Institute of Mathematics and Statistics

University of Sao Paulo

Analysis of Scheduling Algorithms on a Parallel Programming Context

Peter Ngugi Nyumu
Supervisor: Prof. Dr Alfredo Goldman

Acknowledgment

I would like to thank my supervisor Prof. Alfredo Goldman for his support and enormous amount of time he spent with me in this studies. My family, friends, colleagues, and finally the department of computer science.

Contents

1 Introduction

On most distributed computing platforms, a requirement to achieve high performance is careful scheduling of distributed application components onto available resources. Scheduling algorithms have been intensively studied as a basic problem in traditional parallel and distributed systems. Traditional scheduling models generally produce poor grid schedules in practice. The reason can be found by going through the assumptions underlying traditional systems [1], all resources resides within a single administrative domain, to provide a single and system image, the scheduler controls all of the resources, and the resource pool is invariant.

As the scheduling process on the context of operational systems have been extensively studied, we in this work intends to analyze how these algorithms can be applied in a grid computing context, specifically in Master/Slave scheduling model.

2 Categories of Scheduling Algorithms

We have different categories of scheduling algorithm, which rely on the architectural aspect of the computer or the linkage of the computers.

2.1 Uniprocessor Scheduling Algorithms

Uniprocessor is divided into off-line and on-line algorithms.

Off-line algorithms generate scheduling information prior to system execution and the system utilizes this information during the runtime. In almost all cases ordering of the execution process is required.

One of the desired outcome in off-line algorithms is the cost reduction of the context switches caused by the preemption. This can be done by carefully choosing algorithms which does not result in a high numbers of preemptions. Also it is desirable to increase the chances that a feasible schedule can be found.

Off-line algorithms are good for applications where all characteristics are known a priori and change very infrequently. This include characteristics like execution times, deadlines, and ready times. On the down side off-line algorithms need large amount of off-line processing time to produce the final schedule, and also are usually inflexible. The algorithms can not handle an environment that is not completely predictable. A major advantage of off-line scheduling is significant reduction in run-time resources, including processing time, for scheduling. How-ever, since it is inflexible, any change requires recomputing the entire schedule. The real advantage of off-line scheduling is that in a predictable environment it can guarantee system performance.

On-line algorithms generate scheduling information while the system is running. The algorithms require a large amount of run-time processing. On-line algorithms have a serious problem which can occur related to priority based preemptive. This can happen when a lower priority task is using some resources which are required by a higher priority task, which may lead to blocking of a high priority task.

On-line scheduling advantage is that there is no requirement to know tasks characteristics in advance and they tend to be flexible and easily adaptable to environment changes. Due to lack of knowledge of the characteristics in advance, it severely restricts the potential for the system to meet timing and resource sharing requirements.

On-line scheduling algorithms can be divided into Static-priority based algorithms and Dynamic-priority based algorithms, which are discussed as follows.

2.1.1 Static-priority based algorithms

On-line Static-priority based algorithms may be either preemptive or non-preemptive. Preemptive algorithms are common in practice, and ease in sharing of resources in a fair way, on the other hand non-preemptive algorithms aren't flexible in some areas like taking care of priorities in the processing of tasks.

On-line static-priority based algorithms have about two main disadvantages, low processor utilization and poor handling of aperiodic and soft-deadline tasks.

The other feature in the static-priority scheduling of periodic systems, is that all the jobs generated by an individual task are required to be assigned the same priority, this priority is different from the priorities assigned to jobs generated by other tasks in the system. As a result, the run-time scheduling problem becomes a problem of associating a unique priority with each task in the system. Some of the specific results we have are as follows [14]:

- The Rate Monotonic assignment algorithm, which is for implicit deadline synchronous periodic task systems. It assigns priorities to tasks in inverse proportion to their *period* parameters with ties broken arbitrarily, is an optimal priority assignment;
- The Deadline Monotonic priority assignment, which is for constrained deadline synchronous periodic task sets. It assigns priorities to tasks in inverse proportion to their *deadline* parameters with ties broken arbitrarily, is an optimal priority assignment;
- The computation complexity of determining an optimal priority assignment remain open, for the constrained deadline periodic task sets.

2.1.2 Dynamic-priority based algorithms

Dynamic-priority based algorithms are flexible, requires large amount of on-line resources, its also good to note that many dynamic priority algorithms contain some off-line components. The off-line components help to reduce the amount of on-line resources required while still retaining the flexibility of a dynamic algorithm. With the utilization of spare processing capacity to service soft and aperiodic tasks, dynamic-priority based algorithms provide better response to aperiodic tasks or soft tasks while still meeting the timing constraints of hard periodic tasks.

When it come to prioritizing of jobs, the dynamic-priority scheduling algorithms have no restrictions upon the manner in which priorities are assigned to individual jobs. The earliest deadline first scheduling algorithm (EDF) is a dynamic-priority algorithm, which at each instant in time chooses to execute the currently active job with the smallest deadline, is an optimal scheduling algorithm for scheduling arbitrary collections of independent real-time jobs in the following sense. EDF is the algorithm of choice, since any feasible task system is guaranteed to be successfully scheduled, this solve the run-time scheduling problem for preemptive uniprocessor dynamic-priority scheduling. [14].

There are other dynamic algorithms, such as least laxity first algorithm, which will be studied later in this work.

2.2 Multiprocessor Scheduling Algorithms

In multiprocessor scheduling algorithms, they is more than one processor available to execute the tasks. The Pfair scheduling [28] is one of the few known optimal methods for scheduling tasks on multiprocessor systems. However, the optimal assignment of tasks to processors is, in almost all practical cases, an NP-hard problem.

The following assumptions may be made to design a multiprocessor scheduling algorithm:

- job preemption is permitted
- job migration is permitted
- job parallelism is forbidden

Scheduling theorists distinguish between at least three different kinds of multiprocessor machines:

- identical parallel machines

- uniform parallel machines
- unrelated parallel machines

Multiprocessor scheduling techniques fall into two general category:

- Global scheduling algorithms

  Global scheduling algorithms store the tasks that have arrived but do not finish their executions in one queue which is shared among all processors.

- Partitioning scheduling algorithms

  Partitioning scheduling algorithms partition the set of tasks such that all tasks in a partition are assigned to the same processor. Tasks are not allowed to migrate, hence the multiprocessor scheduling problem is transformed to many uniprocessor scheduling problems.

2.3 Scheduling Algorithms in Computer Networks

There is data and control path as two groups of requirements when it comes to different level of service in the network. To enforce different service we deploy the data path mechanism, responsible for classifying and mapping user packets to their intended service class as well as controlling the amount of network resources that each service class can consume. To enable users and network agreement on service definitions by identifying which user is entitled to a given service also letting the network appropriately allocate resources to each service we use the control mechanisms.

The quality of service in the network is greatly impacted by the nature of scheduling mechanism which are employed to link the network. The basic function of the scheduler is to arbitrate between the packets that are ready for transmission on the link. In network transmission there are various scheduling policies which are considered to be able to choose the best scheduling algorithm;

- The performance guarantee given by the scheduling algorithm.
- The efficiency of the algorithm in enforcing services.
- The complexity of the algorithm, in the sense of the number of operation per packet transmission.
- The basic mechanism and parameters used by the algorithm to make scheduling decisions.
- The flexibility of the algorithm in handling traffic in excess of the amount for which the service guarantees have been requested.

3 Scheduling Problems

In the off-line setting, there is no distinction among the design of scheduling algorithms and checking feasibility of a task system. The scheduling algorithms provides a feasible sequence of jobs where tasks are always completed by their deadlines. In the on-line setting, the real-time scheduling theory focuses on two different scheduling problems:

- the feasibility analysis problem: given a task system and a scheduling environment, determine whether there exists a schedule that meets all deadlines.
- the run-time scheduling problem: given a task system that is known to be feasible, determine a scheduling algorithm that schedules the system to meet all deadlines, according to the supported scheduling environment.

There are three different approaches to check the feasibility / schedulability of a task system:

- Simulation of a scheduling algorithm until the task system is in the periodic state. This interval of time is called the feasibility interval. These tests can be performed if the scheduling algorithm is robust.
- Utilization factor based analysis: these tests are usually polynomial-time algorithms that provide sufficient schedulability conditions (Liu, Layland (1973)).
- Time-demand based analysis: these tests determine the worst-case workload of jobs within an interval of time. Simple extensions of these tests allow to compute the worst-case response times of tasks.

4 Scheduling in Operating System

The scheduling refers to the way processes are assigned to run on the available processor. The assignment is carried out by software known as a scheduler or is sometimes referred to as a dispatcher. The scheduler has the following function; switching context, switching to user mode and jumping to the proper location in the user program to restart that program.

The context switch which is a process of storing and restoring the context of the CPU, such that multiple processes can share a single CPU resource. This process enables preemption to occur thus helping in satisfying scheduling policy priority constraint. The period of time for which a process is allowed to run in a preemptive multitasking system is generally called the time slice, or quantum. The scheduler is run once every time slice

to choose the next process to run for example an algorithm like Round-Robin use context switch technique as later discussed.

In a context switching, the state of the first process is saved, so that, when the scheduler gets back to the execution of the first process, it can restore this state and continue. The registers the process has been using, the program counter and any other operating system data are included as the state of the process, which are stored in process control block(PCB). When the execution of the process get its turn to continue processing, the PCB is loaded.

To take care of different constraints involved in scheduling such as priority, we have, time the processor is available, preemption or non-preemption, scheduling algorithms which have different properties. The following criteria are used to in comparing different kinds of scheduling algorithms.

- CPU Utilization. This is the capacity to keep the CPU busy as much as possible as long as there are jobs to process.
- Throughput. A measure of work in terms of the number of processes that are completed per time unit.
- Turnaround time. The interval from the time of submission of a process to the time of completion.
- Waiting time. The sum of periods spend waiting in the ready queue.
- Response time. The time from the submission of a request until the first response is produced.
- The best optimization occur when it is possible to maximize CPU utilization and throughput and to minimize turnaround time, waiting time, and response time.

5 Some Operating Systems Scheduling Algorithms

Lets look at some of the scheduling algorithms;

5.1 Round Robin Algorithm (RR)

Round robin model is a time sharing processor studied by Kleinrock. There is time segmentation in Q seconds in length. At the end of every interval, a job arrives with a XQ probability, with the average number of arrivals per second in X. The procedure of servicing is a first come first service based, the server picks the first job at the queue,

process exactly Q seconds. If the job is finished it leaves the system, if not it join at the end of the queue. Its starvation free. It handle process with no priority.

In process scheduling, the tasks which the size are strongly varying may not be desirable to use. This problem is solved by introducing the politics of time sharing, giving each task a time slot called quantum. The task is interrupted if the time slice is over and give chance to the next task in the queue.

In Data scheduling where we have fairly equal data packets the round robin with windows, achieves the max-min fairness. This is a simple technique of allocating link capacity among competing sessions in a packet network. if we have allocation r as the function that assigns each session a rate consistent with its demand and the link capacities, it satisfies the max-min fairness criterion if no other allocation has a rate list that is lexicographically greater than the rate list of r, which can be described as fairness. This means that the smallest rate assigned to any session by r is as large as possible and,subject to that constraint, the second- smallest assigned rate is as large as possible.[6].

5.2 Weighted Round Robin (WRR)

Flexibility queuing policy with priority concept, is the great motivation of weighted round-robin. The number of time slot for each class depends on the value of its parameter, known as weight factor. The WRR queuing policy guarantees a minimum service for lower priority classes even in times of high traffic loads of higher priority classes. It give fairness and control over the priorities by giving different weight for each class.

For each class-queue has a counter that specifies the number of cells that can be sent from it. The counter value is initially set equal to the weight value assigned to that class. Cells from various classes are sent in a cycle from these classes whose counter values are greater than zero, with one cell being sent for each unit time. After sending a cell, a class counter value is reduced by one. When the counter value and/or the queue length has reached zero in all classes, all counters are reset to their weight values. In this situation, burstiness in input traffic becomes a major factor in creating delay, for a class may at times have a higher cell arrival rate than its weight value for one counter reset cycle can support. In such a cycle the counter value will reach zero before all the cells in the queue have been sent and cell scheduling for this class will be suspended until the next counter reset. Cells that arrive then, when the counter value is less than the queue length, will have to wait until the next or an even later counter reset.

When it comes to networks, WRR queuing support flows of different bandwidth, each queue can be assigned a different percentage of the output port's bandwidth. Also it ensures that low priority queues are not denied access to buffer space and output port bandwidth.

Before packets are assigned to a queue, there are classified into various service classes. WRR queuing allows higher bandwidth queues to send more than a single packet each time that it is visited during a service round or, allows each queue to send only a single packet each time that it is visited.

Some of the benefit of WRR queuing include: The WRR queuing can be applied to high speed interface in both core and at the edge of the network since it can be implemented in hardware. To avoid bandwidth starvation the WRR queuing ensures that all classes have access of a given configured amount of network bandwidth. Also there is stability in the network and equitable management due to classification of traffic by service class.

The major limitation of weighted round-robin queuing is that it provides the correct percentage of bandwidth to each service class only if all of the packets in all of the queues are the same size or when the mean packet size is known in advance.

5.3 Fair Queuing Algorithm (FQ)

In this algorithm, the ability to control the promptness or delay, allocation somewhat independently of the bandwidth and buffer allocation is desirable, and we can say is a goal of this algorithm. Allocating bandwidth and buffer space in a fair manner. Fairness is defined as, the max-min fairness criterion states that an allocation is fair if;

- No user receives more than its request,
- No other allocation scheme satisfying condition above has a higher minimum allocation,
- the second condition remains recursively true as we remove the minimal user and reduce the total resource accordingly, $\mu_{total} \leftarrow \mu_{total} \tilde{n} \mu_{min}$,

the above definition, assume that all users have equal rights of the resources.

The way the algorithm works, it is simple to allocate buffer space fairly by dropping packets, when necessary, from the conversation with the largest queue. The round-robin service provides a fair allocation of packets-sent but fails to guarantee a fair allocation of bandwidth because of variations in packet sizes. To illustrate how, this unfairness can be avoided, consider where transmission occurs in a bit-by-bit round robin fashion. This service discipline allocates bandwidth fairly since at every instant in time each conversation is receiving its fair share. Where $R(t)$ denote the number of rounds made in the round-robin service in time t and $N_{ac}(t)$ denote the number of active conversations. Then, $\frac{\partial R}{\partial t} = \frac{\mu}{N_{ac}(t)}$, where μ is the linespeed of the gateway outgoing line. A packet of size P whose first bit gets serviced at time t_0 will have its last bit serviced P rounds later, at time t such that $T(t) = R(t_0) + P$. Let t_i^{α} be the time that packet i belonging to conversation

α arrives at the gateway, and S_i^α as time packet started and F_i^α as time packet finished. Thus we have $F_i^\alpha = S_i^\alpha + P_i^\alpha$, where P_i^α is the size of the packet [18].

Its good to note that the $R(t)$ and $N_{ac}(t)$, and the quantities F_i^α and S_i^α, depend on packet arrival time t_i^α, not on the actual packet transmission times.

In a preemptive version of this algorithm, newly arriving packets whose finishing number F_i^α is smaller than that of the packet currently in transmission preempt the transmitting packet.

To obtain some quantitative results on the promptness, or delay, performance of a single FQ gateway, we consider a very restricted class of arrival streams in which there are only two types of sources. There are FTP-like file transfer sources, which always have ready packets and transmit them whenever permitted by the source flow control, and there are Telnet-like interactive sources, which produce packets intermittently according to some unspecified generation process[18].

5.4 Start-time Fair Queuing (SFQ)

In Start-time Fair Queuing algorithm, each packet is associated with a start and finish tag, this packets are scheduled in the increasing order of the start tag, which is defined as $v(t)$ in time t of the packets. The algorithm is defined as follows [20];

- When a packet p_f^j arrives, its marked with $S(p_f^j)$ computed as follows

$$S(p_f^j) = maxv(A(p_f^j)), F(p_f^{j-1}), \text{ where } j \geq 1$$

  where $F(p_f^j)$ is the finish tag of p_f^j, which is defined;

$$F(p_f^j) = S(p_f^j) + \frac{l_f^j}{r_f}, \text{ where } j \geq 1$$

  where $F(p_f^0) = 0$, and r_f is the weight of the flow f.

- The server virtual time $v(t)$ at t, is defined to be equal to the start tag of the packet in the service at time t^1. Initially the server virtual time is 0. The $v(t)$ is set to the maximum of finish tag assigned to any packet that have been serviced at time t^1, at the end of a busy period.

- Packets are serviced in increasing order of the start tags.

The computation of $v(t)$ in SFQ is not expensive because it examine the start tag in the service only. The algorithm have a complexity of $O(logQ)$, where Q is the number of flow in a service.

There are two server models, namely Fluctuation Constrained server and Exponentially Bounded fluctuation server, this division of server is meant to cater for server which can be shared by multiple type of traffic with different priorities.

A server is a Fluctuation Constrained server with parameters $(C, \delta(C))$, if for all intervals $[t_1, t_2]$ in a busy period of the server, the work done by the server, denoted by $W(t_1, t_2)$ satisfies [20];

$$W(t_1, t_2) \geq C(t_1 - t_2) - \delta(C)$$

A server is an Exponentially Bounded Fluctuation server with parameters $(C, B, \alpha, \delta(C))$, if for all intervals $[t_1, t_2]$ in a busy period of the server, the work done by the server, denoted by $W(t_1, t_2)$, satifies [20],

$$P(W(t_1, t_2) < C(t_1 - t_2) - \delta(C) - \gamma \leq Be^{\alpha\gamma}, \text{ where } 0 \leq \gamma$$

For fairness guarantee, $| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} |$, is the bound shown right, for any interval in which both flows f and m are backlogged. This is achieved by establishing a lower and upper bound in $W_f(t_1, t_2)$, as follows;

- if flow f is backlogged throughput the interval $[t_1, t_2]$, then in the SFQ server;

$$r_f(v_2 - v_1) - l_f^{max} \leq W_f(t_1, t_2)$$

  where $v_1 = v(t_1)$ and $v_2 = v(t_2)$

- In the server during any interval $[t_1, t_2]$;

$$W_f(t_1, t_2) \leq r_f(v_2 - v_1) + l_f^{max}$$
$$\text{where } v_1 = v(t_1) \text{ and } v_2 = v(t_2).$$

To guarantee throughput and delay, the r_f is interpreted as the rate assigned to flow f. The are three theorem that help to establish the throughput and delay guarantee to both model of SFQ servers [20].

5.5 Weighted Fair Queuing Algorithm (WFQ)

Weighted fair queuing (WFQ) was developed by Lixia Zhang and by Alan Demers, Srinivasan Keshav, and Scott Shenke. WFQ is the basis for a class of queue scheduling disciplines that are designed to address limitations of the Fair Queuing (FQ) model.Fair queuing is the foundation for a class of queue scheduling disciplines that are designed to ensure that each flow has fair access to network resources and to prevent a busty flow from consuming more than its fair share of output port bandwidth. In fair queuing, packets are first classified into flows by the system and then assigned to a queue that is specifically dedicated to that flow. Queues are then serviced one packet at a time in round-robin order. The following are limitation of FQ:

- The objective of FQ is to allocate the same amount of bandwidth to each flow over time. FQ is not designed to support a number of flows with different bandwidth requirements.
- FQ provides equal amounts of bandwidth to each flow only if all of the packets in all of the queues are the same size. Flows containing mostly large packets get a larger share of output port bandwidth than flows containing predominantly small packets.
- FQ is sensitive to the order of packet arrivals. If a packet arrives in an empty queue immediately after the queue is visited by the round-robin scheduler, the packet has to wait in the queue until all of the other queues have been serviced before it can be transmitted.

With the above limitation offered by the FQ, the WFQ improve the scheduling service by supporting the flow with different bandwidth requirements by giving each queue a weight that assigns it a different percentage of output port bandwidth.WFQ also supports variable-length packets, so that flows with larger packets are not allocated more bandwidth than flows with smaller packets.

WFQ supports the fair distribution of bandwidth for variable-length packets by approximating a generalized processor sharing (GPS) system. While GPS is a theoretical scheduler that cannot be implemented, its behavior is similar to a weighted bit-by-bit round-robin scheduling. In a weighted bit-by-bit round-robin scheduling the individual bits from packets at the head of each queue are transmitted in a WRR manner. This approach supports the fair allocation of bandwidth, because it takes packet length into account. As a result, at any moment in time, each queue receives its configured share of output port bandwidth.

Each packet is classified and placed into its queue, the scheduler calculates and assigns a finish time for the packet , which is the order in which each packet would eventually be fully assembled is determined by the order in which the last bit of each packet is

transmitted. As the WFQ scheduler services its queues, it selects the packet with the earliest (smallest) finish time as the next packet for transmission on the output port.

The following are the benefits of weighted fair queuing It guarantees a weighted fair share of output port bandwidth to each service class with a bounded delay.

It ensures a minimum level of output port bandwidth independent of the behavior of other service classes, thus providing protection to each service.

Weighted fair queuing have the following limitations:

The WFQ have a highly clustered service, that mean any problem within the service class can negatively affect the performance of other flows within the same service class.

Due to serialization delay introduced by high-speed links and the lower computational requirements of other queue scheduling, minimizing delay to the granularity of a single packet transmission on high speed interface may be not be worth the computational expense.

Although the supported WFQ delay bounds may be better compared to other queue scheduling, it can sometime be quite large.

The interactive scan of the state on each packet arrival and departure require maintenance which can be expensive since the algorithm is complex.[9].

5.6 Self-Clocked Fair Queuing (SCFQ)

Self-clocked fair queuing algorithm is an alternative to fair queuing scheme which is much simpler and provide desirable performance. It is based on the notion of the system's virtual time, viewed as the indicator of progress of work in the system, except that the virtual time is referenced to the actual queuing system itself, rather than to a hypothetical system. The system's virtual time at any moment t may be estimated from the service tag of the packet receiving service at t .

The SFCQ algorithm looks as follows;

- F_k^i is tagged to each arriving packet P_k^i, before it is placed in a queue. The packets in the queue are picked up for service in an increasing order of the associated service tags.
- For each session k, the service tags are computed as follows;

$$F_k^i = \frac{1}{r_k} L_k^i + max(F_k^{i-1}, v(a_k^i)), \text{ where } i = 1, 2, \ldots \ k \in K, \text{ and } F_k^i = 0$$

- System virtual time $v(t)$, at time t, is defined equal to the service tag of the packet receiving service at that time. Thats is

$v(t) \triangleq F_l^j$, $s_l^j < t \leq d_l^j$, where for packet p_l^j, s_l^j and d_l^j are start and finish time of service.

- When there is no more packet in the queue, the algorithm set to zero the virtual time $v(t)$, and the packet counts i for each session k.

The importance of offering service to the packet with the lowest service tag in the queue, is to make the scheme equate the normalized services of all the sessions, regardless of how long each session has been backlogged or absent, leading to the accumulation of service credit by the absent sessions.

In order to check the fairness of queuing system S, it is not reasonable to compare the received normalized services $w_k^S(t)$, $k \in K$, with each other or with $v^S(t)$, since the service opportunities missed by sessions during their absence intervals should also be accounted for. The following definitions are motivated by this observation [19]:

The virtual time of session k in system S, $v_k^S(t)$, is defined as the sum of the missed and the received normalized services of k on system S,

$$v_k^S(t) \triangleq u_k^S(t) + w_k^S(t),\ k \in K$$

The service lag of a session k in system S is defined as the difference between the system's and the session's virtual times

$$\delta_k^S(t) \triangleq v_k^S(t) - v_k^S(t),\ k \in K$$

The following theorem lays the basis of the fairness of the SCFQ scheme;

Theorem: The service lag of each session k in the SCFQ system is bounded as follows;

$$0 \leq \delta_k(t) \leq \tfrac{1}{r_k} L_k^{max},\ k \in K.$$

where L_k^{max} is the maximum size of packets of session k.

with the above theorem, we have the following lemma which proves it;

Lemma: While a session is absent or each time a session becomes backlogged in the SCFQ system, its service lag is zero [19],

$$\delta_k(t) = 0,\ k \notin B(t), \text{ or } k \text{ becomes backlogged at } t.$$

5.7 Deficit Round Robin (DRR)

Deficit round robin algorithm is an algorithm proposed to tackle one of the main unfairness we find in round robin algorithm, when we have packets of different sizes used by different flows.

Round-robin is used to service the queues with a quantum of service attached to each queue, where if a queue was not able to send a packet in a previous round because of its packet size was too large, the remainder from the previous quantum is added to the quantum of the next round. Hence the deficit are kept track off, queues that were shortchanged in a round are compensated the next round.

To describe the fairness in DRR we define two measures; FM, which measures the fairness of the queuing discipline and $work$, which measures the time complexity of the queuing algorithm.

The work to process a process involves, enqueuing and dequeuing. For DRR to do what is expected, we have the following definitions [17].

- Work is defined as the maximum of the time complexities to enqueue or dequeue a packet.
- A flow is backlogged during the interval I of an execution if the queue for flow i is never empty during interval I.
- Let $FM(t_1, t_2)$ be the maximum, over all pair flow i, j that are backlogged in the interval (t_1, t_2), of$(\frac{sent_i(t_1,t_2)}{f_i} - \frac{sent_j(t_1,t_2)}{f_j})$, with f_i and f_j as some quantity settable by a manager, that expresses the ideal share to be obtained by the flow i.

In DRR we assume that we have f_i quantities, which indicate the share given to flow i, for each flow i is allocated Q_i worth of bits each round. Define $Q = Min_i(Q_i)$, meaning that the share f_i allocated to flow i is $\frac{Q_i}{Q}$. Packets coming from different flows are stored in different queues, let the number of bytes sent out for queue i in around k be $b_i(k)$.

Each queue i is allowed to send out packets in the first round subject to the restriction that $b_i(1) \leq Q_i$, if queue i is empty after being serviced, a state variable, DC_i is reset to zero, contrary to that DCP_i is set to $(Q_i - b_i(k))$.

In DRR algorithm for all i, $0 \leq DC_i < Max$ is an invariant which proves that the algorithm execute in the manner described[17].

The DRR service the queue in round robin manner. A round is one round robin iteration over the queues that are backlogged. By considering any execution on the DRR scheme in (t_1, t_2) as any interval, of any execution such that flow i is backlogged during (t_1, t_2). Having m as the number of round robin service opportunities received by flow i

during this interval, then we can see that the number of bytes sent on behalf of flow i is roughly mQ_i.

$$mQ_i - Max \leq sent_i(t_1, t_2) \leq mQ_i + Max$$

In terms of fairness in DRR algorithm, the following theorem shows that the fairness measure for any interval is bounded by a small constant. For an interval (t_1, t_2) in any execution of the DRR service discipline

$$FM(t_1, t_2) < 2Max + Q, \text{ remembering that } Q = Min(Q_i)$$

The size of the various Q variables in the algorithm determines the number of packets that can be serviced from a queue in a round, hence the latency of a packet and the throughput of the router is dependent on the value of the Q variables. We have that, the work of DRR is $O(1)$ if all $i, Q_i \geq Max$. If $Q \geq Max$, we have a guarantee of sending at least one packet every time we visit the queue, hence the worst case complexity is $O(1)$.

5.8 Elastic Round Robin (ERR)

The Elastic round robin (ERR), is an improved round robin algorithm which was first designed, for wormhole networks, it can be used in a wide variety of contexts whenever there is a shared resource that needs to be allocated fairly among multiple requesting entities [12].

Consider n flows, each flow is associated with queue which have packets. The scheduler dequeues packets from these queues according to a scheduling discipline and forwards them for transmission over an output link. The length of time it takes to dequeue a packet is proportional to the size of the packet however, to apply this work to wormhole networks, it is required that the scheduling algorithm not make any assumptions about the length of a packet prior to completely transmitting the packet.

With the algorithm, [12], the enqueue routine is called whenever a new packet arrives at a flow. The main part of the algorithm is the dequeue, which schedules packets from the queues corresponding to different flows.

With the use of linked list, which in this case is called activeList, the ERR scheduler serves the flow i at the head of this list. If the queue of the flow i becomes empty, it is removed from the list. this is done after serving the flow i. When the queue of flow i is not empty after it has received its round robin service opportunity, flow i is added back to the tail end of the list. Its good to note that the ERR assume the equal weight in all traffic flow, to improve on this we have another algorithm called weighted ERR.

The work complexity of the ERR scheduler is defined as the order of the time complexity with respect to n of enqueuing and dequeuing a packet transmission, this is by considering an execution of the ERR scheduling discipline over n flows. This work complexity of an ERR scheduler is $O(1)$ [12]

The fairness of a scheduling discipline is best measured in comparison to the GPS scheduling algorithm. The Absolute Fairness Bound (ABS) of a scheduler S, is the quantity used, defined as the upper bound on the difference between service received by a flow under S and that under GPS overall possible intervals of time. The metric used to analyze is called relative fairness bound, defined as the maximum difference in the service received by any two flows over all possible intervals of time. This metric is used due to the fact that the ABS is often difficult to derive analytically [12]

5.9 Credit Based Fair Queuing Algorithm (CBFQ)

The algorithm discussed here is based on a simple rate-based scheduling algorithm for packet-switched networks.With the use of counters set to keep track of the credits accumulated by each traffic flow, the bandwidth share allocated to each flow, and the size of the head-of-line (HOL) packets of the different flows, the algorithm decides which flow to serve next.

The bandwidth share S_i and a counter K_i, where i = 1, ... ,J is associated with traffic flow. The counter stores the number of transmission credits earned by the corresponding traffic flow. The decision on which packet is to be served next is based on the bandwidth shares, value of the counters, and the size of the head-of-line packets. It clear that the session which need short time to earn enough credits are served first as a consequence, it reduces the waste of bandwidth. The counter is reset to zero after every transmission [11]

CBFQ Algorithm:

1. Initialization phase

$$\forall \text{ l, l} \in 1 \ldots \text{ J,Set } K_i \leftarrow 0.$$

Operation

2. Let J = $j_i \ldots j_k$ such that queues j_i,... ,j_k are currently backlogged.
3. Let L_{jn} be the size of the HOL packet of queue $j_n \in J$
4. Sort the backlogged queues according to
$\frac{L_{j1}-K_{j1}}{S_{j1}} \leq \frac{L_{j2}-K_{j2}}{S_{j2}} \leq \ldots \leq \frac{L_{jk}-K_{l}jk}{S_{jk}}$

5. Transmit the HOL packet of queue j_1, and update the counter as follow
   $K_{jn} \leftarrow K_{jn} + \frac{L_{j1}-K_{j1}}{S_{j1}}$ - S_{jn}, $\forall\ j_n\ \epsilon$ J $\setminus\ j_1\ K_{j1} \leftarrow 0$

6. Go to 3

- **Fairness**

  Let J(t_1, t_2) denote the set of queues that are continuously backlogged during (t_1, t_2). $W_i(t_1, t_2)$ is defined as the amount of traffic transmitted from queue i during (t_1, t_2), and MAX_i, i = 1,...,J, as the maximum packet size of the stream.

  $$\forall l, l' \in J(t_1, t_2), \mid \frac{W_l(t_1,t_2)}{S_l} - \frac{W_{l'}(t_1,t_2)}{S'_l} \mid \leq \frac{MAX_l}{S_l} + \frac{MAX_{l'}}{S_{l'}}$$

  The fairness bound guaranteed by CBFQ is as expected the same as the ones guaranteed by the alternative algorithm such as SCFQ. This can be proven, by first proving that $0 \leq K_l \leq MAX_l$ at time t, and if we let $A_l(t_1, t_2)$ be the amount of credit earned by stream during (t_1, t_2), it can be seen that $W_l(t_1, t_2) \leq A_l(t_1, t_2) + K_l(t_1)$ - $K_l(t_2)$

- **Complexity**

  The order of the terms in step 4 of the algorithm above does not change, expect for the queue that has just been served. Thus no requirement to fully sort the queues in step 4 each time after a packet is served. Counter in step 5 can be updated parallel to other operation since its independent, hence the complexity is $O(1)$. Knowing the other operations amounts to $O(1)$. we come to conclusion that the complexity of the algorithm is $O(log(J))$ [11].

5.10 CBQF-F Algorithm

This is a version of CBFQ which deal with fixed packet size networks, which is in the ATM environment. In this case the packets are short an the transmission speed is very high. CBFQ-F sacrifices a small amount of fairness bound to achieve smaller processing overheads.

The algorithm is based on a set of counters K_i, i = 1, ... ,J, associated with each queue. Frames of cycle of service of variable length are defined without any prior knowledge of their length. At the beginning of each cycle, the backlogged queue j_1 whose share of bandwidth S_{j1} is the largest among active queues is chosen as the reference queue to calculate the relative share of bandwidth each backlogged queue should receive in the cycle. All the counters are increased by this relative share which is simply the ratio of the queue's share to S_j1. The algorithm operates as follows;

1. Initialization phase

   $\forall$l, l $\in$ 1... J,Set $K_i \leftarrow 0$.

2. Operation

   Let J = j_i... j_k such that queues j_i,...,j_k are currently backlogged and $S_{j1} \geq S_{j2} \geq \ldots \geq S_{jk}$.

3. Update the counter as follows:

   $K_{jn} \leftarrow K_{jn} + \frac{S_{jn}}{S_{j1}}, \forall\ j_n \in \text{J} \setminus j_1$

4. Let K = l ϵ J:$K_l \geq 1$

5. while K $\neq \emptyset$.

6. Transmit the HOL packet of queue l $\in$ K, and update the counters as follows:

   $K_l \leftarrow K_l$ - 1 if queue l is nonempty else $K_l \leftarrow 0$.

7. Let K = K $\setminus$ l

8. end while

9. Go to 3

- **Fairness**

  Consider the interval of time (t_1, t_2) and define j_1(t_1) and j_1 (t_2) as the reference queues at times t_1 and t_2 respectively. With the same definition as for CBFQ, we can prove the following fairness bound for CBFQ-F.

$$\forall t_1, t_2 \text{ and } \forall l, l' \in J(t_1, t_2), \mid \frac{W_l(t_1,t_2)}{S_l} - \frac{W_{l'}(t_1,t_2)}{S_{l'}} \mid \leq \frac{1}{S_l} + \frac{1}{S_{l'}} + \frac{1}{S^*_{l1}}$$

  where $S^*_{l1} \stackrel{\Delta}{=} min(S_{j1(t)}, S_{j2(t)})$

- **Complexity**

  In general, the bandwidth shares are allocated at the connection setup and remain constant during the lifetime of connection. There is a priority change in step 4 of the algorithm, hence time scale is larger than the packet level time scale. To avoid additional complexity in step 4, the elements of K are not sorted according to their counters values. This means the construction of K and the selection of the elements of K to receive service next are done by comparing the values of counters to 1. This can be implemented in parallel by the simple hardware device such as a gate and requires only $O(1)$ operation.

5.11 Earliest Deadline First Scheduling (EDF)

The earliest deadline first is also known as nearest deadline first, it uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority, while the lowest priority belongs to the task with the latest deadline. Priorities are dynamic, therefore periods of tasks can change anytime. The algorithm has the schedulability bound of 100 for all task sets. The CPU utilization of task T_i is computed as the ratio of its worst-case execution time E_i to its request period P_i.

A transient situation occurs when the worst case utilization is above 100 per cent, this may lead to a critical task failing in the expense of a lesser critical task. In this algorithm transient situation can occur, thus there is no guarantee of which task may fail [13].

5.12 Least Laxity First (LLF)

Its also known as the minimum laxity first. Laxity is the measure of flexibility of scheduling a task. It assigns higher priority to a task with the least laxity. The laxity $L_i(t)$, of a real time task T_i in time t, with task deadline $D_i(t)$ and $E_i(t)$ as the time of computation left, is defined as follows:

$$L_i(t) = D_i(t) + E_i(t)$$

The laxity calculated above assures that even if a task is delayed at most $L_i(t)$, it will still meet the deadline. Any task with zero laxity should be executed immediately without preemption, this assures that it will meet the deadline. On the other hand negative laxity indicates that the task will miss the deadline, no matter when it is picked up to execute.

The disadvantage of least laxity first is that it is impractical to implement due to laxity ties, this is when two or more tasks have the same laxities, leading to frequent context switches among the corresponding tasks, as a result there is degrading of the system performance [13].

5.13 Modified Least Laxity First (MLLF)

This algorithm was proposed to solve the problem of the least laxity first algorithm by reducing the number of context switches. This happen by deferring the context switch which is known as laxity inversion. It is good to note that it is safe even if the laxity tie occurs. Laxity inversion applies to the duration that the currently running task can continue running with no loss in schedulability, even if there exist a task (or tasks) whose laxity is smaller than the current running task.

As a result the algorithm avoid the degrading of the system performance. When there is no laxity ties, the algorithm perform as least laxity first. If it exist laxity ties, running

task continue running with no preemption as long as the deadline of the tasks is not missed [13].

5.14 Maximum Urgency First Algorithm (MUF)

This algorithm is a mixed priority scheduling since it is a combination of fixed and dynamic priority scheduling. MUF help to solve the transient overload problem, by allocating urgency to each task which is defined by two fixed priorities (criticality and user priority), and dynamic priority that is inversely proportional to laxity. Criticality has high precedence over the dynamic priority while user priority has lower precedence than the dynamic priority.

The assignment of priorities occurs in two phases. The first phase, is the assigning of static priorities to tasks, this priorities once assigned do not change when the system start. The first phase include the following steps;

- Sorting tasks from shortest to longest period. To guarantee no failure even when the transient overload occurs, it define the critical set as the first N task, such that the load factor of the CPU does not exceed a 100 per cent.
- All tasks in critical set are assigned high criticality, while the rest are considered to be low criticality.
- Every task in the system is assigned an optional unique user priority.

The second phase deals with the run-time behavior of the MUF scheduler, which select task for execution. The algorithm is executed when the task arrive at the ready queue, as follows;

- If there is only one highly critical task, it is picked and executed.
- Incase of more than one highly critical task, select the one with the highest dynamic priority. Note that the task with the least laxity is considered to be the one with the highest priority.
- Incase of more than one task with the same laxity, the task with the highest user priority is selected.

One of the disadvantage with this algorithm is the possibility failing a critical task in certain situation, since the rescheduling operation is performed whenever a task arrive to the ready queue. A task with a minimum laxity may be selected whose remaining execution time is greater than the remaining time to another task laxity. This problem

occurs due to the rescheduling operation whenever a new task is added to the ready queue. Modified maximum Urgency First algorithm (MMUF), is an algorithm proposed to deal with this problem.

In MMUF, there is an introduction of an importance parameter, which is used in place of task request intervals to create the critical set. The importance parameter is a fixed priority defined as user priority or any other optional parameter, which expresses the task critical degree [13].

5.15 Gang Scheduling

Gang scheduling was introduced by Ousterhout in the context of the Medusa system on Cm[21]. He explained the intuition behind using Gang scheduling to allow efficient use of busy waiting for fine-grain synchronization, and developed three algorithms for its implementation. However no analysis of the performance implications was done.

Less Research have been done, to explore the potential of gang scheduling since then, Blasewicz et. al. developed off-line algorithms for gang scheduling, essentially using dynamic programming [22]

6 Scheduling in Parallel Computing

A parallel computer is a connected configuration of processors and memories. The choice space available to a computer architect includes the network topology, the node processor, the address space organization, and the memory structure. These choices are based on the parallel computation model, the current technology, and marketing decisions.

In parallel computing the scheduling heuristics can be grouped into two categories: online mode and batch-mode heuristics. In On-line mode the task is scheduled on to a machine as soon as it arrive at the scheduler. On the other hand, in batch-mode, tasks are not scheduled onto the machines as they arrive, instead they are collected into a set that is examined for scheduling at a prescheduled times called scheduling event.

Parallel computing rely on multithreading for efficient execution, thus relying on the schedule of the threads among the processors. Two models used in parallel machine are Cilk and Kaapi, both of them use work stealing algorithm for execution.

Kaapi is based on a global address space called global memory and allows to describe data dependencies between tasks that access objects in the global memory. In Kaapi a thread represents a computation. Thread interface in KAAPI is a subset of POSIX threads except for the functionalities about the management of signal and cleanup routines. A thread could be an active control flow executing some computation or are inactive. At deployment step, one of the processes is designed to be the main process which starts the

thread that executes the main task. This thread is called the main thread of the distributed execution. For each user level, the thread runs in a non-preemptive manner. A suspension occurs, in the case of a thread executing a blocking instruction, such as locking an already locked mutex or waiting for a condition variable. Then, the kernel thread is said idle and it switches to the scheduler thread which is responsible to assigning thread. The Work-stealing algorithm first tries to find a suspended thread ready for execution, like a standard thread scheduler. In case of success, the elected thread resumes its execution on the idle kernel thread. Else, the scheduler thread chooses at random a thread and call, if specified, the steal function attribute, as defined in previous section. If the function returns true then the newly created thread is scheduled on the kernel thread which becomes active[25].

A Cilk multithreaded computation can be viewed as a directed acyclic graph that unfolds dynamically. A Cilk program consists of a collection of Cilk procedures, each of which is broken into a sequence of threads, which form the vertices of the directed acyclic graph. Each thread is a nonblocking C function, which means that it can run to completion without waiting or suspending once it has been invoked. As one of the threads from a Cilk procedure runs, it can spawn a child thread which begins a new child procedure. A spawn is like a subroutine call, except that the calling thread may execute concurrently with its child, possibly spawning additional children. A thread cannot spawn children and then wait for values to be returned, since threads cannot be blocked. Instead, the thread must additionally spawn a successor thread to receive the children's return values when they are produced. A thread and its successors are considered to be parts of the same Cilk procedure. Cilk computation unfolds as a spawn tree composed of procedures and the spawn edges that connect them to their children, but the execution is constrained to follow the precedence relation determined by the dag of threads. Cilk's scheduler uses the technique of work stealing in which a processor (the thief) who runs out of work selects another processor (the victim) from whom to steal work, and then steals the shallowest ready thread in the victim's spawn tree. Cilk's strategy is for thieves to choose victims at random[26].

7 Scheduling in Cluster

Cluster is a collection of inter-connected and loosely coupled stand-alone computers working together as a single, integrated computing resource. Clusters are commonly, but not always, connected through fast local area networks. Clusters are usually deployed to improve speed and/or reliability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability.

Cluster deploy the Master-Slave model of scheduling with the use of on-line mode for mapping[27]. The master node is the unique server in cluster systems. It is responsible

for running the file system and also serves as the key system for clustering middleware to route processes, duties, and monitor the health and status of each slave node. A slave node within a cluster provides the cluster a computing and data storage capability.

The application model best suited for cluster environment is the Parallel Tasks (PT) model[23]. A PT is a task that regroups elementary operations, typically a numerical routine or a nested loop, and which contains in itself enough parallelism to be executed by more than one processor. usually the PT tasks use on-line mode, and the time can be estimated or not depending on the type of applications. Parallel Tasks can be classified as rigid jobs or moldable jobs. Rigids jobs, the number of required processors is fixed by the user at the submission time. On the otherhand, in the case of Moldable jobs, the number of processors is determined by the scheduling algorithm before execution time. In both cases the number of the processor does not change until the execution of job is finished.

Master/Slave model is used in cluster computing, this is done in a way one of the machine in the cluster is given the role of a scheduler, hence a master, and the rest of the machines within the cluster are automatically slaves, here the master machine within the cluster determines the way to process tasks, which depend on the scheduling algorithm deployed.

8 Scheduling in Grid

Grid computing are a technology closely related to cluster computing. The key differences between grids and traditional clusters are that grids connect collections of computers which do not fully trust each other, and hence operate more like a computing utility than like a single computer. In addition, grids typically support more heterogeneous collections than are commonly supported in clusters.

Grid computing is optimized for workloads which consist of many independent jobs or packets of work, which do not have to share data between the jobs during the computation process. Grids serve to manage the allocation of jobs to computers which will perform the work independently of the rest of the grid cluster. Resources such as storage may be shared by all the nodes, but intermediate results of one job do not affect other jobs in progress on other nodes of the grid.

In grid computing, many unique characteristics make the design of scheduling algorithms more challenging, including heterogeneity and autonomy, performance dynamism, resource selection and computation-data separation. Lets try to look at this unique characteristics [3]:

- **Heterogeneity and Autonomy**

In Grid computing, resources are distributed in multiple domains in the Internet, hence the computational and storage nodes, and the underlying networks connecting them, are heterogeneous. The heterogeneity results in different capabilities for job processing and data access.

Contrary to grid computing, in traditional parallel and distributed systems, scheduler has full information about all running or pending tasks, resource utilization, and it as well manages the task queue and resource pool, this because in this case, the computational resources are usually managed by a single control point. On the other hand, Grid scheduler does not have full control of the resources, remembering that the resources in this case are usually autonomous, resulting in the diversity in local resource management and access control policies. Application types, resource requirements, performance models, and optimization objectives, are parameters which represents the heterogeneity and autonomy on the Grid user side.

- **Performance Dynamism**

  Grid schedulers work in a dynamic environment, that means the performance of available resources is constantly changing. The change comes from site autonomy and the competition by applications for resources leading to Grid resources not being dedicated to a Grid application. For example job priority and the network traffic flow, which will always affect the performance of the scheduler. This make it hand to evaluate the grid performance using the tradition method.

- **Resource Selection and Computation-Data Separation**

  The cost of data staging in traditional systems can be neglected or can be determined before execution, which is usually a constant. This is due to the fact that the input sources and output destinations are determined before the application is submitted. The grid environment is completely different due to the large number of heterogeneous computing sites and storage sites connected via wide area networks, the computation sites of an application are usually selected by the Grid scheduler according to resource status and certain performance models.

  In Grid computation, the inter-domain communication cost cannot be neglected since the bandwidth of the underlying network is limited and shared by a host of background loads. In most cases the grid applications are data intensive, making it important to consider the data staging cost. This make it tricky because the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site. Hence, there a challenge to design and implement efficient and effective grid scheduling systems.

9 Adapting the Algorithms in a Master/Slave Environment

In parallel and distributed computing we take the operating system scheduling algorithms and try to adapt them to work in differents kind of models. In out case we look how to adapt these algorithms in a Master/Slave model with processes arriving in sequential manner, in a heterogeneous sets.

In Master/Slave environment, there is no need of preemption of tasks, since slaves are independent and consequently do not need to be running simultaneously. In this environment, we can work with or without communication among master and slaves, clearly when we have the communication we have more information, and for any waiting task we can be able to know when we will have available slave or slaves, the downside is that, master-slave communication just add cost of processing.

Round Robin algorithm, is an easy algorithm to adapt in Master/Slave environment. It simply sends a task to each slave one by one according to the order of arrival. Here it's easy to see that no need of preemption since every process is being allocated to an independent slave and thus giving each process time to be processed. At the same time it can occur with or without the communication among the master and the slaves, incase of communication there is an extra cost paid, but also the master has more information which will help in the allocation of the next process.

Weighted round Robin (WRR), as discussed above it attach a weight factor to establish the priority concept which lack in the round robin scheduling operating system algorithm. In Master/Slave environment, in the case where we have an available slave the WRR will function like a normal round robin, arriving and being allocated to one of the slave. In a situation whereby there is no slave available for processing, then the weight factor can be considered, in a manner that when the slave is available, the process with largest weight factor will get the priority of getting the chance to be allocated to the available slave.

Deficit Round Robin (DRR), as discussed earlier, this algorithm try to address the issue of fairness when we have different size of tasks. If there is slave machine available, then the master machine will not worry about the fairness issue, since the whole point is to be balanced when allocating resources to different process, and inn this case can start processing without delay. On the otherhand, its likely to have a situation where, there is no slave available, hence have to weight for the slave availability, in this case since we are not applying any priority, then which ever the size of the task, when the slave is available, then the master will allocate the process and since preemption is not needed then the processing of the task is guaranteed to take place.

Elastic Round Robin (ERR), designed to fairly serve multiple requesting entities. In Master/Slave environment, when there is a slave machine available at the arrival of any request by the master, with or without priority then the algorithm will work with no

problem. In the situation of no slave available, then like scheduling in operating system, the algorithm can use the linked list as an activelist to store the arrival of the task, then the master will serve the linked list from the header, in that manner it will ensure that multiple users can utilize the resources. In case of a situation where we have priority, and no slave available, then we can apply the weighted ERR, where we have weight factor attached to the task, the large the weight factor the higher the priority, that way when the slave is available, the master will first schedule a task with higher priority.

Fair Queuing Algorithm, the idea behind the fair queuing algorithm is that, the process don't get more resources (i.e. time for processing), than required and also assume that all the users have equal rights of resources. In Master/Slave environment, its easy to see that the fairness concept is already incorporated in it, since the master will always allocate a user a slave to process a task and as soon the slave have finished it returns the results to the master, and make itself available for the next task. When there is no slave available, the tasks will simply wait for the next available slave.

Start-time Fair Queuing (SFQ), each task is associated with a start and finish tag. SFQ is an interesting algorithm, in a situation we have a waiting list of tasks to be processed and we would like to know the likely time the slave machines will be available. To be able, to apply this algorithm efficiently, its is important to have master-slave communication.

Weighted fair queuing (WFQ), though it is designed to take care of different size of bandwidth, in a situation where there is a slave machines available, the WFQ, will behave like the fair queue algorithm, such that the master will allocate the task to a slave available. Since the scheduling environment is a non-preemptive one, then the size of the bandwidth will not affect in any way the functionality of the algorithm.

Self-Clocked Fair Queuing (SCFQ), use the system virtual time to put a tag on task, it is well applicable in a situation where we have master-slave communication, and the slave can notify the master the exact time the task start processing. The information can be used to estimate how long the task would take to finish, but as indicated before that mean there will always be a cost of communication.

General CBFQ, the accumulation of credit is applicable in a situation we have waiting tasks, that mean that we don't have slave available for processing. Different bandwidth of the network, can be served with fairness by the master schedule, again if is no need to wait for resources, meaning that the slave are available, then there is no need to calculate the credit tag.

Earliest deadline first (EDF), use a deadline like a priority property to schedule the task. Like many algorithm, when there is a slave machine available, then the master will just schedule the task. EDF will apply the deadline property when there is no slave machine available and there are more than one task waiting to be scheduled.

Least Laxity First, it is interesting to calculate the laxity component when we have

more than one waiting task to be allocated to slaves. That way we can know the flexibility, the master scheduler can have to schedule any given task, to the contrary if we have slave available there is no need to calculate the the laxity component which also mean less cost.

Modified Least Laxity First, is designed to reduce number of context switches in operating system scheduling, but in the case of Master/Slave environment, there is no preemption, hence it cannot be adapted in this context.

Maximum Urgency First (MUF), as seen above, MUF help to solve the transient overload problem, by allocating urgency to each task which is defined by two fixed priorities (criticality and user priority), and dynamic priority that is inversely proportional to laxity. This algorithm can be adapted this manner in a situation where tasks have to wait for unavailable slaves. That way the task with highest priority will get more urgency and vice versa.

10 Appendix

In this section we look at some important scheduling related materials, this include, the scheduling resources, the taxonomy of the operating system as well as for grid scheduling.

10.1 Resource Scheduling

Scheduling algorithms are meant to help in resource sharing. This resources are divided in to two divisions,this is, *processing resources*, resources which are or not needed together with the processor (machine) during the processing of a task, and *Input-output*, resources are the other kind of resources that are needed either before or after the processing of a task on a processor.

There are three categories of resources from the resource constraints viewpoint include:

- renewable - resources once used may be used again after being released from a task (total usage).
- nonrenewable - resources once used by some task cannot be assigned to any other task (total consumption).
- doubly constrained - if both total usage and total consumption are constrained.

  There are two resource categories from the resource divisibility viewpoint include:
- discrete - resource that can be allocated to tasks in discrete amounts from a given finite set of possible allocations.
- continuous resources - resource can be allocated arbitrary, a priori unknown, amounts from a given interval.

We are going to look at discrete and renewable resources in this work

- **Scheduling with Processing Resources**

  In this area we will consider a case of independent tasks and non-preemptive scheduling. The problem of scheduling unit-tasks on two processors with arbitrary resource constraints and requirements can be solved optimally by the Garey and Johnson algorithm for P2 $\mid$ res. . . , $p_j = 1 \mid C_{max}$. The key idea of the Algorithm is to have a correspondence between maximum matching in a graph displaying resource constraints and the minimum-length schedule.

  Limiting the resources to one process can lead to a faster algorithm by optimal schedule, which would be produced by ordering tasks in non-increasing order of the

resource requirements and assigning tasks in that order to the first free processor on which a given task can be processed because of resource constraints, leading to solving the problem in $O(nlogn)$. Further more if the resource requirement are allowed only for 0 -1, the same problem can be solved by $O(n)$ time.

If we consider the P|res..., $p_j = 1|C_{max}$, this is when the number of resources types, resource limits and resource requirement are fixed. The problem is solvable in linear time, even for an arbitrary number of processors. We can generalize this problem to m processors, and consider the case with non-unit processing times and task belonging to a fixed number of k of classes only, thus solving the problem using dynamic programming, proposed by blazewics et al [9]. The time complexity of the algorithm depends on the product of the number of states and the maximum number of decisions which can be taken at the state of the algorithm. With fixed number of task classes k and of processors m, the complexity can be bounded to $O(n^{k(m+1)})$. We can have a complexity of $O(n^{k(p+1)})$, in case of unrestricted number of processors, but with fixed upper bound on task processing times p is specified.

The other problem, which be considered is P | res..., $p_j = 1$ | C_{max}, which follows that when we consider the non-preemptive case of scheduling of unit length tasks we have five polynomial-time algorithms, and since its a non-preemptive problem, then we have NP-hard.

- **Management of Processing Resources**

  Job scheduling and tool management problem are two problem which are encountered in resource management. For efficient management, it is important to find the job sequence that minimizes the total number of tool switches. It's good to note that the two problems are related and is NP-hard for a magazine capacity of two tools. If the job sequence is given (the offline version), then the tool switching problem has simple optimal solutions. The optimal tool management is described as *k-server problems*. It is shown that there is an analogy between the tool switching problem and the paging problem in computer memory systems.

  There are two main application areas for server problems in computer and manufacturing systems, the first one is two level computer memory system, where a computer information stored in n pages of memory, k of which are fast access and the remaining $n - k$ are slow access memory. The second one is the tool-switching problem, where we have a two-level tooling system consisting of the tool magazine of an numerically controlled machine which has room for k tools and a main tooling reservoir which can contain $n - k$ tools.

- **Scheduling with Input/Output Resources**

  This are kinds of resources which are required by a job at the beginning or at the end

of the job processing. To describe the input/output resource in which theoretical result exist, we describe robotic cells. Robotic cells consist of *m-machines* arranged in a circular layout and served by a single central robot(the input/output resources). A line machining castings for truck differential assemblies is described in the form of a three-machine robotic cell where a robot has to transfer heavy mechanical parts between large machines.

The original application has the form of a flow shop to produce a large number of castings. For flow shop with m greater than 3 machines, the problem of robotic scheduling turn to be NP-hard.

Cyclic robot moves for the production of the parts can also be considered in this area of input/output resources. cyclic robot define a $k-cycles$ as a production cycle of exactly k parts, this k parts enter the system at M_0, and leaves the system at M_{m+1} and each time the robot executes the $k-cycle$, the system returns to the small state.

10.2 Taxonomy in Operating system

Taxonomy are created to help in assessing different features of the operating system as well as systematics way of considering the system failures and their affects on application programs. The taxonomy considers the facilities of any operating system at three levels, the functions, a number of supporting services, and the lowest level the individual application programming interface routines[24].

The functions include;

- Provision of secure and timely data flow,
- Controlled access to processing facilities.
- Provision of secure data storage and memory management.
- Provision of consistent execution state.
- Provision of health monitoring and failure management.
- General provision of computing resources

The services include; Partitioning, Main memory management, Data and program loading, Intra-partition communication, Inter-partition communication, External communication, Scheduling, Event notification, Timing watchdog, File-store management, Resource sharing and locking, Initialization and configuration management.

10.3 A Taxonomy of Grid Scheduling Algorithms

As mentioned in the earlier section, scheduling problem in grid computation becomes more challenging because of some unique characteristics. In this section, we look at the taxonomies of grid scheduling which will lead to the discussion of scheduling algorithms [2].

- **Local vs. Global**

  The local scheduling uses a single processor to allocate and execute the tasks, while in global scheduling the the system allocate processes to multiple processors to optimize a system-wide performance objective. Grid scheduling falls in the category of global scheduling.

- **Static vs. Dynamic**

  Under global scheduling we have a choice between static and dynamic scheduling, this choice indicates the time at which the scheduling or assignment decisions are made. In static scheduling, information regarding all resources in the grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. On the other hand, dynamic scheduling the basic idea is to perform task allocation as the application executes, which is important in real-time mode applications as well as in situation where you cannot determine the execution time.

- **Optimal vs. Suboptimal**

  All information regarding the state of resources and the jobs is known in this case, hence an optimal assignment could be made based on some criterion function, such as minimum makespan and maximum resource utilization.The NP-Complete nature of scheduling algorithms and the difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories.

- **Approximate vs. Heuristic**

  Suboptimal approximate is a sufficiently "good" solution taken, instead of searching the entire solution space for an optimal solution. The factors which determine whether this approach is worthy of pursuit include [2];

  - Availability of a function to evaluate a solution.
  - The time required to evaluate a solution.
  - The ability to judge the value of an optimal solution according to some metric.
  - Availability of a mechanism for intelligently pruning the solution space.

The other branch in the suboptimal category is called heuristic. This branch represents the class of static algorithms which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It also represents the solutions to the scheduling problem which cannot give optimal answers but only require the most reasonable amount of cost and other system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation. Not restricted by formal assumptions, heuristic algorithms are more adaptive to the Grid scenarios where both resources and applications are highly diverse and dynamic.

- **Distributed vs. Centralized**

  The responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In a computational Grid, there might be many applications submitted or required to be rescheduled simultaneously. The centralized strategy has the advantage of easy implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck. For example, Sabin et al [4] propose a centralized metasheduler which uses backfill to schedule parallel jobs in multiple heterogeneous sites. Arora et al [5] present a completely decentralized, dynamic and sender-initiated scheduling and load balancing algorithm for the Grid environment. A property of this algorithm is that it uses a smart search strategy to find partner nodes to which tasks can migrate.

- **Cooperative vs. Non-cooperative**

  We considered whether the nodes involved in job scheduling are involved in cooperation between the distributed components (cooperatively), this is when, each grid scheduler has the responsibility to carry out its own portion of the scheduling task, but all schedulers are working toward a common system-wide goal. The other category occurs when the decision making process is done independently of other processors (non- cooperatively), in this case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system[2].

10.4 OAR Scheduler

There are many scheduler used, depending with the scheduling application, in our case we take a close look of OAR scheduler[23], which is a batch scheduler. Some of the features includes priority scheduling with queues, reservations, and backfilling. First Come First Served algorithm is implemented in OAR.

OAR has the following features: OAR handles submission and deletion of jobs with priority queues.

OAR supports clusters of SMPs, that means each node is associated to a maximum weight and each job has weight. A node can only run tasks whose weights do not add up to more than its maximum weight.

OAR handles properties, means the users can select which nodes are candidates for running their job.

OAR handles reservations, means its possible for a user to ask for nodes ahead of time. Acception of the reservation means that the system will start on time.

Although most jobs are intrinsically moldable, OAR does not handle moldable tasks, it deal with rigid jobs.

Exist some limitations in OAR environment such as;

Since the batch scheduling systems are on-line, all events that take place in the future of the current scheduling time are unknown to the scheduler.

Scheduling in presence of machine unavailability is difficult task, thus very simple scheduling problems becomes NP-Hard, when constraints of scheduling with reservations are considered.

11 Challenges and Courses Involved

At this section i will look at the courses which were involved in my studies as well as the challenges i encountered along my research.

11.1 Challenge Encountered

Up until i started researching and reading some articles, about scheduling algorithm, i did not know that it's a big field of studies and research, with lot of materials to study, and limited time, i was able to study a portion of scheduling field. Some of the materials are not available in digital form, in that manner it just made the research work harder.

I had a big challenge to sought out which kind of materials, which i needed for my research, here my supervisor played a major role, by helping me to pick what i needed and also helping to cave the path of my research.

11.2 Courses involved

- **MAC0422 - Operating System**

In operating system course, among the topics covered were process management (creation, synchronization, and communication); processor scheduling; deadlock prevention, avoidance, and recovery; main-memory management; virtual memory management (swapping, paging, segmentation and page-replacement algorithms); control of disks and other input/output devices; file-system structure and implementation; and protection and security.

- **Mac5758/0461 - Introduction to Scheduling and application**

  This course, gave an introduction to scheduling of different resources. Upon what i was able to study in this course, it facilitated a better understanding in my research.

- **MAC0431 - Introduction to Parallel and Distibution Computation**

  In this course, i had an introduction of parallel computing, understanding the concept of parallel computing, where it is applicable and its limitation.

- **MAC0412 - Computers Organization**

  In this course, i had an opportunity to study the hardware and software components like algorithm, I/O system, programming language, compiler and architecture and how this components affect the performance.

- **MAC0438 - concurrent programming**

  In this course we looked at Concurrent programming vital role in systems where many events appear to occur simultaneously. The course aimed to provide an introduction to the problems common to concurrent systems such as operating systems, distributed systems and real-time systems, and practical knowledge of the programming constructs and techniques offered by modern concurrent programming languages.

  Other courses like, **MAC0110 - Introduction to Computation, MAC0122 - Principal of Algorithm Development MAC0323 - Data Structure**, also contributed to my formation of understanding of the area of interest.

12 Future

This studies was designed to lay a groundwork of deeper understanding of scheduling algorithm, about what it is and what it involves. In that manner my supervisor designed my studies as a stepping stone to greater understanding of the area and for further studies in my masters.

The masters program will give me a chance to have a closer look of the subject, in a way the undergraduate course cannot.

References

[1] F. Berman, High-Performance Schedulers, chapter in The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1998.

[2] T. Casavant, and J. Kuhl, A Taxonomy of Scheduling in General-purpose Distributed Computing Systems, in IEEE Trans. on Software Engineering Vol. 14, No.2, pp. 141–154, February 1988.

[3] Y. Zhu, A Survey on Grid Scheduling Systems, Department of Computer Science, Hong Kong University of science and Technology, 2003.

[4] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment, in the Proc. of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science; Vol. 2862, Washington, U.S.A, June 2003.

[5] M, Arora, S.K. Das, R. Biswas, A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments, in Proc. of International Conference on Parallel Processing Workshops (ICPPW'02), pp.:499 – 505, Vancouver, British Columbia Canada, August 2002.

[6] Round-Robin Scheduling for Max-Min Fairness inEfficient Scheduling and Execution of Scientific Workflow Tasks by Laura Bright, David Maier. Department of Computer Science, Portland State University, Portland, Oregon.

[7] Time-shared Systems: A Theoretical Treatment: LEONARD KLEINROCK. Department of Engineering, University of California, Los Angeles

[8] Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms Dimitrios Stiliadis and Anujan Varma, Member, IEEE

[9] Bennet, J. and Zhang, H. "WF2Q: Worst-case Fair Weighted Fair Queueing." Proceedings of IEEE INFOCOM Low latency and efficient packet scheduling for streaming applications Eric Hsiao-Kuang Wu *, Ming-I Hsieh, Hsu- Te Lai Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan, ROC

[10] Quality-of-service in packet networks: basic mechanisms and directions. R. Guerin V. Peris

[11] Credit-Based Fair Queueing (CBFQ): A Simple Service-Scheduling Algorithm for Packet-Switched Networks Brahim Bensaou, Member, IEEE, Danny H. K. Tsang, Senior Member, IEEE, and King Tung Chan, Member, IEEE

[12] Fair and Efficient Packet Scheduling Using Elastic Round Robin; Salil S. Kanhere, Student Member, IEEE, Harish Sethu, Member, IEEE, and Alpa B. Parekh

[13] A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks; Vahid Salmani, Saman Taghavi Zargar, and Mahmoud Naghibzadeh

[14] Overview of real-time scheduling problems, J. Goossens and P. Richard, Universite de Bruxelles

[15] Evaluation of Surplus Round Robin Scheduling Algorithm; Dessislava Nikolova and Chris Blondia, University of Antwerp, Department of Mathematics and Computer Science, Performance Analysis of Telecommunication Systems Research Group

[16] D. Stiliadis, 1996, "Traffic Scheduling in Packet-Switched Networks: Analysis, Design, and Implementation", Ph.D. Dissertation, University of California at Santa Cruz, USA.

[17] Efficient Fair Queueing Using Deficit Round Robin, M Shreedhar and George Varghese

[18] Analysis and Simulation of a Fair Queueing Algorithm, Alan Demers, Srinivasan, Scott Shenker Xerox PARC

[19] A Self-Clocked Fair Queueing Scheme for Broadband Applications; S. Jamaloddin Golestani, Bellcore

[20] Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks Pawan Goyal, Harrick M. Vin, and Haichen Cheng, Distributed Multimedia Computing Laboratory, Department of Computer Sciences, University of Texas at Austin

[21] Ousterhout, J K Scheduling techniquesfor concurrent systems. 3rd Internation confrence Distributed Computing systems

[22] Blasewicz, J. Drabowski, M. and Weglarz. J Scheduling multiprocessor tasks to minimize schedule length. IEEE Trans. Comput. C-35

[23] A pragmatic analysis of scheduling environments on new computing platforms, Lionel Eyraud, Laboratoire ID-IMAG, Institut National Polytechnique de Grenoble,

[24] Assessing Operating Systems for Use in Safety Related Systems; R. H. Pierce, MSc.; CSE International Ltd; Flixborough, M. Nicholson, PhD.; Department of Computer Science, University of York, A. G. Faulkner, MSc.; CSE International Ltd; Flixborough.

[25] KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors Thierry Gautier, Xavier Besseron, Laurent Pigeon INRIA,

[26] The Cilk system of Parallel Multithreaded computing; Christopher F. Joerg;Department of Electrical Engineering and Computer Science, MIT.

[27] On Load Balancing Model for Cluster Computers; Huajie Zhang; School of Math, Physics and Information Engineering College of Zhejiang Normal University.

[28] Pfair Scheduling: Beyond Periodic Task Systems; James H. Anderson and Anand Srinivasan, Department of Computer Science, University of North Carolina