A Case for Scaling Applications to Many-core with OS Clustering

Xiang Song
Work with
Haibo Chen, Rong Chen, Yuanxuan Wang and Binyu Zang
Parallel Processing Institute, Fudan University
Since a Long Time Ago ...

GNU/Linux
Land is growing ...
People proposed to raise fish

Corey, fos, Barrellfish
But...

People are refining gnu to make it more delicious than before

Fish appears to be delicious for some people

RCU
McKenney, et.al.
2002

Sloppy Counter
Boyd-Wickizer, et.al.
2010

Barrelfish
Baumann, et.al.
2009

fos
Wentzlaff, et.al.

Corey
Boyd-Wickizer, et.al.
2008
Fish or Cerberus? A choice.
Commodity OS *scale well* with a *small number of cores* for many applications.
Basic idea

– *Clustering* multiple OSes atop a VMM
– Serve one application with *POSIX interface*

Goal

– Evaluate a middle point of improving scalability of commodity OSes
– Backward compatible to POSIX interface
Challenges

• An app’s resources spans across multiple OSes
  – Process/thread, address space, fs, network

• Should Provide
  – Distributed process/thread management
  – Single shared-memory interface
  – Efficient resource sharing
Outline

• Cerberus Architecture
• Challenges & Solutions
• Implementation
• Evaluation
Cerberus Architecture

- User
- System call virtualization
- Resource sharing layer
- VMM
- Hardware
Detailed Architecture

VM1
- Cerberus Daemon
- Super Process
  - Threads
- Shared Memory Area
- SuperP module
- Address Space
- File System
- Network

VM2
- Cerberus Daemon
- Super Process
  - Threads
- Shared Memory Area
- SuperP module
Outline

• Cerberus Architecture
• **Challenges & Solutions**
• Implementation
• Evaluation
Challenge A

Need to run an app on a cluster of VMs

How does Cerberus spawn a thread/process on a remote VM?
Support Super Process

Remote Process/Thread Spawning

VM1

SuperP module

Root process

Process

Remote fork

Register Stack

VM2

SuperP module

Process

Fork

Resident

Fork

Super Process

Shared Memory Area

Inter-VM call

User

Kernel

Inter-VM call

VMM
Challenge B

We have threads/processes on different VMs

How does Cerberus identify a thread/process?
Process management
– Virtual PID for each process/thread

1. Magic Num
2. VM ID
3. Child Num

Super Process

VM 1
Threads

VM 2
Threads

VM 3
Threads
Challenge C

Threads/processes distributed across VMs, they need to cooperate

How does Cerberus virtualize system calls?
Two types of system calls

- Access **local** state or stateless
  - Native syscall handler (e.g., `gettimeofday`)

- Access or modify **global** state
  - Cerberus syscall handler (e.g., `signal`)

System call virtualization
Example: Signal

- **Process** in VM1
- **Super Process** in VM2
- **Signal handler**
- **Sigkill VPID**
- **Local?**
- **VM ID**
- **Cross-VM call handler**

**System call virtualization**

**Cerberus syscall layer**

**Native syscall layer**

**User**

**Kernel**

**VMM**
Challenge D

Threads/processes distributed among multiple VMs need to share data

How does Cerberus provide address space sharing?
Address space sharing

Sharing a sub-tree of page table
– Corey (Boyd-Wickizer, et.al.)
Address space sharing

- Thread I
  - Private entry
  - Shared entry
  - L3 page table

- Thread II
  - Shared entry
  - L3 page table

- Thread III
  - Shared entry
  - L3 page table

- Thread IV
  - Private entry
  - L3 page table

Application

L2 page table
Challenge E

We have threads/processes sharing resources

How does Cerberus provide resource sharing?
File system and Network sharing
– Most of accesses will be local

- Fast local access
- Acceptable remote access

![Diagram showing support resource sharing with Request Dispatcher connecting to Local Resources for OS-I, OS-II, and OS-III.]
Support Resource Sharing

• Example – remote file read
Outline

• Cerberus Architecture
• Challenges & Solutions

• Implementation
• Evaluation
Implementation

Based on Xen 3.3.0 + para-virtual VM

Add 1,800 lines in Xen VMM
  – Management of super process
  – Efficient data sharing

8,800 lines for SuperP module
  – Support for super process
  – Resource sharing
  – Other support code
## Implementation

Virtualize 35 POSIX system calls

<table>
<thead>
<tr>
<th></th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process/thread creation and exit</td>
<td>fork, clone</td>
</tr>
<tr>
<td>Process/thread communication</td>
<td>futex, signal</td>
</tr>
<tr>
<td>Memory management</td>
<td>brk, mmap</td>
</tr>
<tr>
<td>Network operations</td>
<td>socket, connect</td>
</tr>
<tr>
<td>File system operations</td>
<td>open, read</td>
</tr>
<tr>
<td>Security</td>
<td>unhandled</td>
</tr>
<tr>
<td>Real time signal</td>
<td>unhandled</td>
</tr>
<tr>
<td>Debugging</td>
<td>unhandled</td>
</tr>
<tr>
<td>Kernel modules</td>
<td>unhandled</td>
</tr>
</tbody>
</table>
Outline

• Cerberus Architecture
• Challenges & Solutions
• Implementation

• Evaluation
Experimental Setup

Software

Benchmarks
- Histogram
- Dbench

Hardware

AMD Machine
- 48 Core (8 X 6) Opteron

Oprofile/Xenoprof
Evaluation environment

Three systems

- Linux 2.6.18
- Xen-Linux Dom0
- Cerberus
  - 1-core/domain
  - 2-core/domain
Application Benchmarks

Evaluation on the AMD machine

- Histogram
  - 4 GByte input in ramfs
  - 1 thread/core

- Dbench
  - 1 client/core
histogram

Execution time (secs)

Cores

Linux
-×-Xen-Linux
△Cerberus 1-core/dom
+Cerberus 2-core/dom
## Histogram analysis

- **Top 3 hottest functions**

<table>
<thead>
<tr>
<th>System</th>
<th>Top 3 functions</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Linux 48 threads</td>
<td>__up_read</td>
<td>38.6%</td>
</tr>
<tr>
<td></td>
<td>__down_read_trylock</td>
<td>35.9%</td>
</tr>
<tr>
<td></td>
<td>calc_hist</td>
<td>8.3%</td>
</tr>
<tr>
<td>Cerberus 2-core/VM 48 threads</td>
<td>calc_hist</td>
<td>22.5%</td>
</tr>
<tr>
<td></td>
<td>sh x86 emulate cmpxchg guest 2</td>
<td>8.9%</td>
</tr>
<tr>
<td></td>
<td>/xen-unknown</td>
<td>8.3%</td>
</tr>
</tbody>
</table>
Dbench

Throughput (MB/sec)

num of cores

Linux
-Xen-Linux
Cerberus 1-core/dom
Cerberus 2-core/dom
## Dbench analysis

- **Top 3 hottest functions**

<table>
<thead>
<tr>
<th>System</th>
<th>Top 3 functions</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Linux</td>
<td>ext3_test_allocatable</td>
<td>66.6%</td>
</tr>
<tr>
<td>48 processes</td>
<td>bitmap_search_next_usable_block</td>
<td>18.2%</td>
</tr>
<tr>
<td></td>
<td>journal_dirty_metadata</td>
<td>0.02%</td>
</tr>
<tr>
<td>Cerberus</td>
<td>sh_x86_emulate_cmpxc hg_guest_2</td>
<td>11.2%</td>
</tr>
<tr>
<td>2-core/VM</td>
<td>/xen-unknown</td>
<td>8.67%</td>
</tr>
<tr>
<td>48 processes</td>
<td>sh_x86_emulate_write__guest_2</td>
<td>5.2%</td>
</tr>
</tbody>
</table>
Limitations

Scenarios, Cerberus won’t profit

- Applications that clone a number of short-lived, intensively communicating threads/processes

- Applications with frequent remote resource accesses

- Applications with frequent small-size memory mapping operations
Conclusion

Cerberus system
  – Provide applications with the familiar **POSIX** programming interface
  – Improve scalability of commodity OSes

Experiments on Cerberus
  – Some applications can **scale better** on Cerberus
Future Work

• Cerberus without a VMM?
  – VMM incurs non-trivial overhead, it is not essentially necessary
  – A small dedicated thin software layer is sufficient

• Adjust Linux to make it cooperate with Cerberus
  – Minimize overheads associated with forks/mmaps/signals
Thanks

Cerberus

See the *past* and *present*

Looking for the *future*

Questions?

Parallel Processing Institute

http://ppi.fudan.edu.cn
Backup
Cerberus with more cores per domain

- The performance grows down when the number of cores per domain increases
  - The Shadow paging mode
  - Overhead introduced by the virtual layer
Micro Benchmarks

• Remote fork/clone

<table>
<thead>
<tr>
<th></th>
<th>1 process/thread</th>
<th>24 processes/threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fork</td>
<td>5.40ms</td>
<td>31.77ms</td>
</tr>
<tr>
<td>Clone</td>
<td>3.21ms</td>
<td>30.79ms</td>
</tr>
</tbody>
</table>

• Cross-VM message passing
  – 10.24μs for inner-chip
  – 11.34 μs for inter-chip
Micro Benchmarks

• Local operations vs. remote operations
  – Sending signal
  – Sending and receiving packages
  – Read files
Micro Benchmarks

• Local operations vs. remote operations
  – Sending signal
    • Cost of ping-ponging 1000 signals

<table>
<thead>
<tr>
<th></th>
<th>Intel machine</th>
<th>AMD machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Linux</td>
<td>7.9ms</td>
<td>4.0ms</td>
</tr>
<tr>
<td>Xen-Linux</td>
<td>38.7ms</td>
<td>74.1ms</td>
</tr>
<tr>
<td>Cerberus local</td>
<td>43.1ms</td>
<td>72.3ms</td>
</tr>
<tr>
<td>Cerberus remote</td>
<td>25.8ms</td>
<td>45.0ms</td>
</tr>
</tbody>
</table>

– Read files
Micro Benchmarks

• Local operations vs. remote operations
  – Sending and receiving packages
  • Cost of ping-ponging one packet 1000 times

<table>
<thead>
<tr>
<th></th>
<th>Local host</th>
<th>Remote host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Linux</td>
<td>11.25ms</td>
<td>125.5ms</td>
</tr>
<tr>
<td>Xen-Linux</td>
<td>42.9ms</td>
<td>132.6ms</td>
</tr>
<tr>
<td>Cerberus local</td>
<td>43.1ms</td>
<td>131.8ms</td>
</tr>
<tr>
<td>Cerberus remote</td>
<td>87.1ms</td>
<td>154.7ms</td>
</tr>
</tbody>
</table>
Application Benchmarks

Apache and Memcached
  – A pool of 16 dual-core client machine

Cerberus configuration
  – 2 core/domain
  – 1 NIC/domain
  – 2 instances/domain
Apache

Throughput (requests/sec)

- Linux
- Xen-Linux
- Cerberus

Cores

1 2 4 6 8 10 12 14 16
Throughput (10000 requests/sec)

num of cores

- Linux
- Xen-Linux
- Cerberus
Apache performance

• The profiling result shows
  – The CPU utilization much lower in 16-core than in 1-core
    • 38.9% for Linux
    • 41.8% for Xen-Linux