Resource Management

Resources in Distributed Systems

- From a user's point of view, the set of available resources in a distributed system acts like a single virtual system.
- A resource can be
  - logical, such as a shared file, or
  - physical, such as a CPU.
- When a user submits a process for execution:
  - The resource manager controls the assignment of resources to processes.
  - The resource manager also routes the processes to suitable nodes (processors) according to assignments.

Process Migration

- Besides data communication, distributed systems facilitates resource sharing by migrating a local process and executing it at a remote node.

- A process may also be executed remotely if the expected response time will be better.

- Two types of migrations:
  - Migrated before execution starts - where to migrate the process and how to manage the resources in the systems.
  - Migrated after execution starts - how to migrate the process.

Desirable Features of Good Scheduling Algorithms

- 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

Stability

- An unstable resource management algorithm leads to process thrashing.
  - Process thrashing means that all the nodes of the system are spending all of their time migrating processes without accomplishing any useful work in an attempt to properly schedule the processes for better performance.
  - Process thrashing may occur
    - When the scheduling algorithms do not consider global system state or
    - When the processes in transit are not considered during the load-balance.
- A good resource management algorithm should avoid process thrashing.

Task Assignment 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 task assignment algorithms seek to assign the resources so as to achieve goals such as:
  - Minimization of IPC costs
  - Quick turnaround/response time
  - A high degree of parallelism
  - Efficient utilization of system resources in general

Task Assignment 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 task assignment algorithms seek to assign the resources so as to achieve goals such as:
  - Minimization of IPC costs
  - Quick turnaround/response time
  - A high degree of parallelism
  - Efficient utilization of system resources in general

centralize

Task Assignment 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 task assignment algorithms seek to assign the resources so as to achieve goals such as:
  - Minimization of IPC costs $\longleftarrow$ centralize
  - Quick turnaround/response time
  - A high degree of parallelism
  - Efficient utilization of system resources in general $\longleftarrow$ parallelism

Assumptions

- A process has already been split into pieces called tasks.
- The amount of computation required by each task and the speed of each processor are known.
- The cost of processing each task on every node of the system is known.
- The IPC costs between every pair of tasks is known.
- Other constraints, such as resource requirements of the tasks and the available resources at each node, precedence relationships among the tasks, and so on, are also known.
- Reassignment of the tasks is generally not possible.

An Example of Assignment Problem

- The system is made up of six tasks $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ and two nodes $\{n_1, n_2\}$.
- The inter-task communication costs (c_{ij}) and the execution costs (x_{ab}) of the tasks are given below:

Inter-task communication costs (c_{ij})

	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

Execution costs (x_{ab})

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

- The goal is to minimize the total cost (communication and execution costs).

Solutions of the Serial and Optimal Assignments

Serial assignment	
Task	**Node**
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_2
t_5	n_2
t_6	n_2

Optimal assignment	
Task	**Node**
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_1
t_5	n_1
t_6	n_2

Serial assignment execution cost (x) = $x_{11} + x_{21} + x_{31} + x_{42} + x_{52} + x_{62}$
= 5 + 2 + 4 + 3 + 2 + 4 = 20

Serial assignment 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

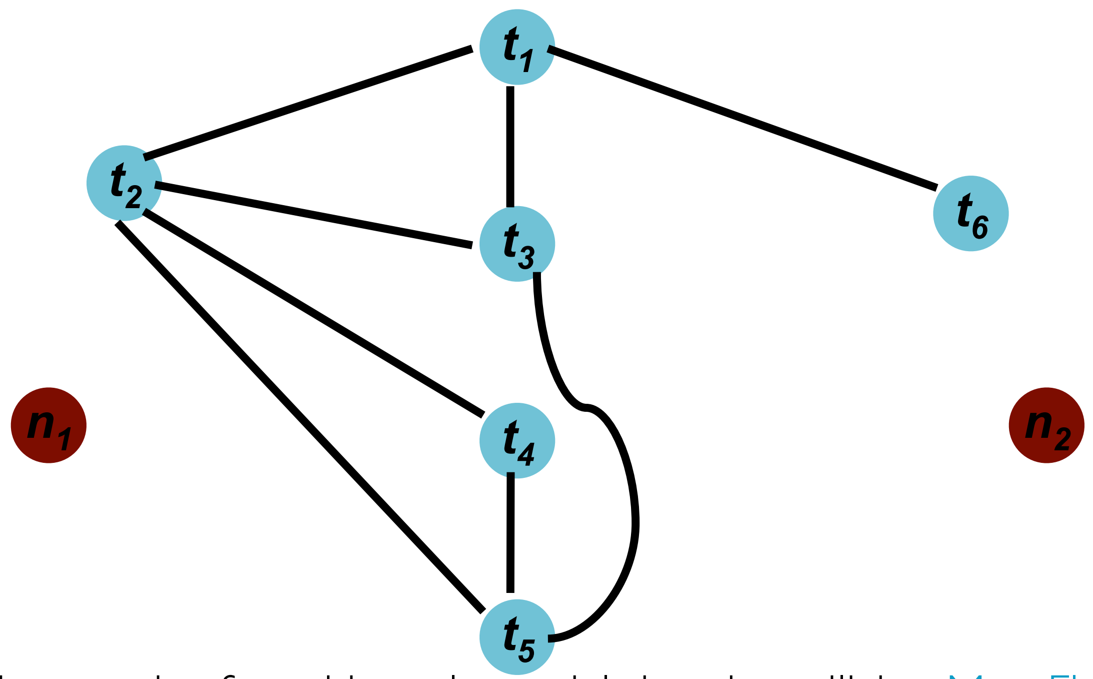
Serial assignment total cost = $x + c$ = 20 + 38 = 58

Optimal assignment execution cost (x) = $x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{62}$
= 5 + 2 + 4 + 6 + 5 + 4 = 26

Optimal assignment communication cost (c) = $c_{16} + c_{26} + c_{36} + c_{46} + c_{56}$
= 12 + 0 + 0 + 0 + 0 = 12

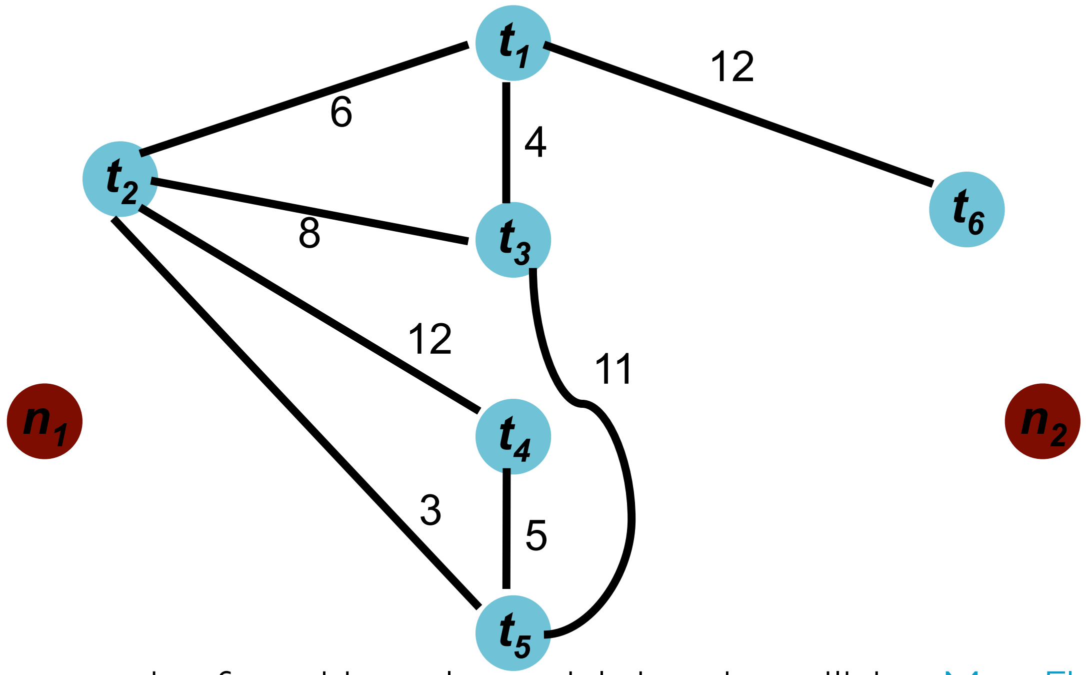
Optimal assignment total cost = $x + c$ = 26 + 12 = 38

Assignment Graph with Minimum Cost Cut



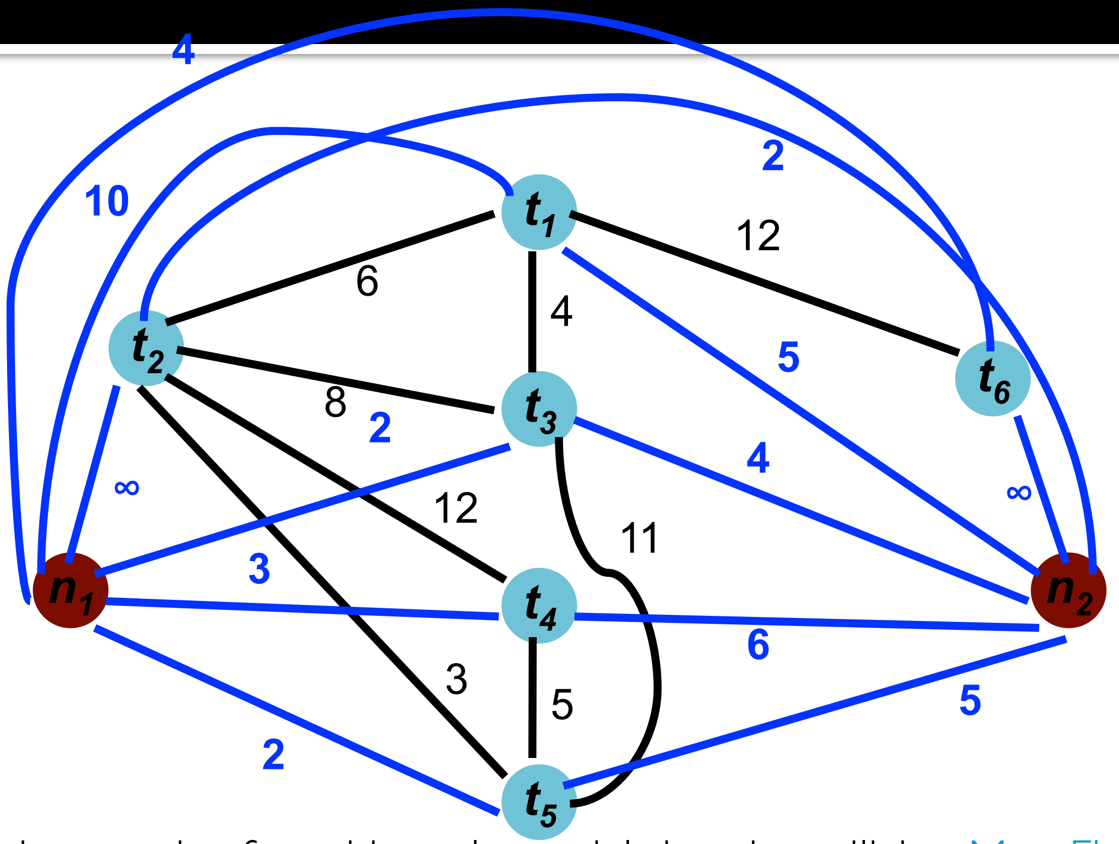
- An optimal solution can be found in polynomial time by utilizing Max-Flow/Min-Cut algorithm.
- However, the problem is known to be NP-hard for an arbitrary number of execution nodes.
  - Lo, V.M., "Heuristic algorithms for task assignment in distributed systems," *Computers, IEEE Transactions on* , vol.37, no.11, pp.1384,1397, Nov 1988.

Assignment Graph with Minimum Cost Cut



- An optimal solution can be found in polynomial time by utilizing Max-Flow/Min-Cut algorithm.
- However, the problem is known to be NP-hard for an arbitrary number of execution nodes.
    - Lo, V.M., "Heuristic algorithms for task assignment in distributed systems," *Computers, IEEE Transactions on* , vol.37, no.11, pp.1384,1397, Nov 1988.

Assignment Graph with Minimum Cost Cut

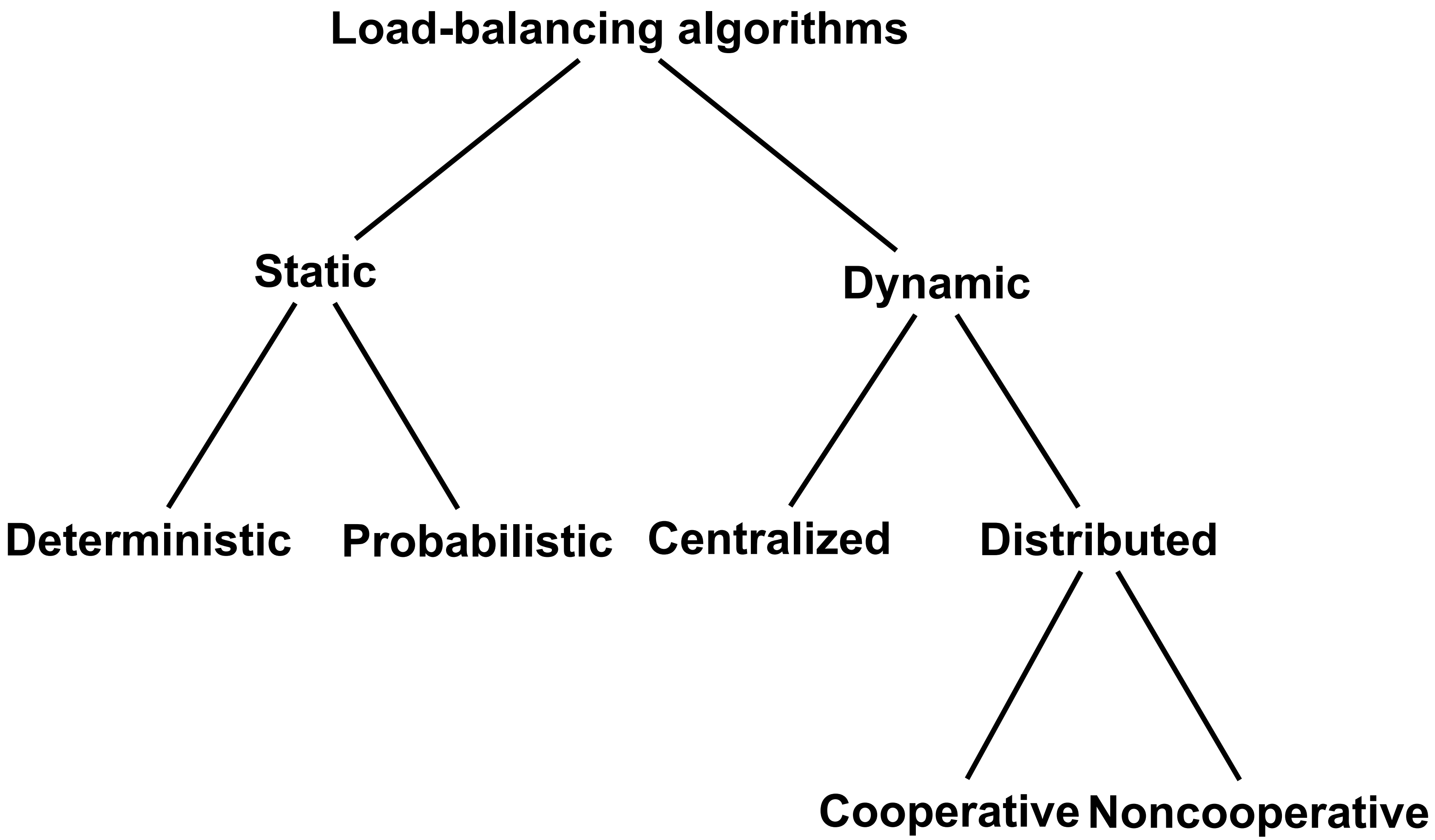


- An optimal solution can be found in polynomial time by utilizing Max-Flow/Min-Cut algorithm.
- However, the problem is known to be NP-hard for an arbitrary number of execution nodes.
    - Lo, V.M., "Heuristic algorithms for task assignment in distributed systems," *Computers, IEEE Transactions on* , vol.37, no.11, pp.1384,1397, Nov 1988.

Load-Balancing Approach

- Based on the intuition, it is desirable for the load in a distributed system to be balanced evenly and obtain a better resource utilization.
- A load-balancing algorithm tries to balance the total load by transparently transferring the workload from heavily loaded nodes to lightly loaded nodes.
- The basic goal is to maximize the total system throughput.

Taxonomy of Load-Balancing Algorithms



Static vs. Dynamic Load-Balancing Algorithms

- Static algorithms use only information for the average behavior of the system, ignoring the current state of the system.
- Static load-balancing algorithms are simpler.
- Dynamic algorithms responds to system state and are better able to avoid those states with unnecessarily poor performance.

Static Algorithms May Be Deterministic or Probabilistic

- Deterministic algorithms use the information about the properties of the nodes and the characteristics of the processes to be scheduled to allocate processes to nodes.
  - Task assignment algorithms belong to the category of deterministic static load-balancing algorithms.

- Probabilistic load-balancing algorithms use the information regarding statistic attributes of the system to formulate simple process placement rules.

- The deterministic approach is difficult to optimize and costs more to implement.
- The probabilistic approach is easier to implement but often suffers from having poor performance.

Dynamic Algorithms May Be Centralized or Distributed

- Centralized dynamic scheduling algorithm
  - Responsibility of scheduling physically resides on a single node.
  - All requests for process scheduling are handled by the *centralized server node*.
  - In the basic method, the other nodes periodically send status to centralized server to update the current information.
  - The problem is reliability.
- Distributed dynamic scheduling algorithm
  - The work involved in making process assignment decisions is physically distributed among the various nodes of the system.
  - It is composed of *k* physically distributed entities, also called *local controllers*.
  - Each controller runs asynchronously and concurrently with the others, and each is responsible for making scheduling decisions for the processes of a predetermined set of nodes.

Distributed Algorithms May Be Cooperative or Noncooperative

- In cooperative algorithms, each entity cooperates with each other to make scheduling decision.
- In noncooperative algorithms, individual entities act as autonomous entities and make scheduling decisions independently.
- The cooperative algorithms are more complex and involve greater overhead than noncooperative one.
- The stability of a cooperative algorithm is better than that of a noncooperative algorithm.

Issues on Designing Load-Balancing Algorithms

- Load estimation policies
  - Number of process
  - Service time:
    - Memoryless method
    - Past-repeats
    - Distribution method.

In modern computing, we will have foreground and background processes. These policies do not take into account background processes. In many cases, we need to sample the load for a longer time interval.

- Process Transfer policies: When to transfer the local process or accept the remote process.
  - Single threshold approach may cause process thrashing.
  - Double threshold approach accounts for the overhead that the load-balancing algorithm may incur in transferring and receiving a remote process.
    - High mark and low mark.
    - system load > High mark: both local and remote processes are rejected.
    - low mark < system load < high mark: local processes are accepted and remote processes are rejected.
    - system load < low mark: both local and remote processes are accepted.

Location Policies

- Threshold: based on the workload state to select the node to transfer.
  - To avoid the overhead, a limit on node probing can be set to stop the process.
  - 3 or 5 is almost as good as the performance with a large probe limit (e.g., 20).

- Shortest:
  - Distinct nodes are chosen at random, and each is polled in turn to determine its load.
  - The process is transferred to the node with minimum load unless the node is in the prohibiting state.
  - However, the performance is not significant better than the threshold approach.

Location Policies

- Bidding:
  - The system is tuned into a distributed computational economy with buyers and sellers of services.
  - Managers are the nodes which need to execute their processes on the other nodes.
  - Contractors are the nodes which could execute the processes from the other nodes.
  - The bidding approach is totally autonomy.
  - Drawbacks:
    - communication overhead
    - how to determine the pricing policy.

- Pairing:
  - Two nodes that differ greatly in load are paired temporarily.
  - Approaches:
    - Randomly selects the partner.
    - Comparing the time to complete on the partner node and local node.
    - Expected time on partner node / that on local node.

State Information Exchange Policies

- Periodic broadcast
  - fruitless message.
  - heavy network traffic and scalability
- Broadcast when state changes
  - avoid the problem of fruitless messages.
  - Only broadcast the message when the system is either under-loaded or over-loaded
- On-demand exchange
  - When a node is either under-loaded or over-loaded, it broadcasts the requests.
  - An improvement:
    - When the node is under-loaded, only overloaded nodes reply the messages.
    - When the node is over-loaded, only under-loaded notes reply the messages
- Exchange by polling
  - Broadcasting is poor in scalability.
  - Randomly selected the nodes to be polled.
  - The state information only exchanged between the polling nodes and polled nodes

Priority Assignment and Migration limitation

- Priority Assignment Policy:
  - Selfish: local processes have higher priority.
  - Altruistic (unselfish): remote processes have higher priority.
  - Intermediate: if the number of local processes is greater than that of the remote processes, the local processes are assigned higher priority.
  - Performance effect:
    - Selfish yields the worst response time of the three policies. Because the extremely poor performance of remote processes.
    - Altruistic priority achieves the best response time performance.
    - Intermediate policy has worse response time than altruistic approach.
- Migration-Limiting Policies:
  - Uncontrolled
  - Controlled:
    - migration count is used to limit the number of migration

Load-Sharing Approach

- Load balancing in the strictest sense is not achievable.
- For the proper utilization, it is not required to balance the load on all the nodes.
- The *dynamic load-sharing algorithms* prevent the nodes from being idle while some other nodes have more than two processes.

Issues in Designing Load-Sharing Algorithms

- Load estimation policies
- Process transfer policies:
  - All-or-nothing strategy
- Location policies
  - Sender-initiated location policy
  - Receiver-initiated location policy
  - Performance comparison:
    - At light to moderate system loads: sender-initiated is preferable to receiver-initiated
    - At high system loads: receiver-initiated policies are preferable.
- State information exchange policies
  - Broadcast when state changes
  - Poll when state changes

Discussion - Resource Management for Mobile Cloud Systems

- Mobile Cloud Computing (MCC) refers to the computing system consisting of mobile devices and cloud servers.
- The goals of MMC include
  - Reduce resource consumption on mobile devices, including energy, computation, and storage capacity, so as to prolong the lifetime of mobile devices and shorten the response time.
  - Enhance the performance such as QoS so as to improve the user experience on mobile devices.

Challenges of Resource Management on MMC

- Communication Overhead in terms of delay
  - Most of mobile devices communicate with cloud servers via telecommunication network or Wi-Fi network.
  - The communication links are unreliable and have limited bandwidth to migrate workload.
    - Preload the application on cloud server
    - Compose the services dynamically
- Energy consumption on communication is ***not*** negligible.
  - Energy consumed by CPU is relatively small for light workload.
  - Only heavy workloads are suitable for offloading/migration. (How do we know in advance?)

Challenges of Resource Management on MCC (cont.)

- Heterogeneous runtime environment:
  - Mobile devices are designed with low power processors:
    - GPU on mobile devices are 10 to 100 times slower, compared to GPU on desktop devices.
  - Applications designed for mobile devices are limited to use less powerful CPU/GPU.
    - Using lower accurate algorithm to obtain the results in short time.
  - Migration applications designed for mobile devices to cloud server:
    - Duplicating or emulating mobile runtime on cloud servers.
    - Migrating mobile applications to cloud servers.
    - Minimize the change on mobile application and increase the applicability of the approach.

Additional Observations

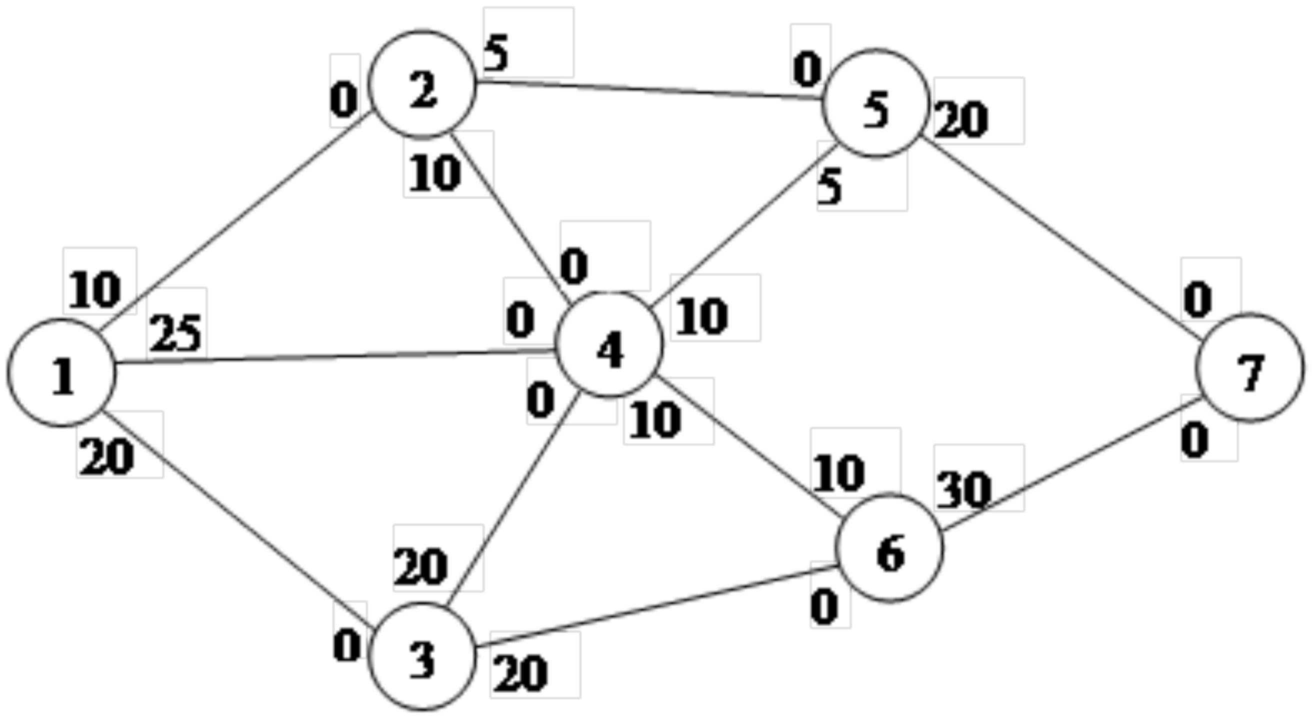
- Applications on mobile devices demand timely response, i.e., soft real-time or firm real-time.
- Cloud servers have much more resources, including computation and storage.
- Executing mobile applications on cloud servers is NOT the best execution model to ***provide the computation service.***
  - ***Remote execution*** leads to longer response time.
  - ***Emulating the execution* leads to resource misuse.**
- Can we manage the resource better?

Summary

- A resource manager schedules the processes to free resources in a distributed system that can optimize
  - resource usage
  - response time
  - network congestion
  - scheduling overhead
- The task assignment approach
  - schedules processes to minimize the inter-process communication and improve the turnaround time.
- The load-balancing approach
  - attempts to equalize the average workload on all the nodes.
- The load-sharing approach
  - attempts to keep all nodes busy.

Max-Flow Min-Cut Theorem

- Find a max-flow from source node to destination node on a given network graph

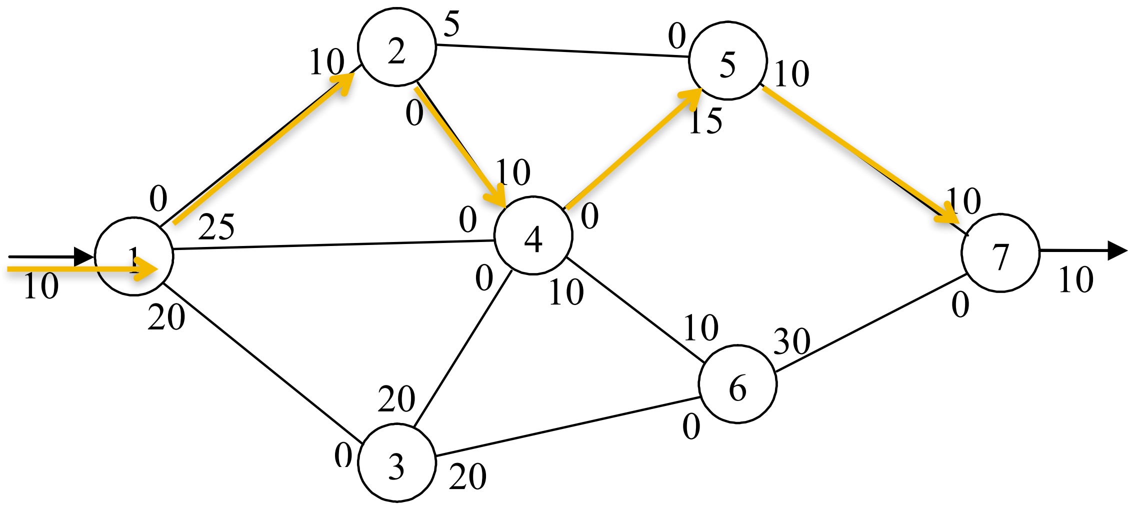


Ford and Fulkerson Algorithm

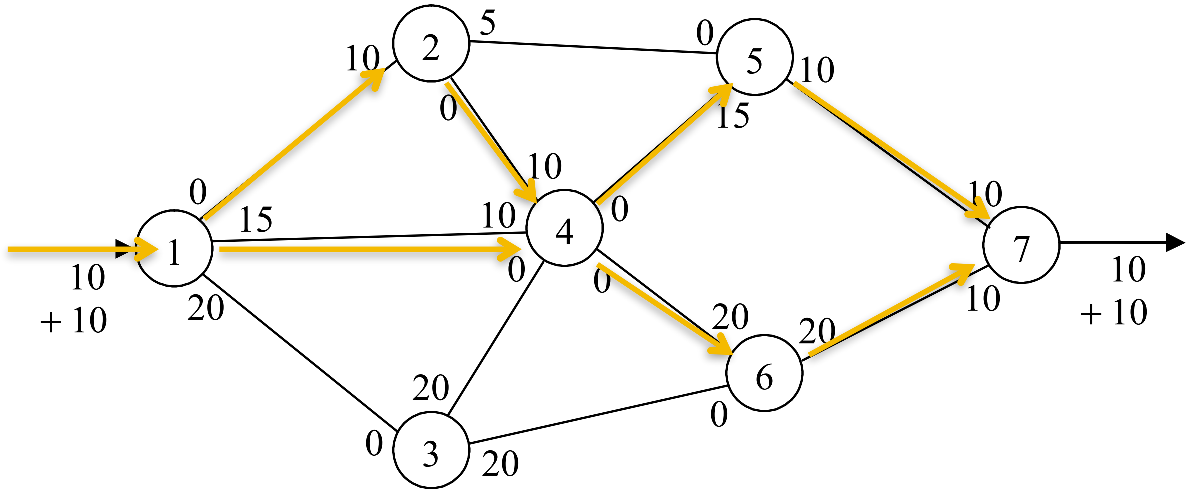
- Find any path from the origin node to the destination node that has a strictly positive flow capacity remaining. If there are no more such paths, exit.
- Determine f, the maximum flow along this path, which will be equal to the smallest flow capacity on any arc in the path (the bottleneck arc).
- Subtract f from the remaining flow capacity in the forward direction for each arc in the path. Add f to the remaining flow capacity in the backwards direction for each arc in the path.
- Go to Step 1.

Example of Ford and Fulkerson Algorithm

- f=10

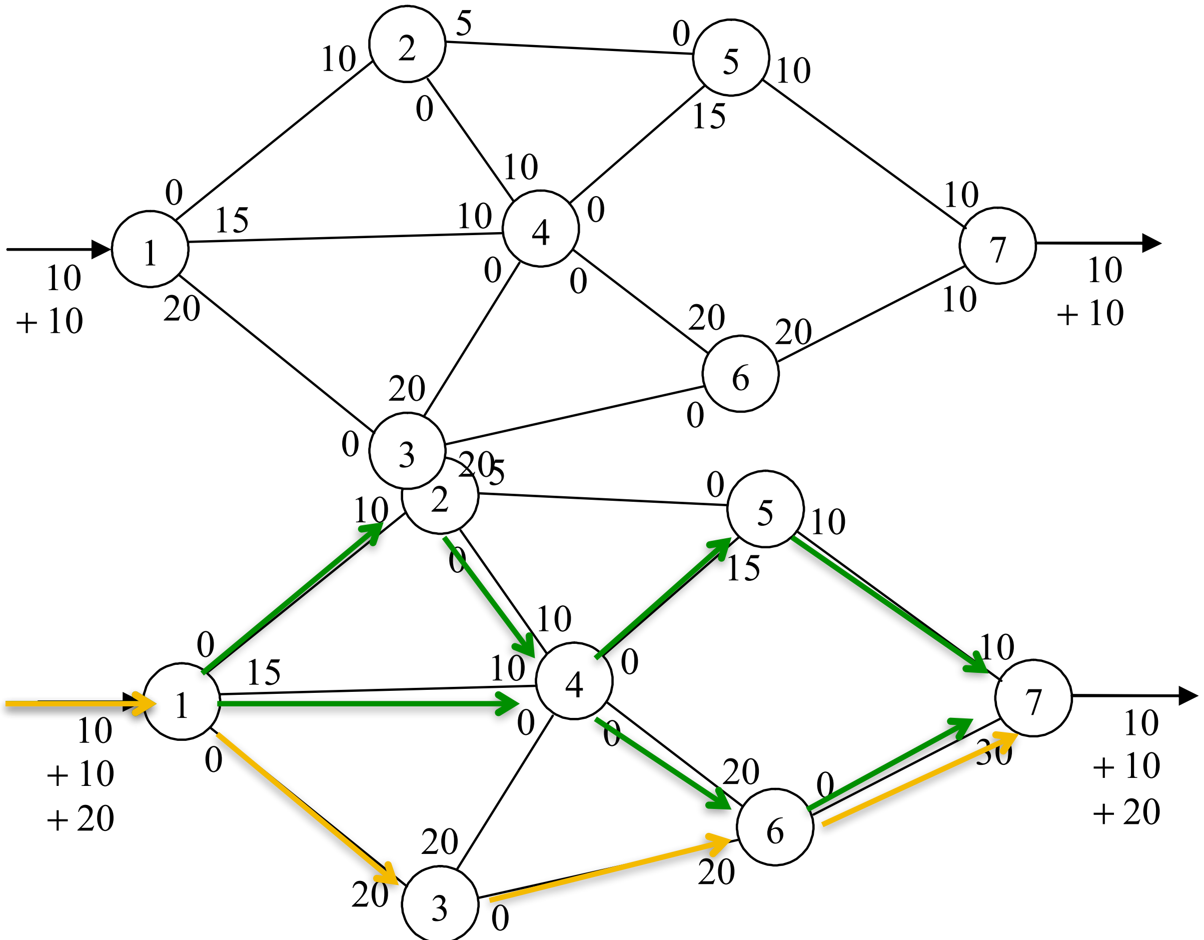


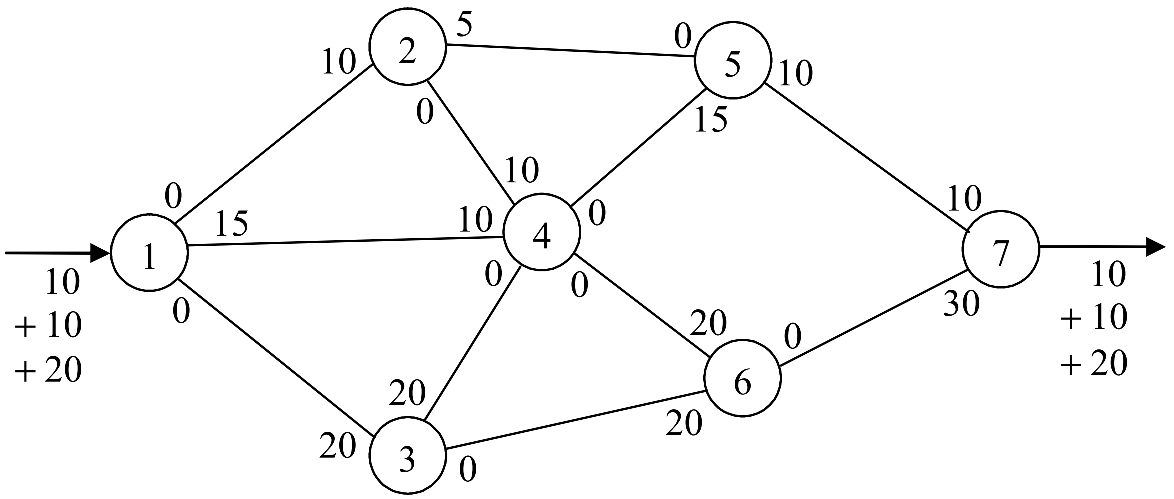
- F=10+10



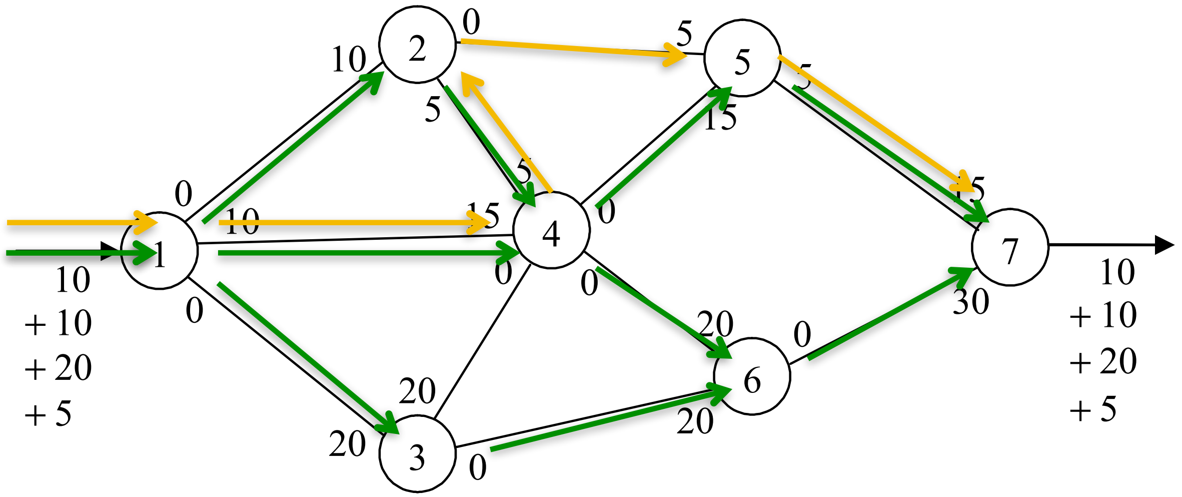
1
2
3
4
5
6
7
10
+ 10
0
15
20
10
5
0
0
10
10
0
0
0
10
15
0
20
20
0
20
10
10
20
10
+ 10

- f=20+20

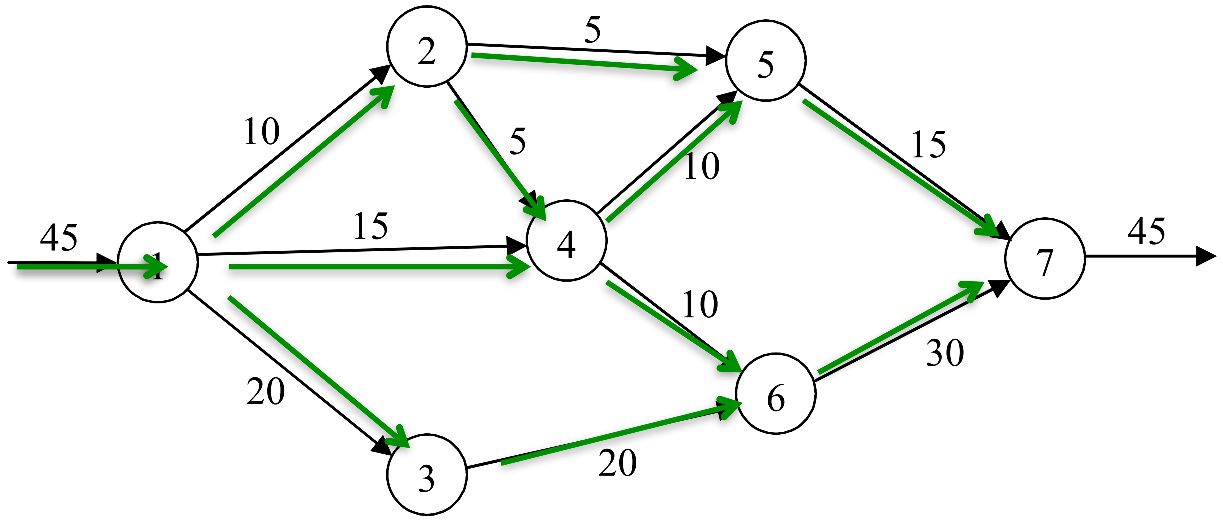




1
2
3
4
5
6
7
10
+ 10
+ 20



1
2
3
4
5
6
7
10
+ 10
+ 20
+ 5


45
1
10
15
20
2
5
5
3
20
4
10
10
5
6
15
30
7
45