Predictable Communication and Migration in the Quest-V Separation Kernel

Ye Li, Richard West, Zhuoqun Cheng, Eric Missimer

Boston University
Background

- Quest-V Separation Kernel \([\text{WMC’13, VEE’14}]\)
  - System is partitioned into a collection of \textit{sandboxes}
    - Each \textit{sandbox} encapsulates one or more CPU cores, region of memory, and subset of I/O devices
    - Like a distributed system on a chip
    - Explicit communication channels b/w sandboxes for data exchange and address space migration

- Useful in safety-critical systems where component failures can be isolated and recovered w/o full system reboots
Quest-V uses H/W virtualization for resource partitioning

Each partition, or sandbox, manages its resources w/o involving trusted hypervisor

- cf. (RT)-Xen, XtratuM, PikeOS, WindRiver/Mentor Graphics Hypervisor, etc.

Hypervisor typically only needed for bootstrapping system + managing comms channels

Eliminates costly hypervisor traps

- ~1500 clock cycles VM-Exit/Enter Xeon E5506
Quest-V Overview

Sandbox 1

VCPU

PCPU(s)

IO Devices

Monitor

Sandbox 2

VCPU

PCPU(s)

IO Devices

Monitor

Sandbox M

VCPU

PCPU(s)

IO Devices

Monitor

... Communication + Migration

Sandbox

Address Space

Thread
Problem

- Multi-threaded apps may need to communicate
- Threads may need to be migrated between sandboxes
  - for load balancing, schedulability, resource affinity
- How do we guarantee predictable communication?
- How do we migrate threads w/o violating service guarantees...
  - of migrating threads?
  - of threads in destination sandbox?
- Complicated by each sandbox having own local scheduler and clock
Predictability

- VCPUs for budgeted real-time execution of threads and system events (e.g., interrupts)
  - Threads mapped to VCPUs
  - VCPUs mapped to physical cores
- Sandbox kernels perform scheduling on assigned cores
  - Avoid VM-Exits to Monitor – eliminate cache/TLB flushes
VCPU Scheduling Framework

Address Space

Threads

Main VCPUs

I/O VCPUs

PCPUs (Cores)
VCPU Scheduling Framework

- VCPUs are divided into two classes:
  - **Main VCPUs** for conventional tasks
  - **I/O VCPUs** for I/O event threads (e.g. ISRs)

- See RTAS’11 for more details

- In this work focus is on Main VCPUs
  - Implement Sporadic Server policy
  - \( C \) budget every \( T \) period
Inter-sandbox communication in Quest-V relies on message passing primitives built on shared memory.

- Monitors update EPT mappings to establish private message passing channels between specific sandboxes.

- The lack of both a global clock and global scheduler creates challenges for a system requiring strict timing guarantees.
Communication Model

- A comms channel is *half duplex* w/ capacity $B$ bytes
- A sender thread ($\tau_s$) is mapped to a VCPU $V_s$ with parameters $C_s$ and $T_s$
- A receiver thread ($\tau_r$) is mapped to a VCPU $V_r$ with parameters $C_r$ and $T_r$
- $\tau_s$ sends an $N$-byte msg at $\delta_s$ time units per byte
- $\tau_r$ replies with an $M$-byte msg at $\delta_r$ time units per byte
- Before replying, $\tau_r$ consumes $K$ units of processing time
- What is the worst case round-trip comms delay $\Delta_{WC}$?
Case 1: All messages fit in one channel slot \((M, N \leq B)\)

\[
\Delta_{WC}(N, M) = S(N) + (T_s - C_s) + R(N, M) + (T_r - C_r) + S(M) + (T_s - C_s)
\]

\[
S(N) = \left\lfloor \frac{N \cdot \delta_s}{C_s} \right\rfloor \cdot T_s + (N \cdot \delta_s) \mod C_s
\]

\[
R(N, M) = \left\lfloor \frac{[N + M] \cdot \delta_r + K}{C_r} \right\rfloor \cdot T_r + ([N + M] \cdot \delta_r + K) \mod C_r
\]
Inter-Sandbox Communication

- 5 different experiments to predict the worst-case round-trip communication time
- Core i5-2500K 4-core CPU, 8GB RAM
- $M = N = B = 4$KB, $\delta_s, \delta_r$ calculated w/ caches disabled

<table>
<thead>
<tr>
<th>Case #</th>
<th>Sender VCPU</th>
<th>Receiver VCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>20/100</td>
<td>2/10</td>
</tr>
<tr>
<td>Case 2</td>
<td>20/100</td>
<td>20/100</td>
</tr>
<tr>
<td>Case 3</td>
<td>20/100</td>
<td>20/130</td>
</tr>
<tr>
<td>Case 4</td>
<td>20/100</td>
<td>20/200</td>
</tr>
<tr>
<td>Case 5</td>
<td>20/100</td>
<td>20/230</td>
</tr>
</tbody>
</table>

Table: Parameters C(ms)/T(ms)
Case 2: One way communication and messages take multiple slots ($N > B$ and $M = 0$)

- Can be used to estimate address space transfer delay during migration

\[
\Delta'_{WC}(N) = \left\lceil \frac{N}{B} \right\rceil \cdot (S(B) + (T_s - C_s) + R(B, 0) + (T_r - C_r))
\]
Inter-Sandbox Communication

- One-way communication experiments to send 4MB messages through a 4KB channel
- \( N = 4\text{MB}, M = 0, B = 4\text{KB} \)

<table>
<thead>
<tr>
<th>Case #</th>
<th>Sender VCPU</th>
<th>Receiver VCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>20/50</td>
<td>20/50</td>
</tr>
<tr>
<td>Case 2</td>
<td>10/100</td>
<td>10/100</td>
</tr>
<tr>
<td>Case 3</td>
<td>10/100</td>
<td>10/50</td>
</tr>
<tr>
<td>Case 4</td>
<td>10/100</td>
<td>10/200</td>
</tr>
<tr>
<td>Case 5</td>
<td>5/100</td>
<td>5/130</td>
</tr>
<tr>
<td>Case 6</td>
<td>10/200</td>
<td>10/200</td>
</tr>
</tbody>
</table>

Table: VCPU Parameters
Predictable Migration

- Quest-V supports the migration of VCPUs and associated address spaces for several reasons:
  - To balance loads across sandboxes
  - To guarantee the schedulability of VCPUs and threads
  - For closer proximity to needed resources such as I/O devices
Predictable Migration

- Quest-V predictable migration interface:

```c
bool vcpu_migration(uint32_t time, int dest, int flag);
```

- The migration function is non-blocking
- `flag` can be set to MIG_STRICT, MIG_RELAX, or 0
Migration Criteria

- If VCPU $V_m$ issues a migration request with MIG_STRICT flag, the following must hold:

$$E_m \geq \Delta_{mig}$$

- $E_m$ is the relative time of the next event for VCPU $V_m$, which is either a replenishment or wakeup
- $\Delta_{mig}$ is the migration cost
Migration with Message Passing

- Transfer a thread’s address space and VCPU information using messages passed over a communication channel
- An estimate of the worst-case migration cost requires:
  - The execution time ($\delta_f$) and cost ($\Delta_f$) of fragmenting the migrated state into a sequence of messages
  - The communication delay to send the messages ($\Delta_t$)
  - The execution time ($\delta_a$) and cost ($\Delta_a$) of re-assembling the transferred state at the destination
Migration with Message Passing

- Assume the sender migration thread is associated with VCPU $V_s$ and receiver migration thread is associated with VCPU $V_r$.
- The worst-case migration cost is:

\[
\Delta_{mig} = \Delta_f + \Delta'_{WC} + \Delta_a
\]

\[
\Delta_t = \Delta'_{WC}
\]

\[
\Delta_f = \left\lfloor \frac{\delta_f}{C_s} \right\rfloor \cdot T_s + \delta_f \mod C_s + T_s - C_s
\]

\[
\Delta_a = \left\lfloor \frac{\delta_a}{C_r} \right\rfloor \cdot T_r + \delta_a \mod C_r + T_r - C_r
\]
Migration with Message Passing

- Migration with message passing usually spans numerous migration VCPU periods ($\Delta'_{WC}$ is very large)
- This makes it difficult to satisfy a migration request with MIG_STRICT flag
- Quest-V monitors support migration through direct memory copy to dramatically reduce overhead
Migration with Direct Memory Copy

1. Push quest_tss address(es) to destination and resume schedule
2. Copy quest_tss structure(s)
3. Move addr space and VCPU from source
Migration with Direct Memory Copy

- With direct memory copy, the worst-case migration cost can be defined as:

\[ \Delta_{mig} = \left\lfloor \frac{\delta_m}{C_r} \right\rfloor \cdot T_r + \delta_m \mod C_r + T_r - C_r \]

- \( C_r \) and \( T_r \) are the budget and period of the migration thread’s VCPU in destination sandbox

- \( \delta_m \) is the execution time to copy an address space and its quest_tss data structures to the destination
Clock Synchronization

- Quest-V sandboxes use Local APIC Timers and Time Stamp Counters for time related activities
- These time sources are not guaranteed to be synchronized
- Quest-V adjusts time for each migrating address space to compensate for clock skew

\[ \delta_{ADJ} = TSC_d - TSC_s - 2 \times RDTSC_{cost} - IPI_{cost} \]

- \( TSC_d \) and \( TSC_s \) are the destination and source TSCs
- \( RDTSC_{cost} \) and \( IPI_{cost} \) are the average costs of reading a TSC and sending an IPI
Predictable Migration

To verify the predictability of the Quest-V migration framework, we designed several experiments.

<table>
<thead>
<tr>
<th>VCPU (C/T)</th>
<th>Sandbox 1</th>
<th>Sandbox 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>20/100</td>
<td>Shell</td>
<td>Shell</td>
</tr>
<tr>
<td>10/200 (10/50)</td>
<td>Migration Thread</td>
<td>Migration Thread</td>
</tr>
<tr>
<td>20/100</td>
<td>Canny</td>
<td></td>
</tr>
<tr>
<td>20/100</td>
<td>Logger</td>
<td>Logger</td>
</tr>
<tr>
<td>10/100</td>
<td>Comms 1</td>
<td>Comms 2</td>
</tr>
</tbody>
</table>

Table: Migration Experiment VCPU Setup
Predictable Migration

- Canny is migrated using message passing
- Migration requested with MIG_RELAX flag

![Graph showing Canny and Comms 1 & 2 with time in seconds and data rate in fps or x1000 KB/s]
Predictable Migration

- Canny is migrated using direct memory copy
- Migration requested with MIG STRICT flag

Graph showing migration overhead and time (seconds) for Canny, Comms 1, Comms 2, and Migration.
Predictable Migration

- For comparison, the same experiment was repeated without a dedicated migration thread.
Conclusions

- Quest-V supports predictable inter-sandbox communication and migration
- Quest-V operates like a chip-level distributed system
  - Static partitioning of machine resources
  - Migration for load balancing and resource affinity
  - Comms channels built on protected shared memory
- Message passing versus direct memory copy
- Future? Lazy migration of hot pages of address spaces
- Extend comms across different network transport media
Thank You!

For more details, please visit:

www.questos.org