http://www.pluralsight.com/training

Both databases perform well on reads where the hot data set fits in memory. Both also emphasize join-less data models (and encourage denormalization instead), and both provide indexes on[documents](http://www.mongodb.org/display/DOCS/Indexes) or [rows](http://www.datastax.com/dev/blog/whats-new-cassandra-07-secondary-indexes), although MongoDB's indexes are currently more flexible.

Cassandra's storage engine provides constant-time writes no matter how big your data set grows. Writes are more problematic in MongoDB, partly because of the b-tree based storage engine, but more because of the [global write lock](http://www.mongodb.org/display/DOCS/How+does+concurrency+work).

when we were assessing NoSQL databases, was the querying - Cassandra is basically just a giant key/value store, and querying is a bit fiddly (at least compared to MongoDB), so for performance you'd have to duplicate quite a lot of data as a sort of manual index. MongoDB, on the other hand, uses a "query by example" model . , Cassandra is the best for high insert volume

We decided to go with Cassandra and it's been really great. We don't have any benchmarking platform yet, but the initial tests shows that Cassandra outperforms MySql (About 3000% faster for writes). We're using Thrift to talk to Cassandra and it is a really active community behind it (mainly Twitter), so there aren't tons of articles, but the articles are very usefull. I'll let you know how this end

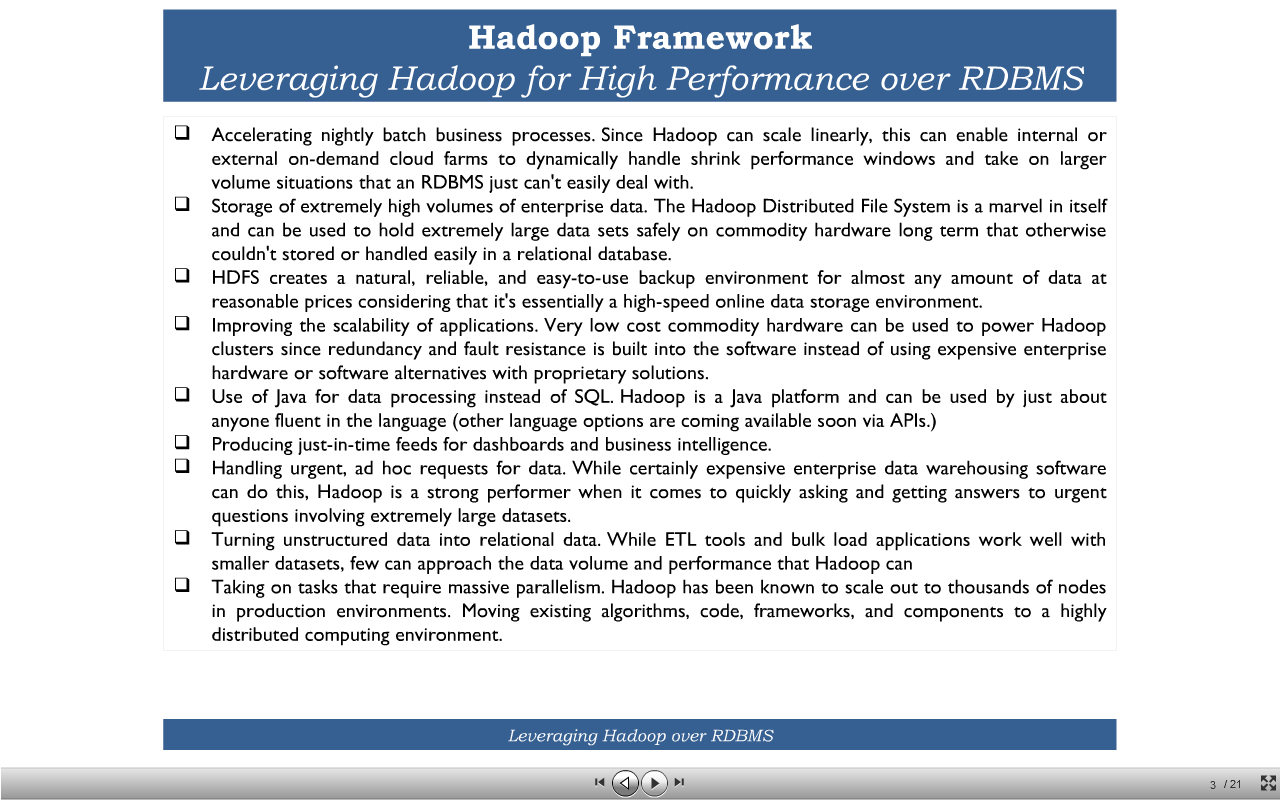
Master –slave configuration for DB failovers

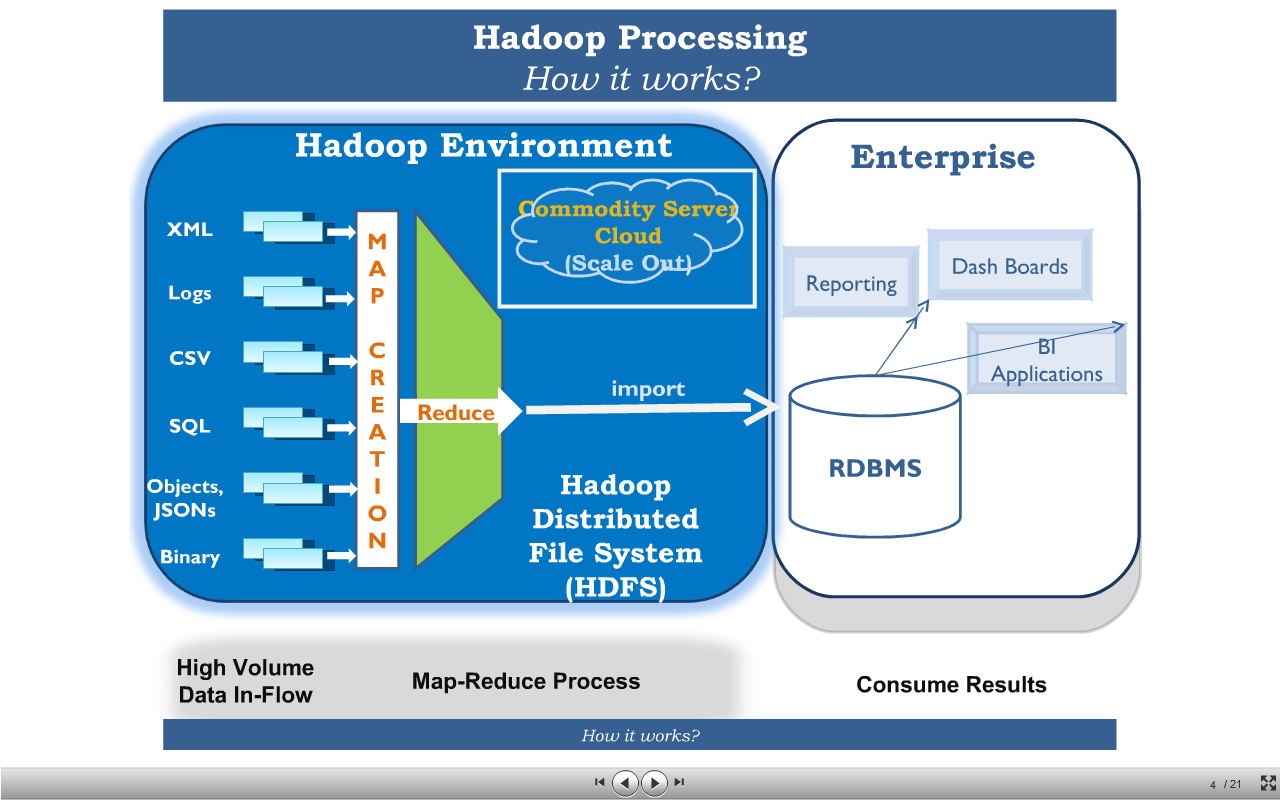
Throughput – the rate at which data transfers/delivers to the terminal/node etc.,

if we are testing a web application. If a web app receives 50 requests per second, but can only handle 30 transactions per second, the other 20 requests end up waiting in a queue. When presenting performance test results, throughput performance is often expressed as **transactions per second**, or **TPS**

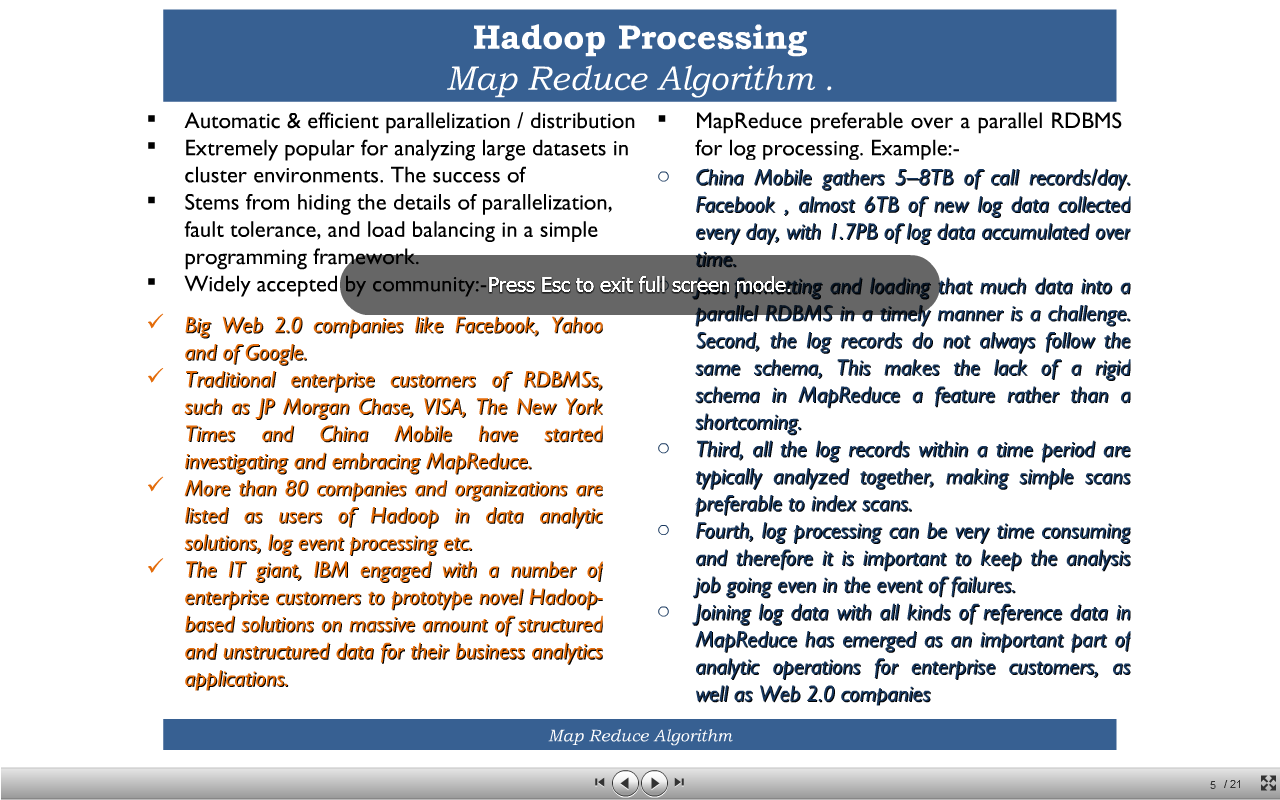
Latency – Time delay experienced on the system one way /round trip

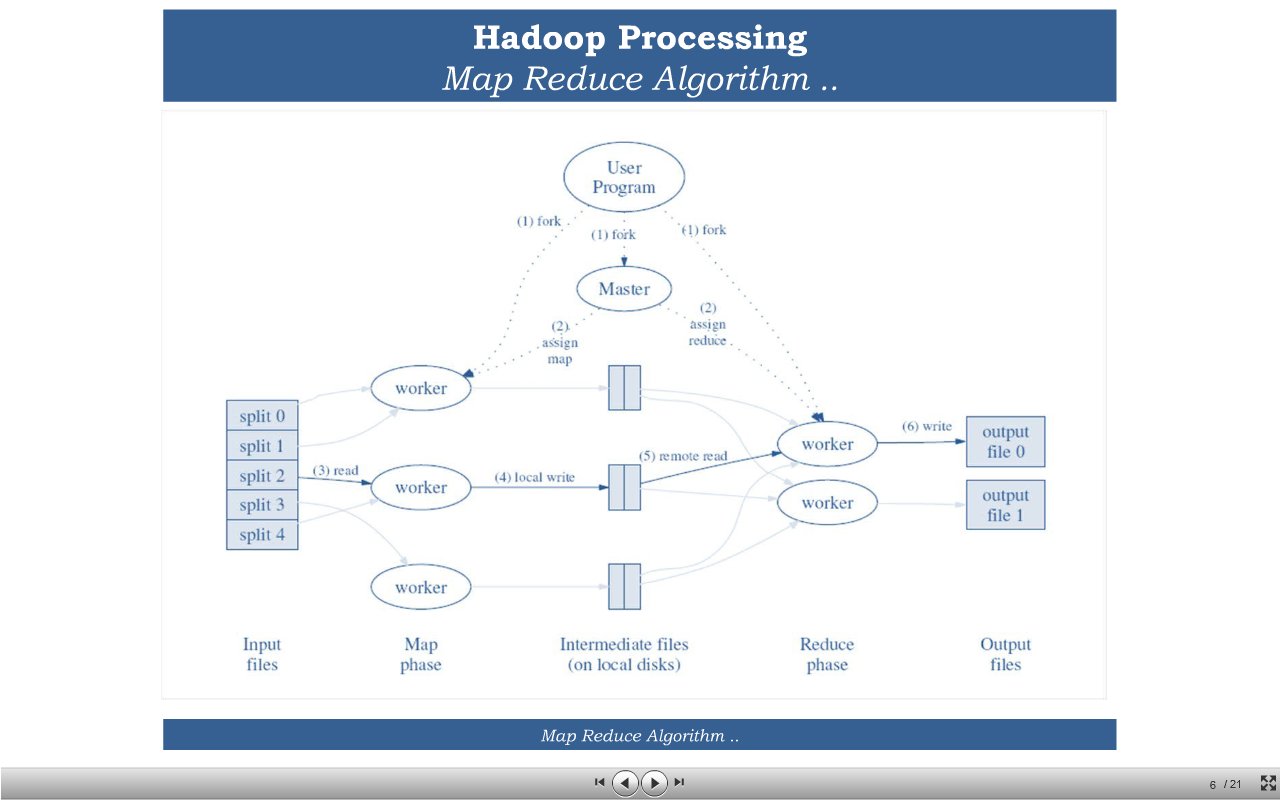
Memcached– distributed object caching system generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering

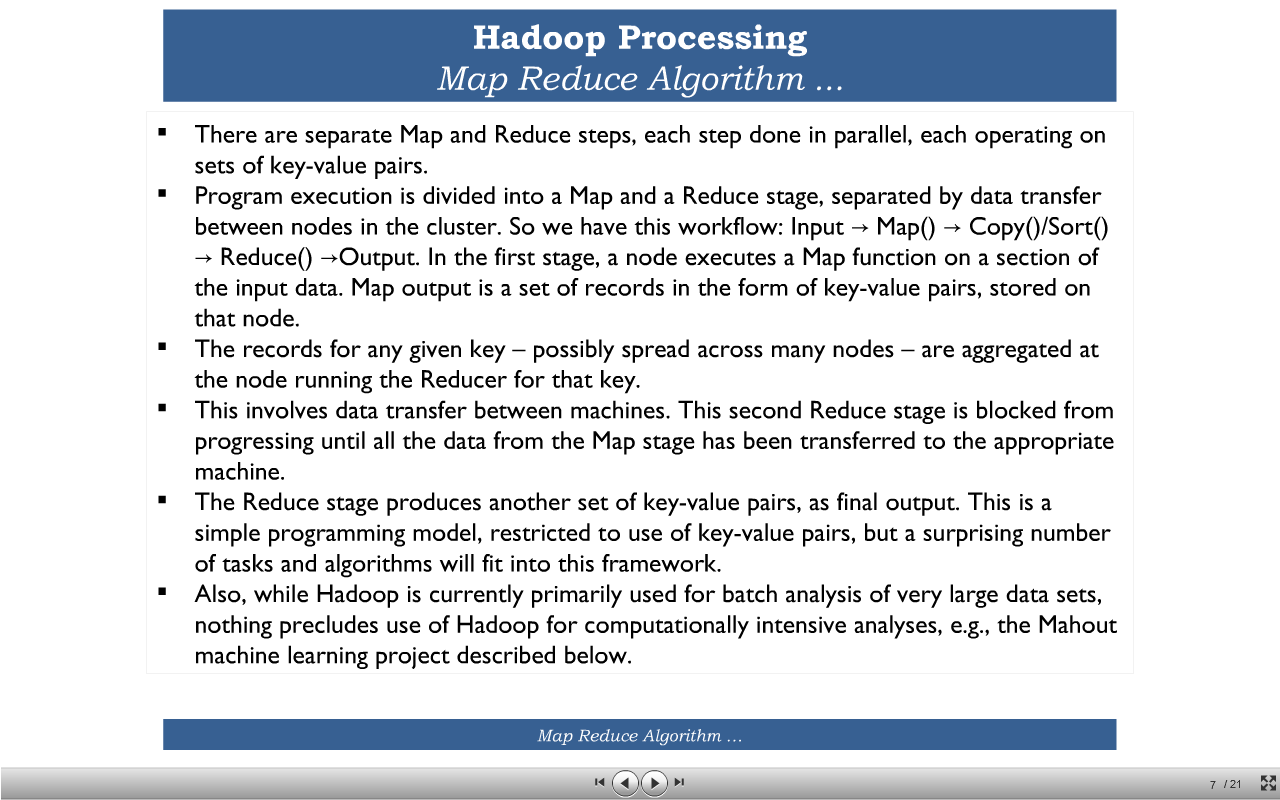




http://www.slideshare.net/koolhits/hadoop-a-natural-choice-for-data-intensive-log-processing







<http://www.slideshare.net/mongodb/webinar-how-financial-firms-create-a-single-customer-view-with-mongodb>

Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load-balanced if the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine.

Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more fine-grained.

On the other hand, if splits are too small, then the overhead of managing the splits and of map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of an HDFS block, 64 MB by default, although this can be changed for the cluster (for all newly created files), or specified when each file is created.

Map tasks write their output to the local disk, not to HDFS. Why is this? Map output is intermediate output: it’s processed by reduce tasks to produce the final output, and once the job is complete the map output can be thrown away. So storing it in HDFS, with replication, would be overkill. If the node running the map task fails before the map output has been consumed by the reduce task, then Hadoop will automatically

rerun the map task on another node to re-create the map output

Reduce tasks don’t have the advantage of data locality—the input to a single reduce task is normally the output from all mappers. In the present example, we have a single reduce task that is fed by all of the map tasks. Therefore, the sorted map outputs have to be transferred across the network to the node where the reduce task is running, where

they are merged and then passed to the user-defined reduce function. The output of the reduce is normally stored in HDFS for reliability.

Since the namenode holds filesystem metadata in memory, the limit to the number of files in a file system is governed by the amount of memory on the namenode

CDN– content delivery networks

Akamai

Lime light networks

Network file systems – dell ,NetApp, EMC2

**1-**[***Hadoop***](http://hadoop.apache.org/)**:** Hadoop is basically 2 things, a distributed file system (**HDFS**) which constitutes Hadoop's storage layer and a distributed computation framework(**MapReduce**) which constitutes the processing layer. You should go for Hadoop if your data is very huge and you have offline, batch processing kinda needs. Hadoop is not suitable for real time stuff. You setup a Hadoop cluster on a group of commodity machines connected together over a network(called as a cluster). You then store huge amounts of data into the HDFS and process this data by writing MapReduce programs(or jobs). Being distributed, HDFS is spread across all the machines in a cluster and MapReduce processes this scattered data locally by going to each machine, so that you don't have to relocate this gigantic amount of data.  
  
**2- *[Hbase](http://hbase.apache.org/" \t "_blank)* :**Hbase is a distributed, scalable, big data store, modelled after Google's BigTable. It stores data as key/value pairs. It's basically a database, a **NoSQL database** and like any other database it's biggest advantage is that it provides you random read/write capabilities. As I have mentioned earlier, Hadoop is not very good for your real time needs, so you can use Hbase to serve that purpose. If you have some data which you want to access real time, you could store it in Hbase. Hbase has got it's own set of very good API which could be used to push/pull the data. Not only this, Hbase can be seamlessly integrated with MapReduce so that you can do bulk operation, like indexing, analytics etc etc.  
  
**Tip :** You could use Hadoop as the **repository** for your static data and Hbase as the **datastore** which will hold data that is probably gonna change over time after some processing.  
  
**3-**[***Hive***](http://hive.apache.org/) **:**Originally developed by Facebook, Hive is basically a**data warehouse**. It sits on top of your Hadoop cluster and provides you an SQL like interface to the data stored in your Hadoop cluster. You can then write SQLish queries using Hive's query language, called as HiveQL and perform operations like store, select, join, and much more. It makes processing a lot easier as you don't have to do lengthy, tedious coding. Write simple Hive queries and get the results. Isn't that cool??RDBMS folks will definitely love it. Simply map HDFS files to Hive tables and start querying the data. Not only this, you could map Hbase tables as well, and operate on that data.  
  
**Tip :** Use Hive when you have warehousing needs and you are good at SQL and don't want to write MapReduce jobs. One important point though, Hive queries get converted into a corresponding MapReduce job under the hood which runs on your cluster and gives you the result. Hive does the trick for you. But each and every problem cannot be solved using HiveQL. Sometimes, if you need really fine grained and complex processing you might have to take MapReduce's shelter.  
  
**4-**[***Pig***](http://pig.apache.org/)**:**Pig is a **dataflow language** that allows you to process enormous amounts of data very easily and quickly by repeatedly transforming it in steps. It basically has 2 parts, the Pig**Interpreter** and the language, **PigLatin**. Pig was originally developed at Yahoo and they use it extensively. Like Hive, PigLatin queries also get converted into a MapReduce job and give you the result. You can use Pig for data stored both in HDFS and Hbase very conveniently. Just like Hive, Pig is also really efficient at what it is meant to do. It saves a lot of your effort and time by allowing you to not write MapReduce programs and do the operation through straightforward Pig queries.  
  
**Tip :**Use Pig when you want to do a lot of transformations on your data and don't want to take the pain of writing MapReduce jobs.  
  
**5- *[Sqoop](http://sqoop.apache.org/" \t "_blank)* :**Sqoop is a tool that allows you to transfer data between relational databases and Hadoop. It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import. Not only this, imports can also be used to populate tables in Hive or HBase. Along with this Sqoop also allows you to export the data back into the relational database from the cluster.  
  
**Tip :**Use Sqoop when you have lots of legacy data and you want it to be stored and processed over your Hadoop cluster or when you want to incrementally add the data to your existing storage.  
  
**6- *[Oozie](http://oozie.apache.org/" \t "_blank)* :**Now you have everything in place and want to do the processing but find it crazy to start the jobs and manage the workflow manually all the time. Specially in the cases when it is required to chain multiple MapReduce jobs together to achieve a goal. You would like to have some way to automate all this. No worries, Oozie comes to the rescue. It is a scalable, reliable and extensible **workflow scheduler system**. You just define your workflows(which are **Directed Acyclical Graphs**) once and rest is taken care by Oozie. You can schedule MapReduce jobs, Pig jobs, Hive jobs, Sqoop imports and even your Java programs using Oozie.  
  
**Tip :**Use Oozie when you have a lot of jobs to run and want some efficient way to automate everything based on some time (frequency) and data availabilty.  
  
**7-**[***Flume***](http://flume.apache.org/)**/*[Chukwa](http://incubator.apache.org/chukwa/" \t "_blank)* :**Both Flume and Chukwa are **data aggregation tools** and allow you to aggregate data in an efficient, reliable and distributed manner. You can pick data from some place and dump it into your cluster. Since you are handling BigData, it makes more sense to do it in a distributed and parallel fashion which both these tools are very good at. You just have to define your flows and feed them to these tools and rest of things will be done automatically by them.  
  
**Tip :**Go for Flume/Chukwa when you have to aggregate huge amounts of data into your Hadoop environment in a distributed and parallel manner.  
 **8-**[***Avro***](http://avro.apache.org/)**:**Avro is a **data serialization system**. It provides functionalities similar to systems like Protocol Buffers, Thrift etc. In addition to that it provides some other significant features like rich data structures, a compact, fast, binary data format, a container file to store persistent data, RPC mechanism and pretty simple dynamic languages integration. And the best part is that Avro can easily be used with MapReduce, Hive and Pig. Avro uses **JSON** for defining data types.  
  
**Tip :**Use Avro when you want to serialize your BigData with good flexibility.  
  
The list is actually pretty big, but I have covered only the most significant tools. Over time if I feel like something else should be mentioned here I would definitely do that.

**1-**[***Hadoop***](http://hadoop.apache.org/)**:** Hadoop is basically 2 things, a distributed file system (**HDFS**) which constitutes Hadoop's storage layer and a distributed computation framework(**MapReduce**) which constitutes the processing layer. You should go for Hadoop if your data is very huge and you have offline, batch processing kinda needs. Hadoop is not suitable for real time stuff. You setup a Hadoop cluster on a group of commodity machines connected together over a network(called as a cluster). You then store huge amounts of data into the HDFS and process this data by writing MapReduce programs(or jobs). Being distributed, HDFS is spread across all the machines in a cluster and MapReduce processes this scattered data locally by going to each machine, so that you don't have to relocate this gigantic amount of data.  
  
**2- *[Hbase](http://hbase.apache.org/" \t "_blank)* :**Hbase is a distributed, scalable, big data store, modelled after Google's BigTable. It stores data as key/value pairs. It's basically a database, a **NoSQL database** and like any other database it's biggest advantage is that it provides you random read/write capabilities. As I have mentioned earlier, Hadoop is not very good for your real time needs, so you can use Hbase to serve that purpose. If you have some data which you want to access real time, you could store it in Hbase. Hbase has got it's own set of very good API which could be used to push/pull the data. Not only this, Hbase can be seamlessly integrated with MapReduce so that you can do bulk operation, like indexing, analytics etc etc.  
  
**Tip :** You could use Hadoop as the **repository** for your static data and Hbase as the **datastore** which will hold data that is probably gonna change over time after some processing.  
  
**3-**[***Hive***](http://hive.apache.org/) **:**Originally developed by Facebook, Hive is basically a**data warehouse**. It sits on top of your Hadoop cluster and provides you an SQL like interface to the data stored in your Hadoop cluster. You can then write SQLish queries using Hive's query language, called as HiveQL and perform operations like store, select, join, and much more. It makes processing a lot easier as you don't have to do lengthy, tedious coding. Write simple Hive queries and get the results. Isn't that cool??RDBMS folks will definitely love it. Simply map HDFS files to Hive tables and start querying the data. Not only this, you could map Hbase tables as well, and operate on that data.  
  
**Tip :** Use Hive when you have warehousing needs and you are good at SQL and don't want to write MapReduce jobs. One important point though, Hive queries get converted into a corresponding MapReduce job under the hood which runs on your cluster and gives you the result. Hive does the trick for you. But each and every problem cannot be solved using HiveQL. Sometimes, if you need really fine grained and complex processing you might have to take MapReduce's shelter.  
  
**4-**[***Pig***](http://pig.apache.org/)**:**Pig is a **dataflow language** that allows you to process enormous amounts of data very easily and quickly by repeatedly transforming it in steps. It basically has 2 parts, the Pig**Interpreter** and the language, **PigLatin**. Pig was originally developed at Yahoo and they use it extensively. Like Hive, PigLatin queries also get converted into a MapReduce job and give you the result. You can use Pig for data stored both in HDFS and Hbase very conveniently. Just like Hive, Pig is also really efficient at what it is meant to do. It saves a lot of your effort and time by allowing you to not write MapReduce programs and do the operation through straightforward Pig queries.  
  
**Tip :**Use Pig when you want to do a lot of transformations on your data and don't want to take the pain of writing MapReduce jobs.  
  
**5- *[Sqoop](http://sqoop.apache.org/" \t "_blank)* :**Sqoop is a tool that allows you to transfer data between relational databases and Hadoop. It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import. Not only this, imports can also be used to populate tables in Hive or HBase. Along with this Sqoop also allows you to export the data back into the relational database from the cluster.  
  
**Tip :**Use Sqoop when you have lots of legacy data and you want it to be stored and processed over your Hadoop cluster or when you want to incrementally add the data to your existing storage.  
  
**6- *[Oozie](http://oozie.apache.org/" \t "_blank)* :**Now you have everything in place and want to do the processing but find it crazy to start the jobs and manage the workflow manually all the time. Specially in the cases when it is required to chain multiple MapReduce jobs together to achieve a goal. You would like to have some way to automate all this. No worries, Oozie comes to the rescue. It is a scalable, reliable and extensible **workflow scheduler system**. You just define your workflows(which are **Directed Acyclical Graphs**) once and rest is taken care by Oozie. You can schedule MapReduce jobs, Pig jobs, Hive jobs, Sqoop imports and even your Java programs using Oozie.  
  
**Tip :**Use Oozie when you have a lot of jobs to run and want some efficient way to automate everything based on some time (frequency) and data availabilty.  
  
**7-**[***Flume***](http://flume.apache.org/)**/*[Chukwa](http://incubator.apache.org/chukwa/" \t "_blank)* :**Both Flume and Chukwa are **data aggregation tools** and allow you to aggregate data in an efficient, reliable and distributed manner. You can pick data from some place and dump it into your cluster. Since you are handling BigData, it makes more sense to do it in a distributed and parallel fashion which both these tools are very good at. You just have to define your flows and feed them to these tools and rest of things will be done automatically by them.  
  
**Tip :**Go for Flume/Chukwa when you have to aggregate huge amounts of data into your Hadoop environment in a distributed and parallel manner.  
 **8-**[***Avro***](http://avro.apache.org/)**:**Avro is a **data serialization system**. It provides functionalities similar to systems like Protocol Buffers, Thrift etc. In addition to that it provides some other significant features like rich data structures, a compact, fast, binary data format, a container file to store persistent data, RPC mechanism and pretty simple dynamic languages integration. And the best part is that Avro can easily be used with MapReduce, Hive and Pig. Avro uses **JSON** for defining data types.  
  
**Tip :**Use Avro when you want to serialize your BigData with good flexibility.  
  
The list is actually pretty big, but I have covered only the most significant tools. Over time if I feel like something else should be mentioned here I would definitely do that.