目 录

1 前言

在提出 snapshot 优化方案之前，我先分析一下 snapshot 的流程，以便之后优化快照时思路更加清晰。

2 snapshot 的流程

2.1 快照入口函数

```
# nova/api/openstack/compute/servers.py Controller._action_create_image()
def _action_create_image(self, req, id, body):
    # id是instance id
    context = req.environ['nova.context']
    # body == {u'createImage': {u'name': u'snap1', u'metadata': {}}}
    entity = body.get("createImage", {})

    image_name = entity.get("name")

    ...

    props = {}
    metadata = entity.get('metadata', {})

    try:
        props.update(metadata)
    ...
    # 根据context和req得到Instance类
    instance = self._get_server(context, req, id)

    bdms = objects.BlockDeviceMappingList.get_by_instance_uuid(
                context, instance.uuid)

    try:
        # 判断root分区是否是volume
        if self.compute_api.is_volume_backed_instance(context, instance,
                                                        bdms):
            img = instance['image_ref']
            if not img:
                properties = bdms.root_metadata(
                        context, self.compute_api.image_api,
                        self.compute_api.volume_api)
                image_meta = {'properties': properties}
            else:
                image_meta = self.compute_api.image_api.get(context, img)

            # Snapshot the given volume-backed instance
            image = self.compute_api.snapshot_volume_backed(
                                                    context,
                                                    instance,
                                                    image_meta,
                                                    image_name,
                                                    extra_properties=props)
        else:
            # 做快照的正常流程
            image = self.compute_api.snapshot(context,
                                              instance,
                                              image_name,
                                              extra_properties=props)
    ...

    # 以下代码的功能: build location of newly-created image entity
    image_id = str(image['id'])
    url_prefix = self._view_builder._update_glance_link_prefix(
            req.application_url)
    image_ref = os.path.join(url_prefix,
                             context.project_id,
                             'images',
                             image_id)

    resp = webob.Response(status_int=202)
    resp.headers['Location'] = image_ref
    return resp
```

2.2 使用 RPC 与 nova-compute 服务通信

```
# nova/compute/api.py API.snapshot()
def snapshot(self, context, instance, name, extra_properties=None):
    # instance: nova.db.sqlalchemy.models.Instance
    # name: name of the snapshot
    # extra_properties: dict of extra image properties to include
    #                   when creating the image.
    # returns: A dict containing image metadata

    # 函数功能: Create new image entry in the image service.
    #          This new image will be reserved for the compute
    #          manager to upload a snapshot or backup.
    image_meta = self._create_image(context, instance, name,
                                    'snapshot',
                                    extra_properties=extra_properties)

    # 更改instance的task_state
    instance.task_state = task_states.IMAGE_SNAPSHOT_PENDING
    # 调用Instance.save()方法更改数据库
    instance.save(expected_task_state=[None])

    self.compute_rpcapi.snapshot_instance(context, instance,
                                          image_meta['id'])

    return image_meta
```

```
# nova/compute/rpcapi.py ComputeAPI.snapshot_instance()
def snapshot_instance(self, ctxt, instance, image_id):
    # server: the destination host for a message.
    # server == instance['host']
    cctxt = self.client.prepare(server=_compute_host(None, instance),
            version=version)
    cctxt.cast(ctxt, 'snapshot_instance',
               instance=instance,
               image_id=image_id)
```

2.3 nova-compute 中的快照动作

```
# nova/compute/manager.py ComputeManager.snapshot_instance()
def snapshot_instance(self, context, image_id, instance):
    # context: security context
    # instance: a nova.objects.instance.Instance object
    # image_id: glance.db.sqlalchemy.models.Image.Id
    try:
        # 修改instance的task_state
        instance.task_state = task_states.IMAGE_SNAPSHOT
        # 调用Instance.save()方法更改数据库
        instance.save(
                    expected_task_state=task_states.IMAGE_SNAPSHOT_PENDING)
    ...

    self._snapshot_instance(context, image_id, instance,
                            task_states.IMAGE_SNAPSHOT)
```

_snapshot_instance() 函数如下：

```
# nova/compute/manager.py ComputeManager._snapshot_instance()
def _snapshot_instance(self, context, image_id, instance,
                       expected_task_state):
    # self.driver.get_info(instance)["state"]
    current_power_state = self._get_power_state(context, instance)
    try:
        # 修改instance的power_state
        instance.power_state = current_power_state
        instance.save()
        if instance.power_state != power_state.RUNNING:
            state = instance.power_state
            running = power_state.RUNNING
        def update_task_state(task_state,
                              expected_state=expected_task_state):
            instance.task_state = task_state
            instance.save(expected_task_state=expected_state)
        # 调用LibvirtDriver.snapshot()函数实现快照功能
        self.driver.snapshot(context, instance, image_id,
                             update_task_state)
        # 记录instance状态
        instance.task_state = None
        instance.save(expected_task_state=task_states.IMAGE_UPLOADING)
    ...
```

2.4 实现快照功能的核心函数

```
    def snapshot(self, context, instance, image_id, update_task_state):
        try:
            # 调用virConnect.lookupByName()返回virDomain对象
            virt_dom = self._lookup_by_name(instance['name'])
        ...

        base_image_ref = instance['image_ref']

        # 得到instance的image的相关数据
        base = compute_utils.get_image_metadata(
            context, self._image_api, base_image_ref, instance)

        # Retrieves the information record for a single disk image by image_id
        snapshot = self._image_api.get(context, image_id)

        # 通过virtDomain.XMLDesc(0)得到xml配置文件，从而得到instance的磁盘路径
        disk_path = libvirt_utils.find_disk(virt_dom)
        # 使用“qemu-img info disk_path”获得磁盘信息
        source_format = libvirt_utils.get_disk_type(disk_path)

        image_format = CONF.libvirt.snapshot_image_format or source_format

        # NOTE(bfilippov): save lvm and rbd as raw
        if image_format == 'lvm' or image_format == 'rbd':
            image_format = 'raw'

        # metadata = {'is_public': False,
        #             'status': 'active',
        #             'name': snp_name,
        #             'properties': {
        #                            'kernel_id': instance['kernel_id'],
        #                            'image_location': 'snapshot',
        #                            'image_state': 'available',
        #                            'owner_id': instance['project_id'],
        #                            'ramdisk_id': instance['ramdisk_id'],
        #                            }
        #             }
        metadata = self._create_snapshot_metadata(base,
                                                   instance,
                                                   image_format,
                                                   snapshot['name'])

        # 获得快照的名称
        snapshot_name = uuid.uuid4().hex

        # LIBVIRT_POWER_STATE = {
        #     VIR_DOMAIN_NOSTATE: power_state.NOSTATE,
        #     VIR_DOMAIN_RUNNING: power_state.RUNNING,
        #     VIR_DOMAIN_BLOCKED: power_state.RUNNING,
        #     VIR_DOMAIN_PAUSED: power_state.PAUSED,
        #     VIR_DOMAIN_SHUTDOWN: power_state.SHUTDOWN,
        #     VIR_DOMAIN_SHUTOFF: power_state.SHUTDOWN,
        #     VIR_DOMAIN_CRASHED: power_state.CRASHED,
        #       VIR_DOMAIN_PMSUSPENDED: power_state.SUSPENDED,
        # }
        # 调用virDomain.info()返回[state, maxMemory, memory, nbVirtCPU, cpuTime]
        state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]

        # 动态快照要求QEMU 1.3 and Libvirt 1.0.0
        # Instances with LVM encrypted ephemeral storage只支持静态快照
        if (self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
                                  MIN_QEMU_LIVESNAPSHOT_VERSION,
                                  REQ_HYPERVISOR_LIVESNAPSHOT)
             and source_format not in ('lvm', 'rbd')
             and not CONF.ephemeral_storage_encryption.enabled):
            live_snapshot = True
            try:
                # 终止虚拟机磁盘上的active block job
                virt_dom.blockJobAbort(disk_path, 0)
            ...
        else:
            live_snapshot = False

        if state == power_state.SHUTDOWN:
            live_snapshot = False

        # virDomain.managedSave() does not work for LXC
        # 如果是静态快照，需要执行virDomain.managedSave()函数
        if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
            if state == power_state.RUNNING or state == power_state.PAUSED:
                # 卸载虚拟机的pci设备
                self._detach_pci_devices(virt_dom,
                    pci_manager.get_instance_pci_devs(instance))
                # 关闭SR-IOV端口
                self._detach_sriov_ports(context, instance, virt_dom)
                # This method will suspend a domain and save its memory contents to
                     a file on disk.
                virt_dom.managedSave(0)

        # 返回Qcow2类
        # snapshot_backend是nova.virt.libvirt.imagebackend.Qcow2 object
        snapshot_backend = self.image_backend.snapshot(instance,
                disk_path,
                image_type=source_format)

        ...

        update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
        # snapshot_directory == /var/lib/nova/instances/snapshots
        snapshot_directory = CONF.libvirt.snapshots_directory
        # 确保有这个目录存在
        fileutils.ensure_tree(snapshot_directory)
        with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                # 得到快照路径
                out_path = os.path.join(tmpdir, snapshot_name)
                if live_snapshot:
                    os.chmod(tmpdir, 0o701)
                    # 动态快照的步骤如下:
```

```
                # disk_path为/var/lib/nova/instances/vm-uuid/disk
                # out_path为/var/lib/nova/instances/snapshots/snapshot_name
                # 首先调用“qemu-img create -f qcow2 -o backing_file=disk_path的backing_file,size=disk_path的virtual_size out_path.delta”创建镜像
                # 然后调用domain.blockRebase(disk_path, disk_delta, 0,
                #           libvirt.VIR_DOMAIN_BLOCK_REBASE_COPY |
                #           libvirt.VIR_DOMAIN_BLOCK_REBASE_REUSE_EXT |
                #           libvirt.VIR_DOMAIN_BLOCK_REBASE_SHALLOW)函数将disk_path的内容拷贝给out_path.delta
                # 然后使用“qemu-img convert”命令将out_path.delta拷贝到out_path
                # 这条命令在拷贝过程中，会先把后端镜像和增量镜像合并，然后在拷贝到另外一个镜像文件中，所以会比较久
                self._live_snapshot(virt_dom, disk_path, out_path,
                                    image_format)
            else:
                # 使用“qemu-img convert”命令将虚拟机磁盘拷贝到out_path
                # 这条命令在拷贝过程中，会先把后端镜像和增量镜像合并，然后在拷贝到另外一个镜像文件中，所以会比较久
                # 如果不想新建立一个镜像，只是想做个快照，那么这是可以改进的一点
                snapshot_backend.snapshot_extract(out_path, image_format)
        finally:
            new_dom = None
            # NOTE(dkang): because previous managedSave is not called
            #              for LXC, _create_domain must not be called.
            if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
                if state == power_state.RUNNING:
                    # 这种情况new_dom == virt_dom
                    new_dom = self._create_domain(domain=virt_dom)
                elif state == power_state.PAUSED:
                    # 这种情况new_dom == virt_dom.createWithFlags(libvirt.VIR_DOMAIN_START_PAUSED)
                    new_dom = self._create_domain(domain=virt_dom,
                            launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
                if new_dom is not None:
                    # 安装原有的pci设备
                    self._attach_pci_devices(new_dom,
                        pci_manager.get_instance_pci_devs(instance))
                    # 开启SR-IOV端口
                    self._attach_sriov_ports(context, instance, new_dom)

        # Upload that image to the image service
        with libvirt_utils.file_open(out_path) as image_file:
            self._image_api.update(context,
                                   image_id,
                                   metadata,
                                   image_file)
```

3 snapshot 优化方案

3.1 最基本的情况

3.1.1 最基本的情况的描述

假设现在我们想迅速地备份一个虚拟机，而且现在这个虚拟机的镜像链只有两个，也就是只有一个当初创建虚拟机用的后端镜像和一个增量文件。

在这里我说一下我们想实现哪些功能：

1. 虚拟机原先有一个磁盘镜像，它的路径是 disk_path，现在我们要把它作为后端镜像，并且在它的基础上创建一块增量镜像。
2. 将后端镜像上传到镜像服务器上，以备之后根据这块镜像创建虚拟机。

原先我们 horizon 上已经有一个创建快照的按键，现在我希望把这个按键名称改为“创建完整的虚拟机镜像”。然后在添加一个按键，名称为“迅速备份虚拟机”。

因为从网页命令到底层函数的消息的传递我不了解，这里就不提，而是把自己在底层函数的开发方案的实现方法提出来。这里的修改方法是根据原先的 snapshot 流程更改的，如果不熟悉 snapshot 流程，可以参看上一节。

当然不局限于我这种实现方法，如果有更好的实现方法，当然可以把我这个方案完全推翻。

3.1.2 入口函数的改动

我的想法是新的快照动作与原先的快照动作共用一个入口函数，只是传入的 body 参数改一下。

本来传入的 body 参数是：

```
body == {u'createImage': {u'name': u'snap1', u'metadata': {}}}
```

现在新的快照动作对应的 body 参数可以是：

```
body == {u'snapshot': {u'name': u'snap1', u'metadata': {}}}
```

```
# nova/api/openstack/compute/servers.py Controller._action_create_image()
def _action_create_image(self, req, id, body):
    # id是instance id
    context = req.environ['nova.context']
    # body == {u'createImage': {u'name': u'snap1', u'metadata': {}}}
    entity = body.get("createImage", {})

    # 新增加的代码
```

```
        action = "createImage"
        if not entity:
            entity = body.get("snapshot", {})
            action = "snapshot"

        image_name = entity.get("name")

        # 更改的代码
        if not image_name:
            msg = _("%(action) entity requires name attribute") % {"action":action}
            raise exc.HTTPBadRequest(explanation=msg)

        props = {}
        metadata = entity.get('metadata', {})
        common.check_img_metadata_properties_quota(context, metadata)
        try:
            props.update(metadata)
        except ValueError:
            msg = _("Invalid metadata")
            raise exc.HTTPBadRequest(explanation=msg)

        instance = self._get_server(context, req, id)

        bdms = objects.BlockDeviceMappingList.get_by_instance_uuid(
                    context, instance.uuid)

        try:
            # 因为对linux的device mapping机制不了解，这里先不动
            if self.compute_api.is_volume_backed_instance(context, instance,
                                                          bdms):
                img = instance['image_ref']
                if not img:
                    properties = bdms.root_metadata(
                            context, self.compute_api.image_api,
                            self.compute_api.volume_api)
                    image_meta = {'properties': properties}
                else:
                    image_meta = self.compute_api.image_api.get(context, img)

                # Snapshot the given volume-backed instance
                image = self.compute_api.snapshot_volume_backed(
                                                       context,
                                                       instance,
                                                       image_meta,
                                                       image_name,
                                                       extra_properties=props)
            else:
                # 做快照的正常流程
                # 为这个函数添加一个action的参数
                image = self.compute_api.snapshot(context,
                                                  instance,
                                                  image_name,
                                                  extra_properties=props,
                                                  action=action)
        except exception.InstanceInvalidState as state_error:
            # 修改了这里的代码
            common.raise_http_conflict_for_instance_invalid_state(state_error,
                action)
        except exception.Invalid as err:
            raise exc.HTTPBadRequest(explanation=err.format_message())

        image_id = str(image['id'])
        url_prefix = self._view_builder._update_glance_link_prefix(
                req.application_url)
        image_ref = os.path.join(url_prefix,
                                 context.project_id,
                                 'images',
                                 image_id)

        resp = webob.Response(status_int=202)
        resp.headers['Location'] = image_ref
        return resp
```

3.1.3 修改 RPC 与 nova-compute 服务通信部分的代码

在上面的 _action_create_image() 函数中调用 compute_api.snapshot() 函数时，我们增加了一个 action 参数，对它的修改如下：

```
    # 之所以将action参数设置为关键字参数，是怕有其他地方调用snapshot，关键字参数能
        把影响降到最小
    def snapshot(self, context, instance, name, extra_properties=None, action="
        createImage"):
        image_meta = self._create_image(context, instance, name,
                                        'snapshot',
                                        extra_properties=extra_properties)
        instance.task_state = task_states.IMAGE_SNAPSHOT_PENDING
        instance.sace(excepted_task_state=[None])

        # 仍然是多传递一个参数action
        self.compute_rpcapi.snapshot_instance(context, instance, image_meta['id'],
            action=action)

        return image_meta
```

相应地修改 compute.rpcapi.py 文件中的 snapshot_instance() 函数：

```
    # 增加一个关键字参数action
    def snapshot_instance(self, ctxt, instance, image_id, action="createImage"):
        version = '3.0'
        cctxt = self.client.prepare(server=_compute_host(None, instance),
                version=version)
        cctxt.cast(ctxt, 'snapshot_instance', instance=instance, image_id=image_id,
             action=action)
```

随后在 nova-compute 服务中修改相应的函数。

3.1.4 在 nova-compute 中修改相应的快照动作

在 compute.manager.py 文件中添加 snapshot_instance 函数，只是增加一个关键字参数：

```
    def snapshot_instance(self, context, image_id, instance, action="createImage"):
        try:
            instance.task_state = task_states.IMAGE_SNAPSHOT
            instance.save(exceptd_task_state=task_states.IMAGE_SNAPSHOT_PENDING)
        except exception.InstanceNotFound:
            LOG.debug("Instance not found, could not set state %s for instance",
                  task_states.IMAGE_SNAPSHOT, instance=instance)
            return
        except exception.UnexpectedDeletingTaskStateError:
            LOG.debug("Instance being deleted, snapshot cannot continue", instance=
                  instance)
            return

        # 仍然是多传递一个参数action
        self._snapshot_instance(context, image_id, instance, task_states.
            IMAGE_SNAPSHOT, action=action)
```

仍然是将 _snapshot_instance() 函数多增加一个关键字参数 action，同时因为这个函数已经开始调用 LibvirtDriver 类中的函数，在这里开始我觉得就可以和原先的快照流程分开了。

也就是在 LibvirtDriver 类中新增加一个 snapshot_overlay 函数，然后在 _snapshot_instance 函数根据 action 参数选择跳转到 snapshot() 函数还是跳转到 snapshot_overlay 函数。

```
    def _snapshot_instance(self, context, image_id, instance, exceptd_task_state,
        action="createImage"):
        context = context.elevated()

        current_power_state = self._get_power_state(context, instance)
        try:
            instance.power_state = current_power_state
            instance.save()

            LOG.audit(_('instance snapshotting'), context=context, instance=
                  instance)

            if instance.power_state != power_state.RUNNING:
                state = instance.power_state
                running = power_state.RUNNING
                LOG.warn(_('trying to snapshot a non-running instance: '
                           '(state: %(state)s expected: %(running)s)'),
                           {'state': state, 'running': running},
                           instance=instance)

            self._notify_about_instance_usage(context, instance, "snapshot.start")

            def update_task_state(task_state, excepted_state=expected_task_state):
                instance.task_state = task_state
                instance.save(expected_task_state=expected_state)

            # 修改的代码：根据action选择LibvirtDriver中的参数
            if action == "createImage":
                self.driver.snapshot(context, instance, image_id, update_task_state)
            else:
                self.driver.snapshot_overlay(context, instance, image_id, update_task_state)

            instance.task_state = None
            instance.save(expected_task_state=task_states.IMAGE_UPLOADING)

            self._notify_about_instance_usage(context, instance, "snapshot.end")
        except (exception.InstanceNotFound,
                exception.UnexpectedDeletingTaskStateError):
            msg = 'Instance disappeared during snapshot'
            LOG.debug(msg, instance=instance)
            try:
                iamge_service = glance.get_default_image_service()
                image = image_service.show(context, image_id)
                if image['status'] != 'active':
                    image_service.delete(context, image_id)
            except Exception:
                LOG.warning(_("Error while trying to clean up image %s"),
                            image_id, instance=instance)
        except exception.ImageNotFound:
            instance.task_state = None
            instance.save()
            msg = _("Image not found during snapshot")
            LOG.warn(msg, instance=instance)
```

3.1.5 新增加一个核心函数 snapshot_overlay

根据上一节的描述我们知道，我们需要在 LibvirtDriver 类中增加一个 snapshot_overlay 函数，用于实现我们想实现的功能。

在这里我再说一下我们想实现哪些功能：

1. 虚拟机原先有一个磁盘镜像，它的路径是 disk_path，现在我们要把它作为后端镜像，并且在它的基础上创建一块增量镜像。
2. 将后端镜像上传到镜像服务器上，以备之后根据这块镜像创建虚拟机。

针对功能一的实现细节，我有以下的想法：

1. 首先将虚拟机磁盘镜像改名，加个后缀".base"，它的路径也就变成了 disk_path.base。
2. 然后根据这个镜像文件创建增量文件，这个增量文件的名字和原先磁盘镜像的名字相同，所以它的路径也就是 disk_path。

3. 重启虚拟机，虚拟机就会开始使用这个增量文件。

针对可能的疑问，我这里作出回答:

```
问: 虚拟机运行时，将磁盘镜像更名是否会影响它的运行?
答: 我在kvm中进行过操作，发现是不会影响虚拟机的运行的。

问: 虚拟机磁盘镜像更名后，虚拟机的数据改动是否会保存?
答: 会保存，而且是保存到原先的镜像文件中。

问: 根据原先的虚拟机镜像创建增量镜像后，虚拟机的数据改动是否会保存到增量镜像
    中?
答: 数据的改动会保存到当前的后端镜像中，也保存到增量镜像中。
    不过需要知道的是，当虚拟机重启后，虚拟机开始使用这个增量镜像，所以虚拟机的
        数据改动会保存到增量镜像中，不会再保存到原先的镜像文件中。
```

针对功能二我有想补充的话:

```
当后端镜像传到镜像服务器以后，我们应该知道一个事实，这个后端镜像也是有自己的
    backing file的。如果想要根据这个后端镜像创建原先的虚拟机，必须保证它的
    backing file是存在的。
```

新增加 snapshot_overlay 函数如下:

```python
def snapshot_overlay(self, context, instance, image_id, update_task_state):
    try:
        virt_dom = self._lookup_by_name(instance['name'])
    except exception.InstanceNotFound:
        raise exception.InstanceNotRunning(instance_id=instance['uuid'])

    base_image_ref = instance['image_ref']

    base = compute_utils.get_image_metadata(context, self._image_api,
        base_iamge_ref, instance)

    snapshot = self._image_api.get(context, image_id)

    disk_path = libvirt_utils.find_disk(virt_dom)
    source_format = libvirt_utils.get_disk_type(disk_path)

    image_format = CONF.libvirt.snapshot_image_format or source_format

    if image_format == 'lvm' or image_format == 'rbd':
        image_format = 'raw'

    metadata = self._create_snapshot_metadata(base,
                                              instance,
                                              image_format,
                                              snapshot['name'])

    snapshot_name = uuid.uuid4().hex

    state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]

    if(self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
                                            MIN_QEMU_LIVESNAPSHOT_VERSION,
                                            REQ_HYPERVISOR_LIVESNAPSHOT)
            and source_format not in ('lvm', 'rbd')
            and not CONF.ephemeral_storage_encryption.enabled):
            live_snapshot = True

            try:
                virt_dom.blockJobAbort(disk_path, 0)
            except libvirt.libvirtError as ex:
                error_code = ex.get_error_code()
                if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
                    live_snapshot = False
                else:
                    pass
        else:
            live_snapshot = False

        if state == power_state.SHUTDOWN:
            live_snapshot = False

        if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
            if state == power_state.RUNNING or state == power_state.PAUSED:
                self._detach_pci_devices(virt_dom, pci_manager.
                    get_instance_pci_devs(instance))
                self._detach_sriov_ports(context, instance, virt_dom)
                virt_dom.managedSave(0)

        snapshot_backend = self.image_backend.snapshot(instance,
                                                       disk_path,
                                                       image_type=source_format)

        if live_snapshot:
            LOG.info(_LI("Beginning live snapshot process"),
                     instance=instance)
        else:
            LOG.info(_LI("Beginning cold snapshot process"),
                     instance=instance)

        update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
        snapshot_directory = CONF.libvirt.snapshots_directory
        fileutils.ensure_tree(snapshot_directory)
        with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                out_path = os.path.join(tmpdir, snapshot_name)
                if live_snapshot:
                    # openstack貌似不支持动态创建快照，所以不修改这里的代码
                    os.chmod(tmpdir, 0o701)
                    self._live_snapshot(virt_dom, disk_path, out_path, image_format
                        )
                else:
                    # 修改的代码
                    # 首先将虚拟机磁盘镜像改名，加个后缀“.base”
                    disk_path_base = disk_path + '.base'
                    out_path = disk_path
                    utils.execute('mv', disk_path, disk_path_base)
                    disk_path = disk_path_base
                # 然后在原磁盘文件上创建增量文件
                libvirt_utils.create_overlay(disk_path, out_path, image_format)
                # 强制重启虚拟机
                virt_dom.reset()
        finally:
            new_dom = None
            if CONF.libvirt.type != 'lxc' and not live_snapshot:
                if state == power_state.RUNNING:
                    new_dom = self._create_domain(domain=virt_dom)
                elif state == power_state.PAUSED:
                    new_dom = self._create_domain(domain=virt_dom,
                            launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
                if new_dom is not None:
                    self._attach_pci_devices(new_dom, pci_manager.
                        get_instance_pci_devs(instance))
                    self._attach_sriov_ports(context, instance, new_dom)
            LOG.info(_LI("Snapshot extracted, beginning image upload"),
                instance=instance)

        update_task_state(task_state=task_states.IMAGE_UPLOADING,
                          expected_state=task_states.IMAGE_PENDING_UPLOAD)

        # 将原先的磁盘文件上传到镜像服务器
        with libvirt_utils.file_open(disk_path) as image_file:
            self._image_api.update(context,
                                   image_id,
                                   metadata,
                                   image_file)
            LOG.info(_LI("Snapshot image upload complete"),
                     instance=instance)
```

3.1.6 在 virt/libvirt/utils.py 创建新函数

```
def create_overlay(disk_path, out_path, dest_format):
    if dest_fmt != 'qcow2':
        raise

    qemu_img_cmd = ('qemu-img', 'create', '-b', disk_path, '-f', dest_fmt,
        out_path)
    execute(*qemu_img_cmd)
```

3.2 镜像链为 3 的情况

3.2.1 镜像链为 3 的情况的描述

原先最基本的情况是：虚拟机的镜像链只有两个，也就是只有一个当初创建虚拟机用的后端镜像和一个增量文件。

现在的情况是：我们已经为备份过一次虚拟机，也就是现在虚拟机的镜像链为 3，有最初创建虚拟机用的后端镜像，一个备份用的镜像和一个最新的增量文件，也就是 base <– overlay1 <– overlay2。

如果再备份一次虚拟机的话，虚拟机的镜像链就会变成 4，为了保持虚拟机的性能，我们决定在备份虚拟机时，能够将 overlay1 和 overlay2 合并，保持镜像链为 3。

3.2.2 核心函数的修改

这里的修改方法参考了《快照相关原理》的第六节“删除快照”。

当镜像链为 4 时，镜像链为：base <- overlay1 <- overlay2 <- overlay3。为了保持镜像链为 3，我们现在需要先把 overlay2 的内容融入 overlay1 中，然后把 overlay3 的后端镜像变为 overlay1。

修改后的核心函数如下：

```python
    def snapshot_overlay(self, context, instance, image_id, update_task_state):
        try:
            virt_dom = self._lookup_by_name(instance['name'])
        except exception.InstanceNotFound:
            raise exception.InstanceNotRunning(instance_id=instance['uuid'])

        base_image_ref = instance['image_ref']
        base = compute_utils.get_image_metadata(context, self._image_api,
            base_iamge_ref, instance)
        snapshot = self._image_api.get(context, image_id)
        disk_path = libvirt_utils.find_disk(virt_dom)
        source_format = libvirt_utils.get_disk_type(disk_path)

        image_format = CONF.libvirt.snapshot_image_format or source_format

        if image_format == 'lvm' or image_format == 'rbd':
            image_format = 'raw'

        metadata = self._create_snapshot_metadata(base,
                                                  instance,
                                                  image_format,
                                                  snapshot['name'])

        snapshot_name = uuid.uuid4().hex

        state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]

        if(self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
                                  MIN_QEMU_LIVESNAPSHOT_VERSION,
                                  REQ_HYPERVISOR_LIVESNAPSHOT)
            and source_format not in ('lvm', 'rbd')
            and not CONF.ephemeral_storage_encryption.enabled):
            live_snapshot = True

            try:
                virt_dom.blockJobAbort(disk_path, 0)
            except libvirt.libvirtError as ex:
                error_code = ex.get_error_code()
                if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
                    live_snapshot = False
                else:
                    pass
        else:
            live_snapshot = False

        if state == power_state.SHUTDOWN:
            live_snapshot = False

        if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
            if state == power_state.RUNNING or state == power_state.PAUSED:
                self._detach_pci_devices(virt_dom, pci_manager.get_instance_pci_devs(instance))
                self._detach_sriov_ports(context, instance, virt_dom)
                virt_dom.managedSave(0)

        snapshot_backend = self.image_backend.snapshot(instance,
                                                       disk_path,
                                                       image_type=source_format)

        if live_snapshot:
            LOG.info(_LI("Beginning live snapshot process"),
                     instance=instance)
        else:
            LOG.info(_LI("Beginning cold snapshot process"),
                     instance=instance)

        update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
        snapshot_directory = CONF.libvirt.snapshots_directory
        fileutils.ensure_tree(snapshot_directory)
        with utils.tempdir(dir=snapshot_directory) as tmpdir:
            try:
                out_path = os.path.join(tmpdir, snapshot_name)
                if live_snapshot:
                    # openstack貌似不支持动态创建快照，所以不修改这里的代码
                    os.chmod(tmpdir, 0o701)
                    self._live_snapshot(virt_dom, disk_path, out_path, image_format)
                else:
                    # 修改的代码
                    # 获得虚拟机镜像的后端镜像的文件名
                    backing_file = libvirt_utils.get_disk_backing_file(disk_path, basename=False)
                    # 获得后端镜像的后缀
                    suffix = backing_file.split(".")[-1]
                # 我们本身备份虚拟机时，会把原先虚拟机磁盘加一个“.base”后缀
                # 现在如果suffix == “.base”，那么说明这个虚拟机已经备份过一次，那么之后我们需要合并overlay1和overlay2
                # 如果虚拟机备份过一次，我们还需要将overlay1再加个后缀“.base”
                if suffix == ".base":
                    utils.execute('mv', backing_file, backing_file + '.base')
                    backing_file = backing_file + '.base'
                out_path = disk_path
                utils.execute('mv', disk_path, disk_path + '.base')
                disk_path = disk_path + '.base'
                # 然后在原磁盘文件上创建增量文件
                libvirt_utils.create_overlay(disk_path, out_path, image_format)
                # 强制重启虚拟机
                virt_dom.reset()
                # 判断虚拟机是否备份过一次
                # 如果备份过一次，现在我们的快照链如下:
                # base <- overlay1 <- overlay2 <- overlay3
                if suffix == ".base":
                    # 因为overlay1改名了，所以先要更改overlay2的后端镜像名
                    utils.execute('qemu-img', 'rebase', '-b', backing_file, disk_path)
                    # 然后将overlay2融入overlay1中
                    utils.execute('qemu-img', 'commit', disk_path)
                    # 然后将overlay1作为overlay3的后端镜像
                    utils.execute('qemu-img', 'rebase', '-u', '-b', backing_file, out_path)
                    # 最后删除overlay2这个镜像
                    utils.execute('rm', disk_path)
                    # 这是为了之后上传后端镜像时，上传的是backing_file
                    disk_path = backing_file
        finally:
            new_dom = None
            if CONF.libvirt.type != 'lxc' and not live_snapshot:
                if state == power_state.RUNNING:
                    new_dom = self._create_domain(domain=virt_dom)
                elif state == power_state.PAUSED:
                    new_dom = self._create_domain(domain=virt_dom,
                            launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
                if new_dom is not None:
                    self._attach_pci_devices(new_dom, pci_manager.get_instance_pci_devs(instance))
                    self._attach_sriov_ports(context, instance, new_dom)
            LOG.info(_LI("Snapshot extracted, beginning image upload"),
                instance=instance)

        update_task_state(task_state=task_states.IMAGE_UPLOADING,
                          expected_state=task_states.IMAGE_PENDING_UPLOAD)

        # 将原先的磁盘文件上传到镜像服务器
        with libvirt_utils.file_open(disk_path) as image_file:
            self._image_api.update(context,
                                   image_id,
                                   metadata,
                                   image_file)
            LOG.info(_LI("Snapshot image upload complete"),
                     instance=instance)
```