参考：

@

大白话理解“最晦涩”的Paxos算法及在数据库高可用上的使用

https://www.toutiao.com/i6509964089970655747/

@BigTable 有什么值得称道（牛）的地方

https://www.zhihu.com/question/19551534

@重读 Amazon Dynamo 论文有感

<https://zhuanlan.zhihu.com/p/98640498>

https://zhuanlan.zhihu.com/p/101965292

———- ——-———- ——-———- ——-———- ——-———- ——-———- ——-

Q1

1.1

第10章36页

1.2

好处1:

计算和存储可以分别按需进行扩展，而不是一切扩展

例子：比如我存储照片的空间不足了，我直接增加amazon s3硬盘就就行，计算完全不需要增加

好处2:

不同集群，可以选择更适合的服务器，可以有不同的升级周期

例子：比如存储我可以用amazon s3使用更偏重于硬盘的服务器，计算可以偏重于内存和cpu的服务器，相互之间不影响。而且不同服务器集群可以有不同的升级周期，比如计算可以随便几星期升级和更换，硬盘服务器几年一换

好处3:

区分了运维人员不同的责任，方便管理

例子：如果负责硬盘和ssd维护的人负责存储，负责内存和cpu调优的人员专注于计算层维护，不需要一个运维既懂存储硬件又懂cpu和内存，提高了效率，方便招聘

1.3

chunck version number 是：由客户端们节点汇报给chunServer询问当前chunk的lesase和其他节点位置，然后chunserver生成的。

mutation seiral number是： 是由本地节点自己维护的状态，因为有全局变异勋勋，然后传输到总节点的。

翻译：

the global mutation order is defined first by the lease grant order chosen by the master, and within a lease by the serial numbers assigned by the primary.

the mutation to its own local state in serial number order. The primary forwards the write request to all secondary replicas

1.4

shuffle就是混淆、洗牌的意思。

mapredue中shuflle是：

map端五则贵的输出按指定规则打断的数据，方便reduce接受

按照partition、key对中间结果进行排序合并,输出给reduce线程

Spark的Shuffle是在MapReduce Shuffle基础上进行的调优。其实就是对排序、合并逻辑做了一些优化。在Spark中Shuffle write相当于MapReduce 的map，Shuffle read相当于MapReduce 的reduce。

Spark丰富了任务类型，有些任务之间数据流转不需要通过Shuffle，但是有些任务之间还是需要通过Shuffle来传递数据，比如宽依赖的group by key以及各种by key算子。宽依赖之间会划分stage，而Stage之间就是Shuffle，如下图中的stage0，stage1和stage3之间就会产生Shuffle。

1.5

“application”, “job”, “task” and “task attempt” in YARN.   
In terms of YARN,

the programs that are being run on a cluster are called **applications**.

In terms of MapReduce they are called **jobs**.

So, if you are running MapReduce on YARN, job and application are the same thing (if you take a close look, job ids and application ids are the same).

MapReduce job consists of several tasks (they could be either map or reduce tasks, this are **task**).

If a task fails, it is launched again on another node. Those are **task attempts**.

———- ——-———- ——-———- ——-———- ——-———- ——-———- ——-

Q2

2.1

docker container run -p 10000:8000 -it jupyter/r-notebook /bin/bash --name r-notebook-container0 -v $(pwd):/jupyter/r-notebook/data

2.2

because in 2.1 I have : /bin/bash

when everyting is ok, after I run the command , it should return

roo@abelname: /app#

then I run in roo@abelname: /app#: jupyter notebook

when the notebook is running, I open my browser，visit <http://127.0.0.1:8000>

2.3

sudo docker exec -it r-notebook-container0 ps -eaf

2.4

docker container run -p 10000:8001 -it jupyter/pyspark-notebook /bin/bash --name pyspark-notebook1 -v $(pwd):/jupyter/pyspark-notebook/data

2.5

because in 2.4 I have : /bin/bash

when everyting is ok, after I run the command , it should return

roo@abelname: /app#

then I run in roo@abelname: /app#: jupyter notebook

when the notebook is running, I open my browser，visit 1

2.6

It should be at:

$(pwd):/jupyter/pyspark-notebook/data/project/\*. ipynb

2.7

虚拟化就是vm多了一层guest OS，同时Hypervisor会对硬件资源进行虚拟化，

docker直接使用硬件资源

虚拟机的Guest层，还有Hypervisor层在Docker上已经被Docker Engine层所取代，在这里我们需要知道，Guest OS 是虚拟机安装的操作系统，是一个完整的系统内核，另外Hypervisor可以理解为硬件虚拟化平台，它在后Host OS以内核驱动的形式存在。

虚拟机实现资源的隔离的方式是利用独立的Guest OS，以及利用Hypervisor虚拟化CPU、内存、IO等设备来实现的，对于虚拟机实现资源和环境隔离的方案，Docker显然简单很多。

然后Docker并没有和虚拟机一样利用一个独立的Guest OS执行环境的隔离，它利用的是目前当前Linux内核本身支持的容器方式，实现了资源和环境的隔离，简单来说，Docker就是利用Namespace 实现了系统环境的隔离，利用了cgroup实现了资源的限制，利用镜像实例实现跟环境的隔离。

举例子，本例子中如果我使用虚拟化，我可以想到2种方式：

方式1:我就用python3 -m venv tutorial-env 构建2个不同环境，让她们python版本之间和安装的库之间相互隔离。（不知道符合课堂标准不，可以不写方式1，只写方式2）

方式2:或者我使用虚拟机，每一个/r-notebook，pyspark-notebook一个虚拟机

虚拟机虽然也隔离了，明显多一层虚拟机os（操作系统层），没有docker直接用到硬件层，更轻更简单。

———- ——-———- ——-———- ——-———- ——-———- ——-———- ——-

Q3