Experiments in Enabling Automated Migration Testing

Amit Shah
Red Hat
22 Oct 2013
Thanks

- Juan
- Markus
- Orit
Migration

Source

Destination
Migration

Source

$ qemu <cmdline>

Destination

$ qemu <cmdline> 
-incoming <...>
Migration

Source

$ qemu <cmdline>

(qemu) migrate <...

Destination

$ qemu <cmdline> -incoming <...>
Migration

Source

Wire Format / savevm Format

Destination

$ qemu <cmdline>

(qemu) migrate <...>

$ qemu <cmdline>
-incoming <...>
The Problem

- Very easy to break migration between different QEMU versions
- By the time breakage is found, it can be too late to fix
  - e.g. released versions have bugs
- Migration may erroneously succeed
- When migration fails, it can't tell much about the reason
- Impact of such breakages can be huge
- Currently, no way to prove migration compatibility
Can We Identify Problems Earlier?
Ways to Check Migration Breakage

- Dynamic
- Static
Ways to Check Migration Breakage: Dynamic
Dynamic Checking

- Needs a live state
- Only acts on devices present in the VM at the time of migration
- Subsections present based on device state
savevm file sanity checks
savevm format is...
savevm Format is a Blackbox

- savevm format is ugly – Juan Quintela
- Doesn't contain machine type information
- Doesn't contain array or list sizes
- Array sizes change based on configuration
- “01”: binary data or a new section?
  - No size information -> prone to lose sync
- savevm file by itself is indecipherable
savevm Format Checks: Solutions?

- Can send vmstate info before the actual vmstate so some analysis possible
- Fixing entirely essentially means 'qemu --incoming <...>' is sufficient to accept migration
- Need to fix QEMU and the world before we can bring this to reality
Ways to Check Migration Breakage: Static
Static Checking

• Does not need a live state
• All compiled-in devices can be checked
• Bugs can be identified before committing patches
• Exact cause of potential failure can be pinpointed
Migration Metadata Checks
Migration Metadata

```c
static const VMStateDescription vmstate_acpi = {
    .name = "piix4_pm",
    .version_id = 3,
    .minimum_version_id = 3,
    .minimum_version_id_old = 1,
    .load_state_old = ACPI_LOAD_OLD,
    .post_load = vmstate_acpi_post_load,
    .fields = (VMStateField []) {
        VMSTATE_PCI DEVICE(parent_obj, PIIX4PMState),
        VMSTATE_UINT16(ar.pml.evt.sts, PIIX4PMState),
        VMSTATE_UINT16(ar.pml.evt.en, PIIX4PMState),
        VMSTATE_UINT16(ar.pml.cnt.cnt, PIIX4PMState),
        VMSTATE_STRUCT(apm, PIIX4PMState, 0, vmstate_apm, APMState),
        VMSTATE_TIMER(ar.tmr.timer, PIIX4PMState),
        VMSTATE_INT64(ar.tmr.overflow_time, PIIX4PMState),
        VMSTATE_STRUCT(ar.gpe, PIIX4PMState, 2, vmstate_gpe, ACPIGPE),
        VMSTATE_STRUCT(pci0_status, PIIX4PMState, 2,
            vmstate_pci_status, struct pci_status),
        VMSTATE_END_OF_LIST()
    }
};
```
Migration Metadata: Subsections

```c
.subsections = (VMStateSubsection[]) {
    {
        .vmsd = &usbredir_bulk_receiving_vmstate,
        .needed = usbredir_bulk_receiving_needed,
    },
```

- Optional sections
- Based on device state at time of migration
- 'needed' check at source to determine eligibility
Migration Metadata Checker

- Look at migration metadata
- Output metadata information for all devices
- Diff that output from one QEMU binary with another
- Tell whether migration from version x to y will break, (and how).
Sample Output

piix4_pm
  Name: apm
    Version id: 0
    Size: 224
    Type: Struct
  APM State
    Name: apmc
      Version id: 0
      Size: 1
      Type: Single
    Name: apms
      Version id: 0
      Size: 1
      Type: Single
Bugs a Static Checker Can Flag
Bugs a Static Checker Can Flag

```c
static const VMStateDescription vmstate_acpi = {
    .name = "piix4_pm",
    .version_id = 3,
    .minimum_version_id = 3,
    .minimum_version_id_old = 1,
    .load_state_old = ACPI_LOAD_OLD,
    .post_load = vmstate_acpi_post_load,
    .fields = (VMStateField []) {
        VMSTATE_PCIDEVICE(parent_obj, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.sts, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.en, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.cnt.cnt, PIIX4PMState),
        VMSTATE_STRUCT(apm, PIIX4PMState, 0, vmstate_apm, ACPIState),
        VMSTATE_TIMER(ar.tmr.timer, PIIX4PMState),
        VMSTATE_UINT64(ar.tmr.overflow_time, PIIX4PMState),
        VMSTATE_STRUCT(ar.gpe, PIIX4PMState, 2, vmstate_gpe, ACPIGPE),
        VMSTATE_STRUCT(pci0_status, PIIX4PMState, 2, vmstate_pci_status, 
            struct pci_status),
        VMSTATE_END_OF_LIST()
    }
};
```
Bugs a Static Checker Can Flag

```c
static const VMStateDescription vmstate_acpi = {
    .name = "piix4_pm",
    .version_id = 4,
    .minimum_version_id = 3,
    .minimum_version_id_old = 1,
    .load_state_old = ACPI_LOAD_OLD,
    .post_load = vmstate_acpi_post_load,
    .fields = (VMStateField []) {
        VMSTATE_PCI_DEVICE(parent_obj, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.sts, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.en, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.cnt.cnt, PIIX4PMState),
        VMSTATE_STRUCT(apm, PIIX4PMState, 0, vmstate_apm, APMState),
        VMSTATE_TIMER(ar.tmr.timer, PIIX4PMState),
        VMSTATE_INT64(ar.tmr.overflow_time, PIIX4PMState),
        VMSTATE_STRUCT(ar.gpe, PIIX4PMState, 2, vmstate_gpe, ACPIGPE),
        VMSTATE_STRUCT(pci0_status, PIIX4PMState, 2, vmstate_pci_status, struct pci_status),
    }
};
```

- If version changes without any other field changes, flag it as a bug
Bugs a Static Checker Can Flag

```c
static const VMStateDescription vmstate_acpi = {
    .name = "piix4_pm",
    .version_id = 3,
    .minimum_version_id = 3,
    .minimum_version_id_old = 1,
    .load_state_old = ACPI_LOAD_OLD,
    .post_load = vmstate_acpi_post_load,
    .fields = (VMStateField []) {
        VMSTATE_PCI_DEVICE(parent_obj, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.sts, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.en, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.cnt.cnt, PIIX4PMState),
        VMSTATE_STRUCT(apm, PIIX4PMState, 0, vmstate_apm, APMState),
        VMSTATE_TIMER(ar.tmr.timer, PIIX4PMState),
        VMSTATE_INT64(ar.tmr.overflow_time, PIIX4PMState),
        VMSTATE_STRUCT(ar.gpe, PIIX4PMState, 2, vmstate_gpe, ACPIGPE),
        VMSTATE_STRUCT(pci0_status, PIIX4PMState, 2, vmstate_pci_status,
                        struct pci_status),
        VMSTATE_END_OF_LIST()
    }
};
```
Bugs a Static Checker Can Flag

static const VMStateDescription vmstate_acpi = {
    .name = "piix4_pm",
    .version_id = 3,
    .minimum_version_id = 3,
    .minimum_version_id_old = 1,
    .load_state_old = ACPI_LOAD_OLD,
    .post_load = vmstate_acpi_post_load,
    .fields = (VMStateField []) {
        VMSTATE_PCI DEVICE(parent_obj, PIIX4PMState),
        VMSTATE_UINT32(ar.pm1.evt.sts, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.evt.en, PIIX4PMState),
        VMSTATE_UINT16(ar.pm1.cnt.cnt, PIIX4PMState),
        VMSTATE_STRUCT(apm, PIIX4PMState, 0, vmstate_apm, ACPIState),
        VMSTATE_TIMER(ar.tmr.timer, PIIX4PMState),
        VMSTATE_INT64(ar.tmr.overflow_time, PIIX4PMState),
        VMSTATE_STRUCT(ar.gpe, PIIX4PMState, 2, vmstate_gpe, ACPIGPE),
        VMSTATE_STRUCT(pci0_status, PIIX4PMState, 2, vmstate_pci_status, struct pci_status),
        VMSTATE_END_OF_LIST()
    }
};

- If some field changes without version change, flag it as a bug
Bugs a Static Checker Can Flag

- Subsections
  - If one is introduced and version is bumped, flag it

<table>
<thead>
<tr>
<th>Src</th>
<th>Dest</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present</td>
<td>Present</td>
<td>Normal subsection processing</td>
</tr>
<tr>
<td>Present</td>
<td>Absent</td>
<td>Warning: this device might break; audit all ways this device gets in this state</td>
</tr>
<tr>
<td>Absent</td>
<td>Present</td>
<td>Nothing to worry about</td>
</tr>
</tbody>
</table>
Substructures
Substructures

piix4_pm
- Name: apm
  - Version id: 0
  - Size: 224
  - Type: Struct

APM State
- Name: apmc
  - Version id: 0
  - Size: 1
  - Type: Single

- Name: apms
  - Version id: 0
  - Size: 1
  - Type: Single
Substructures

piix4_pm
  Name: apm
  Version id: 0
  Size: 224
  **Type: Struct**

APM State
  Name: apmc
  Version id: 0
  Size: 1
  Type: Single

Name: apms
  Version id: 0
  Size: 1
  Type: Single
Substructures

piix4_pm
  Name: apm
    Version id: 0
    Size: 224
    Type: Struct
APM State
  Name: apmc
    Version id: 0
    Size: 1
    Type: Single
  Name: apms
    Version id: 0
    Size: 1
    Type: Single
Substructures

piix4_pm
  Name: apm
  Version id: 0
  Size: 224
  Type: Struct

APM State
  Name: apmc
  Version id: 0
  Size: 1
  Type: Single

Name: apms
  Version id: 0
  Size: 1
  Type: Single

const VMStateDescription vmstate_apm =
{
  .name = "APM State",
  .version_id = 1,
  .minimum_version_id = 1,
  .minimum_version_id_old = 1,
  .fields = (VMStateField[]) {
    VMSTATE_UINT8(apmc, APMState),
    VMSTATE_UINT8(apms, APMState),
    VMSTATE_END_OF_LIST()
  }
};
Substructures: Problems

- Version field in a substructure is ignored
- Currently need to add a new subsection for each new field
- Prone to be broken easily

```c
const VMStateDescription vmstate_apm = {
  .name = "APM State",
  .version_id = 1,
  .minimum_version_id = 1,
  .minimum_version_id_old = 1,
  .fields = (VMStateField[]) {
    VMSTATE_UINT8(apmc, APMState),
    VMSTATE_UINT8(apms, APMState),
    VMSTATE_END_OF_LIST()
  }
};
```
Next Steps
To Do

- Not everything is converted to vmstate yet
- Fix substructures
- We need something like
  - `qemu -M pc-1.6 -dump-vmstate`
    - (and / or)
    - QMP command to dump vmstate
- Place vmstate metadata for released versions and corresponding machine types somewhere in tests/, run checks before committing / before releasing
To Do

- Longer term
  - Make vmstate definitions an IDL
    - Helps in keeping vmstate info in one place
    - Helps reviewing
    - Newer state fields added / removed are immediately identified
Thank You