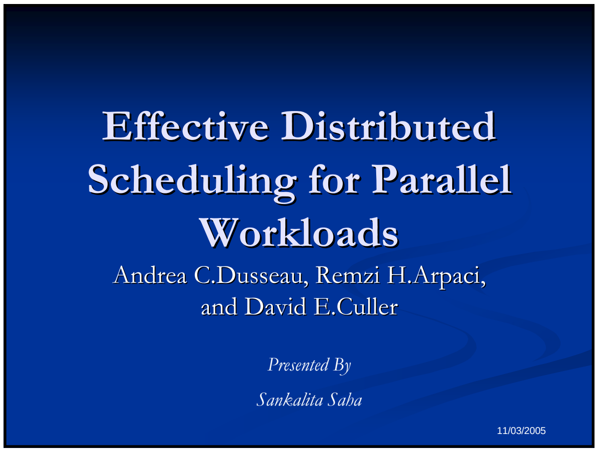


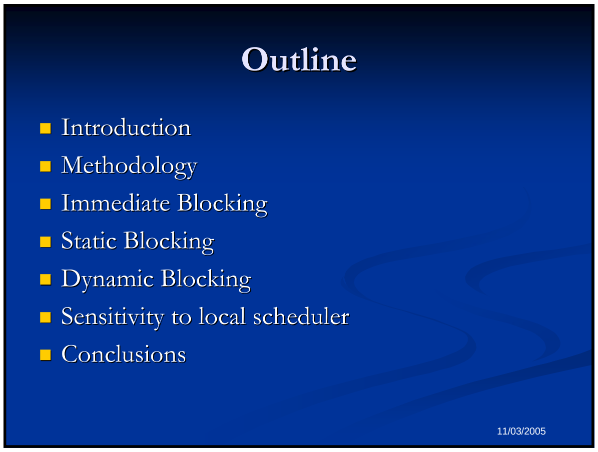
# Effective Distributed Scheduling for Parallel Workloads

Andrea C.Dusseau, Remzi H.Arpaci,
and David E.Culler

*Presented By*

*Sankalita Saha*

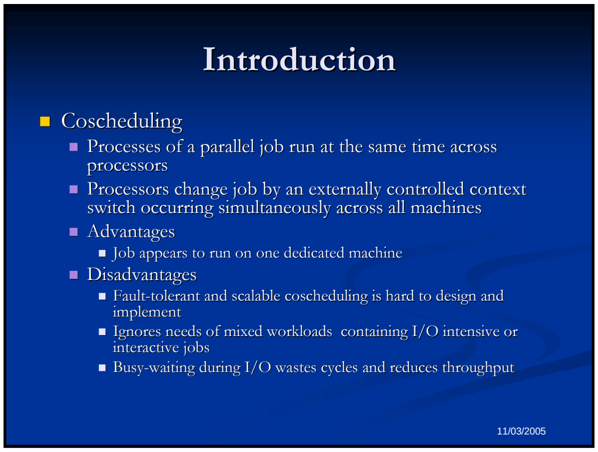
11/03/2005



## Outline

- Introduction
- Methodology
- Immediate Blocking
- Static Blocking
- Dynamic Blocking
- Sensitivity to local scheduler
- Conclusions

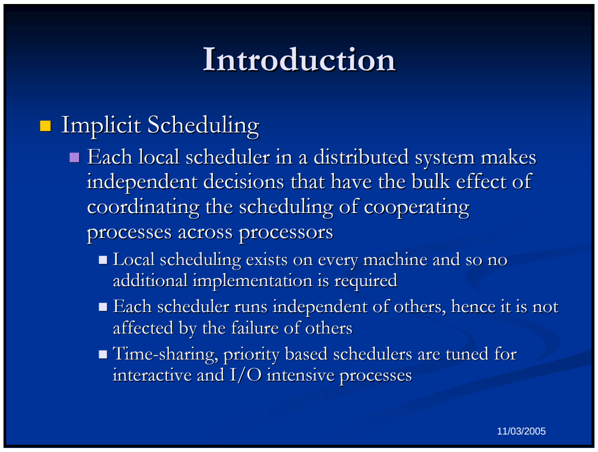
11/03/2005



## Introduction

- Coscheduling
  - Processes of a parallel job run at the same time across processors
  - Processors change job by an externally controlled context switch occurring simultaneously across all machines
  - Advantages
    - Job appears to run on one dedicated machine
  - Disadvantages
    - Fault-tolerant and scalable coscheduling is hard to design and implement
    - Ignores needs of mixed workloads containing I/O intensive or interactive jobs
    - Busy-waiting during I/O wastes cycles and reduces throughput

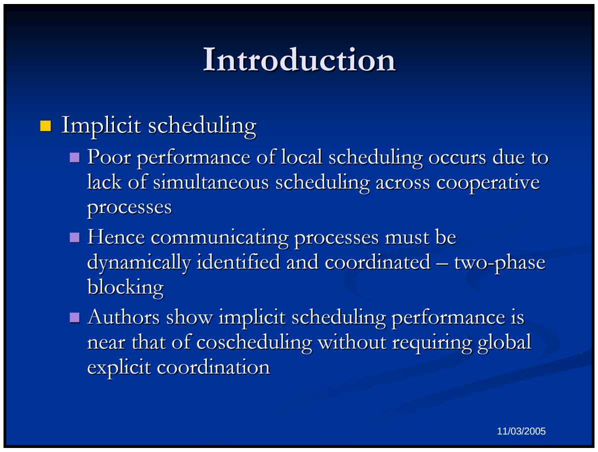
11/03/2005



## Introduction

- Implicit Scheduling
  - Each local scheduler in a distributed system makes independent decisions that have the bulk effect of coordinating the scheduling of cooperating processes across processors
    - Local scheduling exists on every machine and so no additional implementation is required
    - Each scheduler runs independent of others, hence it is not affected by the failure of others
    - Time-sharing, priority based schedulers are tuned for interactive and I/O intensive processes

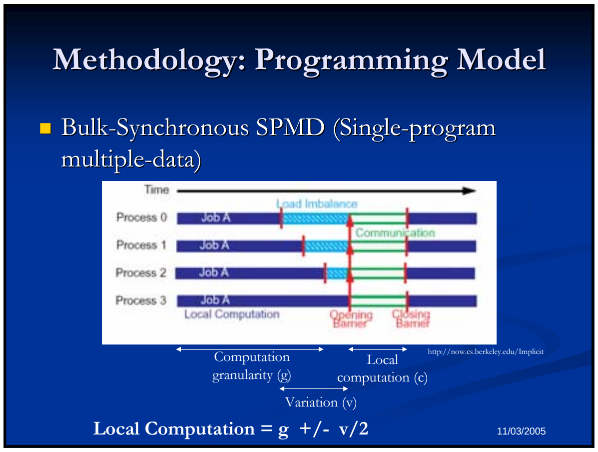
11/03/2005



## Introduction

- Implicit scheduling
  - Poor performance of local scheduling occurs due to lack of simultaneous scheduling across cooperative processes
  - Hence communicating processes must be dynamically identified and coordinated – two-phase blocking
  - Authors show implicit scheduling performance is near that of coscheduling without requiring global explicit coordination

11/03/2005



## Methodology: Programming Model

- Bulk-Synchronous SPMD (Single-program multiple-data)

http://now.cs.berkeley.edu/Implicit

**Local Computation = g +/- v/2**

11/03/2005

## Methodology: Simulation Parameters

- 3 different communication patterns are investigated
  - BARRIER
    - No communication
  - NEWS
    - Grid communication pattern
    - Each process depend upon its 4 neighbors
  - TRANSPOSE
    - P read phases
    - On *i*th read  process *p* reads data from process *(p+i) mod P*

## Methodology: Local Scheduling

- Local scheduler closely matches the scheduler of Unix System V Release 4 (SVR4)
  - When a job sleeps on an event, it is placed in blocked list and is not scheduled
  - When a message arrives for the blocked job, it is given the highest priority and it handles the message
  - If the message unblocks the job, then it is given a new priority and placed on the run queue, else it is returned to the blocked list

## Methodology: Local Scheduling

- Other characteristics of the scheduler
  - Job's priority is lowered if it runs for its time quanta without releasing the processor; job's priority is raised if it sleeps frequently
  - A starvation timer runs every second; a job's priority increases if doesn't completes it's time quanta before the starvation timer expires
  - The individual quanta and starvation timers expire independently across processors
  - If multiple parallel jobs arrive at the same time, the processes are randomly ordered in the local scheduling queues

## Immediate Blocking

- Gives superior performance for coarse and medium-grain computations coupled with high load-imbalance
  - When imbalance is high, processor can switch to another job and execute it instead of spinning uselessly
- Coscheduling is strictly superior for fine-grained jobs regardless of load imbalance

## Immediate Blocking

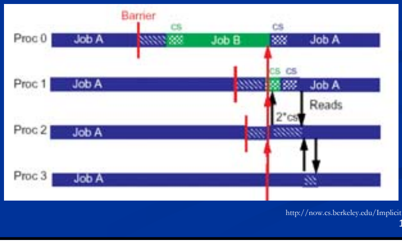
- Sensitivity to machine parameters
  - For high-latency network and low context-switch cost, local  scheduling with immediate blocking outperforms coscheduling

## Static Blocking Algorithms

- Spin-time = context-switch time
  - For coarse-grained high load imbalance programs, performs similar to immediate blocking
  - For fine-grained computations, performs better than immediate blocking
    - Cooperative processes become dynamically coordinated after executing barrier due to scheduler policy of raising priority of process when it receives a message

## Static Blocking Algorithms

- Spin-time ≥ 2 × context-switch time
  - To maintain dynamic scheduling coordination, processes at reads and barriers must spin at least for the amount of skew due to distributed scheduling



http://now.cs.berkeley.edu/Implicit

---

## Static Blocking Algorithms

- Spin-time ≥ 2 × context-switch time
  - Performance improves
  - If load imbalance is slightly greater than spin-time, then performance falls as processes don't wait long enough

---

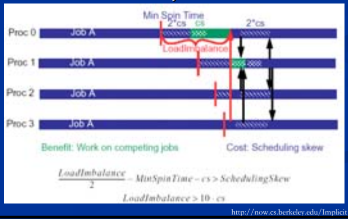
## Adaptive Blocking Algorithms

- Load-Imbalance Oracle
  - There exists a minimum spin-time before blocking *S*, that ensures that the coordinated processes remain in coordination.
    - *S* = 2 × context-switch time
  - There exists a load-imbalance, *V*, past which it is more beneficial to block at a barrier than to spin until barrier completes
    - *V* > 10 × context-switch time
  - For all load-imbalances greater than V, this outperforms coscheduling

---

## Adaptive Blocking Algorithms

- Load-Imbalance Oracle
  - Load imbalance *v* is determined by comparing the amount of useful work done by switching to another process and loss to the system due to blocking



http://now.cs.berkeley.edu/Implicit

---

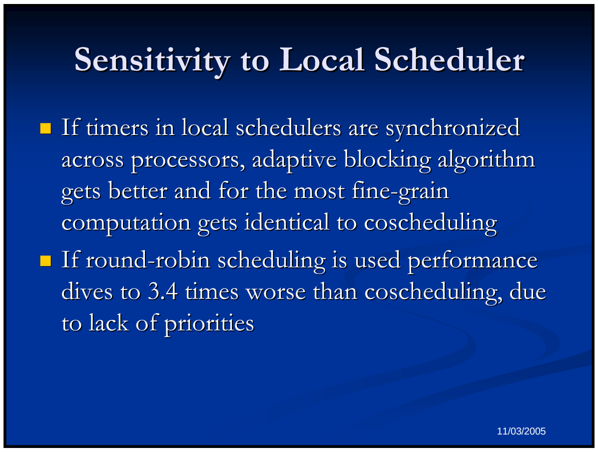
## Adaptive Blocking Algorithms

- Local Approximation of Load-Imbalance
  - Each executing process approximates load imbalance by sampling the time for barriers to complete
    - If scheduling of the job is coordinated across processors, then each process sees an average wait-time of $v/2 + 2\times L$ for opening barrier and $2\times L$ for closing barrier
    - L is the network latency
  - *v* is approximated by recording wait-times and removing largest 10% of data as outliers due to scheduling irregularities and then using the largest
  - Performance worsens for fine grain processes compared to oracle as some valid wait times may be thrown out

---

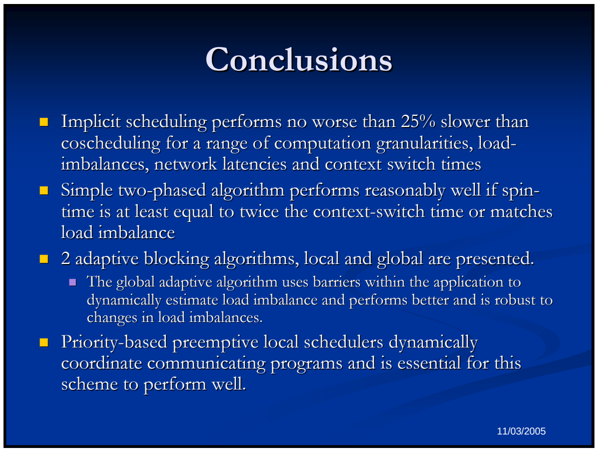
## Adaptive Blocking Algorithms

- Global Approximation of Load-Imbalance
  - Root process in a barrier can observe distribution of waiting times more accurately than individual processes.
  - Each process sends a message to root process on barrier completion which after receiving all messages broadcasts a barrier completion message.
  - Root process calculates appropriate spin time for barriers and provides that figure along with barrier completion notifications
  - Performs as well as oracle



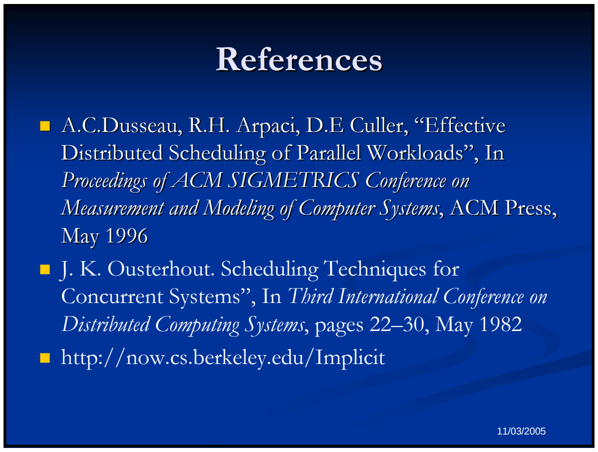
## Sensitivity to Local Scheduler

- If timers in local schedulers are synchronized across processors, adaptive blocking algorithm gets better and for the most fine-grain computation gets identical to coscheduling
- If round-robin scheduling is used performance dives to 3.4 times worse than coscheduling, due to lack of priorities



## Conclusions

- Implicit scheduling performs no worse than 25% slower than coscheduling for a range of computation granularities, load-imbalances, network latencies and context switch times
- Simple two-phased algorithm performs reasonably well if spin-time is at least equal to twice the context-switch time or matches load imbalance
- 2 adaptive blocking algorithms, local and global are presented.
  - The global adaptive algorithm uses barriers within the application to dynamically estimate load imbalance and performs better and is robust to changes in load imbalances.
- Priority-based preemptive local schedulers dynamically coordinate communicating programs and is essential for this scheme to perform well.



## References

- A.C.Dusseau, R.H. Arpaci, D.E Culler, “Effective Distributed Scheduling of Parallel Workloads”, In *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, ACM Press, May 1996
- J. K. Ousterhout. Scheduling Techniques for Concurrent Systems”, In *Third International Conference on Distributed Computing Systems*, pages 22–30, May 1982
- http://now.cs.berkeley.edu/Implicit