Motivation: Trend #1
Emergence/acceptance of VM abstraction
- VMware, MS, UML, IBM Virtual Hosting services (circa '05)
- Envisioned/Used mostly in closed, managed environments

Motivation: Trend #2
Apps running on shared 3rd party hosts
- PlanetLab and Emulab experimental testbeds
- IBM, Akamai, Speedera edge-computing & hosting services

Motivation: Trend #3
Need to isolate independent constituents
- Virtual web hosting; e.g., Mozilla Application VM (circa '01)
- PlanetLab's use of VMs for various services on a single host
- Shared infrastructures; e.g., Grids, Sensoria, overlays

VM Isolation
- Memory isolation:
  "VMs provide robust, secure isolation between host and guest memory address spaces."

- Failure isolation:
  "VM isolation ensures that if one VM crashes or hangs, it can’t impact other VMs or the host."

- Performance/QoS isolation:
  "Need to ensure that a VM does not monopolize host resources to the detriment of other VMs’ performance/QoS."
  Subject of this talk

Shared Host Resources: Issues
- Under-provisioned Host ➔ Overload
  - Inefficient use of host resources
  - Unpredictability due to OS thrash mitigation measures
  - Unfair/uninformed allocation of resources to applications
Resource Allocation: How?

- OS or VMM micromanages access to resources
  - Adds complexity to common infrastructure
  - Agnostic to application adaptation
  - Special APIs not suitable for open systems
  - Not easily extensible to multiple resources
- Reservation based allocation
  - Inefficient, especially with highly dynamic applications
  - Incompatible with inherently "best-effort" resources
  - Hosting infrastructure must police applications
- Best-effort allocation with overload protection
  - Simple common infrastructure
  - Applications must "live with" allocated resources
  - Not fair across disparate apps; not predictable

Resource Allocation: Wish List

- Simple hosting infrastructure
  - Macro, not micro-management; OK to monitor, police, ...
- Application autonomy
  - No explicit coordination between applications or with host
- Performance isolation
  - Applications with different bottlenecks need not tussle
- Convergence to fairness
  - System should converge to a fair allocation of resources
- Efficient resource utilization
  - No overload; no underutilization

Our Solution: E2E-style

- Minimal Host Functionality
  - Best-effort, round-robin-style resource allocation
- Desirable Host Functionality
  - Provide "congestion" feedback signal to apps
  - Implement policing of non-compliant apps
- Adaptive Resource Consumption by Apps
  - Probe available resources and react to congestion
  - Adaptation mechanisms may vary to suit apps
  - Compliance, or friendliness is well defined

An Instance of Host Sharing

- VMs as applications
  - VMs used to provide isolation, namely safety and security
  - Hosts are powerful enough to support many VMs
  - VMs compete for host resources and may exhibit radically different resource needs (e.g., memory-bound, CPU-bound, I/O-bound, net-bound, ...)

Our E2E Solution: Friendly VMs

- VMs adapt their resource consumption rate based on congestion feedback signal

Benefits:
- Minimal resource management in host OS/VMM
- Friendly (efficient and fair) sharing of system resources among VMs
- Transparent to the application on top as well as the OS hosting the VMs below

Elements of the solution:
1. What constitutes the feedback signal?
2. How to control consumption rate?
3. What adaptation strategy is appropriate?
FVM: (1) Feedback Signal

- **Virtual Clock Time (VCT)**
  - VCT is the time interval between two consecutive virtual clock ticks (of the VM)
  - VCT is the VM response time; it is analogous to the RTT of a network flow
  - Use VCT (or derivative thereof) to generate feedback signal

VCT: Illustration #1
VCT: A Barometer of Load

- VCT grows with load (e.g., #VMs)
  - Slow growth ~ Linear → Efficient
  - Superlinear → Thrashing → Inefficient

FVM: Congestion Signal

\[ VCT(t) = \min(VCT_{\text{min}}(f^i), \{VCT^*(k) : k = 1, \ldots, n\}) \]

Slowdown:
\[ \rho = \frac{VCT(t)}{VCT^*(t)} \]

R > Threshold (H) → Congestion = True
where \( H = \frac{VCT^*}{VCT_{\text{min}}} \)
FVM: (2) Consumption Control

- **Multi-Programming Level (MPL) Control:**
  - A thread as a unit of consumption of host resources; VM is a multi-threaded application
  - # of active threads allowed for a VM constitute a cap on its resource consumption
  - Adjust # of active threads through suspension or resumption of threads within a VM

- **Rate Control:**
  - Force VM to periodically sleep (or timeout)

FVM: MPL Control

FVM: Rate (timeout) Control

FVM: (3) Adaptation Strategy

- **AIMD (Additive-increase/Multiplicative-decrease)**
  - Adjust # of threads
    - No Congestion: \( \text{thread}_{\text{max}} = \text{thread}_{\text{max}} + a \)
    - Congestion: \( \text{thread}_{\text{max}} = \frac{\text{thread}_{\text{max}}}{b} \)
  - Adjust execution rate (timeout period)
    - No Congestion: \( \text{rate} = \text{rate} + a \)
    - Congestion: \( \text{rate} = \frac{\text{rate}}{b} \)

- Different increase/decrease rules that match application requirements (e.g., smoother adaptation) could co-exist as long as they are “compatible” [JinGuoBestavrosMatta’02]

Host: Requirements

- **Required:**
  - Unbiased On-Demand Allocation
  - RR scheduler

- **Desirable:**
  - Policing Functionality (friendliness incentive)
    - Identify/penalize misbehaving applications
### FVM: Analytic Model

**FVM: Analytic Model Results**

Linearized model allows us to:
- Relate convergence & transient characteristics to parameters, e.g., AIMD/EWMA constants, various delays, gain, ...
- Sketch adaptation transients numerically

### FVM: Convergence

- Congestion signal constitutes prices fed back to VMs as the load on the host varies
- Convergence and stability can be proved through Lyapunov function [Kelly’99]

### f-UML: A FVM Prototype

- Based on User Mode Linux (UML) VM
  - UML is a VM abstraction that allows guest Linux systems to run at user-level on top of a Linux host
- Added ~ 500 lines to UML code
  - VCT measurement, congestion signal generation, and controller implemented in a single function `fvm_adapt()` which is added to the `time_handler()` for SIGALRM/SIGVALRM

### f-UML: Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Period</td>
<td>5 sec</td>
</tr>
<tr>
<td>Window of $VCT_{\text{min}}$</td>
<td>60 sec</td>
</tr>
<tr>
<td>EWMA constant for $VCT$</td>
<td>0.3</td>
</tr>
<tr>
<td>Initial limit on the number of thread</td>
<td>10</td>
</tr>
<tr>
<td>Slowdown threshold</td>
<td>2.5</td>
</tr>
<tr>
<td>AIMD additive constant (MPL control)</td>
<td>1 thread</td>
</tr>
<tr>
<td>AIMD additive constant (rate control)</td>
<td>0.1 Hz (1/T_s)</td>
</tr>
<tr>
<td>AIMD multiplicative constant</td>
<td>1.5</td>
</tr>
</tbody>
</table>

### f-UML: Evaluation

- **Web server experiments**
  - 4 VMs on host, with Apache 2.0 running on each VM
  - Client requests invoke memory-bound CGI scripts
- **VMs tested**
  - Original UML
  - f-UML prototype (with MPL control)
- **Metrics (per VM & averaged over 4 VMs)**
  - VCT
  - Throughput
  - Fairness Index
- UML vs UML: Baseline Results

- UML vs UML: Throughput

- UML vs UML: Fairness

- Friendly wrappers:
  - Could an application be made friendly post-mortem?
  - Could friendliness be "strongly typed"?
  - What is the role of compilers in casting friendliness?

- FVM: Food for Thought

- VM resource consumption throttling:
  - What if all threads are not created equal?
  - Which thread should be suspended?
  - What knobs could be exposed to programmers?

- FVM framework extensions:
  - Other passive feedback signals?
  - Active inference of congestion? In-lined benchmarks?
  - Other adaptation rules? PI, PID, SIMD, …?
  - Extension to friendliness over host clusters, grids, …

- FVM: Food for Thought

- Friendly wrappers:

- FVM: Food for Thought

- Friendly Virtual Machines
  Leveraging a Feedback-Control Model for Application Adaptation

- Intermission

VMware Inc.
June 29th, 2006
From \( f \)-UML to \( f \)-VMware VMs?

Current \( f \)-UML prototype:
- Paravirtualized VMs
- Requires guest OS modification
- Assumes threads are homogenous
- Works only on single physical host
- ...

Requirements for use in VMware VMs
- Full virtualized VMs
- No Guest OS modification
- Threads are heterogeneous
- Cluster

VCT Measurement

Recall our notion of VCT:
- Time interval between two consecutive virtual clock ticks (of the VM), a.k.a., \( VM \) descheduled time

\( f \)-UML (paravirtualized VMs):
- Time between consecutive invocations of the \( timer \_ handle() \) for SIGALRM/SIGVALRM virtual timer interrupts

\( f \)-VMware VMs:
- Use VMware Timer Sponge – a VMware tool that measures VM descheduled time

VM Descheduled Timing Issues

- Timer interrupts are only delivered when VM is scheduled
- When VM is descheduled, it may “fall behind” and accumulate a backlog of timer interrupts
- If the backlog of timer interrupts is not delivered properly, then
  - Guest timing will not be very accurate
  - Guest statistical accounting process will be distorted
- How do we deal with the backlog of timer interrupts?

Solution: Timer Sponge

A per-VCPU “timer sponge” process (“\( VmDesched \)”) is introduced within the guest OS to explicitly represent the time that VM was descheduled
- Normally sleeps
- Wakes up when there is a backlog of pending interrupts
- Delivers all backlog while the sponge process is running

- Timekeeping: all timer interrupts get delivered
- Process accounting:
  - All descheduled time contributes to \( vmdesched \) process
  - Guest CPU utilization: total – \( vmdesched \)%
  - Other processes should not be affected

Timer Sponge Implementation

Timer Sponge for FVM

Possible feedback signals for adaptation:
- VCT: \( vmdesched \) time
  - The interval between two consecutive invocation of \( VMDesched \) thread
- The number of accumulated interrupts
  - Time interrupts
  - I/O interrupts
  - ...
- VM load, process load
  - ...
Extensions

Resource Consumption Control:
- Differentiated thread control
  - Threads are heterogeneous
  - Classify threads based on resource consumption or importance
  - Weighted AIMD or other control rules for different classes
  - Suspending or resuming current threads in thread pool
  - Creating or killing threads to adjust thread pool size
- ... 

Other Signals and Control

Resource-Specific Signal and Control:
- CPU
- Memory
- Disk
- Network
  - ...

- Available VMware tools: VMware Balloon driver which provides memory-specific signal and control

VMware Memory Balloon

When memory is overcommitted:
- Need to reclaim memory from one or more VMs
- Guest OS in each VM knows best the information about which pages are least valuable and thus should be reclaimed

VMware Balloon Mechanism [W02]:
- Insert balloon driver in guest to allocate/free physical pages
- Grant reclaimed memory to other VMs
- Native paging algorithms is used in guest OS

Memory Balloon Concept

Inflate balloon (more pressure)
- May page out
- Increase memory demand

Deflate balloon (less pressure)
- May page in
- Decrease memory demand

Feedback Signal
- Idle memory
- ...

FVM via Memory Balloons

Using Memory Balloons was shown to be efficient, but is it fair?

- Could VMs be friendly to one another w.r.t. memory usage by voluntarily adjusting their usage?

- Could memory balloons be combined with other feedback/control signals to build Friendly VMs?
Host Support for FVMs

**Friendly Configuration:**

- Add a “friendly configuration” to standard min, max and share configurations
- Host may impose some policing rules when VM acts unfriendly
  - Dynamically adjust their min or max share
  - Kill VM
- Host Support for FVMs

FVM Support for VM Cluster

- Currently FVM framework provides efficiency and fairness for VMs on the same host
- Can it support friendly VMs over clustered hosts?

FVM Support for VM Cluster

- VMs on different hosts may not be friendly to VMs on other host in the same cluster.
- The reason is that the feedback signals are only from local host.

FVM Support for VM Cluster

- Goal: Enable friendliness over a cluster
- Idea: Aggregate feedback signal from all hosts in the cluster
- Result: Load imbalance across hosts
- Solution: Detect load imbalance and migrate FVM from overloaded host
FVM Support for VM Cluster

- How? Use VMware's Distributed Resource Scheduler (DRS) to rebalance load
- Opportunity! DRS is the natural place for enabling FVMs over clusters


More information available from WING Publications
http://www.cs.bu.edu/groups/wing

Other Related Projects

- Inference of host characteristics
  - Could a VM actively or passively infer the mechanisms employed on the host?
  - Results for inference of effective cache sizes and cache management strategies
- Embedding of richer topologies
  - From virtualization of a VM to virtualization of a network of VMs
  - Results on embedding QoS-constrained interconnections of VMs in a hosting network

Thank you!