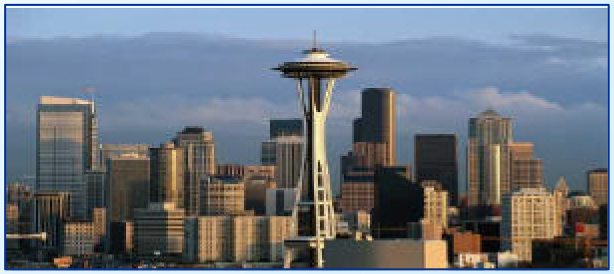
Lightweight Virtual Machines

Steven D. Gribble, Andrew Whitaker

Department of Computer Science and Engineering

University of Washington



{gribble,andrew}@cs.washington.edu

Some context for the next hour

- **This is a new research project starting at UW**
  - high risk, high reward
  - significant implementation complexity, possibly rife with conceptual and design pitfalls
- **This is your chance to have huge impact!**
  - tell us if you believe the story, the approach, etc.
  - help us pick driving applications

©2001, Steven D. Gribble

Research agenda

- **Interesting new set of applications is emerging**
  - they all require lightweight protection domains
    - hundreds per physical machine, rapid context switching
    - complete isolation between the domains
- **Our research goal**
  - to design, build, and evaluate one way of doing this
    - virtual machines
      - think VMware, not JVM

©2001, Steven D. Gribble

Meta-outline

- **Steve Gribble *(the "what")***
  - motivating the applications
  - exploring tradeoffs between methods
  - identifying core challenges with VM's
- **Andrew Whitaker *(the "how")***
  - picking an architecture to virtualize
  - resource management strategies
  - some simple first steps (risk reduction!)

©2001, Steven D. Gribble

Outline

- Introduction
- **Driving applications and their characteristics**
- Argument for virtual machines
- Key challenges

©2001, Steven D. Gribble

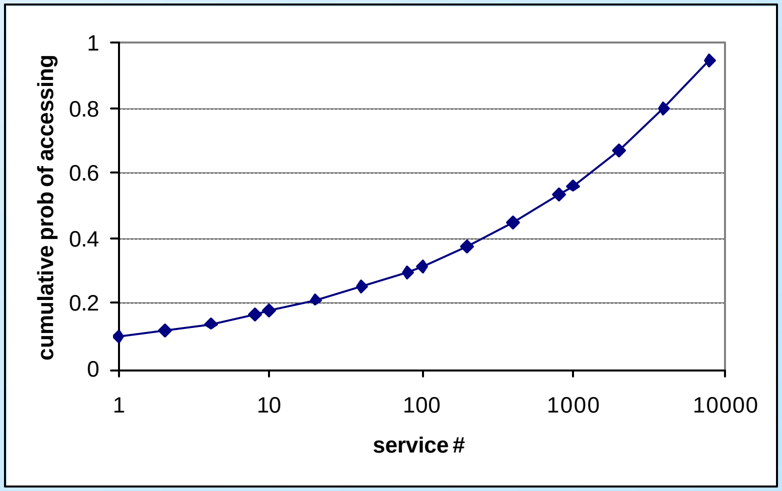
Content delivery: not just static anymore

- **Recent progression of content-delivery architectures**
  - CDNs, proxy caches, P2P, …
    - premise same for all: replicate static content
  - but: large and increasing fraction of content is dynamic
    - 20-40% of web requests are to dynamic content [Wolman99]
    - these systems have or soon will “hit the wall”
- **Need to think about distributing dynamic content!**
  - inject content-generation code into CDNs, caches, …
    - infrastructure must completely distrust this code
    - an isolation and security challenge
      - existing research doesn’t adequately solve isolation problem

©2001, Steven D. Gribble

Content delivery: challenges of scale

- **High degree of concurrency in caches, servers**
  - lessons from web proxy caches
    - hundreds/thousands web pages in hot set
    - O(100) simultaneous requests at any time
- **Driven by Zipfian popularity distributions**
  - 50% of access to 6% sites
  - 20% of accesses to least popular 50% of sites
  - need fast context switching!



©2001, Steven D. Gribble

Pushing Internet services

- **Vision for future applications: the network is computer**
  - requires scalable, available hosting infrastructure
    - also requires software architecture (same reasons)
- **Barrier to deployment of new services is high**
  - cost of physical equipment large
    - >=1 physical machine, rack space, power, admin, etc.
  - stifles grassroots service innovation
- **Ideal: push new services into virtual hosting site**
  - most will be unpopular: must multiplex large number of services
  - same isolation, multiplexing, context switching issues as before

©2001, Steven D. Gribble

Measurement code

- **Measuring the wide-area Internet is interesting**
  - Access, NIMI, etc.: sprinkle machines across WAN
    - researchers share machines for experiments
    - upload measurement, analysis code into machines
  - leads to a dilemma
    - experiments need to run for long periods
    - yet, for isolation, they are currently time-division mux'ed
  - instead: run many experiments concurrently
    - need way of safely mux'ing resources
- **Efficiency is key challenge here**
  - can't perturb/reduce throughput

©2001, Steven D. Gribble

What do these have in common?

- **Host must execute untrusted code**
  - need a watertight protection domain to isolate
- **Large degree of concurrency required**
  - implies protection domains must be lightweight
    - so can run hundreds simultaneously
  - implies fast context switching between domains
    - Zipf: implies swapping domains in/out at tail
  - implies careful control of resource mux'ing
- **Little/no data sharing between domains is necessary**
  - possibly not true for CGI's backed by DB?

©2001, Steven D. Gribble

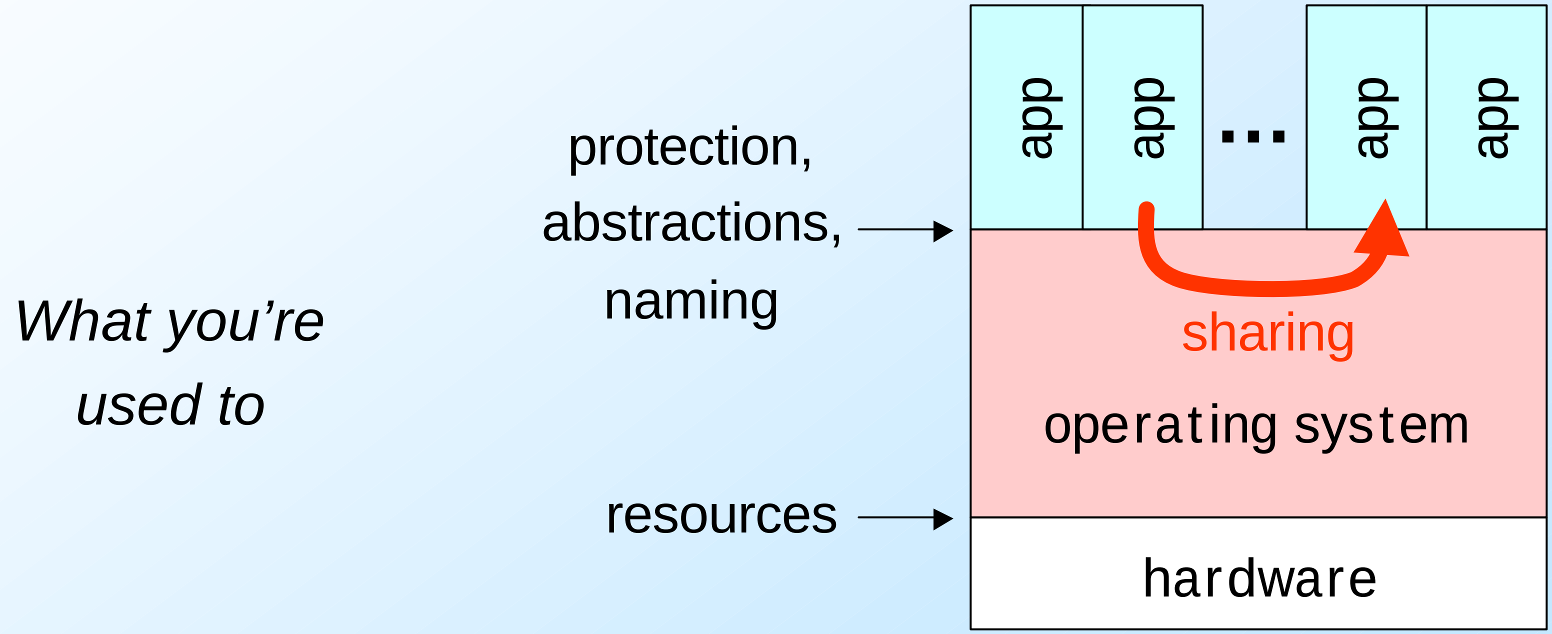
Outline

- Introduction
- Driving applications and their characteristics
- **Argument for virtual machines**
- Key challenges

©2001, Steven D. Gribble

Our intended approach

- **Virtualize at the HW interface level using *virtual machine monitors***


What you're used to

protection, abstractions, naming

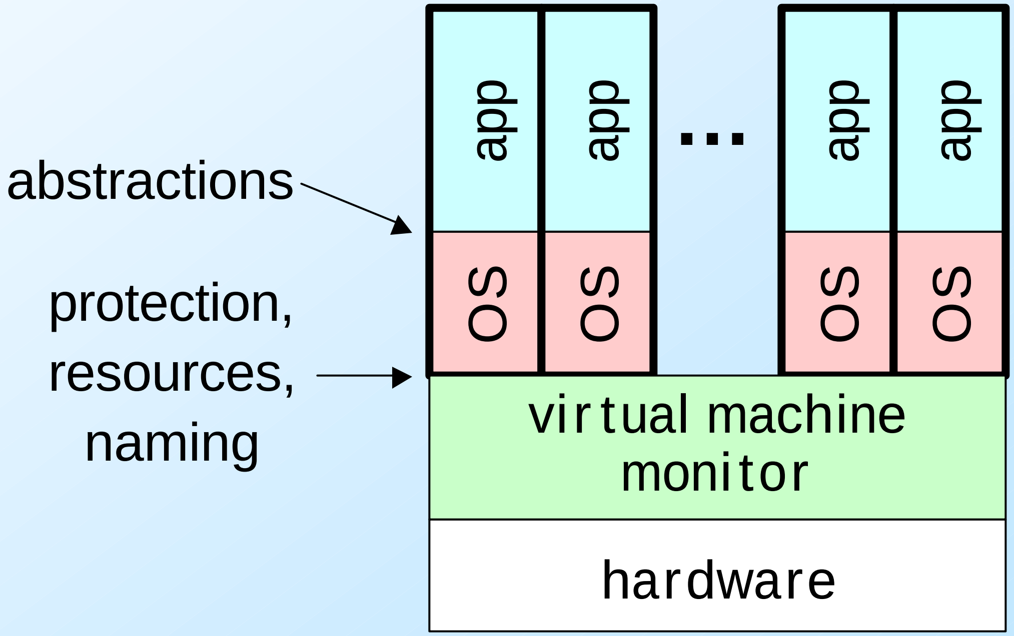
resources

©2001, Steven D. Gribble

Our intended approach

- **Virtualize at the HW interface level using *virtual machine monitors***

Virtual machine monitors



abstractions

protection, resources, naming

app app … app app

OS OS OS OS

virtual machine monitor

hardware

©2001, Steven D. Gribble

Why VMs?

Three characteristics argue for VMs:

1. **VM’s don’t impose fixed, high-level abstractions**
    - as compared with OS’s
2. **VM’s provide a simple, intuitive sharing model**
    - virtual networks between virtual machines
3. **VM’s enforce private name spaces**
    - impossible to ***name*** resources in another VM

©2001, Steven D. Gribble

1. No fixed, high-level abstractions

- **Fixed abstractions make it hard to express isolation**
  - e.g., virtual address spaces are too coarse-grained
  - e.g., DB's need record-level isolation, c.f. file system
  - virtual machines: defer abstractions to higher layer
    - don't impose single protection interface on apps
- **High level abstractions have "layer-below" problems**
  - semantic gap between abstraction and the resources being protected below abstraction
    - shared file descriptors bypassing FS access control
    - packet sniffer capturing shared files through NFS
    - forced core dumps reveal passwords

©2001, Steven D. Gribble

Compare VMs with Exokernel

- **Exokernel: MIT ultra-microkernel OS**
  - all physical hardware names directly exposed to apps (“libOS”)
    - avoid imposing inappropriate abstractions
  - resources can be shared across protection domains
    - thus, protection enforced at level of hardware
      - but below level of abstraction (disk page vs. file)
    - must map down abstraction semantics safely
- **Virtual machine monitors**
  - protection at same level as Exokernel (hardware)
  - no high-level abstractions: expose physical names
    - but: physical names are virtualized
      - hence no sharing of resources across domains
      - avoids complexity of protection below abstraction

©2001, Steven D. Gribble

2. Simple, intuitive sharing model

- **Protection can be represented by access control matrix**
  - a reference monitor enforces policy
  - two sources of security flaws:
    - badly expressed policy
    - bugs in (complex) monitor
      - monitor = OS, JRE, …

	/etc/pwd	/etc/motd
root	R,W	R,W
gribble		R

- **Virtual machines simplify both!**
  - simpler reference monitor (narrower abstractions)
  - start with **no** sharing
    - relax by allowing share-by-copy over virtual network
    - at least some hope of getting this right!
  - VMs: applications are principals, not users

Some alternatives…

- **Simplifying policies, learning policies, etc.**
  - monitor at syscall API level
    - techniques (e.g., machine learning) to deduce OK behavior
  - appeal to simpler physical metaphors
    - WindowBox: virtual windows desktops
      - still must enforce isolation at syscall level
- **Supplement existing reference monitors**
  - Janus, TCP wrappers, software wrappers
    - Janus: hard to “compile” high level policies into filters
  - Fluke: recursive reference monitors allow policy specialization
    - but again, at OS API level

©2001, Steven D. Gribble

3. Private namespaces

- **All protection domains have private namespaces**
  - many vulnerabilities come from global namespaces
    - aliasing: many names refer to same object
    - escalation of privilege: move to different column in matrix
- **One protection domain cannot name (let alone access) a resource in another protection domain!**
  - makes sharing impossible: so, allow virtual ethernet
    - single “choke point”, forces copies rather than access
    - switching, IDS, firewalls directly applicable
- **Virtualization is a level of indirection from HW**
  - transparently insert/change physical devices, migrate code, …

©2001, Steven D. Gribble

Compare with type-safe languages

- **Java, Modula-3: apps cannot forge references**
  - simpler to enforce access control with a reference monitor
    - example: no buffer overrun vulnerabilities!
  - but, all of these languages come with runtimes to access OS
    - security policy to protect this
    - same level-below + policy complexity flaws here
- **Virtual machine**
  - type-safety not important
    - all nameable resources inside one protection domain
    - TCB is entire virtual machine
  - abstractions on top of protected resources, not at same level

©2001, Steven D. Gribble

Outline

- Introduction
- Driving applications and their characteristics
- Argument for virtual machines
- **Key challenges**

©2001, Steven D. Gribble

Resource management

- **VMM at lowest-level of resource consumption**
  - possibility of accounting for all resources
    - fair-queueing of network, disk bandwidth!
  - no issue of resource principals
    - VM is only principal
- **But, VMM is unaware of abstractions**
  - danger of bad decisions
    - readahead, double-paging, etc.

©2001, Steven D. Gribble

Virtualization overhead

- **Getting rid of virtualization overhead**
  - – non-virtualizable instructions make this really hard
    - want to run VM in user-mode to protect monitor
    - privileged instructions must throw exception
      - – then, VM can catch and emulate them
    - what if instruction set isn't built this way?
      - – e.g., x86 ISA!!
      - – hairy, nasty binary-rewriting + VM tricks to get around
  - – lesson: pick physical architecture carefully

©2001, Steven D. Gribble

What OS do we run?

- **Remember the goal of 100’s of VMs?**
  - implies cannot run stock Linux or Win2K
  - need to select/modify/build something else
    - there be dragons here
- **But: protection is below level of OS**
  - can eliminate protection complexity from OS
- **Also: can pick what devices to virtualize**
  - further simplifies life (get rid of TCP/IP stack?)

©2001, Steven D. Gribble

Some final thoughts

- **Once you buy into VMs, a lot comes “for free”**
  - further relax sharing constraints
    - safe access to shared protection domains
      - copy-on-write disks, non-persistent disks
      - append-only log disks (LFS without cleaner!)
  - checkpoint/migration/recovery
    - simple to capture entire machine state
    - once you can capture it, you can move it, copy it, etc.
      - underlying hardware names are virtual!

©2001, Steven D. Gribble