Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Parameter setting and exploration of TAGS using a genetic algorithm

Hagit Sarfati Eitan Bachmat Sagit Kedem-Yemini

***Abstract*— We consider the performance of TAGS, a multi-host job assignment policy. We use a genetic algorithm to compute the optimal parameter settings for the policy. We then explore the performance of the policy using the optimal parameters, when the job size distribution is a heavy-tailed Bounded Pareto distribution with parameter α. We show that TAGS only operates at low inter-arrival rates. At low rates it is very efficient in comparison with other standard policies. At high rates TAGS has to be combined with other policies to achieve good performance. We also show that the performance is nearly symmetrical around the value $\alpha = 1$, with the best performance when $\alpha = 1$.**

Keywords: Multiple host task assignment, Heavy-tailed distributions, Genetic algorithm.

I. Introduction

Many installations such as web server farms and computing centers have a multitude of hosts which can serve any incoming request. There has been a growing body of research regarding scheduling policies for such multi-host systems. When the job size distribution is exponential it is claimed that the `Least-work-remaining` policy is optimal, [10]. However, recent empirical data has suggested that many workloads which are typical of networked systems are *heavy-tailed*, rather than exponential, [4], [5], [12]. Heavy-tailed distributions have very large variance and it is well known that high variance in job size adversely affects response time. In the case of unknown job sizes these considerations have led Harchol-Balter, [6] to suggest a policy where each host is responsible for a certain range of job sizes, thus reducing the variance at each host. The policy was named TAGS (Task Assignment based on Guessing Size). Harchol-Balter has shown that using two servers such policies can dramatically reduce response time in comparison with `Random` or `Least-Work-Remaining` policies when the workload is heavy-tailed. As noted already in [6] one of the major problems in exploring TAGS systems with more than two hosts is to de■ne the optimal set of ranges. We developed a simple genetic algorithm which chooses good range parameters. The algorithm execution time is fairly quick and its performance is not affected by the number of hosts. It allows the development of a dynamic version of TAGS for environments in which the job size distribution changes. Using the parameters chosen by the genetic algorithm we were able to explore the performance of TAGS on systems with many hosts. We arrived at many interesting and illuminating conclusions. In particular, we determine the optimal number of hosts in a TAGS system and show that the number is fairly small. We also study the loads and ranges of the different hosts for various job size distributions. Following these discoveries we were able in some cases to develop a mathematical explanation for the observed phenomenon, [1], which led to a much better understanding of the complex behavior of TAGS.

Hagit Sarfati is an M.Sc student at the Department of Industrial Engineering, Ben-Gurion University, Beer-Sheva, Israel, 84105. hagitbachmat@yahoo.com

Eitan Bachmat is a member of the Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel, 84105. ebachmat@cs.bgu.ac.il, tel. +972-8-6477858, fax +972-8-6477650

Sagit Kedem-Yemini is a Ph.D student at the Department of Industrial Engineering, Ben-Gurion University, Beer-Sheva, Israel, 84105. ksagit@bgu.ac.il

The paper is organized as follows:
In section 2, we provide background information on multi-host scheduling policies and on some heavy-tailed (Bounded Pareto) job size distributions.
In section 3 we present the genetic algorithm.
In section 4 we present the results of the analysis of TAGS using the genetic algorithm.
In section 5, we summarize the paper and point to future work.

II. Preliminaries

A. TAGS *and other assignment methods*

In this paper we consider multi-host assignment policies in the case where job sizes are not known. We will assume though, that the job size distribution is known or can be deduced by collecting statistics. This is a reasonable assumption for many current systems. We introduce several policies. The ■rst, the TAGS scheduling policy, is the subject of investigation in this paper. It was introduced in [6]. The main feature of this policy is variance reduction at the different hosts. For more on variance reduction policies see [7], [8], [13].

The TAGS policy has been described in, [6], as follows:
Consider hosts numbered $1, \ldots, h$. The i'th host, $i < h$, has a number s_i associated with it, where $s_1 < s_2 < \ldots < s_{h-1}$. All incoming jobs are dispatched to host 1. They are serviced in ■rst come, ■rst served (FCFS) order. If the job completes before s_1 processing time units, it leaves the system. If a job is not complete after s_1 time units it is killed and put at the end of the queue of host 2, where it starts from scratch. More generally, If a job at host i uses s_i time, it is killed and put at the end of the queue of host $i + 1$. Each host services jobs in FCFS order.

In general, given a job size distribution F, the choice of s_i, $i = 1, \ldots, h-1$, can be used to minimize various objective functions. The purpose of the genetic algorithm is to ■nd s_i which minimize average response time.

In addition to TAGS, there are other, more classical assignment policies. We brie󰀀y described, some of the better known policies.

The Random assignment policy assigns each incoming job to a uniformly random host. The job runs on the assigned host to completion.

The Round Robin assignment policy assigns the i'th arriving job to host $(i - 1 \bmod h) + 1$.

The Least-Work-Remaining assignment policy assigns each incoming job to the host with the smallest amount of remaining work at the time of arrival. It is known that the Least-Work-Remaining policy can be implemented without knowledge of job sizes using a central queue dispatcher.

These classical policies share some basic features. They are all work preserving, namely, no jobs are killed. They are all load balanced. The job size distribution experienced by each individual host is the same as the original job size distribution entering the system.

In contrast, TAGS is not work preserving, hence, the actual load experienced by a TAGS system is greater than the incoming load. Depending on the choice of range parameters, s_i, TAGS may not be load balanced. We will see later on that in many cases, the optimal choice of parameters may lead to strongly unbalanced systems. The job size distribution experienced by individual hosts in a TAGS system has much smaller range and variance than the original input stream. This last property is the key to the success of TAGS in certain instances.

We can also consider hybrid assignment policies which combine TAGS with Random (or Round robin). In an hybrid (h, l) policy the hosts are divided into l groups, each containing h hosts. Each group is assigned jobs using Random, while jobs which are assigned to a given group are processed using TAGS assignment on the h hosts of the group. The hybrid policies have the same basic properties as TAGS.

B. Bounded Pareto distributions

The job size distributions which were used in our experiments were Bounded Pareto, see [6] for more details. A distribution is said to be Bounded Pareto if its density f has the form

$$f(s) = cs^{-\alpha-1} \tag{1}$$

in a bounded range $k \leq s \leq p$ and $0 < \alpha < 2$. The constant, $c > 0$, is a normalizing constant which ensures that $\int_k^p f(s)\ ds = 1$. A simple computation shows that the normalizing constant is

$$c = \frac{k^\alpha}{1 - (\frac{k}{p})^\alpha}$$

We denote this Bounded Pareto distribution by $B(k, p, \alpha)$. We also let $r = p/k$ denote the range of the distribution.

III. The genetic algorithm for optimizing the choice of S_i

In this section we describe a simple, mutation based, genetic algorithm for 󰀀nding the values of s_i, $i = 1, \ldots, h-1$, which optimize response time in a TAGS system with h hosts and job size distribution F. For an analysis of mutation based algorithms, see [2], [14]. Having a fast and simple algorithm for 󰀀nding the optimal parameters for a TAGS system is important since job size distributions may vary in time. By keeping track of job sizes in a time window one can employ the genetic algorithm to dynamically 󰀀nd the best parameters. This is ideal for situations in which the distribution does not change abruptly. We may expect that the parameter values from previous data are still ef󰀀cient.

Other methods such as gradient descent can also be applied to this problem, when there is a reasonable analytical approximation to the objective function. As we will show, in our case gradient descent can only be applied with great dif󰀀culty. The simplicity of the genetic procedure and the 󰀀exibility of considering non-analytical functions, or dynamic objective functions, give it a distinct advantage.

A. The objective function

The objective function for the algorithm was the average waiting time for jobs in the system. The average waiting time has no exact analytical formula because the input stream to the second host and beyond is not Poisson. Instead we use the approximation which assumes a Poisson input stream to all hosts. The approximation is conservative since the input streams, to all but the 󰀀rst host, tend to be more regular than Poisson, having near constant inter-arrivals. Such input streams lead to better response times than a Poisson stream. The approximation we use is based on the Pollaczek-Khinchine formula and is taken from the appendix to [6]. Computing the objective function with more precision would require, simulating the TAGS policy. While this is possible, it would be time consuming and would slow down the computations by orders of magnitude. In the present case, when a reasonable and conservative approximation exists, it is probably best to use simulations only for local searching, after the genetic algorithm has identi󰀀ed good candidate solutions.

The input to the objective function is

- h - The number of hosts.
- F - A job size distribution which in our case is a Bounded Pareto job size distribution $B(k, p, \alpha)$.
- ρ - The average incoming load-per-host. The value ρ represents the load per host for a load balanced, work preserving system. The actual loads in the TAGS system will be different.
- A set of parameters, $k < s_1 < s_2 < \ldots < s_{h-1} < p$ for the TAGS algorithm.

The objective function is calculated as follows.

1) We compute the 󰀀rst and second moments of $B(k, p, \alpha)$.

$$E(X^j) = \frac{k^\alpha}{1 - (\frac{k}{p})^\alpha} \frac{k^{j-\alpha} - p^{j-\alpha}}{\alpha - j} \tag{2}$$

2) We calculate the inter-arrival rate

$$\lambda = \frac{1}{E(X)} h\rho \tag{3}$$

3) We compute, p_i, the portion of jobs which pass through host i but not through host $i+1$ in the TAGS system. These are precisely the jobs with size $s_i < x \le s_{i+1}$

$$p_i = \frac{k^\alpha}{1-(\frac{k}{p})^\alpha}(s_{i-1}^\alpha - s_i^\alpha) \tag{4}$$

We also compute the portion of jobs which pass through host i, that is, jobs of size $s_i < x$.

$$p_i^{visit} = \frac{k^\alpha}{1-(\frac{k}{p})^\alpha}(s_{i-1}^\alpha - p^\alpha) \tag{5}$$

4) We compute the first and second moments of the distribution of jobs which visit host i but not host $i+1$. This is the same as formula (2) with s_{i-1} and s_i replacing k and p respectively.

$$E(X_i^j) = \frac{s_{i-1}^\alpha}{1-(\frac{s_{i-1}}{s_i})^\alpha} \frac{s_{i-1}^{j-\alpha} - s_i^{j-\alpha}}{\alpha - j} \tag{6}$$

5) The jobs which pass through host i consist of those which do not pass onto host $i+1$ and those who do. The former have average service time $E(X_i^j)$ which is given by formula (6), while the latter have service time s_i at host i. We conclude that the average waiting time at host i is the weighted average

$$E(X_i^{visit}) = \frac{p_i}{p_i^{visit}} E(X_i) + (1 - \frac{p_i}{p_i^{visit}}) s_i \tag{7}$$

Similarly for the second moment

$$E((X_i^{visit})^2) = \frac{p_i}{p_i^{visit}} E(X_i^2) + (1 - \frac{p_i}{p_i^{visit}}) s_i^2 \tag{8}$$

6) We compute the arrival rate of host i

$$\lambda_i = \lambda p_i \tag{9}$$

and the utilization of host i

$$\rho_i = \lambda_i E(X_i^{visit}) \tag{10}$$

7) Since we are assuming a Poisson input stream we may apply the Pollaczek-Khinchine formula to obtain the waiting time at host i

$$E(W_i^{visit}) = \frac{\lambda_i E(X_i^{2(visit)})}{2(1-\rho_i)} \tag{11}$$

The waiting time of a job which finishes at host i is then

$$E(W_i) = \sum_{j=1}^{i} E(W_i^{visit}) \tag{12}$$

and finally the average waiting time for all jobs is the weighted average

$$E(W) = \sum_{i=1}^{h} E(W_i) p_i \tag{13}$$

$E(W)$ is our target function.

The objective function is computed in steps. If we were to write directly the average waiting time $E(W)$ as a function of the input we would obtain an extremely long expression. Manipulating such huge expressions is difficult. In particular, differentiating such expressions without error can only be done using specialized, software. This makes gradient methods much harder to implement.

B. The genetic algorithm

We describe the genetic algorithm which we used for optimizing the expression $E(W)$, described above. The algorithm uses only mutations, without a cross over operator. Other applications of mutation-only or mutation-based genetic algorithms can be found in [3], [9], [11], among others. In addition to mutations, we employ a very basic elitism mechanism.
The original algorithm worked as follows:

1) The population in each iteration consists of n chromosomes.
2) Each chromosome is a sequence $k < s_1 < s_2 < \ldots < s_{h-1} < p$ of possible values for the s_i.
3) There is a given number m of reserved chromosomes. A reserved chromosome is one that passes unchanged to the next generation according to rules which are described below. This reservation rule is our implementation of elitism.
4) In each iteration, all the non reserved chromosomes undergo the following mutation:
An integer value $1 \le i \le h-1$ is chosen uniformly. The value s_i in the chromosome is replaced by a new value s_i^{new} which is chosen uniformly in the range $[s_{i-1}, s_{i+1}]$. All the remaining values in the chromosome remain unchanged.
5) After all the mutations have been performed, the objective function is calculated for all chromosomes.
6) The population is ranked according to the objective function. The m chromosomes with the top scores are moved automatically to the next generation and are tagged as reserved chromosomes. Each chromosome is assigned a probability q_i. The i'th ranked chromosome is assigned the probability $q_i = \frac{(n-i+1)^2-(n-i)^2}{n^2}$. After that the entire population is sampled $n-m$ times, including the reserved chromosomes. The $n-m$ chosen chromosomes, together with the m reserved chromosomes constitute the next generation.
7) The process is repeated until no improvement in the best score is found for a given number of generations.

The algorithm as described above had difficulty dealing with a very large range r which is typical of applications. Specifically, the mutation process, as described in step 4, was unable to explore the range using the uniform distribution. The problem was resolved by working with logarithmic values. The mutation step 4 is replaced by:

4') An integer value $1 \leq i \leq h-1$ is chosen uniformly. we choose a value γ uniformly in the range $[log_{10}(s_{i-1}), log_{10}(s_{i+1})]$. The value s_i in the chromosome is replaced by the new value $s_i^{new} = 10^\gamma$. All the remaining values in the chromosome remain unchanged.

We note that a uniform distribution in logarithmic values means that each order of magnitude has the same probability of being chosen. We found that the new version performed very well regardless of range size.

C. Performance of the genetic algorithm

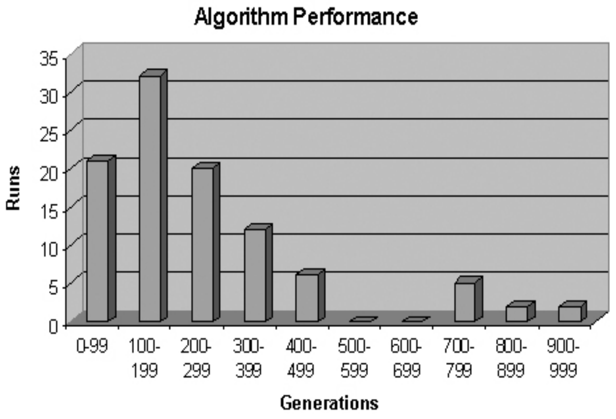


Fig. 1. Number of generations to stabilization.

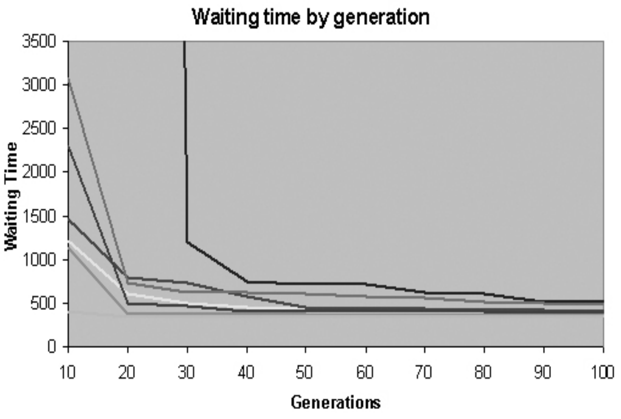


Fig. 2. Waiting time reduction by generation.

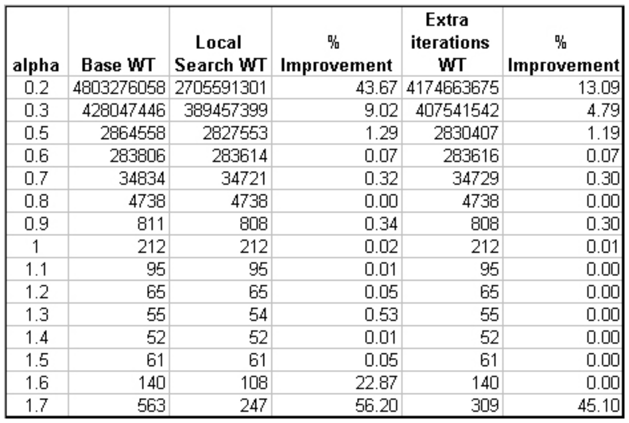
Figures 1 and 2 show the rate in which the genetic algorithm reaches a stable state, with 8 hosts and $\alpha = 0.9$. Figure 1 shows the number of generations (with population size 100) required to obtain an objective function value within 10% of the best value found after 10000 generations. The histogram shows the results for 100 different runs. As can be seen the total number of computations is usually less than $50,000$ (generations * population size) and in all cases was less than 10^5. In comparison, the size of the search space is about $10^{24}/8! \approx 3 \times 10^{18}$. Figure 2 shows the best value of the objective function over successive generations for several randomly chosen runs. Again, we see that the algorithm nearly reaches its best value after less than 100 generations.

D. Localizing the genetic algorithm

We also employed a localized version of the genetic algorithm. The localized version differs from the original genetic algorithm by replacing step $4'$ by:

4") An integer value $1 \leq i \leq h-1$ is chosen uniformly. we choose a value γ in the range $[log_{10}(s_{i-1}), log_{10}(s_{i+1})]$ using a normalized Gaussian with cutoffs, centered at s_i. The value s_i in the chromosome is replaced by the new value $s_i^{new} = 10^\gamma$. All the remaining values in the chromosome remain unchanged.

The choice of a Gaussian centered at s_i means that the new value s_i^{new} chosen in step 4" will tend to be closer to s_i than a new value chosen according to 4'. Such a strategy makes sense near a locally optimal value. Conversely, if this strategy works better than the basic version it may provide an indication that we are close to an optimal value. The disadvantage of the localized version is that it explores less at the initial stages of the search, therefore, takes longer to stabilize. Consequently we combined the 2 versions. We ran the basic genetic algorithm until the waiting time seemed to stabilize. We then ran the localized version, starting with the last generation of the basic algorithm until it seemed to stabilize.



alpha	Base WT	Local Search WT	% Improvement	Extra iterations WT	% Improvement
0.2	4803276058	2705591301	43.67	4174663675	13.09
0.3	428047446	389457399	9.02	407541542	4.79
0.5	2864558	2827553	1.29	2830407	1.19
0.6	283806	283614	0.07	283616	0.07
0.7	34834	34721	0.32	34729	0.30
0.8	4738	4738	0.00	4738	0.00
0.9	811	808	0.34	808	0.30
1	212	212	0.02	212	0.01
1.1	95	95	0.01	95	0.00
1.2	65	65	0.05	65	0.00
1.3	55	54	0.53	55	0.00
1.4	52	52	0.01	52	0.00
1.5	61	61	0.05	61	0.00
1.6	140	108	22.87	140	0.00
1.7	563	247	56.20	309	45.10

Fig. 3. Extra iterations vs. Localized strategy.

The table in figure 3 presents a comparison of the localized version and the basic one. The results are for 4-host systems with $\rho = 0.5$. The "Base WT" column of the table shows the results of the basic algorithm. The "Local search WT" column shows the results after continuation with the localized version. For comparison we also continued running the basic algorithm for the same number of extra generations as the localized version. The "Extra iterations WT" column presents the results after the extra generations with the basic version. As can be seen, the localized version always out-performed the basic version, but for most values of α only by a very small amount. This result lends strong credibility to the assertion that we are close to an optimum. The relatively large improvements for the more extreme values of α can be explained by considering figure 5. As can be seen from the figure, some of the hosts in these cases have very high utilization near 1. The term $\frac{1}{2(1-\rho)}$ in the Pollaczek-Khinchine equation shows that the waiting time is very sensitive in this region. A large step can simply move to a solution with $\rho > 1$ which will lead to infinite waiting time. Therefore small steps are more likely to lead to an improvement in the objective function.

IV. Results

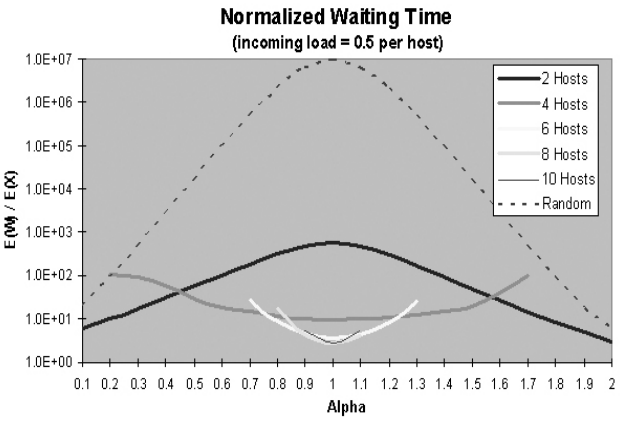


Fig. 4. The performance of TAGS vs. Random assignment.

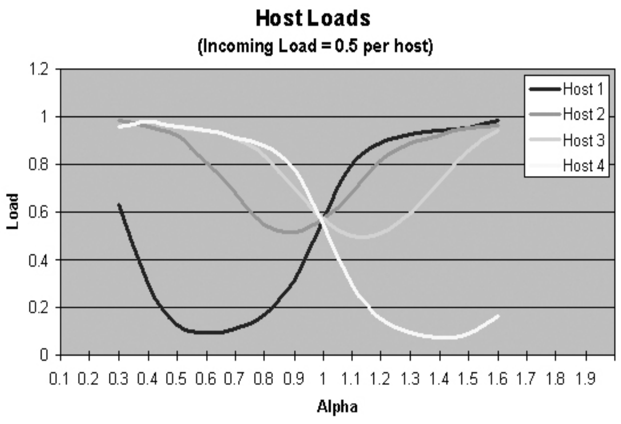


Fig. 5. The loads on each host as a function of α.

The results of our experiments are summarized in figures 2 – 4. In all the experiments we used Bounded Pareto distributions of the form $B(1, 10^{10}, \alpha)$. All these distributions have the same range of $r = 10^{10}$. Since the range and the value of α are both invariant under changes of time units, a fixed range provides a good normalization for Bounded Pareto distributions. In addition the incoming load is set to be proportional to the number of hosts (0.5 per host). We wish to compare TAGS, Random and hybrid assignments.

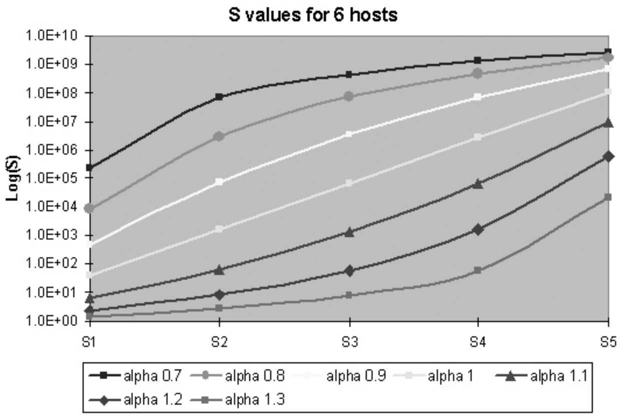


Fig. 6. The values of s_i as a function of α.

Consider a system with 30 hosts. We can execute on such a system a TAGS assignment policy, a Random assignment policy or a hybrid $(2, 15)$, $(3, 10)$, $(5, 6)$, $(6, 5)$, $(10, 3)$, $(15, 2)$ $(30, 1)$ assignment policy. In the case of a hybrid $(2, 15)$ assignment policy, the 30 host system is subdivided into 15 2-host subsystems. Jobs are assigned randomly among the 15 subsystems of hosts. Therefore, the load per host in each 2-host subsystem remains 0.5. Each 2-host subsystem runs a TAGS policy, so we conclude that the performance of the hybrid $(2, 15)$ policy will be the same as that of a 2-host TAGS system with load 0.5 per host. Similarly, the other hybrid policies have the same performance as that of a TAGS system with $3, 5, 6, 10, 15, 30$ hosts respectively. The hybrid $(1, 30)$ system coincides with the Random policy. We see that a comparison of the possible hybrid systems with TAGS and Random is achieved by looking at the performance of TAGS.

Apart from a comparison of different assignment policies we would also like to know how well the policies perform for different job size distributions. In particular we are interested in the performance of the assignment policies as α varies in the domain $0 < \alpha < 2$. We recall that small values of αcorrespond to a very heavy tail, which gets lighter as α increases.

Given a job size distribution, we let $E(X)$ denote its average job size. This is the average response time in an ideal system which has no queues, i.e., at extremely low arrival rates. Given a system with any assignment policy we let $E(W)$ denote its response time. Our system performance metric is $E(W)/E(X)$ which compares the performance of the system to a gold standard system with no queues. This measure allows us to produce a fair comparison of performance for different values of α. A direct comparison via $E(W)$ would not be consistent because, for fixed range r, the distributions $B(1, r, \alpha)$ produces different average job sizes.

In figure 4 we see a comparison of the performance of the Random assignment policy, with TAGS assignment for $2, 4, 6, 8$ and 10 hosts across all values of α. We observe that the performance of Random is worst when $\alpha = 1$. Its response time for $\alpha = 1$ is a staggering 10^6 times slower than a system with no queues. For a TAGS system with 2 hosts, $\alpha = 1$ still leads to the worst performance. However, the performance improvement over Random is already the largest at $\alpha = 1$. For more than 2 hosts,the performance of TAGS is best at $\alpha = 1$ and is orders of magnitude better than Random. It is interesting to see that TAGS performs best where Random performs worst.

Next we observe that, as we move to 4 hosts and above, there are many values of α near 0 and 2 for which TAGS has no performance numbers. The reason is that in these cases the actual system load is greater than the incoming load and exceeds the system capabilities, regardless of the values of the parameters s_i. This is caused by the overhead that TAGS created by killing jobs and restarting them from scratch. In these cases there are no values for s_i for which the load on each host is less than 1. Consequently, there is no stable TAGS system in these cases and queues will explode, resulting in in□nite average waiting times. We note that this is not a problem of the genetic algorithm, it represents a real problem of TAGS.

Following our experimental results we were able to derive an analytic criterion for the existence of a stable TAGS system, [1]. We have veri□ed that in the cases in which the genetic algorithm did not □nd any legitimate values for the s_i, such values simply do not exist.

Having more hosts improves performance only in a small range around $\alpha = 1$ and improvement stops completely beyond 10 hosts. Returning to our example of a 30 host system with load 0.5 per host, we see that different hybrid systems perform best for different values of α. When $\alpha < 0.5$ or $\alpha > 1.5$, a $(2, 15)$ hybrid system is best. Then, for $0.5 < \alpha < 0.7$ and $1.3 < \alpha < 1.5$ a system with 7 groups of 4 hosts and one group of 2 hosts would be better. When $|\alpha - 1| < 0.3$, the performance of a $(6, 5)$ hybrid system is better. Finally, when $\alpha = 1$ an $(3, 10)$ hybrid system would be best.

Another important observation which is repeated in many of the experiments is the near symmetry in performance between the Bounded Pareto distribution with parameter α and that with parameter $2 - \alpha$.

Figures 5 and 6 provide us with information on the structure of the optimal values of s_i and on the load of each host. Figure 5 shows the load of the different hosts on a 4-host system. We observe that for small values of α the hosts which are responsible for large job sizes, hosts $2, 3$ and 4, remain very busy and their load never drops below 0.5. Host 1, on the other hand, remains relatively free, even though it is responsible for a large portion of the jobs. The reason is that for small values of α, there are relatively many large jobs and so the hosts responsible for such jobs remain heavily loaded. When $\alpha > 1$ the situation is reversed. Most hosts are responsible for the many small jobs and their utilization is high. In fact the utilization of hosts 1 and 4 and of hosts 2 and 3 nearly mirror each other with respect to $\alpha = 1$. At the central point $\alpha = 1$, the optimal values of s_i form a geometric sequence as can be seen in □gure 6 (linear graph in the logarithmic scale). The system is also load balanced. The load balancing together with the variance reduction achieved by making each host responsible for a relatively small range are the keys to the impressive results of TAGS in this case. We see again that there is near inverted symmetry in the values of s_i between α and $2 - \alpha$.

V. Conclusions and future work

In this paper we presented a fast and simple genetic algorithm for □nding good range parameters for the TAGS multi-host assignment policy. The algorithm was based on mutations, and also incorporated a small degree of elitism. Using the algorithm we analyzed the performance of TAGS and compared it with the more classical Random assignment policy, assuming a Poisson arrival process and a Bounded Pareto job size distribution.

In accordance with [6] we have found that in many cases TAGS affords performance which is orders of magnitude better than Random. The limiting factor for achieving these large performance gains is system load. TAGS systems do not handle load well because they kill and restart jobs. The solution is to employ hybrid methods which combine small groups, managed using TAGS with a Random policy between the groups. Among Bounded Pareto job size distributions, TAGS performs best when $\alpha = 1$. The performance when the job size distribution is $B(k, p, \alpha)$ is nearly the same as for $B(k, p, 2 - \alpha)$. We have also shown that the behavior of the values of s_i is nearly anti-symmetrical when comparing the workloads $B(k, p, \alpha)$ to $B(k, p.2 - \alpha)$.

The genetic algorithm approach becomes even more important when we consider TAGS systems with heavy-tailed inter-arrival distributions. In that context we do not even have reasonable analytic approximations, hence the objective function can only be deduced using simulations. In such cases, a heuristic search approach becomes absolutely essential. We hope to explore this problem in future work.

The many insights we have gained by using the results of the genetic algorithm can also be seen as the starting point for many future analytical investigations. The results that show that the load that a TAGS system can handle depends on α have led, very recently, to more precise analytical formulas for the load handling capabilities of TAGS, see [1]. In particular, it has been shown that, regardless of the values of p and k, a TAGS system with job size distribution $B(k, p, \alpha)$ can never handle a total system load of more than $(1 - \alpha)^{-1/\alpha}$, when $\alpha < 1$, and $\frac{\alpha-1}{\alpha}$, when $\alpha > 1$. We also hope to shed more light on the observation that for $\alpha = 1$ the optimal s_i form a nearly geometric sequence and to □nd generalizations to other values of α.

Acknowledgments: We would like to thank the anonymous referee for suggesting the localized version of the genetic algorithm.

References

[1] E. Bachmat and H. Sarfati, Load handling capabilities and performance of TAGS, submitted, available at www.cs.bgu.ac.il/ ebachmat.

[2] T. Bck, Optimal Mutation Rates in Genetic Search. *Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA*, Morgan Kaufmann, 2-8, 1993.

[3] E.K. Burke, J.P. Newall and R.F. Weare, A Memetic Algorithm for University Exam Timetabling, *Lecture Notes in Computer Science vol. 1153. The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT 95), Burke, E.K., Ross, P. (eds)*, Springer-Verlag, Berlin, Heidelberg, New York, 241-250, 1996.

[4] M.E. Crovella and A. Bestavros, Self-similarity in world wide web traf□c: Evidence and possible causes. *IEEE/ACM Transactions on networking*, Vol. 5(6), 835-846, 1997.

[5] M.E. Crovella, M.S. Taqqu and A. Bestavros, Heavy-tailed probability distributions in the world wide web. In *A practical guide to heavy tails*, chapter 1, 1-23, Chapman and Hall, New York, 1998.

[6] M. Harchol-Balter, Task assignment with unkown duration, *Journal of the ACM*, vol. 49(2), 260-288, 2002.

[7] H. Feng, V. Misra and D. Rubenstein, Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems, *Performance evaluation*, vol. 62, 475-492, 2005.

[8] M. Harchol-Balter, M. Crovella and C. Murta, On choosing a task assignment policy for a distributed server system, *IEEE Journal of parallel and distributed computing*, Vol. 59, 204-228, 1999.

[9] T.L. Lau, and E.P.K. Tsang, Applying a Mutation-Based Genetic Algorithm to Processor Con□guration Problems *8th International Conference on Tools with Artificial Intelligence (ICTAI '96)*, 17-24, 1996.

[10] R.D. Nelson and T.K. Philips, An approximation for the mean response time for the response time of shortest queue routing, *Performance evaluation review*, vol. 7, 181-189, 1989.

[11] X. Pan, Jian Zhang and K.Y. Szeto, Application of Mutation Only Genetic Algorithm for the Extraction of Investment Strategy in Financial Time Series, *International conference on Neural Networks and Brain,ICNNB'05*, Vol. 3, 1682-1686, 2005.

[12] D.L. Peterson and D.B. Adams, Fractal patterns in DASD I/O traf□c. In *CMG Proceedings*, 1996.

[13] B. Schroeder and M. Harchol-Balter, *Evaluation of Task Assignment Policies for Supercomputing Servers: The Case for Load Unbalancing and Fairness*, *Proc. of the 9th IEEE Symposium on High Performance Distributed Computing (HPDC)* , 2000.

[14] S.A. Stanhope and J.M. Daida, (1+1) Genetic Algorithm Fitness Dynamics in a Changing Environment, *CEC-99: Congress in Evolutionary Computation , Piscataway*, IEEE Press, 1851 1858, 1999.