Communication-Driven Scheduling: Incorporating Parallel Jobs into the UNIX Environment

Andrea C. Dusseau

Computer Science Division
University of California, Berkeley

Motivation

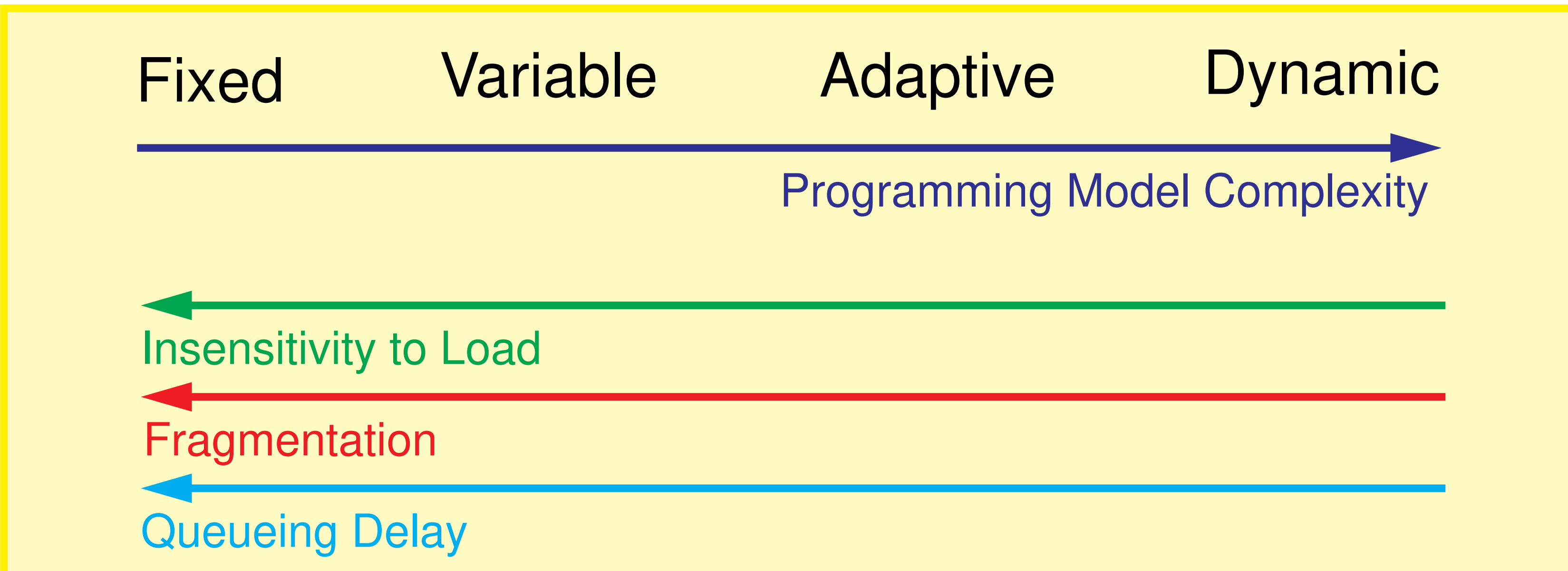
- How should we schedule parallel jobs in NOW?
- Look at scheduling of parallel jobs in MPPs
  - \+ Supports fine-grain communication by providing illusion of whole machine
  - \- Poor interaction with I/O system
  - \- Same policies for short and long jobs
- Look at scheduling of sequential jobs in UNIX
  - \+ Shares efficiently through multi-level feedback queues
  - \+ Sleep while waiting for I/O and paging
  - \+ Reward short/interactive jobs with higher priority
  - \- No notion of communicating/parallel jobs
- Goal: Combine best characteristics of both environments

Outline

- Background
  - Parallel Scheduling in MPPs
    - Space-Sharing
    - Time-Sharing
  - "Parallel" Scheduling in Unix
    - Simulation Results
- Communication-Driven Scheduling
  - Policy
  - Preliminary Simulation Results
- Conclusions & Future Work

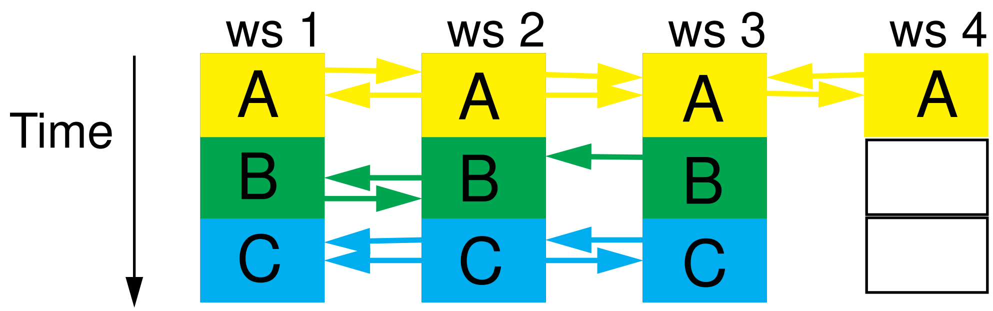
Parallel Scheduling: Space-Sharing

- Advantages of Space-sharing vs. Time-sharing
  - + Reduce operating system overhead of context-switching
  - + Provide all physical memory to one application
  - + High throughput due to sublinear speedup
- Four types of Space-sharing



Parallel Scheduling: Time-Sharing

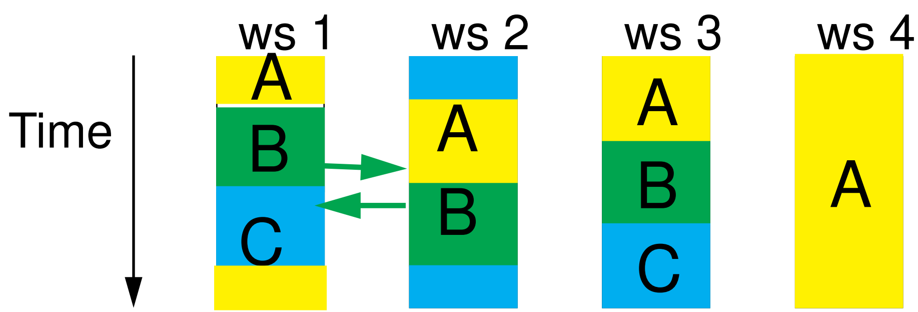
- Advantages of Time-sharing vs. Space-sharing
  + No queueing delay to start job
  + Interactive response times for short jobs
  + No restrictions on programming model
- Gang or Co-scheduling: Simultaneous context-switch



- Assume Time-sharing is used
  - May partition machine at higher level
  - Both parallel and sequential share machine

Parallel Jobs in Unix

- No concept of parallel jobs
  - Each independently schedules job with highest priority
- Dynamic priority based on characteristics of job
  - Reward interactive jobs with higher priority
  - Penalize CPU hogs with lower priority
- Local scheduling vs. Coscheduling



Messages may arrive for non-scheduled job

+ Scalable, fault-tolerant
+ Supports non-blocking I/O
+ Share with sequential jobs

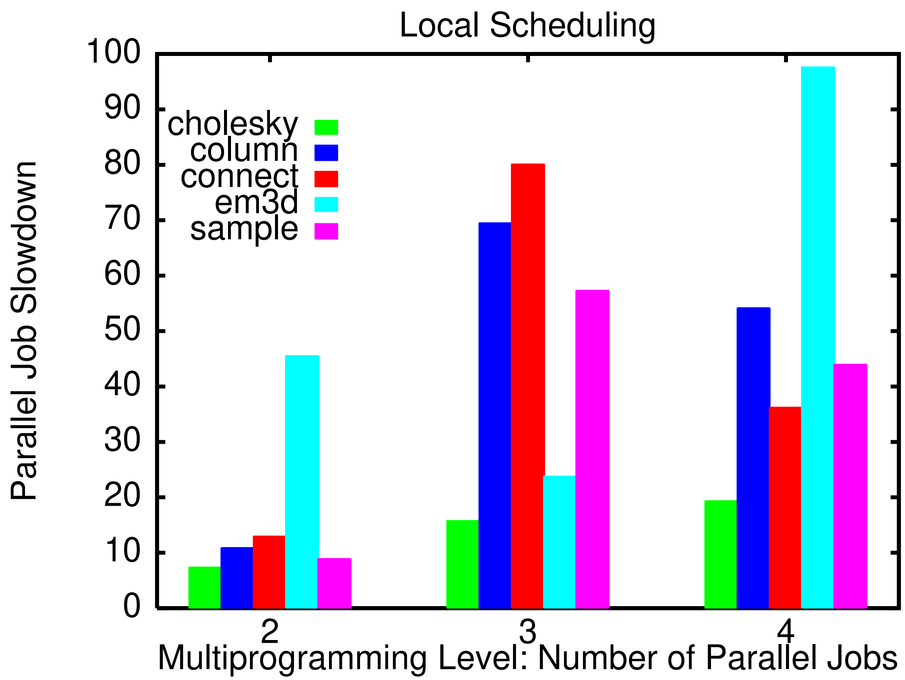
- Large delays for communication

Local Scheduling Simulations

How does local scheduling perform vs. coscheduling?

5 Split-C Kernels

em3d synchronizes most frequently (10 ms between barriers)



Run one application on 64-node CM-5

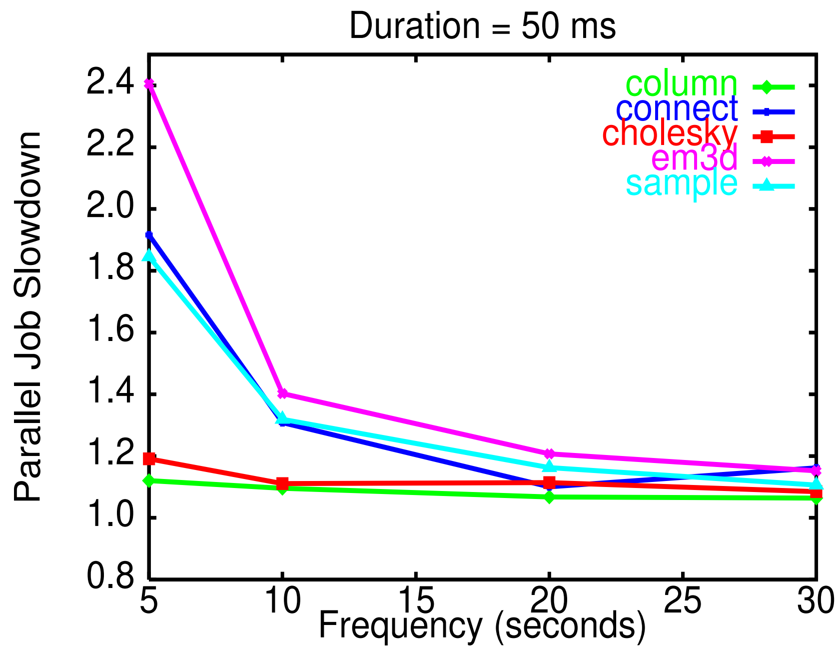
Simulate other jobs by inserting delays into execution

Nearly 100x slowdown with 4 parallel jobs

Communication dependencies lead to unacceptable performance

Local Scheduling Simulations

What is the effect of locally scheduled sequential jobs?



One sequential job can block parallel job on all nodes

Schedule sequential jobs between communication phases of parallel job

Communication-Driven Scheduling

- Goals: Achieve best of local scheduling and coscheduling
  - Distributed: Local scheduling decisions only
  - Efficient and Fair: Processor sharing
  - Dynamic: Coschedule communication phases
- Modify local scheduling policy to consider communication

$$Priority = f(CPUPriority, CommunicationPriority)$$

- CPU Priority: Maintain fairness while sharing CPU
  - Original SVR4 priority of job
- Communication Priority: Hints to reach consensus
  - Boost priority while receiving messages
  - Priority of sending process carried to receivers

Unix SVR4

- Each job belongs to a priority class

Real-time	100-159	CPU Priority
System	60-99	
Time-shared	0-59	

- Time-shared class
  - CPU Priority = User Priority + Kernel Priority
- User Priority: Modified by system calls
- Kernel Priority: Dynamic
  - Job's characteristics (CPU usage)
  - Entries in `Dispatch Parameter Table`

SVR4 Time-Shared Class

- Dispatch Parameter Table

global priority	quantum length	quantum level	sleep level	starve length	starve level
0	20	0	50	1	50
1	20	0	50	1	50
⋮	⋮	⋮	⋮	⋮	⋮
10	16	1	51	1	51
11	16	2	51	1	51
⋮	⋮	⋮	⋮	⋮	⋮
58	4	48	58	1	59
59	2	49	59	32000	59

Communication Priority

$$CommunicationPriority = f(ReceptionRate, CarryPriority)$$

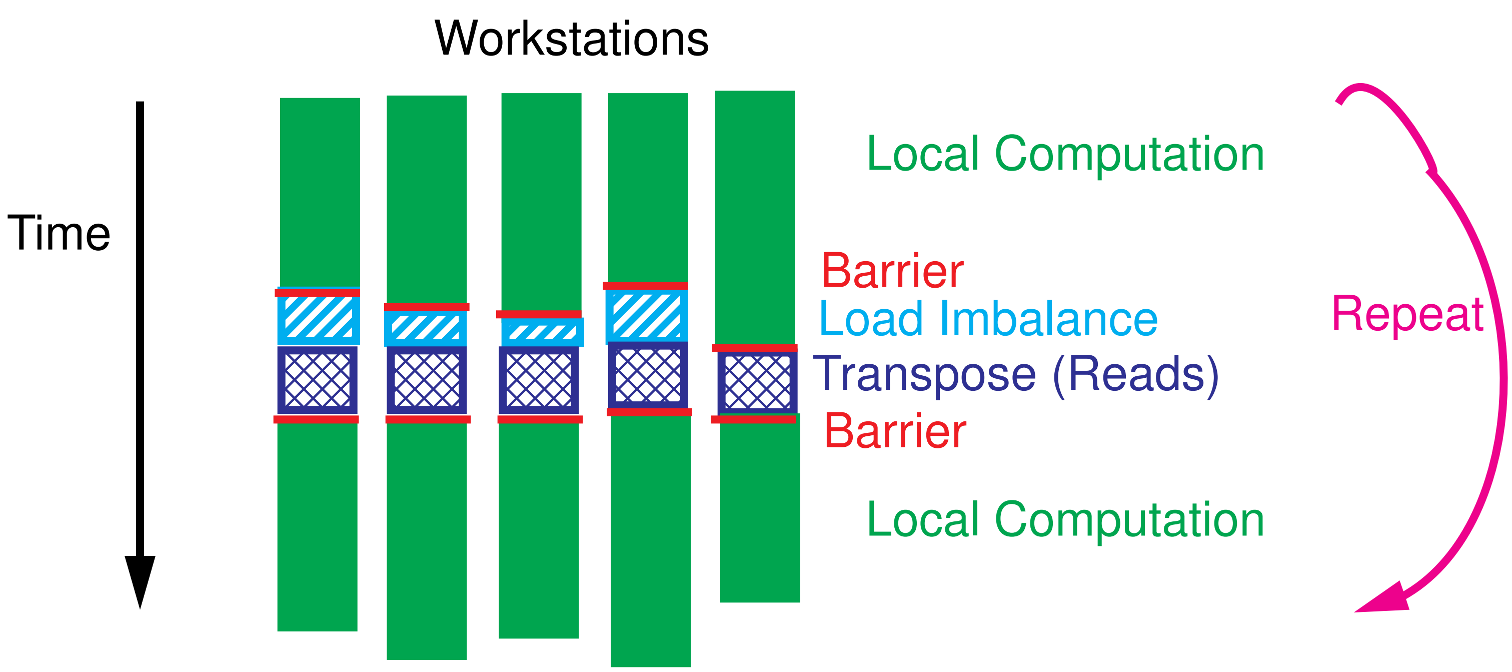
- Reception Rate: Schedule when receiving from many processes
  - Increment count when receive message
  - Decay over time
- Carry Priority: Schedule when receiving from new and high-priority processes
  - Priority of sender attached to each message
  - Begin at CPU priority
  - Decay over time

Non-Blocking Communication

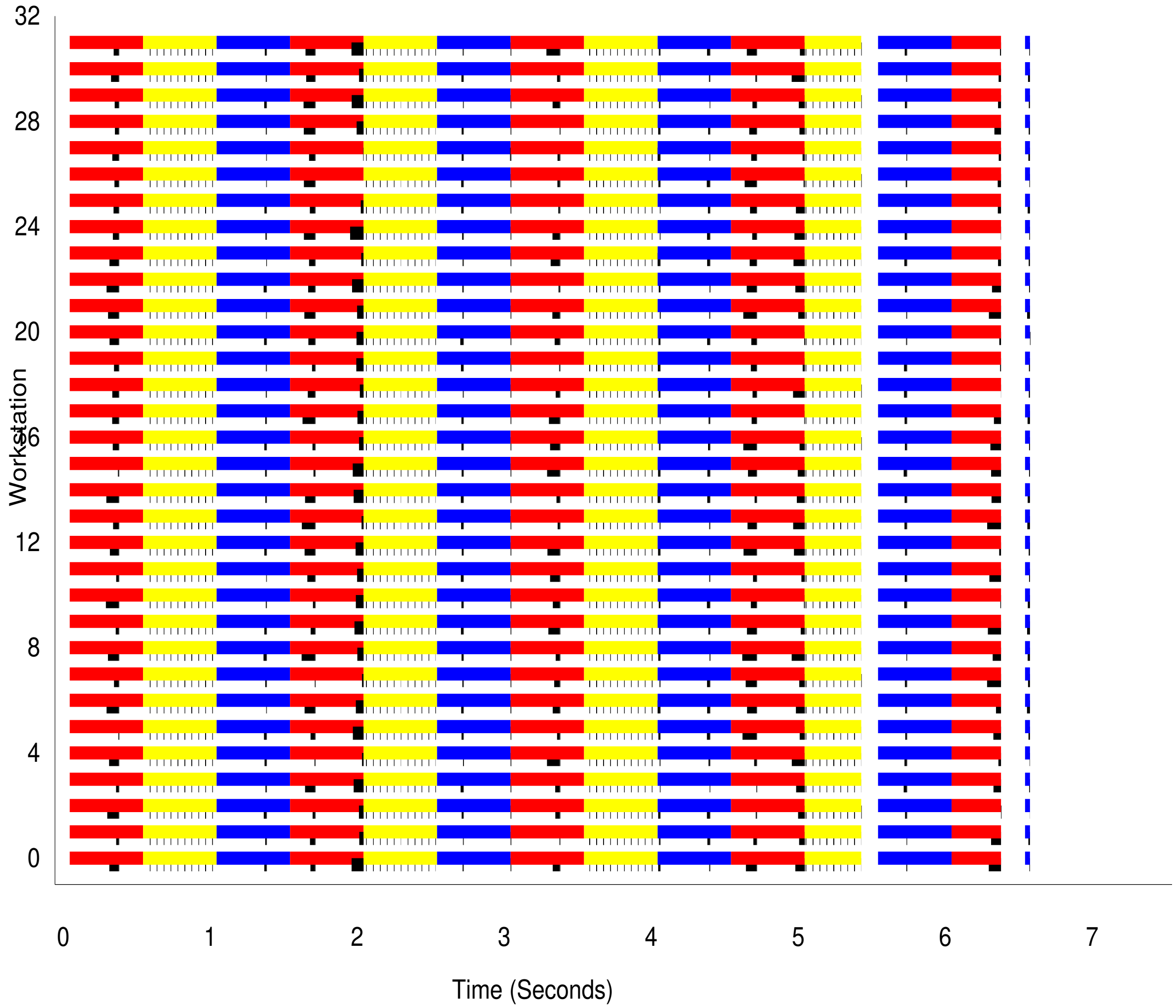
- Problem: Destination process not be scheduled
  - Long communication/synchronization delays
  - Solution: Time-Out for Read-Requests and Barriers
- When Time-Out occurs, schedule another process
- Unblock process when message arrives
- How to pick Time-Out length?
  - Greater than Round-Trip + Context-Switch
    - Message may schedule destination process
  - Barrier Time-Out: Function of Communication Priority
    - Distinguish between slow and not-scheduled processes
    - Other processes scheduled, but not at barrier

Simulation Environment

- Event-Driven Simulator
  - Synthetic Bulk-Synchronous Applications
  - Arrival Time, Number of Workstations



Coschedule Performance



3 Jobs

32 Workstations

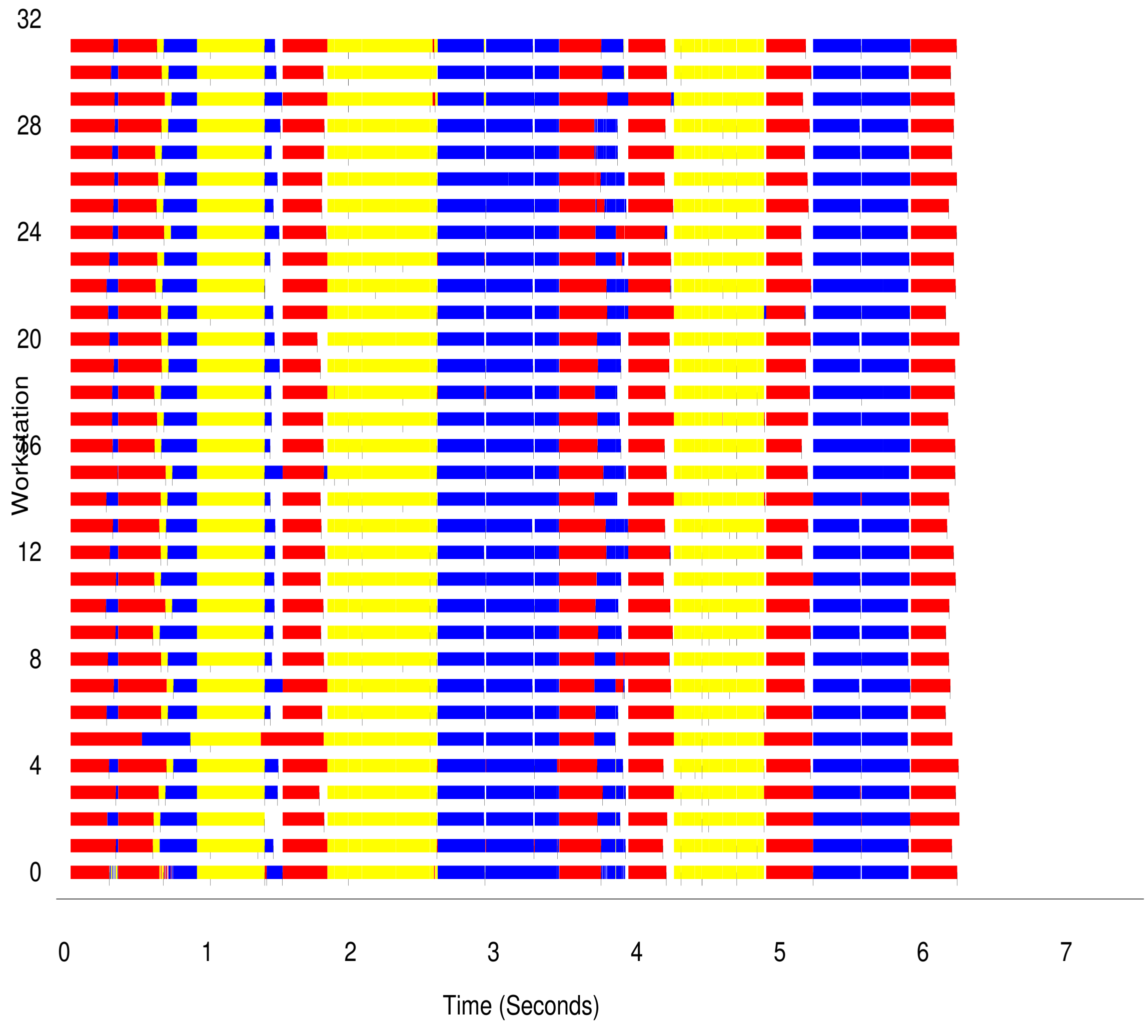
500ms quantum

Local: 250ms
Imbalance:100ms
Phases: 6

Local: 45ms
Imbalance:1ms
Phases: 40

Local: 325ms
Imbalance:20ms
Phases: 5

Communication-Driven Performance



32 Workstations

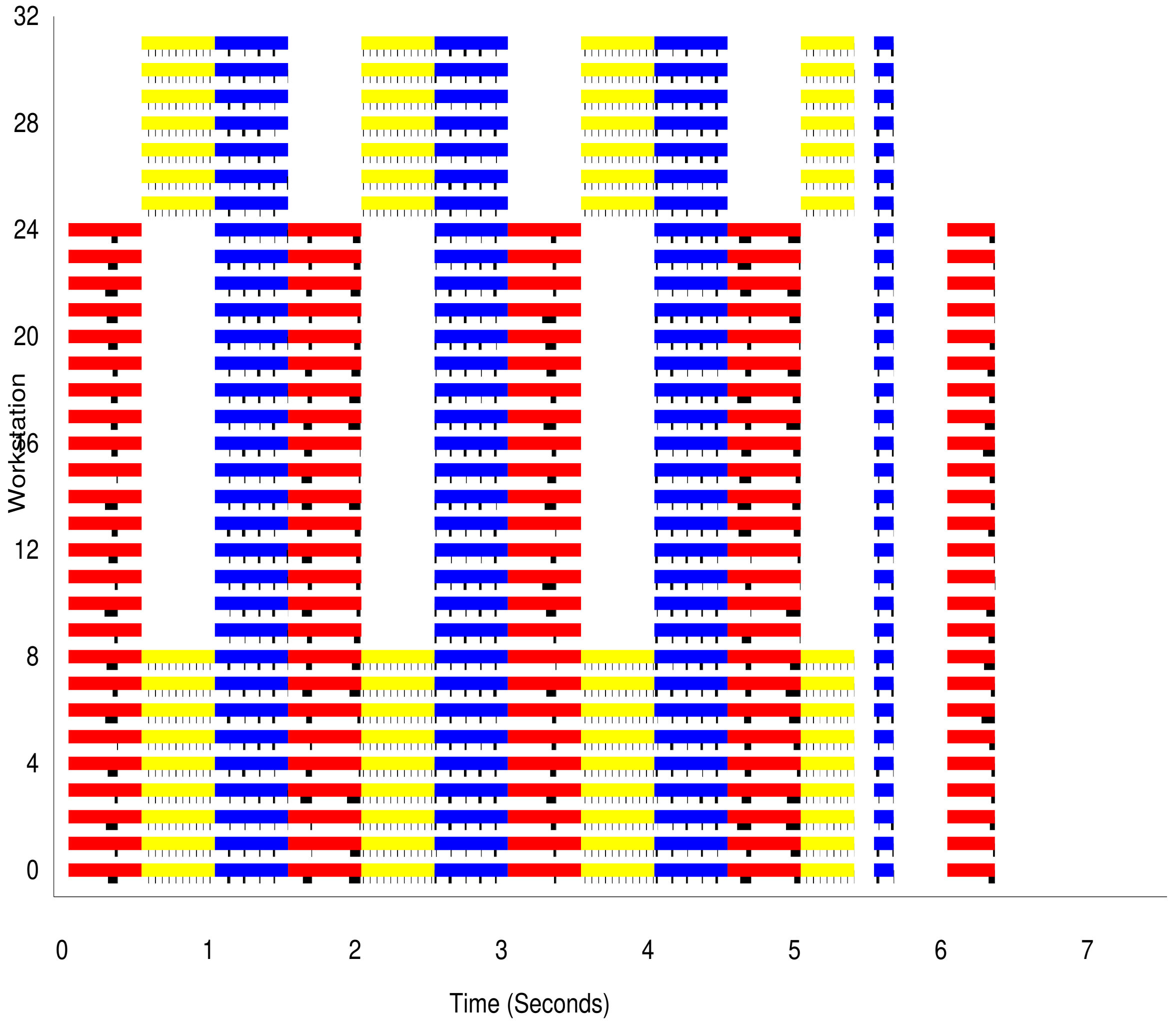
Context-Switch: 100us

Local: 250ms
Imbalance:100ms
Phases: 6

Local: 45ms
Imbalance:1ms
Phases: 40

Local: 325ms
Imbalance:20ms
Phases: 5

Coschedule Performance

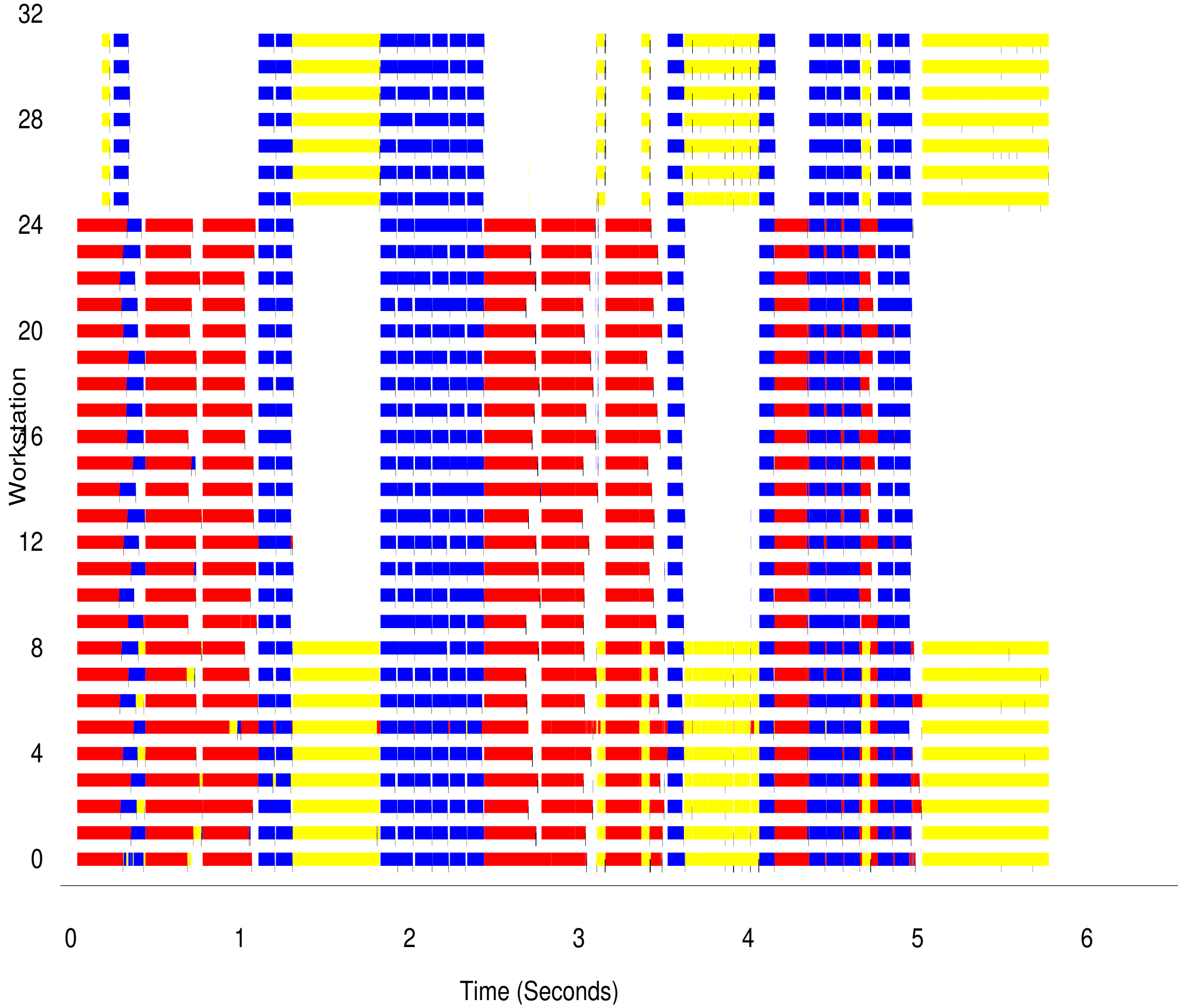


Local: 250 ms
Imbalance:100ms
Phases: 6
Workstations: 25

Local: 45 ms
Imbalance:1 ms
Phases: 40
Workstations: 16

Local: 82 ms
Imbalance:20ms
Phases: 5
Workstations: 32

Communication-Driven Performance



Local: 250 ms
Imbalance:100ms
Phases: 6
Workstations: 25

Local: 45 ms
Imbalance:1 ms
Phases: 40
Workstations: 16

Local: 82 ms
Imbalance:20ms
Phases: 5
Workstations: 32

Conclusions

- Appears to have potential...

Future Work

- Simulations
  - More synthetic communication patterns
  - Use traces from Split-C applications
  - Incorporate interactive sequential jobs
  - Incorporate I/O
- Implementation
  - Add parallel scheduling class to Solaris