An Analysis of Massively Distributed Evolutionary Algorithms

Travis Desell, David P. Anderson, Malik Magdon-Ismail, Heidi Newberg,
Boleslaw K. Szymanski and Carlos A. Varela

***Abstract*— Computational science is placing new demands on optimization algorithms as the size of data sets and the computational complexity of scientific models continue to increase. As these complex models have many local minima, evolutionary algorithms (EAs) are very useful for quickly finding optimal solutions in these challenging search spaces. In addition to the complex search spaces involved, calculating the objective function can be extremely demanding computationally. Because of this, distributed computation is a necessity. In order to address these computational demands, top-end distributed computing systems are surpassing hundreds of thousands of computing hosts; and as in the case of Internet based volunteer computing systems, they can also be highly heterogeneous and faulty. This work examines asynchronous strategies for distributed EAs using simulated computing environments. Results show that asynchronous EAs can scale to hundreds of thousands of computing hosts while being highly resilient to heterogeneous and faulty computing environments, something not possible for traditional distributed EAs which require synchronization. The simulation not only provides insight as to how asynchronous EAs perform on distributed computing environments with different latencies and heterogeneity, simulation also serves as a sanity check because live distributed systems require problems with high computation to communication ratios and traditional benchmark problems cannot be used for meaningful analysis due to their short computation time.**

I. Introduction

Computational science is placing new demands on optimization algorithms as the size of data sets and the computational complexity of scientific models continue to increase. As these complex models have many local minima, evolutionary algorithms (EAs) are very useful for quickly finding optimal solutions in these challenging search spaces. In addition to the complex search spaces involved, calculating the objective function can be extremely demanding computationally. In order to address these computational demands, top-end distributed computing systems are surpassing hundreds of thousands of computing hosts.

As the size of computing systems continues to increase, having scalable EAs becomes very important. Additionally, larger computing systems increase the potential for faulty computing hosts. In grid computing and especially Internet computing, increasing the number of hosts means that hosts will be spread out farther geographically, increasing the heterogeneity of communication, and that they will consist of different architectures and operating systems, increasing the heterogeneity of calculation times. Because of this, the distributed evolutionary algorithms used must also be resilient to faults and heterogeneous communication and computation times.

This work discusses different strategies for computing EAs on distributed environments. In particular, sequential strategies which require synchronization between successive populations are compared to asynchronous strategies that do not have explicit dependencies. A simulation framework is used to examine these different distributed EA strategies on various simulated computing environments. A simple homogeneous environment which represents computing clusters or supercomputers is used to compare the scalability of sequential (or parallel) EAs to asynchronous EAs. A computing environment with various communication latencies is used to examine the effect of heterogeneity on asynchronous EAs. Lastly, a complex Internet-like distributed computing environment is modeled using data from the MilkyWay@Home volunteer computing project[1].

Results show that while parallel EAs have difficulty scaling to massively distributed computing environments, which can have thousands or more computing hosts, asynchronous EAs scale well to a hundred thousand computing hosts. Additionally, asynchronous EAs are shown to require *less* evaluations to reach a solution as the heterogeneity of the computing environment is increased. Furthermore, results show that for certain benchmark problems on the MilkyWay@Home-like computing environment, some asynchronous EAs lose the ability to find the solution, while others gain the ability to find the solution when they could not on other computing environments. This is particularly interesting as the heuristics used to evolve the population did not change, only the order in which the results were received. These results show that not only are asynchronous EAs an effective solution to the problems involved in massively distributed computing, but that there is potential for improving the robustness or convergence rates of EAs by changing the order in which they process individuals.

II. Evolutionary Algorithms for Continuous Search Spaces

Current popular approaches to global optimization for continuous search spaces involve differential evolution, particle swarm optimization and genetic search. In general, individuals are sets of parameters to an objective function which is trying to be optimized. Applying the objective function to an individual provides the fitness of that individual, and the

Travis Desell, Malik Magdon-Ismail, Boleslaw K. Szymanski and Carlos A. Varela are with the Department of Computer Sciences, Heidi Newberg is with the Department of Physics, Applied Physics and Astronomy, Rensselaer Polytechnic Institute, Troy, New York 12180, USA.

David P. Anderson is with the U.C. Berkeley Space Sciences Laboratory, University of California, Berkeley, Berkeley, California 94720, USA.

[1]http://milkyway.cs.rpi.edu

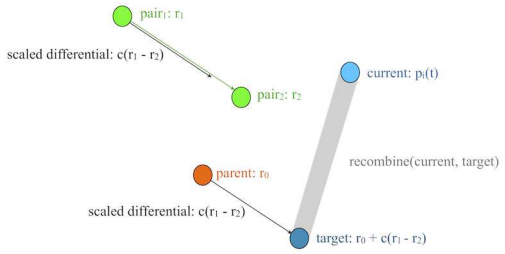

Fig. 1. A two dimensional example of how an individual moves in differential evolution.

evolutionary algorithms evolve individuals through different heuristics to try and find the best possible fitness, which optimizes the objective function.

A. Genetic Search

Genetic search is the most classical example of an evolutionary algorithm. It generates successive populations of individuals by applying *selection*, *mutation* and *recombination* to the individuals in the previous population. *Selection* selects the best members of the previous population. *Mutation* takes an individual and modifies a parameter at random. Typically the parameter is mutated to any new possible value; and a common improvement is do adaptively reduce the distance from the original parameter the mutation can take. *Recombination* typically takes two individuals as parents and generates a child individual by combining the parents' parameters. However many variants use more than two parents [1], [2], [3], [4]. Where the i^{th} parameter of the child, c_i, is generated from the i^{th} parameter of the parents, p_i^1 ... p_i^n, D is the number of parameters in the objective function, and r is a random number generator, some common recombination operators are:

- binomial recombination, $bin(p^1, p^2)$:

$$c_i = \begin{cases} p_i^1 \text{ if } r[0,1) < \sigma \text{ or } i = r(0,D) \\ p_i^2 \text{ otherwise} \end{cases} \quad (1)$$

- exponential recombination, $exp(p^1, p^2)$:

$$c_i = \begin{cases} p_i^1 \text{ from } r[0,1) < \sigma \text{ or } i = r(0,D) \\ p_i^2 \text{ otherwise} \end{cases} \quad (2)$$

- simplex recombination, $simplex(p^1, ..., p^n)$:

$$c_i = p_i^{worst} + r(0,1) * (average(p^1, ..., p^n)_i - p_i^{worst}) \quad (3)$$

B. Differential Evolution

Differential evolution is an evolutionary algorithm used for continuous search spaces developed by Storn and Price over 1994–1995 [5]. Unlike other evolutionary algorithms, it does not use a binary encoding strategy or a probability density function to adapt its parameters, instead it performs mutations based on the distribution of its population [6]. For a wide range of benchmark functions, it has been shown to outperform or be competitive with other evolutionary algorithms and particle swarm optimization [7].

Differential evolution evolves individuals by selecting pairs of other individuals, calculating their differential, scaling it and then applying it to another parent individual. Some kind of recombination (*e.g.*, binary or exponential) is then performed between the current individual and the parent modified by the differentials (see Figure 1). If the fitness of the generated individual is better than the current individual, the current individual is replaced with the new one. Differential evolution is often described with the following naming convention, "de/*parent*/*pairs*/*recombination*", where *parent* describes how the parent is selected (*e.g.*, best or random), *pairs* is the number of pairs used to calculate the differentials, and *recombination* is the type of recombination applied.

In general, a new potential individual $n_i(l+1)$ for a new population $l+1$ is generated from the i^{th} individual $x_i(l)$ from the previous population l, and selected if its fitness, $f(x)$, is greater than the previous individual:

$$x_i(l+1) = \begin{cases} n_i(l+1) \text{ if } f(n_i(l+1)) > f(x_i(l)) \\ x_i(l) \text{ otherwise} \end{cases} \quad (4)$$

The j^{th} parameter is calculated given p pairs of random individuals from the population l, where $r(l)^0 \neq ... \neq r(l)^{2p}$. θ, ϕ and σ are the user defined *parent scaling factor*, *recombination scaling factor* and *crossover rate*, respectively. $b(l)$ is the best individual in the population l. Two popular variants are:

- *de/best/p/bin*:

$$n_i(l+1) = bin(x_i(l), \theta b(l)_j^0 + \phi \sum_{k=1}^{p} [r(l)_j^{1k} - r(l)_j^{2k}]) \quad (5)$$

- *de/rand/p/bin*:

$$n_i(l+1) = bin(x_i(l), \theta r(l)_j^0 + \phi \sum_{k=1}^{p} [r(l)_j^{1k} - r(l)_j^{2k}]) \quad (6)$$

Which are used in this work. For more detail, Mezura-Montes *et al.* have studied many different variants of differential evolution on a broad range of test functions [8].

C. Particle Swarm Optimization

Particle swarm optimization was initially introduced by Kennedy and Eberhart [9], [10] and is a population based global optimization method based on biological swarm intelligence, such as bird flocking, fish schooling, etc. This approach consists of a population of particles, which "fly" through the search space based on their previous velocity, their individual best found position (cognitive intelligence) and the global best found position (social intelligence). Two user defined constants, c_1 and c_2, allow modification of the balance between local (cognitive) and global (social) search.

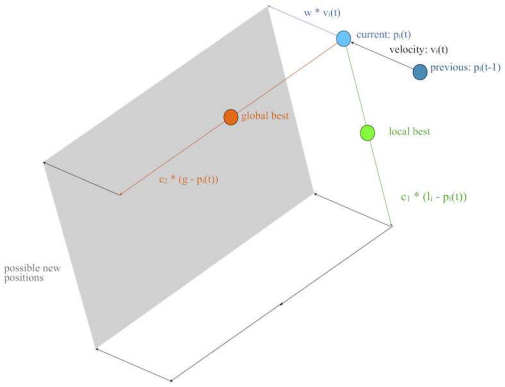

Fig. 2. A two dimensional example of how a particle moves in particle swarm optimization.

Later, an inertia weight ω was added to the method by Shi and Eberhart to balance the local and global search capability of PSO [11] and is used in this work and by most modern PSO implementations. The population of particles is updated iteratively as follows, where x is the position of the particle at iteration t, v is it's velocity, p is the individual best for that particle, and g is the global best position (Figure 2 shows how a single particle can move in two dimensions):

$$\begin{aligned} v_i(t+1) &= \omega * v_i(t) \\ &\quad +c_1 * rand() * (p_i - x_i(t)) \\ &\quad +c_2 * rand() * (g_i - x_i(t)) \\ x_i(t+1) &= x_i(t) + v_i(t+1) \end{aligned} \tag{7}$$

III. Distributed Evolutionary Algorithms

There have been many different approaches to making EAs work on different distributed computing systems. In general these approaches are either sequential, with distinct synchronization points, or asynchronous, without distinct synchronization points.

A. Sequential Approaches

Sequential approaches to distributed EAs typically use a single-population strategy, where new populations are generated repeatedly by evaluating each individual in parallel; repeating this process until the population has reached convergence criteria [12], [13]. It is possible to increase scalability past the population size by additionally evaluating the objective function in parallel. This type of approach is best suited to highly reliable and homogeneous computing nodes, as found in clusters and supercomputers.

B. Hybrid Approaches

Hybrid approaches involve *islands* of populations [14]. The different populations are evaluated sequentially and then asynchronously migrate selected individuals to neighboring islands when certain criteria are met [15], [16], [17], [18]. Tasoulis *et al.* have shown that having moderate values for migration result in the best convergence for differential evolution across a variety of benchmarks [19]. It has been shown that super-linear speedup can be attained using this method, as smaller populations can converge to minima quicker than larger populations [20], [21]. However, having populations of different sizes and/or populations running on clusters of different speeds can have varying negative effects on the performance of the search. As each island can be parallelized in the same manner as a single population EA, this approach is well suited to grid computing systems, where islands can be assigned to individual clusters within the grid. Island EAs have also been used in peer-to-peer computing systems [22].

C. Asynchronous Approaches

Asynchronous approaches to distributed EAs typically use a single-population and a master-worker model. One approach is to generate the population and have workers request individuals to evaluate, using a work-stealing model [23], [24]. Many approaches to particle swarm use a method where individual particles are calculated in parallel and newly found global best positions are then broadcast asynchronously [25], [26]. However, as with single-population sequential approaches these strategies are limited by the population size used by the EA.

In order to overcome the scalability limitations presented by these different strategies, the authors have developed asynchronous optimization, a strategy which is not limited in scalability [27], [4]. Similar to steady state genetic search, which generates one individual at a time and uses it to monotonically update a population; asynchronous optimization generates new individuals based on a populations current state in response to requests for work and later inserts the results to the population when and if they are reported. As the heuristics for generating new individuals in the discussed EAs are randomized, this allows asynchronous evolutionary algorithms to generate as many unique individual as are required to satisfy all potential workers. There are also no dependencies between generated individuals, so if a worker fails and does not report the fitness of its individual, the search does not need to wait for that individual to be recalculated.

IV. Simulation Framework

In order to test the effect of different distributed computing platforms and their size on the convergence rates of different EAs, a simulation environment was developed. This environment can simulate heterogeneity and failures, while quickly evaluating different search methods using various benchmark functions. The simulated computing environment requests parameter sets to be evaluated then returns those results after a simulated amount of time (or not at all). In addition to testing the search methods on functions known to be difficult to find a global optimum for, this also provides a method to evaluate the effect of asynchrony in a controlled environment, *e.g.*, the number of updates to the population that occur before the result of a parameter set is reported can be generated through different probability distributions and

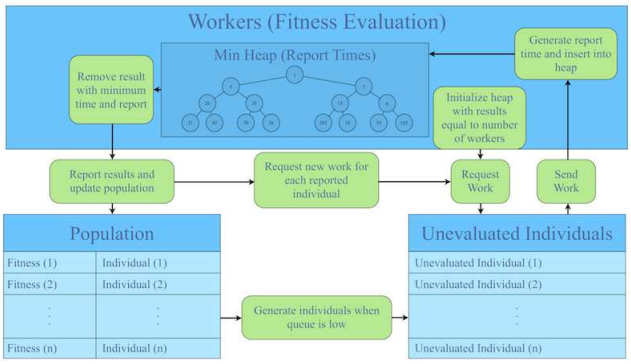

Fig. 3. The simulated evaluation framework. A simulated environment is used instead of different distributed computing environments. Computed results are stored in a heap and inserted into the population in order of their simulated report time.

the minimum and maximum update times for parameter sets can be fixed.

The simulation framework consists of two basic parts. The first allows users to specify templates which control the amount of asynchrony and fault rates in the system. Users can specify the amount of time it takes for results to be calculated and the number of faults that occur. The amount of time it takes for a result to be calculated is probabilistic, specifying the percentage chance for a result to take the time between a minimum and maximum given time and a probabilistic distribution function. Currently, uniform and gamma distributions are implemented. The gamma distribution was chosen because it typically is used to model waiting times, which is ideal for this type of simulation. Multiple distributions can be used, which is important in modeling grid and Internet-like environments which may have different result time distributions if different types of hardware are used.

The other part is the simulated evaluation environment (see Figure 3). The simulations run with a specified number of workers, allowing the size of the simulated computing environment to be modified. A heap is used to handle results that are currently being calculated by the simulated environment. This allows insert and removal of results to be done in $O(log(n))$ time. The heap is initially populated with a number of results equal to the number of workers and each is given a report time (trip time plus calculation time) that is calculated using the specified distributions in the simulation template. If a result is determined to be faulty another trip and calculation time are generated, and this is repeated until the result is determined to be non-faulty and the result time is the sum of all generated computation and trip times. Following this, the minimum value is removed from the heap and inserted into the search and the current simulation time is set to the report time of that result. A new result is then generated with a new report time, which is the current time plus the result calculation time, and inserted into the heap. This essentially simulates workers requesting a new result, calculating the result and sending it to the server. Faults require the initial worker to error out and then have the same work sent to a different worker.

V. Results

The simulation environment was used to compare the asynchronous EA strategy to the basic single population evolutionary algorithm strategy. While the island strategy of distribution is a popular one, the authors feel that in some sense it is independent of actually making an evolutionary algorithm parallel. For example, many hybrid strategies use multiple populations or *neighborhood* strategies which divide populations into distinct subpopulations, without any form of distribution [28], [29], [2], [30], [31]. Asynchronous EAs and parallel EAs are more primitive to the distribution problem, as each island population can be evolved using either. Additionally, island style EAs can be performed by asynchronous EAs, by subdividing the population server side.

A. Optimization Parameters

For genetic search, a mutation rate of 0.3 was used, and the simplex recombination method was used, with $l_1 = -1.5$, $l_2 = 1.5$ and 2 parents (previous work has shown this to outperform other recombination operators [27]). Particle swarm used an inertia weight, $\omega = 0.5$, and $c_1 = c_2 = 2.0$. Differential evolution used binomial recombination with either best or random parent selection, one pair, a recombination scale of 0.5 and a recombination rate of 0.5.

B. Test Functions

The Ackley, Griewank, Rastrigin, Rosenbrock and Sphere problems were used as benchmark optimization problems, which are described in detail in various other work [27], [32], [33], [34]. The Ackley and Sphere problems used 10 search parameters, and the Griewank, Rastigin and Rosenbrock problems used 5 search parameters. A solution was assumed to be reached when the fitness of best member in the population was less than 10^{-10}, and the results presented are the average of 20 different searches.

C. Simulated Environments

Three different simulated computing environment types (homogeneous, heterogeneous and internet-like) were used to examine the effects of heterogeneity and the scalability of asynchronous EAs and parallel EAs.

1) Homogeneous Environments: The simulated homogeneous environment models computing on a cluster, supercomputer or even a graphical processing unit (GPU), where computation time and latency on the distributed processors (or threads) is uniform. This type of computing environment was simply modeled by using a fixed result report time of 1 was used. The number of parallel computing hosts used was increased from 100 to $100,000$. As parallel EAs require a population size equal to the amount of computing hosts, the population size of the parallel EAs was increased accordingly. Asynchronous EAs have no such limitation, so a fixed population size of 100 was used for all amounts of computing hosts, however for the Rastrigin and Rosenbrock

	Simulated Computing Hosts			
	100	1000	10000	100000
Ackley				
PGS	169.76	345.37	380.47	N/A
AGS	169.76	45.23	18.18	10.57
Speedup	0	7.66	21.11	N/A
PPSO	128.13	76.04	36.18	27.47
APSO	128.13	19.63	9.49	6.72
Speedup	0	3.87	3.81	4.08
PDE/best	N/A	61.41	33.34	21.30
ADE/best	N/A	61.41	13.17	7.53
Speedup	N/A	0	2.53	2.83
PDE/rand	85.74	123.44	164.72	186.69
ADE/rand	85.74	25.38	10.81	6.51
Speedup	0	4.86	15.23	28.67
Griewank				
PGS	15498.63	11720.21	8752.28	N/A
AGS	15498.63	1545.68	214.19	52.96
Speedup	0	7.58	40.89	N/A
PDE/rand	1109.67	1379.61	1414.353	1415.571
ADE/rand	1109.67	142.16	39.636	18.557
Speedup	0	9.71	35.67	76.28
Rastrigin				
PGS	9458.45	666.64	585.84	N/A
AGS	9458.45	132.86	40.60	21.42
Speedup	0	5.02	14.4	N/A
PPSO	285.76	180.52	88.85	68.59
APSO	285.76	45.00	18.51	11.17
Speedup	0	4.01	4.80	6.14
PDE/best	N/A	70.44	65.95	56.93
ADE/best	N/A	70.44	26.64	14.47
Speedup	N/A	0	2.47	3.93
PDE/rand	603.37	876.09	915.63	908.44
ADE/rand	603.37	107.33	32.30	16.25
Speedup	0	8.18	28.33	55.87
Rosenbrock				
PPSO	21182.11	7940.44	910.84	121.52
APSO	21182.14	2464.42	691.08	257.89
Speedup	0	3.22	1.32	0.47
PDE/best	N/A	58.48	53.01	50.25
ADE/best	N/A	58.48	25.02	16.53
Speedup	N/A	0	2.12	3.13
PDE/rand	243.19	212.68	202.89	199.07
ADE/rand	243.19	56.94	25.94	15.78
Speedup	0	3.73	8.08	12.59
Sphere				
PGS	20911.79	15329.22	12811.55	12019.73
AGS	20911.79	2074.40	263.94	75.07
Speedup	0	7.39	48.71	160.25
PPSO	300.68	221.32	174.10	142.78
APSO	300.68	65.38	30.97	20.99
Speedup	0	3.38	5.61	7.1
PDE/best	77.97	69.06	63.79	59.16
ADE/best	77.97	37.66	24.72	19.93
Speedup	0	1.86	2.6	3.10
PDE/rand	372.29	434.41	436.33	434.02
ADE/rand	372.29	98.58	46.60	28.70
Speedup	0	4.40	9.36	15.12

Fig. 4. Average number of iterations to solution for asynchronous EA and parallel EA strategies for different benchmark problems. Asynchronous EAs used a fixed population size of 100, except for ADE/best on the Ackley, Rastrigin and Rosenbrock problems, which required a population size of 1000. The parallel EAs used a population size equal to the number of processors.

problems a population size of 100 was insufficient to solve the problem so a population size of $1,000$ was used.

Figure 4 shows the number of iterations taken to reach a solution, where an iteration consists of sending a work to each computing host and the receiving those results. It is important to note that increasing the population size of a parallel EA to account for the amount of computing hosts can actually increase the time to a solution, as in the case of parallel differential evolution with random parent selection (PDE/random) for the Rastrigin problem. And in many other cases, increasing the population size to match the amount of computing hosts for the parallel EAs has no noticeable improvement to the time to solution. Of all the parallel EAs, parallel particle swarm optimization (PPSO) seemed to be the most resilient to increasing the population size to match the number of computing hosts, often showing improved time to solution. On the other hand, using asynchronous EAs always resulted in improved time to convergence as the number of computing hosts was increased.

2) Heterogeneous Environments: Parallel EAs are poorly suited to heterogeneous environments, as they have to wait for the slowest computing host to calculate the fitness of its individual before they can progress to generating and evaluating the next population. Asynchronous EAs do not have any dependencies however, so they are better suited to these types of computing environments. A simple heterogeneous environment was used to evaluate the effect of heterogeneity on asynchronous EAs. A fixed compute time of 1 was used, and the range of possible communication times was increased from (0,1] to (0,1000]. Communication times were generated uniformly across the possible range. All the tests used a fixed number of $10,000$ simulated computing hosts and population sizes of 100 unless otherwise noted.

Figure 5 shows the number of evaluations taken to reach a solution for the different benchmark problems on simulated heterogeneous environments. While parallel EAs would require the same number of evaluations to reach the solution as heterogeneity does not effect the order in which results are used to generate subsequent populations; asynchronous EAs required *less* evaluations to reach a solution as the heterogeneity increased. For some searches this effect tapered off after reaching a certain result report time range, however in general further heterogeneity did not begin to increase the number of evaluations taken to reach a solution.

3) Internet-like Environments: In order to test the scalability of asynchronous EAs on more realistic and massively distributed heterogeneous environments, the time to report results were examined using data from the MilkyWay@Home project. Figure 6 shows the frequency of report times for a sample of $10,000$ results reported to the MilkyWay@Home server. The different results were separated into both GPU and CPU results, as GPUs can perform the calculation orders of magnitude faster than CPUs. The result report time is the time it took from the moment a parameter set was generated to the time the parameter set was reported with a result and inserted into the search population. The different points

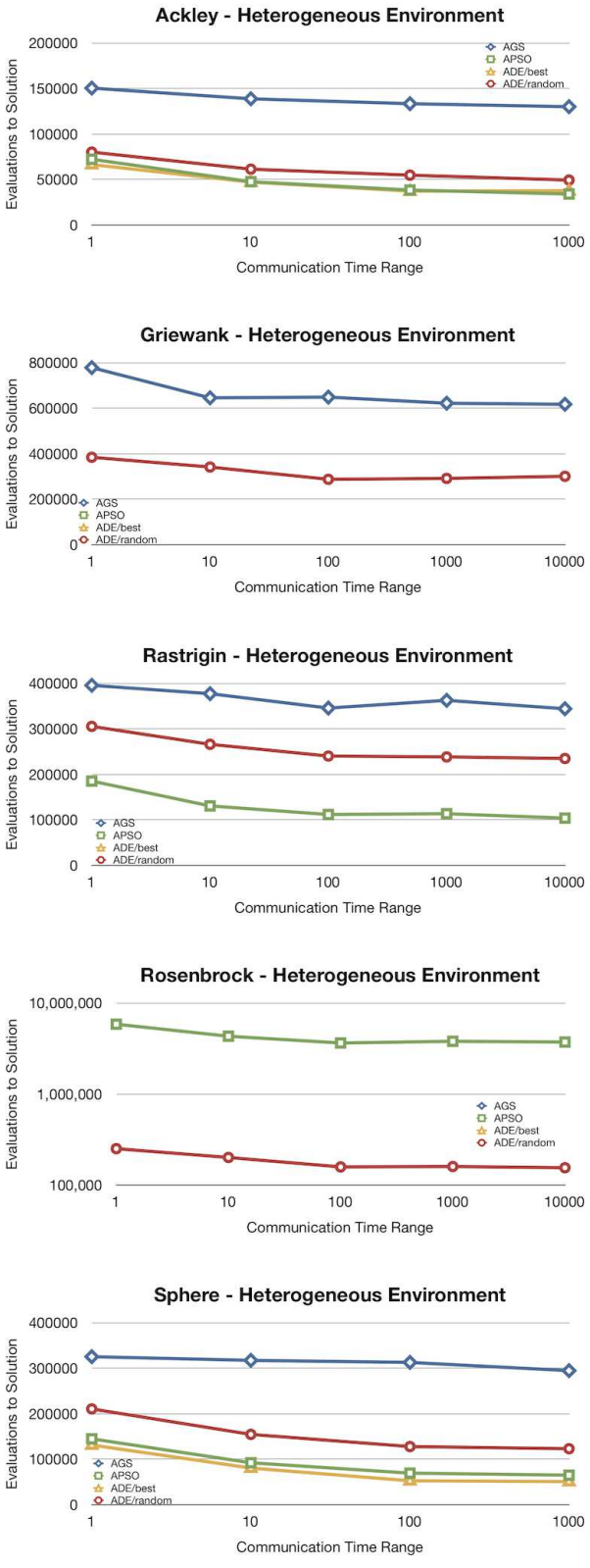

Fig. 5. Number of evaluations to solution for asynchronous EAs on simulated heterogeneous environments. A fixed compute time of 1 was used, and communication times were generated uniformly and randomly between 0 and the given range.

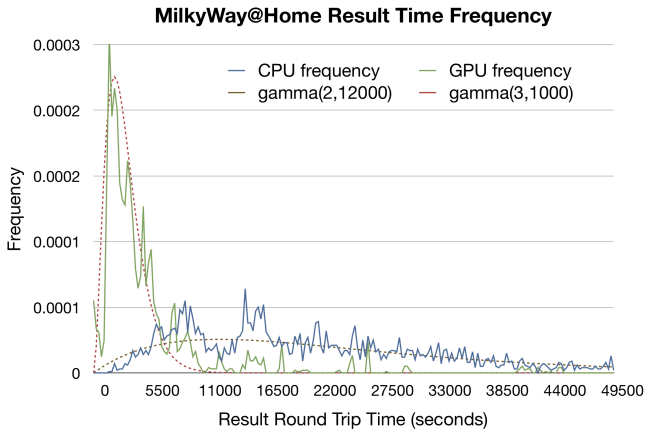

Fig. 6. Frequency of time taken to download, calculate and report results to the MilkyWay@Home server for GPU and CPU processors.

on the chart were found by calculating the frequency of a result being reported in 250 second bins, and dividing this frequency by the total number of data points and the bin size.

It was possible to fit the frequency GPU and CPU result times using the gamma distribution (see Equation 8), which is commonly used in probability to model waiting times. The gamma distribution function for a random variable x takes two positive input parameters, a shape parameter α and an inverse scale parameter β:

$$gamma(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{8}$$

$$\Gamma(\alpha) = (\alpha-1)!, \text{ if } \alpha \text{ is a positive integer} \tag{9}$$

Figure 6 shows the gamma distributions used to model the GPU and CPU wait times. These gamma distributions were used to randomly generate result times. As both α and β were integers, random result times could be generated as follows:

$$rand_{gamma(\alpha,\beta)} = \frac{1}{\beta} \Sigma_{i=1}^{\alpha} (\ln U_i) \tag{10}$$

Where U_i are all uniformly distributed on (0,1] and independent. As 40% of the results received by MilkyWay@Home were GPU results, the simulation generated result report times using $gamma(3, 1000)$ 40% of the time and with $gamma(2, 12000)$ otherwise.

Figure 7 shows the simulated time to solution for asynchronous EAs on a MilkyWay@Home-like computing environment, as the number of computing host increases from 100 to $100,000$. For all benchmark problems, as the number of simulated computing hosts increased the time to solution decreased, which is a promising result because it shows that in a more realistic environment, asynchronous EAs still scale well to very large environments. A population size of 100 was used for all searches.

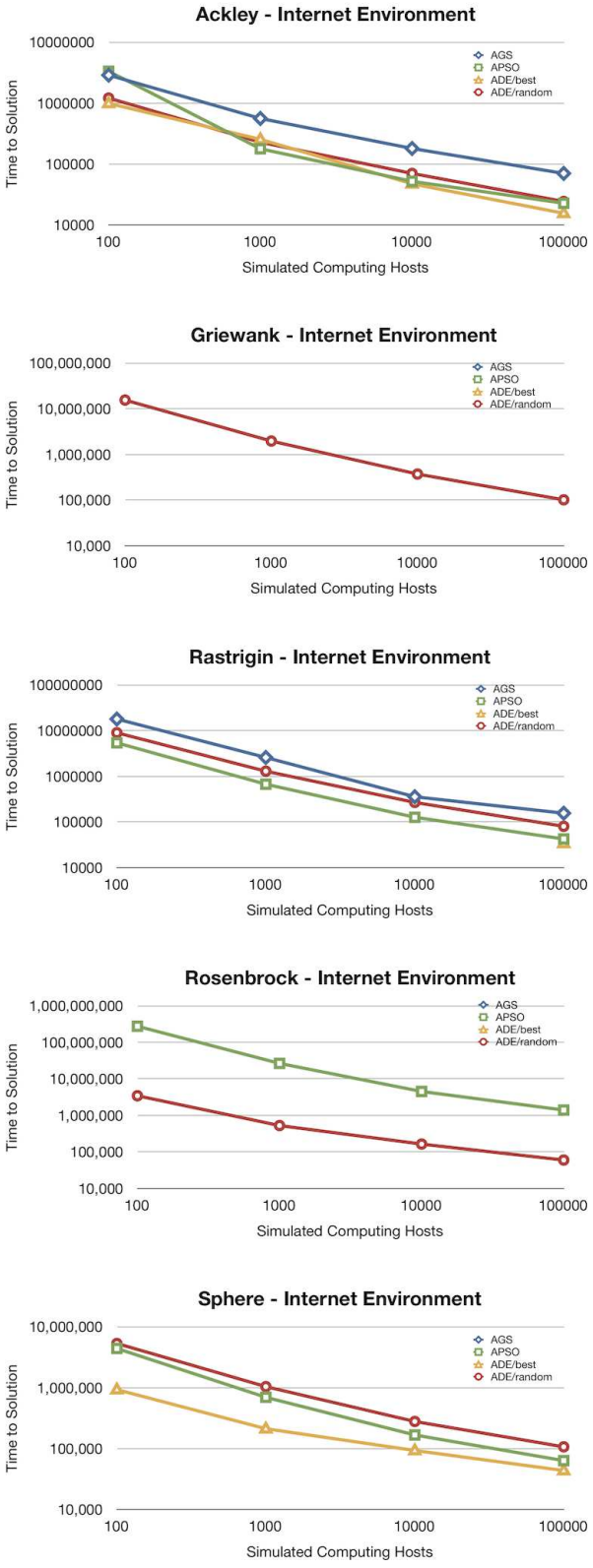

Fig. 7. Simulated time to solution for asynchronous EAs on a simulated internet-like environment for the different benchmark problems.

Interestingly, ADE/best, which could not solve the Rastrigin test function with a population size of 100 on a homogeneous environment or with smaller numbers of simulated computing hosts, was able to solve the test function when the number of workers increased to $100,000$. Not only could it solve the problem, but found solutions in less time than the other EAs that were used. Another unexpected result is that while AGS could solve the Sphere and Griewank problems in both the simulated homogeneous and simple heterogeneous environments, it could not solve the test function in a MilkyWay@Home-like environment. This is especially interesting in that the Sphere function is well defined and supposedly easy to solve as there is only a single minimum. These results seem to indicate that a realistic heterogeneous environment may harm the exploitative ability of AGS.

VI. Conclusions

This work presents different strategies for distributed evolutionary algorithms (EAs). A simulation framework is used to evaluate the performance of different EAs on various distributed computing systems. In addition to using simple simulated homogeneous and heterogeneous computing environments, data from the MilkyWay@Home volunteer computing project[2] is used to simulate an actual Internet-like distributed computing environment with thousands of heterogeneous hosts. It is shown that using an asynchronous strategy over a sequential strategy can greatly increase the scalability of an evolutionary algorithms. Traditional sequential distributed computing strategies are shown to not provide any improvement or can even increase the time to solution. Additionally, the asynchronous strategy presented is shown to be resilient to faults and highly heterogeneous computing environments, even taking less evaluations to reach a solution as the heterogeneity increases.

The EAs examined in this work only consist of single population strategies as these are seen as a more primitive problem which can be applied to multi-population (or island) strategies. As island strategies have been shown to potentially provide super-linear speed speed up [20], [21], future work would involve analyzing asynchronous EAs within islands on large and heterogeneous computing environments. Additionally, the MilkyWay@Home-like computing environment generated interesting results where certain EAs stopped being able to find the solution to benchmark problems, while others gained the ability to find the solution to benchmark problems they otherwise could not solve. This is particularly interesting considering that the heuristics used to generate new children did not change, only the order in which the results were received was changed. This provides a promising area for further research as to the effect of heterogeneity on distributed EAs.

As computing environments continue to become more distributed and heterogeneous, results from this work show that using an asynchronous strategy for distributed EAs

[2]http://milkyway.cs.rpi.edu

can provide significant benefits in terms of scalability and resilience to heterogeneity and faults.

ACKNOWLEDGMENT

Special thanks go to the Marvin Clan, David Glogau, and the Dudley Observatory for their generous donations to the MilkyWay@Home project, as well as the thousands of volunteers that made this data used in this work a possibility. This work has been partially supported by the National Science Foundation under Grant Numbers 0612213, 0607618, 0448407 and 0947637. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] J. Yen, J. C. Liao, B. Lee, and D. Randolph, "A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 29, no. 2, pp. 173–191, April 1998.

[2] S. C. Satapathy, J. Murthy, P. Prasada, V. Katari, S. Malireddi, and V. S. Kollisetty, "An efficient hybrid algorithm for data clustering using improved genetic algorithm and nelder mead simplex search," in *International Conference on Computational Intelligence and Multimedia Applications*, vol. 1, December 2007, pp. 498–510.

[3] V. Katari, S. C. Satapathy, J. Murthy, and P. P. Reddy, "Hybridized improved genetic algorithm with variable length chromosome for image clustering," *IJCSNS International Journal of COmputer Science and Network Security*, vol. 7, no. 11, pp. 121–131, November 2007.

[4] T. Desell, B. Szymanski, and C. Varela, "An asynchronous hybrid genetic-simplex search for modeling the milky way galaxy using volunteer computing," in *Genetic and Evolutionary Computation Conference*, Atlanta, Georgia, July 2008.

[5] R. Storn and K. Price, "Minimizing the real functions of the ICEC'96 contest by differential evolution," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, pp. 842–844.

[6] H.-P. Schwefel, *Evolution and Optimization Seeking*. New York: John Wiley & Sons, 1995.

[7] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Congress on Evolutionary Computation 2004 (CEC2004)*, vol. 2, June 2004, pp. 1980–1987.

[8] E. Mezura-Montes, J. Velzquez-Reyes, and C. A. C. Coello, "A comparative study of differential evolution variants for global optimization," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 2006, pp. 485–492.

[9] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[10] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Sixth International Symposium on Micromachine and Human Science*, 1995, pp. 33–43.

[11] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, May 1998, pp. 69–73.

[12] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George, "Parallel global optimization with the particle swarm algorithm," *International Journal for Numerical Methods in Engineering*, vol. 61, no. 13, pp. 2296–2315, December 2004.

[13] S. Baskar, A. Alphones, and P. N. Suganthan, "Concurrent PSO and FDR-PSO based reconfigurable phase-differentiated antenna array design," in *Congress on Evolutionary Computation*, vol. 2, June 2004, pp. 2173–2179.

[14] E. Cantu-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[15] H. Imade, R. Morishita, I. Ono, N. Ono, and M. Okamoto, "A grid-oriented genetic algorithm framework for bioinformatics," *New Generation Computing: Grid Systems for Life Sciences*, vol. 22, pp. 177–186, January 2004.

[16] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient hierarchical parallel genetic algorithms using grid computing," *Future Generation Computer Systems*, vol. 23, pp. 658–670, May 2007.

[17] T. Peachey, D. Abramson, and A. Lewis, "Model optimization and parameter estimation with Nimrod/o," in *International Conference on Computational Science*, University of Reading, UK, May 2006.

[18] A. Lewis and D. Abramson, "An evolutionary programming algorithm for multi-objective optimisation," in *IEEE Congress on Evolutionary Computation (CEC2003)*, vol. 3, December 2003, pp. 1926–1932.

[19] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *Congress on Evolutionary Computation 2004 (CEC2004)*, vol. 2, June 2004, pp. 2023–2029.

[20] E. Alba and J. M. Troya, "Analyzing synchronous and asynchronous parallel distributed genetic algorithms," *Future Generation Computer Systems*, vol. 17, pp. 451–465, January 2001.

[21] J. Berntsson and M. Tang, "A convergence model for asynchronous parallel genetic algorithms," in *IEEE Congress on Evolutionary Computation (CEC2003)*, vol. 4, December 2003, pp. 2627–2634.

[22] G. Folino, A. Forestiero, and G. Spezzano, "A JXTA based asynchronous peer-to-peer implementation of genetic programming," *Journal of Software*, vol. 1, pp. 12–23, August 2006.

[23] B.-I. Koh, A. D. George, and R. T. Haftka, "Parallel asynchronous particle swarm optimization," *International Journal of Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, July 2006.

[24] G. Venter and J. Sobieszczanski-Sobieski, "A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations," in *Sixth World Congresses of Structural and Multidisciplinary Optimization*, May 2005, pp. 1–10.

[25] J. R. Prez and J. Basterrechea, "Particle swarms applied to array synthesis and planar near-field antenna measurements," *Microwave and Optical Technology Letters*, vol. 50, no. 2, pp. 544–548, February 2008.

[26] L. Xu and F. Zhang, "Parallel particle swarm optimization for attribute reduction," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, vol. 1, July 2007, pp. 770–775.

[27] T. Desell, "Asynchronous global optimization for massive scale computing," Ph.D. dissertation, Rensselaer Polytechnic Institute, 2009.

[28] L. Wei and M. Zhao, "A niche hybrid genetic algorithm for global optimization of continuous multimodal functions," *Applied Mathematics and Computation*, vol. 160, no. 3, pp. 649–661, January 2005.

[29] R. Chelouah and P. Siarry, "Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions," *European Journal of Operational Research*, vol. 148, no. 2, pp. 335–348, July 2003.

[30] P. Koduru, S. Das, and S. Welch, "A particle swarm optimization-Nelder Mead hybrid algorithm for balanced exploration and exploitation in multidimensional search space," in *IC-AI*, H. R. Arabnia, Ed. CSREA Press CSREA Press, 2006, pp. 457–464.

[31] Z. G. Wang, M. Rahman, Y. S. Wong, and J. Sun, "Optimization of multi-pass milling using parallel genetic algorithm and parallel genetic simulated annealing," *International Journal of Machine Tools and Manufacture*, vol. 45, no. 15, pp. 1726–1734, December 2005.

[32] J. Liu, W. Xu, and J. Sun, "Quantum-behaved particle swarm optimization with mutation operator," in *International Conference on Tools with Artificial Intelligence*, November 2005.

[33] Z. Dingxue, G. Zhihong, and L. Xinzhi, "An adaptive particle swarm optimization algorithm and simulation," in *IEEE International Conference on Automation and Logistics*, August 2007, pp. 2399–2042.

[34] F. V. D. Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, June 2004.