# Failure Prediction in Cycle Sharing Distributed Systems

Saurabh Bagchi
Dependable Computing Systems Lab
School of ECE, Purdue University

**Joint work with: Joanne Ren, Ahmed Amin, Prof. Rudi Eigenmann**

*Relevant papers in:*
*HPDC '06, JOGC '07*

*Work supported by*
*Indiana 21st Century R&T Fund,*
*National Science Foundation,*
*Purdue Research Foundation*



# Greetings come to you from …

## What are Cycle Sharing Distributed Systems?

- Systems with following characteristics
  - Harvests idle cycles of Internet connected hosts
  - Enforces host owners' priority in utilizing resources
  - Resource becomes unavailable whenever owners are "active"
- Popular examples: SETI@Home, protein folding



## What are Fine-Grained Cycle Sharing Systems?

- Cycle Sharing systems with following characteristics
  - Allows foreign jobs to coexist on a machine with local ("submitted by owner") jobs
  - Resource becomes unavailable if slowdown of local jobs is observable
  - Resource becomes unavailable if machine fails or is intentionally removed from the network

**Fine-Grained Cycle Sharing: FGCS**



## Trouble in "FGCS Land"

- Uncertainty of execution environment to remote jobs
- Result of fluctuating resource availability
  - Resource contention and revocation by machine owner
  - Software-hardware faults
  - Abrupt removal of machine from network
- Resource unavailability is not rare
  - More than 450 occurrences per machine in traces collected during 4 months on 20 machines



## How to handle fluctuating resource availability?

- Reactive Approach
  - Do nothing till the failure happens
  - Restart the job on a different machine in the cluster
- Proactive Approach
  - Predict when resource will become unavailable
  - Migrate job prior to failure and restart on different machine, possibly from checkpoint
- Advantage of proactive approach: Completion time of job is shorter

IF, prediction can be done accurately and efficiently



## Our Contributions

*Prediction of Resource Availability in FGCS*

- Multi-state availability model
  - Integrates general system failures with domain-specific resource behavior in FGCS
- Prediction using a semi-Markov Process model
  - Accurate, fast, and robust
- Neural network model for prediction of software failures

Integration of failure prediction in scheduler for production FGCS

- Reduction in application completion time



## Outline

- Multi-State Availability Model
  - Different classes of unavailability
  - Methods to detect unavailability
- Prediction Algorithm
  - Semi-Markov Process model
- Evaluation Results
  - Computational cost
  - Prediction accuracy
  - Robustness to irregular history data
- Neural Network based Prediction
- Evaluation Results
- Results of Failure Prediction in a Scheduler



## Two Types of Resource Unavailability

- **Target applications** – Long running, either CPU or memory intensive, batch mode
- ***UEC*** – Unavailability due to Excessive Resource Contention
  - Resource contention among one ***guest*** job and ***host*** jobs (CPU and memory)
  - Policy to handle resource contention: Host jobs are sacrosanct
    - Decrease the guest job's priority if host jobs incur *noticeable slowdown*
    - Terminate the guest job if slowdown still persists
- ***URR*** – Unavailability due to Resource Revocation
  - Machine owner's intentional leave
  - Software-hardware failures



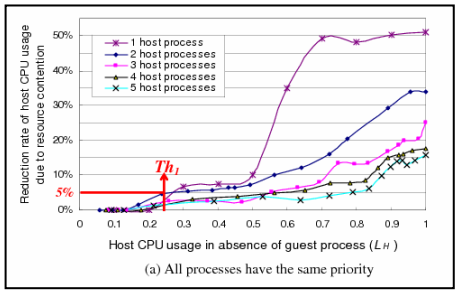
## Studies on Resource Contention

- Detecting UEC requires quantification of noticeable slowdown of host jobs
  - *Noticeable slowdown* of host jobs cannot be measured directly
- Our detection method
  - Threshold for acceptable slowdown in host CPU usage is 5%
  - Empirically find the correlation between observed machine CPU usage and effect on host job due to contention from the guest job

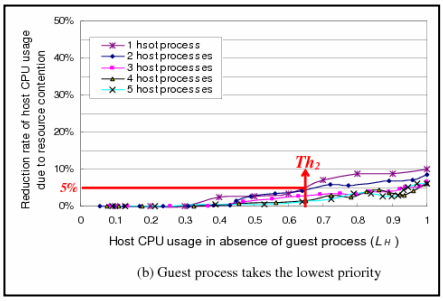


# Studies on Resource Contention: CPU

- Experiment settings
  - CPU-intensive guest job
  - *Host group*: Multiple host jobs with different CPU usages
  - Measure CPU reduction of host group for different sizes of host group



(a) All processes have the same priority



(b) Guest process takes the lowest priority



# Studies on Resource Contention: CPU

- CPU contention study shows the existence of two thresholds $Th_1$ and $Th_2$ in machine CPU usage
- UEC can be detected by observing machine CPU usage on Linux systems


- Other candidate policies for regulating guest job
  - Vary priority with a finer granularity: Experiments show no significant difference for $L_H$ in (20%, 50%)
  - Always run guest job with lowest priority: Too conservative and may incur unacceptable slowdown

## Studies on Resource Contention: CPU & Memory

- Evaluate larger applications that have significant CPU and memory utilization
- Experiment settings
  - Guest applications: SPEC CPU2000 benchmark suite
  - Host workload: Musbus Unix benchmark suite
  - 300 MHz Solaris Unix machine with 384 MB physical memory
  - Measure host CPU reduction by running a guest application together with a set of host workload
- Observations
  - Memory thrashing happens when jobs desire more memory than the system has
  - The memory thrashing persists even if guest job priority is reduced
  - Thus CPU and memory contention can be considered independent



## Resource Unavailability due to Software Failure

- **Assumption**: Machines experiencing high and changing loads have higher probability of failure
  - Example: Continual increase in memory utilization or a constant increase in swap cache size due to memory leak
- **Approach**:
  - Use a stateful model for the failures in the system
  - Compute the model parameters offline based on empirical data
  - Monitor some system state parameters at runtime
  - Predict likelihood of failure based on these parameters fitted into the model

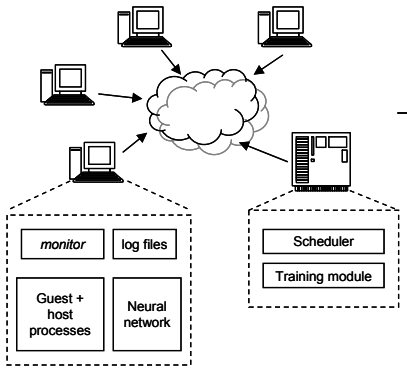


# Stateful Model for Software Failures

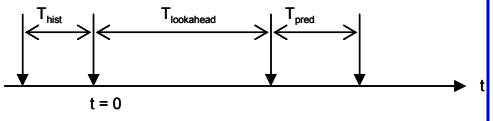
- Requirements:
  - Monitoring process should be lightweight and require no administrative privilege
  - The prediction should be efficient
  - The prediction should be incremental
  - The prediction should not rely on large amounts of log data
- Candidates: Bayesian networks, Neural networks, etc.



# Neural Network Based Prediction




System Model

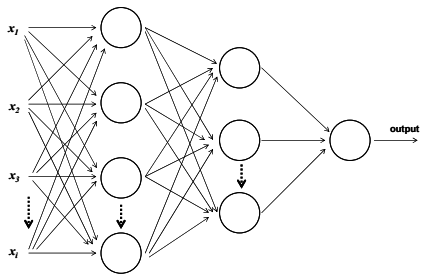


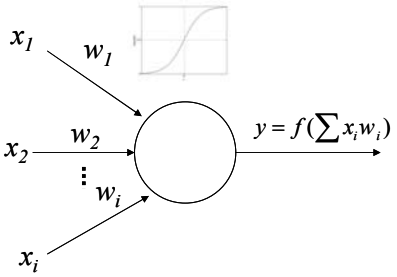

$T_{hist}$ – Time over which historical state information is provided

$T_{lookahead}$ – Time in the future when the prediction is needed

$T_{pred}$ – Time over which the prediction is done

# Neural Network Internals




- Two phases:
  - Training phase: Adjust the weights of the neural network using logged data (offline)
  - Prediction phase: Feed input parameters and get predicted output values (runtime)
- Input parameters at runtime:
  - Categorized into CPU, memory, I/O, and overall system state
  - Set of parameters representing current machine state and some previous time stamps
  - Capturing trends should be more indicative of failure than single anomaly



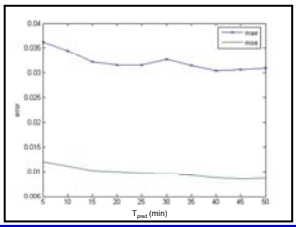
# Performance of NN prediction

$err_i = | predicted_i - actual_i |$ *Mean square error (mse) =* $\frac{1}{n}\sum_{i=1}^{n} err_i^2$

$$y(t) = \frac{1}{T_{pred}} \sum_{j=t}^{j=t+T_{pred}} x(j)$$

Where, *y(t)* is the probability of failure at time *t* (calculated from the empirical data and fed into the ANN during the training phase)

and *x(j)* is 1 if there is a failure at time *j* and 0 otherwise.

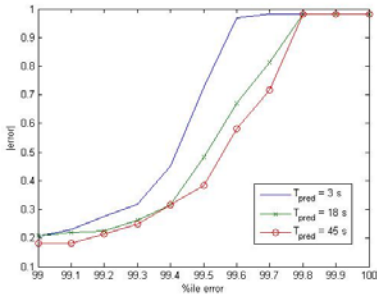


That doesn't look quite right, does it?

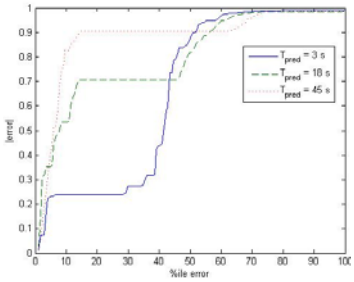
# Performance of NN Prediction

$n^{th}$ error percentile defined by:

If the $n$th percentile error is $\alpha$, it means $n$% of the errors have a value less than $\alpha$.



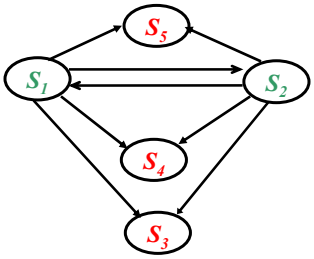

Percentile errors for $T_{lookahead} = T_{hist} = 0$




Percentile errors for a failure streak of 537 failures, $T_{hist}$ = 12 sec



# Multi-State Resource Availability Model




$S_1$: Machine CPU load is [0%, $Th_1$]
$S_2$: Machine CPU load is ($Th_1$, $Th_2$]
$S_3$: Machine CPU load is ($Th_2$ ,100%] -- UEC
$S_4$: Memory thrashing -- UEC
$S_5$: Machine unavailability -- URR

For guest jobs, $S_3$, $S_4$, and $S_5$ are unrecoverable failure states

## Resource Availability Prediction

- Goal of Prediction
  - Predict temporal reliability (TR)

    *The probability that resource will be available throughout a future time window*
- Semi-Markov Process (SMP)
  - States and transitions between states
  - Probability of transition to next state depends only on current state and amount of time spent in current state (independent of history)
- Algorithm for TR calculation:
  - Construct an SMP model from history data for the same time windows on previous days

    *Daily patterns of host workloads are comparable among recent days*
  - Compute TR for the predicted time window



## Why SMP?

- Applicability – fits the multi-state failure model
  - Bayesian Network models
- Efficiency – needs no training or model fitting
- Accuracy – can leverage patterns of host workloads
  - Rules out: Last-value prediction
- Robustness – can accommodate noises in history data

## Where does the NN model fit in?

- Computation of transition to state $S_5$ is done using the neural network (NN) model
- Operates concurrently with the SMP computation



# Background on SMP

- Probabilistic Models for Analyzing Dynamic Systems

  $S$ : state

  $Q$ : transition probability matrix

  $Q_i(j)$ = Pr {the process that has entered $S_i$ will enter $S_j$ on its next transition};

  $H$ : holding time mass function matrix

  $H_{i,j}(m)$ = Pr {the process that has entered $S_i$ remains at $S_i$ for m time units before the next transition to $S_j$}

- Interval Transition Probabilities, $P$

  $P_{i,j}(m) = \Pr\{S(t_0+m)=j \mid S(t_0)=i\}$



# Solving Interval Transition Probabilities

- Continuous-time SMP
  - Backward Kolmogorov integral equations

$$P_{i,j}(m) = \sum_{k \in S} \int_0^m Q_i(l) \cdot H_{i,k}(\cdot) \cdot P_{k,j}(m-u)du$$

Too inefficient for online prediction

- Discrete-time SMP
  - Recursive equations

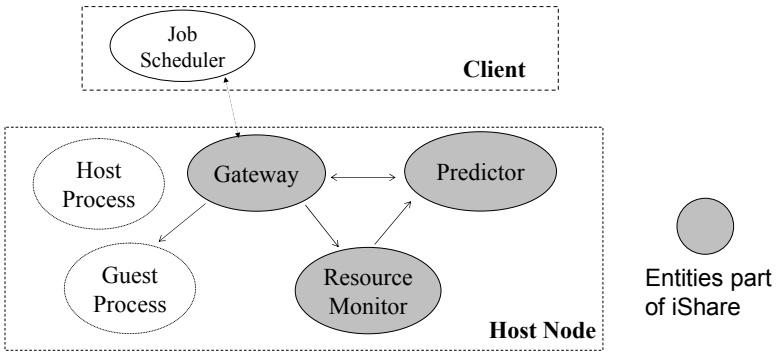
$$P_{i,j}(m) = \sum_{l=1}^{m-1} P_{i,k}^{1}(l) \times P_{k,j}(m-l) = \sum_{l=1}^{m-1} \sum_{k \in S} Q_i(k) \times H_{i,k}(l) \times P_{k,j}(m-l)$$

- Availability Prediction

  TR(W): the probability of not transferring to $S_3$, $S_4$, or $S_5$ within an arbitrary time window, W of size $T$

$$TR(W) = 1 - [P_{init,3}(T/d) + P_{init,4}(T/d) + P_{init,5}(T/d)]$$

# System Implementation in iShare




Non-intrusive monitoring of resource availability

- UEC – use lightweight system utilities to measure CPU and memory load of host processes in non-privileged mode
- URR – use lightweight utilities for measurement of system state parameters, with limited history, fed to NN



# Evaluation of Availability Prediction

- Testbed
  - A collect of 1.7 GHz Redhat Linux machines in a student computer lab at Purdue
    - Reflect the multi-state availability model
    - Contain highly diverse host workloads
  - 1800 machine-days of traces measured in 3 months (09/06-11/06)
- Statistics on Resource Unavailability

| Categories | Total amount | UEC | | URR |
|---|---|---|---|---|
| | | CPU contention | Memory contention | |
| Frequency | 405-453 | 283-356 | 83-121 | 3-12 |
| Percentage | 100% | 69-79% | 19-30% | 0-3% |

# Evaluation Approach

- Metrics
  - Overhead: monitoring and prediction
  - Accuracy
  - Robustness
- Approach
  - Divide the collected trace into training and test data sets
  - Parameters of SMP are learnt based on training data
  - Evaluate the accuracy by comparing the prediction results for test data
  - Evaluate the robustness by inserting noise into training data set



# Reference Algorithms: Linear Time Series Models

- Widely used for CPU load prediction in Grids: Network Weather Service*
- Linear regression equations**
- Application in our availability prediction
  - Predict future system states after observing training set
  - Compare the observed TR on the predicted and measured test sets

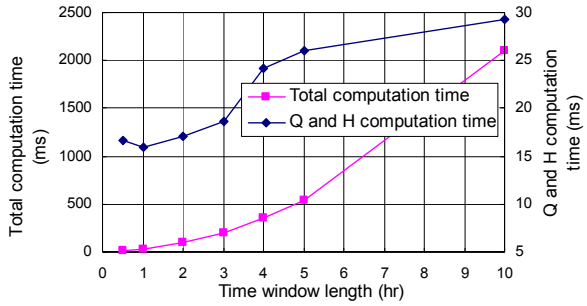
*R. Wolski, N. Spring, and J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *JFGCS*, 1999

** Toolset from P. A. Dinda and D. R. O'Halaron. "An evaluation of linear models for host load prediction". In Proc. Of HPDC'99.



# Overhead

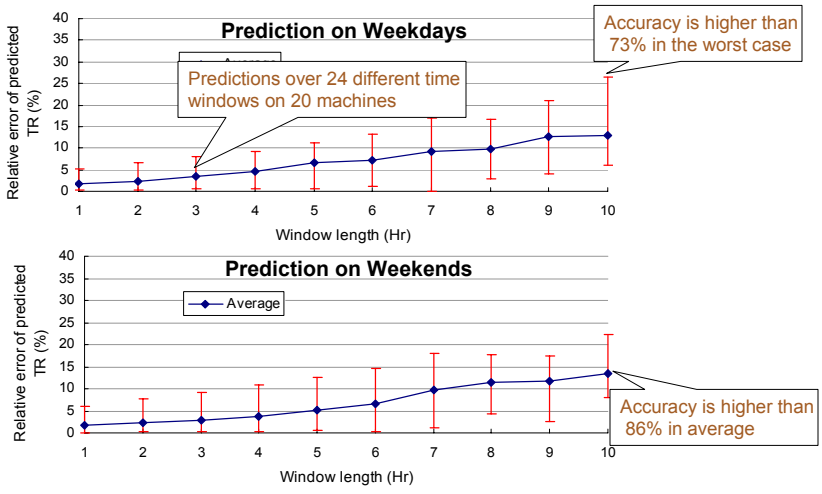
- Resource Monitoring Overhead: CPU 1%, Memory 1%
- Prediction Overhead




*Less than 0.006% overhead to a remote job*

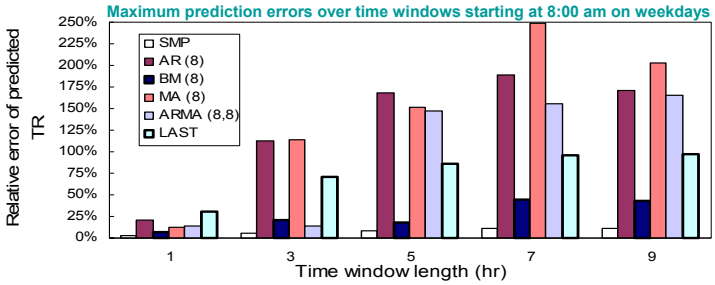


# Prediction Accuracy




*Relative error = abs($TR_{predicted}$-$TR_{empirical}$)/$TR_{empirical}$*

# Comparison with Linear Time Series Models



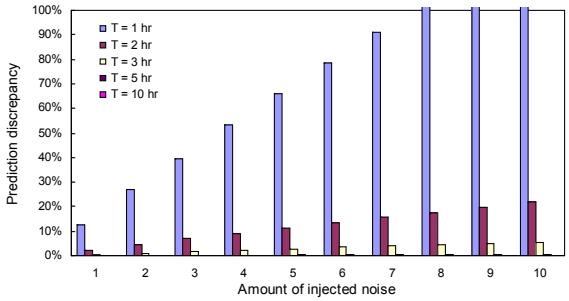

Resource Prediction System: http://www.cs.cmu.edu/~cmcl/remulac/remos.html

| Model | Description |
|---|---|
| AR(p) | Purely autoregressive models with p coefficients |
| BM(p) | Mean over the previous N values (N < p) |
| MA(p) | Moving average models with p coefficients |
| ARMA(p,q) | Autoregressive moving average models with p+q coefficients |
| LAST | Last measured values |



# Prediction Robustness

Randomly insert unavailability occurrences between 8:00-9:00 am on a weekday trace




1) Predictions on smaller time windows are more sensitive
2) On large time windows (> 2 hours), intensive noise (10 occ urrences within one hour) causes less than 6% disturbance in the prediction

## iShare Scheduler

- Compare three schedulers
  - iShare which uses failure prediction
  - Condor which does matchmaking but is oblivious to failures
  - Idealized omniscient algorithm
- How to integrate failure prediction

Estimated Job Completion Time (JCT) without failures = $\frac{TL}{CR \times [1 - L_{t_0}(TL)]}$

MTTF = $\int_0^\infty TR(t)dt$

Effective Task Length (ETL) = MTTF $\times$ CR $\times$ (1-$L_{t_0}$(MTTF))



## iShare Scheduler

- Selection algorithm for iShare scheduler

1. Compare MTTF with JCT.
2. If JCT > MTTF for each machine, select the one with largest ETL.
3. If JCT < MTTF, select the one with minimum job completion time considering failures (JCTF)

$$TL = CR \times (1 - L_{t_0}(JCTF)) \times \int_0^{JCTF} TR(t)dt$$

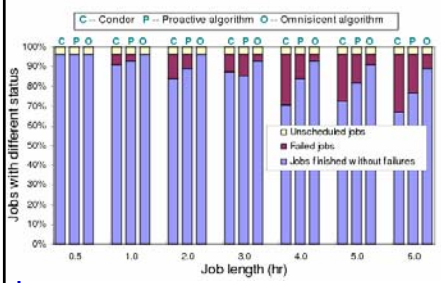


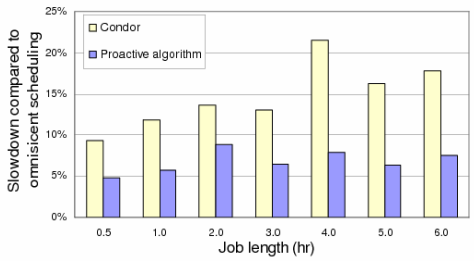
# Experiments for Schedulers

- Experiments performed to compare three scheduling algorithms
  - Using GridSim but using availability traces from testbed
  - Jobs submitted with submission times between 6 am and 10 pm, 7 different lengths from 0.5 to 6 hours
- Experiment Metrics
  - When a job is submitted, if no resource is available, then wait for 5 minutes and retry; if still no resource available, then *unscheduled*
  - A scheduled job either returns as *finished*, or if resource becomes unavailable then *failed*
  - A failed job is restarted until it completes successfully



# Results from Scheduler Experiments




- Fraction of unscheduled jobs same
- Difference between proactive and omniscient algorithms increase with job length
- For large jobs, fraction of failures lower for iShare than Condor
- Non monotonicity at 3 hours

- Omniscient algorithm has no overhead for prediction
- Two sources of slowdown – prediction overhead, ineffectiveness of prediction
- Effect of failure and restart adversely affects Condor
- Improvement in makespan 4-14%

## Summary on Related Work

- Fine-grained cycle sharing with OS kernel modification Ryu and Hollingsworth, *TPDS*, 2004
- Critical event prediction in large-scale clusters Sahoo, et. al., *ACM SIGKDD*, 2003
- CPU load prediction for distributed compute resources Wolski, et. al., *Cluster Computing*, 2000
- Studies on CPU availability in desktop Grid systems Kondo, et. al., *IPDPS*, 2004



## Conclusion

- For practical FGCS systems, runtime prediction of resource unavailability is important
- Resource unavailability may occur due to resource contention or resource revocation
- Prediction system based on an SMP and a NN model is
  - Fast: < 0.006% overhead
  - Accurate: > 86% accuracy in average
  - Robust: < 6% difference caused by noise
- Prediction system helps a runtime scheduler
- Generality
  - Testbed contains highly diverse host workloads
  - Accuracy was tested on workloads for different time windows on weekdays/weekends



# Thanks!



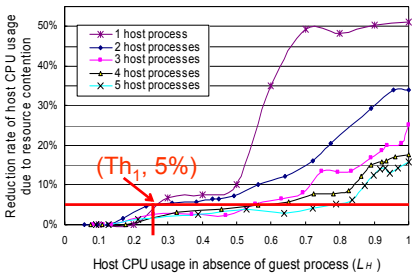
# Backup Slides

- Resource contention studies, 27-29
- Linux scheduler, 30
- Details on reference algorithms for failure prediction, 31

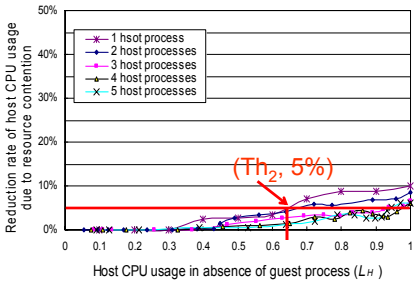


# Empirical Studies on Resource Contention

- CPU Contention
  - CPU-intensive guest applications
  - host groups consisting of multiple processes with diverse CPU usage
  - 1.7 GHz Redhat Linux machine



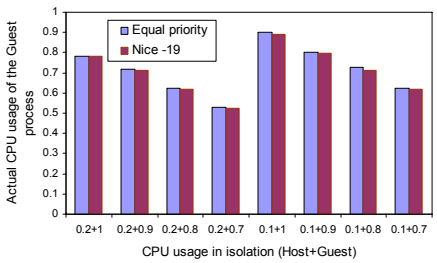

All processes have the same priority



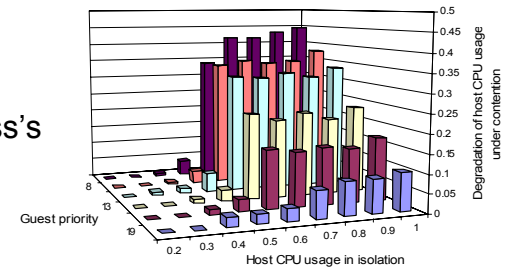

Guest process takes the lowest priority

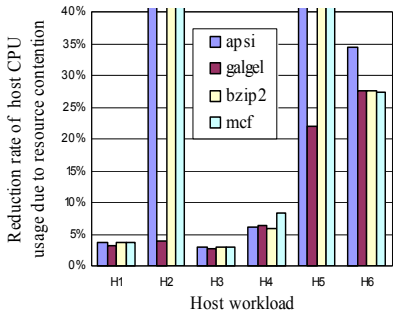


Restrict resource contention by minimizing guest process's priority from its creation

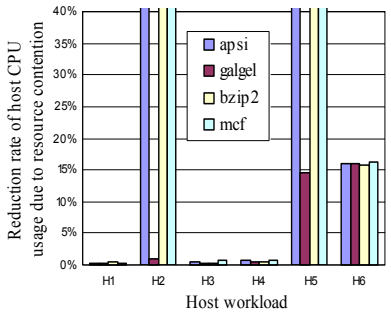



Restrict resource contention by finely tuning guest process's priority






Guest process with priority 0




Guest process with priority 19

- Memory thrashing happens when processes desire more memory than the system has
- Impacts of CPU and memory contention can be isolated
- The two thresholds, $Th_1$ and $Th_2$, can still be applied to quantify CPU contention



```
While (1) {
  If exists p such that p.state = RUNNABLE
    Foreach process p
      p.quanta = 20 + p.niceLevel + 1/2 * p.quanta;
  While exists a process p
    such that (p.state = RUNNABLE) and (p.quanta > 0)
      Select p with largest p.quanta;
      Decrement p.quanta;
      Run p;
}
```

Linux CPU scheduler



# Details on Reference Algorithms

- AR(p) – An autoregressive model is simply a linear regression of the current value of the series against one or more prior values of the series. p is the order of the AR model. Linear least squares techniques (Yule-Walker) are used for model fitting.
- BM(p) – Average on previous N values. N is chosen to minimize the squared error
- MA(p) - A moving average model is conceptually a linear regression of the current value of the series against the white noise or random shocks of one or more prior values of the series. Iterative non-linear fitting procedures (Powell's methods) need to be used in place of linear least squares.
- ARMA(p,q) - a model based on both previous outputs and their white noise
- LAST – the previous observations from the last time window of the same length are used for prediction