http://dwgeek.com/commonly-used-hadoop-hive-commands.html/

http://dwgeek.com/hadoop-hive-analytic-functions-examples.html/

http://dwgeek.com/category/apache-spark/

- Spark can parallelize reading a single gzip file.

  The best you can do split it in chunks that are gzipped.

  However, Spark is really slow at reading gzip files. You can do this to speed it up:

```
file_names_rdd = sc.parallelize(list_of_files, 100)
lines_rdd = file_names_rdd.flatMap(lambda _: gzip.open(_).readlines()
```

- Below formula is used to calculate the cluster size of hadoop:
  H=crs/(1-i)
  Where c=average compression ratio. This depends upon the type of compression used and size of the data. When no compression is used, c value will be 1.
  R=replication factor. It is set to 3 by default in production cluster.
  S = size of data to be moved to Hadoop. This could be a combination of historical data and incremental data. The incremental data can be daily for example and projected over a period of time (3 years for example).
  i = intermediate factor. It is usually 1/3 or 1/4. Hadoop's working space dedicated to storing intermediate results of Map phase.
  Example: With no compression i.e. c=1, a replication factor of 3, an intermediate factor of .25=1/4 H= 13S/(1-1/4)=3S/(3/4)=4S With the assumptions above, the Hadoop storage is estimated to be 4 times the size of the initial data size.
  This is the formula to estimate the number of data nodes (n):
  n= H/d = c*r*S/(1-i)*d

  where d= disk space available per node. All other parameters remain the same as in 1. Example: If 8TB is the available disk space per node (10 disks with 1 TB , 2 disk for operating system etc were excluded.). Assuming initial data size is 600 TB. n= 600/8=75 data nodes needed

- **Cluster**: A cluster in Hadoop is used for distirbuted computing, where it can store and analyze huge amount structured and unstructured data
  To setup a cluster we need the below :
  1) Client machine: which will make request to read and write the data with the help of name and data node
  2) Name Node: Will take care of storing and data with the help of HDFS and parallel computing with the help oof Map reduce
  3) Data Node: Will make all the processing/computation of data. Data node will receive instruction from Name node and process them accordingly

4) Resource manager and Task manager: Which resides in Name node and data node for managing resource

The decision to make to prepare cluster will consider the below points.

1) Ingestion rate: It is the data we can expect on daily basis on an average
2) Replication factor: It will help to create the data copies which can be used when there is a failure a data node.It can be specified in **hdfs-site.xml**. for multi node cluster **replication factor by default is 3** and for single node cluster, it is a 1.Replication factor will occupy the disk space depending on the factor count.It can be modified depending on t he rate we receive the data
3) Size of hard disks: Size of disk which will be installed each data node
4) Buffer memory: Amount of memory kept aside for storing intermediate results of map results

Daily Ingestion rate ***1 TB***
Replication Factor ***3***
Size of Hard Disk ***48 (12 * 4 TB)***
Buffer memory ***25% or 0.25***
Memory to be stored in HD ***1 * 3 = 3TB***
Memory can be used for storing and processing ***48-(48*0.25) = 36 TB***
Number of Nodes reqd ***(3*365)/36 =~31 Nodes***

- **Table 1.1 Sizing Recommendations**

Machine Type	**Workload Pattern/ Cluster Type**	**Storage**[1]	**Processor (# of Cores)**	**Memory (GB)**	**Network**
Slaves	Balanced workload	Twelve 2-3 TB disks	8	128-256	1 GB onboard, 2x10 GBE mezzanine/external
	Compute-intensive workload	Twelve 1-2 TB disks	10	128-256	1 GB onboard, 2x10 GBE mezzanine/external
	Storage-heavy workload	Twelve 4+ TB disks	8	128-256	1 GB onboard, 2x10 GBE mezzanine/external

Machine Type	**Workload Pattern/ Cluster Type**	**Storage**[1]	**Processor (# of Cores)**	**Memory (GB)**	**Network**
NameNode	Balanced workload	Four or more 2-3 TB RAID 10 with spares	8	128-256	1 GB onboard, 2x10 GBE mezzanine/external
ResourceManager	Balanced workload	Four or more 2-3 TB RAID 10 with spares	8	128-256	1 GB onboard, 2x10 GBE mezzanine/external

Some personal notes:

- Bare minimum, depending on replication factor of 3, you need about 50TB (10x3=30TB 80% rule: 40TB usable, this give you 8TB to work with) - So 5 Nodes at 10TB a piece for HDFS
- HDFS can only use a maximum of 80% of total cluster space
- More nodes = faster YARN jobs
- Hive & PIG can read compressed data as if it was uncompressed
- I'd personally start with 5 nodes, 3 for a zookeeper quorum and 2 can be assigned NameNodes
- All of them can be set to DataNodes
- Don't forget about an Ambari server, and make this a VM...you'll need backups and snapshots daily
- Upgrades/updates can blow stuff up, always have a plan if a job fails
- Sometimes the most basic install doesn't go as planned ;)
- **Start small and scale out! Hadoop is built on this type of thinking!**

Is NoSQL the right technology choice?

As data gets "bigger," measured in terms of volume, velocity, and variety, NoSQL is increasingly the right choice for application data. For applications (and developers) that prioritize speed, manageability, agile development, and easier horizontal scale, NoSQL is becoming the default.

But, as MongoDB used to highlight on its community site, sometimes a relational database is a better fit:

- **Problems requiring SQL**. If your application depends upon SQL for queries, you're likely going to be better off with an RDBMS.

- **Systems with a heavy emphasis on complex transactions such as banking systems and accounting**. There are workarounds and databases like Cassandra and MongoDB have evolved since this was written, making them better fits for applications that require multi-object transactions, for example. But these still aren't the ideal use case for NoSQL databases.
- **Traditional Non-Realtime Data Warehousing**. This is decreasingly the case, but historically this hasn't been a strong suit for NoSQL.

SPARK SPILL QUESTION:

Shuffle spill happens when there is not sufficient memory for shuffle data.

`Shuffle spill (memory)` - size of the deserialized form of the data in memory at the time of spilling
`shuffle spill (disk)` - size of the serialized form of the data on disk after spilling
Since deserialized data occupies more space than serialized data. So, Shuffle spill (memory) is more.

Noticed that this **spill memory size is incredibly large with big input data**.
In summary, you spill when the size of the RDD partitions at the end of the stage exceed the amount of memory available for the shuffle buffer.

You can:

1. Manually `repartition()` your prior stage so that you have smaller partitions from input.
2. Increase the shuffle buffer by increasing the memory in your executor processes (`spark.executor.memory`)
3. Increase the shuffle buffer by increasing the fraction of executor memory allocated to it (`spark.shuffle.memoryFraction`) from the default of 0.2. You need to give back `spark.storage.memoryFraction`.
4. Increase the shuffle buffer per thread by reducing the ratio of worker threads (`SPARK_WORKER_CORES`) to executor memory

Hadoop Hive Date Functions

Date types are highly *formatted* and very *complicated*. Each date value contains the **century, year, month, day, hour, minute, and second**. We shall see how to use the **Hadoop Hive date functions**with an examples. You can use these functions as **Hive date conversion functions** to manipulate the date data type as per the application requirements. Below are the most commonly used Hadoop Hive DateTime functions:

Date Function	Description
current_timestamp()	Returns the current date and time of the system. There is no now() function is Hadoop Hive.
current_date()	Returns the current date of the system without any time part.
add_months(timestamp date, int months)	Adds month value to specified date or timestamp values.
to_date(timestamp date)	Converts Hive timestamp value to date data type.
date_add(timestamp startdate, int days)	Adds days to specified timestamp value. Hive does not support interval data type in

	date_add. You can use interval type directly to add or substract from date values
date_sub(timestamp startdate, int days)	Substract specified number of days from the date or timestamp value.
datediff(timestamp enddate, timestamp startdate)	timestamp startdate) Returns number of days between the two date or timestamp values.
from_unixtime(bigint unixtime[, string format])	Converts the number of seconds from the Unix epoch to the specified time into a string.
month(timestamp date), minute(timestamp date), hour(timestamp date), day(timestamp date), second(timestamp date)	Returns month, minutes, hours, days, seconds from the timestamp.

trunc(timestamp, string unit)	Strips off fields from a TIMESTAMP value.
unix_timestamp()	Gets current time stamp using the default time zone.
unix_timestamp(string date)	Converts time string in format yyyy-MM-dd HH:mm:ss to Unix time stamp.
from_utc_timestamp(timestamp, string timezone)	Converts a specified UTC timestamp value into the appropriate value for a specified time zone

- **Get first day of the given timstamp using HiveQL**
  ```
  hive> select trunc(current_timestamp(), 'MONTH');
  2017-10-01
  ```

SQL Queries:

```
create table vikasemp(emp_id int, salary int);
insert into vikasemp values(1,1301);
insert into vikasemp values(2,1199);
insert into vikasemp values(3,1495);
insert into vikasemp values(4,3497);
insert into vikasemp values(5,2495);
insert into vikasemp values(6,1098);
```

insert into vikasemp values(7,879);

insert into vikasemp values(8,785);

1. Select emp_id, salary from (Select emp_id,salary,dense_rank() over (order by salary desc)as sal_rank from employee)a where sal_rank=2;

Create table vikasman(emp_id int, salary int, manager_id int);

Insert into vikasman values(1,1301,4);

insert into vikasman values(2,1199,7);

insert into vikasman values(3,1495,6);

insert into vikasman values(4,3497,);

insert into vikasman values(5,2495,4);

insert into vikasman values(6,1098,5);

insert into vikasman values(7,879,8);

insert into vikasman values(8,785,6);

```
vikasman.emp_id          vikasman.salary vikasman.manager_id

1        1301    4

2        1199    7

3        1495    6

5        2495    4

4        3497    NULL

7        879     8

6        1098    5

8        785     6
```

3 select distinct emp_id,emp_salary,manager_id,man_salary from (
select a.emp_id,a.salary as emp_salary,a.manager_id,c.salary as man_salary from vikasman a left join (select b.manager_id,a.salary from vikasman a join vikasman b on b.manager_id=a.emp_id)c on a.manager_id=c.manager_id)d where emp_salary > man_salary;

4 Select emp.emp_id from employee emp left join employee man on emp.emp_id=man.manager_id where man.manager_id is null;

create table vikastable1(x int, y int);

insert into vikastable1 values(20,20);

insert into vikastable1 values(20,21);

insert into vikastable1 values(23,22);

insert into vikastable1 values(22,23);

insert into vikastable1 values(21,20);

5 Select a.x,a.y from vikastable1 a where a.x*a.y=a.y*a.x and (a.x-a.y >0 or a.x-a.y=0);