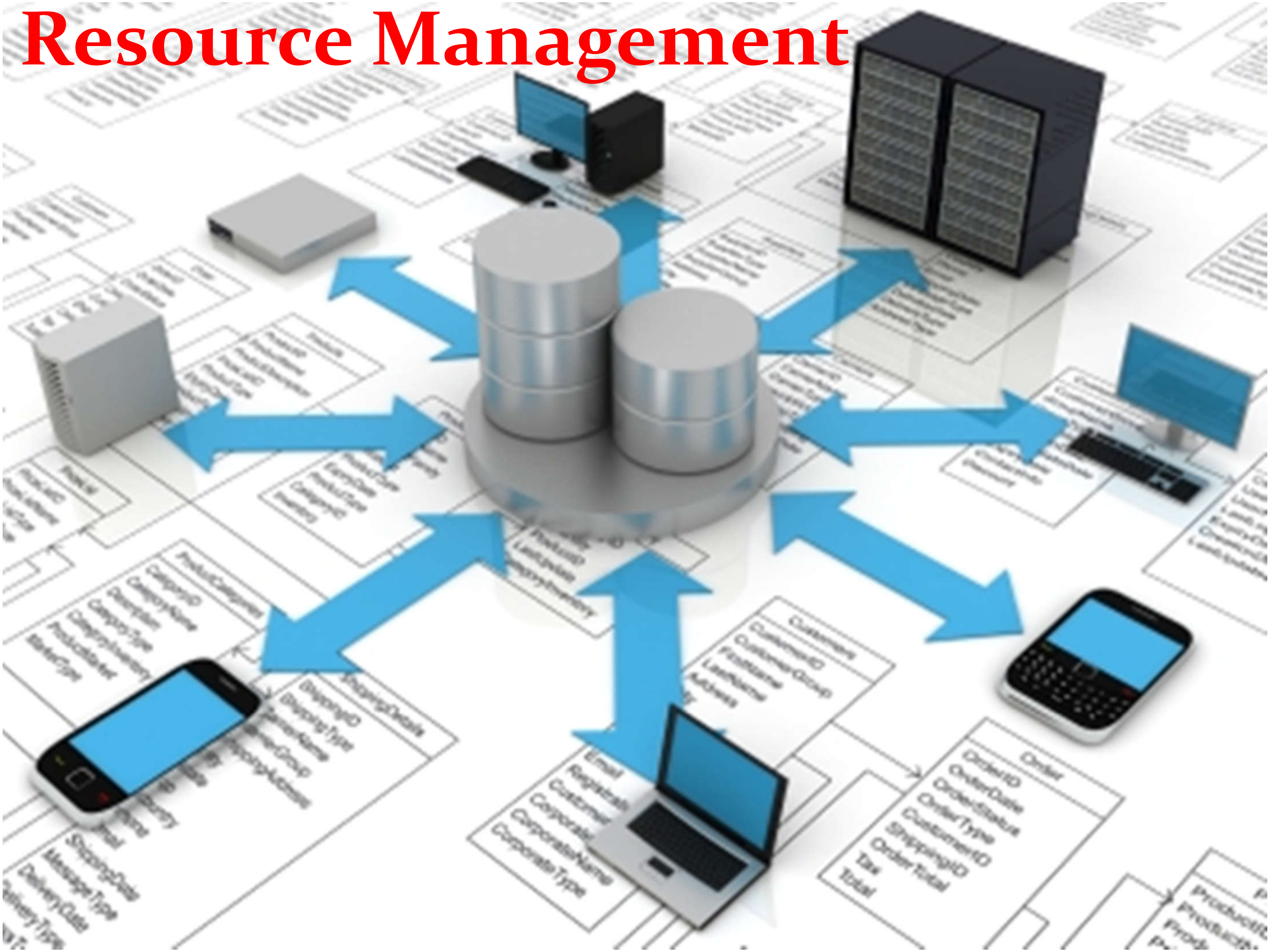
Resource Management



Resource Management

- A resource can be a logical, such as a shared file, or physical, such as a CPU (a node of the distributed system).
- One of the functions of a distributed operating system is to assign processes to the nodes (resources) of the distributed system such that the resource usage, response time, network congestion, and scheduling overhead are optimized.

Resource Management

- 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.

- 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.

Desirable features of a good global scheduling algorithm

- No a priori knowledge about the processes
- Dynamic in nature
- Quick decision making capability
- Balanced system performance and scheduling overhead
- Stability
- Scalability
- Fault tolerance
- Fairness of service

Desirable features of a good global scheduling algorithm

- **No a priori knowledge about the processes**

  Scheduling algorithms that operate based on the information about the characteristics and resource requirements of the processes pose an extra burden on the users who must provide this information while submitting their processes for execution.

- **Dynamic in nature**

  Process assignment decisions should be dynamic, I.e., be based on the current load of the system and not on some static policy. It is recommended that the scheduling algorithm possess the flexibility to migrate a process more than once because the initial decision of placing a process on a particular node may have to be changed after some time to adapt to the new system load.

Desirable features of a good global scheduling algorithm

- **Quick decision making capability**

  Heuristic methods requiring less computational efforts (and hence less time) while providing near-optimal results are preferable to exhaustive (optimal) solution methods.

- **Balanced system performance and scheduling overhead**

  Algorithms that provide near-optimal system performance with a minimum of global state information (such as CPU load) gathering overhead are desirable.

  This is because the overhead increases as the amount of global state information collected increases.

  This is because the usefulness of that information is decreased due to both the aging of the information being gathered and the low scheduling frequency as a result of the cost of gathering and processing the extra information.

Desirable features of a good global scheduling algorithm

- **Stability**

  Fruitless migration of processes, known as processor thrashing, must be prevented.

  Example. If nodes n1 and n2 observe that node n3 is idle and then offload a portion of their work to n3 without being aware of the offloading decision made by the other node. Now if n3 becomes overloaded due to this it may again start transferring its processes to other nodes.

  This is caused by scheduling decisions being made at each node independently of decisions made by other nodes.

Desirable features of a good global scheduling algorithm

- **Scalability**

  A scheduling algorithm should scale well as the number of nodes increases.

  An algorithm that makes scheduling decisions by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability.

  This will work fine only when there are few nodes in the system.

  This is because the inquirer receives a flood of replies almost simultaneously, and the time required to process the reply messages for making a node selection is too long as the number of nodes (*N*) increase.

  Also the network traffic quickly consumes network bandwidth.

  A simple approach is to probe only *m* of *N* nodes for selecting a node.

Desirable features of a good global scheduling algorithm

- **Fault tolerance**

  A good scheduling algorithm should not be disabled by the crash of one or more nodes of the system.

  Also, if the nodes are partitioned into two or more groups due to link failures, the algorithm should be capable of functioning properly for the nodes within a group.

  Algorithms that have decentralized decision making capability and consider only available nodes in their decision making have better fault tolerance capability.

Desirable features of a good global scheduling algorithm

- **Fairness of service**

  Global scheduling policies that blindly attempt to balance the load on all the nodes of the system are not good from the point of view of fairness of service.

  This is because in any load-balancing scheme, heavily loaded nodes will obtain all the benefits while lightly loaded nodes will suffer poorer response time than in a stand-alone configuration.

  A fair strategy that improves response time of the former without unduly affecting the latter is desirable.

  Hence load-balancing has to be replaced by the concept of load sharing, that is, a node will share some of its resources as long as its users are not significantly affected.

Task assignment Approach

- Each process is viewed as a collection of tasks. These tasks are scheduled to suitable processor to improve performance. This is not a widely used approach because:
  - It requires characteristics of all the processes to be known in advance.
  - This approach does not take into consideration the dynamically changing state of the system.
- In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process. The following are typical assumptions for the task assignment approach:
  - Minimize IPC cost (this problem can be modeled using network flow model)
  - Efficient resource utilization
  - Quick turnaround time
  - A high degree of parallelism

Task Assignment Approach

- **Assumptions:**

  1) A process has already been split up into pieces called tasks. This split occurs along natural boundaries (such as a method), so that each task will have integrity in itself and data transfers among the tasks are minimized.

  2) The amount of computation required by each task and the speed of each CPU are known.

  3) The cost of processing each task on every node is known. This is derived from assumption 2.

  4) The IPC costs between every pair of tasks is known. The IPC cost is 0 for tasks assigned to the same node. This is usually estimated by an analysis of the static program. If two tasks communicate n times and the average time for each inter-task communication is t, them IPC costs for the two tasks is n * t.

  5) Precedence relationships among the tasks are known.

  6) Reassignment of tasks is not possible.

Task Assignment Approach

- **Goal:**

  Goal is to assign the tasks of a process to the nodes of a distributed system in such a manner as to achieve goals such as the following goals:

  - Minimization of IPC costs
  - Quick turnaround time for the complete process
  - A high degree of parallelism
  - Efficient utilization of system resources in general

  These goals often conflict.

  Example, while minimizing IPC costs tends to assign all tasks of a process to a single node, efficient utilization of system resources tries to distribute the tasks evenly among the nodes. So also, quick turnaround time and a high degree of parallelism encourage parallel execution of the tasks, the precedence relationship among the tasks limits their parallel execution.

Task Assignment Approach

Also note that in case of m tasks and q nodes, there are m^q possible assignments of tasks to nodes . In practice, however, the actual number of possible assignments of tasks to nodes may be less than m^q due to the restriction that certain tasks cannot be assigned to certain nodes due to their specific requirements (e.g. need a certain amount of memory or a certain data file).

Task Assignment 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

		Tasks					
		t1	t2	t3	t4	t5	t6
Tasks	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

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

Task Assignment Example

- Task t6 cannot be executed on node n1 and task t2 cannot be executed on node n2 since the resources they need are not available on these nodes.

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.

Task Assignment Example

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 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.

Task Assignment Example

Serial Assignment	
Task	Node
t1	n1
t2	n1
t3	n1
t4	n2
t5	n2
t6	n2

Optimal Assignment	
Task	Node
t1	n1
t2	n1
t3	n1
t4	n1
t5	n1
t6	n2

Task Assignment Example

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.

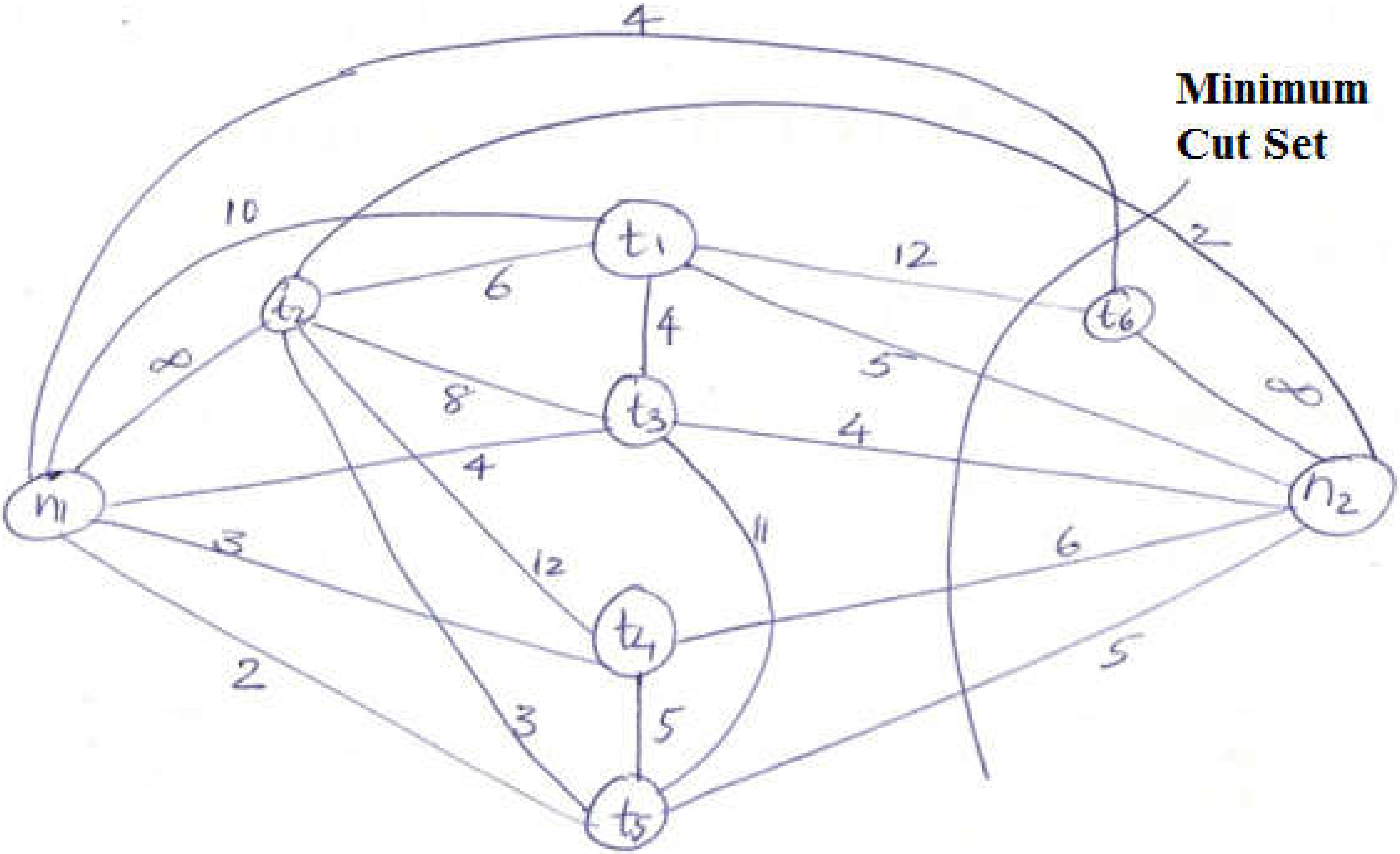
Each task node is reachable from either n1 or n2.

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.

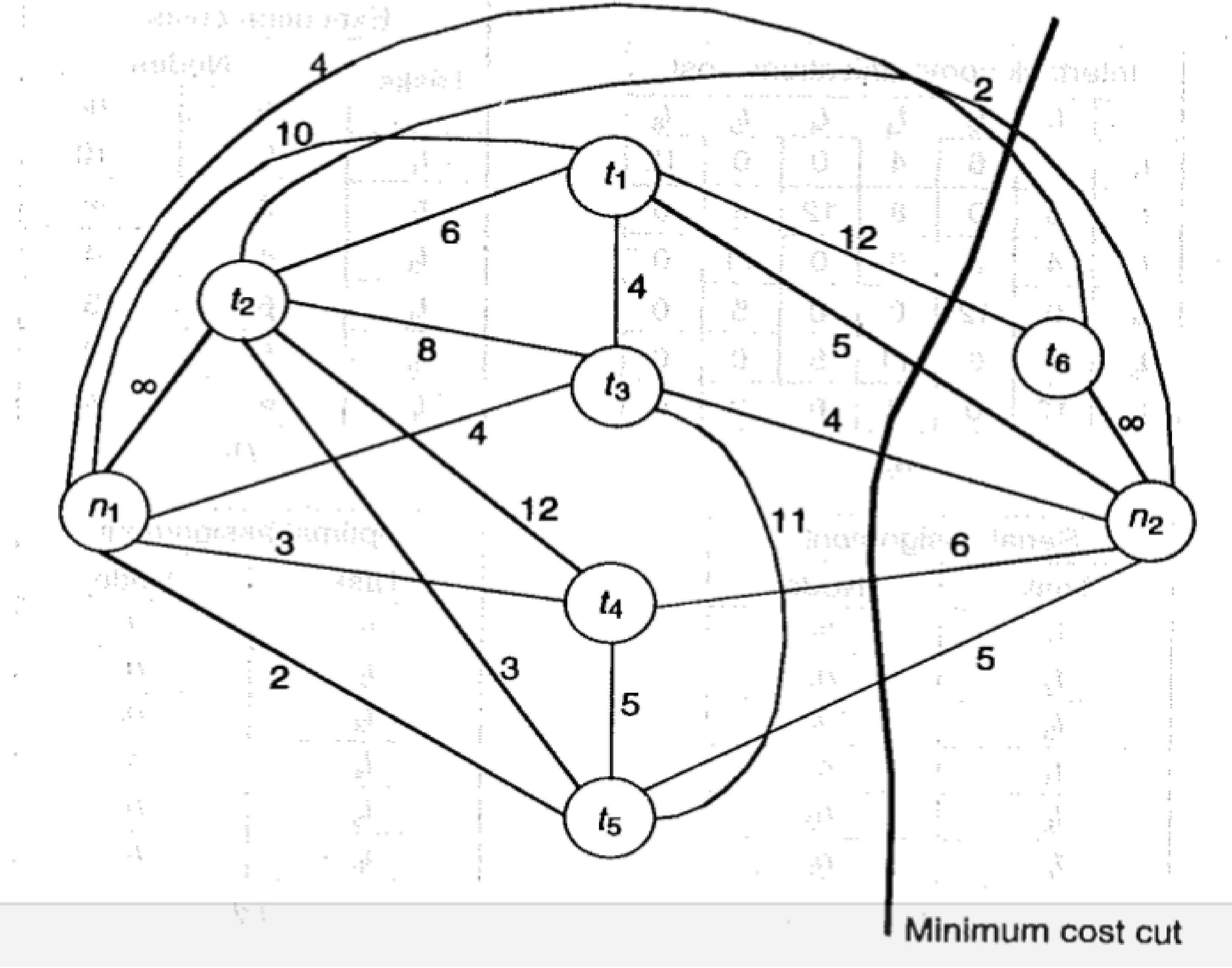
Assignment Graph with Minimum Cost Cut



Minimum Cut Set

Min Cut Set = 2 + 4 + 12 + 5 + 4 + 6 + 5 = 38

Assignment Graph with Minimum Cost Cut


4
10
2
t_1
6
12
4
t_2
8
5
t_6
t_3
∞
4
4
∞
n_1
12
11
n_2
3
6
t_4
2
3
5
5
t_5
Minimum cost cut