Classification of Task Partitioning Strategies in Distributed Parallel Computing Systems
CHAPTER 3
CLASSIFICATION OF TASK PARTITIONING STRATEGIES IN DISTRIBUTED PARALLEL COMPUTING SYSTEMS

3.1 Introduction

Distribution of tasks amongst various computing nodes is a challenging problem in high performance distributed computing systems. To choose the appropriate strategy for required system is difficult without meaningful comparison of existing task partitioning and load balancing strategies. Effectiveness of strategies depends upon the number of factors including efficiency, interconnection topology, communication mode, program structure, throughput and computing capabilities of high performance computing structures. A number of task partitioning and load balancing strategies have been proposed, each of which produces better results under different circumstances. The main goal of this chapter is to unravel the mystery of strategies and to classify when and where each strategy is appropriate. This chapter provides a common terminology and classification mechanism. Scheduling is a function which is used to assign jobs to the different processors [6]. It’s a two-step process, processor allocation and assignment. A job is an autonomous program that executes in its domain. Resource allocation is done by the scheduler over two dimensions, time and space and at two level jobs and threads. In running state, job constitutes threads to reduce overhead.

If a software package is used to execute parallel jobs instead of threads, it increased load of parallel computing systems. Threads and communication between them may be static or dynamic [10]. For example, in MIMD architecture [1], number of threads and the communication pattern between the threads can change dynamically during execution in the parallel computing systems. If multiple executing entities are part of the same application, then we treat entities as threads and applications as jobs [30]. A parallel job is the collection of tasks having some precedence relationship. A task can be identified as an executable fragment that must
be sequentially executed without partial parallel execution [2]. All parallel jobs cannot be fully parallelized. Effective task partitioning and load balancing of large task is required to achieve high performance in a parallel and distributed system. Increasing demand of high performance computing systems amongst various fields of science shows keen interest in the parallel computing. Selection of appropriate strategies for a particular system is a deciding factor for successful execution of the tasks. In a homogeneous architecture, serial fraction of computation executes at the speed of any of the identical processors. Sequential bottlenecks can be greatly reduced by executing tasks on heterogeneous parallel computers by running critical tasks on faster processors. Efficiency analysis of heterogeneity presented by [4, 5]. Processor allocation in efficient manner deals with the determination of the number of processors allocated to a job [7]. Time complexity analysis tool for task partitioning has been developed by Pugh and Nirkhe [8]. It accurately estimates the execution time of a general real time program employing on high level structures. Towsley and Nelson [9] proposed an analytical model for partitioning independent tasks on different processors. There is no ideal task partitioning strategy for all diverse computing architectures. Under the different architectures some important task partitioning strategies are as follows:

### 3.2 Deterministic task partitioning strategies

In deterministic task partitioning, characteristics of an application such as communication cost, execution time, synchronization and data dependency are known in advance [11]. Dynamic scheduling performs scheduling at runtime and the allocation of the processes may change during the execution of tasks. Static scheduling cannot support load balancing and fault tolerance while dynamic scheduling support these parameters. Deterministic scheduling on a Network of Workstation (NOW) is NP-complete in strong sense while nondeterministic is not strongly NP-complete. Purely static task partitioning with OpenCL, determines the best partitioning across processors in a system. It divides tasks into many chunks based on the availability of processors [13].
3.2.1 Papadimitriou and Yannakakis scheduling: Papadimitriou and Yannakakis [34] scheduling approximate absolute achievable lower bound start time of a node by using e-value attribute. From the first node up to exit node e-value have been computed recursively to assign the priority of the nodes. After calculating e-value of the node, each node inserted into a cluster in such a way that ancestors have data arrival time larger than e-value of the node.

3.2.2 Linear Clustering with Task Duplication Scheduling: LCTD (Linear Clustering with Task Duplication) [35] scheduling identifies the edges among clusters. These edges determine completion time after linear clustering of DAG. After that it attempts to duplicate the parent nodes corresponding to these edges in order to reduce start time of some nodes in the cluster.

3.2.3 Edge Zeroing Scheduling: Edge Zeroing (EZ) Scheduling [33] tries to select computing environment by merging edge weights. This scheduling strategy finds...
edges with the largest weight in every step. If merging does not decrease execution
time then two clusters are merged (zeroing the largest weight) for the reduction of
completion time. Ordering of the nodes is defined in the resulting cluster based upon
the static b-level of the nodes. DAG edges have been sorted in descending order of
dges weights.

3.2.4 Modified Critical Path Scheduling: Modified Critical Path (MCP) [31]
scheduling decides priorities of nodes on the basis of ALAP (As Last as Possible)
parameter. It computes ALAP of the existing node and constructs a list of nodes in
ascending order of ALAP times. MCP finds processors idle time slot for a given node.
It selects first node from the node list and insert it into the processor of earliest
execution time [15]. It cannot guarantee an optimal schedule for fork and join
structures.

3.2.5 Earliest Time First Scheduling: ETF (Earliest Time First) scheduling [32]
technique select the node of the smallest start time from the node of earliest start time
in ready queue at each step. Earliest start time is computed by examining start time of
the node on all processors exhaustively. When two nodes have same EST, then ETF
breaks the tie by selecting the node with higher static priority. The priorities are
assigned to the nodes by computing static b-level.

3.3 Dynamic task partitioning strategies

Position Scan Task Scheduling (PSTS) is pure dynamic load balancing. It can be
used in centralized and decentralized manner [14]. It is based on divide and conquers
principle in which higher grid of dimension k is divided into grids of dimension (k-1),
until the dimension is (1). Position Scan Load Balancing (PSLB) technique is two
phase technique. In the first phase, the system modeled as a hypercube only at one
time. In the second phase, load balancing is performed by obtaining necessary
information for each system node. On the basis of the prior information, PSLB
strategy calculates its future load. After that load is migrate according to the
capabilities of the nodes. An important aspect of this strategy is that, in the case of
load migration, each node knows exactly where to send its extra work load.
Communication takes place only between neighboring nodes.
3.3.1 **Evolutionary task scheduling**: Evolutionary Task Scheduling [26] depends upon previous scheduling stages. Use of heuristic in initialization phase and specific mutation operator are two parameters, which provide beneficial results for effective scheduling in a static environment. In consistent environment, tasks moves from higher level processors to lower level processors. This can be achieved by using “rebalancing” operator. But in the case of inconsistent environment less greedy operators are used to provide the best results. In this scheduling, information collected from the previous state is used to select the environment by the scheduler.

3.3.2 **Dynamic Priority Scheduling**: Dynamic Priority Scheduling (DPS) [27] assign priorities to the tasks based upon difference of bottom level (b-level) and top level (t-level). This technique tries to schedule minimum schedule length in Directed Acyclic Graphs (DAGs) [3]. In Dynamic process assignment, the scheduler selects more important tasks before less important ones. Mapping and scheduling of strategies depend upon the processor scheduler, network architecture and the DAG structures [28]. The t-level is the length of the longest path (execution cost + communication cost) between the target and entry tasks of DAG. While b-level is the large path between target and exit tasks. In this way, t-level determines earliest start time. This start time consider as a critical path of GDA [12].

Dynamic Level Scheduling (DLS) [29] uses Generalized Dynamic Level (GDL) to determine dynamic priorities for all tasks in the ready queue. GDL gives priorities to the processors according to their speeds. Many other factors are taken into account due to different execution costs on each processor. It considers factors like median of the execution cost over all processors, average execution cost and maximum or minimum costs for the computation of schedule levels.

3.4 **Preemptive tasks partitioning strategies**

Preemptive Deterministic Scheduling (PDS) ensures replica behavior while preserving concurrency. Replication is achieved by the threads and there is no communication between the threads. Performance improvement is achieved due to multithreading by exploiting concurrency in thread execution. Multithreaded replica shows non deterministic behavior. Only one physical thread can be scheduled at given time, so it shows poor scalability and performance. PDS schedules multiple threads at
the same time, so it shows five times throughput than Nondeterministic Preemptive Scheduling (NDPS) [11]. Optimal preemptive schedule subject to release date has been used to minimize the execution time on the homogeneous platforms. This strategy performs two steps. In the first step, minimize maximum completion time on the desired number of machines. In second step, it determine maximum lateness with respect to due dates for the jobs [12] on arbitrary number of machines.

Fast Preemptive Scheduling (FPS) [17] strategy simulates preemptive task execution at a very low overhead. It requires small runtime support in heterogeneous and homogeneous parallel computing environment [22]. Preemptive Task Scheduling (PTS) [18] strategy can also be used for homogeneous distributed memory systems.

3.4.1 Rate Monotonic-Decrease Utilization-Next Fit Scheduling: RM (Rate Monotonic)-DU (Decrease Utilization)-NFS (Next Fit Scheduling) [19] strategy does not suffer from execution time anomalies. It's a scheduling for periodically arriving tasks in multiprocessor environment. If the following assumption does not hold; (1) assume that a task meet its deadline if it does so when all tasks are executed at their maximum execution time, or (2) assume that a task meets its deadline, if it does so when all tasks arrive frequently, then this situation is known as scheduling anomalies. RM-DU-NFS is the combination of DU and NFS which is based on high system utilization bound.

3.4.2 Optimal Preemptive Scheduling: Optimal Preemptive Scheduling [20] is used to schedule different jobs on multiple parallel uniform machines. By assigning shortest remaining processing time jobs to the fastest available machine. Flow time has been minimized by serving jobs preemptions in increasing order of their remaining processing time. Shortest Remaining Processing Time on Fastest Machine (SRPT-FM) rule is also optimal for the discounted flow time criteria. Flow time is minimized by scheduling jobs according to the shortest remaining processing time on the fastest machine. Minimization of makespan has been achieved by using (LRPT-FM rule) Longest Remaining Processing Time on Fastest Machine [21] scheduling.

3.4.3 Preemption Threshold Scheduling: Preemption Threshold Scheduling (PTS) [23] disables preemption up to a specific level, called preemption threshold. Regular priority assignment to the arriving tasks is achieved by PTS. Tasks can preempt, only if the priority of the arriving task is higher than the threshold of running task.
3.4.4 Fixed Preemption Point Scheduling: In Fixed Preemption Point (FPP) scheduling [24] strategy, a task is assigned in a non-preemptive mode and a preemption can take place at a specific point of the code, which is known as preemption point. In this way, a task is divided into subtasks. If the highest priority task arrives then preemption is postponed till the next preemption point. So the tasks cooperate with each other by the preemption point.

3.5 Non-Preemptive Partitioning Strategies

Loose Synchronization Algorithms (LSA) uses non-preemptive deterministic scheduler to maintain multithreaded replica consistency. Optimal Finish Time (OFT) non preemptive strategy is known as NP complete with many uniform processors [14]. Largest Processing Time First (LPT) scheduling is near optimal for non-preemptive OFT.

3.5.1 Multiple Strict Bound Constraints scheduling: Multiple Strict Bound Constraints (MSBC) scheduling is non-preemptive static strategy for heterogeneous computing systems. It performs alternative task priority scheduling instead of Heterogeneous Earliest Finish Time (HEFT) scheme. It’s also used to enhance the performance of parallel computing applications on heterogeneous platforms due to macro data flow graph implementation [16].

3.5.2 Earliest Deadline First scheduling: EDF (Earliest Deadline First) Strategy, schedules periodic or sporadic tasks on single processor without preemption [25]. Periodic tasks are invoked after a certain time interval while sporadic tasks are invoked in arbitrary time but within the limited time constraints. EDF is universal for the set of periodic or sporadic tasks.

This chapter provides a suitable framework for comparing past work in the area of distributed parallel computing systems. Ideal performance of the strategy depends upon the requirement used for distributed parallel processing systems. From the brief discussion of scheduling strategies, it’s clear that there is no ideal strategy for all parallel computing systems. In this chapter, dynamic, preemptive and non-preemptive task partitioning and load balancing strategies had been briefly discussed. Advancement of the technology is a key factor for mapping heuristics of task partitioning strategies. A researcher decides his direction of the research problem
according to the existing strategies. A comparative analysis of the recent scheduling strategies provides the standard of existing work in the field of the parallel computing systems. In the static scheduling, most challenging direction is to extend DAG scheduling for the heterogeneous environment.

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Name of Strategy</th>
<th>Type of Strategy</th>
<th>Architecture</th>
<th>Remark</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PY (Papadimitriou and Yannakakis) Scheduling</td>
<td>Static</td>
<td>Homogeneous</td>
<td>DAG based scheduling. Not Optimal.</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>LCTD (Linear Clustering with Task Duplication)</td>
<td>Static</td>
<td>Homogeneous</td>
<td>DAG based scheduling. Not Optimal</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>EZ (Edge Zeroing)</td>
<td>Static</td>
<td>Homogeneous</td>
<td>DAG based scheduling</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>MCP (Modified Critical Path)</td>
<td>Static</td>
<td>Homogeneous</td>
<td>DAG based scheduling</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>ETF (Earliest Time First)</td>
<td>Static</td>
<td>Homogeneous</td>
<td>DAG based scheduling. Optimal Scheduling.</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Position Scan Task Scheduling (PSTS)</td>
<td>Dynamic</td>
<td>Heterogeneous</td>
<td>It can be used in centralized and decentralized manner.</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Evolutionary Task Scheduling</td>
<td>Static and Dynamic</td>
<td>Heterogeneous</td>
<td>Previous state information decides the scheduling environment</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>DPS (Dynamic Priority Scheduling)</td>
<td>Dynamic</td>
<td>Heterogeneous</td>
<td>Difference of top level and bottom level of nodes decide the priority of the tasks. Its DAG based scheduling.</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>DLS (Dynamic Level Scheduling)</td>
<td>Dynamic</td>
<td>Heterogeneous</td>
<td>DAG based scheduling. Priorities are assigning to the tasks on the basis of GDL (Generalized Dynamic Level) in ready list at every</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>LMT (Level Min Time) Scheduling</td>
<td>Dynamic</td>
<td>Heterogeneous</td>
<td>DAG based scheduling. During first phase level sorting has been used and in second phase greedy heuristic is applied for assigning priority.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------</td>
<td>---------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>FPS (Fast Preemptive Scheduling)</td>
<td>Preemptive</td>
<td>Heterogeneous and Homogeneous</td>
<td>Scheduling cost is very low</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>PTS (Preemptive Task Scheduling)</td>
<td>Preemptive</td>
<td>Homogeneous</td>
<td>Cannot be implemented in Heterogeneous</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PDS (Preemptive Deterministic Scheduling)</td>
<td>Preemptive</td>
<td>Heterogeneous</td>
<td>Multiple threads and there is no inter replica communication</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Optimal Preemptive Scheduling subject to release date</td>
<td>Preemptive</td>
<td>Homogeneous</td>
<td>Many step process</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>RM-DU-NFS Scheduling</td>
<td>Preemptive</td>
<td>Heterogeneous</td>
<td>Free from scheduling anomalies</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Optimal Preemptive Scheduling with discount flow time objective</td>
<td>Preemptive</td>
<td>Homogeneous</td>
<td>SRPT-FM rule is also optimal for discounted flow time criteria</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>SRPT-FM Scheduling</td>
<td>Preemptive</td>
<td>Homogeneous</td>
<td>Minimize flow time</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>LRPT-FM Scheduling</td>
<td>Preemptive</td>
<td>Homogeneous</td>
<td>Minimize makespan</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>PTS (Preemptive Threshold Scheduling)</td>
<td>Preemptive</td>
<td>Real Time System</td>
<td>It's used to reduce unnecessary preemption</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>FPP (Fixed Preemption Point) Scheduling</td>
<td>Preemptive</td>
<td>Real Time System</td>
<td>Non preemptive task make preemptions with the help of preemption point in code of the job</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>LSA (Loose Synchronization Algorithm) Scheduling</td>
<td>Non-preemptive</td>
<td>Heterogeneous</td>
<td>Its multithreaded replica strategy in which leader (thread) control to (follower) thread</td>
<td></td>
</tr>
</tbody>
</table>
Comparative analysis of various task partitioning and scheduling strategies has been discussed above in detail. This analysis is highly beneficial to choose the appropriate strategy under the different requirements and architectures.

### Table 2: Comparison of different task partitioning strategies under various architectures

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Type</th>
<th>Constraints</th>
<th>Complexity</th>
<th>Architecture</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFT (Optimal Finish Time)</td>
<td>Non-preemptive</td>
<td>Homogeneous</td>
<td>NP-Complete</td>
<td>Universal for the set of periodic or sporadic tasks</td>
<td>14</td>
</tr>
<tr>
<td>LPTF (Largest Processing Time First)</td>
<td>Non-preemptive</td>
<td>Homogeneous</td>
<td>Its near optimal to non-preemptive OFT</td>
<td>Homogeneous</td>
<td>15</td>
</tr>
<tr>
<td>MSBT (Multiple Strict Bound Constraints)</td>
<td>Non-preemptive, static</td>
<td>Heterogeneous</td>
<td>Its use macro data flow graph implementation</td>
<td>Heterogeneous</td>
<td>16</td>
</tr>
<tr>
<td>EDF (Earliest Deadline First) Scheduling</td>
<td>Non-preemptive</td>
<td>Real Time System</td>
<td>Universal</td>
<td>Homogeneous</td>
<td>25</td>
</tr>
</tbody>
</table>
REFERENCES


