Bolt: Faster Reconfiguration in Operating Systems

Sankaralingam Panneerselvam
Michael M. Swift
Nam Sung Kim

University of Wisconsin, Madison, WI
Benefits of Processor Reconfiguration

Energy Savings

Performance Inflation

Power Constraints

Disaggregated Server Architectures

VM Scaling (e.g. IBM DLPAR)
Support for Reconfiguration

• Linux: **Hotplug** mechanism
  – Designed as a
    • Reliability mechanism to *remove faulty cores*
    • Testing mechanism to vary *number of cores*
  – Uncommon operation
  – Latency in the order of *tens or hundreds of milliseconds*
Faster Reconfiguration

• Processors enter/exit sleep state in 10 -100μs
  – Major overhead is contributed by the software
• Faster reconfiguration can enable high performance and energy efficient systems

How to achieve faster reconfiguration?
We built Bolt to reconfigure up to 20x faster than Hotplug.
Outline

• Background on Hotplug

• Design of Bolt

• Evaluation Results
**Hotplug Overview**

- OS subsystems (like scheduler, watchdog …) **subscribe** for CPU reconfiguration events
  - 50 different subscriptions from various kernel subsystems

- **Notifications** sent during reconfiguration event (Addition-Online/Removal-Offline)
  - Thread migration
  - Watchdog enable/disable
CPU Offline

1. Notifications sent at Hotplug stages
2. Cleanup done during notifications
3. stop_machine halts the entire system
4. Global Hotplug lock serializes reconfiguration

- Run in the context of CPU to be removed
  * CPU is moved to offline state after the DYING notification is sent
1. Different message notifications are sent at various stages of Hotplug.

2. Notifications allow OS to initialize software structures for the new CPU.

3. Global Hotplug lock is held and thus all reconfiguration events are serialized.

- Run in the context of CPU to be added

* Startup Interrupt is sent to wake up the CPU.
What makes Hotplug slow?

• Assumption: Reconfiguration events are uncommon

• Three limitations
  – **Synchronous**: All work done on critical path
  – **Pessimistic**: Clear/Initialize s/w structures
  – **Serial**: No reconfiguration of multiple cores
Outline

• Background on Hotplug

• Design of Bolt

• Evaluation Results
Bolt Overview

• Change in assumption
  – Online ↔ Offline state changes are frequent

• Bolt handles Hotplug limitations
  – **Asynchronous**: Defer work from the critical path
  – **Optimistic**: Re-use software structures
  – **Parallel**: Bulk Interface to reconfigure multiple core at once
Bolt

• Bolt is built on existing Hotplug infrastructure
  – Retains subscription/notification mechanism

• Classify subsystems notification handler into
  – **Critical**: Need to be executed immediately for correctness (E.g. Migrating threads)
  – **Non-Critical**: Immediate action is not necessary (E.g. Freeing a memory structure)
Categorization of Hotplug Operations

Classify ~50 Hotplug callbacks into one of these categories

- Hardware management (Microcode update)
- Destroy/Create
- Park/Unpark
- Global Bitmask Updates
- Software structures cleanup/initialize (per-cpu structures)
- State Migration (Threads from runqueue)
- Re-organization
Critical Operations

• Handled immediately for correctness

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Migration</td>
<td>threads in runqueue, softirqs in per-cpu queue</td>
</tr>
<tr>
<td>Hardware Management</td>
<td>microcode update, updating MTRR registers</td>
</tr>
<tr>
<td>Bitmask updates</td>
<td>global structures like cpu_online_mask</td>
</tr>
<tr>
<td>Re-organization</td>
<td>scheduler domain structures</td>
</tr>
</tbody>
</table>

• Critical operations are handled synchronously
Non-Critical Operations

<table>
<thead>
<tr>
<th>Category</th>
<th>How handled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software structures</td>
<td>Memory cleanup deferred to a master thread</td>
</tr>
<tr>
<td></td>
<td>Memory cleanup deferred to a master thread</td>
</tr>
<tr>
<td></td>
<td>sysfs file access is checked during open(...) invocation</td>
</tr>
<tr>
<td>Thread operations</td>
<td>Parking/Unparking made asynchronous</td>
</tr>
<tr>
<td></td>
<td>Thread re-use without destroying them</td>
</tr>
</tbody>
</table>

• ~30 out of 50 callbacks were identified as non-critical operations

• Bolt *removes them from critical path* but retaining the interface
1. **Aggregation** - Avoid redundant execution

The `cpumask` passed to `down_prepare`, `dead`, and `post_dead` messages.

2. **Parallel execution** – DYING message is handled in parallel by all cores in `cpumask`.

Offline
Bulk Interface

CPU_UP_PREPARE

CPU_STARTING

CPU_STARTING

CPU_ONLINE

Global Lock

Online

Aggregation

Parallel Execution

Aggregation
Outline

• Background on Hotplug

• Design of Bolt

• Evaluation Results
Experimental Platform

• Two platforms
  – x86: 4-core Intel i5-2500K (Sandybridge)
  – ARM: big.LITTLE Exynos 5410 (Odroid development board)

• Experiments: Latency measured for Hotplug operations
  – Done in an idle system at the highest processor frequency
[CPU Offline] Bolt vs. Hotplug

Time in Milliseconds

- **x86**: 13.3
- **x86 - Bolt**: 0.53
- **Exynos**: 7.5
- **Exynos - Bolt**: 0.57

CPU states:
- DOWN PREPARE
- RCU
- DEAD
- STOP MACHINE (DYING)
- PARK
- POST DEAD

Usenix ATC 2015
[CPU Online] Bolt vs. Hotplug

Time in Milliseconds

- x86: 4.03
- x86 - Bolt: 0.38
- Exynos: 12.6
- Exynos - Bolt: 0.58

Usenix ATC 2015
Bulk Interface (x86)

Time in Milliseconds

Number of cores reconfigured

<table>
<thead>
<tr>
<th>Hotplug</th>
<th>Bolt - Sequential</th>
<th>Bolt - Bulk Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.08</td>
<td>0.77</td>
<td>0.71</td>
</tr>
<tr>
<td>4.03</td>
<td>2.05</td>
<td>1.2</td>
</tr>
<tr>
<td>13.3</td>
<td>26.5</td>
<td>39.2</td>
</tr>
<tr>
<td>0.53</td>
<td>0.58</td>
<td>0.6</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Conclusion

• Hotplug is currently used as a reconfiguration mechanism

• Bolt reduces reconfiguration latency by separating critical from non-critical operations

• Bolt’s principles can be applied to any reconfiguration mechanism
Questions