分布式系统

作业一

中山大学计算机学院　计算机科学与技术

19335174　施天予

一、请解释（分布式）透明性的含义，并且给出各种类型的透明性实例。

透明性：隐藏进程和资源在多台计算机上分布这一事实。

- 访问：多个计算机用不同操作系统，它们文件命名方式不同，引发的文件操作方式的差异应该对用户和应用程序隐藏。
- 位置：从www.prehall.com这个 URL 看不出 Prentice Hall 的主 Web 服务器的所在位置。
- 迁移：资源移动但不会影响访问方式，用户不会感觉到资源移动。
- 重定位：移动通信用户从一个地点到另一个地点，可一直使用无线膝上型计算机，甚至不用中断连接。
- 复制：同个资源的多个副本名字相同，用户以为只有一个。
- 并发：用户不会感觉到他人也在使用自己正使用的资源。
- 故障：用户不会注意到某个资源无法正常工作，以及系统随后从故障中恢复的过程。

二、可以通过应用多种技术来取得扩展性，都有哪些技术？

分布式系统有三种扩展方式，分别为规模扩展，地理扩展和管理扩展。

扩展技术：

- 隐藏通信延迟：尽量避免等待远程服务对请求的影响，利用异步通信
- 分布：在多个机器上划分数据和计算
- 复制：在多个不同的机器上创建多个数据副本

三、分布式系统设计存在多个谬误，请针对不同的谬误给出一些具体的技术修正方案。比如考虑到网络是不可靠的，进行请求重新连接等。

八个谬误

- 网络可靠：网络并不可靠，可以采用主动事务数据复制机制来使得网络不可靠的情况下每种信息完成传输，或多次请求来进行网络的连接。
- 延迟为零：实际的网络情况下延时不为零，可以采用云端技术和云框架进行处理，将事务数据存储到云端。

- 传输带宽无限：带宽容量有限，传输数据的量有可能大于带宽的总量。可以利用数据复制技术同时运行多个数据库从而进行分布式的分流，管理员也可以对特定的文件数据设置更高的优先级。
- 网络安全：网络并不安全，可以通过设置防火墙来降低入侵的可能性。
- 拓扑结构不会改变：采用云计算的方式，数据能够在不同环境之间复制来保持一致性。
- 只有一位管理员：管理员不可能只有一个，可以加入管理员管理控制系统来解决。
- 传输成本为零：为了降低传输成本，采用主动架构，让一切服务器与集群拥有可读取性和可写入性，且始终彼此同步并不存在计划外停机，能够达到百分之一百的资源利用率。
- 网络同构：网络之间并不同构，当前采用跨平台的开发方案来应对不同的网络环境，采用多环境适用的软件。

四、请从一些开源分布式软件如 Hadoop、Ceph 分布式文件系统、Apache Httpd、Spark 等找出 1-2 处能够体现透明性的样例代码，并解释是何种类型的透明性。

分布式系统 consul 中的 api.go 文件中各种 API 接口，体现了访问的透明性，对于本地的机器和位于网络上的机器的操作都采用了一致的接口，使得系统被认为是一个独立的整体。

```
// Client provides a client to the Consul API
type Client struct {
    modifyLock sync.RWMutex
    headers   http.Header

    config Config
}
// Headers gets the current set of headers used for requests. This returns a
// copy; to modify it call AddHeader or SetHeaders.
func (c *Client) Headers() http.Header {
    c.modifyLock.RLock()
    defer c.modifyLock.RUnlock()

    if c.headers == nil {
        return nil
    }

    ret := make(http.Header)
    for k, v := range c.headers {
        for _, val := range v {
            ret[k] = append(ret[k], val)
        }
    }
```

```
24      return ret
25  }
26  // AddHeader allows a single header key/value pair to be added
27  // in a race-safe fashion.
28  func (c *Client) AddHeader(key, value string) {
29      c.modifyLock.Lock()
30      defer c.modifyLock.Unlock()
31      c.headers.Add(key, value)
32  }
33  // SetHeaders clears all previous headers and uses only the given
34  // ones going forward.
35  func (c *Client) SetHeaders(headers http.Header) {
36      c.modifyLock.Lock()
37      defer c.modifyLock.Unlock()
38      c.headers = headers
39  }
```