Dynamic Resource Management using Operating System-Level Virtualization

Computer Science 4490z
Undergraduate Research Project Thesis

Alexander Pokluda

Department of Computer Science
University of Western Ontario

April 10, 2010

Outline

Outline

1. Introduction
2. OpenVZ
   - Resource Management
   - Managing Resource Stress Situations
3. System Overview
   - Architecture Overview
   - Design Overview
   - Source Code Availability and Documentation
4. Validation
   - Test Environment
   - Experimental Results
5. Conclusion
   - Future Work
   - Summary

Outline

Outline

1. Introduction
2. OpenVZ
   - Resource Management
   - Managing Resource Stress Situations
3. System Overview
   - Architecture Overview
   - Design Overview
   - Source Code Availability and Documentation
4. Validation
   - Test Environment
   - Experimental Results
5. Conclusion
   - Future Work
   - Summary

Outline

Outline

1. Introduction
2. OpenVZ
   - Resource Management
   - Managing Resource Stress Situations
3. System Overview
   - Architecture Overview
   - Design Overview
   - Source Code Availability and Documentation
4. Validation
   - Test Environment
   - Experimental Results
5. Conclusion
   - Future Work
   - Summary

Types of Virtualization

Definition

Virtualization is the abstraction of computer resources

- **Platform virtualization** enables the execution of one or more **virtual machines** on a single computer
- Modern types of platform virtualization include:
  - *full virtualization*
  - *hardware-assisted virtualization*
  - *paravirtualization*
  - *operating system-level virtualization*
- In operating system-level virtualization, all virtual machines (also known as **virtual environments**) share one operating system kernel
- Operating system-level virtualization has very little overhead so applications can achieve near-native performance

Motivation for Using Virtualization

- By using virtual machines to run many independent software systems on a single physical server, greater resource utilization levels can be achieved
- Greater resource utilization levels mean that less physical resources are required and overall costs are reduced

Problem Statement

New Problem

How do we effectively manage the resources of a cluster of hardware nodes as a single unit?

- This thesis expands upon a system called **Golondrina**
- Golondrina works by identifying localized **resource stress** situations then attempting to dissipate them by reallocating system resources and, if necessary, by **migrating** or **replicating** virtual environments

Contributions:

- memory management studied
- heuristic developed
- new architecture
- CPU code as plug-in
- memory plug-in developed
- documentation
- testing

Outline

OpenVZ

What is OpenVZ?

OpenVZ is a mature open-source community project that implements operating system-level virtualization using Linux

OpenVZ provides four primary controls for per-container resource accounting and limiting:

- **user beancounters**
- **disk quota management**
- **CPU fair scheduler**
- **configurable input/output priorities**

System administrators can also use standard Linux resource management and accounting mechanisms such as `tc` and `iptables`

Memory Allocation

- OS-level virtualization makes it trivial to add or remove arbitrary amounts of memory to or from a container in real time. In OpenVZ, this is done with the user beancounters.
- There are currently 24 beancounters and each has 5 attributes: `held`, `maxheld`, `barrier`, `limit`, `failcnt`

Three main user beancounters relate directly to memory management:

`vmguarpages` guaranteed virtual memory pages

`privvmpages` private virtual memory pages

`oomguarpages` out-of-memory guaranteed memory pages

- In Linux and OpenVZ memory is overcommitted by default

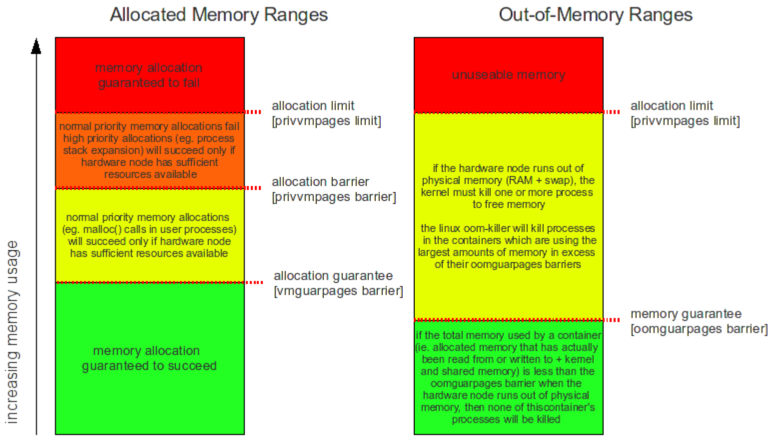

Figure: Typical relative settings of the `privvmpages`, `vmguarpages` and `oomguarpages` user beancounters for a container

Golondrina and OpenVZ Resource Management

First Version

- monitored a single resource: the processor
- monitored CPU time using the virtual file `/proc/vz/vestat`
- dissipated resource stress situations using **migrations** or **replications**
- has been packaged as a plug-in in the current version

Current Version

- memory management plug-in developed
- monitors memory usage within each container and for hardware node using `user_beancounters` virtual file and `vzmemcheck`
- dissipates resource stress using **user beancounters**, **migrations** and **replications**

Managing Resource Stress Situations

- Goal: ensure resource availability for each virtual machine while achieving high levels of resource utilization on each hardware node
- Ideally containers should be distributed such that a predetermined level of resource utilization, say x, is achieved on each hardware node
- This is equivalent to the NP-complete subset-sum problem

Proof.

Let S_0 be the set of containers to be distributed across several hardware nodes. If we wish to achieve a target resource utilization level of x on each hardware node, then we need to find a subset $s_0 \subseteq S_0$, such that the sum of the resource usage of each container in s_0 is exactly x. Once this subset has been found, the containers in s_0 are assigned to a hardware node and the process repeats for $S_1 = S_0 - s_0$. $\square$

- Finding good resource allocation strategies is challenging and and active area of research

Resource Stress Indicators for Memory

1. An increase in `oomguarpages failcnt` value
   - **raw score**: the increase in `oomguarpages failcnt`, i.e. the number of processes that have been killed
   - **normalized score** $= \begin{cases} 0.0 & \text{if } \textit{rawscore} \text{ is } 0 \\ 1.0 & \text{otherwise} \end{cases}$
2. An increase in `privvmpages failcnt` value
   - **raw score**: the increase in `privvmpages failcnt`, i.e. the number of failed memory allocation attempts
   - **normalized score** $= \begin{cases} 0.0 & \text{if } \textit{rawscore} \text{ is } 0 \\ 1.0 & \text{otherwise} \end{cases}$
3. Current memory usage (`oomguarpages held` + `kmemsize held` + `*buf held`) versus `oomguarpages barrier`
   - **raw score**: (`oomguarpages held` + `kmemsize held` + `*buf held`) / `oomguarpages barrier`, i.e. the fraction of guaranteed memory used
   - **normalized score** = min{1.0, *rawscore*}
4. `privvmpages held` versus `privvmpages barrier`
   - **raw score**: `privvmpages held` / `privvmpages barrier`, i.e. the fraction of memory the memory allocation guarantee used
   - **normalized score** = min{1.0, *rawscore*}

Resource Stress Resolution for Memory

A basic heuristic policy for resolving a memory stress situation in a container:

1. Increase the memory limit for the stressed container
2. Migrate stressed container to another hardware node
3. Migrate another container
4. Alternatively, start a replica

A basic heuristic policy for resolving a memory stress situation on a hardware node:

1. Migrate the container that is using the largest amount of resources
2. Migrate the container that is using next largest amount of resources
3. Repeat

Outline

Architecture Overview

- The system has been architected as a research system that will be a test bed for various resource management polices
- The primary architectural pattern for Golondrina is “client-server”

Client Component

- collects resource usage statistics and sends them to the server

Gate Component

- manages configuration for an external load balancer

Server Component

- Analyses resource usage statistics to identify resource stress situations
- Instructs client components to adjust resource limits and perform migrations and replications
- Instructs gate component to update load balancing configuration

Architecture Overview

- The system has been architected as a research system that will be a test bed for various resource management polices
- The primary architectural pattern for Golondrina is “client-server”

Client Component

- collects resource usage statistics and sends them to the server

Gate Component

- manages configuration for an external load balancer

Server Component

- Analyses resource usage statistics to identify resource stress situations
- Instructs client components to adjust resource limits and perform migrations and replications
- Instructs gate component to update load balancing configuration

Architecture Overview

- The system has been architected as a research system that will be a test bed for various resource management polices
- The primary architectural pattern for Golondrina is “client-server”

Client Component

- collects resource usage statistics and sends them to the server

Gate Component

- manages configuration for an external load balancer

Server Component

- Analyses resource usage statistics to identify resource stress situations
- Instructs client components to adjust resource limits and perform migrations and replications
- Instructs gate component to update load balancing configuration

Architecture Overview

- The system has been architected as a research system that will be a test bed for various resource management polices
- The primary architectural pattern for Golondrina is “client-server”

Client Component

- collects resource usage statistics and sends them to the server

Gate Component

- manages configuration for an external load balancer

Server Component

- Analyses resource usage statistics to identify resource stress situations
- Instructs client components to adjust resource limits and perform migrations and replications
- Instructs gate component to update load balancing configuration

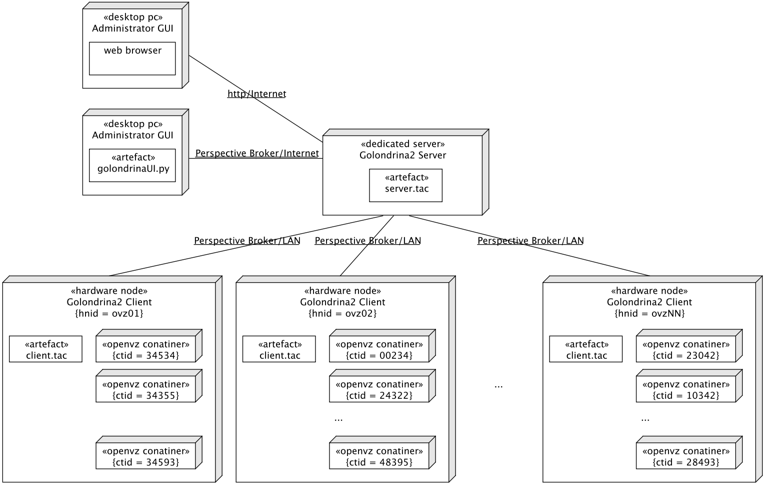

Figure: Golondrina deployment diagram

Design Overview

- The software for collecting and analysing resource usage statistics has been encapsulated in **plug-ins**
  - sensor plug-ins
  - overload identifier plug-ins
  - overload resolver plug-ins
- Each overload resolver plug-in is given a priority value
- Once a resource stress has been identified, the overload resolver plug-ins for the stressed resource(s) are each given a chance in turn to try and dissipate the resource stress
- Certain aspects of a policy's state (e.g. a threshold value) can up updated at runtime
- A configuration manager subsystem maintains a database of configuration information for the whole system and can be used to alter the system's run-time behaviour

Source Code Availability and Documentation

- The source code developed as part of this thesis is free software, anyone is free to redistribute it and/or modify it under the terms of the GNU General Public Licence
- Source code and documentation is available at: `http://alexanderpokluda.ca/trac/cs4490`

Outline

Test Environment

Two configurations were investigated for the virtual machine to be placed under load:

- TPC-W benchmark
- LAMP software stack

- Joomla! was selected as the specific application to be tested running on the LAMP stack
- Apache JMeter was used for load generation and performance monitoring
- Containers created using CentOS 5.3 template
- Also installed: Apache HTTP Server ver. 2.2.3, MySQL ver. 5.0.45, PHP ver. 5.1.6, and Joomla! ver. 1.5.14
- Three identical hardware nodes, each with 3.40 GHz dual-core processor, 2 GiB RAM, 2 GiB swap
- Apache HTTP Server **prefork** module used

Experimental Results

- Test 0 – No Memory Stress
- Test 1 – Unresolved Memory Stress
- Test 2 – Memory Stress Resolved Locally
- Test 3 – Memory Stress Resolved with Migration

Test	Avg	Min	Max	Std Dev	Err %	Throughput	Fail Count
0	448	166	2275	243.72	0.00	8.0	0
1	1082	124	55576	5817.40	2.22	3.6	1002
2	866	115	49214	4748.84	1.63	4.9	809
3	370	137	4354	419.42	9.24	5.5	1143

Table: Results for Four Tests, each with four runs

Outline

Future Work

- Experiments involving memory stresses and replication
- Study the interaction between policies when a container is experiencing a stress for more than one resource

Possible improvements in future versions:

- Add a mechanism to reclaim resources
- Use remote storage for container private areas
- Distribute decision making
- Add memory resource usage prediction

Summary

- Golondrina is a system that performs dynamic resource management among a cluster of hardware nodes
- Different models of virtualization and the advantages of operating system-level virtualization were discussed
- The heuristic currently used to detect and dissipate memory stress situations was presented
- The architecture and design of Golondrina was discussed
- The functionality of the system was validated using a series of experimental tests
- A brief summary of experiments yet to be performed and possible future enhancements was presented