Load Distribution

Definition: Load Distribution

Transfer load (tasks) from heavily loaded servers to idle or lightly loaded ones

Definition: Load sharing

Reduce the likelihood of an unshared state (e.g. idle servers)

Definition: Load balancing

Go a step further than load sharing by attempting to equalise loads at all servers

1 Where does the workload come from

1. Queue length of waiting tasks: proportional to response time
2. CPU Utilisation
3. Network bandwidth utilisation

Load information collection:

- **Central coordinator**: collects server load information centrally and globally
- **Local approach**: a server locally collects load information of neighbouring servers

2 Simple task transfer

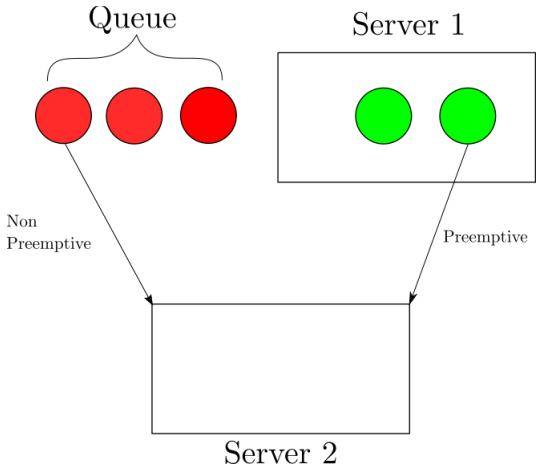
Definition: Non-preemptive task transfers

Transfer tasks that have not started executing

- Transfer only the request(or task) without processing states
- Good for load sharing but difficult for load balancing

Definition: Preemptive task transfers

Transfer tasks that are partially executed - this is expensive as it involves collection and transmission of task states



3 Routing mechanisms based on network network packet content

Layer-4 routing

- Determines the target server without referring the message/request content
- **Content-blind routing**: server selection is purely based on information from the IP header

Layer 7 Routing:

- Examine a request at the application level and select a server accordingly
- Can support sophisticated dispatching policies, but may induce significant latency (also called content aware routing)

3.1 Implementations

Definition: Two-way

Response from FE

Definition: One-way

Response from the server directly

Two way is less scalable because of the overhead in the front end

Layer 4 vs Layer 7 routing:

- Layer 4 routing is efficient and requires only simple FE servers
- Layer 7 routing is less scalable
- Layer 7 routing benefits from server specialisation and content partition

Layer	Approach	Routing	Data Flow	Pros	Cons
Layer 4	Two way	Content blind	inbound/ outbound	flexible, portable	switching throughout, TCP connection grain control
Layer 4	One way	Content blind	inbound	simple, fast routing	network topology, TCP connection grain control
Layer 7	Two way	Content aware	inbound/ outbound	simple, caching, server specialisation, content partition, SSL session reuse	switching bottleneck, slowest routing
Layer 7	One way	Content aware	inbound	server specialisation, content partition, SSL session reuse	Switching bottleneck, complex routing

4 Approaches for load distribution

Static:

- Decisions are hard-coded into an algorithm
- Simple in implementation
- A prior knowledge of system is required

Dynamic:

- Make decision during runtime based on system states
- Correctness of load distribution depends on the timeliness of parameters collected

Adaptive:

- Enhance the dynamic approach by allowing making choices of decision algorithms and the frequency of collecting load information based on system states

5 Constructing a load distribution algorithm

Four main components:

- Transfer
- Selection
- Location
- Information policies

These define what information is required to collect and maintain supporting decision making, and what procedures are required to follow for distributing workload

5.1 Transfer policy

Decide whether a server needs to transfer tasks

- Thresholds: number of tasks, processor utilisation
- Role: a server becomes sender/receiver when its load over/under a threshold
- Issue: sensitive to time duration of a task transfer

5.2 Selection policy

Determines which task(s) to transfer

- Tasks cause a server overload
- Estimated task execution time
- Or serve response time improvement
- Minimise location-dependent system calls made by the selected task

5.3 Location policy

Decide the receiving server for a task

- Polling is generally used
- Can be done serially or in parallel(using multicast)

5.4 Information policy

Decide when, where and what information to collect

- Demand-driven: a server collects the state of other servers only when it becomes either a se4nder or receiver
- Periodic: servers exchange load information periodically
- State-change-driven: servers disseminate state information whenever their state changes by a certain degree

6 Load distributing algorithms

Definition: Sender-initiated

Distribution initiated by an overloaded server

Definition: Receiver-initiated

Distribution initiated by lightly loaded servers

Definition: Symmetric

Initiated by both senders and receivers. Has advantages and disadvantages of both the approaches

Definition: Adaptive

Sensitive to the state of the system

6.1 Sender and Receiver Initiated

	Sender Initiated	Receiver Initiated
Transfer Policy	Use thresholds • **Sender** if queue length exceeds threshold T • **Receiver** if accepting a task will not make queue length exceed T	Use thresholds • **Receiver** if queue length below T • **Sender** if queue length above T
Selection policy	Only newly arrived tasks	Only newly arrived tasks
Location policy	Identify receiver • Random: Task transferred to server at random – No need for state collection – Unnecessary task transfers may occur • Threshold: Poll a server to find out if it is a receiver, Receiver must accept the task irrespective of when the task actually arrives • Shortest: Poll servers. Select the receiver with shortest task queue length	Polling: • Poll a random server, transferring a task transfer when queue length > threshold • Repeat the process to attempt transferring tasks from another server until PollLimit is reached
Information Policy	demand-driven	demand-driven
Stability	Can become unstable at high loads • At high loads, it may become difficult for senders to find receivers • Also, the number of senders increase at high system loads thereby increasing the polling activity • Polling activity may make the system unstable at high loads	”Not unstable” since there are lightly loaded systems that have initiated the algorithm

6.2 Symmetric

- Senders search for receivers and vice-versa
- Low loads: senders can find receivers easily
  High loads: receivers can find senders easily
- May have the disadvantages of both
  - Polling at high loads can make the system unstable
  - Receiver-initiated task transfers can be preemptive and so expensive

6.3 Adaptive algorithm

Main idea

- Aim: Limit sender's polling actions at high load to avoid instability
- Classify servers based on the collected state information and poll adaptively

The process

- Each server maintains three lists. All servers are assumed to be receivers initially
- Location policy at sender:
  - Sender polls the head of the receiver list
  - Polled server puts the sender at the head of its sender list, and informs the sender whether it is a receiver, a sender, or an OK server
  - If the polled server is still a receiver, the new task is transferred
  - Else sender updates lists and polls the next potential receiver
  - If this polling process fails to identify a receiver, the task can still be transferred during a receiver-initiated dialogue
- Location policy at receiver:
  - Receivers obtain tasks from potential senders. Lists are scanned in the following order
    * Head to tail in senders list (most up to date info used), tail to head in OK list (lead up to date used), tail to head in receiver list
    * Least up to date used in the hope that status might have changed
  - Transfer a task if a sender is found
  - If the server is not a sender, both the polled server and receiver update each other's status
  - Polling process stops if a sender is found or a static PollLimit is reached

At high loads, sender initiated polling gradually reduces as servers get removed from receiver list(and become senders) whereas at low loads, sender will generally find some receiver

At high loads, receiver initiated works and can find a sender whereas at low loads receiver may not find senders, but that does not affect the performance

Algorithm dynamically becomes sender-initiated at low loads and receiver-initiated at high loads

Hence, algorithm is stable and can use non-preemptive transfers at low loads (sender initiated)

7 Selecting an algorithm

- If a system never gets highly loaded, sender initiated algorithms work better
- Receiver-initiated algorithms are better for high loads
- Widely fluctuating loads: symmetric algorithms
- Widely fluctuating loads and high migration cost for preemptive transfers: sender-initiated algorithms
- Heterogeneous work arrival: adaptive algorithms