Lessons learned from OSV

Avi Kivity, Glauber Costa
Cloudius Systems
Introduction to OS
Why C++ for systems programming?
Examples
QEMU and OS\(^V\) requirements
Typical Cloud Stack

Your App
Application Server
JVM
Operating System
Hypervisor
Hardware
A Historical Anomaly

Your App

Application Server

JVM provides protection and abstraction

Operating System provides protection and abstraction

Hypervisor provides protection and abstraction

Hardware
Too Many Layers, Too Little Value

<table>
<thead>
<tr>
<th>Property/Component</th>
<th>VMM</th>
<th>OS</th>
<th>runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware abstraction</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Isolation</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Resource virtualization</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Backward compatibility</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Memory management</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>I/O stack</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

Duplication
less is more.
The new **Cloud Stack - OS^v**

- **Your App**
- **Application Server**
- **JVM**
- **Core**
- **Hypervisor**
- **Hardware**

**Single Process**

**Kernel space only**

**Linked to existing JVMs**

**App sees no change**
The new Cloud Stack - OS

- **Memory**: Huge pages, Heap vs Sys
- **I/O**: Zero copy, full aio, batching
- **Scheduling**: Lock free, low latency
- **Tuning**: Out of the box, auto
- **CPU**: Low cost ctx, Direct signals,..
Van Jacobson == TCP/IP

Common kernel network stack

Leads to servo-loop:
Van Jacobson == TCP/IP

Net Channel design:
Dynamic heap, sharing is good

JVM Memory

Lend memory

System memory
Architecture ports

- **64-bit x86**
  - KVM - running like a bat out of hell
  - Xen HVM - running (still slow :-( )
  - VMware - planned in 2 months
- **64-bit ARM - planned**
- **Others - patches welcome**
Management

OSv application deployment

Deploy your Java applications into OSv by following these steps:

- Upload your application zip file (see example project).
- Activate the uploaded application by starting it.

+ Add files...
Choose Files: No file chosen
Status

- Runs:
  - Java, C, JRuby, Scala, Groovy, Clojure, JavaScript
- Outperforms Linux:
  - SpecJVM, MemCacheD, Cassandra, TCP/IP
- 400% better w/ scheduler micro-benchmark
- < 1sec boot time
- ZFS filesystem
- Huge pages from the very beginning
Milestones

- Formation, 12/2012
- Seed, 02/2013
- KVM, networking, 04/2013
- Outperform Other OSs, 07/2013
- OSS launch, limited GA, Beginning 2014
- First OEM revenue, Q1/2015
- OSS launch, Memcached outperform by 40%, 9/2013

TCP/IP works; Performance: >15Gbps netperf, 7/2013

UDP, 03/2013

TCP offload, 8/2013

OSS launch, Memcached outperform by 40%, 9/2013

Git init osv, 12/2012

64 vcpu kvm support, 02/2013

TCP/IP works; Performance: 50Mbps, 4/2013

UDP, 03/2013

TCP offload, >15Gbps netperf, 7/2013

ZFS mount, 6/2013

> 1Gbps netperf, 6/2013

OSS launch, Memcached outperform by 40%, 9/2013

Java hello world, 01/2013

Virtio blk over ram FS, 2/2013

Cassandra works; Cassandra outperforms Linux, 8/2013
Two languages called C++

1. Strongly typed object oriented language specialized in leveraging synergies within business process for on demand needs of global companies in a dynamic paradigm shift
Two languages called C++

2. A macro language for generating C
Two languages called C++

2. A macro language for generating C

- An elaborate macro language
  - Reduce boilerplate
  - Reduce C macros
  - More libraries, reuse
  - Less duplication

- Let the compiler write your C code
Scoped locking

```c
int before(struct something *p)
{
    int r;
    r = -ENOENT;
    if (!p)
        goto out2;
    mutex_lock(&p->lock);
    r = -EINVAL;
    if (!p->y)
        goto out1;
    mutex_lock(&p->y->lock);
    r = ++p->y->n;
    mutex_unlock(&p->y->lock);
    out1:
    mutex_unlock(&p->lock);
    out2:
    return r;
}
```

```c
int after(something* p)
{
    if (!p)
        return -ENOENT;
    WITH_LOCK(p->lock) {
        if (!p->y)
            return -EINVAL;
        WITH_LOCK(p->y->lock)
            return ++p->y->n;
    }
}
```
Performance and tracing

TRACEPOINT(trace_mutex_lock, "%p", mutex *);
TRACEPOINT(trace_mutex_lock_wait, "%p", mutex *);

// ...

void mutex::lock()
{
    trace_mutex_lock(this);

    $ perf stat mutex_lock mutex_lock_wait sched_switch
    mutex_lock      mutex_lock_wait      sched_switch
                    11                  0            2
                    885                 0           181
                    154                 0           152
                    154                 0           154
                    404                 0           190
                    222                 0           157
                    150                 0           152
Atomic allocation & initialization

Allocate memory and initialize it in one step

- No need to track the size
- No error checking between steps
Containers

- `vector<foo>` - growable array
- `unordered_map<key, value>` - growable hash table
- `list<bar>` - doubly linked list
- `set<whatever>` - sorted balanced tree

Reduce the role of laziness in determining key data structures
rcu_ptr<vector<device>> device_list;

// update:
device_list.assign(new_device_list);

// read:
auto list = device_list.read();
Reference counted objects

shared_ptr<device> - fully automatic reference counting

intrusive_ptr<device> - full manual control
function<void (int level)>&& irq_handler;
function<u64 (hw_addr addr, unsigned size)>> read_callback;

irq_handler = my_irq_handler;
read_callback = bind(this, &my_device::read);
function<void (int level)>& irq_handler;
function<u64 (hw_addr addr, unsigned size)>& read_callback;

irq_handler = my_irq_handler;
read_callback = bind(this, &my_device::read);
Signals and slots

signal <void ()> system_reset;

...

system_reset.connect([&] { reset_bar0(); });
...

system_reset();
Conclusions

- OS\textsuperscript{V} experience shows modern system programming is made easier in C++
- Boilerplate (and silly mistakes) reduced
- Easy, fast to use and build frameworks
- More fun too!
- Lessons applicable to QEMU
Resources

http://osv.io

https://github.com/cloudius-systems/osv

@CloudiusSystems

osv-dev@googlegroups.com

#osv on FreeNode