分布式系统第一次作业

何泽 18340052

I

> 分布式系统的扩展方式有哪些?各有哪些利弊?

- 隐藏通信延迟

  即利用异步通信技术，设计分离的响应消息处理器

  - 优点：可以尽量避免等待远程服务对请求的响应
  - 缺点：并不是所有应用都适合这种模式；而且某些交互式应用的用户发出请求后，处于等待无所事事状态

- 分布

  即在多个机器上划分数据和计算

  - 优点：可以最大程度利用资源，加快计算速度
  - 缺点：需要网络进行通信，不适合网络差的地区

- 复制和缓存

  即在多个不同的机器上创建多个数据副本，用于复制文件服务器和数据库、Web站点进行镜像、Web缓存(在浏览器或者代理位置)或文件缓存(在服务器和客户端)。

  - 缺点

    - 设计多个副本(缓存或者复制)，导致不一致:修改一个副本后会让该副本与其他副本内容不一致
    - 总是保证副本的一致性需要对每次修改进行全局同步
    - 全局同步由于其高昂的代价导致难以应用到大规模的解决方案中

II

> 分布式系统设计面临哪些挑战?请举例回答。

- 系统设计
    - 正确的接口设计和抽象
    - 如何换分功能和可扩展性
- 一致性
    - 如何一致性共享数据
- 容错
    - 如何保障系统在出现失效情况下正常运行
- 不同的部署场景
    - 集群
    - 广域分布
    - 传感网络
- 实现
    - 如何最大化并行
    - 性能瓶颈是什么
    - 如何平衡负载

III

> 请从一些开源分布式软件中找出能够体现透明性的样例代码，并解释是何种类型的透明性。

我找的是`GlusterFS`，这是一种开源分布式文件系统，在他的GitHub上的代码中，我找到了它的节点之间建立套接字并传输数据的代码，网址 https://github.com/gluster/glusterfs/blob/master/events/src/glustereventsd.py ，截取的代码如下：

```
class
GlusterEventsRequestHandler(socketserver.BaseRequestHandler):

    def handle(self):
        data = self.request[0].strip()
        if sys.version_info >= (3,):
            data = self.request[0].strip().decode("utf-8")

        logger.debug("EVENT: {0} from {1}".format(repr(data),

 self.client_address[0]))
        try:
            # Event Format <TIMESTAMP> <TYPE> <DETAIL>
            ts, key, value = data.split(" ", 2)
        except ValueError:
            logger.warn("Invalid Event Format {0}".format(data))
            return

        data_dict = {}
        try:
            # Format key=value;key=value
            data_dict = dict(x.split('=') for x in
value.split(';'))
        except ValueError:
            logger.warn("Unable to parse Event {0}".format(data))
            return

        for k, v in data_dict.items():
            try:
                if k in AUTO_BOOL_ATTRIBUTES:
                    data_dict[k] = boolify(v)
                if k in AUTO_INT_ATTRIBUTES:
                    data_dict[k] = int(v)
            except ValueError:
```

```
                # Auto Conversion failed, Retain the old value
                continue

        try:
            # Event Type to Function Map, Received event data will be in
            # the form <TIMESTAMP> <TYPE> <DETAIL>, Get Event name for the
            # received Type/Key and construct a function name starting with
            # handle_ For example: handle_event_volume_create
            func_name = "handle_" + all_events[int(key)].lower()
        except IndexError:
            # This type of Event is not handled?
            logger.warn("Unhandled Event: {0}".format(key))
            func_name = None

        if func_name is not None:
            # Get function from handlers module
            func = getattr(handlers, func_name, None)
            # If func is None, then handler unimplemented for that event.
            if func is not None:
                func(ts, int(key), data_dict)
            else:
                # Generic handler, broadcast whatever received
                handlers.generic_handler(ts, int(key), data_dict)


def signal_handler_sigusr2(sig, frame):
    utils.load_all()
    utils.restart_webhook_pool()


def UDP_server_thread(sock):
    sock.serve_forever()


def init_event_server():
```

```
    utils.setup_logger()
    utils.load_all()
    utils.init_webhook_pool()

    port = utils.get_config("port")
    if port is None:
        sys.stderr.write("Unable to get Port details from
Config\n")
        sys.exit(1)

    # Creating the Eventing Server, UDP Server for IPv4 packets
    try:
        serverv4 = UDPServerv4((SERVER_ADDRESSv4, port),
                   GlusterEventsRequestHandler)
    except socket.error as e:
        sys.stderr.write("Failed to start Eventsd for IPv4:
{0}\n".format(e))
        sys.exit(1)
    # Creating the Eventing Server, UDP Server for IPv6 packets
    try:
        serverv6 = UDPServerv6((SERVER_ADDRESSv6, port),
                   GlusterEventsRequestHandler)
    except socket.error as e:
        sys.stderr.write("Failed to start Eventsd for IPv6:
{0}\n".format(e))
        sys.exit(1)
    server_thread1 = threading.Thread(target=UDP_server_thread,
                     args=(serverv4,))
    server_thread2 = threading.Thread(target=UDP_server_thread,
                     args=(serverv6,))
    server_thread1.start()
    server_thread2.start()
```

这里体现了访问和位置的透明性

- 访问透明性：可以看出无论传输的数据有什么表示形式，在传输的时候都会打包成同一种格式，隐藏数据表示形式的不同
- 位置透明性：可以看出申请数据的数据包和答复的数据包，资源的位置被隐藏，体现了位置的透明性