Harnessing Virtual Machine Resource Control for Job Management

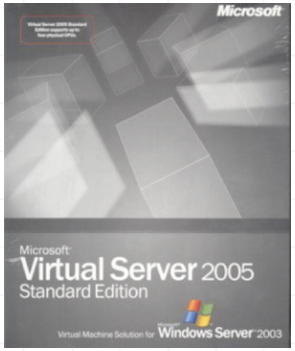
Laura Grit, David Irwin, Varun Marupadi, Piyush Shivam, **Aydan Yumerefendi**, Jeff Chase, and Jeannie Albrecht

Duke Systems

Power of Virtualization

- Virtualization offers new resource management controls
  - Performance isolation and slivering
  - Save/restore/checkpoint/migrate

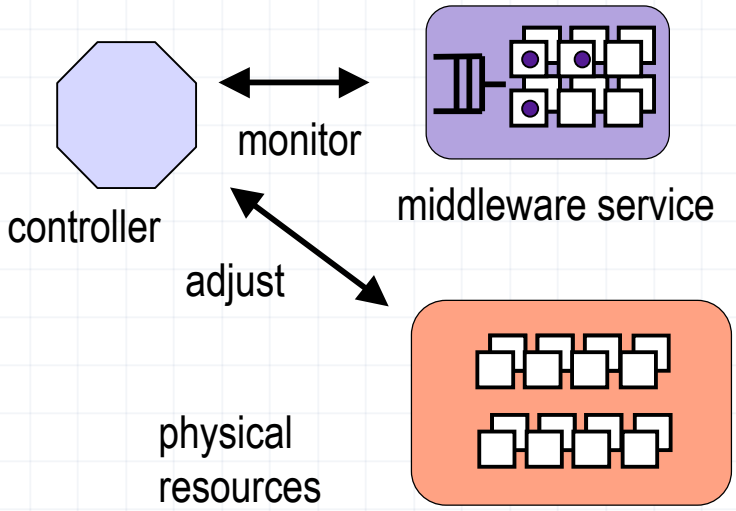
- Potential to revolutionize cluster management
  - Robust/flexible application management
  - Intelligently control collections of VMs



- How should we expose and harness the new control mechanisms to improve cluster and grid computing?

Dynamic Provisioning of Middleware

- Instantiate on demand complete middleware environments
  - Power on a machine & install operating system
  - Install and configure the middleware service
  - Update membership information

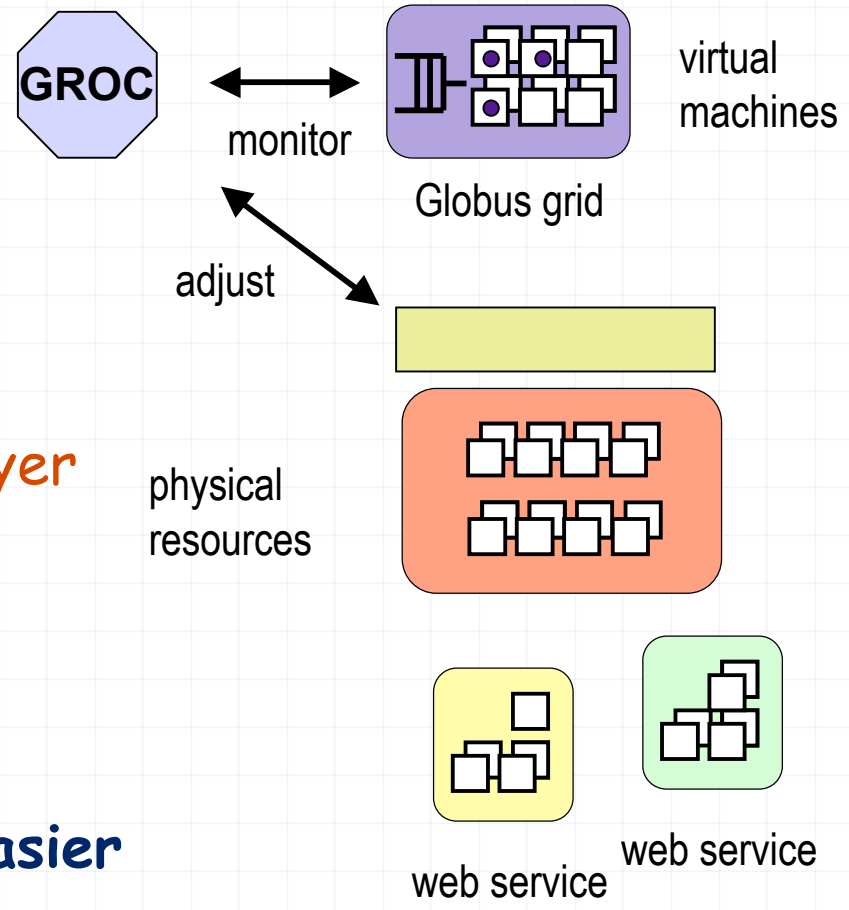


- No need to modify middleware
  - Control operations performed by a dedicated controller

- Benefits
  - Grow/shrink depending on demand
  - Share a pool of resources with other services and environments

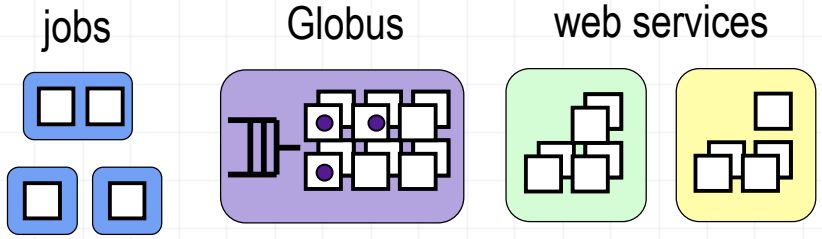
Whole Grid Virtualization

- Instantiate complete grid environments within "distributed" virtual machines [SC 06]
- Grid runs a controller (GROC)
  - Set Linux + Globus in each VM
  - Add/remove workers to Torque
- GROC interacts with a control layer
  - Arbitrates resource access
  - Offers buttons to be "pushed"
- **VMs made the whole process easier**

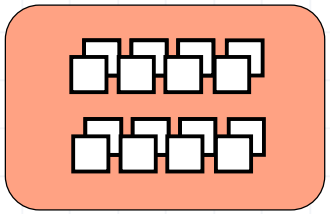


Job-level Virtualization

- Run jobs in isolated virtual machines
  - Checkpoint, stop, start, restore, migrate
  - Enforce isolation, make promises
  - Customize software stack
    - Package jobs as appliances



- Run jobs/services/complete environments from a common pool of resources



- How should we support job-level virtualization?

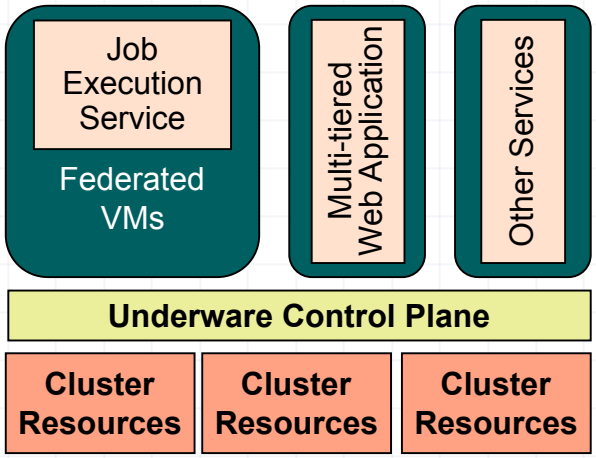
Option 1: Retrofit

- Retrofit virtualization into existing middleware systems
  - Use a familiar interface to access new functionality
  - …but bundles virtualization with specific middleware
    - Introduces obstacles for adoption
  - …but virtualization integrated into middleware:
    - Cannot share resources with other services if they do not share the same middleware layer

- There is an opportunity to rethink the architecture and produce a more streamlined, efficient, and flexible design

Option 2: Redesign

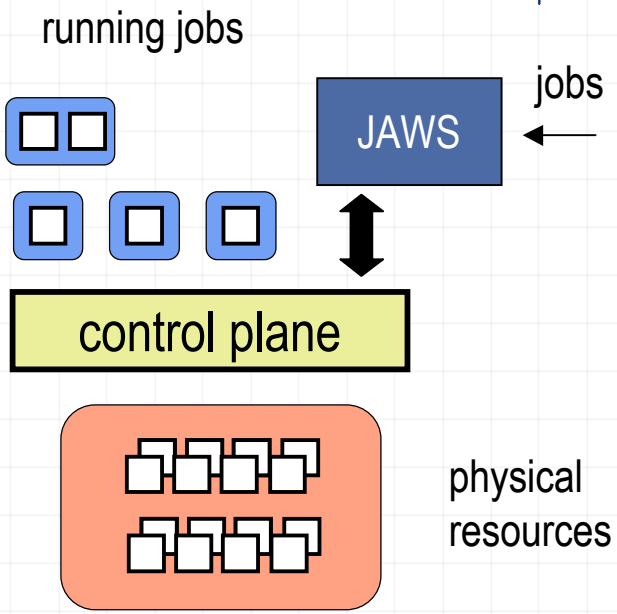
- Do not confine resource management to middleware
  - Middleware should not restrict resource sharing
- Introduce a new software layer: underware control plane
  - Sits below the middleware and operating systems
- Manages pools of resources
  - Shares them among multiple services
  - Custom arbitration policies
- Offers control & management mechanisms



JAWS

- Streamlined job execution service
  - Instantiates isolated VMs

  for each job using the control plane



- Manages jobs but is not a job scheduler
  - Scheduling & arbitration functions

  move to underware

- Main benefits:
  - Share a pool of (federated) resources with other services
  - Use new virtualization control mechanisms on a per job basis
  - Simplified and streamlined architecture

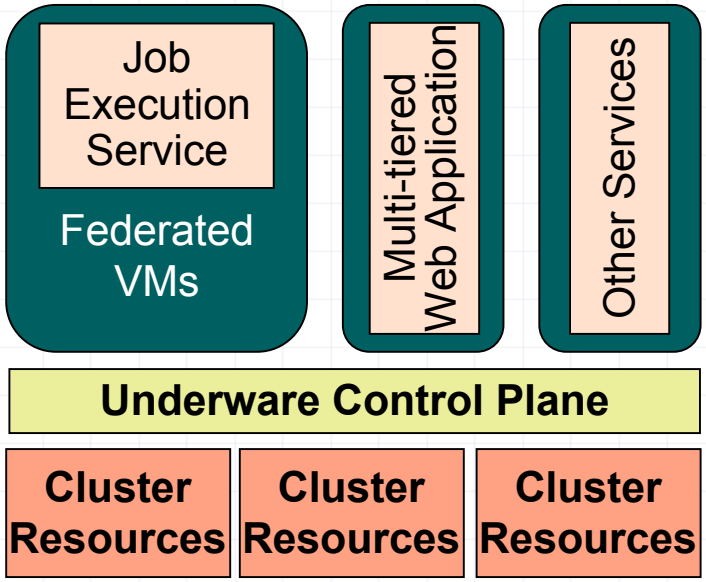
Outline

- Introduction & motivation
- Moving functionality to underware
- Job service design
- Resource control with active learning
- Summary

Underware

- Common control plane
  - Foundational layer for distributed resource management

- Key elements
  - Leases and isolation
  - Control & management interfaces
  - Federation
  - Policies

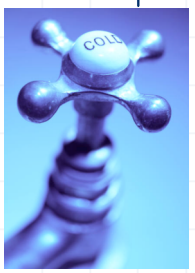


Leases and Isolation

- Leases are contracts over resource commitment
  - Grant rights to exclusive control over some quantity of resources for a period of time
  - Dynamic and renewable
  - E.g., leases for CPU capacity, memory, storage capacity, etc.

- Offer guarantees for control over resources
  - Resource isolation and predictability
- Enable advance reservations
  - Reserve resources to meet anticipated future load

Resource Control & Management Interfaces



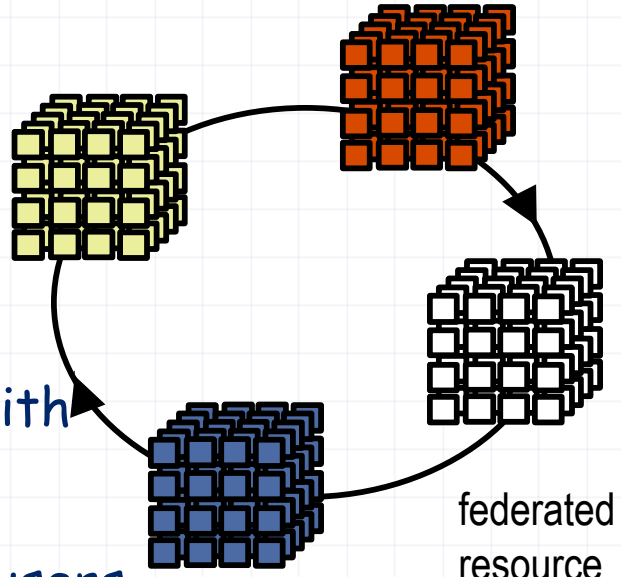
- Control interfaces
  - Clients can control the resources bound to their leases
    - Add/remove resources, adjust resource shares

- Management interfaces
  - Virtual machines: snapshot, stop, start, re-image, migrate,
  - Storage: snapshot, use saved snapshot, change capacity

- Secure programmatic access
  - Enable autonomic self-management

Federation

- Federating resource providers is attractive
  - Access resources beyond what a single provider can offer
  - Use resources more efficiently



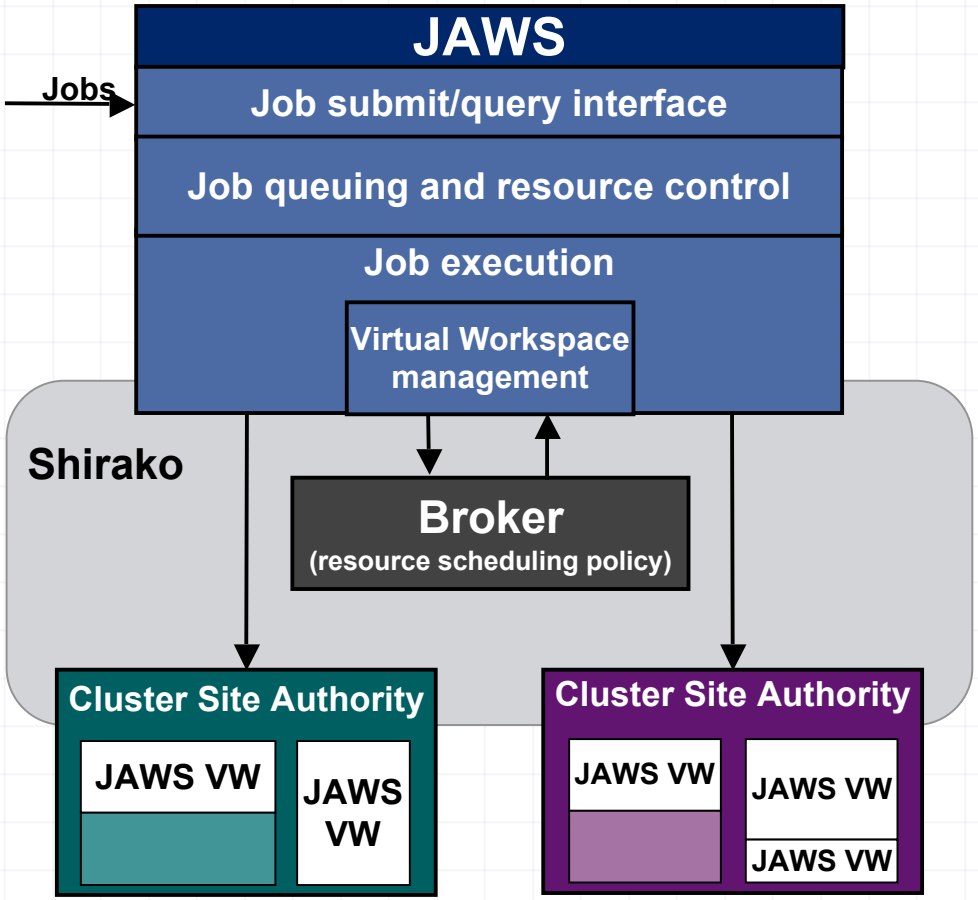
- Federation requires trust relationships
  - **Globus**: explicit federation
    - Users maintain separate relationship with each resource provider
  - **Underware**: federation is transparent to users
    - Trust relationship with a single entity
    - But may procure resources from multiple providers

Policies

- Resource management involves making choices
  - How much, when, for how long, where, and to whom
  - No "one size fits all"
- Underware enables using custom policies
  - Meet specific needs and requirements
    - Improve energy efficiency, give priority, meet deadlines
- Underware is an ideal location for resource sharing policies
  - Provides an excellent vantage point

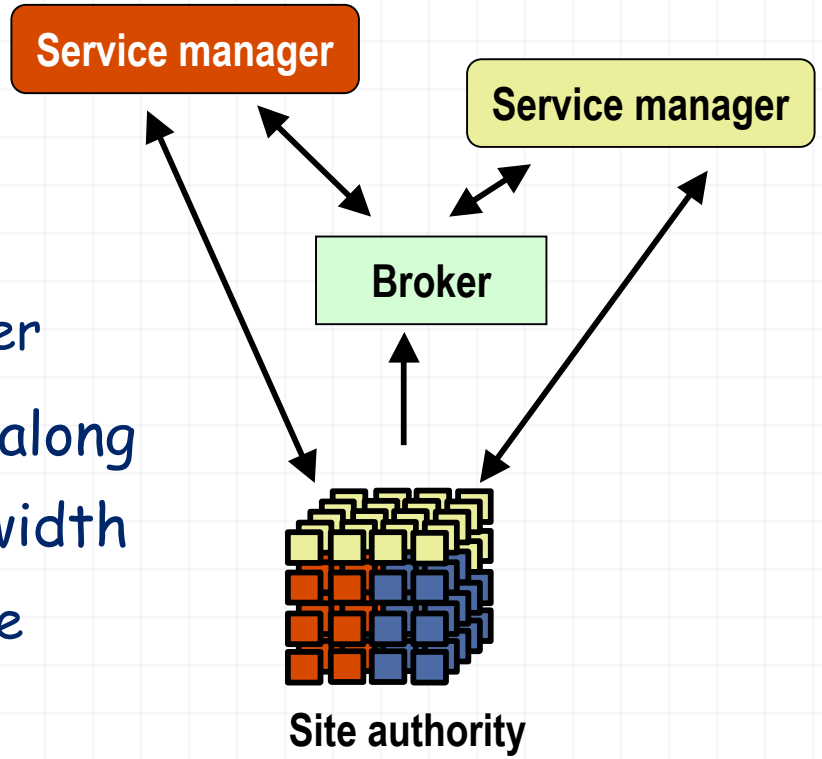
JAWS

- JAWS is a thin veneer above the underware layer
  - Determine resource requirements
  - Obtain resources
  - Initialize workspace
  - Monitor and execute
- The resource control plane takes care of everything else



Shirako

- Underware implementation [SOSP'03, USENIX'06]
  - Toolkit for lease-based resource management
  - 2+ years of development
- Three types of servers (roles)
  - Site authority: provider
  - Service manager: consumer
  - Broker: arbitrator and provisioner
- Allocates virtual machines sized along memory, CPU, and network bandwidth
  - Based on Xen but can incorporate other technologies



Separation of Responsibilities

- JAWS:
  - How big a workspace should be?
- Shirako:
  - How to subdivide the pool of resources among all jobs and other services?
    - EDF, Proportional Share, Backfill
    - Worst fit/best fit bin packing

Resource Control with Active Learning

- How much resources does a job need for its execution?
  - Give too little: slow progress
  - Give too much: wasted resources
- Same job can be executed multiple times
  - Opportunity to learn a job's resource requirements
- Active learning of performance models
  - Execute jobs over different resource configurations
  - Learn the application performance model

NIMO

- A system for learning performance models using active learning [Shivam'06]

- Performance model is a function of
  - Resource profile
  - Data profile
  - Application profile

- Select relevant sample points to learn the model quickly
  - Execute application using a specific profile
  - Analyze instrumentation data and update the model

JAWS & NIMO

- Originally
  - NIMO learned models using a specially configured dedicated heterogeneous cluster of physical machines
    - Cumbersome, error prone
  - Learning & validation of models required one month

- We integrated NIMO and JAWS
  - No need for a specially configured dedicated cluster
  - Learning & validation completed within a day
  - JAWS can use NIMO models to make better choices
    - Estimate runtime, meet deadlines
    - Limit resource waste

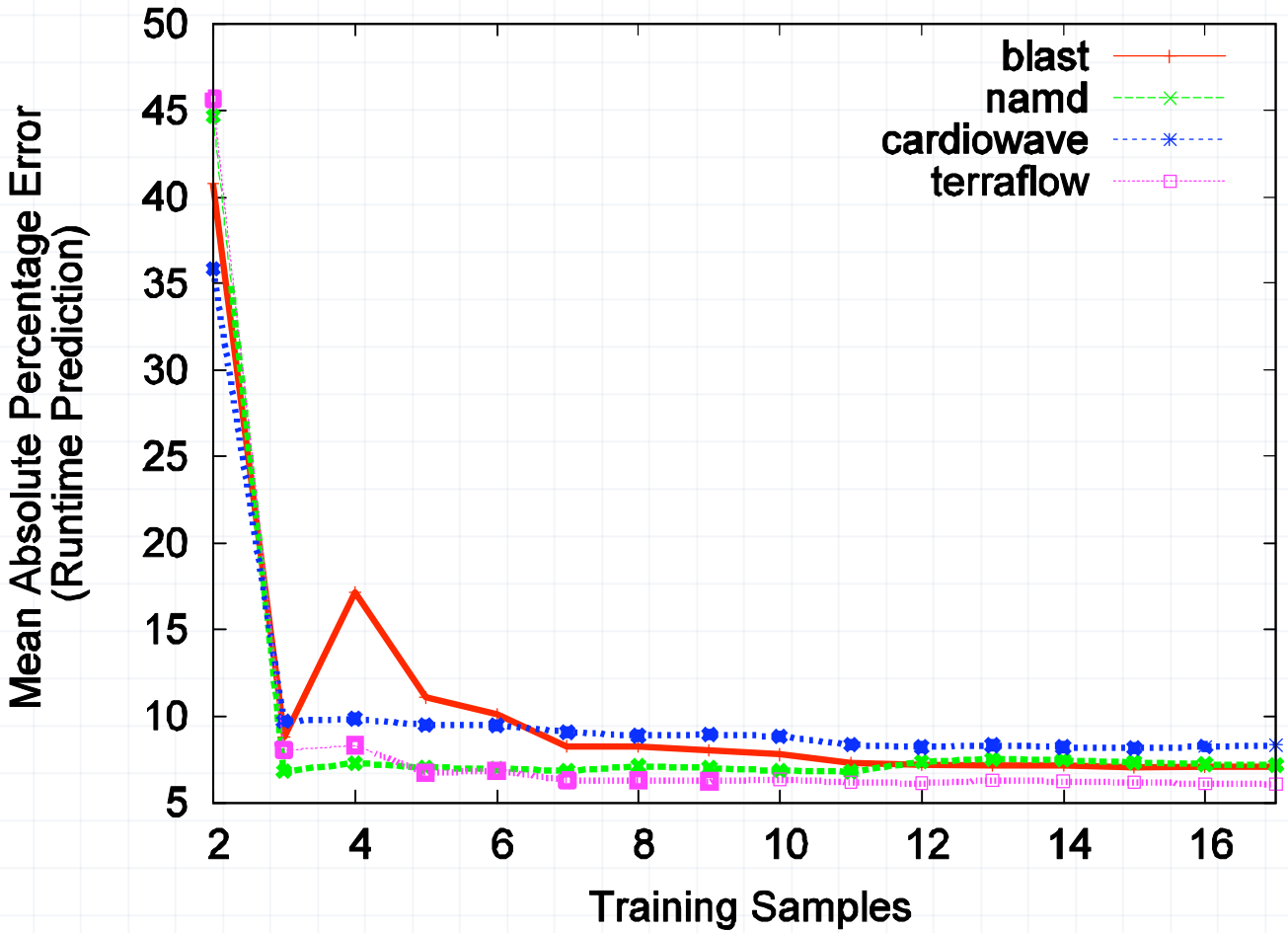
Summary

- **Job-level virtualization promises important benefits**
  - More efficient use of resources
  - More robust and easy to manage applications
- **We can redesign a job scheduler to get the most out of virtualization**
  - Move functionality into a common resource control plane
  - Leverage virtualization on a per job basis
- **Active learning offers additional benefits**
  - Size virtual machines appropriately and limit resource waste

Questions?

http://www.cs.duke.edu/nicl

Model Accuracy



90 possible configurations

Failures, Persistence, and Recovery

- Underware detects and repairs resource failures
  - VM control mechanisms help preserve and restore state
    - Restore failed computation from saved state
  - Failures can be exposed to services
    - Procure and use replacement resources

- Underware maintains persistent state
  - Records resource management decisions
  - Can quickly recover after a crash

The Case for Job-level Virtualization

- Whole grid virtualization has limitations
  - Enables dynamic provisioning
  - … but new control mechanisms are not available on a per job basis

- Scientific jobs can benefit from the new control mechanisms
  - Long running, require a complex hierarchy of resources
    - Need for checkpointing, suspend, resume, etc.
  - Infrastructure offers limited predictability and isolation
    - Hard to meet deadlines and make promises
  - Dependencies on libraries, operating systems, middleware, etc.
    - Require customized environments: management problem

Job-level Virtualization

- Run jobs in dedicated virtual workspaces
  - A collection of hardware resources and customized software environment required for the execution of a job
- Advantages:
  - Apply new control mechanisms per job
  - Run jobs/services/complete environments from a common pool of resources
- How should a virtual workspace service be designed?

Job Workspaces in JAWS

- For each job in the job queue:
  - Determine the desired workspace characteristics
  - Request the workspace from the Shirako broker
  - Run and monitor/manage the job once the workspace is ready
- JAWS resource policy:
  - How big a workspace should be?
- Resource sharing/scheduling performed by Shirako broker
  - EDF, Proportional Share, Backfill
  - Worst fit/best fit bin packing