An Efficient Job Scheduling Technique in Trusted Clusters for Load Balancing

Shakti Mishra, Dharmender Singh.Kushwaha, Arun Kumar Misra
CSED, MNNIT Allahabad India
{shaktimishra,dsk,akm}@mnnit.ac.in

Abstract—**Although there has been tremendous increase in PC power and most of it is not fully harnessed, yet certain computation intensive application tend to migrate their process with the aim of reducing the response time. Cluster computing is the area that aims just at this. Clustering provides means to improve availability of services, sharing computational workload and performing computation intensive application. However, these benefits can only be achieved if the computing power of cluster is used efficiently and allocated fairly among all the available nodes. As is the case with our desktops, it is usually seen that clusters also suffer from underutilization. A number of approaches proposed in the past share only idle CPU cycles and not use the resources of systems when the machine has its own local processes to execute. We propose a priority-based scheduling approach for run queue and multilevel feedback queue scheduling approach for migrated tasks that doesn't degrade the performance of local jobs too. Simulation and experimental results have been able to show that priority-based run queue management and multilevel feedback queue scheduling for migrated tasks can increase overall throughput by about 28-33 percent.**

Keywords- Cluster;Load Balancing; Scheduling; Priority; Multilevel feedback Queue.

I. INTRODUCTION & SURVEY OF RELATED WORK

Clustering provides a better alternative to High Performance Computing (HPC) since the cost of highly available machines such as idle workstations and personal computers is significantly low in comparison to traditional supercomputers [1]. It has potential to improve availability of services, sharing computational workload and performing computation intensive application through efficient resource usage. The computational requirement of various applications can be met using cluster technology in an effective manner. However, these benefits can only be achieved if the computing power of clusters used efficiently and allocated fairly among all the available nodes.

Most of the machines do not use their full CPU capacity at any point of time. So, the fundamental policy of each machine in computer supported co-operative working (CSCW) is to share idle CPU cycles of these machines with any remote process which is demanding for CPU while not deteriorating the performance of original machine. But at the same time, the proposal has a constraint that the resources of systems can't be shared when the machine has its own local processes to execute. This becomes bottleneck when real time processes arrives on a machine. These remote real time processes begin to starve since they can't be scheduled on machine until unless CPU becomes idle. Thus, proper mixing of local and remote processes ensures no starvation policy for both.

Scheduling and interleaving of tasks in an optimal manner is mandatory for utilizing full capability of computing nodes with reduced completion time. The goal of the scheduling is to exploit the true potential of the system.

Cluster based distributed systems resolves complex problems by partitioning the task into sub tasks and then scheduling them in such a way so that each machine is assigned equal work and thus, balancing the load across the cluster with reduced waiting and response time, while ensuring little migration overhead.

The computing environment of nodes depends upon the cluster usage pattern that is broadly classified as Network of Workstations (NOW) and PMMPP (poor man's Massively Parallel Processors) (Table 1). NOW mode of cluster usage pattern is based on using idle cycles of personal computers or workstations and this implies an infinitely higher priority for workstations owner processes over remote processes (migrated processes from other workstations) [8]. MPP mode involves dedicated cluster for execution of high performance application.

TABLE I. CLUSTER USAGE PATTERN

Mode	Type Of Workload	Workstations Usage	Major Projects
Network of Workstations (NOW)	Regular Workload HPC Workload	Idle Cycles	CONDOR, MOSIX
MPP (Massively Parallel Processors)	HPC Workload	Dedicated cluster for HPC application	Beowulf, RWC PC

Typically, the jobs arrived on various workstations as a result of load balancing, contains different processes that may be dependent or independent from each other.

Depending upon these different type of processes, scheduling decisions may vary. Basically, scheduling choices are based upon two facts; (a) Number of processors available, (b) Process type (typically involves communication and synchronization pattern of processes). Number of processors available is further categorized as bounded number of processors or unbounded number of processors [3].

The author in [7] discusses about minimization of migration cost and defines a strategy as to which parts of the program should migrate. Many of the researchers [10] have tried to resolve issues like longer freeze time that may be due to unavailability of competing resources but their approach resolves pre-fetching of memory pages for process migration.

Although various approaches for scheduling tasks in clusters have been proposed and implemented by previous researchers [6, 9]. We find that each scheduling approach has its own assumption; however, for a load balancing system, we propose a combined approach, priority-based run queue management and multilevel feedback queue (MLFQ) scheduling approach for migrated tasks. Authors in [11, 12] claim that Multilevel Feedback Queue (MLFQ) scheduling proves viable for general purpose systems, however in this scheme CPU intensive processes suffers from starvation. MLFQ scheduling is chosen because of the several advantages as following:

- MLFQ uses priorities to decide which job should run at a given time: a job with higher priority (i.e., a job on a higher queue) is the one that will run.
- MLFQ uses the history of the job to predict its future behavior.
- MLFQ scheduling uses priority boost technique to raise the priority of processes to ensure that no process starves due to lower priority.

The proposed scheduling approach tries to resolve following issues:

- To prevent frequent migration of process due to unavailability of nodes by ensuring proper mixing of local and remote processes in run queue.
- Identification of critical processes by assigning highest priority and scheduling these processes immediately on one of the available processors.
- No starvation policy for any process while considering the priority and criticality of process.
- Scheduling local and remote jobs in run queue with same priority in round robin fashion so that neither of these processes may starve.
- Boosting priority of computation intensive remote processes in MLFQ to reduce remigration and congestion across the network.

The rest of the paper is organized as follows. Section 2 describes system model. In Section 3, process scheduling algorithms investigated in this paper are discussed. The performance analysis of algorithms is carried out in Section 4, followed by conclusion.

II. SYSTEM MODEL

A. Base Model

In our previous work [14], we have proposed that the cluster contains group of trusted nodes. A common node between two clusters is selected as Process Migration Server which we will now referred as Process Management Server (PMS). In case, if no node overlaps in two cluster, then a least loaded node would be referred as PMS as discussed in [13] . In order to reduce the overhead of polling and broadcasting periodically, each node of the cluster sends its load status information to PMS, when it changes. As node sends their load statistics, PMS updates its Node_Status_Table (NST) (Fig.1). PMS has several daemon processes which handle following functions:

1. Collecting load statistics from all other nodes and maintaining and updating NST.
2. Registering each node via registration module and maintaining trust among all nodes [4, 5].
3. Group multicasting the list of least loaded nodes to overloaded nodes.

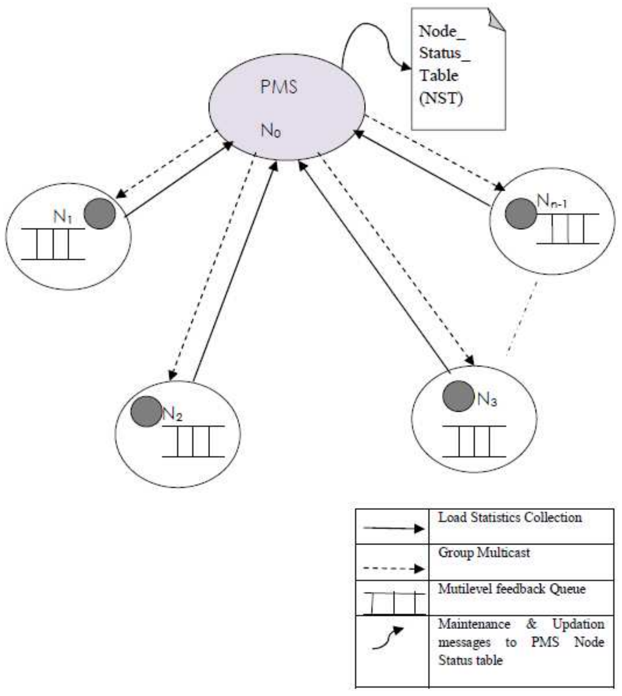

PMS
N0
Node_ Status_ Table (NST)
N1
N2
N3
Nn-1
Load Statistics Collection
Group Multicast
Mutilevel feedback Queue
Maintenance & Updation messages to PMS Node Status table

Figure 1. Base Model of JMM with MLFQ

This has been illustrated in Fig. 1 in which trusted nodes send their status updates information to PMS. A local process scheduler computes various parameters on each node like CPU utilization, resource availability including input / output resources, network bandwidth and memory usage including the number of processes in the process queue. CPU utilization refers to the CPU contribution to the functioning of a node. It is considered to be high if less number of CPU cycles is wasted. Each node also has an updated status of resources available in the system.

B. Local Scheduler

A local scheduler is responsible for maintaining multilevel queues on each node as shown in Fig. 2. The objective is to locate a process in the highest priority queue and assign the CPU to it. It is invoked, directly or in a lazy way, by several kernel routines. The scheduler keeps track of what processes are doing and adjusts their priorities periodically. When a resource request or migration request arrives from some node, the scheduler keeps these processes in the highest priority queue and adjusts their priority repeatedly and also checks whether the resource needed by the process is available; if not, it yields the CPU to some other process by invoking scheduler [2].

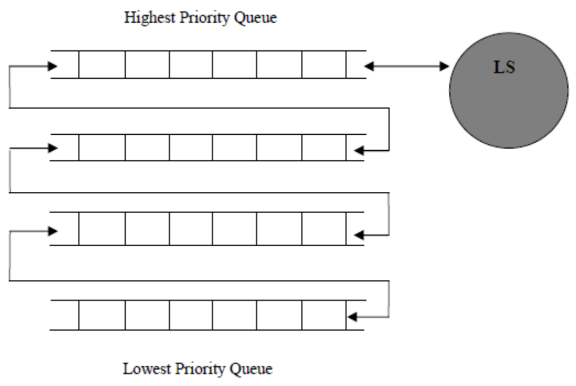


Figure 2. Design of Local Scheduler

The local scheduler must have information about available and occupied resources. There may be some jobs in which node status is underloaded while the node is suffering from memory unavailability. In this case, the process would migrate due to insufficient memory.

III. PROCESS SCHEDULING STRATEGIES

A. *Node Selection Strategy*

- *Least Busy CPU First (LBCF):* On the basis of information provided by PMS, an overloaded node may choose least busy node for migration.
- *Neighbor CPU First (NCF):* PMS maintains a closest vector node for each of the node in the state table. The criteria may be chosen as minimum number of hops from overloaded node to an idle node.
- *Random Selection:* An overloaded node can randomly choose any idle node for process migration. The selection criteria may be the resource availability, memory availability, network bandwidth, compatibility among system and many other factors. This random selection is based upon greedy approach.

B. *Scheduling Strategy*

Our proposed scheduling strategy is partitioned into two sections. The first section deals with dynamic run queue management by appropriately loading processes to main memory from process pool while second one deals with run queue CPU allocation to processes. CPU scheduler is a critical piece of the operating system software that manages the CPU resource allocation to tasks. It typically strives for maximizing system throughput, minimizing response time, and ensuring fairness among the running tasks in the system [16]. Our proposed scheduling strategy is based on the priority and criticality of the processes. The term critical process refers to a process whose execution should not be delayed or the process which has strict time constraints. Such processes carry highest priority. We propose that if the process is critical, its execution should only be suspended if the node that receives this process is executing its own critical local/remote process.

1) *Priority-based Run Queue Management*

Based on the above discussion, we have four types of processes. The convention for these processes is listed in Table 2.

TABLE II. PROCESS CONVENTION

S. No.	Process Type	Convention
1	Local Process	Li
2	Remote Process	Ri
3	Local Critical Process	C Li
4	Remote Critical Process	CRi

The process scheduler selects the job from the pool using one of the following cases:

[Case A] When an idle node receives a migration request of (local or remote) critical process, it is immediately chosen for execution.

[Case B] When a remote process (R_i) gets migrated on idle node, it executes on remote machine until unless a critical local process arrives.

If a critical local process arrives on the same machine while execution of a remote process, then it preempts the remote process to waiting queue. However, if in the mean time, local process waits for an I/O resource, the remote process dequeues from waiting queue and get chance to execute. Linger-Longer approach [8] provides this facility for fine-grained idle periods to run foreign jobs with very low priority.

[Case C] An underloaded node has critical local process to execute and simultaneously it gets a migration request of critical remote process. Then, it simply rejects the request.

[Case D] A local process arrives on an underutilized node currently executing an remote process, then both processes start their execution in round robin fashion.

We consider the case where a remote job is being executed on an idle node and simultaneously local process arrives. Condor [15] uses pre-emption technique to resolve this problem while the approach does not consider the case of starvation of remote process if the frequency of local process is too high. Here, we follow the round robin scheduling. This approach scores over previous systems that support collaborative working while utilizing resources and CPU efficiently with no starvation. The scenarios discussed in CASE A and CASE B can be described by the data shown Table 3 which shows different processes with equal priority and their arrival time with their execution time.

TABLE III. SCHEDULING DATA FOR CASE[A]

Process	Arrival Time	Execution Time
L1	0	3
L2	1	1
L3	4	4
CR1	5	2
R1	6	1

The scheduling policy for above shown processes can be described as Fig. 3. Initially, the local process executes as per round-robin fashion. However, at time unit 5 when Critical remote process (CR1) arrives, scheduler preempts the CPU from local process L3 and CR1 starts its execution.

As CR1 finishes its execution, local process L3 and remote process R1 continue their execution in the round robin order.

Let us consider another case with different processes and their characteristics as depicted in the table (Table 4) given below. Here the initial execution of local and remote processes is same as Fig. 4 until remote critical process R1 arrives. As CL1 arrives, all local and remote processes are put to waiting queue and CL1 starts its execution. Now, if at the same time critical remote process (CR1) arrives, the request is discarded by scheduler.

TABLE IV. SCHEDULING DATA FOR CASE [B] & CASE [C]

Process	Arrival Time	Execution Time
L1	0	3
L2	1	1
L3	4	4
R1	5	2
CL1	6	3
CR1	7	1

Now, consider Table 5. with different type of processes and their priority with their arrival and execution time,

TABLE V. SCHEDULING DATA FOR CASE[D]

Process	Arrival Time	Execution Time	Priority
L1	0	3	1
L2	1	1	2
L3	4	4	2
R1	5	2	3
CL1	6	3	0

The scheduling mechanism (Fig. 5) depends upon the priority of processes. At time interval 0, L1 starts its execution. At time interval 1, L2 arrives. Since, the priority of L1 is greater than from L2, L1 continues its execution and L2 waits. At time interval 3, L2 starts its execution and then L3. Remote process R1 arrives at 5 unit of time, scheduler compares the priority of R1 and L3, since R1 carries lower priority, it is put in to the waiting queue. At time interval 6, critical local process (CL1) arrives, scheduler preempts the CPU from local process L3 and CL1 starts its execution. As CL1 finishes its execution, local process L3 finishes first and then and remote process R1 continues its execution.

2) *Multilevel Feedback Queue Approach for Migrated Tasks*

A remote job aware multilevel feedback queue based scheduling for migrated tasks is shown in Fig. 6.

A multilevel feedback queue consisting of five queues; each assigned a different time quantum, the CPU switching time and the priority levels are given in the Table 6.

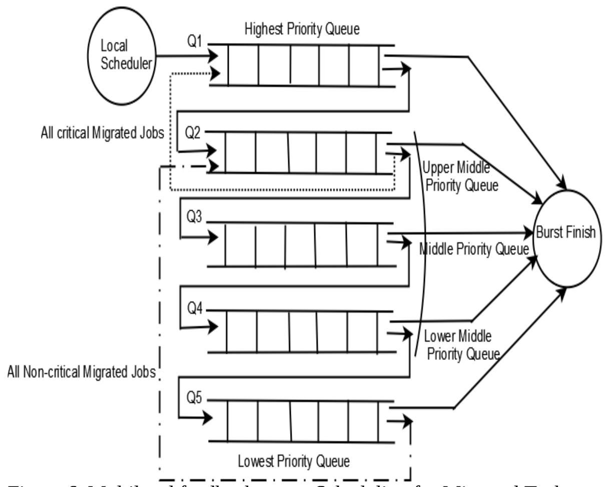

Local Scheduler
Q1
Highest Priority Queue
All critical Migrated Jobs
Q2
Upper Middle Priority Queue
Q3
Middle Priority Queue
Burst Finish
Q4
Lower Middle Priority Queue
All Non-critical Migrated Jobs
Q5
Lowest Priority Queue

Figure 6. Multilevel feedback queue Scheduling for Migrated Tasks

TABLE VI. MLFQ FOR MIGRATED TASK PARAMETES

Queue	CPU switching time (ms)	Priority level
Q1	16	Highest
Q2	32	Upper Middle
Q3	64	Middle
Q4	128	Lower Middle
Q5	256	Lowest

IV. PERFORMANCE EVALUATION

We computed the average waiting time and mean response time for local and remote processes. Migration overhead is considered negligible in this analysis to simplify the model and simulations.

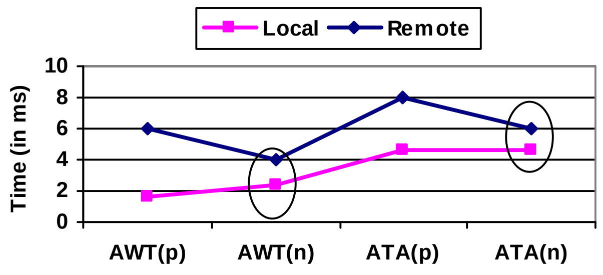


Figure 7. Average Waiting & Turnaround Time Comparison of previous approach with proposed new approach for data given in Table3.

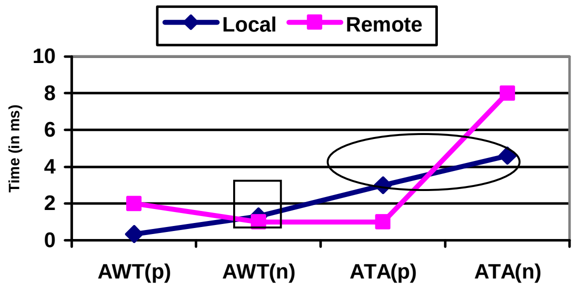


Figure 8. Average Waiting & Turnaround Time Comparison of previous approach with proposed new approach for data given in Table5.

Fig. 7 shows the comparison of average waiting & turnaround time of proposed new approach (AWT (n) & (ATA(n)) with previous approaches (AWT(p) & ATA(p)). We observe from the encircled values in Fig. 7 that proposed approach is able to reduce the average waiting and turnaround time of local and remote processes by 28-33%.

The simulation analysis proves that for data set given in Table 5, previously proposed approaches didn't accept any remote process when the machine had its own local processes to be executed. With our approach, the system is able to accommodate remote process along with local process with marginal increment in the waiting & turnaround time of local processes.

The values enclosed in square area in Fig. 8 shows that using our proposed approach, the average waiting time of remote processes is decreased in comparison to others, and significantly lowered as compared to local processes. From the values encircled in the same, we also observe the reduced turn around time of remote process with allowable increment in the turnaround time of local processes.

V. CONCLUSION

This paper presents an optimized scheduling approach for trusted cluster environment that resolves important issues of remote process starvation in case of local processes arrival, frequent migration of remote processes and selection criteria of idle node.

The experimental results establish that priority-based scheduling increases overall throughput by about 28-33 percent. In some cases, we also find that the proposed approach is able to accommodate more number of remote processes than the previous approaches with marginal increment in waiting and turnaround time of local processes. This in turn shall allow more users going the cluster computing way without the concern of degraded system performance and further investments in scalability and redundancy.

REFERENCES

[1] R. Buyya, "High Performance Cluster Computing: Architecture and Systems" Vol. 1. Pearson Education, pp. 61-68.

[2] O'Reilly, "Understanding the Linux Kernel", O'Reilly Media, 2002.

[3] S. Pasham and W. Lin, "Efficient task scheduling with duplication of bounded number of processors", 11th International Conference on Parallel and Distributed Systems (ICPADS'05)vol. 1, pp. 543-549. 2005

[4] Shakti Mishra, D.S.Kushwaha, and A.K.Misra," A Cooperative Trust Management Framework for Load Balancing in Cluster Based Distributed Systems", In IEEE proceedings of International Conference on Recent Trends in Information, Telecommunication and Computing, ITC 2010. pp. 121-125. March 2010

[5] Shakti Mishra, D.S.Kushwaha, and A.K.Misra," A Novel Approach for Building a Dynamically Reconfigurable Trustworthy Systems", Information Processing and Management: Proc. of International Conference on Recent Trends in Business Administration and Information Processing, BAIP 2010, CCIS 70, Springer-Verlag, Berlin Heidelberg, pp. 258-262. March 2010.

[6] H. Rajaei, "Simulation of Job scheduling for small scale clusters", Proc. of Winter Simulation Conference, pp. 1195-1201. 2006.

[7] H. Karatza, "A comparison of load sharing and job scheduling in Network of workstations", I. J. of Simulation Vol. 4 (3-4), pp. 4-11. 2003.

[8] K. D. Ryu and J. K. Hollingsworth, "Exploiting Fine-Grained Idle Periods in Network of Workstations", IEEE Transactions on Parallel and Distributed Systems, Vol.11, No.7,, pp. 683-698. July 2000.

[9] T. Akgun, "BAG Distributed Real-Time Operating System and Task Migration", Turkish Journal Electrical Engineering, Vol. 9, NO. 2, pp.123-136. 2001.

[10] Ho, R.S.C., Cho-Li Wang, and Lau, F.C. "Lightweight Process Migration and Memory Prefetching in Open MOSIX", IEEE International Symposium on Parallel and Distributed Processing IPDPS, pp. 1-12. 2008.

[11] Arpaci Dusseau, "Scheduling Multilevel feedback Queue", Operating System, Ch. 7, pp. 1-8.

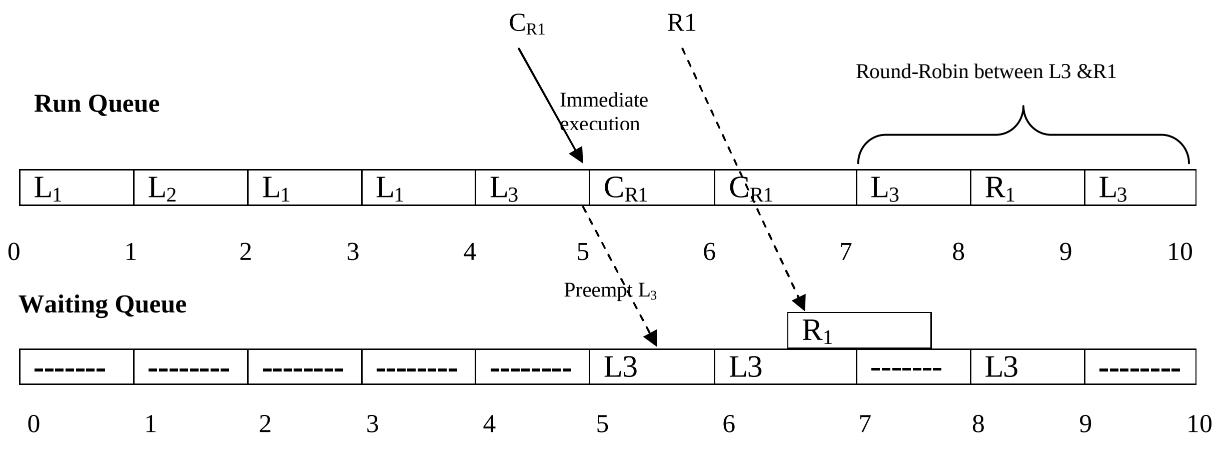
[12] K. Hoganson, "Reducing MLFQ Starvation with Feedback and Exponential Averaging", Southeastern Conference on Consortium for Computer Science in Colleges (CCSC), Vol.25 Issue 2. pp. 196-202. Dec' 2009.

[13] Shakti Mishra, D.S.Kushwaha, and A.K.Misra, "Hybrid Load Balancing in Auto-configurable Trusted Clusters", Journal of Computer Science and Engineering, Vol. 2 (1), pp. 16-25. July 2010.

[14] Shakti Mishra, D.S.Kushwaha, and A.K.Misra," Jingle- Mingle: A Hybrid Reliable Load Balancing Approach for a Trusted Distributed Environment". 5th IEEE International Joint Conference on INC, IMS & IDC, NCM 2009, pp. 117-122. Aug' 2009.

[15] M. J. Litzkow, M.Livny, M. W. Mutka, "Condor-A Hunter of Idle Workstations", 8th IEEE International Conference of Dstributed Computing and Systems, pp. 104-111. 1988.

[16] Siddha S., Pallipadi V., Mallick A., "Process Scheduling Challenges in Multicore Processors", Intel Technology Journal, Vol. 11 , Issue 04, pp. 361-369. 2007



Run Queue
C_R1
R1
Immediate execution
Round-Robin between L3 &R1
L1 L2 L1 L1 L3 CR1 CR1 L3 R1 L3
0 1 2 3 4 5 6 7 8 9 10
Waiting Queue
Preempt L3
R1
------- -------- -------- -------- -------- L3 L3 ------- L3 --------

Figure 3. Proposed Scheduling Mechanism for CASE A

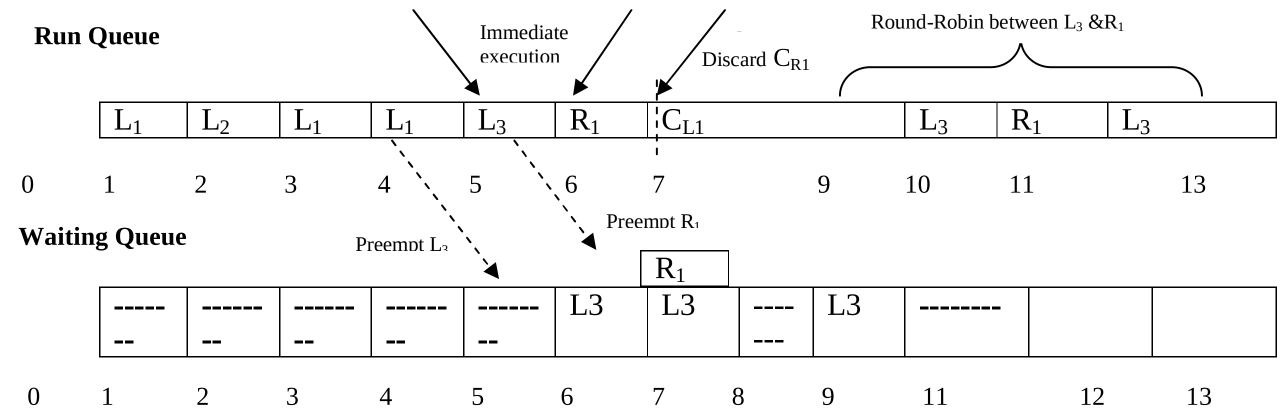


Figure 4. Proposed Scheduling Mechanism for CASE B & CASE C

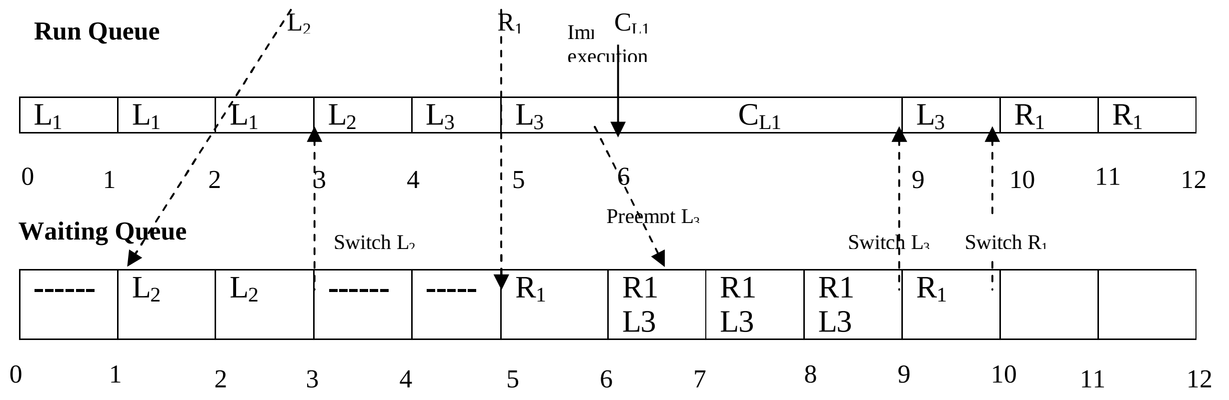


Figure 5. Proposed Scheduling mechanism for CASE D