Operating Systems Z02-54c

Processor scheduling in distributed systems

(random, maximum threshold, minimum threshold)

Mustafa Tayyip BAYRAM 257639

OVERVIEW

A distributed system is a collection of large number of autonomous computers connected by a communication network which operate in a unified way to accomplish better performance and throughput. One important feature of a distributed system is transparency which hides the availability of multiple resources of different type from the users of the system. Potential power of a distributed system comes from the way it manages its resources. Management of processing resources (processors or CPUs) is done by two policies of the system, namely, processor allocation and processor scheduling.

Load distributing algorithms can be classified as static, dynamic or adaptive. Static means decisions on assignment of processes to processors are hardwired into the algorithm, using a priori knowledge, perhaps gained from analysis of a graph model of the application. Dynamic algorithms gather system state information to make scheduling decisions and so can exploit under-utilised resources of the system at run time but at the expense of having to gather system information. An adaptive algorithm changes the parameters of its algorithm to adapt to system loading conditions. It may reduce the amount of information required for scheduling decisions when system load or communication is high.

Most models assume that the system is fully connected and that every processor can communicate, perhaps in a number of hops, with every other processor although generally, because of communication latency, load scheduling is more practical on tightly coupled networks of homogeneous processors if the workload involves communicating tasks. In this way also, those tasks might take advantage of multicast or broadcast features of the communication mechanism.

A load distributing algorithm has, typically, four components:- transfer, selection, location and information policies.

For Site Location Policy

Processor scheduling in distributed systems (processor assignment for a process) – simulation algorithms:

Random -> this is Sender-Initiated Distributed Heuristic Algorithm

Maximum threshold, minimum threshold -> this is Receiver-Initiated Distributed Heuristic Algorithm

Sender-initiated distributed heuristic algorithm (Eager Algorithm)

- Step 1: when a process is created , it runs on the node that created it unless node is overloaded
    - Overload metrics – too many processes; too big total working set, …
- Step 2: If a node is overloaded, the node selects another node at random and asks it what its load is
- Step 3: If probed node's load is below some threshold value, the new process is sent there
- Step 4: If not, another machine is chosen for probing
    - Probing is not done forever,
- Step 5: If no suitable host is found within N probes, algorithm terminates and the process runs on the originating machine

Receiver-initiated distributed heuristic algorithm

- Step 1: Whenever a process finishes , system checks to see if it has enough work,
- Step 2: If not, it picks some machine at random, and asks for work,
- Step 3: If that machine has nothing to offer, a second and then third machine is asked.
- Step 4: If no work is found after N probes, the node temporarily stops asking, does any other work it has queued up, and tries again after another process finishes.
- Step 5: If no work is available, machine is idle.
- Step 6: After some fixed interval , it begins probing again.

Algorithm Based On Random Location Policicy

The algorithm uses a very simple location policy called. Random which uses no information about other nodes in the system. A task is simply transferred to a node selected at random with no exchange of state information among the nodes. A task transfer may prove to be useless if the receiving node is already overloaded. How should the destination node treat an arriving transferred task ? Well, a simple answer could be that the destination node should treat the transferred task as if a new task arrives there. If so, it can again be transferred to another node at random in case it is also overloaded. This might eventually lead to system instability irrespective of the average load of the system so that the nodes are spending most of their time transferring tasks, with a little or no time spent executing.
The problem of the system instability could be resolved with a simple solution that limits the number of times a task can be transferred using a static transfer limit. This extremely simple combination of a Threshold transfer policy with a static transfer limit provides substantial performance gain over system with no load sharing.

Algorithm Based On Threshold Location Policicy

The objective of the Threshold location policy is to avoid the useless task transfers in the random policy. This objective is achieved at the cost of acquiring and using a small amount of information about candidate destination node. A node is selected randomly and probed to determine whether transferring a task would make its queue length exceed threshold T. If not, the task is transferred to selected node, it must process the task regardless of its state when the task actually arrives. Otherwise, the process of selecting a node at random and probing it keeps on as long as either the suitable node is found or the static probe limit is reached. On non-availability of suitable node within the probe limit, the originating node must execute the task. The threshold location policy provides a substantial performance improvements over random location policy.

Sender Initiated versus Receiver Initiated Algorithms

Maintaining a complete database of idle or lightly loaded machines for migration can be a challenging problem in a distributed system. Various alternatives to this have been proposed.

With sender initiated schemes the overloaded machine takes the initiative by random polling or broadcasting and waiting for replies. With receiver initiated schemes an idle machine offers itself for work to a group of machines and accepts tasks from them.

In some situations a machine can become momentarily idle even though on average it is reasonably busy. As discussed earlier, care must be taken to coordinate migration activity so that senders do not capitalise on this transient period and send a flood of processes to this node. One approach is to assign each site a local load value and an export load value. The local load value reflects the true state of the machine while the export load value decreases more slowly to dampen short-lived fluctuations.