Scheduling Meta-tasks in Distributed Heterogeneous Computing Systems: A Meta-Heuristic Particle Swarm Optimization Approach

Hesam Izakian[1], Ajith Abraham[2], Václav Snášel[3]
[1]Department of Computer Engineering, University of Isfahan, Isfahan, Iran
[2]Machine Intelligence Research Labs –MIR Labs, USA
[3]Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic
hesam.izakian@gmail.com, ajith.abraham@ieee.org, vaclav.snasel@vsb.cz

Abstract

Scheduling is a key problem in distributed heterogeneous computing systems in order to benefit from the large computing capacity of such systems and is an NP-complete problem. In this paper, we present a Particle Swarm Optimization (PSO) approach for this problem. PSO is a population-based search algorithm based on the simulation of the social behavior of bird flocking and fish schooling. Particles fly in problem search space to find optimal or near-optimal solutions. The scheduler aims at minimizing make-span, which is the time when finishes the latest task. Experimental studies show that the proposed method is more efficient and surpasses those of reported PSO and GA approaches for this problem.

1. Introduction

A distributed heterogeneous computing (HC) system consists of a distributed suite of different high-performance machines, interconnected by high-speed networks, to perform different computationally intensive applications that have various computational requirements. Heterogeneous computing systems range from diverse elements or paradigms within a single computer, to a cluster of different types of PCs, to coordinated, geographically distributed machines with different architectures (e.g., grids [1]).

To exploit the different capabilities of a suite of heterogeneous resources effectively and satisfy users with high expectations for their applications, a crucial problem that needs to be solved in the framework of HC is the scheduling problem.

Optimally scheduling is mapping a set of tasks to a set of resources to efficiently exploit the capabilities of such systems. As mentioned in [2] optimal mapping tasks to machines in an HC suite is an NP-complete problem and therefore the use of heuristics is one of the suitable approaches. According to the type of tasks being scheduled, the scheduling problem can be classified into two types: scheduling meta-task and scheduling a directed acyclic graph (DAG) composed of communicating tasks. In this paper, we consider meta-task scheduling problem which is to allocate a set of independent tasks from different users to a set of computing resources.

In recent years some works have been done using pure heuristics to find near-optimal solutions. These heuristics are fast, straightforward and easy to implement. Some popular and efficient pure heuristics are Sufferage [3], min-min [4], max-min [4], LJFR-SJFR [5], min-max [6], etc. Also to improve the quality of solutions, meta-heuristics have been presented for task scheduling problem. The most popular of meta-heuristic algorithms are genetic algorithm (GA) [7], simulated annealing (SA) [8], ant colony optimization (ACO) [9] and particle swarm optimization (PSO) [10]. Ritchie and Levine [11] used a hybrid ant colony optimization, Yarkhan and Dongarra [12] used simulated annealing approach, Page and Naughton [13] and Braun et al. [14], used genetic algorithm, and Abraham et al. [15] and Izakian et al. [16] used PSO for task scheduling in HC systems.

Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which is makespan. Makespan is the time when an HC system finishes the latest task. An optimal schedule will be the one that minimizes the makespan.

PSO is an algorithm that follows a collaborative population-based search model and has been applied successfully to a number of problems, including standard function optimization problems [18], solving permutation problems [19] and training multi-layer neural networks [20] and its use is rapidly increasing.

A PSO algorithm contains a swarm of particles in which each particle includes a potential solution. In contrast to evolutionary computation paradigms such as genetic algorithm, a swarm is similar to a population, while a particle is similar to an individual. The particles fly through a multidimensional search space in which the position of each particle is adjusted according to its own experience and the experience of its neighbors. PSO system combines local search methods (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation [17].

In this paper, we present a version of particle swarm optimization approach for scheduling meta-tasks in HC systems and the goal of scheduler is to minimize the makespan. In order to evaluate the performance of the proposed method, it is compared with genetic algorithm that presented in [14] for scheduling tasks in HC systems and continuous PSO that presented in [17] for task assignment problem in multiprocessor systems. The experimental results show the presented method is more efficient and can be effectively used for HC systems scheduling. The remainder of this paper is organized in the following manner. In Section 2, we formulate the problem, in Section 3 the PSO paradigm is briefly discussed, Section 4 describes the proposed method and Section 5 reports the experimental results. Finally Section 6 concludes this work.

2. Problem definition

An HC environment is composed of computing resources where these resources can be a single PC, a cluster of workstations or a supercomputer. Let $T = \{T_1, T_2, ..., T_n\}$ denotes the set of tasks that in a specific time interval is submitted to HC system. Assume the tasks are independent of each other (with no inter-task data dependencies) and preemption is not allowed (they cannot change the resource they have been assigned to). Also assume at the time of submitting these tasks, m machines $M = \{M_1, M_2, ..., M_m\}$ are within the HC environment. In this paper it is assumed that each machine uses First-Come, First-Served (FCFS) method for performing the received tasks. We assume that each machine in HC environment can estimate how much time is required to perform each task. In [14] Expected Time to Compute (ECT) matrix is used to estimate the required time for executing a task in a machine. An ETC matrix is a $n \times m$ matrix in which n is the number of tasks and m is the number of machines. One row of the ETC matrix contains the estimated execution time for a given task on each machine. Similarly one column of the ETC matrix consists of the estimated execution time of a given machine for each task. Thus, for an arbitrary task T_j and an arbitrary machine M_i, $ETC(T_j, M_i)$ is the estimated execution time of T_j on M_i. In ETC model we take the usual assumption that we know the computing capacity of each resource, an estimation or prediction of the computational needs of each task, and the load of prior work of each resource.

Assume that $C_{i,j}$ $(i \in \{1,2,...,m\}, j \in \{1,2,...,n\})$ is the completion time for performing jth task in ith machine and W_i $(i \in \{1,2,...,m\})$ is the previous workload of M_i, then Eq. (1) shows the time required for M_i to complete the tasks included in it. According to the aforementioned definition, makespan can be estimated using Eq. (2).

$$\sum_{\forall \text{ task } j \text{ allocated to machine } i} C_{ij} + W_i \tag{1}$$

$$makespan = \max\{ \sum_{\forall \text{ task } j \text{ allocated to machine} i} C_{ij} + W_i \}, \quad i \in \{1,2,...,m\} \tag{2}$$

In this paper the goal of scheduler is to minimize makespan.

3. Particle swarm optimization

Particle swarm optimization (PSO) is a population based stochastic optimization technique inspired by bird flocking and fish schooling originally designed and introduced by Kennedy and Eberhart [10] in 1995. The algorithmic flow in PSO starts with a population of particles whose positions, which represent the potential solutions for the studied problem, and velocities are randomly initialized in the search space. In each iteration, the search for optimal position is performed by updating the particle velocities and positions. Also in each iteration, the fitness value of each particle's position is determined using a fitness function. The velocity of each particle is updated using two best positions, personal best position and neighborhood best position. The personal best position, *pbest* , is the best position the particle has visited and *nbest* is the best position the particle and its neighbors have visited since the first time step. Based on the size of neighborhoods two PSO algorithms can be developed. When all of the population size of the swarm is considered as the neighbor of a particle *nbest* is called

global best (*gbest*) and if the smaller neighborhoods are defined for each particle, then *nbest* is called local best (*lbest*). *gbest* uses the star neighborhood topology and *lbest* usually uses ring neighborhood topology. There are two main differences between *gbest* and *lbest* with respect to their convergence characteristics. Due to the larger particle interconnectivity of the *gbest* PSO it converges faster than the *lbest* PSO, but *lbest* PSO is less susceptible to being trapped in local optima. A particle's velocity and position are updated as follows.

$$V_k = V_k + c_1 r_1 (pbest_k - X_k) + c_2 r_2 (nbest_k - X_k); \quad k = 1,2,...,P \tag{3}$$

$$X_k = X_k + V_k \tag{4}$$

Where c_1 and c_2 are positive constants, called acceleration coefficients which control the influence of *pbest* and *nbest* on the search process, P is the number of particles in the swarm, r_1 and r_2 are random values in range $[0, 1]$ sampled from a uniform distribution. Fig. 1 shows the pseudo-code of particle swarm optimization approach.

```
create a swarm with P particles.
initialize the position and velocity of each particle randomly.
calculate fitness value of each position.
calculate pbest and nbest for each particle.
repeat
        update velocity of each particle using Eq. (3).
        update position of each particle using Eq. (4).
        calculate fitness value of each particle.
        update pbest for each particle.
        update nbest for each particle.
until stopping condition is true;
```

Fig. 1. Pseudo-code of particle swarm optimization approach

4. Proposed PSO for task scheduling in HC systems

We propose a version of PSO by adding an heuristic. Particles need to be designed to present a sequence of tasks in available machines in HC system. Also the velocity has to be redefined.

4.1. Particles encoding

One of the key issues in designing a successful PSO algorithm is the representation step, i.e. finding a suitable mapping between problem solution and PSO particle. In this paper each particle's position is encoded in an n-dimensional search space in which n is the number of tasks to be scheduled. The value of each dimension is a natural number included in rang $[1, m]$ indicating the machine number, in which m is the number of available machines in HC system at the time of scheduling. Assume that $X_k = [X_{k1}, X_{k2}, ..., X_{kn}]$ shows the position of kth particle; X_{kj} indicates the machine where task T_j is assigned by the scheduler in this particle. Note that in this encoding method a machine number can appear more than once in a particle.

Since *pbest* and *nbest* are two positions that include the personal best position and neighborhood best position of each particle, therefore the *pbest* and *nbest* encoding is similar to the particle's position. Also in this paper we used start topology for *nbest* (*gbest* PSO).

In our proposed method, velocity of each particle is considered as an $m \times n$ matrix whose elements are real numbers in range $[1, V_{\max}]$. Formally if V_k is the velocity matrix of kth particle, then:

$$V_{kij} \in [1, V_{\max}] \quad (\forall i, j), i \in \{1,2,...m\}, j \in \{1,2,...,n\} \tag{5}$$

4.2 Updating particles

In our proposed method similar to classic PSO, at first the particle's velocity is updated and then it is used for updating the particles' position. Fig. 2 shows the pseudo-code for updating velocity matrix for particle k. In this figure c_1 and c_2 are acceleration coefficients, r_1 and r_2 are random values in range $[0, 1]$ sampled from a uniform distribution and X_k is the position of particle k.

```
for each task j=1,2,...,n do
    if X_kj ≠ pbest_kj then
        V_k(X_kj)j = V_k(X_kj)j − c1r1;
        V_k(pbest_kj)j = V_k(pbest_kj)j + c1r1;
    end

    if X_kj ≠ nbest_kj then
        V_k(X_kj)j = V_k(X_kj)j − c2r2;
        V_k(nbest_kj)j = V_k(nbest_kj)j + c2r2;
    end
end
```

Fig 2. Velocity updating

For updating particle's position we use the updated velocity matrix and a heuristic, η which adds an explicit bias towards the most attractive solutions and is a problem-dependent function. In our proposed method for updating a particle's position, for each task, the probability of its performing on various machines is calculated according to the Eq. (6).

$$p_{kij} = \frac{V_{kij} \times [\eta_{kij}]^{\beta}}{\sum_{l=1,2,...m} V_{klj} \times [\eta_{klj}]^{\beta}} \quad (6)$$

Where p_{kij} is the probability of performing task T_j on machine M_i in particle k, and η_{kij} represents a priori effectiveness of performing task T_j on machine M_i in particle k. Since in this paper we aim at minimizing makespan, the η_{kij} obtains using Eq. (7).

$$\eta_{kij} = \left(\frac{1}{CT_{kij}} \right) \quad (7)$$

In which CT_{kij} is the completion time of task T_j on machine M_i in particle k and can be obtained according to the workload of machine M_i plus required time for executing task T_j on machine M_i.

After obtaining the p_{kij} , $\forall\, i = 1,2,...m$, we can select a machine for task T_j in particle k according to Eq. (8).

$$M_i \leftarrow \begin{cases} \arg\max_{l=1,2,..m} p_{klj} & \text{if } r \le r_0 \\ \textit{roulette wheel selection}, & \text{otherwise} \end{cases} \quad (8)$$

In Eq. (8) $r_0 \in [0,1]$ is a user specified parameter and r is a random number in range (0,1) sampled from uniform distribution.

4.3. Fitness evaluation

Since in this paper the makespan is used to evaluate the performance of scheduler, the Fitness value of each solution can be estimated using Eq. (9).

$$\text{fitness} = \frac{1}{makespan} \quad (9)$$

5. Simulation and Experimental Results

In order to evaluate the performance of the proposed method, it is compared with genetic algorithm that presented in [14] for scheduling tasks in HC systems and continuous PSO that presented in [17] for task assignment problem in multiprocessor systems. The goal of scheduler in these methods is minimizing makespan the same as our proposed method. These methods are implemented using VC++ as well as ours and run on a Pentium IV 3.2 GHz PC. In order to optimize the performance of the proposed method and proposed PSO in [17] and GA in [14], fine tuning has been performed and best values for their parameters are selected. Based on experimental results the proposed PSO algorithm performs best under the following settings: $c_1 = c_2 = 2.0$, $P = 50$, $V_{\max} = 40$, $\beta = 1.0$, $r_0 = 0.8$. Also we used the benchmark that proposed in [14] for simulating the HC environment.

The simulation model in [14] is based on expected time to compute (ETC) matrix for 512 tasks and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the three following metrics: task heterogeneity, machine heterogeneity, and consistency. In ETC matrix, the amount of variance among the execution times of tasks for a given machine is defined as task heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Also an ETC matrix is said to be consistent whenever a machine M_i executes any task T_j faster than machine M_k; in this case, machine M_i executes all tasks faster than machine M_k. In contrast, inconsistent matrices characterize the situation where machine M_i may be faster than machine M_k for some tasks and slower for others. Partially-consistent matrices are inconsistent matrices that include a consistent sub-matrix of a predefined size [14].

Instances consist of 512 tasks and 16 machines and are labeled as u-x-yy-zz as follow:

- u means uniform distribution used in generating the matrices.
- x shows the type of inconsistency; c means consistent, i means inconsistent, and p means partially-consistent.
- yy indicates the heterogeneity of the tasks; hi means high and lo means low.
- zz represents the heterogeneity of the machines; hi means high and lo means low.

In our experiment the initial population for compared methods is generated using two scenarios: (a) randomly generated particles from a uniform distribution, and (b) one particle using the Min-min heuristic (that can achieve a very good reduction in makespan [6, 14]) and

the others are random solutions. The statistical results of over 50 independent runs are compared in Table 1 for scenario (a). In these table the first column indicates the instance name, the second, third, and fourth columns indicate the makespan achieved by GA[14], PSO[17] and our proposed method respectively.

Table 1. Comparison of statistical results between GA [14], PSO [17] and our proposed method for scenario (a)

Instance	GA[14]	PSO[17]	Proposed method
u-c-hi-hi	21508486	13559696	**10173411**
u-c-hi-lo	236653	223008	**191878**
u-c-lo-hi	695320	463241	**371355**
u-c-lo-lo	8021	7684	**6379**
u-i-hi-hi	21032954	23114941	**6642987**
u-i-hi-lo	245107	286339	**149997**
u-i-lo-hi	693461	849702	**228971**
u-i-lo-lo	8281	9597	**4496**
u-p-hi-hi	21249982	22073358	**8325090**
u-p-hi-lo	242258	266825	**162601**
u-p-lo-hi	712203	772882	**293335**
u-p-lo-lo	8233	8647	**5213**

Table 2. Comparison of statistical results between our proposed method and others in scenario (b)

Instance	Min-Min	GA[14]	PSO[17]	Proposed method
u-c-hi-hi	8145395	7892199	7867899	**7796844**
u-c-hi-lo	164490	161634	161437	**160639**
u-c-lo-hi	279651	276489	274636	**266747**
u-c-lo-lo	5468	**5292**	5322	5309
u-i-hi-hi	3573987	3496209	3560537	**3220459**
u-i-hi-lo	82936	81715	81915	**80754**
u-i-lo-hi	113944	112703	113171	**108597**
u-i-lo-lo	2734	**2636**	2680	2644
u-p-hi-hi	4701249	4571336	4580666	**4462357**
u-p-hi-lo	106322	104854	104987	**103794**
u-p-lo-hi	157307	153970	154933	**150375**
u-p-lo-lo	3599	**3449**	3473	3461

As shown Table 1, the proposed PSO approach achieved best results in all instances. Also our method has a large amount of reduction in makespan in all instances; this is because of using heuristic η in our method that can minimize makespan efficiently.

Table 2 shows the statistical results of over 50 independent runs in scenario (b). As shown in this table the min-min heuristic can obtain a good reduction in makespan. In this scenario our method surpasses others in most instances except the instances with low heterogeneity in tasks and machines. Figure 3 shows a comparison of CPU time required to achieve results between compared methods. It is evident that our method needs lowest time for convergence in most cases but by increasing the number of tasks and problem search space the time for achieve results is more increased in PSO and our method rather than GA and in case the number of tasks is 1024, the GA scheduler needs lowest time for convergence.

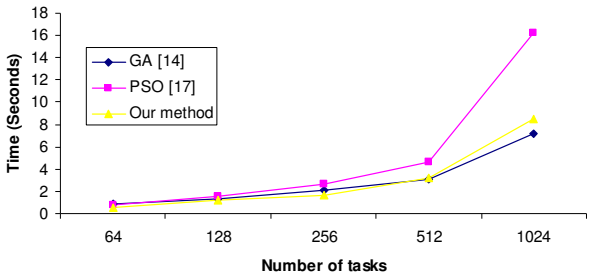

Fig 3. Comparison of convergence time between our proposed and others

6. Conclusions

To exploit the different capabilities of a suite of heterogeneous resources effectively and satisfy users with high expectations for their applications, a crucial problem that needs to be solved in the framework of HC is the scheduling problem. In this paper, we combined particle swarm optimization approach with heuristic for scheduling tasks in distributed heterogeneous systems to minimize makespan. The performance of the proposed method was compared with the GA and continuous PSO through carrying out exhaustive simulation tests and different settings. Experimental results show that our method surpasses others in most cases.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, Internat. J. Supercomput. Appl. 15 (3) (2001) 3–23.

[2] D. Fernandez-Baca, "Allocating modules to processors in a distributed system", IEEE Trans. Software Engrg. 15, 11 (Nov. 1989), pp. 1427-1436.

[3] M. Macheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, "Dynamic mapping of a class of independent tasks onto

heterogeneous computing systems", J. Parallel Distribut. Comput. 59 (2) (1999) 107–131.

[4] R. F. Freund et al, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet", In: 7th IEEE Heterogeneous Computing Workshop (HCW 98), 1998, pp. 184-199.

[5] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids", In: The 8th IEEE International Conference on Advanced Computing and Communications, India, ISBN 0070435480, Tata McGraw-Hill Publishing Co. Ltd, New Delhi, India, pp. 45-52, 2000.

[6] H. Izakian, A. Abraham, V. Snášel, "Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments", IEEE International Workshop on HPC and Grid Applications, 2009.

[7] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1997.

[8] S. Kirkpatrick, C. Gelatt Jr., M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[9] M. Dorigo, Optimization, learning, and natural algorithms, Ph.D. Thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Italy, 1992.

[10] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks (1995) 1942–1948.

[11] G. Ritchie and J. Levine, "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments", In: 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.

[12] A. Yarkhan and J. Dongarra, "Experiments with scheduling using simulated annealing in a grid environment", In: 3rd International Workshop on Grid Computing (GRID2002), 2002, pp. 232–242.

[13] J. Page and J. Naughton, "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms", Artificial Intelligence Review, 2005 pp. 415–429.

[14] H.J. Braun et al, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems" Journal of Parallel and Distributed Computing, 61(6), 2001.

[15] A. Abraham, H. Liu, W. Zhang, TG. Chang, "Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm, pp. 500–507. Springer, Heidelberg, 2006.

[16] H. Izakian, B. Tork Ladani, K. Zamanifar, A. Abraham, "A Novel Particle Swarm Optimization Approach for Grid Job Scheduling", pp. 100-110, Springer, Heidelberg, 2009.

[17] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocessors and Microsystems 26 (2002) 363–371.

[18] P.J. Angeline, Evolutionary Optimization versus Particle Swarm Optimization: Philosophy and Performance Differences, In: Proceedings of the Seventh Annual Conference on Evolutionary Programming (1998) 601–610.

[19] J. Salerno, Using the Particle Swarm Optimization Technique to Train a Recurrent Neural Model, In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (1997) 45–49.

[20] R.C. Eberhart, Y. Shi, Evolving Artificial Neural Networks, In: Proceedings of the International Conference on Neural Networks and Brain (1998) pages PL5–PL13.