ZStack 技术白皮书精选

ZStack 可拓展性的秘密武器 1：

异步架构

扫一扫二维码，获取更多技术干货吧

版权声明

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple （简单）、Strong （健壮）、Smart （智能）、Scalable （弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

ZSTACK--可拓展性的秘密武器 1：异步架构

ZStack 的架构使得其中 99%的任务能被异步执行。基于这点，ZStack 中单一的管理节点可以管理几千台物理服务器，上万台虚拟机，处理成千上万个并发任务。

动机

对于管理大量硬件和虚拟机的公有云而言，可拓展性是一个 IaaS 软件必须解决的关键问题之一。对于一个大概拥有 5 万台物理服务器的中型数据中心，预计可能有 150 万台虚拟机，1 万名用户。虽然用户开关虚拟机的频率不会像刷朋友圈一样频繁，但是在某一时刻，IaaS 系统可能有成千上万个任务要处理，这些任务可能来自 API 也可能来自内部组件。在糟糕的情况下，用户为了创建一台新的虚拟机可能需要等待一个小时，因为系统同时被 5000 个任务阻塞，然而线程池仅有 1000 条线程。

问题

首先，我们非常不赞同一些文章里面描写的关于“**一些基础配套设施，尤其是数据库和消息代理（message brokers）限制了 IaaS 的可拓展性**”的观点。首先，对于数据库而言，IaaS 软件的数据量相比 facebook 和 twitter 而言只能勉强算中小型，facebook 和 twitter 的数据量是万亿级别，IaaS 软件只处于百万级别（对于一些非常大型的数据中心），而 facebook 和 twitter 依旧坚强的使用 Mysql 作为他们主要的数据库。其次，对于消息代理而言，ZStack 使用的 rabbitmq 相对 Apache Kafka 或 ZeroMQ 是一个中型的消息代理，但是它依然可以维持平均每秒 5 万条消息的吞吐量，对于 IaaS 软件内部通信而言这不就足够了么？我们认为足够了。

限制 IaaS 可拓展性的主要原因在于：任务执行缓慢。IaaS 软件上的任务运行非常缓慢，通常一项任务完成需要花费几秒甚至几分钟。所以当整个系统被缓慢的任务填满的时候，新任务的延迟非常大是很正常的。执行缓慢的任务通常是由一个很长的任务路径组成的，比如，创建一个虚拟机，需要经过身份验证服务→调度器→镜像服务→存储服务→网络服务→虚拟

机管理程序，每一个服务可能会花费几秒甚至几分钟去引导外部硬件完成一些操作，这极大的延长了任务执行的时间。

同步和异步

传统的 IaaS 软件使用同步的方式执行任务，他们通常给每一个任务安排一个线程，这个线程只有在之前的任务执行完毕时才会开始执行下一个任务。因为任务执行缓慢，当达到一个任务并发的高峰时，系统会因为线程池容量不足，运行非常缓慢，新来的任务只能被放在队列中等待被执行。

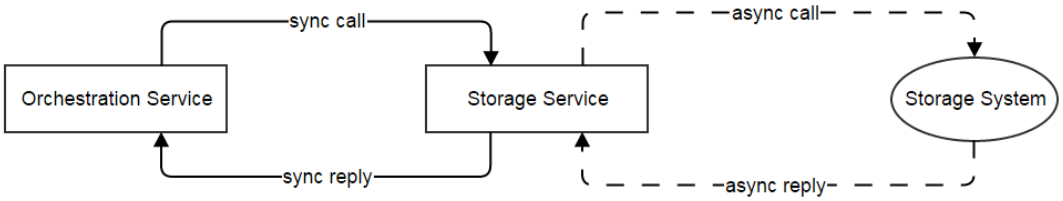
为了解决这个问题，一个直观的想法是提高线程池容量，但是这个想法在实际中是不可行的，即使现代操作系统允许一个应用程序拥有成千上万条线程，没有操作系统可以非常有效率的调度他们。随后有一个想法是把线程分发出去，让不同的操作系统上相似的软件分布式的处理线程，因为每一个软件都有它自己的线程池，这样最终增加了整个系统的线程容量。然而，分发会带来一定的开销，它增加了管理的复杂度，同时集群软件在软件设计层面依旧是一个挑战。最后，IaaS 软件自身变成云的瓶颈，而其他的基础设施包括数据库，消息代理和外部的系统（比如成千台物理服务器）都足够去处理更多的并发任务。

ZStack 通过异步架构来解决这个问题，如果我们把目光投向 IaaS 软件和数据中心的设备之间的关系，我们会发现 IaaS 软件实际上扮演着一个协调者的角色，它负责协调外部系统但并不做任何真正耗时的操作。举个例子，存储系统可以分配磁盘容量，镜像系统可以下载镜像模板，虚拟机管理程序可以创建虚拟机。**IaaS 软件所做的工作是做决策然后把子任务分配给不同的外部系统。**比如，对于 KVM，KVM 主机需要执行诸如准备磁盘、准备网络、创建虚拟机等子任务。创建一台虚拟机可能需要花费 5s，IaaS 软件花费时间为 0.5s，剩下的 4.5s 被 KVM 主机占用，ZStack 的异步架构使 IaaS 管理软件不用等待 4.5s，它只需要花费 0.5s 的时间选择让哪一台主机处理这个任务，然后把任务分派给那个主机。一旦主机完成了它的任务，它将结果通知给 IaaS 软件。通过异步架构，一个只有 100 条线程容量的线程池可以处理上千数的并发任务。

ZStack 的异步方法

异步操作在计算机科学中是非常常见的操作，异步 I/O,AJAX 等都是一些众所周知的例子。然而，把所有的业务逻辑都建立在异步操作的基础上，尤其是对于 IaaS 这种非常典型的集成软件，是存在很多挑战的。

最大的挑战是必须让所有组件都异步，并不只是一部分组件异步。举个例子，如果你在其他服务都是同步的条件下，建立一个异步的存储服务，整个系统性能并不会提升。因为在异步的调用存储服务时，调用的服务自身如果是同步的，那么调用的服务必须等待存储服务完成，才能进行下一步操作，这会使得整个工作流依旧是处于同步状态。



The thread that orchestration service calls storage service has to block until the storage service returns, though the storage service communicates to the external storage system asynchronously

（一个执行业务流程服务的线程同步调用了存储服务，然而存储服务和外部存储系统是异步通信的，因此它依旧需要等到存储服务返回之后，才能往下执行，这里的异步就等同于同步了。）

ZStack 的异步架构包含了三个模块：异步消息，异步方法，异步 HTTP 调用。

1. 异步消息

ZStack 使用 rabbitmq 作为一个消息总线连接各类服务，当一个服务调用另一个服务时，源服务发送一条消息给目标服务并注册一个回调函数，然后立即返回。一旦目标服务完成了任务，它返回一条消息触发源服务注册的回调函数。代码如下：

```
AttachNicToVmOnHypervisorMsg amsg = new AttachNicToVmOnHypervisorMsg();

amsg.setVmUuid(self.getUuid());

amsg.setHostUuid(self.getHostUuid());

amsg.setNics(msg.getNics());
```

```
bus.makeTargetServiceIdByResourceUuid(amsg, HostConstant.SERVICE_ID, self.getHostUuid());

bus.send(amsg, new CloudBusCallBack(msg) {

    @Override

    public void run(MessageReply reply) {

        AttachNicToVmReply r = new AttachNicToVmReply();

        if (!reply.isSuccess()) {

            r.setError(errf.instantiateErrorCode(VmErrors.ATTACH_NETWORK_ERROR, r.getError()));

        }

        bus.reply(msg, r);

    }

});
```

一个服务也可以同时发送一列消息给其它服务，然后异步的等待回复。

```
final ImageInventory inv = ImageInventory.valueOf(ivo);

final List<DownloadImageMsg> dmsgs =

CollectionUtils.transformToList(msg.getBackupStorageUuids(), new Function<DownloadImageMsg,

String>() {

    @Override

    public DownloadImageMsg call(String arg) {

        DownloadImageMsg dmsg = new DownloadImageMsg(inv);

        dmsg.setBackupStorageUuid(arg);

        bus.makeTargetServiceIdByResourceUuid(dmsg, BackupStorageConstant.SERVICE_ID, arg);

        return dmsg;
```

```
    }
});


bus.send(dmsgs, new CloudBusListCallBack(msg) {
    @Override
    public void run(List<MessageReply> replies) {
        /* do something */
    }
}
```

更甚，以设定的并行度发送一列消息也是可以实现的。也就是说，含有 10 条消息的消息列表，可以每次发送 2 条消息，也就是说第 3、4 条消息可以在第 1、2 条消息的回复收到后被发送。

```
     final List<ConnectHostMsg> msgs = new ArrayList<ConnectHostMsg>(hostsToLoad.size());
     for (String uuid : hostsToLoad) {
         ConnectHostMsg connectMsg = new ConnectHostMsg(uuid);
         connectMsg.setNewAdd(false);
         connectMsg.setServiceId(serviceId);
         connectMsg.setStartPingTaskOnFailure(true);
         msgs.add(connectMsg);
}
```

```
bus.send(msgs, HostGlobalConfig.HOST_LOAD_PARALLELISM_DEGREE.value(Integer.class), new
CloudBusSteppingCallback() {

        @Override

        public void run(NeedReplyMessage msg, MessageReply reply) {

            /* do something */

        }

});
```

2.异步的方法

服务在 ZStack 中是一等公民，他们通过异步消息进行通信。在服务的内部，有非常多的组件、插件使用方法调用的方式来进行交互，这种方式也是异步的。

```
protected void startVm(final APIStartVmInstanceMsg msg, final SyncTaskChain taskChain) {

    startVm(msg, new Completion(taskChain) {

        @Override

        public void success() {

            VmInstanceInventory inv = VmInstanceInventory.valueOf(self);

            APIStartVmInstanceEvent evt = new APIStartVmInstanceEvent(msg.getId());

            evt.setInventory(inv);

            bus.publish(evt);

            taskChain.next();

        }
```

```
        @Override
        public void fail(ErrorCode errorCode) {
            APIStartVmInstanceEvent evt = new APIStartVmInstanceEvent(msg.getId());
            evt.setErrorCode(errf.instantiateErrorCode(VmErrors.START_ERROR, errorCode));
            bus.publish(evt);
            taskChain.next();
        }
    });
}
```

回调函数也可以有返回值：

```
public void createApplianceVm(ApplianceVmSpec spec, final
ReturnValueCompletion<ApplianceVmInventory> completion) {
          CreateApplianceVmJob job = new CreateApplianceVmJob();
          job.setSpec(spec);
          if (!spec.isSyncCreate()) {
            job.run(new ReturnValueCompletion<Object>(completion) {
                @Override
                public void success(Object returnValue) {
                  completion.success((ApplianceVmInventory) returnValue);
                }
```

```
            @Override
            public void fail(ErrorCode errorCode) {
              completion.fail(errorCode);
            }
          });
        } else {
           jobf.execute(spec.getName(), OWNER, job, completion, ApplianceVmInventory.class);
        }
}
```

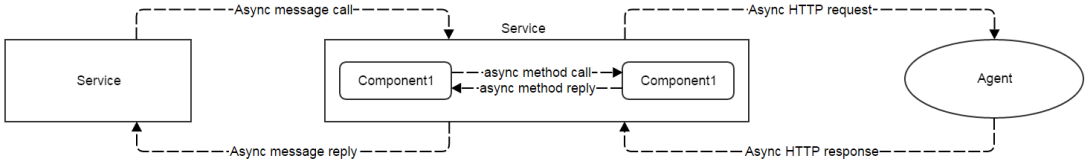
3.HTTP 异步调用

ZStack 使用一组 agent 去管理外部系统，比如：管理 KVM 主机的 agent，管理控制台代理的 agent，管理虚拟路由的 agent 等，这些 agents 全部都是搭建在 Python CherryPy 上的轻量级 web 服务器。因为如果没有 HTML5 技术，如 websockets 技术，是没有办法进行双向通信的，ZStack 每个请求的 HTTP 头部嵌入一个回调的 URL，因此，在任务完成后，agents 可以发送回复给调用者的 URL。

```
RefreshFirewallCmd cmd = new RefreshFirewallCmd();
List<ApplianceVmFirewallRuleTO> tos = new RuleCombiner().merge();
cmd.setRules(tos);


resf.asyncJsonPost(buildUrl(ApplianceVmConstant.REFRESH_FIREWALL_PATH), cmd, new
JsonAsyncRESTCallback<RefreshFirewallRsp>(msg, completion) {
```

```
    @Override
    public void fail(ErrorCode err) {
        /* handle failures */
    }

    @Override
    public void success(RefreshFirewallRsp ret) {
        /* do something */
    }

    @Override
    public Class<RefreshFirewallRsp> getReturnClass() {
        return RefreshFirewallRsp.class;
    }
});
```

通过以上三种方法，ZStack 已经建立了一个可以使所有组件都异步进行操作的全局架构。



总结

为了解决由缓慢且并发的任务引起的 IaaS 软件可拓展性受限的问题，我们演示了 ZStack 的异步架构。我们使用模拟器进行测试后发现，一个具有 1000 条线程的 ZStack 管理节点可

以轻松处理创建 100 万台虚拟机时产生的 10000 个并发任务。虽然单一管理节点的拓展性已经可以满足大多数云的负载需要，考虑到系统需要高可用性以及承受巨大的负载量（10 万个并发任务），我们需要一组管理节点来满足这些需求，如需了解 ZStack 的无状态服务，请阅读下一篇“ZStack 可拓展性秘密武器 2：无状态服务”。