Analysys Paper: Cloudde, A Heterogeneous Differential Evolution Algorithm and Its Distributed Cloud Version

Zhi-Hui Zhan, Member, IEEE
Xiao-Fang Liu, Student Member, IEEE
Huaxiang Zhang, Zhengtao Yu, Jian Weng, Tianlong Gu
Yun Li, Member, IEEE
Jun Zhang, Senior Member, IEEE

IEEE Transactions on Parallel and Distributed Systems Vol.28
NO. 3, March 2017

1 Introduction

1.1 Motivation

The first time I approached a problem with an evolutionary algorithm was trying to crack *Enigma*. Given a sentence with shifted alphabet the solver was programmed to find the best permutation that transforms the encoded sentence into Romanian. The approach was to consider the individuals as random permutations composed by their disjoint cycle decomposition.

The second time was in the context of producing rectangular surface cuts while maximizing a profit.

Lastly, I developed a CUDA solver which processed the multidimensional individuals in parallel. As a benchmark I used the Rosenbrock function and the speedup was noticeable.

From the first two problems I learned that the evolutionary methods often require considerable running time to reach a good optimum. I tackled this phenomenon from varying input and from a sentence processed in 3 minutes, others took hours to solve.

From the latter one, I've noticed that the performance of the evolutionary methods highly depend on the initial configuration of the parameters. The number of individuals, the crossover rate, the mutation probability are very sensitive to obtain an optimal solution.

In this paper both of these problems are addressed. Instead of parallel individuals, the proposed Cloudde algorithm relies on independent heterogeneous populations. Comparing with my previous attempts to use CPU threads or

GPU thread-blocks to process individuals in parallel, this algorithm uses parallel processing using virtual machines which process entire populations.

1.2 About the Proposed Algorithm

The paper presents a double layered master-slave architecture. The master controls the heterogeneous populations. Based on the size of the population, the resources are allocated respectively.

The key point about using this approach is that the populations are sorted based on their average fitness. Each individual has a chance of migrating in any of the superior populations. By doing this, the lower performing populations based by the initial parameter configuration don't use more computing resources than the better performing ones.

1.3 Assumptions

We will use two common differential evolution mutations. We define N as the number individuals, randomly created

$$X_i = [x_{i1}, x_{i2}, ..., x_{iD}]$$

where i is the individual index, D is the problem's dimension.

Each individual i first performs a mutation to generate a new individual vector

$$V_i = X_{best} + F(X_{r1} - X_{r2})$$

based on the best solution or

$$V_i = X_{r1} + F(X_{r1} - X_{r2})$$

based on a randomly selected individual.

Following mutation, DE performs a crossover operation on vectors X_i, V_i to form a candidate solution

$$U_i = [u_{i1}, u_{i2}, ..., U_{iD}]$$

defined as

$$U_{id} = \begin{cases} v_{id} & if\ rand(0,1) \leq CR\ or\ d = r_i \\ x_{id} & otherwise \end{cases}$$

The parameters in this paper are a combination of DE/best, DE/rand mutation scheme and a $CR = 0.1$ or $CR = 0.9$ crossover rate.

Lastly, we define the selection function to decide the candidates which pass on the next generation.

$$X_i = \begin{cases} U_i & if\ fit(U_i) \leq fit(X_i) \\ x_i & otherwise \end{cases}$$

2 Cloudde Algorithm

2.1 Architecture

If we consider M populations, implemented on parallel hardware, the layer of distribution is controlled by a master node. Each population can have different performance, and it would be wasteful for all individuals in an inferior population to evaluate poor solutions.

The architecture remains tree based, dual layered, however, the second layer of individuals is constantly regrouped by migrating individuals in one direction from the weaker performing populations to the better ones.

The algorithm can run sequentially, but our most interest is in the distributed version. We will consider the second layer as a resource pool of virtual machines, first layer being an abstraction in the point of view of the master node.

According to the authors, the algorithm is cloud-ready, virtual machines are at an ease of access nowadays, and the communication flow in the algorithm is intuitive.

2.2 Adaptive Migration Strategy

Cloudde is designed to allow the distributed populations to use different mutation schemes and parameters. The strategy implies calculating the mean fitness of each population and ranking the M populations from the best to the worst. Each individual has a probability to migrate to a better population.

It is defined a migration rate p_m used to control the probability. If the rate is too small, the migration rarely occurs and the populations can't share their progress, fact that is useful at the beginning of the iterative process. When p_m is large, the migration can spread the local convergence, fact that is useful only near the end of the iterative process.

Hence, a good choice is to change the rate during the process. The paper presents briefly a non linearly increasing function to do so:

$$p_m = 0.01 + 0.99\frac{exp(\frac{10g}{G}) - 1}{exp(10) - 1} \in [0.01, 1.00]$$

where g is the current generation and G is the maximum generation that can be achieved.

2.3 Cloud Resource Balancing

The fitness evaluation is the most demanding task in an evolutionary algorithm. Considering we have T VMs in the resources pool and M populations, initially with the same size, the T VMs are evenly allocated into M groups. The migration during the iterative process and the population sizes may vary.

In order to balance at each generation, the authors propose an euclidean metric to define an average unbalance degree u by the standard deviation as:

$$u = \frac{1}{M-1}\sqrt{\sum_{i=1}^{M}(l_i - l)^2}$$

where l is the average load of VMs and l_i is the load with respect to the population size.

Since the regrouping is sensitive to potential latency, the regrouping can be scheduled only in the case when the unbalance degree exceeds a certain cost. In the benchmark section, the authors provide insights about choosing the u_{max} value. Usually this process depends on the minimized problem. In the practical example provided at the end of the paper, the peak value is easy to discover.

3 Benchmarks

The authors provide three main comparisons, between Cloudde and the standard DE configurations, between Cloudde and conventional DE algorithms, and between Cloudde and similar state-of-the-art distributed DE algorithms.

There are 13 famous functions taken into consideration for DE performance, with many local minimums (Quadric, Step, Noise, Rosenbrock, Schwel, Ackley, Griewank and others) within a considerable range surpassing 32-bit numeric values.

In the first comparison, each configuration is given a rank between 1-5, and on the all test functions, Cloudde is achieving a rank of 1.62 on average, surpassing the second place DE / random scheme / 0.1 CR with 2.23.

In the second comparison, Cloudde is competing with jDE, SaDE, JADE and CoDE. This time Cloudde ranks also on the first position with 1.92 average rank, surpassing CoDE which has 2.31 average.

In the last comparison, the mentioned competitors are PDE, IBDDE, APSO and CMA-ES, all of them claimed to be state-of-the-art. Cloudde is on top again with 1.38, on the second place being APSO with 2.08.

The paper also includes different configuration testing, for u_{max}. The unbalance degree seems to perform much better at the value 7, in the context of a practical problem. In this case, the optimal value is easy to spot.

4 Conclusions

The Cloudde algorithm gives promising results in all environments, sequential or distributed. The most promising results are in the most computationally expensive problems, and where speedup is most noticeable.

VMs are particularly accessible in the cloud computing environment, Cloudde is suitable for large size populations. Given the heterogeneous nature of the framework, any other enhanced DE variants can produce even better performance in terms of speed, and robustness.

Personally, I find very useful the concept of adapting resources during the whole evolutionary process, in both the context of sequential or parallel implementations. In my previous experiences the convergence rate dropped dramatically and an adaptive allocation of resources could be decisive.

Lastly, I consider this paper as a great example of heterogeneous framework. I've been used to focus on thinking at the resources of the virtual machines as independent workers, but in this context, the strategy used for the individuals to be orchestrated is contributing greatly to the optimization process.