

Mechanisms for QoS Driven Scheduling

Managing Tasks in a Distributed environment

Scheduling and QoS

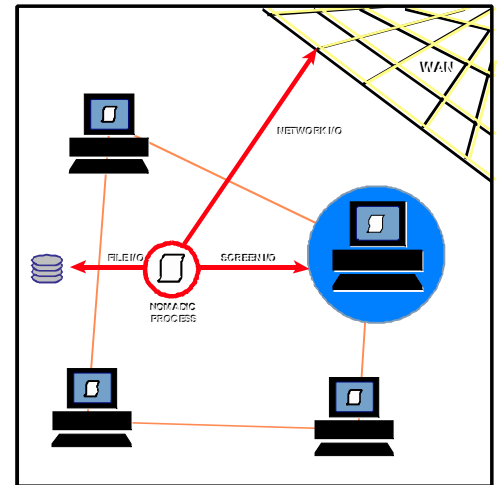
The primary mechanism for the provision of guaranteed quality of service in a distributed computing system is the usage of intelligent scheduling mechanisms. A distributed computation consists of a set of tasks, some are interrelated, some are not. The network bandwidth needed to run these tasks depends upon the granularity of the tasks, co-location and the degree of fault tolerance needed. Also, the turnaround time of a computation is determined by how well the tasks are spread on the set of computing engines.

The QoS metrics we use to determine the efficacy of the underlying system is:

1. *Dilation*: The ratio of actual execution time over ideal execution time. This metric is related to the well-known metric of timeliness.
2. *Fault-Tolerance*: The number of failures that can be tolerated by the running computation.
3. *Network-Demand*: The bandwidth usage of the underlying network. This metric can be varied to control network usage.

These metrics can be tuned to a particular application running on a network with a given QoS properties by using several tunable scheduling mechanisms we have developed. The parameters of the tunable scheduler

1. *Bunching (Granularity Management)*: High granularity leads to better dilation (shorter execution times), but requires higher reliability from the underlying processing system.
2. *Memory Checkpoint Frequency*: The number of times the memory is synchronized in a bunched execution – this dictates the fault-masking behavior and is dependent on the network bandwidth available.



3. *Preemption*: Usage of preemption increases the timeliness, but required higher network bandwidth.

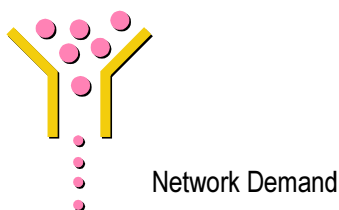
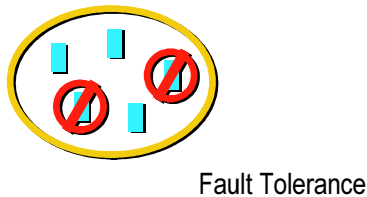
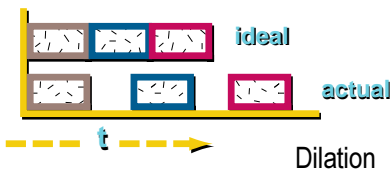
The environment

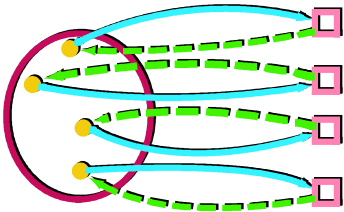
Consider a cluster of workstations, running a parallel application. The application divides itself into a set of tasks. Each task can be long running, or very short running. The scheduler assigns these tasks to a set of workstations. Often the tasks are not of equal length, the machines are not of equal speeds and tasks can create further subtasks. These situations lead to non-optimal matches of workers to tasks causing executions that do not complete as quickly as it would be possible in a better-matched case. Also, the granularities of the tasks may be small, leading to high overhead.

The goodness of a match and the need for managing granularities produces different strategies that has impact on the network traffic generated by the management system. The QoS needs of the application can be used to tune the scheduling of such schedulers to ensure a balance between network usage and speedup.

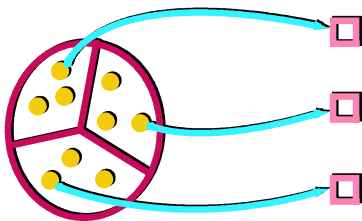
Distributed Scheduling

Our research has addressed such problems in a variety of ways. We have developed scheduling algorithms, both non-preemptive and





Distributed Fault-Tolerant Round Robin



Preemptive Task Bunching

preemptive that provide good throughputs in managing distributed computations, even when the granularities of tasks are small. These techniques lend themselves as ideal mechanisms for QoS driven scheduling. The two major technologies used in building our schedulers are preemption and task bunching. Also intra-bunch checkpointing is another technique of providing higher QoS at the expense of higher Network Demand.

Preemptive Scheduling

All multitasking operating systems use preemptive scheduling. Many multiprocessor systems also employ preemptive inter-task scheduling when they run parallel computations. However, preemptive scheduling in distributed systems is rare, if not nonexistent. We have found that two particular pre-emptive schedulers have characteristics suitable for QoS driven scheduling in distribute systems.

The first algorithm is a variation of the well-known round robin algorithm. We call this the *Distributed, Fault-tolerant Round Robin algorithm*. In this algorithm, a set of n tasks is scheduled on m machines, where n is larger than m . Initially, the first m tasks are assigned to the m machines. Then, after a specified amount of time (time quantum), all tasks are preempted and the next m tasks are assigned. This continues in a circular fashion until all tasks are completed.

The second is the *Preemptive Task Bunching* algorithm. All n tasks are bunched into m bunches and assigned to the m machines. When a machine finishes its assigned bunch, all the tasks on all other machines are preempted and all the remaining tasks are collected and re-bunched (into m sets) and assigned again. This algorithm works well for both large-grained and fine-grained tasks even when machine speeds and task lengths vary.

Performance

The final system runs well, and performance results are very encouraging. We found that

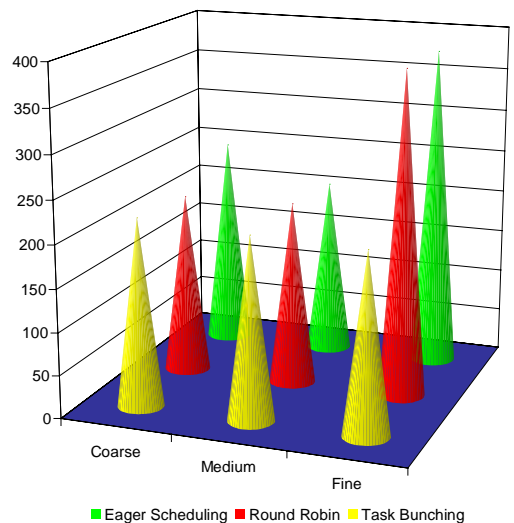
the round-robin scheduler provided acceptable performance on large grained programs, but was hampered by the migration overhead. The task bunching scheduler performed really well in a wide variety of situations.

We are currently incorporating these methods into a front end that controls the QoS metrics for the distributed computation, for later incorporation into a comprehensive resource management system.

**MILAN is a joint project of
New York University
Arizona State University**

**Zvi M. Kedem
New York University
+1 212 998 3101 (phone)
+1 212 477 3265 (fax)
kedem@cs.nyu.edu
<http://www.cs.nyu.edu/milan>**

**Partha Dasgupta
Arizona State University
+1 602 965 5583 (phone)
+1 602 965 2751 (fax)
partha@asu.edu
<http://milan.eas.asu.edu>**



Effects of different scheduling protocols on computations with varying granularities