Linger-Longer: Fine-Grain Cycle Stealing in Networks of Workstations

Kyung Dong Ryu

University of Maryland
Department of Computer Science

Outline

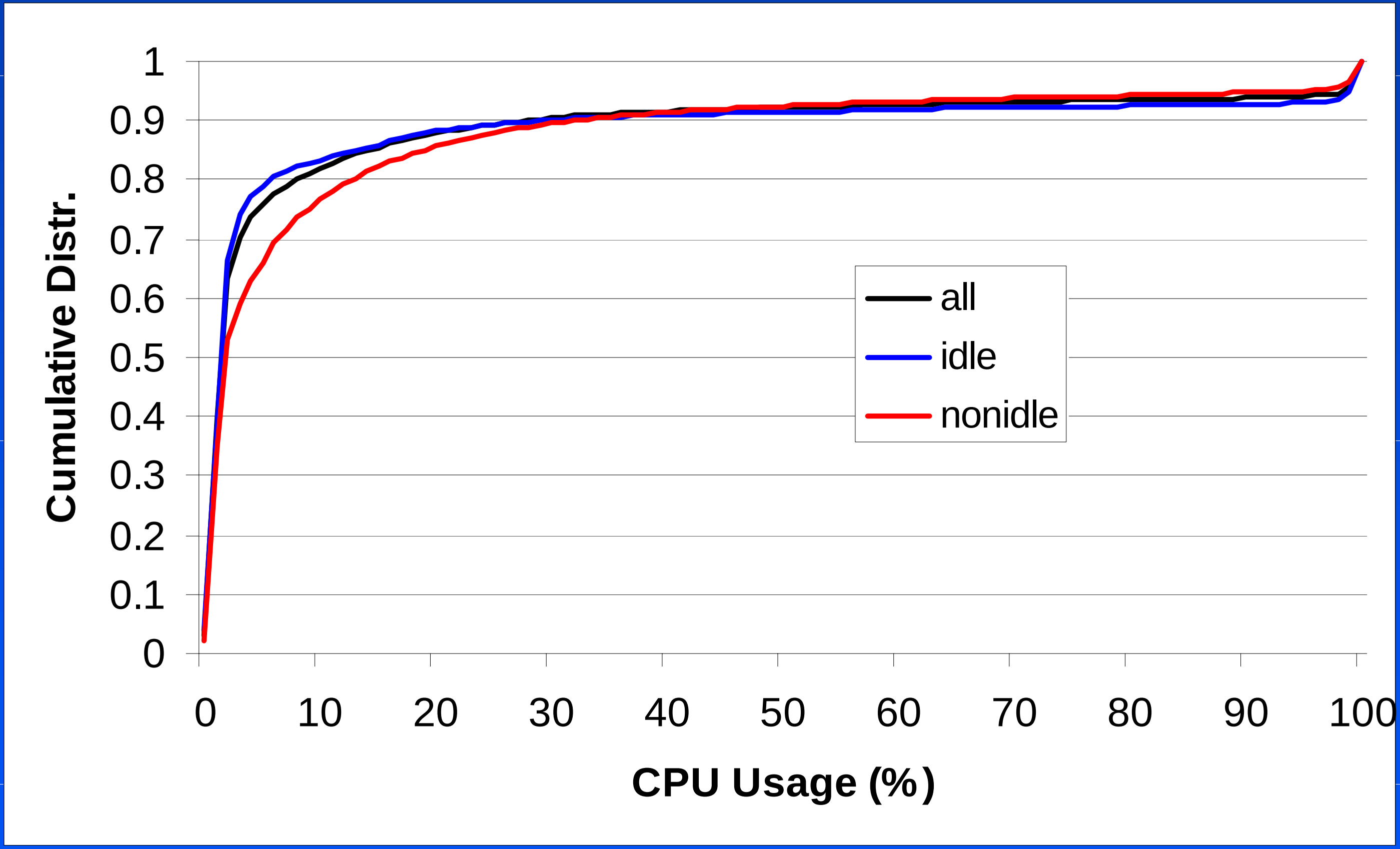
- Introduction & Motivation
- Fine-Grained Cycle Stealing
- Cluster Simulation
- Implementation Mechanisms
- Evaluation
- Conclusion and Future Work

Harvest Idle Cycles

- Networks of Workstations
  - Workstations have high processing power
  - Connected via high speed network (100Mbps+)
  - Long idle time (50-60%) and low resource usage
- Traditional Approach (Coarse-Grained)
  - Goal: Run resource-intensive programs using idle periods
    - while owner is away: send guest job and run
    - when owner returns: stop and migrate job away
  - Examples: Condor and NOW system
- Improvement ?

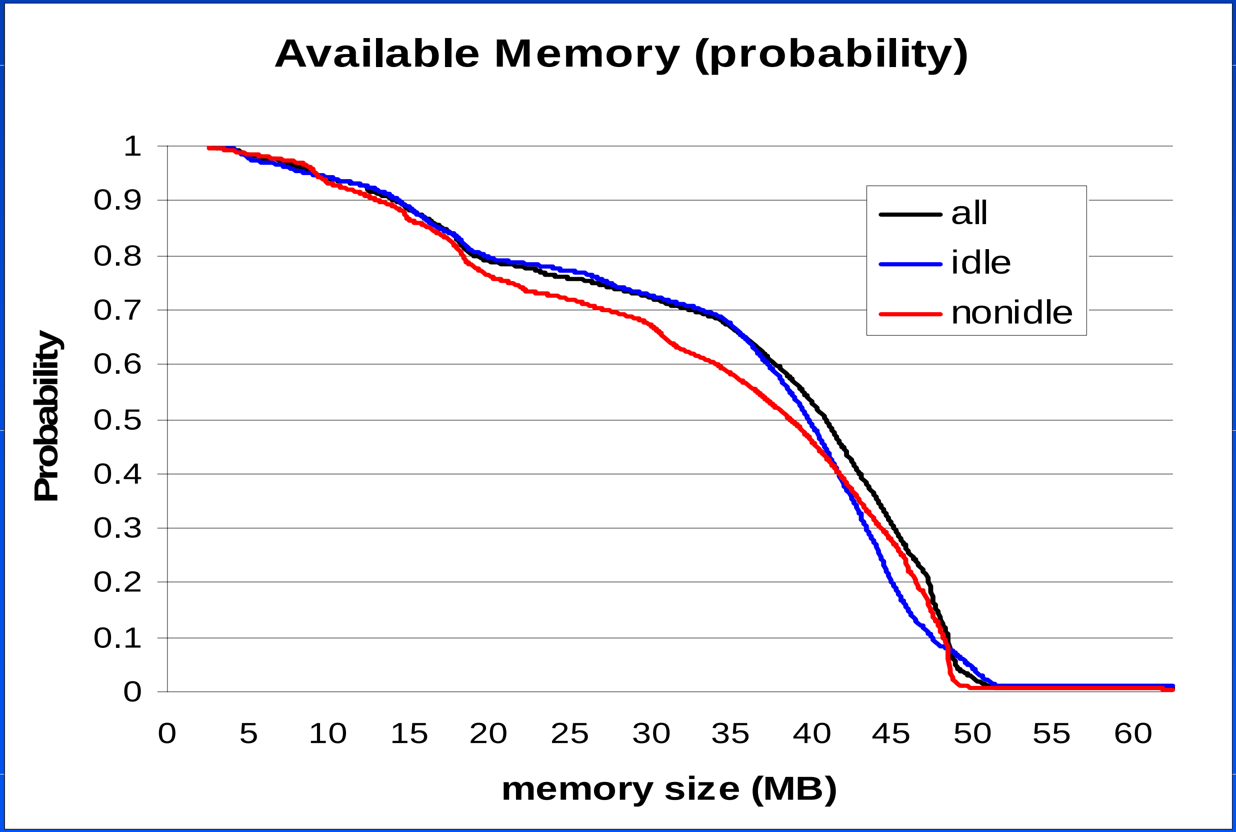
Available Cycles on a "busy" Workstation

- When a user is active
  - CPU usage is low (10% or less for 75% of time)
  - a low priority process could use these cycles



How much Memory is available ?

- 90% of time, at least 14 MBytes is available.

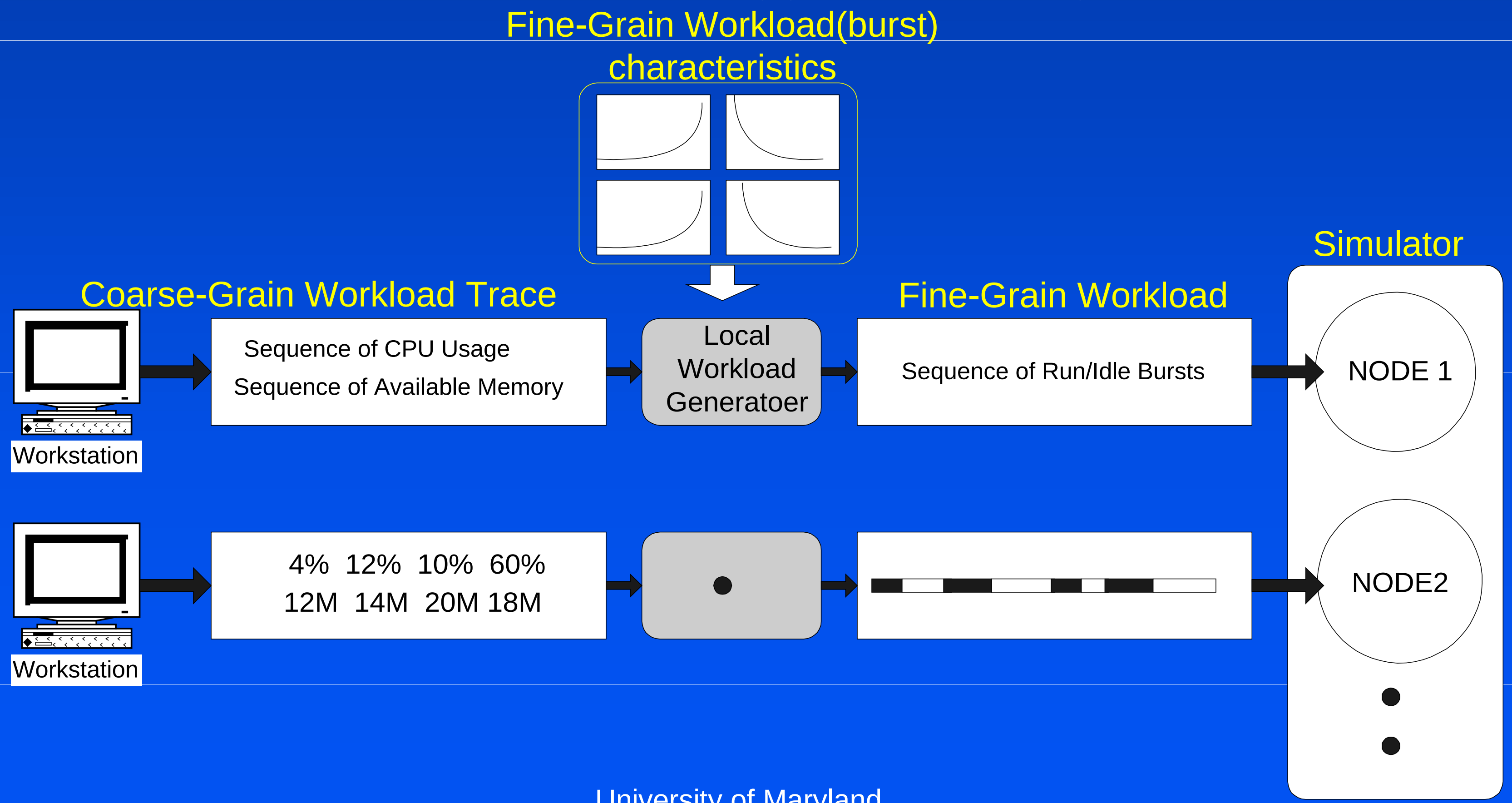


Fine-Grain Cycle Stealing

- Goals:
  - Harvest more available resources using non-idle machines
  - Limit slowdown of local jobs
- Technique
  - Lower guest job's resource priority
    - guest job can *linger* on non-idle node
  - Migration is now optional
    - Migrate guest job to speed up the program
- Local Scheduling
  - Guest job should use only remaining resources
  - Without disruption to machine owner
  - Need Strict Priority Scheduling for resources

Simulation: Generate Local Workload

- Coarse-Grain Usage: Traces (every 2 sec)
- Fine-Grain Run/Idle Bursts: Stochastic model
  - Fine-grain trace data supplies model parameters



Simulation Study: Migration Policies

- Immediate Eviction (IE)
  - when a user returns, migrate the job
  - policy used by Berkeley NOW
  - assumes free workstation or no penalty to stop job
- Pause and Migrate (PM)
  - when a user returns, stop and wait, then migrate the job
  - used by Wisconsin condor
- Linger Longer (LL)
  - when user returns, decrease priority and remain
    - monitor situation to decide when to migrate
  - permits fine grained cycle stealing
- Linger Forever (LF)
  - like Linger Longer, but never migrate

Simulation of Policies

- Model workstation as
  - host process (high priority)
    - requests CPU, then blocks
    - hybrid of trace-based data and model
  - guest process (low priority)
    - always ready to run, and have a fixed total CPU time
  - context switches (each takes 100 micro-seconds)
    - accounts for both direct state and cache re-load
- Study:
  - What is the benefit of Lingering?
  - How much will lingering slow foreground processes?

Cluster Performance: Sequential Jobs

- Metrics
  - Average Job Time: mean time of jobs from submit to exit
  - Variation: stddev. of job execution time (start exec to finish)
  - Family Time: completion time of the last job in the family of processes
  - Throughput: mean of CPU time used by foreign jobs when the number of jobs in system is held constant
- Guest Workload
  - workload 1 : 128 jobs, 600 sec CPU time each
  - workload 2 : 16 jobs, 1800 sec CPU time each
  - All jobs submitted at once.

Cluster Performance: Sequential Jobs

- Results

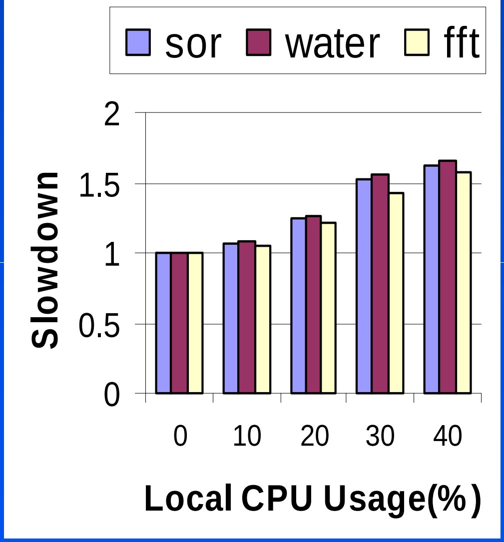
(Simulated Cluster: 64 workstations, 10Mbps Network, 100us Context Switch Time)

	Metric	LL	LF	IE	PM
Workload-1 (128 bg jobs 600 sec)	Avg. Job Time	1044	1026	1531	1531
	Variation	13.7%	20.5%	27.7%	22.5%
	Family Time	1847	1844	2616	2521
	Throughput	52.2	55.5	34.6	34.6
Workload-2 (16 bg jobs 1800 sec)	Avg. Job Time	1859	1861	1860	1862
	Variation	0.9%	1.3%	1.3%	1.6%
	Family Time	1896	1925	1925	1956
	Throughput	15.0	14.7	14.5	14.5

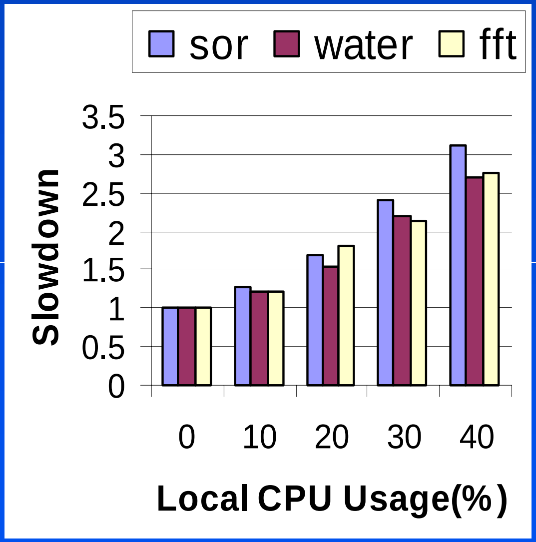
- LF is fastest, but variation is higher than LL
  - Slowest 5% Jobs: LL: 921 sec, LF: 1233 sec
- LF is a 60% improvement over the PM policy
- LL&LF as good as IE or PM in Workload 2
- Slowdown for foreground jobs is under 1%.

Parallel DSM Applications

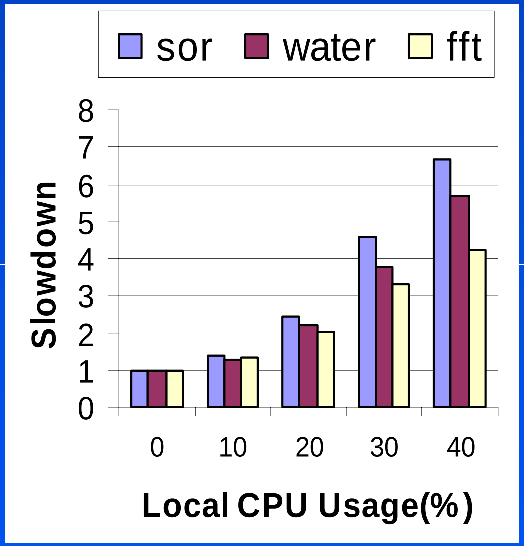
- Simulated three splash benchmark applications
  - Used CVM simulator plus Linger-Longer simulator (LF policy)
  - Varied local load by selecting workload parameters
  - Each application ran on eight nodes



1 non-idle node



4 non-idle nodes



8 non-idle nodes

Outline

- Introduction & Motivation
- Fine-Grained Cycle Stealing
- Cluster Simulation
- Implementation Mechanisms
- Evaluation
- Conclusion and Future Work

Local Scheduling Mechanisms

- Constraints
  - Guest job should use only remaining resources
    - CPU cycles, memory and I/O bandwidth
  - Without disruption to machine owner

- Support Very Low Resource Priority
  - currently available ?
  - How hard to extend kernel ?

Is Unix "nice" sufficient ?

CPU priority is not strict

- run two empty loop processes (guest: nice 19)

OS	Host	Guest
Solaris (SunOS 5.5)	84%	15%
Linux (2.0.32)	91%	8%
OSF1	99%	0%
AIX (4.2)	60%	40%

No memory priority

- run two large memory footprint processes

Policy and Setup	Host time (secs)	Guest time (secs)
Run serially (host then guest)	82	164
Started at the same time, run w/ equal priority	> 5 hours	> 5 hours
Host starts at 0, guest at 10	89	176
Guest starts at 0, host at 10	> 5 hours	> 5 hours

- Answer is NO ! => extend the kernel

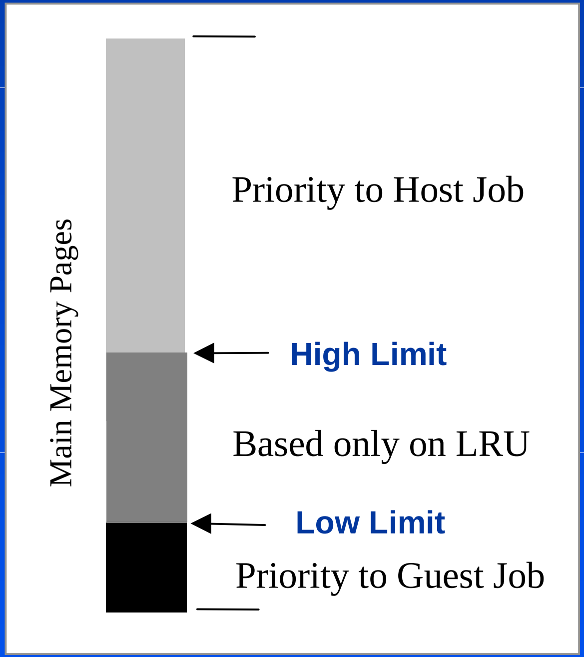
Starvation Level CPU Scheduling

- Original Linux CPU Scheduler
  - Run-time Scheduling Priority
    - nice value & remaining time quanta
    - $T_i = 20 - nice\_level + 1/2 * T_{i-1}$
  - Possible to schedule niced processes

- Modified Linux CPU Scheduler
  - If runnable host processes exist
    - Schedule a host process with highest priority
  - Only when no host process is runnable
    - Schedule a guest process

Prioritized Page Replacement

- New page replacement algorithm



- No limit on taking free pages
- High Limit :
  - Maximum pages guest can hold
- Low Limit :
  - Minimum pages guest can hold

Micro Test (2)

- Large Memory Footprint Job Revisited
  - With Modified CPU Scheduler and Page Replacement

Policy and Setup	Host time (secs)	Guest time (secs)	Host Delay
Host starts then guest,			
Guest niced -19	89	176	8.0%
Linger priority	83	165	0.8%
Guest starts then host			
Guest niced -19	> 5 hours	> 5 hours	> 200
Linger priority	99	255	8.1%

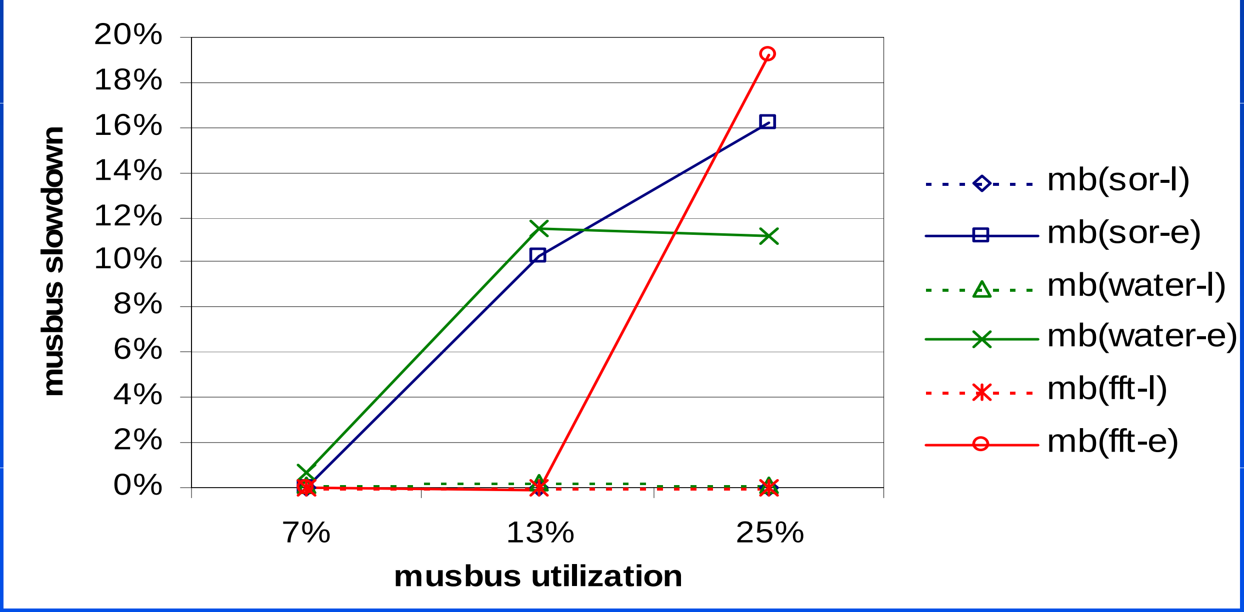
  - Each job takes 82 sec to run in isolation
  - Host-then-guest: Reduce host job delay 8% to 0.8%
  - Guest-then-host: Serialize the execution
    - Switch from guest job to host job
    - guest job resumes after host job finishes

Application Evaluation - Setup

- Experiment Environment
  - Linux PC Cluster
    - 8 pentium II PCs, Linux 2.0.32
    - Connected by a 1.2Gbps Myrinet
- Local Workload for host jobs
  - Emulate Interactive Local User
    - MUSBUS interactive workload benchmark
    - Typical Programming environment
      - Coding, Compiling and Debugging
- Guest jobs
  - Run DSM parallel applications (CVM)
  - SOR, Water and FFT
- Metrics
  - Guest Job Performance, Host Workload Slowdown

Application Evaluation - Host Slowdown

- Run DSM Parallel Applications
  - 3 Host Workloads : 7%, 13%, 24% (CPU Usage)
  - Host Workload Slowdown



  - For Equal Priority:
    - Significant Slowdown
    - Slowdown increases with load
  - No Slowdown with Linger Priority

Conclusions & Future Work

- More Efficient Use of Available Resources
  - 60% increase in throughput
  - Still very limited local job slowdown: a few percent
- Potential for Parallel jobs
  - Acceptable slowdown with Linger Processes
  - Perform better than Reconfiguration in some cases
- Simple kernel modifications improve isolation
  - CPU scheduling: guest process priority
  - Memory Priority: lower/upper limit on guest memory pages
- Current Work
  - I/O & Comm. Prioritization
  - Global (cluster) scheduling policies
  - Integration with Condor
  - Evaluation of prototype implementation