Ch 05 -A system Performance Model

Ch.5 Distributed Process Scheduling [Randy Chow, 97]

Introduction

- Processes (jobs) need to be scheduled before execution.
  - Enhance overall system performance
    - Process completion time
    - Processor utilization
  - Performed locally.
  - Performed globally.
  - Processes
    - executed on remote nodes
    - migrate from one node to node

Introduction (contd.)

- Why scheduling is complex?
  - communication overhead can not be ignored.
  - effect of the system architecture can not be ignored.
  - dynamic behavior of the system must be addressed.
- Chapter presents **a model** for capturing the effect of **communication** and **system architecture** on scheduling.

Section I (Theory)

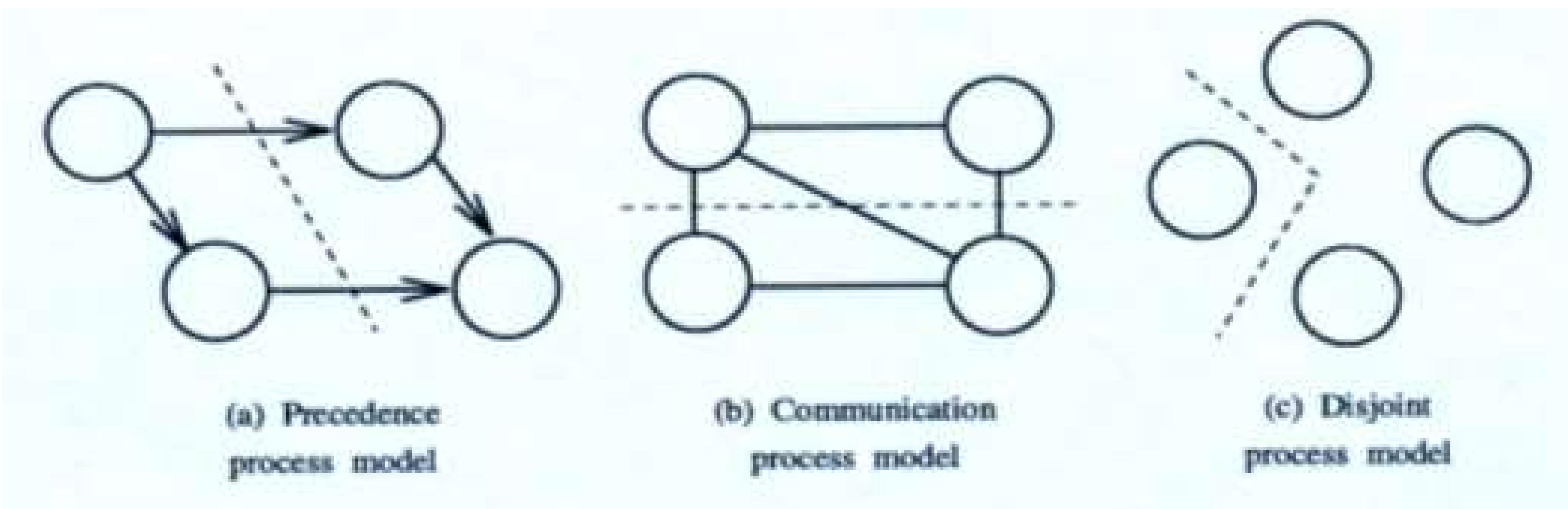
5.1 A system Performance Model

Outline

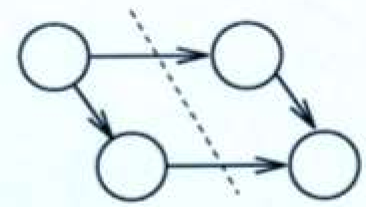
- Process interaction: Example
  - Precedence Process Model
  - Communication Process Model
  - Disjoint Process Model
- Speedup
- Refined Speedup
- Efficiency Loss
- Workload Sharing
- Queuing Models

Process interaction: Example

- We used graph models to describe process communication.
- Eg: A program computation consisting of 4 processes mapped to a two-processor multiple computer system.
  - Process interaction is expressed differently in each of the three models.



Precedence Process Model

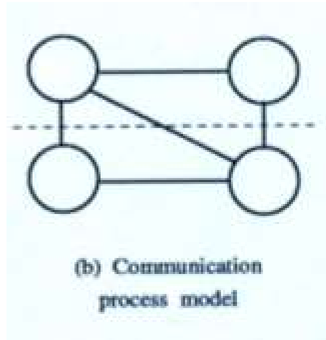


(a) Precedence process model

- The directed edges denote precedence relationship between processes.
- may occur communication overhead
  - If processes connected by an edge are mapped to different processors.
- model is best applied to the concurrent processes generated by concurrent language construct such as *cobegin/coend* or fork/join.

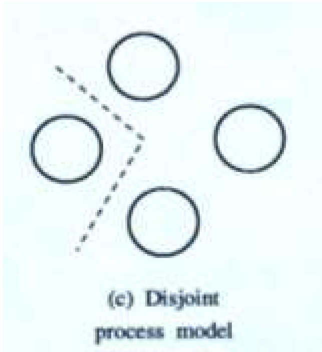
- Scheduling is to minimize the total completion time of the task, includes both computation and communication times.

Communication Process Model



- processes created to coexist and communicate asynchronously.
- edges represent the need for communication.
- the time is not definite, thus the goal of the scheduling may be
  - Optimize the total cost of communication
  - and computation.
- Task is partitioned in such a way that
  - minimizes the inter-processor communication
  - and computation costs of processes on processors

Disjoint Process Model



- process interaction is implicit
- assume that
  - processes can run independently
  - completed in finite time
- Processes are mapped to the processors
  - to maximize the utilization of the processors
  - And minimize the turnaround time of processors. (turnaround time is defined as the sum of services and times due to waiting time of the other processes.)

Speedup

- Speedup Factor is a function of
  - Parallel Algorithm
  - System Architecture
  - Schedule of execution

$$S = F(Algorithm, System, Schedule)$$

- S can be written as:

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} \times \frac{OCPT_{ideal}}{CPT} = S_i \times S_d$$

  - *OSPT*= optimal sequential processing time;
  - *CPT*= concurrent processing time; concurrent algorithm + specific scheduling method
  - $OCPT_{ideal}$ = optimal concurrent processing time on an ideal system; no inter-processor communication overhead + optimal scheduling algorithm.
  - S_i = ideal speedup obtained by using a multiple processor system over the best sequential time
  - S_d = the degradation of the system due to actual implementation compared to an ideal system

Refined Speedup

- To distinguished the role of algorithm, system, and scheduling for speedup is further refined. S_i can be written as:

$$S_i = \frac{RC}{RP} \times n$$

$$RP = \frac{\sum_{i=1}^{m} P_i}{OSPT}$$

$$RC = \frac{\sum_{i=1}^{m} P_i}{OCPT_{ideal} \times n}$$

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} \times \frac{OCPT_{ideal}}{CPT} = S_i \times S_d$$

  - n - # processors
  - m - # tasks in the concurrent algorithm
  - $\sum_{i=1}^{m} P_i$ total computation of the concurrent algorithm > *OSPT*
  - RP (Relative Processing) - it shows how much loss of speedup is due to the substitution of the best sequential algorithm by an algorithm better adapted for concurrent implementation.
  - RP (Relative Concurrency) – measures how for from optimal the usage of the n processor is.
  - RC=1 → best use of the processors
  - A good concurrent algorithm minimize RP and maximize RC

Refined Speedup (contd.)

- S_d can be written as:

$$S = \frac{OSPT}{CPT} = \frac{OSPT}{OCPT_{ideal}} \times \frac{OCPT_{ideal}}{CPT} = S_i \times S_d$$

$$S_d = \frac{1}{1+\rho}$$

$$\rho = \frac{CPT - OCPT_{ideal}}{OCPT_{ideal}}$$

$$S = \frac{RC}{RP} \times \frac{1}{1+\rho} \times n$$

- ρ - efficiency loss (loss of parallelism when implemented on real machine)
  - function of scheduling + system architecture.
- ρ decompose into two independent terms

$\rho = \rho_{sched} + \rho_{syst}$(not easy scheduling & system are intertwined)

- communication overhead can be hidden
- A good schedule hides the communication overhead as much as possible.

Efficiency Loss

- interdependency between scheduling and system factors.
  - X – multiple computer under investigation
  - Y' – scheduling policy that extended for system X from a scheduling policy Y on the corresponding ideal system.
  - ρ can be expressed as:

$$\rho = \frac{CPT(X, Y') - OCPT_{ideal}}{OCPT_{ideal}}$$

$$¿\frac{CPT(X, Y') - CPT_{ideal}(Y)}{OCPT_{ideal}} + \frac{CPT_{ideal}(Y) - OCPT_{ideal}}{OCPT_{ideal}}$$

$$¿\rho_{syst} + \rho'_{sched}$$

Efficiency Loss (contd.)

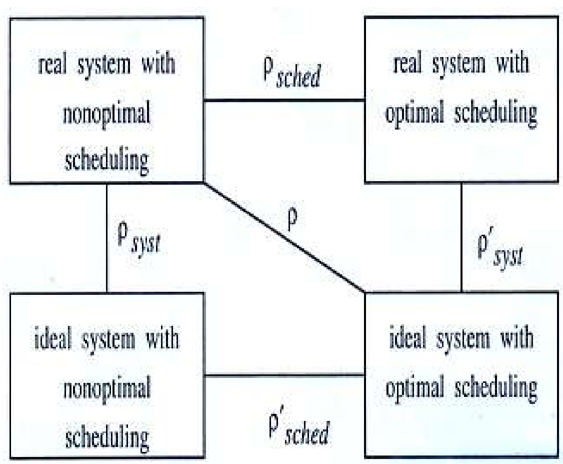
- Similarly, for non-ideal system

$$\rho = \frac{CPT(X,Z) - OCPT_{ideal}}{OCPT_{ideal}}$$

$$\text{¿}\frac{CPT(X,Z) - OCPT(X)}{OCPT_{ideal}} + \frac{OCPT(X) - OCPT_{ideal}}{OCPT_{ideal}}$$

$$\text{¿}\rho_{sched} + \rho'_{syst}$$

- following figure shows decomposition of ρ due to scheduling and system communication.



- Impact of communication on system performance must be carefully addressed in the design of distributed scheduling algorithm

Workload Sharing

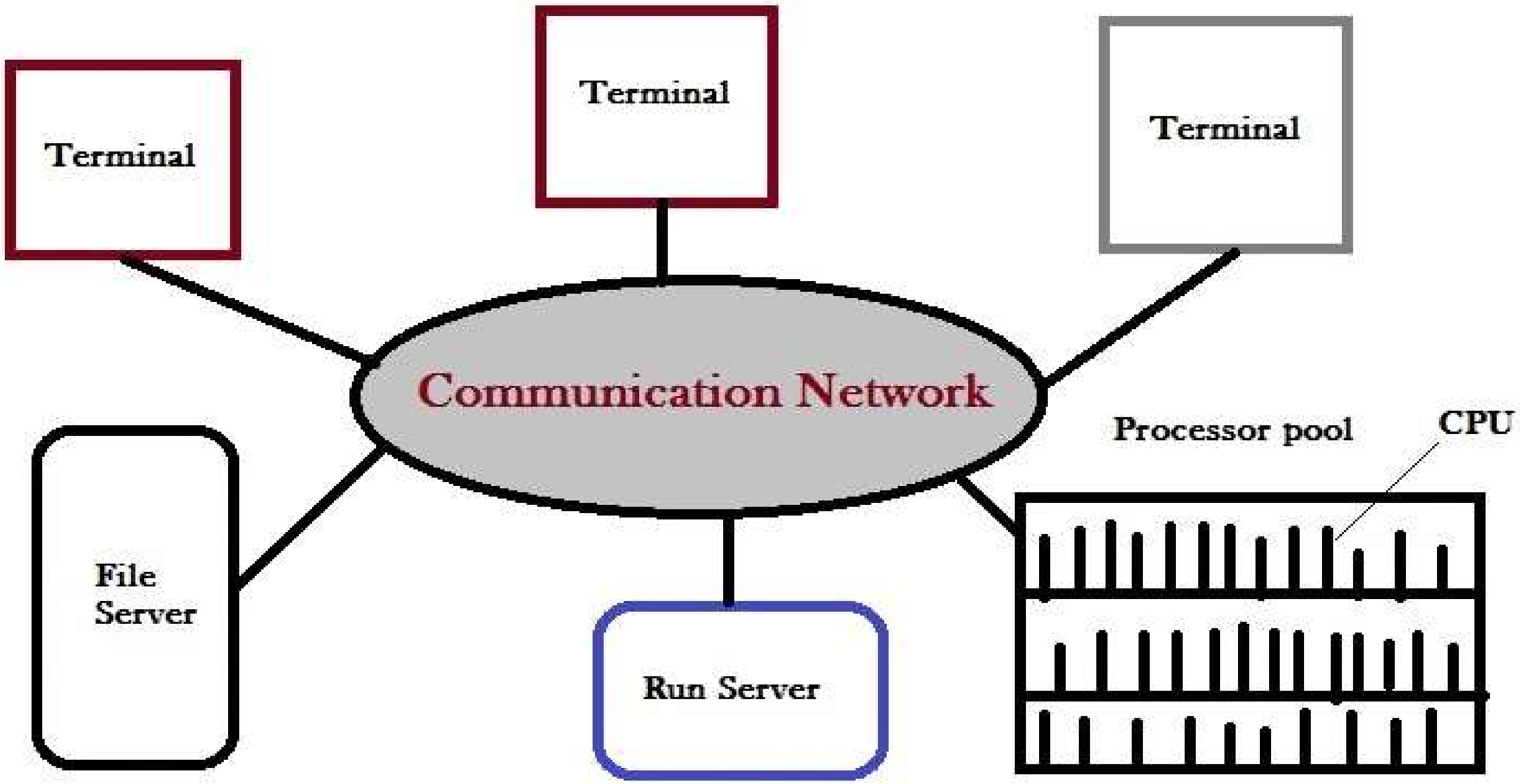
- If processes are not constrained by precedence relations and are free to move around.
- Performance can be further improved by sharing workload.
- processes can be moved from heavily loaded nodes to idle nodes.
- Load sharing – static workload distribution
- Load balancing – dynamic workload distribution
- Benefits of workload distribution
  - Increase processor utilization
  - Improved turnaround time for processors.
- Migration of processes reduce queuing time – cost additional communication overhead.

Load Sharing

- Static workload distributions
- Dispatch process to idle processors statically upon arrival.
- Corresponding to processor pool model.

Processor pool model

A system based on processor pool model of distributed system



Processor pool model

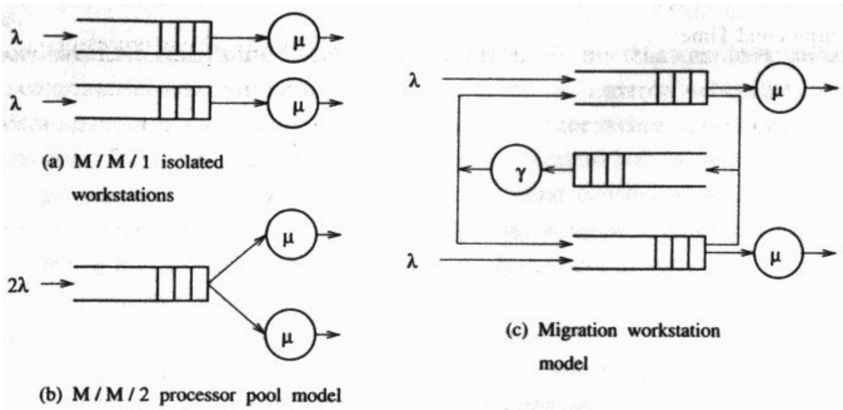
- Consists of multiple **processors** and group of workstations.
- The **model** is based on the observation that most of the time a user does not need any computing power.
- In this **model**, the process is pooled together to be shared by the users as needed.

- The processor pool of process consists of large microcomputers and minicomputers attached to the network.
- Each processor has its own memory to load and run.
- The processors in the pool have no terminals attached directly to them, and the user accesses the system from terminals that are attached to the network via a special device.

Load Balancing

- Dynamic workload distribution
- Migrate processes dynamically from heavily loaded processor to lightly loaded processor
- Corresponding to migration workstation model.

Queuing Models



(a) M / M / 1 isolated workstations

(b) M / M / 2 processor pool model

(c) Migration workstation model

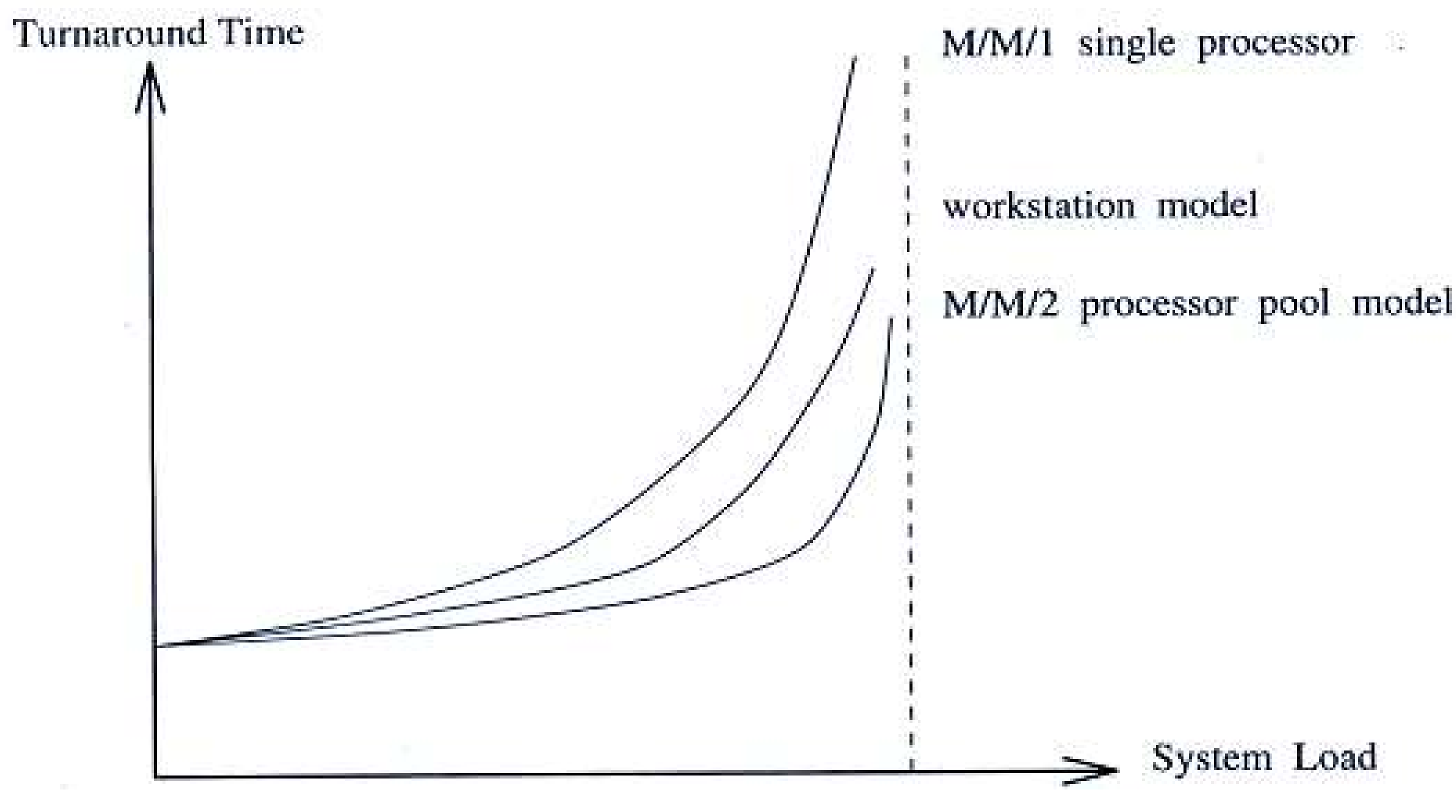
$$TT_1 = \frac{1}{\mu - \lambda}$$

$$TT_2 = \frac{\mu}{(\mu + \lambda)(\mu - \lambda)}$$

- *TTi* – average turnaround time
- λ - arrival rate
- μ - service rate

Queuing Models (contd.)

- Comparison of performance for workload sharing



Introduction to Load Balancing:

- Definition of Distributed systems.
  - Collection of independent loosely coupled computing resources.
- Load Balancing is a “pre-step” to scheduling.

Motivations:

Random Arrival of user tasks.

Varied Computing resources of different hosts.

Homogeneous Vs Heterogeneous systems.

Basic Issues:

Definition of LOAD and Performance as the basics of Load Balancing.

Criterion for Load Balancing:

- Process based factors.
- Resource based factors.
- Algorithmic factors.

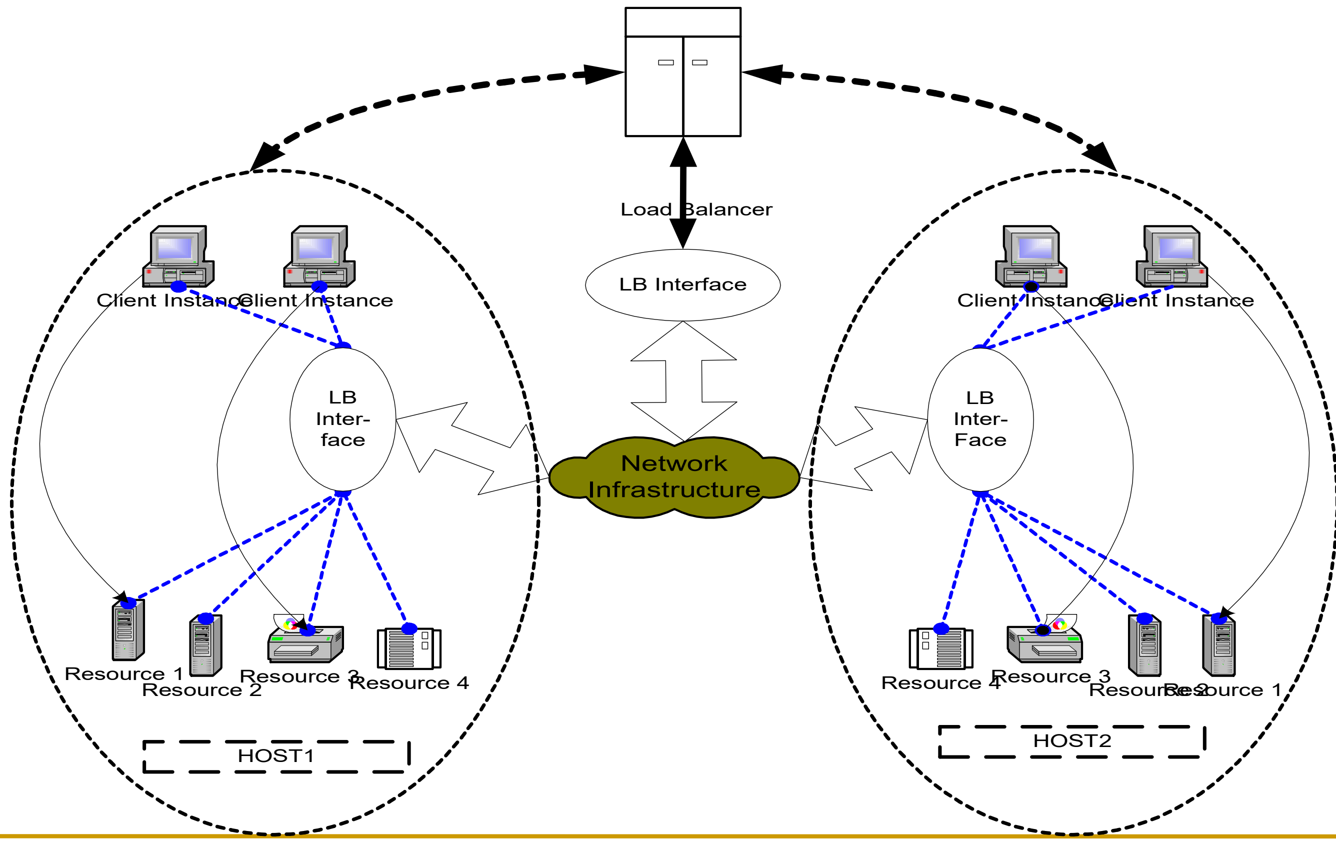
Efficient Evaluation of these criterion.

Load Balancing Vs Sharing:

- Both attempt to maximize response time.
- Balancing implies that load has to equalized rather than just shared.
- Hence Load Balancing is a special Case of Load Distribution policy.
- Load Balancing has more over head.

Load Balancing Architectures:

Centralized Load Balancer:

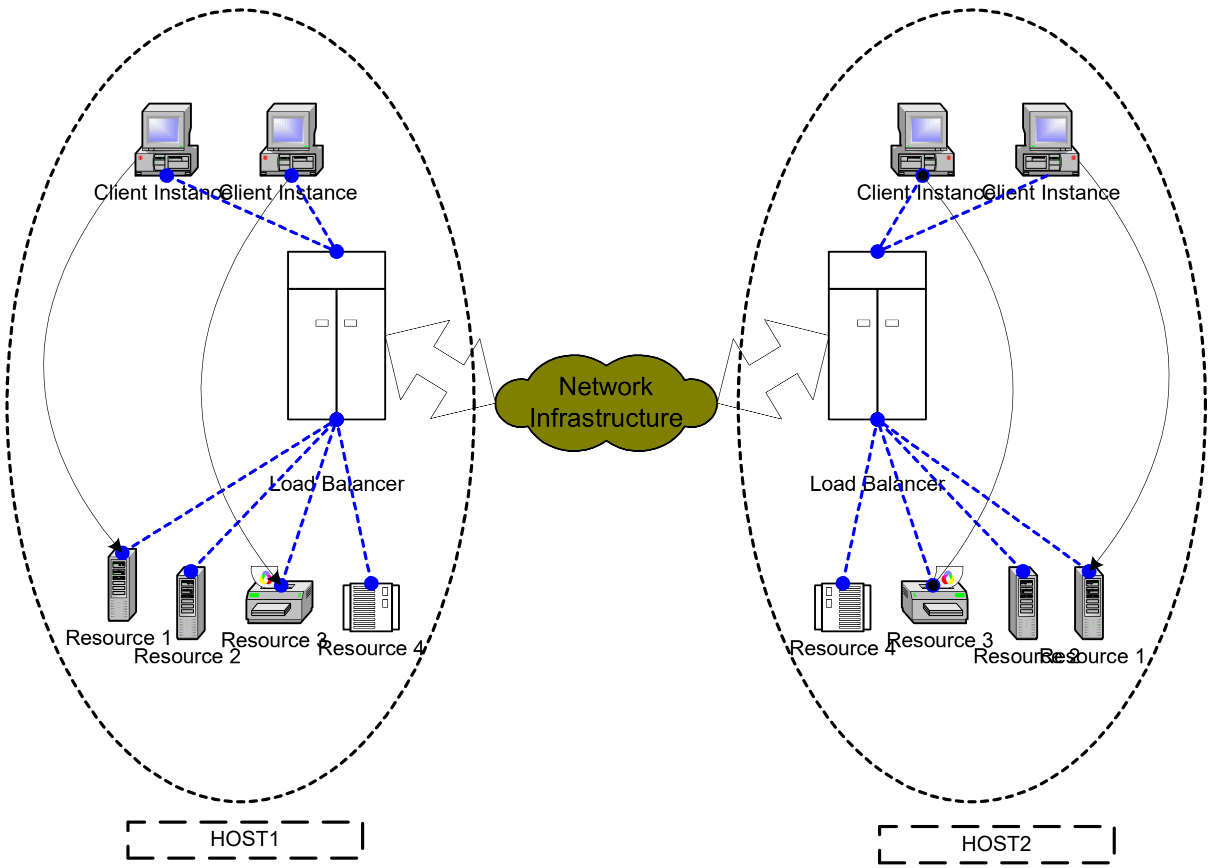


Centralized Load Balancer:

- Advantages:
  - Single Host Implementation.
  - Highly adaptive.
  - No Overhead on individual hosts.
  - Better State Consistency.
  - Centralized transfer of tasks.
- Disadvantages:
  - More expensive.
  - Single point of failure.
  - Relatively lesser scalable.

Load Balancing Architectures:

Peer to Peer Architecture:



De-Centralized Load balancer:

- Advantages:
  - No Single point of failure.
  - Independent unit of operation for each host.
  - Highly scalable.
- Disadvantages:
  - Complex implementation.
  - Overhead on each host due to load of the algorithm.
  - Lesser level of adaptability to heterogeneous environments.

Peripheral Components:

- Client Programs:
  - No knowledge of location of execution i.e. “Location Transparency”.
  - Priority, type of process may influence Load Balancing decisions.
- Resources:
  - Decide the granularity of Load Balancer.
  - Different types of resources.

Peripheral Components:

- Status of resources & collection mechanisms.
  - Broadcast.(periodic, demand driven, & state change)
  - Kernel based monitor. (periodic, demand driven, & state change)
  - Publish - Subscribe Model.

Peripheral Components:

- Communication network:
  - Speed, reliability & adaptability issues.
  - Hetrogeneous requirements for networks.

Load balancing Details:

- Load Balancing policies:
  - Static
  - Dynamic
    - Adaptive(Learning)
    - Non-Adaptive.

Load Balancing details:

- Load & Performance Metrics:
  - Load : Index of CPU Queues, CPU utilization etc.
  - Performance: Measured as an index of average response time for client processes.

Load balancing Details:

- Type of task transfers:
  - Preemptive:
    - Process Migration required.
    - Generally more over head involved as the entire process state is transferred.
  - Non-Preemptive:
    - Based on initial task placement.
    - Simpler & more efficient.

Load Balancing Details:

- Composition of an Load Balancing Algorithm:
  - Transfer policy: Determines the state of a node I.e sender or receiver.
  - Selection policy: Determines “which” task would be transferred.
  - Location policy: “where” to transfer.
  - Information policy: State maintanence.

Load Balancing Details:

- Stability of a Load Balancing Module:
  - Algorithmic stability.
  - System stability.
  - In effective Vs effective algorithms.
  - Stability & Effectiveness of a Load Balancing Algorithm.

Conclusion:

- Load balancing forms an important strategy for the improvement of the average response time for any user process.
- Internet as a major boost for its application.
- Load balancing widely used as a major component of clustered solutions.
- Advances in technologies relating to peripheral components leads to more focus on efficient implementation of the load Balancing Algorithm.

Section II

Recent Work

Recent Work

- Distributed Measurements for Estimating and Updating Cellular System Performance [Liang Xiao et al, 2008]
  - Discuss the number and placement of sensors in a given cell for estimating its signal coverage.
  - Traditional measurements : signal-to-noise ratio, signal-to-interference ratio, outage probability
  - New performance model :
    - improve measurement efficiency
    - minimizing the required number of measurement sensors
- Performance Prediction of Component- and Pattern-based Middleware for Distributed Systems [Shruti Gorappa, 2007]
  - Design patterns, components, and frameworks are used to build various distributed real-time, and embedded (DRE) systems. ex: high-performance servers, telecommunication systems, and control systems.
  - way to quantify the performance of
    - components
    - design patterns

Section III

Future Work

Future Work

- Develop the comparative performance models for different architectures and validating them experimentally
  - Eg: Acceptor/Connector, Leader/Follower, Threadpool and publish/subscribe.
  - Proxy, Pipes/Filters, and Proactor.
- Develop performance model for large–scale, geographically distributed grid systems
  - Have to capture resource heterogeneity, infrastructure characteristics.

References

[1] Randy Chow & Theodore Johnson, 1997,“*Distributed Operating Systems & Algorithms*”, (Addison-Wesley), p. 149 to 156.

[2] Shruti Gorappa,”Performance Prediction of Component- and Pattern-based Middleware for Distributed Systems”, MDS’07

[3] Liang Xiao et al, “Distributed Measurements for Estimating and Updating Cellular System Performance”, 2008 IEEE

[4] Paolo Cremonesi et al,“Performance models for hierarchical grid architectures”, 2006 IEEE