Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN INTEGRATED APPROACH OF DYNAMIC TASK SCHEDULING OF DAG WITH DUAL MODE PROCESSORS-USING MACHINE LEARNING TO OBTAIN OPTIMAL MAKE SPAN

PRASANT SINGH YADAV[1], P.K YADAV[2], SUNIL BHARTI[3]

[1]Ph.D Computer Science Scholar at Dr. A P J Abdul Kalam Technical University Lucknow, India,
[2]Principal Technical Officer at CBRI- Roorkee, Planning & Business Development, India
[3]Associate Professor, GCET, Greater Noida, Department of Information Technology, India
E-mail: [1]pdesire82@gmail.com, [2]pkyadav@cbri.res.in, [3]sirbharti@gmail.com

ABSTRACT

With increasing computing demand the need for tuned intelligence-based solutions is most required. Most of the focus has been given by the researcher to the scheduling of parallel tasks dynamically to more than one processor and in the current scenario, it is more demandable. Although many DAG scheduling algorithms are available but less focused on dynamic scheduling. Through our projected paper we want to introduce the approach Dynamic task scheduling algorithm DTSA for scheduling task at run time using DAG with an additional factor regarding processor self-Reconfiguration Capacity, which is an important parameter of distributed computing System. Through DTSA we want to sketch out an adaptive task arrangement algorithm that gives the hybrid result of run-time scheduling of DAG and adaptation of tenant configuration by the processor according to computing needs. Finally, A DAG-based dynamic task arrangement with dependency consideration between the tasks and with the use of machine learning (ML) for self-reconfiguration of a processor is proposed for obtaining the optimal task allocations with the optimal Makespan.

Keywords: *DAG, DTSA, LTA, TPC-W, CPU Self-Reconfiguration, Machine Learning.*

1. INTRODUCTION

The dynamic assignment of tasks in a multiprocessor organization is the most noticing era these days. Getting high performance in multiple systems is an important part of planning similar tasks. The purpose of scheduling tasks at run time is to make connections of each task to the respective processor with their respective performance to achieve the optimal makespan is provided below the job tracking requirements conditions. Various task arrangement techniques are used in the same surroundings as a computer. Most of these programs look for initial delays between activities [1]. To date, Direct Acyclic Graph (DAG) is the more fruitful method adopted as a precautionary measure between intermediate tasks. The goal of this paper is to create a dynamic scheduling pattern with DAG considering task dependencies between processors. In this model, the master scheduler is accountable for handling the run time task arrangement. Through our projected model of run-time task scheduling, we will demonstrate an approach of dynamic task scheduling algorithm DTSA. We tested this projected algorithm by experimenting with some simulators, resulting in that the proposed DTSA algorithm is accepting all dynamic scheduling conditions for task arrangement and showing improved recital.

Adaptability and performance in burning demand in Distributed computing in the context of variable user needs like change in system configuration, protocols, data format principles, and claim for synchronization to the various user function. Many rising practical uses in communiqué and computing stipulate that their functionality must be flexible in the manufactured system. Furthermore, these days research looking for complex heterogeneous and reconfigurable Systems. CPU Self-Reconfiguration computational systems [2] having the self-adaptation capacity to where hardware components, applications, and the operating system associated with each other are ready to adapt itself within available forms to realize the most effective performance.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

We can use a Machine learning framework that uses standard programming tools to measure the best implementation and the results of its multilevel implementation [26].The learning framework functions as an automated computer system that can prepare, heal, operate, and shield itself without the necessity of individual involvement. To achieve such a scenario, self-adaptive computing systems must monitor their behavior for self-updating, in one or several combinations, for their components (in hardware architectures, operating systems, and running applications), Improves the performance of the system if it can adapt its configuration when the workload is changed to overcome potential failure to complete its tasks. Through this paper, we will demonstrate better performance for task processing systems using the learned model of dynamically reorganizing systems in terms of hardware and software. Here, we trained a machine learning model of performance beyond the recital of various software and hardware configurations, at this point we used this model to direct the runtime reorganization of processors by (learning trigger agent) LTA. We have clearly shown that this trigger agent can improve a significant count when tested with different workloads, compared to static configuration.

2. RELATED WORK

Run-time scheduling algorithms are supposed to support are projected in various literature. A real-time system is a condition for system accountability which allows the operator to operate a computer quickly enough to process. The accuracy of the real-time system follows the strict time condition on the logical outcome of the calculation concerning the result generation. There are two types of scheduling algorithm one is static (or offline) scheduling represented by [1, 2, 3 and 4] and the second is dynamic (or run time) scheduling projected by [5, 6]. Mostly scheduling of task arrangement claims to be independent of real-time functions. Some authors considered the DAG model presenting the correlation between tasks are only compatible with real-time systems. In recent times, real-time DAG has been considered for the observation of timeliness of tasks based on equivalent employment [3, 4]. As proposed by [3, 4], the scheduling algorithm does not ignore the priority relationship between functions. Although all are the static scheduling algorithms So, DAG-based task arrangement algorithms which adapted different environments for run-time task arrangement with strict time limits as in real-time system are rare in literature. Hereby our paper we projected hybrid run task time scheduling DTSA-Dynamic Task Scheduling for real-time systems applicable for a variety of environments to handle many equivalent tasks prepared by DAG. Further, it takes into account the most of the capacity of the processor, as the processors used here have the ability to automate processing as needed with varying load.

Only a few papers directly describe the combination of hardware and software for adaptive tuning performance. Most work related to this sector is approached only by keeping a service level agreement (SLA). Although the many works are similar in literature we are working on necessarily diverse issues, where no authorized service level agreement -SLA to settle on acquiescence. Through this section of the paper, we review the relevant work reported. Mostly processors follow the limits of the configuration as source constraints, are mostly handled by the user only for configuration and strictly follow the initially decided set of rules and regulations. On the other hand, our projected approach learns and trained an appropriate model to actual advancement. [16] Author discussed the self-modification of runtime parameters (allowing threads and connections) by adopting a learning-based model approach. Here authors suggested that a parallel method can be extended for software and hardware amendment. This detailed mathematical model is created for the system as inputs to the current workload; here we considered our work model as a black box or as an unproven volume of workload. [1 Reconstruction] Self-reconstruction is a technology that can quickly modify logic configured to suit application needs at runtime. Although performance improvements have been demonstrated using auto-reassembly, the technique has only been described informally. The exact definition of self-rebuilding based on abstract reconstructive device design is presented in this paper [15]. To use this model as a self-tuning regulator, a system was proposed to evaluate the performance model, using the repeat list-square estimation method as a black box for the system. Which may be an integral part of this loom is proposed to support the service level agreement SLA to obtained the aim (considering delay as a parameter), keeping in the view that to considering most of the service level agreement-SLA prerequisite besides increasing recital [20].

The author uses the queuing model to manage multi-range Internet applications. This approach

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

aims to maintain several different servers in each array and is believed to be available for later unused machines for provisioning. Our approach is that one server at each level should have total processing power [11]. It is proposed to use self-recycling processor architecture to provide better processing performance with a changing workload. The processor can be re-configured to interchange with other attached high configuration processors. Effective-auto-optimization can be accessed by combining a fast-based redesign strategy to schedule reusable co-processors and minimal-square optimization algorithms. In [26], the author proposed an efficient scheduling algorithm, which we would like to demonstrate a custom scheduling algorithm that is a combination of efficient work assignments by considering DAG with auto-configuration of respective allotted processors. Self-rebuilding processors can be tuned for software and hardware to meet computing needs.

3. PROPOSED MODEL AND ALGORITHM

A heterogeneous system is assumed to have the arrival of tasks dynamically and all parallel tasks presented by direct acyclic graph DAG. Constant DAG [7] is indicated by G = (V, E) where V is the set of v nodes and the set of edges directed by E. The task is presented by a node in the DAG, which is the group of instructions that are to be consecutively executed without permission on the identical processor. The calculation cost of the node is presented by the weight of a node and is indicated by w (vi). The edges of the DAG represented by every one of these (vi, vj), respectively to communiqué messages and the interruptions between the nodes. The communication cost of the edge is known as the weight of the edge and is denoted by c (vi, vj). The starting node of the edge is termed as a parent node and the underneath node is termed as the child node. A parentless node termed as the entry node, on the other hand, a child without a node is known as an exit node. A heterogeneous computing system contained a group of processors, P = {P1, P2, P3…Pm}, where the above set represents processors having confined memory. Taken processors are closely connected [8] however; the communiqué costs connecting processors are universal.

Figure 1 illustrates a new real-time arrangement model in a heterogeneous setting. When every one of the equivalent tasks reached to a specific processor called the master scheduler which is responsible for put task in PTQ, it enters a queue which is termed as Base Task Queue (BTQ) to linger until it is listed; Further, with handling to the BTQ, the Master scheduler also deal with two other queues, which are called Sent out Task Queue (STQ) and Finished Task Queue (FTQ) respectively. Introduced encrypted scheduling algorithms in Master Scheduler for working through BTQ. Here Master Scheduler is fully accountable for arrangement each task completed in sent out task queue STQ. Once the scheduling algorithm is introduced, each task among the task queue is configured concerning its reliant on other tasks.

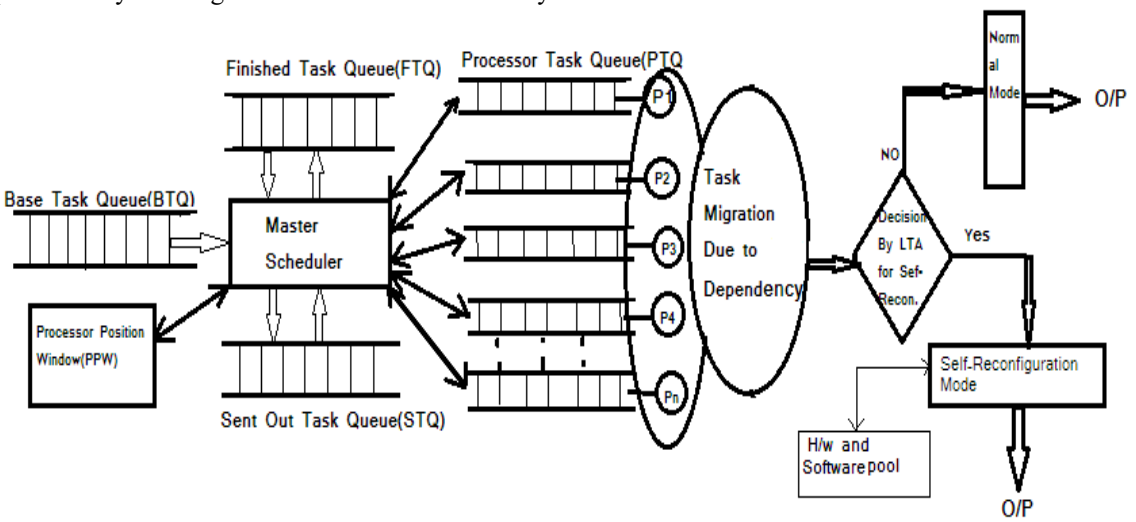

Figure 1: Dynamic Task Scheduling Model With Processor Self-Reconfiguration Decision

As shown in Figure 1 the Dynamic Task Scheduling Model containing processor self-reconfiguration capacity, it decides for self-reconfiguration once the scheduling of tasks is completed. Now the master scheduler put scheduled tasks to the respective processor task queue (PTQ). According to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

respective PTQ Self-rebuilding processors completed assigned tasks by inspection together employing the outcome of their work based on FTQ. If FTQ does not take advantage of its dependent function result, PTQi will then refer to PTQi + 1, PTQi + 2,… PTQn, PTQ1… PTQi-1 to solve the appropriate task and PTQi will refer to that function. The scheduling algorithm will stop working until the PTQ becomes empty. Then Processor Position Window (PPW) put on show the current position of every one processor that when the processors are in an operation state or is it needs Self-Reconfiguration or not and in idle state. When the task is migrating from one processor to another processor we also make a check here for processor business and decision to think about whether go for self-reconfiguration to make speedup computing for the migrated task or release the dependency as soon as possible or left the working of the processor as in normal mode.

We take here a thresh hold value of dependency task so above this value we go for Self-Reconfiguration decision and vice-versa. Additionally, one more parameter temperature variation with changing load by some scientific benchmark Linpack which is one of the most consistent CPU benchmarks, through these two parameters we can know that the processor is overload or not. Temperature is used as a major parameter to show the CPU overloading conditions with the use of a training model in machine learning we take prediction for CPU self-reconfiguration by Learned Trigger Agent (LTA) it takes dynamic decision by machine learning with. If the output value of the LTA is one, then it will decide to self -reconfigure CPU or on the other hand, if the output value of the training agent is 0 then it will not go for self-Reconfiguration. For Self-Reconfiguration it adapts the required hardware and software from the available resource pool as AWS (Amazon Work stations), Soucelab and TPC-W benchmark, etc. The time taken from a static configuration to advance reconfiguration is compensated speedup gained by the processor. Here we adopt two-step policies: firstly, we assigned the task to each processor dynamic task scheduling algorithm which is the optimal method to assign the task to the best available processor at run time.

Assumptions:

- All processors are heterogeneous in the configuration.
- The execution time of each task is already known.
- All cooling conditions related to CPU temperature are taken ideal.
- Time taken in deciding for self-reconfiguration is compensated by the achieved speedup by scheduling.
- We have taken CPU temperature as a major parameter with assumed processor task dependency threshold value for showing CPU overloading.
- CPU utilization is also taken as an overloading factor concerning temperature.
- Another factor for showing CPU overloading is taken as ideal.
- After Self–Reconfiguration Processor dependent task execution time is assumed.

Here we put forward an innovative dynamic task scheduling algorithm DTSA based on the master scheduler model which is responsible for putting task in PTQ with consideration of task dependency . This strategy completes the entire parallel work as soon as possible. This parallel work has a very short response time. Accordance by the DTSA, tasks in the ITQ is determined by its dependence. In the initial task queue ITQ, the front task is always predetermined and allotted to the respective processor by the projected algorithm. As it is well known that all static scheduling algorithms maintain the task priority queue by calculation resulting in fixed priority rank of each task, for the reason that task data of the DAG is predetermined for each task [9]. But, in our projected dynamic task scheduling algorithm DTSA differs from all static scheduling algorithms by changing the task for the period of runtime. Our algorithm focuses on the DTSA processor selection strategy. Although the tasks are predetermined, the tasks depend primarily on selecting the mapped processor. Here two time-indexes are considered first is the earliest completion time ECT of the processor Pi and second is the earliest start time EST of the task v_i on the processor Pi. While choosing a processor to process exacting task from the PTQi with the decision of the self-Reconfiguration requirement in terms of hardware and software for this machine learning is used by our data set taken from some scientific benchmark lencpark and TPC-w to learning trained agent (LTA) to KNN algorithm to find out the optimal decision for CPU Self-Reconfiguration. Given a current task requested to the respected processor of each three processors the respective probabilities.

We take here a thresh hold value of dependency task 15 unit and above it so above this value we go for Self-Reconfiguration decision and CPU utilization

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and the temperature is calculated by task manager by threshold temperature and CPU utilization values are taken respectively which are 95 degrees Celsius and 85 percent, obtained by Some Scientific Benchmarks. Here temperature and task dependency threshold values are taken into consideration that when processors need configuration or not which is decided by learning trained agent (LTA) machine learning model of CPU or processor which is summarized in table 1. With the use of the KNN algorithm (k-nearest neighbors algorithm-, a method for categorizing objects) the obtained correlation between temperature, processor utilization (overloading), and dependent task size is shown in Figures 2, 3, and 4. The relationship between temperature and percentage CPU utilization is taken from Linpack which is one of the most consistent CPU benchmarks, further between task size and percentage CPU utilization relation hypothesis is taken from task manager in computer and correlation between task size and temperature relation taken from Assyad et al 2004 [24].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Show Data Set Of Temperature And Task Dependency With Overloading

S.N.	Temperature	Task Dependency	Processor Condition	Self-Reconfiguration Decision
1	100	15	Overload	1
2	94	15	Overload	0
3	98.8	18	Overload	1
4	97.6	19	Overload	1
5	95	14	Overload	0
6	95.6	15	Overload	1
7	98.1	16	Overload	1
8	94.6	14	Not Overload	0
9	95.4	17	Overload	1
10	92.5	15	Not Overload	0
11	90.4	4	Not Overload	0
12	94.8	9	Not Overload	0
13	88.3	8	Not Overload	0
14	95.2	21	Overload	1
15	86.5	13	Not Overload	0
16	87.3	11	Not Overload	0
17	96.8	17	Overload	1
18	86.2	18	Not Overload	0
19	89.2	13	Not Overload	0
20	85.1	15	Not Overload	0
21	95.8	16	Overload	1
22	85.4	9	Not Overload	0
23	97.2	7	Overload	0
24	85.6	16	Not Overload	0
25	97.7	15	Overload	1
26	86.4	13	Not Overload	0
27	97.4	16	Overload	1
28	88.8	12	Not Overload	0
29	89.4	14	Not Overload	0
30	96.9	20	Overload	1
31	90.7	11	Not Overload	0
32	96.6	17	Overload	1
33	95.9	18	Overload	1
34	94.7	12	Not Overload	0
35	96.3	19	Overload	1

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

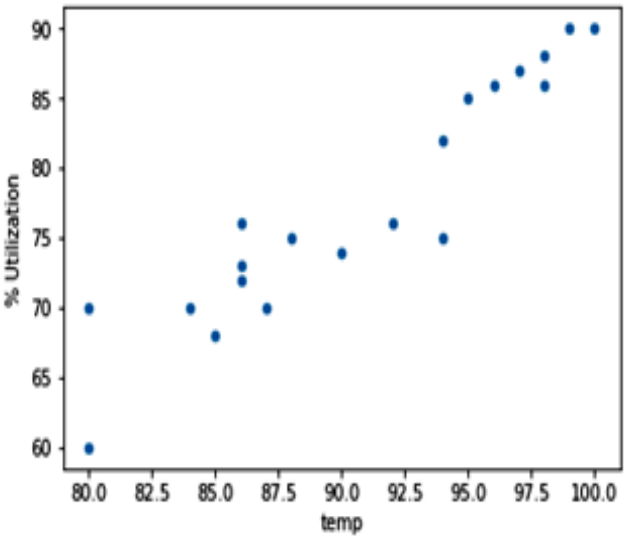

Figure 2: Correlation Between Temperature and Processor Utilization

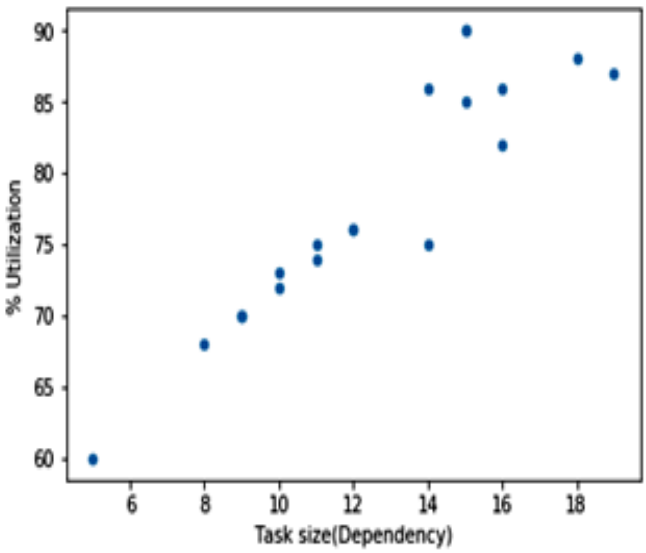

Figure 3: Correlation Between Dependent task size and Processor Utilization

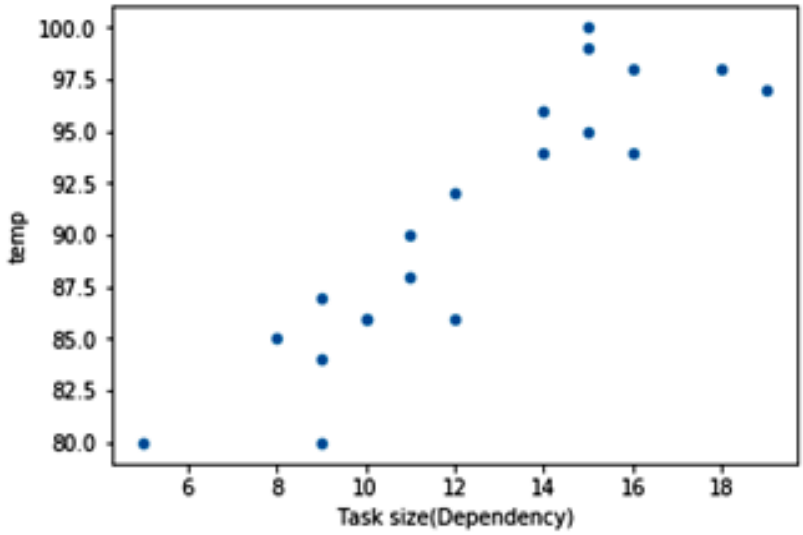

Figure 4: Correlation Between Temperature and Dependent task size

Figure 2 shows the correlation between percentage CPU utilization with respect to temperature taken from Linpack which is one of the most consistent CPU benchmarks, figure 3 shows the correlation between task size and percentage CPU utilization this relation hypothesis is taken from task manager in computer and figure 4 shows the correlation between task size and temperature relation taken from Assyad et al 2004 [24], obtained by KNN algorithm is used to train machine learning trained agent (LTA). In our projected model the parallel task and ready task are assigned to base task queue BTQ and processor task queue respectively PTQ. BTQ and STQ positioned at the master scheduler and tasks are mapped and executed at the separate point from the master scheduler at PTQ for all processors. It is noticeable here that task Scheduling and the execution practice goes on simultaneously, Thus the task scheduler and operational processors must be tuned to each other.

Algorithm:

Step1. Procedure DTSA

Step2. Sort the Sent Out Task Queue as

STQ [ ] = SORT [Ti, Tj]

i = 0;

While (STQ [ ] is not unfilled) do

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

for i = 1 to n
. ptqi = stq [i]
i= i + 1
End for;
End while;

Step3 For every one processor Pk in set of processor do
While (Pk is at running position)
Bounce and chose next ptqk+1
End while

Step4. Pk = ptqk[j]
If (dependent task of ptqk[j] is in ftq)
TP (ptqk[ ],Pk , j, ftq, cpk, CTj)
Else
do
Advanced the pointer to the subsequent ptq
If (reliant task of ptqk[j] is in ftq)
Task Processing TP (ptqk[ ],Pk , j, ftq, cpk, CTj)
Exit do
End if

Step 5. While (check each ptq's)
End if
End for
End DTSA

Step 6. Method for Task Processing (TP)
Method TP (ptqk[ ],Pk,j,ftq,cpk,CTj)
do Tj with Pk

Check CPU overloading

If CPU temperature> 95^0 C && Thresh hold Value of task dependency (15 Unit)→ CPU Overload
Then Go to Self-Reconfiguration (use KNN Algorithm)
{
If H/w < H/w capacity required
Then go for Hardware configuration
{
Go to Hardware pool
}
Else if
$S/W_{CPU}<S/W_{ReqVersion}$

Then go for S/w reconfiguration
{
Go to Software pool
}
Else if
$H/W_{CPU}< H/W_{Capacity\ Required}$ && $S/W_{CPU}<S/W_{ReqVersion}$
Then go for H/w and S/w reconfiguration
{
Go to Hardware and software pool
}
}
Else
Process the assigned task in a Normal way
Steps 7 remove Tj from ptqk
Insert Tj in ftq
. cpk = cpk + Ctj
End Task processing (TP).

4. RESULT AND DISCUSSION

According to the model shown in figure 1 DTSA, the worst-case time complexity for DTSA (m^3), where m is the total number of tasks. Step 2 distributes all the functions in STQ to PTQi. Step 3 shows how to bring the journey to the processor in each ptq. Step 4 examines the availability of TJs in FTQs. If Tj is on hand then carry on with the identical processors, otherwise select the subsequently PTQ before working to obtain the appropriate function. After assigning the task or assigning it to the respective processor, post it to PSW and remove it from PTQi. This progression repeats every single one PTQi field. If any PTQ completes its task set as soon as possible, change the appropriate task from the available PTQs.

If, however, any task is not appropriate to bring that cycle, the processor will have to wait until one of the tasks based on FTQ becomes available. The waiting time is called the processor inactive status. The time intricacy of DTSA is O (m^3), where m is the number of tasks, intimately associated to the span of the STQ and the totality figure of processors, which is interrelated to the previous liaison between the functions in the DAG.

Demonstrating the scheduling model and DTSA, a succession of simulation experimentation was intended. The DAG of the parallel task for simulation is generated randomly. We have taken a total number of the task are 12 and the number of processors is 3. Table 2 represents the input data, randomly by the DAGEN tool which generates the direct acyclic graph with respect to weights edge and nodes.

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Input Task List With Dependency Tasks And Processing Cost Generated By DGEN Tool

Task id	Parent Task id	CP (Cost of Processing)
T0	-	11
T1	T0	3
T2	T0	11
T3	T0	18
T4	T3	5
T5	T1	17
T6	T0	4
T7	T0	16
T8	T4	17
T9	T2	15
T10	T8	2
T11	T4	19

Table 3: Processor Task Queue (Ptqi) At Base Stage

PTQ1	T0	T3	T5	T6
PTQ2	T2	T9	T11	T4
PTQ3	T7	T1	T8	T10

Table 4: Task Migration Between Processors

PTQ1	T0	T3	T5 Migrate to P3 at run time	T6 Migrate to P1 at run time
PTQ2	T2	T9	T11	T4 Migrate to P1 at run time
PTQ3	T7 Migrate to P1 at run time	T1 Migrate to P1 at run time	T8	T10

PTQ has evenly distributed tasks with dependency consideration. The initial or base stage status of PTQi is illustrated in Table 3. The dynamic scheduling is put into operation with the task immigration for the duration of due to dependency at the run time. Table 4 demonstrates the task movement and the runtime irregularity of parallel tasks. The scheduling span for a sole processor for the taken example is 140-time units. The Time in use to finish tasks by the initial processor (P1) is 54-time units, for P2 are 67-time units and P3 is 48-time units. We have taken as a supposition, the total numeral of processors in the in our projected model is assumed are of three in numbers.

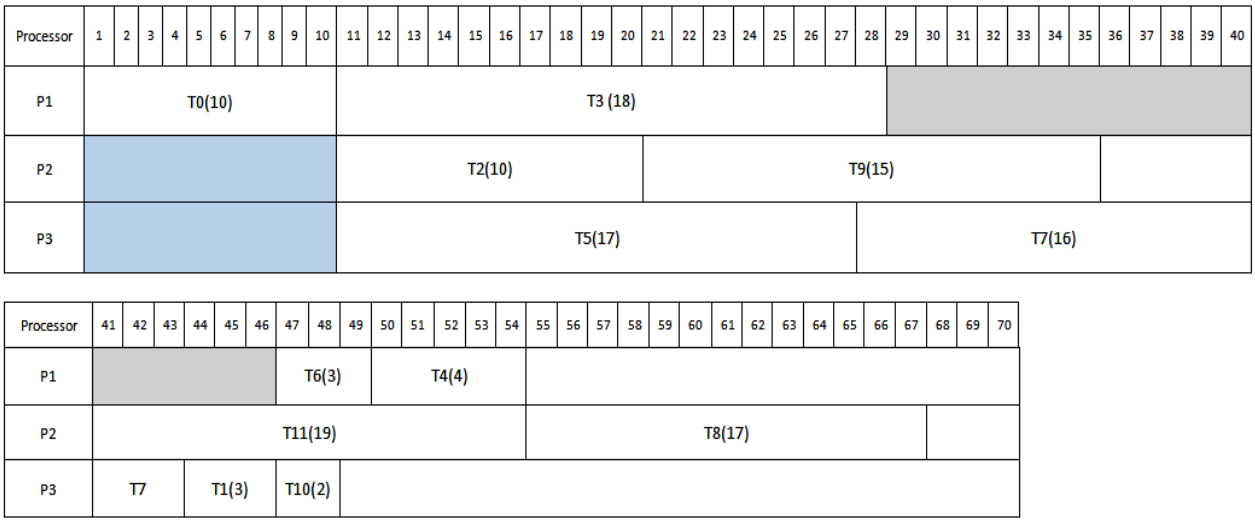

Figure 5: Computation Cost Of Scheduled Task By DSTA

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The minimum schedule Length (SL) can be obtained from

$$\text{Minimum S L} = \frac{\text{Schedule length in single processer}}{\text{Number of processer in architecture}}$$

Figure 5 illustrated the minimum schedule span of the above example which is 46.

$$\text{Hence, the speedup} = \frac{\text{Minimum Schedule length}}{\text{Schedule length}}$$

$$= \frac{46}{69}$$

$$= .66667$$

$$= .66667 \times 100 = 67\%$$

We got a speed up the percentage of this model is 67 % in normal mode processor. For the self-Reconfiguration capacity of processor, in continuation of the above phenomena, the current task assigned to the respected processor of each three processors the respective probabilities and run time dependency anomalies and at the same time CPU loading is calculated by the task manager by threshold temperature value which is 95 degrees Celsius obtained by Some Scientific Benchmarks. Hare temperature threshold values are taken into major consideration with an additional parameter task dependency thresh hold time 15 units when processors need configuration or not which is decided by the learned trigger agent (LTA) machine learning model of CPU or processor. We obtained tables 5 and 6 by Machine learning Agent (LTA) according to sum (thresh hold) assumed standard values.

Table 5: After Self Reconfiguration Decision The Updated Table

S.N.	Processor	Task id	Parent Task id	CP (Cost of Processing)	Decision for Self-Reconfiguration
1	P1	T0	-	11	0
2	P1	T3	T0	3	0
3	P1	T5	T0	11	0
4	P1	T6	T0	14	0
5	P2	T2	T3	5	0
6	P2	T9	T1	14	0
7	P2	T11	T0	4	0
8	P2	T4	T0	13	0
9	P3	T7	T4	14	0
10	P3	T11	T2	12	0
11	P3	T8	T8	2	0
12	P3	T9	T4	12	0

Journal of Theoretical and Applied Information Technology
30th November 2020. Vol.98. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 6: Input Task List With Updated CP

Task id	Parent Task id	Updated CP (Cost of Processing)
T0	-	11
T1	T0	3
T2	T0	11
T3	T0	14
T4	T3	5
T5	T1	14
T6	T0	4
T7	T0	13
T8	T4	14
T9	T2	12
T10	T8	2
T11	T4	12

Now the values of the table no. 5 and 6 are applied through our DTSA algorithm to our initial or base task assignment table no. 4 and run-time anomalies concerning task migration table no. 4 we find the following computations cost of scheduled task by DSTA with self-reconfiguration event with the processor as per computing need of workload.

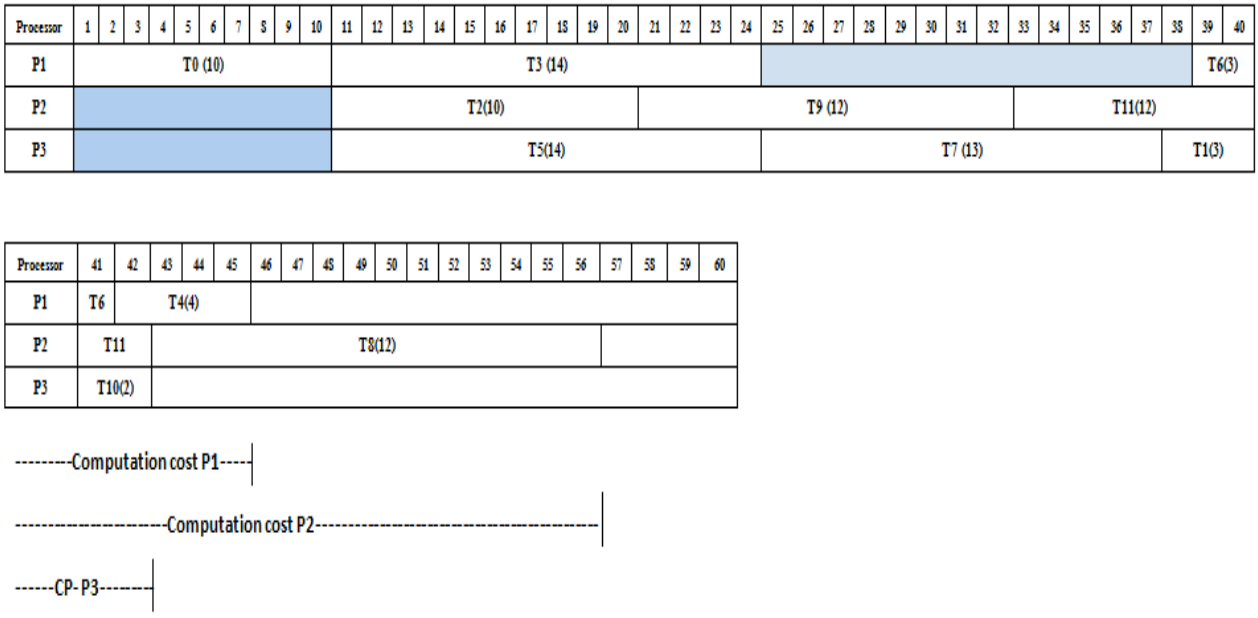

Figure: 6 Computation Cost Of Scheduled Task By DSTA In Self-Reconfiguration Mode

The minimum schedule length (SL) can be obtained from

$$\text{Minimum SL} = \frac{\text{Schedule length in single processer}}{\text{Number of processer in architectu re}}$$

Figure 6 illustrated the minimum schedule length of the above example which is 42.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$\text{Hence, the speedup} = \frac{\text{Minimum Schedule length}}{\text{Schedule length}}$$

$$= \frac{42}{56}$$

$$= 0.75$$

$$= 0.75 \times 100 = 75\%$$

We got to speed up the percentage of this model is 75 % in the Self-reconfiguration mode processor So it is clear that when we use the processor in normal mode with the DSTA algorithm we obtained a speed up 67% which is very good in comparison to various static load scheduling algorithm and further when we use DSTA with the self-reconfiguration mode of the processor with the computing need to be decided my machine learning Agent LTA be obtained speed up value 75% which is greater than DTSA used with the normal processor. But simultaneously with the proper scheduling, with the help of our research work, it is observed that the speed of the parallel system in a distributed computing environment also depends on the number of processors and its types in the structural design with the numeral of tasks arranged. For every case, the number of tasks and the number of processors are straightforwardly relative to each other. However, the structural design used in our mold is not to entertain communiqué costs, it is very low.

And additionally also compared DTSA with former list scheduling algorithms; the Dynamic level scheduling algorithm (DLS) and Levelized Min Time (LMT) which are projected in the literature concerning heterogeneous distributed computing systems. The intricacy of DLS, LMT, algorithms is O (m^3n^2), O (m^2n^2) respectively. Here we selected some recently projected algorithms for upgrading even numerous performance measures such as the total run time are recommended in the literature [16]. The comparison with above mentioned previously published algorithm to our projected DTSA (dynamic task scheduling algorithm) is shown in table 7(a) and 7(b).

Table 7 (A) Comparison Of The Previous Algorithm To DTSA Algorithm

Algorithm	Complexity	Processor	Number of Task	Run Time
LMT	$O(m^2n^2)$	3	10	95
DLS	$O(m^3n)$	3	10	91
DTSA(Proposed Algorithm)	$O(m^3)$	3	12	67
** When Processors Are Used In Normal Mode*				

Table 7(B) Comparison Of The Previous Algorithm To DTSA Algorithm

Algorithm	Complexity	Processor	Number of Task	Run Time
LMT	$O(m^2n^2)$	3	10	95
DLS	$O(m^3n)$	3	10	91
DTSA(Proposed Algorithm)	$O(m^3)$	3	12	56
** DTSA Algorithm Is Used With Self-Reconfiguration Processor Mode*				

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

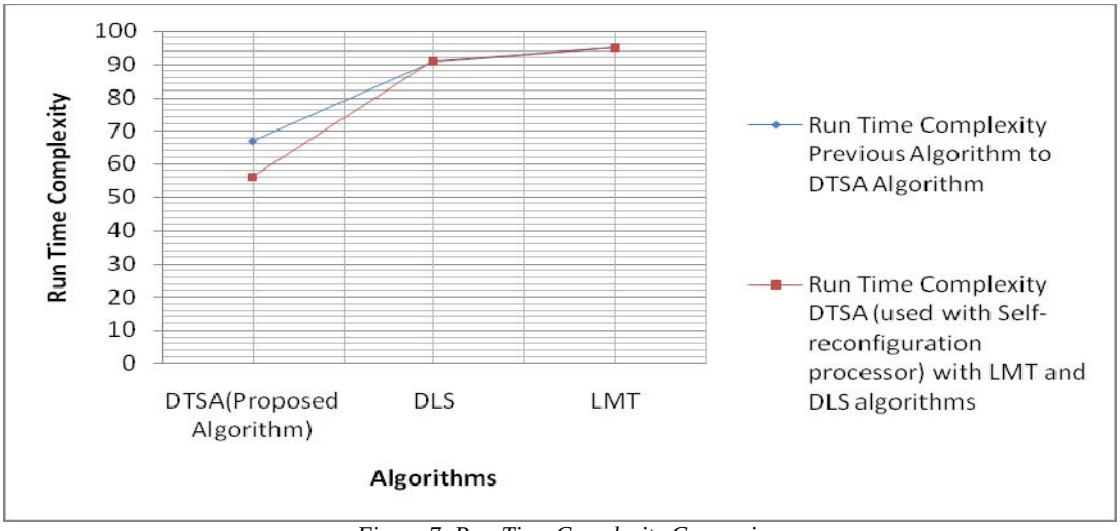

Figure 7: Run-Time Complexity Comparisons

From the comparison detail from both part of table 7 and form figure 7 it is clear that our projected algorithm DTSA performs better than previously published static scheduling algorithms. The result of the projected dynamic scheduling mold and scheduling algorithm provides DTSA based DAG equivalent functions, Good schedule with high speed up a percentage in both cases: normal mode and self-reconfiguration Mode.

5. CONCLUSION AND FUTURE WORK

Here we developed a runtime task configuration model with a dynamic task scheduling algorithm DTSA using machine learning with optimal complexity. Through DSTA we evaluate runtime task alignment algorithms using DAG, which includes auto-reassembly of processors capable of implementing time optimization of tenant configurations to suit computing needs with dependency consideration and involuntary online self-reconfiguration of a system's hardware and software as per changing workload. We found improved results in comparison of other previously published algorithm as DLS and LMT published by other authors. So in both case: Normal mode processor and with the use of self-Reconfiguration mode processor our projected algorithm DTSA confirms noteworthy enhancement in comparison to fixed allotment of resources as used by DLS and LMT methods. Even though our agent (LTA) is just trained for one precise sphere, the method is common and it may be used many potential combinations of operating systems hardware software, and workloads. of our algorithm we obtained better results the previously published algorithm by other authors. In future research, we may have aim to trained learning agents to some other specific domain for reliability and fault tolerance for better outcomes by considering some other parameter fault tolerance and reliability, as well as investigation on different workloads.

REFERENCE:

[1] T. F. Abdelzaher and K. G. Shin., "Combined task and message scheduling in distributed real-time systems" IEEE Transaction on Parallel and Distributed Systems,Vol.10,No.11,Nov.1999,p.1179-1191.

[2] J.C.Palenci *et al.*, "Schedulability analysis for tasks with static and dynamic offsets", In Proceeding of the 19th IEEE Real-Time Systems Symposium, 1998, pp.26-37.

[3] Xiao Qin *et al.*, "Reliability is driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems" In Proceeding of 12th International Conference Parallel and Distributed Computing and Systems, 2000, pp.1456-1466.

[4] Xiao Qin *et al.*," Real-time fault-tolerant scheduling in heterogeneous distributed systems," in proceeding of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, V I, 2000, pp. 421-427.

[5] V.Kalogeraki *et al.*, "Dynamic scheduling for soft real-time distributed object systems," In proceeding of Third IEEE International

Symposium on Object-Oriented Real-Time Distributed Computing, 2000, pp. 114-121.

[6] G.Manimaran *et al.*, "An efficient dynamic scheduling algorithm for multiprocessor real-time systems," IEEE Transaction on Parallel and Distributed system, Vol.9, No.3, 1998, pp. 312-319.

[7] Dan Ma *et al.*, "Dynamic Scheduling Algorithm for Parallel Real-time Jobs in Heterogeneous System" Proceedings of the Fourth International Conference on Computer and Information Technology (CIT"04) IEEE,2004, pp.462-466.

[8] Kai Hwang, "Computer Architecture and Parallel Processing", Mc Graw Hill Publication 1984.

[9] G.J. Joyce Mary *et al.*, "Dynamic Task Scheduling in Multiprocessor and the Swift Embryonic World of Parallel Computing – A Survey", Published in the International Journal of Algorithm, Computing, and Mathematics – Vol. III – No. 4,2010 pp.53-60.

[10] D.I. George *et al.*, "A new DAG based Dynamic Task Scheduling Algorithm (DYTAS) for Multiprocessor Systems", International Journal of Computer Applications Volume 19-No.8, April 2011, pp.24-28.

[11] Michael Kirchhoff *et al.*, "A Real-Time Capable Dynamic Partial Reconfiguration System for an Application-Specific Soft-Core processor", in International Journal of Reconfiguration Computing. Volume 19, June 2019, pp.1-14.

[12] Ira Cohen *et al.*, "Capturing, indexing, clustering, and retrieving system history". In Proceedings of the 20th ACM Symposium on Operating Systems Principles, October 2005, pp105–118.

[13] Ira Cohen *et al.*, "Fast effective rule induction. In Proceedings of the 12th International Conference on Machine Learning (ICML-95), 1995, pp 115–123.

[14] Daniel F Garcia et al, "TPC-W e-commerce benchmark evaluation Computer", IEEE Computer 36(2) Feb 2003 pp.42–48.

[15] M.Karlsson et *al.*, "Dynamic black-box performance model estimation for self-tuning regulators", In Proceedings of the 2nd International Conference on Autonomic Computing, June 2005 pp.172–182.

[16] Anna M Haywood et al, "The relationship among CPU utilization, temperature, and thermal power for waste heat utilization" in the journal of Energy Conversion and Management, Volume 95, May 2015 pp.297-303.

[17] J. Norris et *al.*, "On-Call: Defeating spikes with a free-market application cluster". In Proceedings of the 1st International Conference on Autonomic Computing, May 2004 pp.198–205.

[18] M.Oslake *et al.*, "Capacity model for Internet transactions. Technical Report MSR-TR-99-18, Microsoft Corporation, April 1999,pp 1-13.

[19] Dino Quintero *et al.*, "Introduction to P-Series partitioning", International Business Machines Corporation, November 2004 pp.1-293.

[20] B. Urgaonkar *et al.*, "Dynamic provisioning of multi-tier internet applications". In Proceedings of the 2nd International Conference on Autonomic Computing, June 2005 pp. 217–228.

[21] J. Wildstrom *et al.*, "Towards self-configuring hardware for distributed computer systems" In Proceedings of the 2nd International Conference on Autonomic Computing, June 2005 pp.241–249.

[22] J. Wildstrom *et al.*, "Adapting to workload changes through on-the-fly reconfiguration", Technical Report UT-AI-TR-06-330, May 2006 pp.1-15.

[23] Witten and Franke et al. "Data Mining: Practical machine learning tools with Java implementations", Morgan Kaufmann, San Francisco, pp.1-558, 2000.

[24] Assayad, A. Girault and H. Kalla, "A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints," International Conference on Dependable Systems and Networks, Florence, Italy, 2004, pp. 347-356.

[25] Becker, M *et al.*, "Dynamic and partial FPGA exploitation" Proceedings of the IEEE, 95(2), Feb 2007 pp. 438 –452.

[26] Christopher Claus *et al.*, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system", In Proceedings of the conference on Design, automation and test in Europe, Nice, France, May 2007, pp.498-503.

[27] Prasant Singh Yadav et al., "Scheduling of Task Graph Using DAG with Dual Mode Processors in Heterogeneous Distributed Computing System", In International Journal of Advanced Science and Technology 29(05),May 2020 pp.9806-9817.