目 录

1 分析 resize 流程前的必要知识

1.1 nova 中的 RPC 机制

1.2 重要的数据类型

1.2.1 req

1.2.2 context

```
1    # 根据 req 创建环境上下文 context
2    # context 是 nova/context.py 中的 RequestContext 类
3    context = req.environ["nova.context"]
```

1.2.3 instance

```
1    # 根据 req 和 instance_id 创建 instance
2    # instance 是 nova/context/instance.py 中的 Instance 类
3    instance = self._get_server(context, req, instance_id)
4
5
6    instance_type
7    flavor_id
8    deltas
9    quotas
10   vm_state
```

2 nova-api 阶段

入口函数为：

```
1    # 这个函数在 nova/api/openstack/compute/servers.py
2    def _resize(self, req, instance_id, flavor_id, **kwargs):
3        ...
4        try:
5            # compute_api 是 nova/compute/api.py 中的 API 类
6            self.compute_api.resize(context, instance, flavor_id, **kwargs)
7        ...
```

进一步看 API.resize() 函数：

```
1    # nova/compute/api.py API.resize()
2    def resize(self, context, instance, flavor_id=None,
3               **extra_instance_updates):
```

```
 4          . . .
 5
 6          # filter_properties 与选择本地扩容或选择异地扩容有关
 7          filter_properties = {'ignore_hosts': []}
 8
 9          if not CONF.allow_resize_to_same_host:
10              filter_properties['ignore_hosts'].append(instance['host'])
11
12          if (not flavor_id and not CONF.allow_migrate_to_same_host):
13              filter_properties['ignore_hosts'].append(instance['host'])
14
15          . . .
16
17          # scheduler_hint 挺重要的，是 nova-scheduler 的参数
18          scheduler_hint = {'filter_properties': filter_properties}
19          self.compute_task_api.resize_instance(context, instance,
20                  extra_instance_updates, scheduler_hint=scheduler_hint,
21                  flavor=new_instance_type,
22                  reservations=quotas.reservations or [])
```

```
 1      # nova/conductor/api.py ComputeTaskAPI.resize_instance()
 2      def resize_instance(self, context, instance, extra_instance_updates,
 3                      scheduler_hint, flavor, reservations):
 4          self.conductor_compute_rpcapi.migrate_server(
 5              context, instance, scheduler_hint, False, False, flavor,
 6              None, None, reservations)
```

```
 1      # nova/conductor/rpcapi.py ComputeTaskAPI.migrate_server()
 2      def migrate_server(self, context, instance, scheduler_hint, live, rebuild,
 3                  flavor, block_migration, disk_over_commit,
 4                  reservations=None):
 5          . . .
 6          cctxt = self.client.prepare(version=version)
 7          return cctxt.call(context, 'migrate_server',
 8                      instance=instance, scheduler_hint=scheduler_hint,
 9                      live=live, rebuild=rebuild, flavor=flavor_p,
10                      block_migration=block_migration,
11                      disk_over_commit=disk_over_commit,
12                      reservations=reservations)
```

3 nova-conductor 部分

```
 1      # nova/conductor/manager.py ComputeTaskManager.migrate_server()
 2      def migrate_server(self, context, instance, scheduler_hint, live, rebuild,
 3              flavor, block_migration, disk_over_commit, reservations=None):
 4          . . .
 5          if live and not rebuild and not flavor:
 6              self._live_migrate(context, instance, scheduler_hint,
 7                          block_migration, disk_over_commit)
 8          elif not live and not rebuild and flavor:
```

```
9          ...
10         with compute_utils.EventReporter(context, 'cold_migrate',
11                                          instance_uuid):
12             self._cold_migrate(context, instance, flavor,
13                                scheduler_hint['filter_properties'],
14                                reservations)
15         ...
```

4 冷迁移

4.1 冷迁移中的 nova-conductor 部分

```
1    # nova/conductor/manager.py ComputeTaskManager._cold_migrate()
2    def _cold_migrate(self, context, instance, flavor, filter_properties,
3                      reservations):
4        ...
5        try:
6            ...
7            # 选择目的主机
8            hosts = self.scheduler_client.select_destinations(
9                    context, request_spec, filter_properties)
10           host_state = hosts[0]
11       ...
12
13       try:
14           ...
15           (host, node) = (host_state['host'], host_state['nodename'])
16           self.compute_rpcapi.prep_resize(
17               context, image, instance,
18               flavor, host,
19               reservations, request_spec=request_spec,
20               filter_properties=filter_properties, node=node)
21       ...
```

```
1    # nova/compute/rpcapi.py ComputeAPI.prep_resize()
2    def prep_resize(self, ctxt, image, instance, instance_type, host,
3                    reservations=None, request_spec=None,
4                    filter_properties=None, node=None):
5        ...
6        cctxt = self.client.prepare(server=host, version=version)
7        cctxt.cast(ctxt, 'prep_resize',
8                   instance=instance,
9                   instance_type=instance_type_p,
10                  image=image_p, reservations=reservations,
11                  request_spec=request_spec,
12                  filter_properties=filter_properties,
13                  node=node)
```

4.2 冷迁移中的 nova-compute 部分

4.2.1 目的主机上的操作：prep_resize

```
1    # nova/compute/manager.py ComputeManager.prep_resize()
2    def prep_resize(self, context, image, instance, instance_type,
3                    reservations, request_spec, filter_properties, node):
4        ...
5        with self._error_out_instance_on_exception(context, instance,
6                                                   quotas=quotas):
7            ...
8            try:
9                self._prep_resize(context, image, instance,
10                                 instance_type, quotas,
11                                 request_spec, filter_properties,
12                                 node)
13           ...
```

```
1    # nova/compute/manager.py ComputeManager._prep_resize()
2    def _prep_resize(self, context, image, instance, instance_type,
3            quotas, request_spec, filter_properties, node):
4
5        ...
6        with rt.resize_claim(context, instance, instance_type,
7                             image_meta=image, limits=limits) as claim:
8            ...
9            self.compute_rpcapi.resize_instance(
10                   context, instance, claim.migration, image,
11                   instance_type, quotas.reservations)
```

```
1    # nova/compute/rpcapi.py ComputeAPI.resize_instance()
2    def resize_instance(self, ctxt, instance, migration, image, instance_type,
3                        reservations=None):
4        ...
5        cctxt = self.client.prepare(server=_compute_host(None, instance),
6                version=version)
7        cctxt.cast(ctxt, 'resize_instance',
8                   instance=instance, migration=migration,
9                   image=image, reservations=reservations,
10                  instance_type=instance_type_p)
```

4.2.2 源主机的操作：resize_instance

```
1    # nova/compute/manager.py ComputeManager.resize_instance()
2    def resize_instance(self, context, instance, image,
3                        reservations, migration, instance_type,
4                        clean_shutdown=True):
5        ...
6        with self._error_out_instance_on_exception(context, instance,
7                                                   quotas=quotas):
```

```
8               ...
9               # 获得虚拟机块设备的信息
10              block_device_info = self._get_instance_block_device_info(
11                              context, instance, bdms=bdms)
12
13              # 关闭虚拟机并迁移虚拟机的增量文件
14              disk_info = self.driver.migrate_disk_and_power_off(
15                      context, instance, migration.dest_host,
16                      instance_type, network_info,
17                      block_device_info,
18                      timeout, retry_interval)
19              ...
20              self.compute_rpcapi.finish_resize(context, instance,
21                      migration, image, disk_info,
22                      migration.dest_compute, reservations=quotas.reservations)
23              ...
```

migrate_disk_and_power_off() 是源主机上将虚拟机迁移给目的主机的实现函数，主要利用了 libvirt API。这个函数的分析在《nova 调用 libvirt》中的"nova 扩容时对 libvirt 的调用"一节。

```
1       # nova/compute/rpcapi.py(690) ComputeAPI.finish_resize()
2       def finish_resize(self, ctxt, instance, migration, image, disk_info,
3               host, reservations=None):
4           ...
5           cctxt = self.client.prepare(server=host, version=version)
6           cctxt.cast(ctxt, 'finish_resize',
7                       instance=instance, migration=migration,
8                       image=image, disk_info=disk_info, reservations=reservations)
```

4.2.3 目的主机上的操作：finish_resize

```
1       # nova/compute/manager.py ComputeManager.finish_resize()
2       def finish_resize(self, context, disk_info, image, instance,
3                       reservations, migration):
4           quotas = quotas_obj.Quotas.from_reservations(context,
5                                                       reservations,
6                                                       instance=instance)
7           try:
8               self._finish_resize(context, instance, migration,
9                               disk_info, image)
10          ...
```

```
1       # nova/compute/manager.py ComputeManager._finish_resize()
2       def _finish_resize(self, context, instance, migration, disk_info,
3                       image):
4           ...
5           try:
6               self.driver.finish_migration(context, migration, instance,
7                                           disk_info,
8                                           network_info,
```

6

```
9                                    image , resize_instance ,
10                                   block_device_info , power_on)
11        ...
```

5 在线迁移

5.1 在线迁移中的 nova-conductor 部分

```
1    # nova/conductor/manager.py ComputeTaskManager._live_migrate()
2    def _live_migrate( self , context , instance , scheduler_hint ,
3                       block_migration , disk_over_commit):
4        destination = scheduler_hint.get("host")
5        try :
6            live_migrate.execute(context , instance , destination ,
7                                 block_migration , disk_over_commit)
8            ...
```

```
1    # nova/conductor/tasks/live_migrate.py execute()
2    def execute(context , instance , destination ,
3            block_migration , disk_over_commit):
4    task = LiveMigrationTask(context , instance ,
5                             destination ,
6                             block_migration ,
7                             disk_over_commit)
8    # TODO(johngarbutt) create a superclass that contains a safe_execute call
9    return task.execute()
```

```
1    # nova/conductor/tasks/live_migrate.py LiveMigrationTask.execute()
2    def execute( self ):
3        self._check_instance_is_running()
4        self._check_host_is_up( self.source)
5
6        if not self.destination :
7            self.destination = self._find_destination()
8        else :
9            self._check_requested_destination()
10
11       # TODO(johngarbutt) need to move complexity out of compute manager
12       # TODO(johngarbutt) disk_over_commit?
13       return self.compute_rpcapi.live_migration( self.context ,
14               host=self.source ,
15               instance=self.instance ,
16               dest=self.destination ,
17               block_migration=self.block_migration ,
18               migrate_data=self.migrate_data)
```

```
1    # nova/compute/rpcapi.py ComputeAPI.live_migration()
2    def live_migration( self , ctxt , instance , dest , block_migration , host ,
```

```
3                        migrate_data=None):
4          ...
5          cctxt = self.client.prepare(server=host, version=version)
6          cctxt.cast(ctxt, 'live_migration', instance=instance,
7                     dest=dest, block_migration=block_migration,
8                     migrate_data=migrate_data)
```

5.2 在线迁移中的 nova-compute 部分

```
1      # nova/compute/manager.py ComputeManager.live_migration()
2      def live_migration(self, context, dest, instance, block_migration,
3                         migrate_data):
4          """Executing live migration.
5
6          :param context: security context
7          :param instance: a nova.objects.instance.Instance object
8          :param dest: destination host
9          :param block_migration: if true, prepare for block migration
10         :param migrate_data: implementation specific params
11
12         """
13
14         # NOTE(danms): since instance is not the first parameter, we can't
15         # use @object_compat on this method. Since this is the only example,
16         # we do this manually instead of complicating the decorator
17         if not isinstance(instance, obj_base.NovaObject):
18             expected = ['metadata', 'system_metadata',
19                         'security_groups', 'info_cache']
20             instance = objects.Instance._from_db_object(
21                 context, objects.Instance(), instance,
22                 expected_attrs=expected)
23
24         # Create a local copy since we'll be modifying the dictionary
25         migrate_data = dict(migrate_data or {})
26         try:
27             if block_migration:
28                 block_device_info = self._get_instance_block_device_info(
29                     context, instance)
30                 disk = self.driver.get_instance_disk_info(
31                     instance.name, block_device_info=block_device_info)
32             else:
33                 disk = None
34
35             pre_migration_data = self.compute_rpcapi.pre_live_migration(
36                 context, instance,
37                 block_migration, disk, dest, migrate_data)
38             migrate_data['pre_live_migration_result'] = pre_migration_data
39
40         except Exception:
41             with excutils.save_and_reraise_exception():
42                 LOG.exception(_LE('Pre live migration failed at %s'),
43                               dest, instance=instance)
44                 self._rollback_live_migration(context, instance, dest,
```

```
45                                                  block_migration, migrate_data)
46
47          # Executing live migration
48          # live_migration might raises exceptions, but
49          # nothing must be recovered in this version.
50          self.driver.live_migration(context, instance, dest,
51                                       self._post_live_migration,
52                                       self._rollback_live_migration,
53                                       block_migration, migrate_data)
```

5.2.1 目的主机上的操作：pre_live_migration

```
1       # nova/compute/manager.py ComputeManager.pre_live_migration()
2       def pre_live_migration(self, context, instance, block_migration, disk,
3                              migrate_data):
4           """Preparations for live migration at dest host.
5
6           :param context: security context
7           :param instance: dict of instance data
8           :param block_migration: if true, prepare for block migration
9           :param migrate_data: if not None, it is a dict which holds data
10                                required for live migration without shared
11                                storage.
12
13          """
14          block_device_info = self._get_instance_block_device_info(
15                              context, instance, refresh_conn_info=True)
16
17          network_info = self._get_instance_nw_info(context, instance)
18          self._notify_about_instance_usage(
19                      context, instance, "live_migration.pre.start",
20                      network_info=network_info)
21
22          pre_live_migration_data = self.driver.pre_live_migration(context,
23                                          instance,
24                                          block_device_info,
25                                          network_info,
26                                          disk,
27                                          migrate_data)
28
29          # NOTE(tr3buchet): setup networks on destination host
30          self.network_api.setup_networks_on_host(context, instance,
31                                                  self.host)
32
33          # Creating filters to hypervisors and firewalls.
34          # An example is that nova-instance-instance-xxx,
35          # which is written to libvirt.xml(Check "virsh nwfilter-list")
36          # This nwfilter is necessary on the destination host.
37          # In addition, this method is creating filtering rule
38          # onto destination host.
39          self.driver.ensure_filtering_rules_for_instance(instance,
40                                          network_info)
41
```

```
42            self._notify_about_instance_usage(
43                    context, instance, "live_migration.pre.end",
44                    network_info=network_info)
45
46            return pre_live_migration_data
```

```
1      # nova/virt/libvirt/driver.py LibvirtDriver.pre_live_migration()
2      def pre_live_migration(self, context, instance, block_device_info,
3                             network_info, disk_info, migrate_data=None):
4          """Preparation live migration."""
5          # Steps for volume backed instance live migration w/o shared storage.
6          is_shared_block_storage = True
7          is_shared_instance_path = True
8          is_block_migration = True
9          instance_relative_path = None
10         if migrate_data:
11             is_shared_block_storage = migrate_data.get(
12                     'is_shared_block_storage', True)
13             is_shared_instance_path = migrate_data.get(
14                     'is_shared_instance_path', True)
15             is_block_migration = migrate_data.get('block_migration', True)
16             instance_relative_path = migrate_data.get('instance_relative_path')
17
18         if not (is_shared_instance_path and is_shared_block_storage):
19             # NOTE(dims): Using config drive with iso format does not work
20             # because of a bug in libvirt with read only devices. However
21             # one can use vfat as config_drive_format which works fine.
22             # Please see bug/1246201 for details on the libvirt bug.
23             if CONF.config_drive_format != 'vfat':
24                 if configdrive.required_by(instance):
25                     raise exception.NoLiveMigrationForConfigDriveInLibVirt()
26
27         if not is_shared_instance_path:
28             # NOTE(mikal): this doesn't use libvirt_utils.get_instance_path
29             # because we are ensuring that the same instance directory name
30             # is used as was at the source
31             if instance_relative_path:
32                 instance_dir = os.path.join(CONF.instances_path,
33                                             instance_relative_path)
34             else:
35                 instance_dir = libvirt_utils.get_instance_path(instance)
36
37             if os.path.exists(instance_dir):
38                 raise exception.DestinationDiskExists(path=instance_dir)
39             os.mkdir(instance_dir)
40
41             if not is_shared_block_storage:
42                 # Ensure images and backing files are present.
43                 self._create_images_and_backing(context, instance,
44                                                 instance_dir, disk_info)
45
46         if not (is_block_migration or is_shared_instance_path):
47             # NOTE(angdraug): when block storage is shared between source and
48             # destination and instance path isn't (e.g. volume backed or rbd
49             # backed instance), instance path on destination has to be prepared
```

```
50
51              # Touch the console.log file, required by libvirt.
52              console_file = self._get_console_log_path(instance)
53              libvirt_utils.file_open(console_file, 'a').close()
54
55              # if image has kernel and ramdisk, just download
56              # following normal way.
57              self._fetch_instance_kernel_ramdisk(context, instance)
58
59          # Establishing connection to volume server.
60          block_device_mapping = driver.block_device_info_get_mapping(
61              block_device_info)
62          for vol in block_device_mapping:
63              connection_info = vol['connection_info']
64              disk_info = blockinfo.get_info_from_bdm(
65                  CONF.libvirt.virt_type, vol)
66              self._connect_volume(connection_info, disk_info)
67
68          if is_block_migration and len(block_device_mapping):
69              # NOTE(stpierre): if this instance has mapped volumes,
70              # we can't do a block migration, since that will
71              # result in volumes being copied from themselves to
72              # themselves, which is a recipe for disaster.
73              LOG.error(
74                  _LE('Cannot block migrate instance %s with mapped volumes') %
75                  instance.uuid)
76              raise exception.MigrationError(
77                  _('Cannot block migrate instance %s with mapped volumes') %
78                  instance.uuid)
79
80          # We call plug_vifs before the compute manager calls
81          # ensure_filtering_rules_for_instance, to ensure bridge is set up
82          # Retry operation is necessary because continuously request comes,
83          # concurrent request occurs to iptables, then it complains.
84          max_retry = CONF.live_migration_retry_count
85          for cnt in range(max_retry):
86              try:
87                  self.plug_vifs(instance, network_info)
88                  break
89              except processutils.ProcessExecutionError:
90                  if cnt == max_retry - 1:
91                      raise
92                  else:
93                      LOG.warn(_LW('plug_vifs() failed %(cnt)d. Retry up to '
94                                   '%(max_retry)d.'),
95                               {'cnt': cnt,
96                                'max_retry': max_retry},
97                               instance=instance)
98                      greenthread.sleep(1)
99
100         res_data = {'graphics_listen_addrs': {}}
101         res_data['graphics_listen_addrs']['vnc'] = CONF.vncserver_listen
102         res_data['graphics_listen_addrs']['spice'] = CONF.spice.server_listen
103
104         return res_data
```

11

5.2.2 源主机上的操作：live_migration

```python
# nova/compute/manager.py ComputeManager.live_migration()
def live_migration(self, context, dest, instance, block_migration,
                   migrate_data):
    """Executing live migration.

    :param context: security context
    :param instance: a nova.objects.instance.Instance object
    :param dest: destination host
    :param block_migration: if true, prepare for block migration
    :param migrate_data: implementation specific params

    """

    # NOTE(danms): since instance is not the first parameter, we can't
    # use @object_compat on this method. Since this is the only example,
    # we do this manually instead of complicating the decorator
    if not isinstance(instance, obj_base.NovaObject):
        expected = ['metadata', 'system_metadata',
                    'security_groups', 'info_cache']
        instance = objects.Instance._from_db_object(
            context, objects.Instance(), instance,
            expected_attrs=expected)

    # Create a local copy since we'll be modifying the dictionary
    migrate_data = dict(migrate_data or {})
    try:
        if block_migration:
            block_device_info = self._get_instance_block_device_info(
                context, instance)
            disk = self.driver.get_instance_disk_info(
                instance.name, block_device_info=block_device_info)
        else:
            disk = None

        pre_migration_data = self.compute_rpcapi.pre_live_migration(
            context, instance,
            block_migration, disk, dest, migrate_data)
        migrate_data['pre_live_migration_result'] = pre_migration_data

    except Exception:
        with excutils.save_and_reraise_exception():
            LOG.exception(_LE('Pre live migration failed at %s'),
                          dest, instance=instance)
            self._rollback_live_migration(context, instance, dest,
                                          block_migration, migrate_data)

    # Executing live migration
    # live_migration might raises exceptions, but
    # nothing must be recovered in this version.
    self.driver.live_migration(context, instance, dest,
                               self._post_live_migration,
                               self._rollback_live_migration,
                               block_migration, migrate_data)
```

12

```
1      # nova/virt/libvirt/driver.py LibvirtDriver.live_migration()
2      def live_migration(self, context, instance, dest,
3                          post_method, recover_method, block_migration=False,
4                          migrate_data=None):
5          """Spawning live_migration operation for distributing high-load.
6
7          :param context: security context
8          :param instance:
9              nova.db.sqlalchemy.models.Instance object
10             instance object that is migrated.
11         :param dest: destination host
12         :param post_method:
13             post operation method.
14             expected nova.compute.manager._post_live_migration.
15         :param recover_method:
16             recovery method when any exception occurs.
17             expected nova.compute.manager._rollback_live_migration.
18         :param block_migration: if true, do block migration.
19         :param migrate_data: implementation specific params
20
21         """
22
23         # 'dest' will be substituted into 'migration_uri' so ensure
24         # it does't contain any characters that could be used to
25         # exploit the URI accepted by libivrt
26         if not libvirt_utils.is_valid_hostname(dest):
27             raise exception.InvalidHostname(hostname=dest)
28
29         greenthread.spawn(self._live_migration, context, instance, dest,
30                           post_method, recover_method, block_migration,
31                           migrate_data)
```

```
1      # nova/virt/libvirt/driver.py LibvirtDriver._live_migration()
2      def _live_migration(self, context, instance, dest, post_method,
3                          recover_method, block_migration=False,
4                          migrate_data=None):
5          """Do live migration.
6
7          :param context: security context
8          :param instance:
9              nova.db.sqlalchemy.models.Instance object
10             instance object that is migrated.
11         :param dest: destination host
12         :param post_method:
13             post operation method.
14             expected nova.compute.manager._post_live_migration.
15         :param recover_method:
16             recovery method when any exception occurs.
17             expected nova.compute.manager._rollback_live_migration.
18         :param block_migration: if true, do block migration.
19         :param migrate_data: implementation specific params
20         """
21
22         # Do live migration.
23         try:
```

```
24              if block_migration:
25                  flaglist = CONF.libvirt.block_migration_flag.split(',')
26              else:
27                  flaglist = CONF.libvirt.live_migration_flag.split(',')
28              flagvals = [getattr(libvirt, x.strip()) for x in flaglist]
29              logical_sum = reduce(lambda x, y: x | y, flagvals)
30
31              dom = self._lookup_by_name(instance["name"])
32
33              pre_live_migrate_data = (migrate_data or {}).get(
34                                      'pre_live_migration_result', {})
35              listen_addrs = pre_live_migrate_data.get('graphics_listen_addrs')
36
37              migratable_flag = getattr(libvirt, 'VIR_DOMAIN_XML_MIGRATABLE',
38                                        None)
39
40              if migratable_flag is None or listen_addrs is None:
41                  self._check_graphics_addresses_can_live_migrate(listen_addrs)
42                  dom.migrateToURI(CONF.libvirt.live_migration_uri % dest,
43                                   logical_sum,
44                                   None,
45                                   CONF.libvirt.live_migration_bandwidth)
46              else:
47                  old_xml_str = dom.XMLDesc(migratable_flag)
48                  new_xml_str = self._correct_listen_addr(old_xml_str,
49                                                          listen_addrs)
50                  try:
51                      dom.migrateToURI2(CONF.libvirt.live_migration_uri % dest,
52                                        None,
53                                        new_xml_str,
54                                        logical_sum,
55                                        None,
56                                        CONF.libvirt.live_migration_bandwidth)
57                  except libvirt.libvirtError as ex:
58                      # NOTE(mriedem): There is a bug in older versions of
59                      # libvirt where the VIR_DOMAIN_XML_MIGRATABLE flag causes
60                      # virDomainDefCheckABIStability to not compare the source
61                      # and target domain xml's correctly for the CPU model.
62                      # We try to handle that error here and attempt the legacy
63                      # migrateToURI path, which could fail if the console
64                      # addresses are not correct, but in that case we have the
65                      # _check_graphics_addresses_can_live_migrate check in place
66                      # to catch it.
67                      # TODO(mriedem): Remove this workaround when
68                      # Red Hat BZ #1141838 is closed.
69                      error_code = ex.get_error_code()
70                      if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
71                          LOG.warn(_LW('An error occurred trying to live '
72                                       'migrate. Falling back to legacy live '
73                                       'migrate flow. Error: %s'), ex,
74                                   instance=instance)
75                          self._check_graphics_addresses_can_live_migrate(
76                              listen_addrs)
77                          dom.migrateToURI(
78                              CONF.libvirt.live_migration_uri % dest,
79                              logical_sum,
```

```
80                          None,
81                          CONF.libvirt.live_migration_bandwidth)
82                  else:
83                      raise
84
85          except Exception as e:
86              with excutils.save_and_reraise_exception():
87                  LOG.error(_LE("Live Migration failure: %s"), e,
88                          instance=instance)
89              recover_method(context, instance, dest, block_migration)
90
91          post_method(context, instance, dest, block_migration,
92                  migrate_data)
```