse-grained PEA is capable of evolving the same amount of individuals in a shorter period of time, in comparison to the other PEAs, it remains an open question whether or not these individuals perform as well as the individuals that were evolved by the other PEAs. Further studies are required to investigate this issue.

References

1. D.L.A. Araujo, H.S. Lopes, and A.A. Freitas. Rule Discovery with a Parallel Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program 2000*, pages 89–92, Las Vegas, USA, July 2000.
2. J. Atienza, M. Garcia, J. González, and J.J. Merelo. Jinetic: a distributed, fine-grained, asynchronous evolutionary algorithm using jini, 2000.
3. E. Cantá-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.
4. O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic Fuzzy Systems*. World Scientific, 2001.
5. V. Dhar, D. Chou, and F.J. Provost. Discovering interesting patterns for investment decision making with glower - a genetic learner overlaid with entropy reduction. *Data Mining and Knowledge Discovery*, 4(4):251–280, 2000.
6. R. Flenner. *Jini and JavaSpaces Application Development*. Sams, 2001.
7. E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces(TM) Principles, Patterns and Practice*. Addison-Wesley Longman Ltd., 1999.
8. A.A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computation*, pages 819–845. Springer-Verlag, 2002.
9. D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
10. J.J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, 1981. Vanderbilt University, Computer Science Department, Nashville.
11. J. Hunt. *Java and Object Orientation*. Springer, 1998.
12. H. Ishibuchi, T. Nakashima, and T. Murata. Three-Objective Genetics-Based Machine Learning for Linguistic Rule Extraction. *Information Sciences*, 136:109–133, 2001.
13. L.I. Kuncheva. *Fuzzy Classifier Design*. Heidelberg: Physica-Verlag, 2000.
14. S. Li. *Professional Jini*. Wrox, 2000.
15. D. Michie. Learning concepts from data. *Expert System with Applications*, 15:193–204, 1998.
16. Sun Microsystems. Java distributed event specification, 1998.
17. M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In *Proceedings of the third international conference on knowledge-based intelligent information engineering systems (KES'99)*, pages 88–92. IEEE Press, 1999.
18. E. Pitt and K. McNiff. *Java.RMI*. Addison-Wesley Longman Ltd., 2001.
19. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, 2001.