Fault tolerant scheduling on a hard real time multiprocessor system

S. Ghosh, Rami Melhem, Daniel Mosse

Introduction

- Fault tolerance is important issue in hard real time system
- One way to provide fault tolerance is to schedule multiple copies of task on different processor
- Primary/backup approach algorithm proposed to handle transient faults.

  Tasks are assumed to be periodic and period of any task should be multiple of period of its preceding task.
- Algorithm also assumes that execution time of backup is shorter than primary.

Objective

- Study fault tolerant scheduling primary/backup (PB algorithm) , allows processor transient/permanent faults
- Idea of backup overloading which demands less processor time to provide fault tolerance
- Backup de-allocation idea

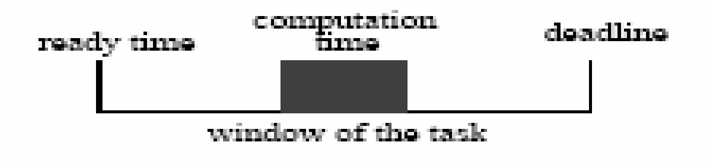
Fault tolerant scheduling problem

- System consist of n interconnected identical processor and there is task scheduling central processor
- Assume task are independent ,no precedence constraints
- Both permanent faults and transient fault handled by the proposed approach. It does not consider software faults or correlated component faults

- Task Ti modeled as Ti = $\langle a_i, r_i, d_i, c_i \rangle$

(arrival time, ready time, deadline, max computation time/WCET)
- Window of task = at least twice large as computation time

$$w_i = d_i - r_i$$



- Task schedules are guaranteed to execute if processor fails any instant of time and second processor does not fail before system recovery from first failure
- If completion guarantee of task not assured ,then task is rejected

Primary/Backup scheduling approach

Two major techniques used while scheduling

- Backup overloading

Scheduling backups for multiple primary task at same time period for efficient processor time utilization

- De-allocation of backup after successful completion of primary

Primary/backup time slots: time slots when primary and backup copy of task scheduled

Overloaded time slot: if backup copies of more than one tasks scheduled to run in same time slot

Forward slack: max amount of time a slot can be postponed without violating timing constraints

Scheduling Restrictions

Let Primary copy $\Pr_i$ and Secondary copy of task Bk_i

- Primary task and secondary can't be scheduled on same processor
- Begin time of secondary task has to be greater then primary

  So that backup can be executed after fault detection
- Both primary and backup to be scheduled between ri and di (ready time and deadline)
- <u>If two primary scheduled on same processor then their backup must not overlap</u>

Algorithm for fault tolerant scheduling of task Ti

- Schedule Pri as early as possible
- Try to overload Bki on existing backup slot .If not possible schedule backup as late as possible on free slot.
- If schedule has been found for both Pri and Bki, then commit the task otherwise reject it.

Algorithm principle

- When new task arrives its primary and backup needs to be scheduled
- While scheduling Primary and backup, list of existing slots is maintained
- Schedule primary as early as possible and backup as late as possible
- After successful completion of primary ,its backup is de-allocated
- After de-allocation the free slot is reutilized for scheduling any task that arrives after de-allocation.

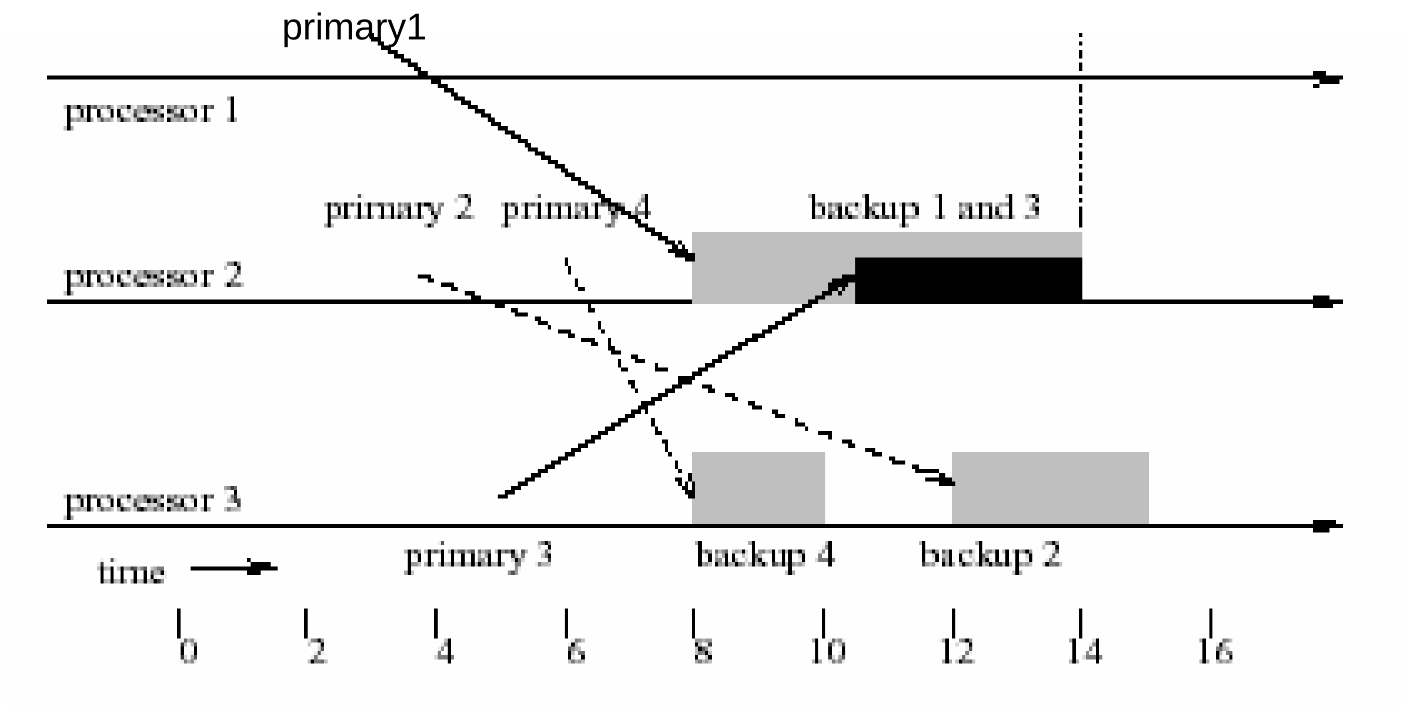
Thus utilization is increased and overhead is reduced

Scheduling 4 task on 3 processors

Primary scheduled as early as possible and backup scheduled as late as possible

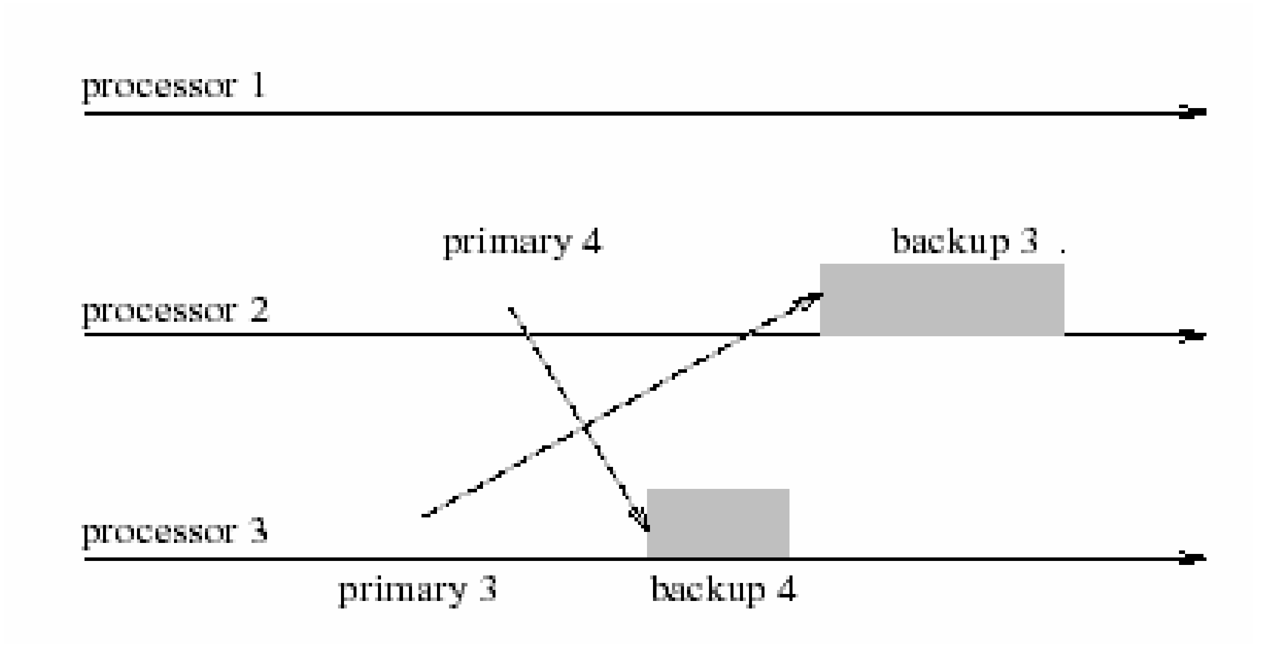
Assumption , release time = ready time i.e. $a_i = r_i$.

And Bk1 and Bk3 overloaded on same time slot



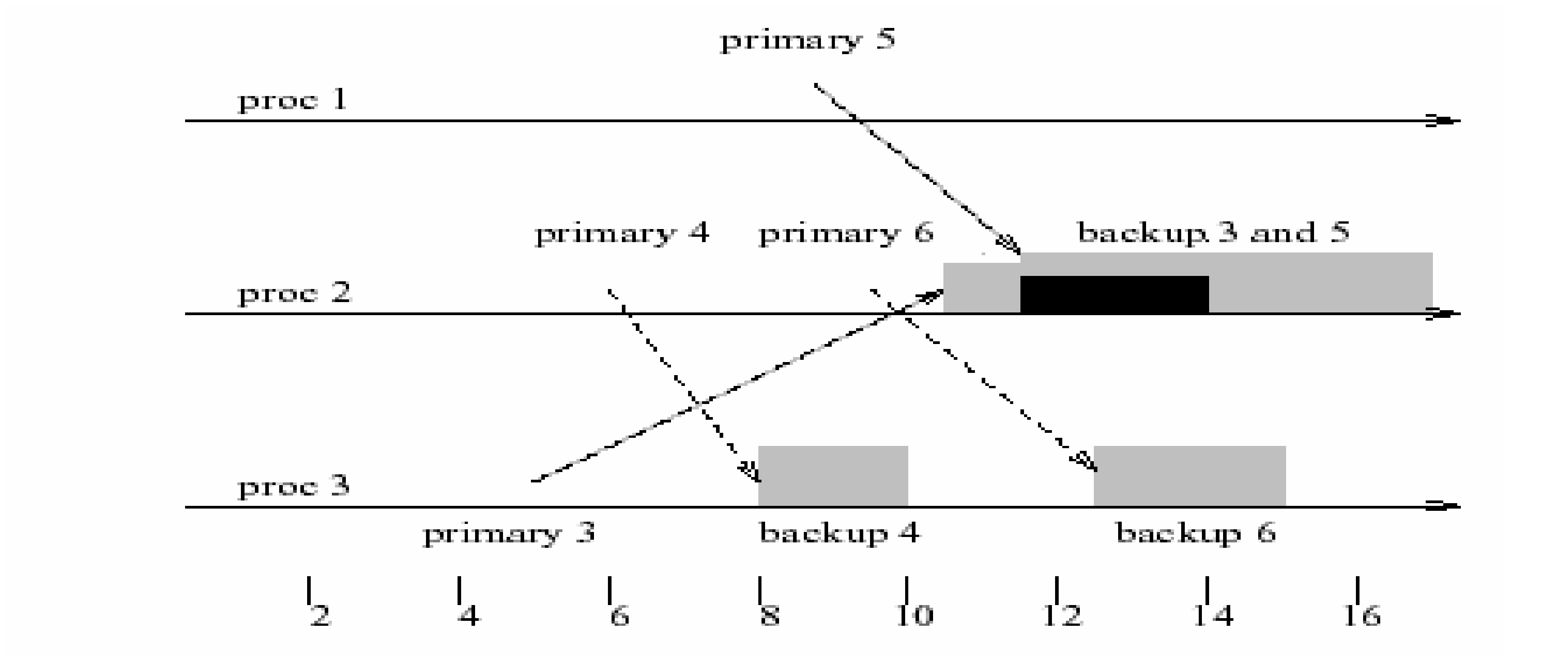
De-allocation

- Completion of task 2 and 1 cause de-allocation of respective backup



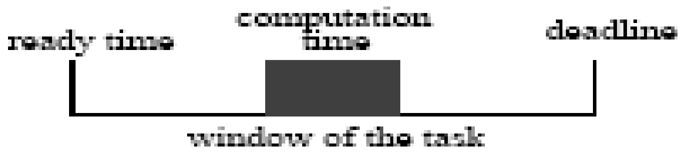
New schedule after arrival of 2 more tasks

Bk3 overloaded with Bk5 and due to de-allocation of Bk1 , Pr6 can be scheduled on processor2.



Primary Task scheduling steps

- Check each processor to see if Pri can be scheduled between ri and di



- If there is free slot larger than Ci then schedule Pri on that Processor
- If Pri can't be scheduled w/o overlapping other time slot slot_j then check if slot_j can be rescheduled
- To check slot_j

  -check slack of slot_j

  -if (slack of slot_j + preceding free slot) > ci then Pri can be scheduled after shifting slot_j

Backup of task scheduling

- If primary task is scheduled on processor Pj then to schedule backup other than Pj

1. First choice to overload existing backup slot
2. If no backup slot can be overloaded then schedule backup on existing free slot

- For primary forward slack is maintained and allowed forward move but backup slot movement not allowed

  -As backup slot may support more than one primary and if its moved primary slack changes

Reasons for scheduling primary before backup

1. Scheduling primary is more difficult than backup
2. To minimize constraints

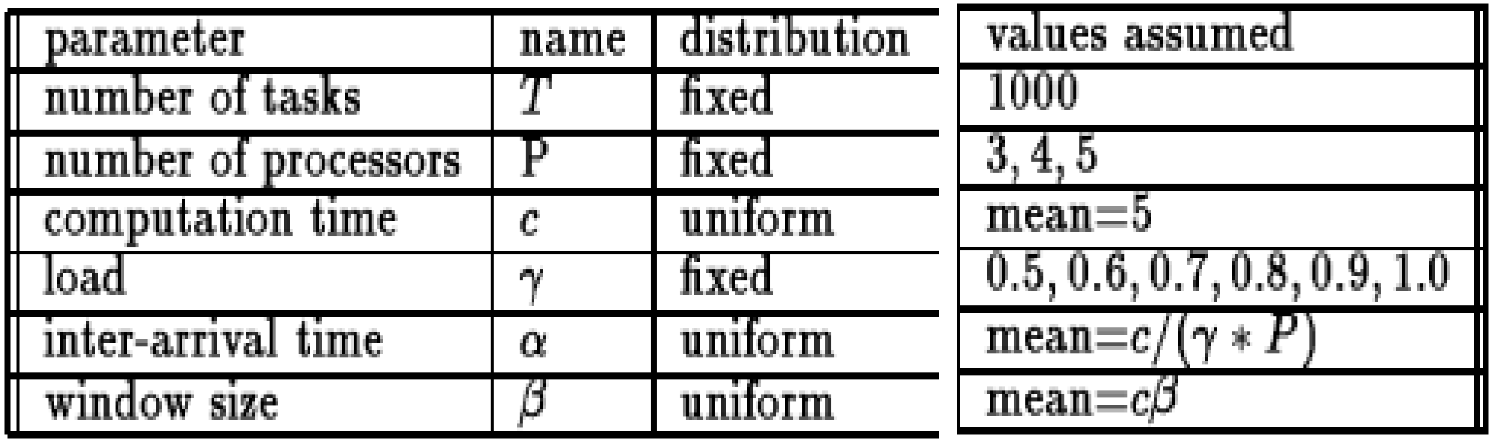
Scheduling backup is easy because

1] It can be overloaded on existing backup

Preferred as it minimizes utilization of available processor time

2] Can be scheduled in any free slot

Simulation parameter



parameter	name	distribution	values assumed
number of tasks	T	fixed	1000
number of processors	P	fixed	3, 4, 5
computation time	c	uniform	mean=5
load	γ	fixed	0.5, 0.6, 0.7, 0.8, 0.9, 1.0
inter-arrival time	α	uniform	mean=$c/(\gamma * P)$
window size	β	uniform	mean=$c\beta$

$\alpha = c/\gamma * P.$ $0 \leq \gamma \leq 1$

Lead time parameter= difference between ready time and arrival time

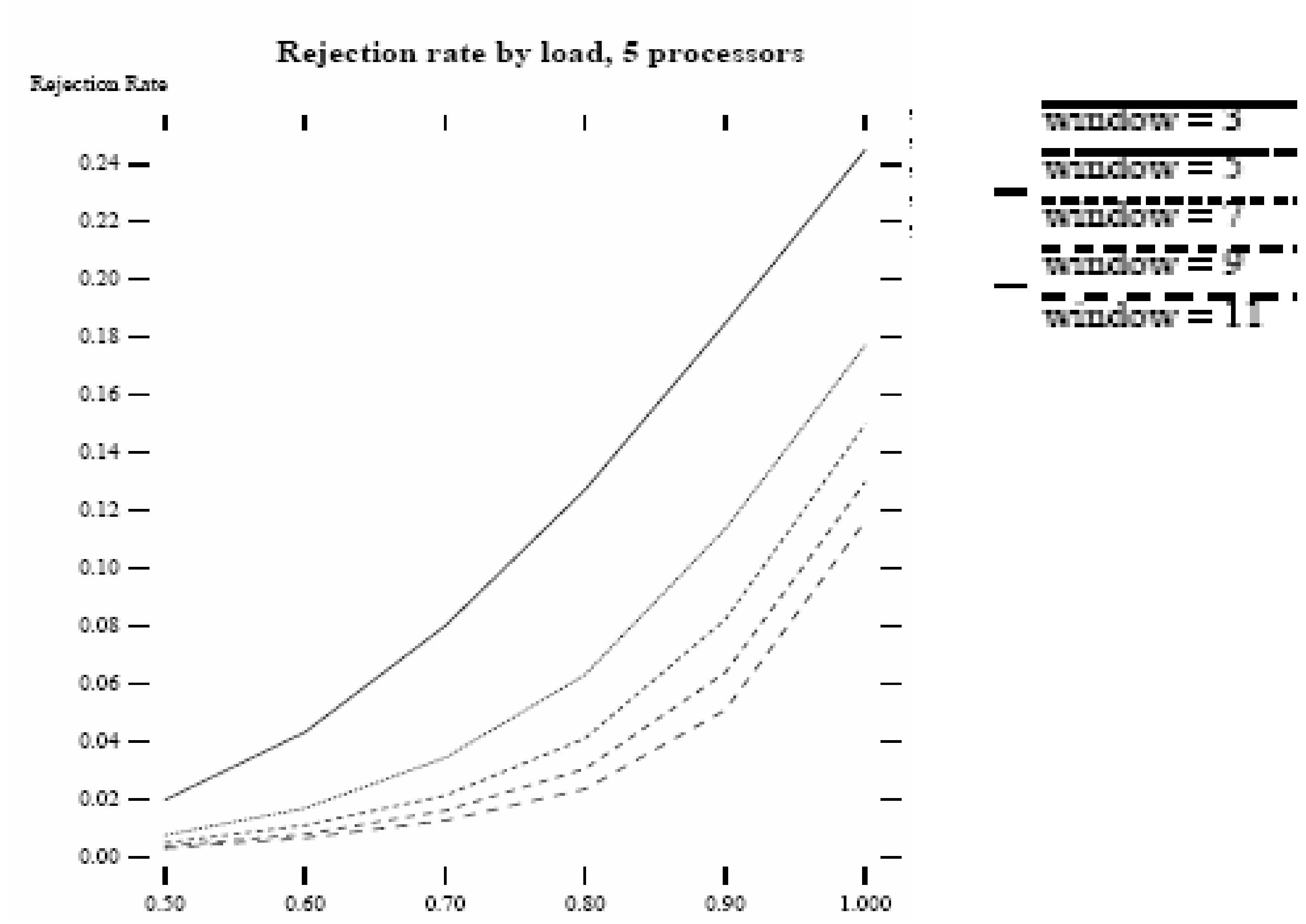
If $\gamma = 1$, $P = 4$, $c = 4$ then avg. inter-arrival time is 1, this means one task will arrive in system every unit of time and thus load on each processor is 1

Load ranges from 0 to 1

Simulation results

Rejection rate as function of load ,for different window sizes for 5 processors

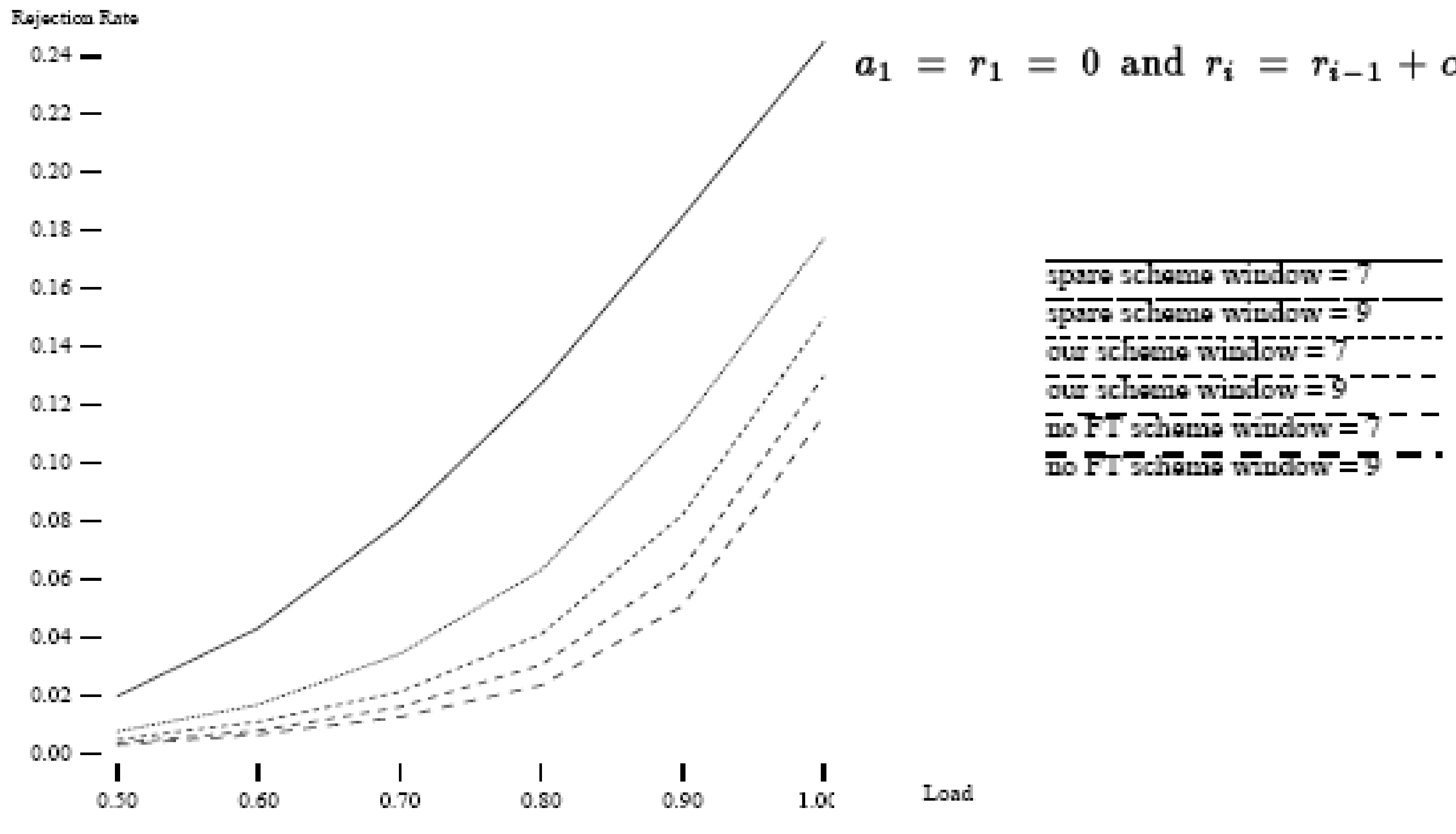
rejection rate increases as increase in load
larger the window size smaller the rejection rate



Comparison of 3 schemes : spare scheme, primary/backup, no FT

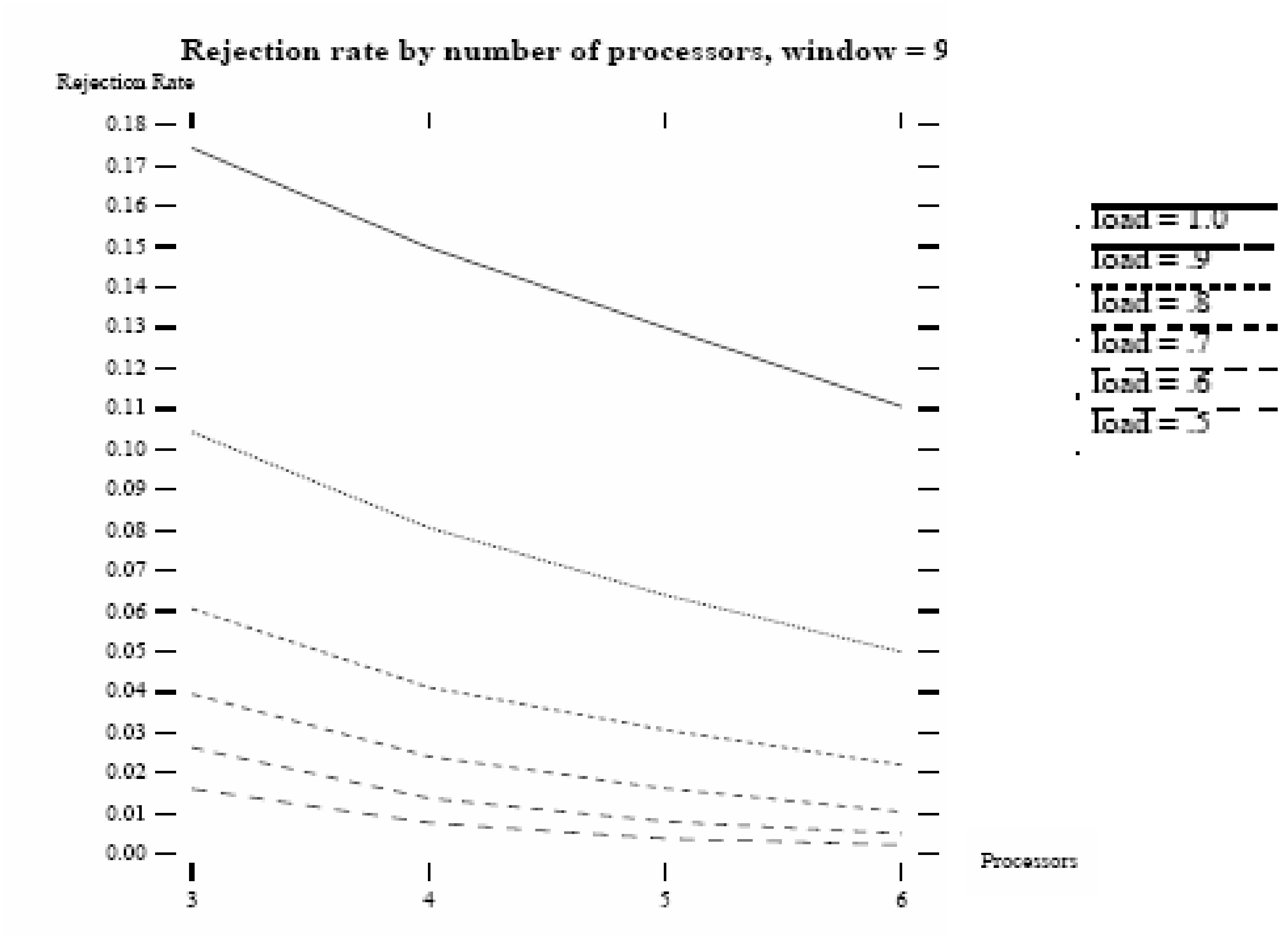
schemes simulation task set consist of 1000 tasks,
assumption lead time=0 arrival time=ready time

$$a_1 = r_1 = 0 \text{ and } r_i = r_{i-1} + \alpha_i,$$

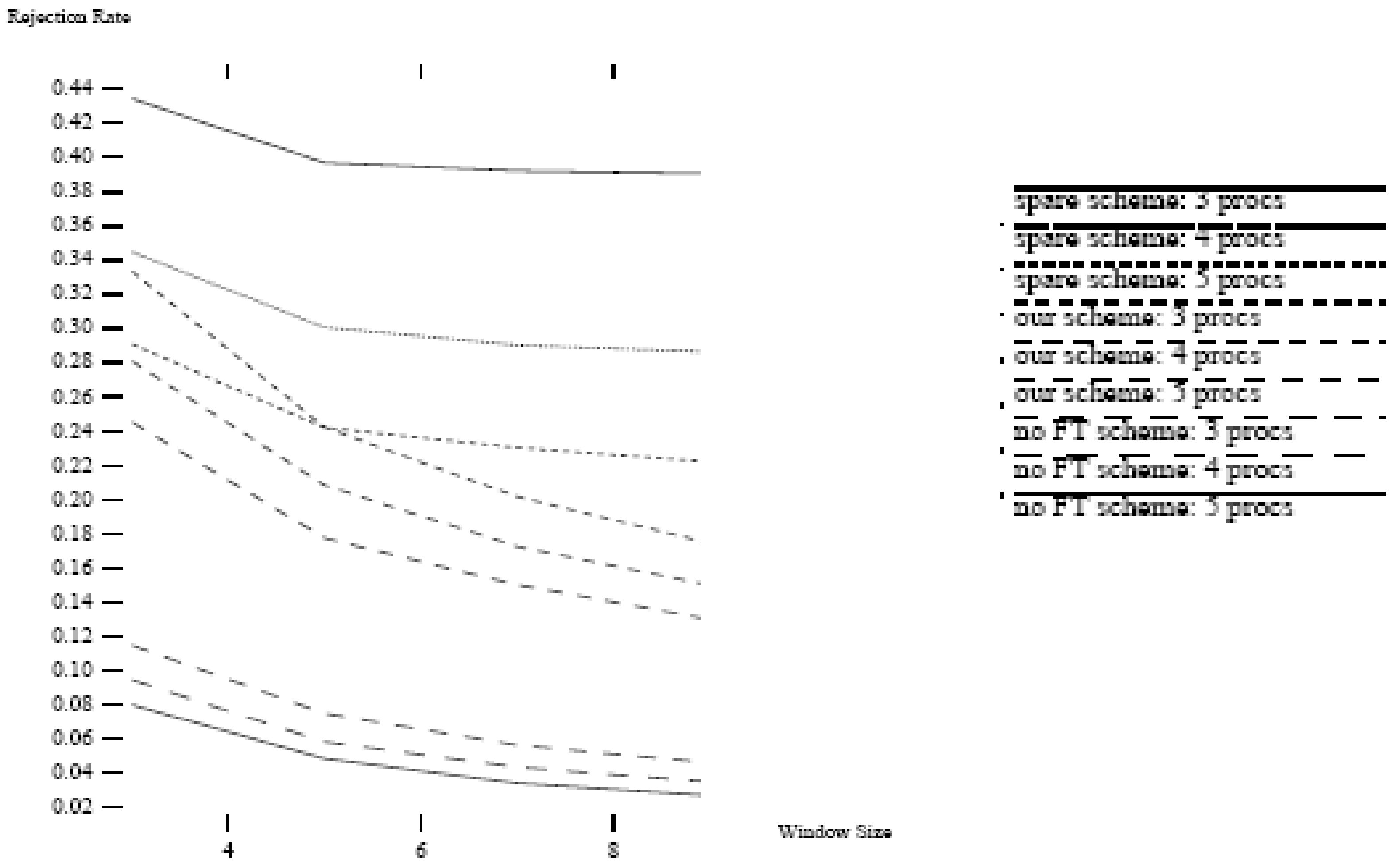


Rejection rate by load, 5 processors

Rejection rate of overloading schemes as function of number of processors



Rejection rate by window size



Runtime behavior

- If there is permanent fault in processor backup of all primary running on that processor executed on respective backups
- Task arriving after fault, both primary and secondary copies scheduled on fault free processors
- Second fault can be tolerated after last primary scheduled on faulty processor has been run its backup schedule and

  last task which has backup on faulty processor is executed
- Transient fault case, copy of currently executing primary is activated and remaining schedule remains same

Conclusion

- Algorithm tolerate more than one fault if separated by sufficient amount of time
- To tolerate 2 simultaneous faults more backup copy need to be scheduled