Load Balancing in Distributed System

Prof. Ananthanarayana V.S.
Dept. Of Information Technology
N.I.T.K., Surathkal
Objectives of This Module

• Show the differences between the terms – CPU scheduling, Job scheduling, Load balancing
• Requirements of Load balancing algorithms
• Basic load balancing algorithms
• Important parameters while designing load balancing algorithms
• Challenges in load balancing algorithms
Road Map

• CPU Scheduling Vs. Load Scheduling
• Load Balancing
• Scheduling Policies
• Load Balancing Components
• Load Balancing Strategies
CPU Scheduling - Conventional

- **Issue:** In multiprogramming environment (with single CPU), which job is scheduled to the processor NEXT?
- **Need:** To allocate the job for execution

DIFFERENT SCHEDULING TECHNIQUES:

1. FIRST COME FIRST SERVE
2. PRIORITY BASED
3. ROUND ROBIN BASED
4. MULTI LEVEL FEEDBACK QUEUES
5. ETC.
Load (Job) Scheduling

• **Issue:** In distributed environment, which job is scheduled to which distributed processor?
• **Need:** To allocate the job for execution
Load Balancing

• **Issue:** Redistribution of processes/task in the DCS.
• **Redistribution:** Movement of processes from the heavily loaded system to the lightly loaded system
• **Need:** To improve the Distributed Systems’ throughput
Job Scheduling

Can be considered as a QUEUEING MODEL - MULTI JOB MULTI QUEUES SYSTEM
Job Scheduling Policies

- **Random:**
  - Simple and static policy
  - The job scheduler will randomly allocate the job to the site $i$ with some probability $p_i$, where $\sum p_i = 1$
  - No site state information is used

- **Cycle:**
  - The job scheduler will allocate the job to site $i$, if the previous job allocation was to site $i-1$ using the function $(i-1)+1$ mode N
  - It is semi static policy, where in the job scheduler remembers the previously allocated site.

- **Join the Shortest Queue (JSQ):**
  - The job scheduler remembers the size of local queue in each site.
  - The job will be allocated to the queue which is shortest at the point of arrival of that job.
Job Scheduling Policies – Parameters of Interest

• Mean response time of jobs
• Mean delay of the jobs
• Throughput of the system

• Obviously the JSQ is having better edge over other two in terms of these parameters.
Load Balancing

• Basically two types:
  – Sender Initiated Load Balancing
  – Receiver Initiated Load Balancing
Sender Initiated Load Balancing
Components of Sender Initiated Load Balancing

• Idea: Node with the higher load (sender) initiate the load balancing process.

• Transfer Policy
  — Policy about whether to keep the task/process at that site or transfer to some other site (or node)

• Location Policy
  — If decided to transfer, policy about where to transfer?

• Note that any load balancing algorithm should have these to components
Transfer Policy

- At each node, there is a queue
- If queue length of a node $< \tau$ (threshold)
  - Originating task is processed in that node only
- Else
  - Transfer to some other node

In this policy each node uses only local state information
Location Policy

• Random Policy

• Threshold Location Policy

• Shortest Location Policy
Random Policy

- Node status information are not used
- Destination node is selected at random and task is transferred to that node
- On receipt of the task, the destination node will do the following:
  - If the threshold of the node is $< \tau$, then accept the task
  - Else transfer it to some other random node
- If number of transfers reach some limit, $L_{\text{limit}}$ then, the last recipient of the task has to execute that task irrespective of its load. This is to avoid unnecessary thrashing of jobs.
Threshold location policy

- Uses node status information for some extent about the destination nodes.
- Selects the node at random. Then, probe that node to determine whether the transferring task to that node would place its load above the threshold.
  - If not, the task is transferred and the destination node has to process that task regardless of its state when the task actually arrives.
  - If so, select the another node at random and probe in the same manner as above.
- The algorithm continues either suitable destination is found or number of probes reaches some limit, $T_{\text{limit}}$. If the number of probes $> T_{\text{limit}}$ then, the originating node should process the task.
Shortest Location Policy

• Uses additional information about the status of other node to make “best” choice of the destination node.
• In this policy, $L_p$ number of nodes are chosen at random and each is polled to determine their queue length.
• The task is transferred to the node having shortest queue length among the one with threshold $< \tau$
• If none exist with the threshold $< \tau$, then the next $L_p$ number of nodes are polled and above step is repeated
• Once group of node selection reaches some limit, $L_s$, then the originator should handle the task
Receiver Initiated Load Balancing
Components of Receiver Initiated Load Balancing

• Idea: Under loaded node (receiver) initiate the load balancing process. Receiver tries to get the task from overloaded node, sender.

• Transfer Policy (threshold policy)
  – Where the decision is based on CPU queue length. If it falls below certain threshold, $\tau$, the node is identified as receiver for obtaining task from the sender

• Location Policy
  – If decided to receive, policy about from where to receive?
Location Policy

• Threshold Location Policy
  – A random node is probed to see whether it can become a potential sender. If so, the task is transferred from the polled node. Else, the process is repeated until a potential sender is found or number of tries reaches a PollLimit.

• Longest Location Policy
  – A pool of nodes are selected and probed to find the potential sender with longest queue length (greater than $\tau$). If found, then the task is received from that sender. Else the above process is repeated with new pool of nodes.
Drawback of Receiver Initiated Algorithm

• Most of the tasks selected for transfers from the senders are all preemptive one.
  – The reason is: The job scheduler always gives higher priority for allocating the fresh job to the processor compared to the existing processes at different stages of execution. So, by the time the receiver decide to pick the task, it underwent some execution.
Symmetrically Initiated Algorithm

- These are algorithms having both sender initiated and receiver initiated components.
- Idea is that at low system loads the sender initiated component is more successful in finding the under loaded nodes and at high system loads the receiver initiated component is more successful in finding the overloaded nodes.
Symmetrically Initiated Algorithm

- Above Average Algorithm

- Adaptive Algorithms
  - Stable Symmetrically Initiated Algorithm
  - Stable Sender Initiated Algorithm
Above Average Algorithm

• **Idea:** There is ‘acceptable range (AR)’ in terms of load is maintained.
  – The node is treated as sender if its load > AR
  – The node is treated as receiver if its load < AR
  – Else it is a balanced node.

• **Transfer Policy:** AR is obtained from two adaptive thresholds and they are equidistance from estimated average load of the system
  – For example, if the estimated average load of the system = 2, then lower threshold (LT) = 1 and upper threshold (UT) = 3
  – So, if the load of the node is <=LT then it is a receiver node. If the load of the node is >= UT, then it is a sender node. Balanced node otherwise.
Above Average Algorithm – Contd.

- **Location Policy**: Consists of two components
  - **Sender Initiated Component**:
    1. The node with load > AR is called sender. The sender broadcasts TOOHIGH message, sets TOOHIGH timeout alarm and listen for ACCEPT message.
    2. On receipt of TOOHIGH message, the receiver (whose load < AR)
       - cancels its TOOLOW timeout alarm
       - sends ACCEPT message to the node which has sent TOOHIGH message
       - increments its load value
       - set AWAITINGTASK timeout alarm
       - if no task transfer within AWAITINGTASK period, then its load value is decremented
    3. On receipt of ACCEPT message, sender sends the task to the receiver. [Note that the broadcasted TOOHIGH message will be received by many receiver and for the first ACCEPT message, the sender will transfer the task].
    4. On expiry of TOOHIGH timeout period, if no ACCEPT message is received by the sender, then sender infers that its estimated average system load is **too low**. To correct the problem, it broadcasts CHANGEAVERAGE message to increase the average estimated load at all other sites.
Above Average Algorithm – Contd.

– Receiver Initiated Component:
  1. Receiver broadcasts TOOLOW message, sets TOOLOW timeout timeout alarm and wait for TOOHIGH message.
  2. On receipt of TOOHIGH message, perform the activities as in step 2 of Sender Initiated Component.
  3. If TOOLOW timeout period expires, then it infers that its estimated average system load is too high and broadcasts CHANGE AVERAGE message to decrease the estimated average load at all sites.
Stable Symmetrically Initiated Algorithm

- **Idea:** In this algorithm, the information gathered during polling is used to classify the node as SENDER, RECEIVER or BALANCED.
- Each node maintains a list for each of the class.
- Since this algorithm updates its lists based on what it learns from (or by) probing, the probability of selecting the right candidate for load balancing is high.
- Unlike Above average algorithm, no broadcasting, hence the number of messages exchanges are less.
- Initially each node assumes that every other node is RECEIVER except itself. So, the SENDER and BALANCED lists are empty to start with.
Stable Symmetrically Initiated Algorithm – Contd.

• Transfer Policy:
  – This policy is triggered when the task originates or departs.
  – This policy uses two thresholds: UT (upper threshold) and LT (lower threshold).
  – The node is sender, if its queue length > UT, a receiver, if its queue length < LT and balanced if LT ≤ queue length ≤ UT
Stable Symmetrically Initiated Algorithm – Contd.

• **Location Policy:** Has two components
  – **Sender Initiated Component:**
    1. When the node becomes sender, it polls the node at the head of its RECEIVER list. The polled node removes the sender under consideration from its RECEIVER list and put it into the head of its SENDER list (i.e., it learns!). It also informs to the sender that whether it is a sender or a receiver or a balanced node.
    2. On receipt of the reply the sender do the following:
      • If the polled node is receiver, sender transfers the task to it and updates its list (putting the polled node at the head of RECEIVER or BALANCED list)
      • Otherwise, updates the list (putting the polled node at the head of SENDER or BALANCED list) and again start polling the next node in its RECEIVER list
    3. The polling process stops if
      • The receiver is found or
      • RECEIVER list is empty or
      • Number of polls reaches POLL-LIMIT
    4. If polling fails, the arrived task has to be processed locally. However, there is a chance of migration under preemptive category.
Stable Symmetrically Initiated Algorithm – Contd.

– **Receiver Initiated Component:**

1. When the node becomes receiver, it polls the node at its head of SENDER list. The polled node updates its list (i.e., places this node at the head of RECEIVER list). It also informs the receiver that whether it is a sender or a receiver or a balanced node.

2. On the receipt of reply the receiver do the following:
   - If the responded node is a receiver or a balanced node, then its list is updated accordingly.
   - Otherwise (i.e., responded node is a sender), the task sent by it is received and the list is updated accordingly

3. The polling process stops if:
   - The sender is found or
   - No more entries in the SENDER list or
   - The number of polls reaches POLL-LIMIT

- Note that at high load, receiver initiates the poll and at low load sender initiates the poll. So this will improve the performance.
Stable Sender Initiated Algorithm

• In this algorithm, there is no transfer of tasks when a node becomes receiver. Instead, its status information is shared. Hence there is no preemptive task transfers.

• The sender initiated component is same as that of stable symmetrically initiated algorithm. (like list generation, learning the status via polling ...etc)

• Stable sender initiated algorithm maintains an array (at each node) called status vector of size = number of nodes in DCS.
Stable Sender Initiated Algorithm

Status vector

1 2 ... j ... N

Node i

- The entry j in the status vector of node i indicates the best guess (receiver / sender / balanced node) about node i by the node j

- **SENDER INITIATED COMPONENT**
  - When the node becomes sender, it polls the node (say j) at the head of its RECEIVER list
  - The sender updates its jth entry of its status vector as sender.
  - Likewise, the polled node (j) updates its ith entry in its status vector based on its reply it sent to the sender node.
  - Note that above two aspects are additional information it learns along with the other things as in sender component of stable symmetrically initiated algorithm

- **RECEIVER INITIATED COMPONENT**
  - When the node becomes receiver, it checks its status vector and informs all those nodes that are misinformed about its current state.
  - The status vector at the receiver side is then updated to reflect this changes
Stable Sender Initiated Algorithm

• Advantages:
  – No broadcasting of messages by the receiver about its status
  – No preemptive transfer of jobs, since no task transfers under receiver initiated component
  – Additional learning using status vector reduces unnecessary polling
Challenges Load Balancing Algorithm

• Scalability:
  – Ability to make quicker decision about task transfers with lesser efforts

• Location transparency:
  – Transfer of tasks for balancing are invisible to the user.

• Determinism:
  – Correctness in the result inspite of task transfers

• Preemption:
  – Transfer of task to the node should not leads degraded performance for the task generated at that node. So, there is a need for preemption of task when the local task arrives at node

• Heterogeneity:
  – Heterogeneity in terms of processors, operating systems, architecture should not be hindrance for the task transfers.
Summary

• Differences between CPU scheduling, Job scheduling and load balancing are discussed.

• Different load balancing algorithms are discussed. They are categorized under sender initiated, receiver initiated, symmetrically initiated and variations of symmetrically initiated algorithms are discussed.
References

• Advanced Operating Systems

• Distributed Operating Systems
  – A S Tanenbaum, Prentice Hall

• Distributed Systems Concepts and Design
  – G Coulouris and J Dollimore, Addison Wesley