Resource Management in Distributed Systems

Task assignment,
Load-balancing and Load-sharing

Introduction

- Distributed systems contain a set of resources interconnected by a network
- Processes are migrated to fulfill their resource requirements
- Resource manager are to control the assignment of resources to processes
- Resources can be logical (shared file) or physical (CPU)
- We consider a resource to be a processor

Types of process scheduling techniques

- Task assignment approach
  - User processes are collections of related tasks
  - Tasks are scheduled to improve performance
- Load-balancing approach
  - Tasks are distributed among nodes so as to equalize the workload of nodes of the system
- Load-sharing approach
  - Simply attempts to avoid idle nodes while processes wait for being processed

Desirable features of a scheduling algorithm I.

- No A Priori Knowledge about Processes
  - User does not want to specify information about characteristic and requirements
- Dynamic in nature
  - Decision should be based on the changing load of nodes and not on fixed static policy
- Quick decision-making capability
  - Algorithm must make quick decision about the assignment of task to nodes of system

Desirable features of a scheduling algorithm II.

- Balanced system performance and scheduling overhead
  - Great amount of information gives more intelligent decision, but increases overhead
- Stability
  - Unstable when all processes are migrating without accomplishing any useful work
  - It occurs when the nodes turn from lightly-loaded to heavily-loaded state and vice versa

Desirable features of a scheduling algorithm III.

- Scalability
  - A scheduling algorithm should be capable of handling small as well as large networks
- Fault tolerance
  - Should be capable of working after the crash of one or more nodes of the system
- Fairness of Service
  - More users initiating equivalent processes expect to receive the same quality of service

Task assignment approach

- Main assumptions
  - Processes have been split into tasks
  - Computation requirement of tasks and speed of processors are known
  - Cost of processing tasks on nodes are known
  - Communication cost between every pair of tasks are known
  - Resource requirements and available resources on node are known
  - Reassignment of tasks are not possible

Techniques for scheduling processes

There are three techniques for scheduling processes of a distributed system:

1) ***Task Assignment Approach,*** in which each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance.

2) ***Load-balancing approach,*** in which all the processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes.

3) ***Load-sharing approach,*** which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

Note:- The task assignment approach has limited applicability to practical situations because it works on the assumption that the characteristics (e.g. execution time, IPC costs etc) of all the processes to be scheduled are known in advance.

Task assignment approach

- Basic idea: Finding an optimal assignment to achieve goals such as the following:
  - Minimization of IPC costs
  - Quick turnaround time of process
  - High degree of parallelism
  - Efficient utilization of resources

Task assignment approach example

There are two nodes, {n1, n2} and six tasks {t1, t2, t3, t4, t5, t6}. There are two task assignment parameters – the task execution cost (x_{ab} the cost of executing task a on node b) and the inter-task communication cost (c_{ij} the inter-task communication cost between tasks i and j).

Inter-task communication cost

	t1	t2	t3	t4	t5	t6
t1	0	6	4	0	0	12
t2	6	0	8	12	3	0
t3	4	8	0	0	11	0
t4	0	12	0	0	5	0
t5	0	3	11	5	0	0
t6	12	0	0	0	0	0

Execution costs

	n1	n2
t1	5	10
t2	2	
t3	4	4
t4	6	3
t5	5	2
t6		4

Task t6 cannot be executed on node n1 and task t2 cannot be executed on node n2 since the

Task assignment approach example

1) **Serial assignment**, where tasks t1, t2, t3 are assigned to node n1 and tasks t4, t5, t6 are assigned to node n2:

Execution cost, $x = x_{11} + x_{21} + x_{31} + x_{42} + x_{52} + x_{62} = 5 + 2 + 4 + 3 + 2 + 4 = 20$

Communication cost, $c = c_{14} + c_{15} + c_{16} + c_{24} + c_{25} + c_{26} + c_{34} + c_{35} + c_{36} = 0 + 0 + 12 + 12 + 3 + 0 + 0 + 11 + 0 = 38$. Hence total cost = 58.

2) **Optimal assignment**, where tasks t1, t2, t3, t4, t5 are assigned to node n1 and task t6 is assigned to node n2.

Execution cost, $x = x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{62}$

$= 5 + 2 + 4 + 6 + 5 + 4 = 26$

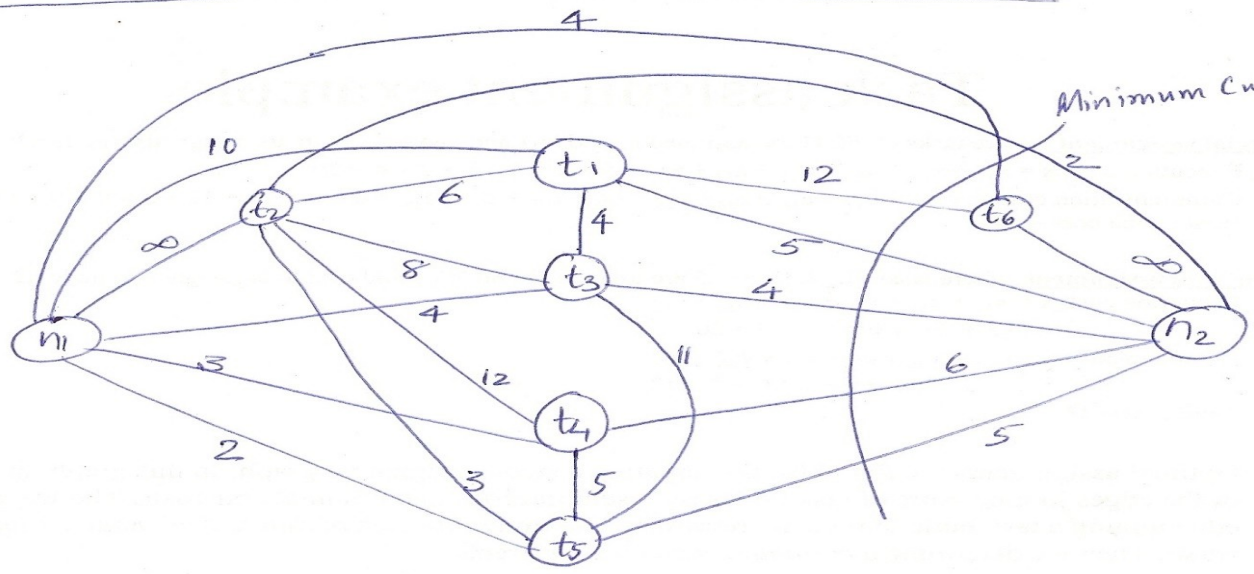
Communication cost, $c = c_{16} + c_{26} + c_{36} + c_{46} + c_{56}$

$= 12 + 0 + 0 + 0 + 0 = 12$

Total cost = 38

Optimal Task assignment

Assignment Graph with Minimum Cost Cut



Min Cutset = 2 + 4 + 12 + 5 + 4 + 6 + 5
= 38

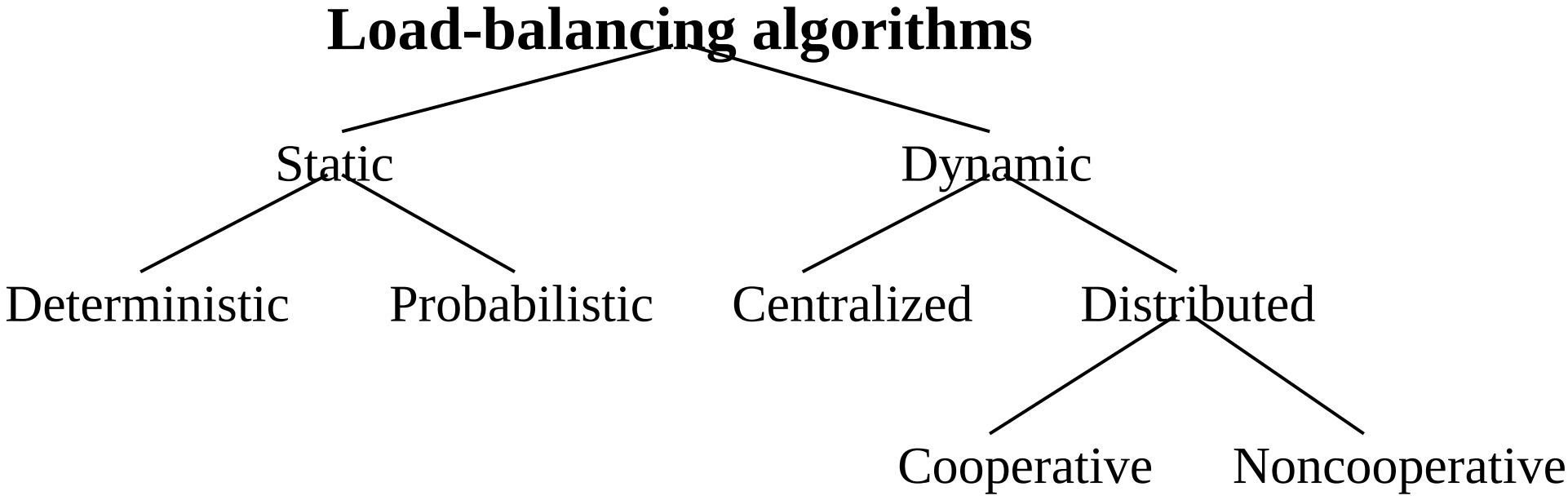
Task assignment approach

Optimal assignments are found by first creating a static assignment graph. In this graph, the weights of the edges joining pairs of task nodes represent inter-task communication costs. **The weight on the edge joining a task node to node n1 represents *the execution cost of that task on node n2 and vice-versa***. Then we determine a *minimum cutset* in this graph.

A ***cutset*** is defined to be a set of edges such that when these edges are removed, the nodes of the graph are partitioned into two disjoint subsets such that nodes in one subset are reachable from n1 and the nodes in the other are reachable from n2. Each task node is reachable from either n1 or n2. The weight of a cutset is the sum of the weights of the edges in the cutset. This sums up the execution and communication costs for that assignment. An optimal assignment is found by finding a minimum cutset.

Load-balancing approach

A Taxonomy of Load-Balancing Algorithms



Load-balancing approach

Type of load-balancing algorithms

- Static versus Dynamic
  - Static algorithms use only information about the average behavior of the system
  - Static algorithms ignore the current state or load of the nodes in the system
  - Dynamic algorithms collect state information and react to system state if it changed
  - Static algorithms are much more simpler
  - Dynamic algorithms are able to give significantly better performance

Load-balancing approach

Type of static load-balancing algorithms

- Deterministic versus Probabilistic
  - Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled
  - Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules
  - Deterministic approach is difficult to optimize
  - Probabilistic approach has poor performance

Load-balancing approach

Type of dynamic load-balancing algorithms

- Centralized versus Distributed
  - Centralized approach collects information to server node and makes assignment decision
  - Distributed approach contains entities to make decisions on a predefined set of nodes
  - Centralized algorithms can make efficient decisions, have lower fault-tolerance
  - Distributed algorithms avoid the bottleneck of collecting state information and react faster

Load-balancing approach

Type of distributed load-balancing algorithms

- Cooperative versus Noncooperative
  - In Noncooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities
  - In Cooperative algorithms distributed entities cooperate with each other
  - Cooperative algorithms are more complex and involve larger overhead
  - Stability of Cooperative algorithms are better

Issues in designing Load-balancing algorithms

- Load estimation policy
  - determines how to estimate the workload of a node
- Process transfer policy
  - determines whether to execute a process locally or remote
- State information exchange policy
  - determines how to exchange load information among nodes
- Location policy
  - determines to which node the transferable process should be sent
- Priority assignment policy
  - determines the priority of execution of local and remote processes
- Migration limiting policy
  - determines the total number of times a process can migrate

Load estimation policy I.

for Load-balancing algorithms

- To balance the workload on all the nodes of the system, it is necessary to decide how to measure the workload of a particular node
- Some measurable parameters (with time and node dependent factor) can be the following:
  - Total number of processes on the node
  - Resource demands of these processes
  - Instruction mixes of these processes
  - Architecture and speed of the node’s processor
- Several load-balancing algorithms use the total number of processes to achieve big efficiency

Load estimation policy II.

for Load-balancing algorithms

- In some cases the true load could vary widely depending on the remaining service time, which can be measured in several way:
  - *Memoryless method* assumes that all processes have the same expected remaining service time, independent of the time used so far
  - *Pastrepeats* assumes that the remaining service time is equal to the time used so far
  - *Distribution method* states that if the distribution service times is known, the associated process's remaining service time is the expected remaining time conditioned by the time already used

Load estimation policy III.

for Load-balancing algorithms

- None of the previous methods can be used in modern systems because of periodically running processes and daemons
- An acceptable method for use as the load estimation policy in these systems would be to measure the CPU utilization of the nodes
- Central Processing Unit utilization is defined as the number of CPU cycles actually executed per unit of real time
- It can be measured by setting up a timer to periodically check the CPU state (idle/busy)

Process transfer policy I.

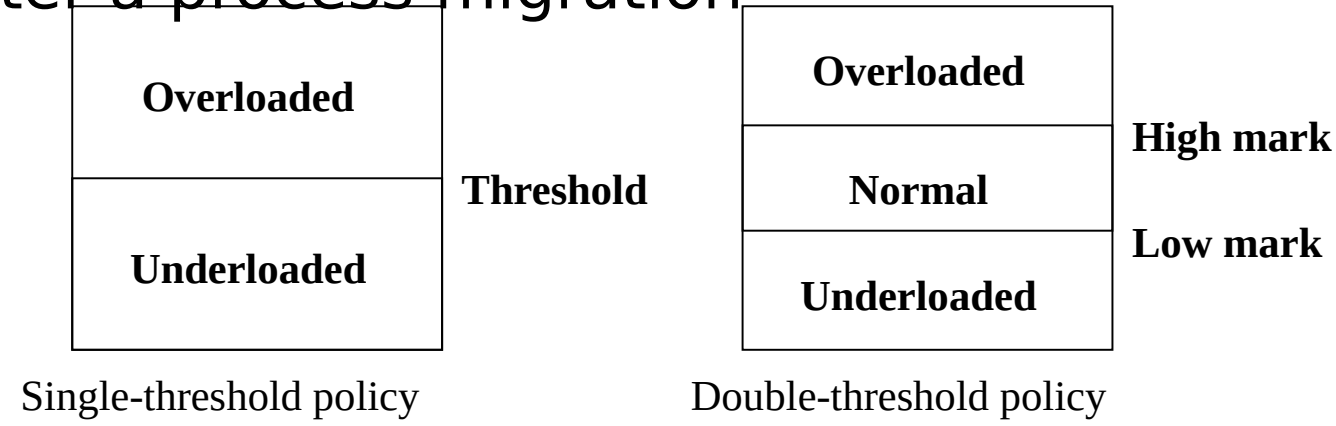
for Load-balancing algorithms

- Most of the algorithms use the *threshold policy* to decide on whether the node is lightly-loaded or heavily-loaded
- Threshold value is a limiting value of the workload of node which can be determined by
  - Static policy: predefined threshold value for each node depending on processing capability
  - Dynamic policy: threshold value is calculated from average workload and a predefined constant
- Below threshold value node accepts processes to execute, above threshold value node tries to transfer processes to a lightly-loaded node

Process transfer policy II.

for Load-balancing algorithms

- Single-threshold policy may lead to unstable algorithm because underloaded node could turn to be overloaded right after a process migration



Single-threshold policy Double-threshold policy

- To reduce instability double-threshold policy has been

Process transfer policy III.

for Load-balancing algorithms

- Double threshold policy
  - When node is in overloaded region new local processes are sent to run remotely, requests to accept remote processes are rejected
  - When node is in normal region new local processes run locally, requests to accept remote processes are rejected
  - When node is in underloaded region new local processes run locally, requests to accept remote processes are accepted

Location policy I.

for Load-balancing algorithms

- Threshold method
  - Policy selects a random node, checks whether the node is able to receive the process, then transfers the process. If node rejects, another node is selected randomly. This continues until probe limit is reached.
- Shortest method
  - L distinct nodes are chosen at random, each is polled to determine its load. The process is transferred to the node having the minimum value unless its workload value prohibits to accept the process.
  - Simple improvement is to discontinue probing whenever a node with zero load is encountered.

Location policy II.

for Load-balancing algorithms

- Bidding method
  - Nodes contain managers (to send processes) and contractors (to receive processes)
  - Managers broadcast a request for bid, contractors respond with bids (prices based on capacity of the contractor node) and manager selects the best offer
  - Winning contractor is notified and asked whether it accepts the process for execution or not
  - Full autonomy for the nodes regarding scheduling
  - Big communication overhead
  - Difficult to decide a good pricing policy

Location policy III.

for Load-balancing algorithms

- Pairing
  - Contrary to the former methods the pairing policy is to reduce the variance of load only between pairs
  - Each node asks some randomly chosen node to form a pair with it
  - If it receives a rejection it randomly selects another node and tries to pair again
  - Two nodes that differ greatly in load are temporarily paired with each other and migration starts
  - The pair is broken as soon as the migration is over
  - A node only tries to find a partner if it has at least two processes

State information exchange policy

I. for Load-balancing algorithms

- Dynamic policies require frequent exchange of state information, but these extra messages arise two opposite impacts:
  - Increasing the number of messages gives more accurate scheduling decision
  - Increasing the number of messages raises the queuing time of messages
- State information policies can be the following:
  - Periodic broadcast
  - Broadcast when state changes
  - On-demand exchange
  - Exchange by polling

State information exchange policy II. for Load-balancing algorithms

- Periodic broadcast
  - Each node broadcasts its state information after the elapse of every *T* units of time
  - Problem: heavy traffic, fruitless messages, poor scalability since information exchange is too large for networks having many nodes
- Broadcast when state changes
  - Avoids fruitless messages by broadcasting the state only when a process arrives or departures
  - Further improvement is to broadcast only when state switches to another region (double-threshold policy)

State information exchange policy III. for Load-balancing algorithms

- On-demand exchange
  - In this method a node broadcast a State-Information-Request message when its state switches from normal to either underloaded or overloaded region.
  - On receiving this message other nodes reply with their own state information to the requesting node
  - Further improvement can be that only those nodes reply which are useful to the requesting node
- Exchange by polling
  - To avoid poor scalability (coming from broadcast messages) the partner node is searched by polling the other nodes on by one, until poll limit is reached

Priority assignment policy for Load-balancing algorithms

- Selfish
  - Local processes are given higher priority than remote processes. Worst response time performance of the three policies.
- Altruistic
  - Remote processes are given higher priority than local processes. Best response time performance of the three policies.
- Intermediate
  - When the number of local processes is greater or equal to the number of remote processes, local processes are given higher priority than remote processes. Otherwise, remote processes are given higher priority than local processes.

Migration limiting policy

for Load-balancing algorithms

- This policy determines the total number of times a process can migrate
  - Uncontrolled
    - A remote process arriving at a node is treated just as a process originating at a node, so a process may be migrated any number of times
  - Controlled
    - Avoids the instability of the uncontrolled policy
    - Use a *migration count* parameter to fix a limit on the number of time a process can migrate
    - Irrevocable migration policy: *migration count* is fixed to 1
    - For long execution processes *migration count* must be greater than 1 to adapt for dynamically changing states

Load-sharing approach

- Drawbacks of Load-balancing approach
  - Load balancing technique with attempting equalizing the workload on all the nodes is not an appropriate object since big overhead is generated by gathering exact state information
  - Load balancing is not achievable since number of processes in a node is always fluctuating and temporal unbalance among the nodes exists every moment
- Basic ideas for Load-sharing approach
  - It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes
  - Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when heavily node exists
  - Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms

Load estimation policies

for Load-sharing algorithms

- Since load-sharing algorithms simply attempt to avoid idle nodes, it is sufficient to know whether a node is busy or idle
- Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes
- In modern systems where permanent existence of several processes on an idle node is possible, algorithms measure CPU utilization to estimate the load of a node

Process transfer policies

for Load-sharing algorithms

- Algorithms normally use all-or-nothing strategy
- This strategy uses the threshold value of all the nodes fixed to 1
- Nodes become receiver node when it has no process, and become sender node when it has more than 1 process
- To avoid processing power on nodes having zero process load-sharing algorithms use a threshold value of 2 instead of 1
- When CPU utilization is used as the load estimation policy, the double-threshold policy should be used as the process transfer policy

Location policies I.

for Load-sharing algorithms

- Location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system, and this policy can be the following:
  - Sender-initiated location policy
    - Sender node decides where to send the process
    - Heavily loaded nodes search for lightly loaded nodes
  - Receiver-initiated location policy
    - Receiver node decides from where to get the process
    - Lightly loaded nodes search for heavily loaded nodes

Location policies II.

for Load-sharing algorithms

- Sender-initiated location policy
  - Node becomes overloaded, it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes
  - When broadcasting, suitable node is known as soon as reply arrives
- Receiver-initiated location policy
  - Nodes becomes underloaded, it either broadcast or randomly probes the other nodes one by one to indicate its willingness to receive remote processes
- Receiver-initiated policy require preemptive process migration facility since scheduling decisions are usually made at process departure epochs

Location policies III.

for Load-sharing algorithms

- Experiences with location policies
  - Both policies gives substantial performance advantages over the situation in which no load-sharing is attempted
  - Sender-initiated policy is preferable at light to moderate system loads
  - Receiver-initiated policy is preferable at high system loads
  - Sender-initiated policy provide better performance for the case when process transfer cost significantly more at receiver-initiated than at sender-initiated policy due to the preemptive transfer of processes

State information exchange policies for Load-sharing algorithms

- In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information, but needs to know the state of other nodes when it is either underloaded or overloaded
- Broadcast when state changes
  - In sender-initiated/receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded/underloaded
  - It is called broadcast-when-idle policy when receiver-initiated policy is used with fixed threshold value value of 1
- Poll when state changes
  - In large networks polling mechanism is used
  - Polling mechanism randomly asks different nodes for state information until find an appropriate one or probe limit is reached
  - It is called poll-when-idle policy when receiver-initiated policy is

SUMMARY

- Resource manager of a distributed system schedules the processes to optimize combination of resources usage, response time, network congestion, scheduling overhead
- Three different approaches has been discussed
  - Task assignment approach deals with the assignment of task in order to minimize inter process communication costs and improve turnaround time for the complete process, by taking some constraints into account
  - In load-balancing approach the process assignment decisions attempt to equalize the avarage workload on all the nodes of the system
  - In load-sharing approach the process assignment decisions attempt to keep all the nodes busy if there are sufficient processes in the system for all the nodes