Synthetic Loads Analysis of Directed Acyclic Graphs for Scheduling Tasks

Apolinar Velarde Martinez
Instituto Tecnológico el Llano Aguascalientes
Aguascalientes, México

***Abstract*—Graphs are structures used in different areas of scientific research, for the ease they have to represent different models of real life. There is a great variety of algorithms that build graphs with very dissimilar characteristics and types that model the relationships between the objects of the problem to solve. To model the relationships, characteristics such as depth, width and density of the graph are used in the directed acyclic graphs (DAGs) to find the solution to the objective problem. These characteristics are rarely analyzed and taken into account before being used in the approach of a solution. In this work, we present a set of methods for the random generation of DAGs. DAGs are produced with three of these methods representing three synthetic loads. Each of the three above characteristics is evaluated and analyzed in each of DAGs. The generation and evaluation of synthetic loads is with the objective of predicting the behavior of each DAG, based on its characteristics, in a scheduling algorithm and assignment of parallel tasks in a distributed heterogeneous computing system (DHCS).**

Keywords*—Directed acyclic graph; distributed heterogeneous computing system (DHCS); Algorithm for scheduling and allocation tasks in a DHCS; parallel tasks

I. Introduction

In recent years there has been growing interest in scientific applications of workflow, which are often modelled by Directed Acyclic Graphs (DAGs) [1]. This type of graphs, which do not have cycles in parallel connections, allow to represent the relations, the interdependencies and the characteristics between the objects, such as kinship relations between people, the structures of web page designs, basic data structures of applications for information visualisation [2], as well as modelling tool in various fields (social sciences, computer science and biology) [1], and in the representation of parallel programs, which are modelled in the scheduling of tasks in Distributed Heterogeneous Computing System (DHCS) [1], [3]-[6].

In the parallel and distributed computing systems, the DAGs, also allow to show the dependencies among the tasks, the precedence restrictions, the communication links, the calculation costs and the communication costs of the tasks that constitute the application to be executed within the system [1], [7], [8]. In addition to the above, the DAGs are used to distribute and represent the different tasks that make up the solution of a problem, and the way in which these tasks are distributed among the processors, by means of the algorithms of the scheduling and allocation of tasks [7].

The algorithms of the scheduling and the allocation of tasks, seek to optimize one or more performance parameters such as the Makespan [9], the maximization of the resources of the system [10], [20], the waiting time [21], etc. by means of techniques derived from operations research, evolutionary algorithms, heuristic techniques and other methods of optimization using operations research techniques, heuristics and meta-heuristics.

To carry out the tests of the scheduling algorithms in the parallel and distributed computing systems, the DAGs are generated in two ways: by generating graphs of real applications, and by the random generation of synthetic loads [4].

For the generation of graphs of real applications, the designs of DAGs of real problems are considered, such as the molecular dynamic code [6], [11], the Gaussian elimination, and the Fast Fourier Transform [11], and so on. These applications are processed with the scheduling algorithm to obtain the results in the performance metrics to be evaluated.

For the generation of synthetic loads constituted by DAGs, different algorithms of random generation of graphs can be used, such as: the Erdös-Rényi method [12], the level-by-level method [13], the fan-in fan-out method [14], the Random Orders method [15], the Márkov chain method [7], [16], the parallel approach method for the random generation of graphs on GPUs [17], and so on. These algorithms use a set of parameters in their execution, to generate the DAGs with specific characteristics or properties.

The different research works specify the characteristics and properties of the DAGs in different ways. In [18], the following properties of the DAGs are mentioned and defined: the depth of the graph, the width, the regularity, the density and the number of jumps. In [4], three characteristics of the DAGs are analyzed: the longest route, the distribution of the out-degree (degree of exit) and the number of edges. In [19] author refers to the following parameters of the DAGs: the critical path and the size of the DAG.

These characteristics of the DAGs are decisive in the results of the performance metrics measured by the task scheduling algorithm that uses synthetic loads, so generating synthetic loads to perform the tests of the scheduling algorithms in distributed heterogeneous systems produce two important problems: first, there is no standard algorithm that produces the graphs with specific properties that you wish to evaluate, in the scheduling algorithms; the generated synthetic loads produce evaluations in the scheduling algorithms that are generally adapted to the performance parameters that the algorithm evaluates, producing excellent results with synthetic loads, but with poor results with loads of real users. Second, the graphs do not adapt to the underlying computation system in which the scheduling

algorithms are tested, so it is necessary to execute different graph generation algorithms to produce synthetic loads, that are evaluated with the scheduling algorithm and, with the distributed heterogeneous system.

As explained in the previous paragraph, this paper describes five methods for the generation of random DAGs. Of these five methods, three of them are chosen: the Erdös-Rényi method, the Márkov chain method and the parallel approach method, with the aim of generating synthetic loads that will be analyzed in the characteristics of depth, width and the density of the graph to predict the behavior of a scheduling algorithm and allocation of tasks in a distributed heterogeneous computing system.

The selection of the three methods of generation of random graphs aims to:

1) Determine and analyze the values that each method generates in the characteristics of the depth, width and density of the graph.
2) Evaluate the convergence speed of each algorithm, considering that two of them run sequentially and the last one in parallel.
3) Obtain synthetic loads, which allow the validation of an algorithm for scheduling and allocation tasks in a distributed heterogeneous computing system.

The manuscript is constituted in following ways: In Section 2 related works, the most common algorithms for the generation of DAGs reported in the literature are described, Section 3 justifies the random generation of DAGs for the construction of synthetic loads to be used in the scheduling and allocation of tasks in SCHD. A set of basic definitions of the DAGs is shown in Section 4. Section 5 describes the way in which parallel tasks are modeled with DAGs. The application of the DAGs in the problem of scheduling and allocation tasks in the DHCS is described in Section 6. Section 7 describes the importance of the characteristics of the generated DAGs. Section 8 describes the results obtained from the evaluation of the two methods of generation of DAGs. Section 9 shows the conclusions obtained in this work, and finally Section 10, shows the works that will be developed in the future in this research area.

II. Related Works

In the problem of task scheduling in heterogeneous distributed systems, different methods have been used for the generation of random graphs. In this section, we describe five of the most common methods that generate random graphs.

The algorithms described here generate the graphs based on a set of parameters that are put into the algorithm. The characteristics of each generated graph are similar to those of the graphs of parallel applications of the real world.

The Erdös-Rényi [12], are two simple, elegant and general mathematical models [17], considered the most popular methods for the random generation of graphs [4]. The first model denoted as $\Gamma_{v,e}$ choosing a uniformly random graph of the set of graphs with v vertices and e edges. The main characteristic of this method is the generation of random graphs, with a fixed number of edges [4].

The second method denoted as $\Gamma_{v,p}$ choose a uniformly random graph of the set of graphs with v vertices, where each edge has the same p probability of existing [12]. From this method, the following properties can be highlighted [4]:

- When the value of v is sufficiently large, the number of edges in the graphs generated tends to $p\binom{n}{2}$.
- There is a high probability of generating a subgraph weakly connected to most vertices, if np tends to a constant greater than 1 and there is no other component connected with more than $Olog(n)$ nodes.
- If $p > \frac{(1+\epsilon)\, ln\, n}{n}$ then it is highly probable that the generated graph will not have isolated vertices.

To date this model has been widely used in many fields of research, among which are: communication engineering, social networks, the spread of viruses and worms in networks, data search and replication in point to point networks, evaluate the similarity of the topologies between biological regulation networks and monotonous systems, and used to study genetic variation in human social networks, among other areas [17].

The level by level method designed specifically for the validation of heuristics that are applied in task scheduling [13]. This method is based on the concept of levels. This concept states that if there is an edge from level a to level b, then there is no path from a vertex in b to a vertex in a the edges are created with probability p exactly as in the Erdös-Rényi $\Gamma_{v,p}$ method. The practical utility of this method is due to the possibility of limiting the size of the critical path of the graph, when the value of the variable k in the algorithm is limited.

The Fan-in/Fan-out method, proposed in [14] uses properties of the branch of mathematics called order theory to analyse and generate random graphs. The operation is based on the generation of randomly ordered sets, which are used to generate task graphs. The fundamental concept of the method is to create a partial order by the intersection of several total orders.

The Random Orders method, proposed in [15], uses properties of the branch of mathematics called theory of order to analyze and generate random graphs. Its operation is based on the generation of partially ordered random sets, which are used to generate task graphs. The fundamental concept of the method is based on creating a partial order by the intersection of several total orders.

The Márkov Chain Method is based on a Márkov chain to generate even random acyclic digraphs of a given size [16]. This method, initially proposed for information visualisation applications, seeks to produce random acyclic digraphs with:

- A prescribed number of vertices uniformly random starting from the empty graph.
- Dimensioned total degree or dimensioned vertex degree.
- A way to control the density of the edges of the resulting graphs.

This algorithm uses the following development: let $V = \{1, ..., n\}$ denote the set of underlying vertices of the considered graph. We define a Márkov chain M with state space of all acyclic digraphs on the set of vertices V A Márkov chain

is completely determined by its transition function, prescribing the probability that the chain goes from a given state to any other possible state. For this case the transition function is as follows:

One position consists of an ordered pair (i, j) of different vertices of V. If X_t denotes the state of the Márkov chain over time t, then T_{t+1} is selected in accordance with rules 1) and 2) described below.

Suppose a position (i, j) which is selected uniformly at random:

1) If the position (i, j) corresponds to an arc e in X_t, then $X_t \backslash e$. The edge e is deleted from the graph associated with X_t.
2) If the position (i, j) does not correspond to an arc e in X_t, then X_{t+1} is obtained from X_t when adding this arc, as long as the underlying graph remains acyclic; otherwise $X_{t+1} = X_t$.

The algorithm obtains the main characteristics of the Márkov chain: aperiodic and irreducible with a symmetric transition matrix, containing a uniform, limiting stationary distribution in the set of all acyclic digraphs on the set of vertices of V.

A proposal to improve the Márkov chain algorithm has been proposed in [7]. In this research work the algorithm is slightly modified and used to generate acyclic digraphs simply connected evenly at random. This type of digraph is widely used in task scheduling, due to the ease of representing parallel programs that are modelled. For this ease, this improved algorithm of the Márkov chain is selected in this research work.

This algorithm consists of two rules T1 and T2, which appear in the following paragraphs:

Let $N \geq 2$ is a fixed integer, and $V = \{1, ..., n\}$ denotes a finite set of vertices. Consider the set A of all acyclic directed graphs on V that is, graphs that do not contain circuits. Next, we define the Márkov chain M on the set A. Because the V set of vertices is fixed, there is no distinction between a digraph in A and the set of its arcs. The transition in any two states in M is given as follows:

X_i is the state of the Márkov chain in time t. Assume a pair of integers (i, j) that have been uniformly drawn at random from the set $V \times V$.

- Rule (T1). If (i, j) is an arc in X_t this is deleted from X_t. This is $X_{t+1} = X_t \backslash (i, j)$.
- Rule (T2). If (i, j) is not an arc in X_t then:
  - This is added to X_t if the resulting graph is acyclic. This is, $X_{t+1} = X_t \cup (i, j)$.
  - In another case nothing is done, this is, $X_{t+1} = X_t$.

When starting the algorithm from a graph with an empty array of arcs, it is possible to apply the rules (T1) and (T2) iteratively to construct an acyclic digraph with a nearly uniform distribution.

The characteristics of the algorithm demonstrated by the authors are:

- The probability of a transition going from a state X to a state $Y \neq X$ is $\frac{1}{n^2}$.
- The generation of the transition matrix as symmetric.
- The convergence of the Markov chain to uniform distribution.
- The irreducibility of the state of space M.

A parallel approach to the random generation of graphs on GPUs is a method proposed in [17], which seeks to solve the problem of the exponential growth of the number of edges in the process of classical generation of graphs with the Erdös-Rényi method.

The general scheme of this research is based on a collection of three sequential algorithms as follows: the first algorithm called ER is the implementation of the random process of the Gilbert model [12]; the second ZER algorithm exploits the availability of an analytical formula for the expected number of edges in the generated graphs, which can be omitted in a geometric approach; A third algorithm, PreLogZER, is implemented to avoid the calculation of logarithms required by the proposed method. The three sequential algorithms are scaled to a parallel format, which is programmed in a GPU environment.

This algorithm was proposed by the authors to be evaluated in a GPU hardware architecture. In our work, the algorithm is evaluated in an architecture of four processing cores using the MPI libraries of the C language.

III. Justification of the Random Generation of Graphs

In the absence of a procedure for generation of standard random graphs, it is necessary to carry out experiments with different scheduling algorithms, to generate the synthetic loads that will be evaluated with the new scheduling algorithm to be executed in the DHCS.

The random generation of DAGs allows the use of different types of graphs that resemble the designs of the real parallel programs, which causes the values obtained in the performance metrics evaluated to be most attached to the workloads generated by the real users in the DHCS.

Then, the generation of random workloads, to validate a new scheduling algorithm is justifiable because:

- It can help to find a counterexample for the algorithm [4]. Although the algorithm is theoretically correct, the random input data can help find errors in the implementation, or help identify bottlenecks in performance.
- It helps to evaluate the performance of the algorithm in contexts not analyzed theoretically [4]. It allows to predict how the algorithm will be executed in real conditions.
- They allow predicting some of the properties of acyclic graphs [16].
- It is possible to obtain DAG's evenly distributed with bounded total grade or bounded vertex degree [16].

- It is likely to obtain, in a large set of examples, all possible or interesting cases that should be tested or studied [7].
- They allow to synthesize data sets with the objective of evaluating the efficiency and effectiveness of the algorithms [17].

IV. Basic Definitions

A graph is a structure that represents relationships and interdependencies between objects, and the characteristics that relate them. Examples of graph applications can be kinship relationships between people, the structure of web page design, basic data structures of applications for information visualization [16], as well as modeling tool in various fields (social sciences, computer science and biology) [7], and in the representation of parallel programs that are modeled in task scheduling in high performance computing systems (HPCS, High Performance Computing System) [18].

Given the variety of graphs in the existing literature [2], in this section we define a special type of graph called acyclic weighted directed graph (which we refer to in the following sections as DAG), using the following definitions.

Definition 1. A graph G is a pair $G = (V, E)$ consisting of a finite set $V \neq \emptyset$ and a set E of two subset elements of V. The elements of V are called vertices. An element $e = \{a, b\}$ of E is called an edge with final vertices a and b. It is said that a and b are incidents with e and that a and b are adjacent or neighbours of each other, and is defined as $e = ab$ or $a\underline{e}b$

Definition 2. To determine the relationship between vertex information (which the connections do not model), a digraph is defined. A digraph exists when the set of connections $A = A(G)$ is directed, they distinguish between the connections $e_{i,j} = (v_i, v_j)$ and $e_{j,i} = (v_j, v_i)$, then the graph $D = (V, A)$ is called directed graph or graph.

Definition 3. Now, if between the existing connections, the digraph has related a number $T(v_i, v_j)$ which represents the cost of communication between the vertex v_i and vertex v_j we have a weighted graph. A weighted graph is a pair (G, W), where G is a graph and W represents a function $W : E \to \mathbb{R}$ in this way, the weight of a connection e is $W(e)$ The weight of the graph is $W(G) = \sum_{e \epsilon E} W(e)$.

Definition 4. Finally, a graph that has no cycles in parallel connections ie it has no connections of the form: e_{v_i,v_i} is called an acyclic graph.

V. Modelling of Parallel Tasks with Graphs

As heterogeneous distributed computing systems (e.g. clusters, grids, clouds, etc.) become commonplace to meet the massive computational demands of executing complex, multitasking scientific applications, the process of assigning these tasks to multiple resources, known as scheduling, is important for application performance.

In recent years, DAGs have received much attention as a result of the growing interest in modelling scientific workflow applications [1].

This modelling, in the DHCS allows to show:

- the dependencies between tasks [8],
- the transmission of data between tasks [1],
- the precedence constraints between task [4,8],
- the communication links between tasks,
- the costs of calculating each of the tasks [8], and
- the costs of communication between tasks [8].

VI. Application of DAGs in the Task Scheduling Problem in DHCS

Without loss of generality and considering the definitions existing in the literature [1], [3]-[6] in this section, it is defined how the DAGs are applied to the problem of the scheduling and the allocation of tasks in a DHCS.

A DAG consists of v nodes $n_1, n_2, \ldots, n_v$ which can be executed on any of the processors available from an DHCS. A node in the DAG represents a task, which is a set of instructions that must be executed sequentially without preferential right on the same processor. A node has one or more entries. When all the entries are available, the node is activated for execution. After its execution, it generates its outputs. A parentless node is called an input node, and a childless node is called an output node. The weight at a node is called the computing cost of a node n_i and is denoted by $w(n_i)$.

The graph also has e directed edges representing a partial order between tasks. The partial order introduces a DAG precedence constraint, and implies that if $n_i \to n_j$, then n_j is a child who can not start until his father n_i complete and send your data to n_i. The weight at one edge is called the communication cost of the edge and is denoted by $c(n_i, n_j)$. This cost is incurred if, n_i and n_j are scheduled in different processors and is considered zero, if n_i and n_j are scheduled on the same processor. For standardisation we specify that a DAG has only a single input node and a single output node.

VII. Characteristics of the Generated DAGs

When experimenting with a new algorithm for scheduling and allocation tasks, it is necessary to carefully observe each of the parameters that constitute the DAGs. The parameters that are observed allow us to avoid biases in the results obtained in the new algorithms, when testing: convergence, speed, the capacity of resource allocation and transfer speeds.

Some of the works that highlight the importance of the parameters of the DAGs are [3], [4], [18], [19]. In this section, three characteristics of the DAGs that are analyzed in synthetic loads are defined.

The depth of the graph, also known as the critical path or the longest path, is the path from the input node to the output node of the DAG, and has the highest values in the total calculation of execution costs of each task, and the total communication costs of the edges [11], [19]. When parallel tasks are scheduled using the DAGs, the algorithms require an appropriate scheduling of tasks located in the critical path.

The width of the DAG, which determines the maximum parallelism in the DAG, that is, the number of tasks in the longest level.

The density of the graph, denotes the number of edges between two levels of the DAG; with a low value in this property, there are few edges and with large values there are many edges in the DAG.

VIII. Simulation and Results

This section explains the procedure carried out to perform the simulation of the experiments and the results obtained.

The first subsection explains the parallel algorithm for the generation of the synthetic loads; in second section, the specifications of the way, in which the DAGs are used in this work are described; finally the results are depicted in three subsections: number of nodes generated (on average) in the critical path, the maximum parallelism in the DAG and the density of the graph.

We describe an additional feature of the methods: the convergence time of the algorithms. This measurement allows us to know, the times that each method consumes in the generation of synthetic loads. The results obtained are shown in a graph for better understanding.

A. Algorithm for the Generation of Synthetic Loads

For the realization of the simulation, an algorithm that runs on a platform of four cores was used. The division of work between the cores is done in the following way:

Core 0

1) Build the symmetric matrix with the number of vertices specified and the probability of inclusion, for the Erdös-Rényi method. Send all parameters to the core 1, to build the DAG with this method.
2) Specifies the maximum number of edges and the probability of inclusion. Send all parameters to the core 2, to build the DAG with this method.
3) Set v representing the set of vertices. Send all parameters to the core 3, to build the DAG with this method.
4) Receives the results of the cores and shows them to the user.

Core 1

1) Process the Erdös-Rényi algorithm. Receives the symmetric matrix.
2) Build the DAG based on the symmetric matrix.
3) Go through all the possible routes in the DAG to get: the depth of the graph that represents the critical route, the width of the DAG and the density of the graph.
4) Measures the time consumed by the algorithm.

Core 2

1) Process the Parallel approach algorithm. Receives both parameters: maximum number of edges and the probability of inclusion.
2) Build the DAG based on this parameters.
3) Go through all the possible routes in the DAG to get: the depth of the graph that represents the critical route, the width of the DAG and the density of the graph.
4) Measures the time consumed by the algorithm.

Core 3

1) Process the Markov algorithm. Receive the set of vertices and build the DAG.
2) Go through all the possible routes in the DAG to get: the depth of the graph that represents the critical route, the width of the DAG and the density of the graph.
3) Measures the time consumed by the algorithm.

The details of each step, within the processing cores are omitted in this research work for reasons of space; and instead, the results obtained in each experiment are highlighted.

B. Specifications

For a better understanding in the treatment of the DAGs, it is necessary to dictate the following specifications:

- There must be a dummy vertice of entry into the DAG, that indicates the input of the parallel program.
- There must be an exit node that indicates the end of the parallel program to be put into the system.
- There must be at least one route that leads from the input node to the exit node.
- There may be vertices without a link to the main node in the DAG, which represent processes that start after the main program has started.

In this work, the parameters of the algorithm are not compared, such as the probability of inclusion; in any case, these parameters remain fixed for the three methods and, once the DAGs are produced the three proposed parameters are observed, measured and analyzed.

C. Number of Nodes Generated (on Average) in the Critical Path

The objective of this experimentation is to verify the number of vertices generated by each method, in the critical route of the DAG. Obtaining a large number of vertices in the critical path, indicates that the costs of executing each task will be high: the task will remain in the system for a long time and will require a high number of system resources.

The parameters used in this experiment are: N and M. The parameter N stands for the finite set of vertices and takes the values of 5 to 20, and M stands for the number of nodes generated (on average) in the critical path.

What results does each method give? Fig. 1 shows the results for this experiment. For the case of the Erdös-Rényi method, there is a direct relationship between the number of vertices of the critical route and, the parameter of the number of vertices to be generated with the algorithm; this relationship allows generating DAGs with short critical routes, which represent a low consumption of processing resources in the HDCS.

The Markov chain method produces longer critical routes, due to the increase in the number of DAG levels. By producing more levels in the DAG, the Markov method also produces an

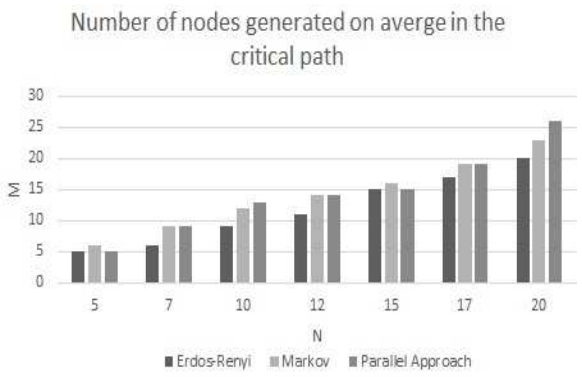

Fig. 1. Number of nodes generated (on average) in the critical path.

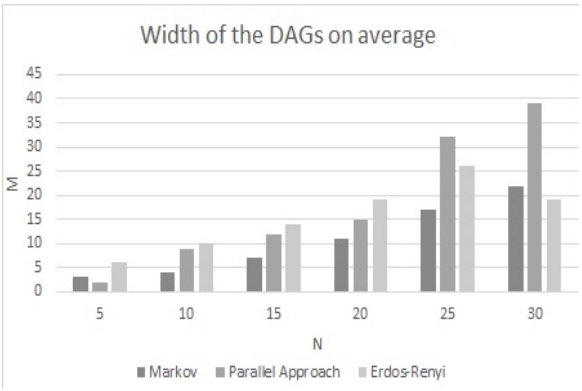

Fig. 2. The maximum parallelism in the DAG: width.

increase in the number of edges. These results allows to test the DAG with heavier loads in the HDCS.

The third method, the parallel approach, can produce critical routes that oscillate between the number of vertices generated and a greater number of vertices. Namely, it is a method that can be used to generate light synthetic loads and heavy synthetic loads, in the same set of tests of the scheduling algorithm.

D. The Maximum Parallelism in the DAG: Width

In general terms, the width determines the maximum parallelism in the DAG, this is the number of tasks in the longest level.

To generate a comparison between the 3 proposed methods, the parameter N was varied with the same values: from 10 to 30 with increments of 5 units.

For the the Markov chain algorithm, the parameters m and p, remain constant during the tests performed. The parameter E is considered a maximum number of edges as N^2; m representing the break points of the intervals, E the maximum number of edges and p the probability of inclusion.

The parameter N (the finite set of vertices) takes the values of 10, 20, 30. Thus the probability of a transition going from a state X to a state $Y \neq X$ is given by the formula $\frac{1}{N^2}$.

The width generated in the DAG with this algorithm occurs at the highest levels, i.e. if the graph is produced with 5 levels the maximum width is reached at level 1 or 2, which implies that the assignment and release of the resources in the distributed heterogeneous system are made at an early stage of the execution of the algorithm. The results obtained with this method and, according to the generated variations are shown in Fig. 2.

Given the random nature of the DAG generation, it is very difficult to determine the behaviour of the algorithm under different conditions, but it allows us to determine the resource usage times.

The parallel approach uses the following parameters: m representing the break points of the intervals, E the maximum number of edges and p the probability of inclusion.

The results obtained according to the generated variations are shown in Fig. 2.

The results obtained with this method show a thinning in the width of the DAG, what can be interpreted as the generation of parallel applications with less load of resources for the algorithm that realises the scheduling.

A very important point in this algorithm is the number of edges produced in the DAG. While the previous method generates few edges, in this method it is observable that at each level a substantial number of these are produced, which indicates the parallel applications that are represented contain high indexes of communication between them, allowing to evaluate the means of distributed system communication.

In this set of experimentations, the last method Erdös-Rényi, shows consistent results as shown in Fig. 2. The levels that occur in the DAG, maintain a strict attachment to the number of vertices that are generated, i.e., if 5 or 10 vertices are generated, at least one level in the DAG is produced with this same number of vertices. The consistency of this method makes it suitable for experimenting with light loads in the planning algorithm.

E. The Density of the Graph

We account for each level the number of edges, to determine the communication levels produced by the generated DAG.

According to the number of localized edges, we have classified connectivity levels as low, dense and very dense. A low level refers to the existence of a connectivity of 50% or less of the nodes between one level and another, that is, if level 1 has 6 vertices and level 2 has 2 vertices, a low connectivity refers to that 3 or less vertices of level 1 are connected with the 2 vertices of level 2. High connectivity refers to 80% or less of the vertices of level 1 are connected to the vertices of level 2. A dense communication level refers to there is a total connectivity between the vertices of a level and the vertices of its immediate lower level.

The parameters used in this experiment are: N and M. The parameter N stands for the finite set of vertices and takes the values of 5 to 20, and M stands for the percentage of edges, generated between DAG levels.

Fig. 3 shows the results obtained in the density of graphs. This experiment show that the Markov chain method stand out above the other two methods, by generating high connectivity;

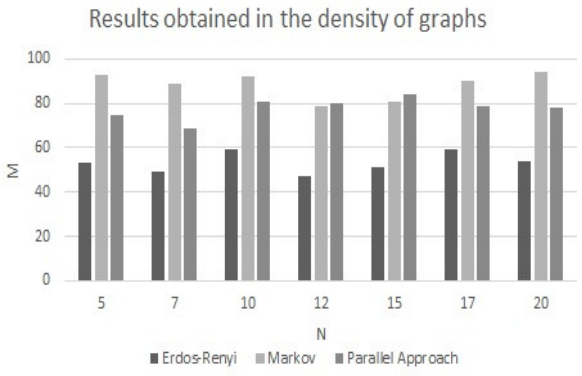

Fig. 3. Results obtained in the density of graphs.

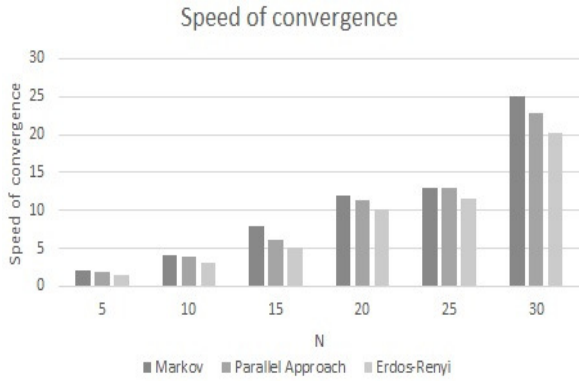

Fig. 4. Convergence time of methods.

with this method the number of connections between all levels of the DAG is very high.

The parallel approach, has an oscillation in the results and exhibits good scalability as the experiment progresses; it can produce dense connectivities and very dense connectivities. is considered in this work as a dual method, which facilitates experimenting with high and low levels of coefficient in the system.

The Erdös-Rényi method holds a stability in the connectivities produced in the DAG. During all the experimentation, the levels remain dense. These results are favorable for experimenting with stable loads in the planning algorithm.

The levels of connectivity of the graph, have been characterized to allow the allocation algorithm to have an “idea" of the location of the tasks in the HDCS. That is, a low connectivity allows the DAG’s tasks to be located in geographically remote computing resources. Whereas a dense connectivity, forces the algorithm to assign tasks as closely as possible.

F. Convergence Time of the Algorithms

Another characteristic observed in the experiments has been the convergence time of the algorithms proposed, i.e. the time that the algorithm needs to generate all the graphs.

The Markov chain method, due to its condition of being a sequential algorithm, its total time for the completion of the generation of total graphs is slightly higher, than the time needed for the parallel approach for random graph method, which was born with a condition of being a parallel algorithm. In summary form in Fig. 4 shows the times of convergence consumed, by each one of the methods.

The Markov chain method is used to represent parallel applications that require a large number of computational system resources, which will be used in the early stages of the algorithm; whereas, the second method facilitates the generation of DAGs with high communication requirements in the system.

The Erdös-Rényi method, accelerates convergence due to the simplicity of the method. In the results obtained, it produces the synthetic loads more quickly.

IX. Conclusions

The synthetic loads used for the evaluation of scheduling algorithms and assignment of tasks in HDCS, are generated by sophisticated methods that do not consider the characteristics of the DAGs in the scheduling process.

Evaluating an algorithm that discriminates or does not consider the properties of the DAGs, can produce amazing results in the evaluations of the scheduling algorithms with synthetic loads, but can generate devastating results when it is evaluated with loads of real users.

Therefore, in this work we evaluated three methods of generation of random graphs, to produce synthetic loads that were analyzed and evaluated to allow:

- Find the weaknesses and strengths of each method, generating synthetic loads constituted by a large number of DAGs.
- The ability of each method to represent parallel programs of real applications that users submit for execution in an HDCS.
- Evaluate the methods in each of the characteristics of the DAGs, and obtain a comparison of the obtained values.

Finally, with the algorithm proposed in this work, synthetic loads are produced. These loads are evaluated and analyzed before being used in an algorithm for planning and assigning tasks. The evaluations and analyzes generated allow us to design an algorithm, with the ability to predict the planning and allocation of computational resources in the target system.

X. Future Works

Currently, we are experimenting with graphs that contain more vertices. The following experiments will have graphs with 20 to 50 vertices, with the same methods of generation.

We are also working with the design of the algorithm of scheduling and allocation of tasks. This algorithm is planned to be designed with a meta-heuristic strategy; the objective computer system to test the algorithm will have a heterogeneous hardware and a heterogeneous operating system. The synthetic loads generated with the methods proposed in this research work, and characterised with 5 parameters, are the

test DAGs that are received in the target system to measure different performance metrics.

ACKNOWLEDGMENT

The authors would like to thank: Tecnológico Nacional de Mexico and Instituto Tecnológico el Llano Aguascalientes, for the support received to carry out this research work.

REFERENCES

[1] W. Zheng and R. Sakellariou, Stochastic DAG scheduling using a Monte Carlo approach. Journal of Parallel Distributed Computing. Vol 73 Num. 12 December 2013. Pp. 1673–1689. Elsevier Science Publishers B. V

[2] D. Jungnickel, Graphs, Networks and Algorithms. Algorithms and Computation in Mathematics. Springer May 2008. Third edition. Vol 1.

[3] L. Briceño, J. Smith, H. Siegel, A. Maciejewski, P. Maxwell, R. Wakefield, A. Al-Qawasmeh, R. Chiang and J. Li. Robust static resource allocation of DAGs in a heterogeneous multicore system. Journal of Parallel and Distributed Computing. Vol 73 Number 12. December 2013. Pp. 1705-1717. Elsevier Science Publishers B. V

[4] D. Cordeiro, G. Mounie, S. Perarnau, D. Trystram, J. Vincent and F. Wagner. Random graph generation for scheduling simulations. Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques. March 15 - 19, 2010. ACM Digital Library. Torremolinos, Malaga, Spain. Pages 10.

[5] Y. Xu, K. Li, L. He and T. Truong. A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization. Journal of Parallel Distributed Computing. Vol. 73 Number 9. September 2013. Pp. 1306-1322 Elsevier Science Publishers B. V.

[6] S. Yu, K. Li and Y. Xu. A DAG task scheduling scheme on heterogeneous cluster systems using discrete IWO algorithm. Journal of Computational Science. Vol 24. October 2016. Elsevier Science Publishers B. V.

[7] G. Melancon and F. Philippe. Generating connected acyclic digraphs uniformly at random. Information Processing Letters. Vol 90. Number 4. May 2004. Pp. 209-213. Elsevier Science Publishers B. V. Elsevier Science Publishers B. V.

[8] Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Journal of Parallel and Distributed Computing. Vol 59 number 3, December 1999. Pp. 38-422 Elsevier Science Publishers B. V.

[9] Ch. Gogos, Ch. Valouxis, P. Alefragis, G. Goulas, N. Voros and E. Housos, Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing. Future Generation Computer Systems. Vol. 60. July 2016. Pp. 48-66. Elsevier Science Publishers B. V.

[10] F. Xhafa and A. Abraham. Computational models and heuristic methods for Grid scheduling problems. Future Generation Computer Systems. Vol. 26 number 4. April 2010. Pp 608-621. doi 10.1016/j.future.2009.11.005 url https://doi.org/10.1016/j.future.2009.11.005, Elsevier Science Publishers B. V.

[11] M. Akbari and H. Rashidi, A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems. Expert Systems with Applications. Vol 60 October 2016 Pp. 234-248 Elsevier Science Publishers B. V.

[12] P. Erdös and A. Rényi. On random graphs I. Publicationes Mathematicae (Debrecen). Vol 6. Debrecen 1956 Pp. 290-297

[13] T. Tobita and H. Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. Journal of Scheduling. Vol. 5 Number 5 September 2002 Pp. 379-394 Wiley Online Library

[14] R. Dick, D. Rodes and W. Wolf. TGFF: Task Graphs For Free. In Proceedings of the 6th International Workshop on Hardware/Software Codesign. March 1998 Pp. 97-101 IEEE Computer Society. Washington, DC, USA

[15] P. Winkler, Random orders. Order. A Journal on the Theory of Ordered Sets and its Applications. Vol 1. Number 4 December 1985. Pp. 317-331 Springer Netherlands

[16] G. Melançon, and I. Dutour and M. Bousquet-Melou, Random Generation of Directed Acyclic Graphs. Electronic Notes in Discrete Mathematics. Vol. 10 November 2001. Pp. 202-207. Elsevier Science Publishers B. V.

[17] S. Bressan, A. Cuzzocrea, P. Karras, X. Lu and S. Nobari. An Effective and efficient parallel approach for random graph generation over GPUs. Journal of Parallel and Distributed Computing. Vol 73 Number 3 March 2013. Pp. 303-316 Elsevier Science Publishers B. V.

[18] P. Dutot T. N'Takpé, F. Suter and H. Casanova. Scheduling Parallel Task Graphs on (almost) Homogeneous Multicluster Platforms. IEEE Transactions on Parallel and Distributed System. Vol. 20 Number 7 January 2009. Pp. 940-952. IEEE Xplore Digital Library

[19] Y. Xu, K. Li, J. Ju and K. Li. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Information Sciences. Vol 270. June 2014. Pp. 255-287 Elsevier Science Publishers B. V.

[20] A. Velarde, E. Ponce de León, E. Diaz and A. Padilla. Planning and Allocation Tasks in a Multicomputer System as a Multi-objective Problem. EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV. Vol 227. July 10-13, 2013. Springer, Heidelberg. Pp. 225-244

[21] A. Velarde and E. Ponce de León. Planning And Allocation of Tasks in a Multiprocessor System as a Multi-Objective Problem and its Resolution Using Evolutionary Programming. IJACSA. Vol 7 Number 3. March 2016. Pp. 349-360 SAI United Kingdom