Introduction
Single Node Slowdown
Managing Virtual Resources
Conclusion

Big Data Processing Architecture

Mike Sevilla, Ike Nassi, Kleoni Ioannidou

October 23, 2012
Big Data...

... is data that is large and complex

- **Social Networks**
  
  Ex: Facebook activity\(^1\) & storage\(^2\) (100PB)

- **Medical Institutions**
  
  Ex: UPMC\(^3\) (3.2PB)

- **Sensor Analysis**
  
  Ex: CERN\(^4\) (40TB / second)

---

\(^1\)Harvey: “Facebook Reaches 1 Billion Active Users”, September 2012

\(^2\)Harris: “For the Future of Big Data Startups, Look at Facebook”, September 2012

\(^3\)Horowitz: “IBM, Oracle to Build UPMC Data Warehouse for Personalized Medicine”, May 2012

\(^4\)Couts: Breaking down Big Data and Internet in the age of variety, volume, and velocity”, September 2012
Industry, Government, & Research Trends

Things are increasing:

- Data
- Demand
  - → processing
  - → storage
- Technology
  - → density of cores
  - → amount of memory
Accommodating These Trends

Q: What is a system supposed to do when there is too much data?
Accommodating These Trends

Q: What is a system supposed to do when there is too much data?

A: Scale the system
Accommodating These Trends

Q: What is a system supposed to do when there is too much data?

A: Scale the system
   - out
     - ++ nodes to the system
     - modify applications
Q: What is a system supposed to do when there is too much data?

A: Scale the system
   - Out
     - ++ nodes to the system
     - modify applications
   - Up
     - ++ resources to the system
     - modify the system instead of the application
Accommodating Industry Trends

Q: What is a system supposed to do when there is too much data?

A: Scale the system

- **out**
  - ++ nodes to the system
  - modify applications

- **up**
  - ++ resources to the system
  - modify the system instead of the application

- treated as mutually exclusive
- scaling out is easier... BUT!
A Solution: Virtualization

Scale up: modify @ system level

- **virtualize** additional types of resources
  - processors, memory, block-based, network
A Solution: Virtualization

Scale up: modify @ system level

- **virtualize** additional types of resources
  → processors, memory, block-based, network

What does this enable?

- simplicity - programming model
- scalability - economically
- performance - self-tunable & less overhead
A Solution: Virtualization

Scale up: modify @ system level

- **virtualize** additional types of resources
  - processors, memory, block-based, network

What does this enable?

- simplicity - programming model
- scalability - economically
- performance - self-tunable & less overhead

Less overhead... huh?
Research Questions

Q: How do we efficiently manage virtual resources?

- schedule, move, and organize
- virtual $\mapsto$ physical resources
  mismatch $\Rightarrow$ access violation
Research Questions

Q: How do we efficiently manage virtual resources?
- schedule, move, and organize
- virtual $\mapsto$ physical resources
  mismatch $\Rightarrow$ access violation

Q: What slows down single node systems?
- which events?
- what information to expose?
- how to detect?
- can we predict?
- can we resolve automatically?

Q: Can existing tools answer these questions?
What slows down single nodes?

Which events?

What information to expose?

(Event Identification)

(Event Information)
What slows down single nodes?

Which events?

What information to expose?

(Event Identification)

(Event Information)
What slows down single nodes?

Which events?

What information to expose?

**Methodology & Exp. Setup**

- we target
  - frequent events
  - events w/ slow response
- 1 node
  - 2CPUS @ 3.47GHz, 6 cores, 192GB RAM
  - Linux v.2.6.32
- workloads: programs

**Single Node**

![Diagram of single node system](image)
Using OProfile to Identify Events

**OProfile** samples hardware performance counters

- (12.2%) `clear_page_c`
- (7.5%) `page_fault`
- (4.9%) `intel_idle`
- (3.1%) `mem_cgroup_charge_common`
- (2.6%) `get_page_from_freelist`
- (2.2%) `copy_page_c`
- (2.1%) `__mem_cgroup_commit_charge`
- (2.1%) `copy_user_generic_string`
- (1.8%) `zap_pte_range`
- (1.7%) `__rmqueue`
- (1.6%) `free_pcpages_bulk`
- (1.5%) `__d_lookup_rcu`
- (1.5%) `find_busiest_group`
- (1.3%) `do_page_fault`
- (1.3%) `release_pages`
- (52.8%) other

**Limitation**: no details on data movement & parameters
Using LTTng-2.0 to Collect Information

**LTTng-2.0** captures kernel events using tracers

<table>
<thead>
<tr>
<th>Listing 1: LTTng-2.0 output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timestamp = 09:44:41.812793841, delta = +0.000000261, name = ust_tests_hello:tptest, stream.packet.context = { cpu_id = 1 }, event.fields = {...,stringfield=&quot;test&quot;,floatfield=2222,...}</code></td>
</tr>
<tr>
<td><code>timestamp = 09:44:41.812795566, delta = +0.000001535, name = sys_poll, stream.packet.context = { cpu_id = 0 }, event.fields = {...,nfds = 3,timeout_msecs = -1 }</code></td>
</tr>
</tbody>
</table>

Tell us the **program/symbol**, **location**, and **parameters**

**Limitation**: no details on frequency or latency
Research Questions

Q: *How do we efficiently manage virtual resources?*

- schedule, move, and organize
- virtual $\mapsto$ physical resources
- mismatch $\Rightarrow$ access violation

Q: *What slows down single node systems?*

- which events?
- what information to expose?
- how to detect?
- can we predict?
- can we resolve automatically?

Q: *Can existing tools answer these questions?*
Access Violations

Examine single node behavior

- How does the **hardware** react to access violations?
  - register values, memory accesses, interrupt activity, ...
- How does the **operating system** react to access violations?
  - function calls, assembly instructions, traps, ...

![Diagram showing Tracing, OS, What happens here, E, PM, and Profiling]

16 / 23
Using FTrace to Examine Operating System Reaction

**FTrace** is a function tracer

---

**Listing 2: Valid Reference**

```c
1 do_page_fault() {
2    down_read_trylock(&mm->mmap_sem)
3    find_vma(mm, address)
4
5    // valid, allocate page frame
6    handle_mm_fault()
7 }
8 .
9 .
```

**Listing 3: Invalid Reference**

```c
1 do_page_fault() {
2    down_read_trylock(&mm->mmap_sem)
3    find_vma(mm, address)
4
5    // invalid, handle faulty addr.
6    bad_area(mm, address)
7    __bad_area_nosemaphore()
8
9    show_signal_msg()
10 }
```

**Limitation:** access violations re-routed by the kernel
Use **GRUB2/GDB** to examine memory without OS intervention

**Linux Blocks Memory Accesses**

```plaintext
(gdb) target exec test
(gdb) break 6
Breakpoint 1 at 0x80483ba: file test.c, line 6.
(gdb) run
Starting program: /home/msevilla/Code/scratch/test
Breakpoint 1, main () at test.c:6
6 int *p = (int *) 0xFFFFFFFF;
(gdb) x 0x8000
0x8000: Cannot access memory at address 0x8000
(gdb)
0x8004: Cannot access memory at address 0x8004
(gdb)
0x8008: Cannot access memory at address 0x8008
(gdb)
```

**Debug GRUB2 Remotely**

[Diagram showing a setup where GRUB can be debugged remotely using a null modem adapter and female-female serial cable]
Tool for Hardware Reaction

Use **GRUB2/GDB** to examine memory without OS intervention

---

**Examine Registers**

```
(gdb) info registers
eax 0x3fe3d14d 1071894861
ecx 0x3fe3d594 1071895956
edx 0x0 0
ebx 0x3fe3d594 1071895956
esp 0x7fda4 0x7fda4
ebp 0x7fdac 0x7fdac
esi 0x3fe3d020 1071894560
edi 0x0 0
ebp 0x3fe3d0ea 0x3fe3d0ea <grub_cmd_hello+10>
eflags 0x210046 [ PF ZF RF ID ]
```
Future Work...

- Use tools to profile/trace big data applications
  - in-memory database (TimesTen, Hana)
- Model big data memory/resource architecture
  - Capture event traces
  - Feed events to simulator
- Insert hooks: transfer control
We learned:
- Linux is very robust
- all tools have limitations
- about events
- about access violations
Questions?

Special thanks to Kleoni for help with this presentation.

Work funded by SAP
Using the Linux src to Examine Boot-up Reaction

Memory (num_physpages)

The detected memory is:
- set by:
  - `init_memory_mapping()`
  - `initmem_init()`
  - `e820_end_of_ram_pfn()`
- updated by:
  - `mtrr_trim_memory()`
  - `init_memory_mapping()`
  - `highmem_pfn_init()`
  - `find_low_pfn_range()`

Flow of Linux Boot-up