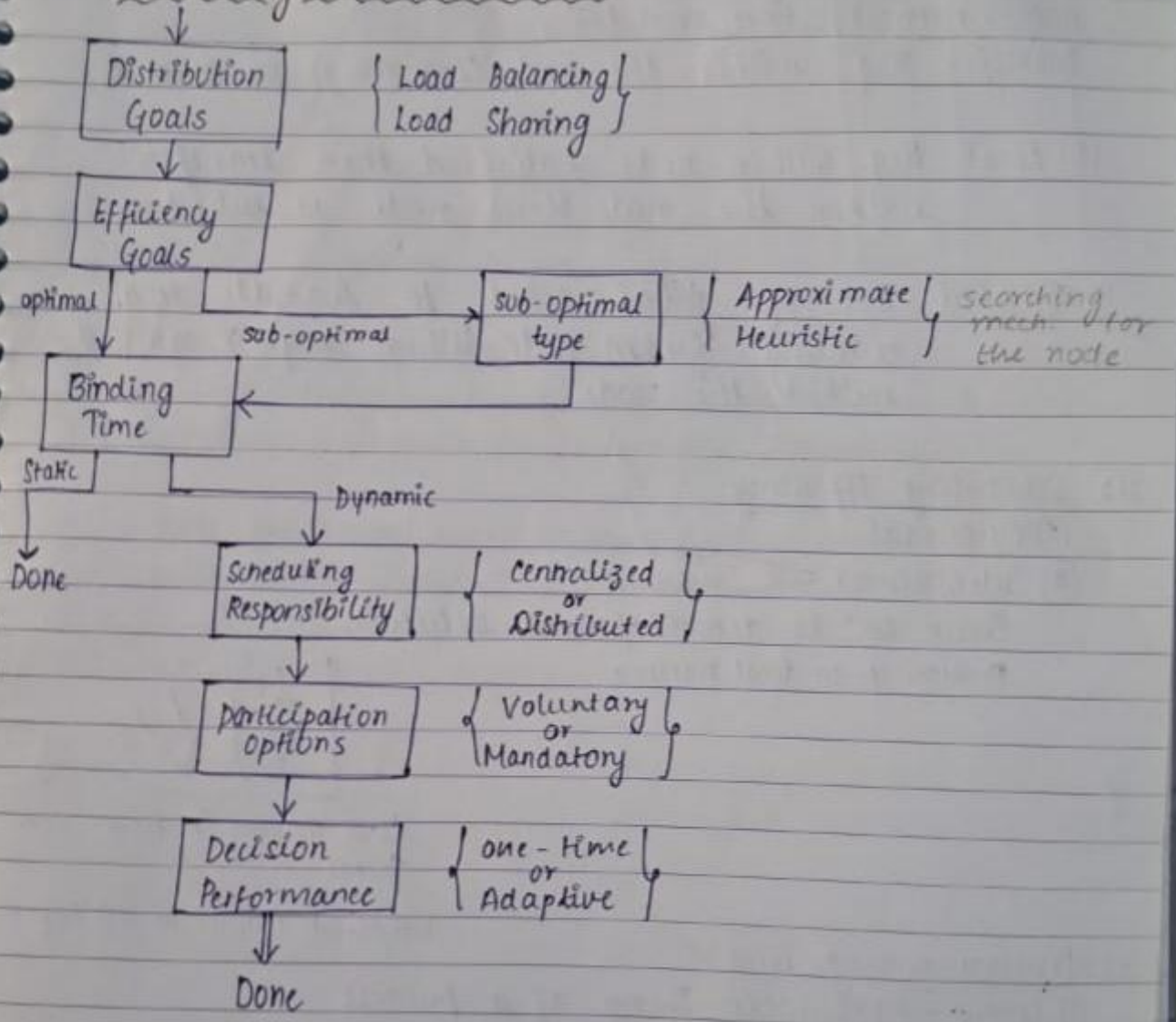
* Static Binding: Process details in req. msg while transferring

2. Scheduling Decision Chart:



2=1 → Optimal Static Scheduling

→ Voluntary optimal one-time distributed dynamic scheduler

→ Mandatory sub-optimal heuristic adaptive distributed dynamic scheduler every node part of decision making mandatorily (to add a node)

SJF → good if fixed no. of process → static
× when dynamic nature of processes
↑ burst time processes will × get chance → starvation

Not optimal due to dynamically changing env.

RR → Prior info about all processes when they will come in the system
dynam. coming processes → add to the end of the queue → wait for prec before

2.1 Level of Scheduling

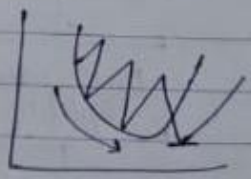
node → network, subnet, workstation
Transfer req. within the node (internally) or outside

(1) Local: Req. within node satisfied then transfer within the node. Here node is subnet

(2) Global: Job of global sched to handle req. coming from node When req. × satisfied within the node

2.2 Scheduling Efficiency

(1) Optimal
(2) Sub optimal ← Approximation, Heuristic
Closer sol" to actual/optimal solution
Problem: go to local minima



Local minima / maxima
Global " "

Weights → A non linear rel"/eq"

2.1.2 Processor-binding time

(1) Static → Executable Image of a process
process assigned to specific processor before execution
(When pgm ready for execution, after successful compilation (eg-.exe) during compile time adding req. resources

(2) Dynamic — leave for scheduler/algo, to see where exe
process on to CPU based on availability
file created & executed

2.1.3 Scheduling responsibility

① Centralized (SPOF problem)

② Distributed — Voluntary

Mandatory → Polling (dist. load, share process from 1 proc. to another)

Allow all nodes to take part in decision making

Voluntary for that node → choice for node to come voluntarily

Mandatory : Compulsory for every node to participate

2.1.4 Scheduler Decision performance

① One time assignment (specific CPU) Real time sys (No migration, process ass. once to)

② Adaptive proc. can migrate to ↓ loaded CPUs dynamically

Once you have added a system for decision making it will work forever

2.2 Task Assignment Approach (PK Sinha)

	t_1	t_2	t_3	t_4	t_5	t_6
t_1	0	6	4	0	0	12
t_2	6	0	8	12	3	0
t_3	4	8	0	0	11	0
t_4	0	12	0	0	5	0
t_5	0	3	11	5	0	0
t_6	12	0	0	0	0	0

* Intercommunication Cost
Moving from 1 task to another task

* Execution cost

Tasks	Nodes	
	n_1	n_2
t_1	5	10
t_2	2	∞
t_3	4	4
t_4	6	3
t_5	5	2
t_6	∞	4

(Infinity (lot of time to execute t_2 on n_2))

sender initiated
receiver "
Symmetrically "

Serial Assignment	
Tasks	Nodes
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_2
t_5	n_2
t_6	n_2

Optimize on the total cost of execution of tasks

→ because t_1 takes 5 s on n_1 & 10s on n_2

Serial approach (t_1 on n_1) (t_2 on n_1)

Total Execution cost = 5 + 2 + 4 + 3 + 2 + 4 = 20

Total inter-task communication cost = 0 + (t_1-t_3)

If two-task communicate n time and if the average time for each inter task communication is t, then the inter task communication cost for the two-task = nxt

2.3 Components

1. A transfer policy
2. " selection "
3. " location "
4. " Information "

(1) A transfer policy

When to transfer a load?

When to know if node is overloaded? (threshold)

T (threshold)

A node is maintaining a queue/any d.s. If T is 10

one more req comes that makes it 11, now node is overloaded it will become a Sender & will transfer to another node

Q. Who will be the receiver?
Node which has adequate remaining req (good diff. b/w Threshold & no of executing req.) then that will become the receiver

If we don't decide on sender & receiver, the system will become Imbalanced transfer policy imp

Q. In which scenario does a system become imbalanced?

(2) A selection policy
⇒ How to select
Individually
Collectively → Polling
Processor thrashing problem (heavily loaded ko dediye hi bhej dega)

(3) A location policy
too much time, bandwidth consumed
If selected the one which is too far, transm" take too much
Select a closer one → comm" cost as parameter, hop count
(1:1 b/w 2 subnets)

(4) An information policy
Maintaining global state inform" → can we decide Scheduling?

RR → preemptive
Fair chance to all proc.
" " " new process

a) Demand-driven (record/change on demand)
b) Periodic (at reg int take info about nodes)
c) State change driven (change in state then record)

This helps in selecting the ~~nodes~~ nodes

2.4 Load Distribution Algorithm

Imp to make the system stable

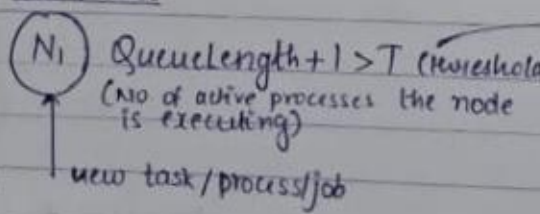
→ System State Information

How many nodes are there & how many processes (running) on each node

1) Static/Deterministic
2) Dynamic
3) Adaptive

⇒ Task/Job/Process to be transferred

→ New Process



Round Robin Scheduling

processes that are transferred / big processes → partially executed one transferred
new process → QueueLength ↑

Preemptive / Migration: Gives chance to large processes to execute as well

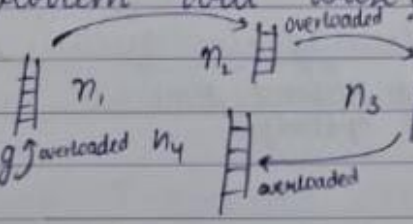
Non preemptive / Non migratory: New process added to be migrated

Q. What info^{n} will be sent from 1 node to another when transferring a large process?
So much inform^{n} PCB, File ptrs, Virtual Mm etc than new processes handling

No scenario of breaking any process in non preemptive (problem of thrashing also)

⇒ Stability of Algo
Factor → Effectiveness (measures if an algo is stable)

Q If proc capability too less than no. of processes, what problem will arise?



heavily loaded system
↓
actual capability of handling load by node > actual load

How will your system perform?

Ans- Queue will become overloaded → system will face a problem of Instability (all nodes became overloaded)
Processor thrashing (select any node, that will X execute your process & at the same time transfer the process to another node again & again making it unstable)

Q. What if the system is underloaded?
Ans- Get all tasks ~~to node~~ some nodes rest nodes idle
Some nodes heavily loaded, rest nodes idle.

If I have transferred node to another node but its not capable enough → check for another node

Divide load equally by balancing technique

⇒ 4 main components

① Transfer policy
Every node maintains a queue
2 categories of nodes — sender (Queue+1 > T → that node becomes sender) / receiver (Queue+1 < T) (ready for receiving job)
(way to decide)

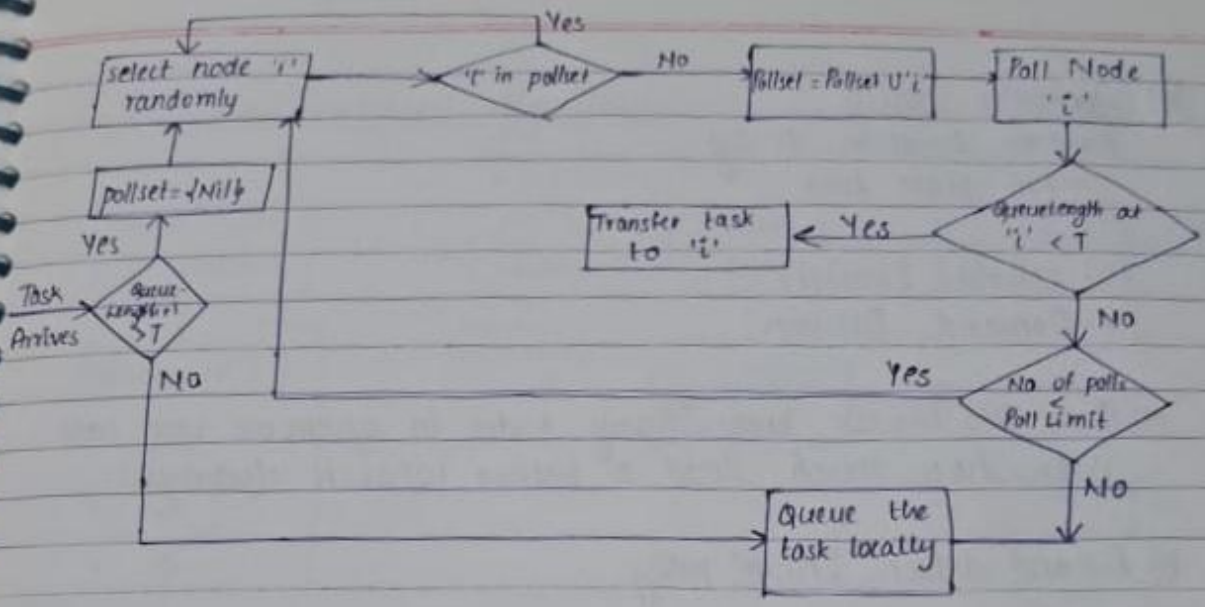
② Location policy (if receiver is ready)
Decides on which node will become sender & receiver (if sender is ready)

③ Selection policy:
New task is selected, one already in queue etc for migration
Advantage: Response time optimized

④ Information policy
About state info". whether collecting state inform"; whether have sufficient inform" in system
→ Info about state
→ From where? (will we get the info)
→ What?

I * Sender-Initiated Algorithm
Tries to find one receiver in the D.S that can execute my task
Proposed by ~~Peterson~~ Eager, Lazowska, Zahorjan (1986)
IEEE transaction on Software Engineering

Pollset list : Like visited list → track checked nodes
Poll limit Select nodes randomly until limit



① Transfer policy

Only considering new task, if new task → Threshold exceeds → makes that node as sender

Receiver T does not exceed → Receiver

Non preemptive algo

② Selec^{n} policy

New task selected

∴ Non preemptive

③ Loc^{n} policy

1) Random
2) Threshold
3) Shortest (Many nodes eligible, out of those receivers the one selected will be one w/ least jobs → shortest in terms of the no. of processes to be executed in the receiver node)

(IV) Information policy

Random location policy
→ No state info

Threshold / Shortest
→ Demand Driven

a) Static Info° Bliy: we know how many nodes in system, we can calc info, how much load, X bother w/ state change

b) Demand driven Inform° policy

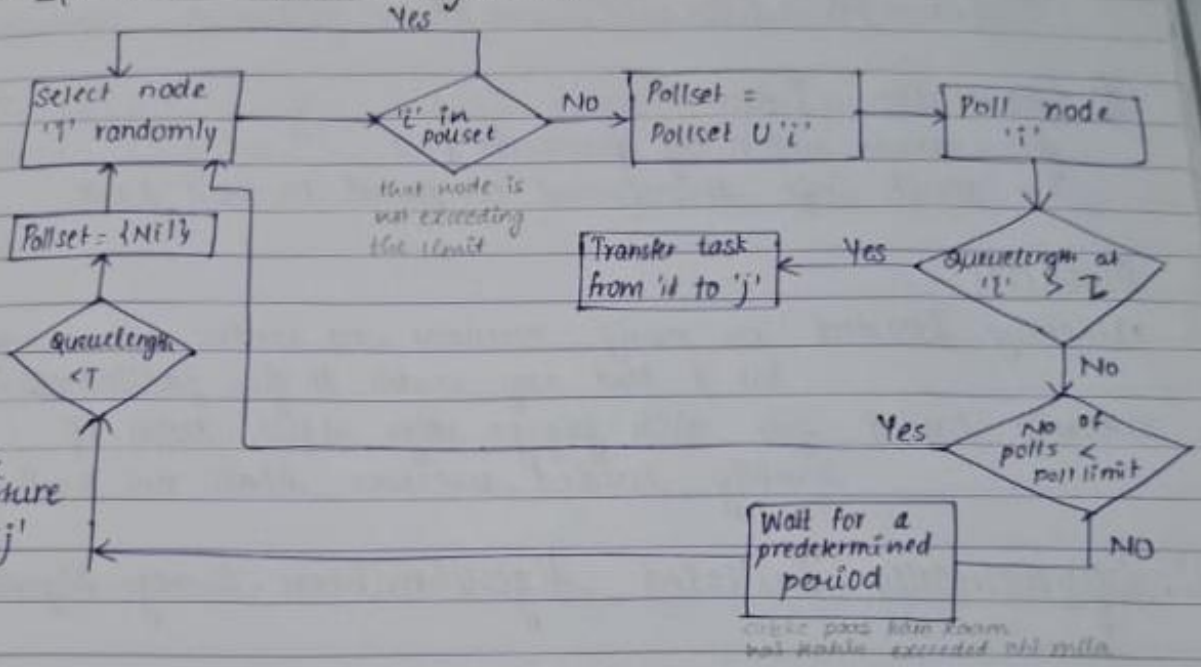
c) Dynamic I·P : Sender/Receiver invoke change in state

Q Is this algo stable? When stable when not?
A- Check if system is effective in underloaded & overloaded system

Underloaded: Too much space in every receiver node sender will get 1 receiver so this is "best sol" for underloaded because even if we select randomly we will get a receiver

Heavily loaded: Even poll limit exceeded, no one ~~tur~~ ready to accept the job, ~~pr~~ process moving from 1 node to another → processor thrashing

II Receiver Initiated Algorithm



Receiver ready to ~~send~~ accept job of sender whose threshold limit is exceeded (Mai khali baitha hoon mujhe kaam do)

Proposed by Shivaratri & Krueger in 1990

- Receiver is ready
- Searching for sender whose T cond" is exceeding

(I) Transfer Policy

threshold cond" khatam hogayi

① Receiver (on departure of a task/~~arrival~~ min" of a job/task → trigger)

② Sender

(II) Selection Policy

→ Non preemptive (New task)

→ preemptive (old task/Subportion of a task) ✓ More effective

Long processes are target processes portion execute karke queue khali hogayi

(III) Location policy
Random / Threshold find a node jiska value exceed no rahe hai not randomly

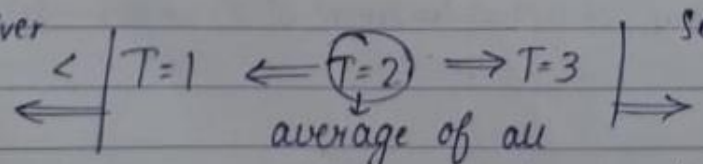
(II) Information Policy
Sun → execute remaining processes' part
Too much info transferred compared to new task

STABILITY?
1) Lightly loaded too many receivers, no sender, wait for long but X find any sender to give job (Unstable)
2) Heavily loaded: job milta jayega (Algo stable even if heavily loaded, receiver khali nhi baithega

III. Symmetrically Initiated Algorithm (Above-Average-Algorithm)

① Too High
② Too Low (too less load → become receiver)
③ Accept (wait then accept → when too low, too high → send)
④ Awaiting Task
⑤ Change-Average

Proposed by Krueger & Finkel
Create scenario where there are senders finding receivers & vice versa



average of all node's queue length & no of processes

HW → Adaptive Algos
Load Sharing policies
→ The V-system
→ The Sprite system
→ condor
→ The stealth Distributed scheduler

Study Yourself from Book

Decide Threshold range, usse bohut neeche chale gaye → become receiver, too much

Threshold range keeps changing as no. of processes keep changing

: Now were absent

* Mu