Checkpoint Internals

Arun

Dec 19, 2013

Topics of Discussion

Checkpoint Basics

- VMX groups
- Monitor Modules
- VMX <-> VMM Interactions

State Machine

- Overview
- Save
  - STUN, SAVE_SYNC, QUIESCE and SAVE
  - CONTINUE_SYNC
- Restore
  - UNSTUN, RESTORE and RESTORE_SYNC

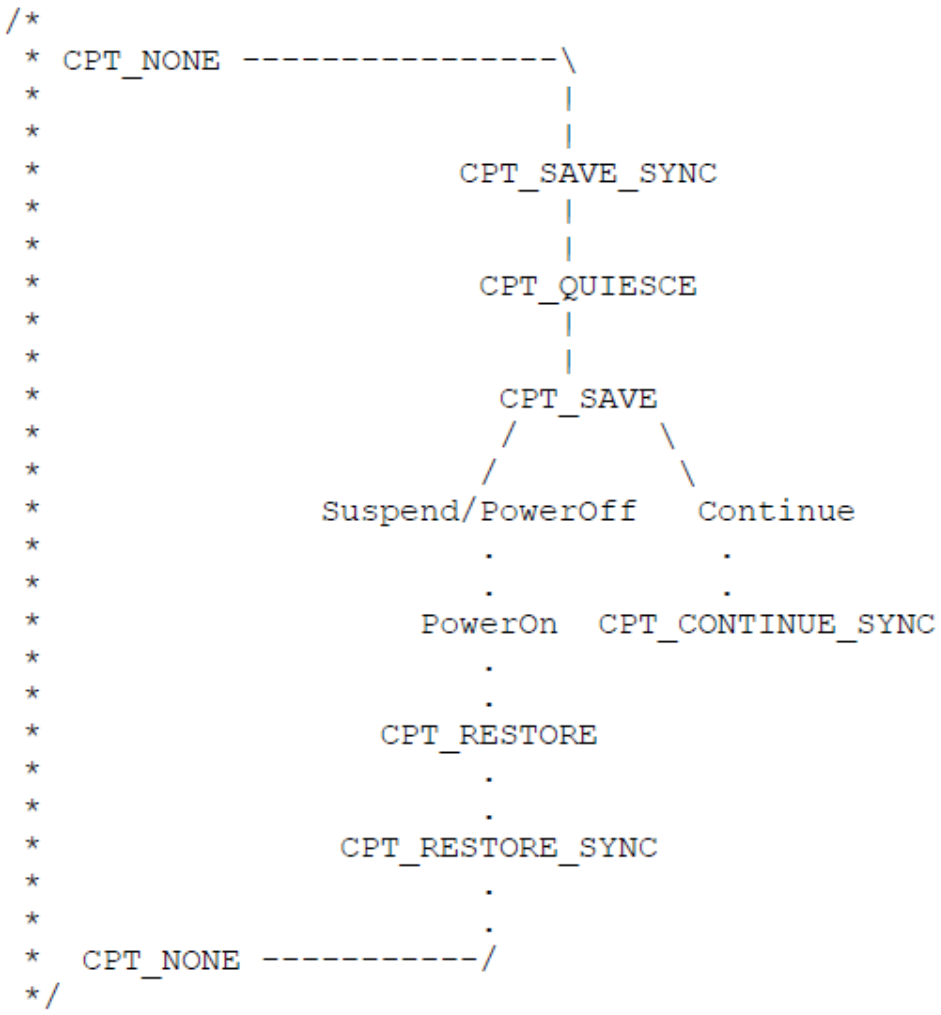
Checkpoint Overview

VMX/VMM interaction, Groups, Monitor modules

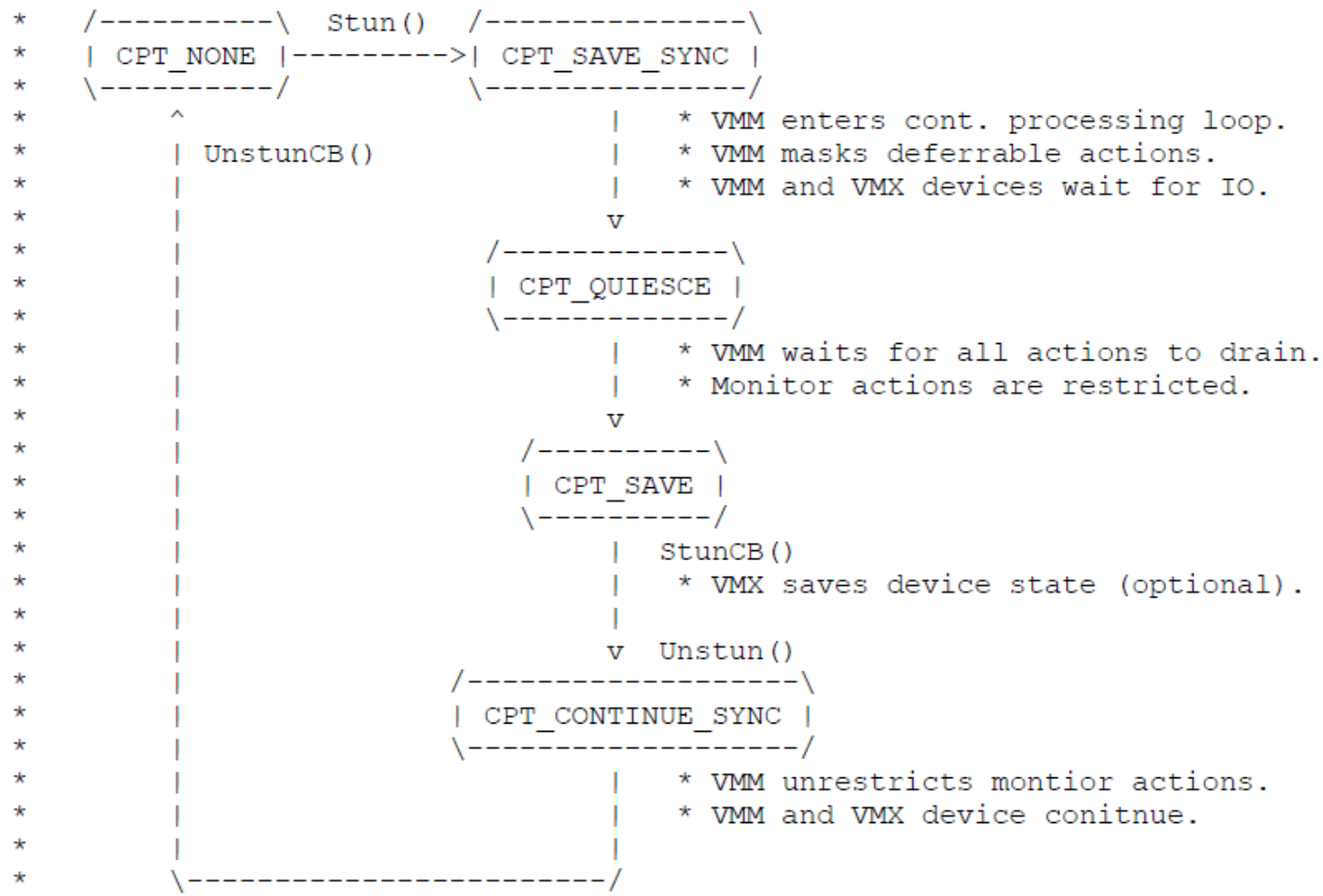
Checkpoint

- STUN
  - Suspend the execution of the VM
  - Stop execution of vCPUs
- SYNC
  - Quiesce I/O operations
  - Drain Monitor actions and restrict actions
- SAVE
  - Serialize device states
  - Save
- RESTORE
  - During Power On detect the VM is restoring and pause VM
  - VMX restores the groups followed by VMM restoring modules
- RESTORE SYNC
  - VMX syncs the group, unpauses the VM then VMM syncs the monitor modules.

Checkpoint State Machine



Checkpoint Save State Machine



Checkpoint Groups

- A checkpoint group is a device or VMX module
- It provides a framework to register and unregister callback functions
  - During sync, checkpoint, save etc

```
typedef struct CptGroup {
    char                *name;
    CptFunctions        funcs;
    void                *clientData;
    DeviceHdr           *deviceHdr;
    Bool                baseCptDone;
    Bool                missingGroup;
} CptGroup;

typedef struct CptFunctions {
    CptCallback                 *checkpoint;
    CptCallback                 *sync;
    CptSyncLazyCallback         *syncLazy;
    CptErrCallback              *errcb;
    CptSaveNeededCallback       *saveNeeded;
    CptMigrateTo                *migrateTo;
} CptFunctions;
```

Checkpoint Groups

2013-12-12T04:01:44.881Z| vcpu-0| I120: saving Checkpoint
2013-12-12T04:01:44.882Z| vcpu-0| I120: saving GuestVars
2013-12-12T04:01:44.883Z| vcpu-0| I120: saving cpuid
2013-12-12T04:01:44.883Z| vcpu-0| I120: saving cpu
2013-12-12T04:01:45.318Z| vcpu-0| I120: saving BusMemSample
2013-12-12T04:01:45.318Z| vcpu-0| I120: saving UUIDVMX
2013-12-12T04:01:45.318Z| vcpu-0| I120: saving StateLogger
2013-12-12T04:01:45.319Z| vcpu-0| I120: saving memory
2013-12-12T04:01:45.321Z| vcpu-0| I120: saving MStats
2013-12-12T04:01:45.322Z| vcpu-0| I120: saving Snapshot
2013-12-12T04:01:45.323Z| vcpu-0| I120: saving pic
2013-12-12T04:01:45.328Z| vcpu-0| I120: saving FTCpt
2013-12-12T04:01:45.328Z| vcpu-0| I120: saving ide1:0
2013-12-12T04:01:45.328Z| vcpu-0| I120: saving scsi0:0

Monitor Modules

- Monitor modules are h/w devices in the monitor
- There are also software module like Migrate, license etc
- MonModule
  - Module Table in VMM
  - Framework for registering callback functions for sync and checkpoint

```
typedef struct MonModule {
   MonModuleInitFunc          init;          // Init, called on boot vcpu.
   MonModuleInitPerThreadFunc initPerThread; // Per-thread init, called on all vcpus after init()
                                             // including on boot vcpu
   MonModuleSyncFunc          sync;          // Cpt-sync, called on boot vcpu.
   MonModuleCptFunc           checkpoint;    // Checkpoint, called on boot vcpu.
} MonModule;
```

- Boot vcpu
  - vcpu 0 on which the machine is booted
  - Responsible for all checkpoint actions

Monitor Modules

```
static MonModule monModules[] = {

54 #ifdef VMX86_SERVER
55 /* Migrate_Sync must come before SVGA_Sync. PR 1109402 */
56 { NULL,          Migrate_InitPerThread,    Migrate_Sync,     NULL},
57 #endif

58 { NULL,          DeviceStats_InitPerThread, NULL,             NULL},
59 { MainMem_Init,  NULL,                     NULL,             NULL},
60 { Rom_Init,      NULL,                     NULL,              NULL},
61 { VGA_Init,      VGA_InitPerThread,        NULL,              NULL},
62 { SVGA_Init,     NULL,                     SVGA_Sync,        NULL},
63 { VIDE_Init,      VIDEVMK_InitPerThread,   VIDEVMK_Sync,     NULL},

64 #ifdef VMX86_SERVER
65 { FTCpt_Init,     FTCpt_InitPerThread, NULL,
```

VMX <-> VMM Communication

- Checkpoint is close interaction between VMX and VMM
- They work in a loop going back and forth
- For every state change both have to complete actions

- VMX -> VMM
  - Monitor actions

- VMM -> VMX
  - User RPC

- For SAVE, in every state transition, VMM runs first followed by VMX
- For RESTORE, VMX runs first followed by VMM

Checkpoint SAVE

Stun, Save Sync, Quiesce, Save, Continue Sync, Unstun

STUN

- Suspend execution of the VM
- Checkpoint_Stun
- VMX level

MigrateDoSuspend

Checkpoint_Stun(usageMode, CPT_STUN_IDLE, MigrateStunCallback, NULL);

CheckpointStun

1. Notify clients on checkpointing
2. prepare checkpoint descriptor
3. Set state to CPT_SAVE_SYNC

   cptd->usageMode = usageMode;

   cptd->requestSync = TRUE;
4. VMX_Pause(TRUE)
5. Notify Poll to change state to CPT.

Confidential
13

VMX Pause

- VMX Sends a pause action to the monitor
- MONACTION_PAUSE leads to cross call which leads to more actions processing via CAP.
- So stun is achieved via combination of monitor actions and cross calls.

```
VMXPauseImpl
    Vmcore_SetPause
        MonitorLoop_PauseMonitor
            MonitorLoopSendPauseAction
                MonitorAction_AddIdemVcpuid(MONACTION_PAUSE, BOOT_VCPU_ID);
```

VMM level Stun

- STOP_CROSSCALL => First stop all vCPUs with a cross call
- CrossCall CB => Request monitorAction subsystem to stop executing guest instr.
- Invoke - CAP on all vCPUS until ActionProcPredicate returns FALSE.

MONACTION_PAUSE

MonitorPauseHandler

- ST_StopVCPUs(ST_Pause)

  CrossCall_Invoke -> STOP_CROSSCALL

  ST_HandleCrossCall
- TimeTracker_MonitorPause
- CrossCall_InvokeAllCB(MonitorPauseCCHandler)

  MonitorAction_ContinueProcessing -> Stop guest instructions.

  MonitorAction_AddIdem(MONACTION_CONTINUE_PROCESSING);

  MonitorActionContinueHandler

  In a loop process all actions queued.
- ST_ReleaseVCPUs

SAVE_SYNC

- For all states, VMX initiates processing with monitor actions
- VMM executes the desired action followed by VMX

- VMM
  - Stop vCPUs
  - Invoke cross call to Execute Sync functions for all modules
  - Followed by CPU sync
  - RPC to VMX
  - Release vCPUs

- VMX
  - Go over each VMX group and call its corresponding sync function
  - Move to next state and post an action to VMM

QUIESCE

- Lite Phase

- VMM
  - Stop vCPUs
  - CAP – Continuous action processing mode
  - Restrict actions which can be posted
  - Check if all vCPUs are quiesced
  - RPC to VMX
  - Release vCPUs

- VMX
  - No –op!
  - Simple move the state machine forward to SAVE
  - Post an action to VMM to move to next state

SAVE

- Heavy Phase
- VMM
  - Cross call for CheckpointDoPreOrPostCptSave (DT, MMU, TLB. BUSMEM etc)
  - Cross call for CheckpointExecuteCptCB (CPU)
  - Switch Swapping to host swapper
  - Go over Module table and call checkpoint functions
  - Get back to VMX via USER RPC
- VMX
  - Call stun call back - MigrateStunCallback
  - Stun call back has to decide if it needs a SAVE
  - Migrate sets dumper and calls CheckpointSave
  - VMX go over groups and calls groups checkpoint function

CONTINUE_SYNC

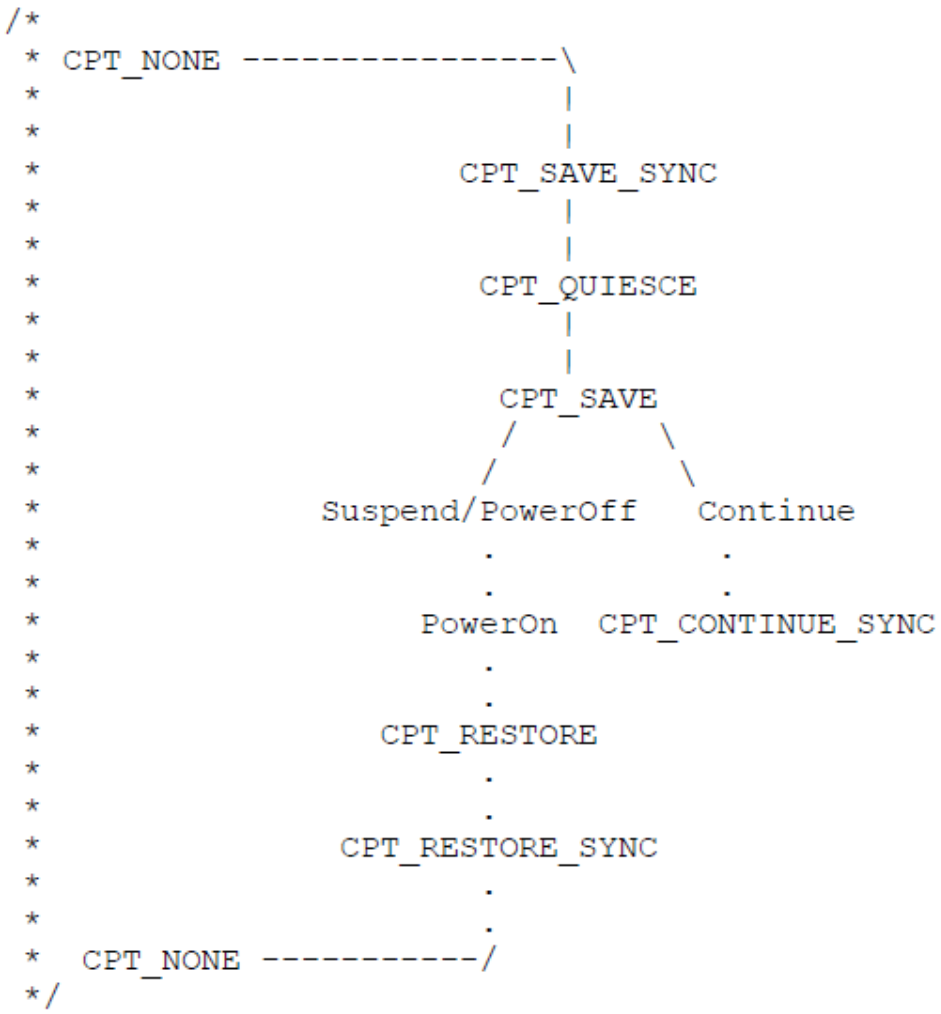
- Optional in case of migration
  - Does not execute if the migration succeeds
  - Only in case of resume, migrate calls Checkpoint_Unstun

- VMM
  - Undo all restrictions on monitor actions
  - Switch swapper
  - Cross call for CheckpointDoPreOrPostCptSave (DT, MMU, BUSMEM, etc)
  - Cross call for CheckpointExecuteSyncCB (CPU and Module table)
  - Get back to VMX
- VMX
  - Go over checkpoint groups and call sync function
  - Unpause VMX!
  - Notify poll
  - Call unstun callback i.e MigrateUnstunCallback

Checkpoint RESTORE

Restore and Restore Sync

Checkpoint State Machine



RESTORE

- Basic idea restore checkpoint and then synchronize.
- VMX first, VMM next.

- Magic is in VMX - module power on table.
  - Phase 1: Early setup in power on table.
    - Right after BIOS power on.
    - Set the VM is restoring from checkpoint state.

  - Phase 2.5: Device setup - Migrate_PowerOn
    - Receive all checkpoint data!

  - Phase 6 : Late initialization - Checkpoint_LatePowerOn
    - Actual restore process happens here!

VMX – module PowerOn table

1. Phase 1 : Early Setup /* Configure */
   - BIOS, Checkpoint, Memsched, MonitorLoop
2. Phase 1.5 : Late early setup
   - Monitor, VMX log, pshare, stats etc
3. Phase 2 : Open the monitor
   - Monitor power on
4. Phase 2.5 : Device support /* Receive checkpoint data */
   - Disk, Migrate, FTcpt, Snapshot etc
5. Phase 3 : Random Cruft
   - Timer tracker, backdoor, MKS, SoundMUX, CptOpts, ACPI, Guest RPC etc
6. Phase 4 : Device Support - Nvman, tools TclUtil etc
7. Phase 5 : Old devices - DMA, CMOS, Keyboard etc
8. Phase 6 : Late Initialization /* Restore */
   - Checkpoint_LatePowerOn, FSR, Monitor_LatePowerOn, FTCpt_LatePowerOn etc

Restore – Early setup

- Early in power on table setup the VM to restore from checkpoint
  - Checkpoint_PowerOn
    CheckpointConfigure

```
1. if (Migrate_Restoring) {                          /* Migration Type = MIGRATE_FROM */
     Migrate_UseMigDumper(cptd);                     /* Use Migration Dumper */
     cptd->mode                  = CPT_RESTORE;
     cptd->usageMode             = CPT_USAGE_MIGRATE / MIGRATE_FT
     cptd->priv->resumedSession  = TRUE;

2. if (CheckpointSuspended) {                        /* If checkpoint file exists */
     cptd->mode                  = CPT_RESTORE;
     cptd->usageMode             = CPT_USAGE_SUSPEND;
```

  - Register checkpoint device
  - Restore checkpoint group to set memory size.
    - For Migration, we don't have the checkpoint so just use memory size from VMX config file.
  - **Pause VMX**

Device setup – Receive checkpoint data

- In power on table, as part of device setup, Migrate_PowerOn runs
- Receive checkpoint from source and store in MigrateInfo => Migrate_CptCache
- VMX

  Migrate_PowerOn
  MigrateWaitForData
  Checkpoint_MigrateRestoreCheckpointGroup

- Vmkernel
  - Source sends chekpoint data to dest in a WRITE_DATA message.

  VMotionRecv_Helper
  VMotionRecvHeplerProcessMessages
  VMotionRecvHandleMessage
  VMotionRecv_ExecHandler
  VMotionRecv_WriteCptData
  Migrate_CptCacheWrite
  Copy data to mi -> cptCache

VMX RESTORE

- Checkpoint_LatePowerOn
  - CheckpointRestore
    - Dumper_BeginRestore
    - Estimate checkpoint size
    - Restore Memory: Check if main memory was stored separately (Not migrating!)
    - Restore cpu, cpuid groups
    - Then go over groups and call their checkpoint function

```
FOR_EACH_GROUP(group) {
        Dumper_BeginRestoreGroup
         group->funcs.checkpoint
         Dumper_EndRestoreGroup
```

VMM RESTORE

- VMM Restore

```
CheckpointProcessRestoreState
    CheckpointExecuteCptCB
      If (VCPUSet_AtomicIsMember(&cptVCPUs, CurVcpuid())) {
          CPU_Restore();
              Intr_Restore();
              DT_Restore();
              FPU_Restore();
             VHV_Restore();
              Priv_Restore();        /* Privelage instructions */
            Smm_Restore();     /* system mgmt mode on Intel cpus */
            HyperV_Restore();
       } else {
            CPU_InitAddedCPU();   /* Initialize a hot added CPU */
       }
```

- Go over module table and call checkpoint function

```
ModuleTable_Checkpoint(curCptState);                    /* Run checkpoint handlers for modules */
 for (i = 0; i < ARRAYSIZE(monModules); i++) {
        monModules[i].checkpoint(mode);
```

- Gets back to VMX through RPC

RESTORE_SYNC

- VMX
  - Go over each group and execute its Sync function
  - Report Power on
  - UnPause VMX

  - VMM
    - Starts running - MonitorInitWork
      - CPU_StartRunning
    - If checkpoint is restoring
      - CheckpointExecuteSyncCB
        - Module table sync functions i.e Migrate_Sync
        - CPU_Sync
        - User RPC to get back to VMX

Thank You