Presentation Script

Good afternoon ladies and gentlemen, I am Zhi An. I have been here since February 27 working on a small project on big data.
Hadoop, some of you may know it, but just keep quiet and let me talk if you do. If you don't, I'll introduce this elephant to you in a short while.

Today's presentation will be pretty light, I am only covering a few things, as you can see...

First, I will first talk about the problem sets that were presented to me.
The two problem sets are actually pretty similar. The first case has to do with web capture of multiple file formats, which results in 30 gigabtyes of data daily and an annual growth of 5%. This case will require a database which we can write and read records multiple times. Case two deals with an even larger volume of data because this is archival data. So we have trillions of records and up to terrabytes of information. In contrast, we only need to write once.
OR
I will skip the first point because Sun Jing has helped me with it. He is always that helpful!

Well if the two slides before didn't make sense, we only need to be aware of these requirements.

- We need a database that is capable of handling what we call Big Data, which is a term to describe huge amounts of data, up to terrabytes and even petabtyes.
- Our storage needs to be hardy and reliable, it should not go down much, and even if it does, recovery should be as simple and quick as possible.
- We want quick and random access to the records for analysis.
- We want to be able to handle multiple file types, HTML, documents, PDF, images etc.

A possible hero to save us is Hadoop. This is the official definition of Hadoop, created by me. Essentially it means having many affordable computers and combining their storage and computation power.

Hadoop has its roots in Nutch, which was originally an open source search engine created by Doug Cutting. Blah blah blah he joined Yahoo and Hadoop is born!

So what are the challenges are that inspired the creation of Hadoop? First, hardwares fail, very often. I'm sure you all experienced that before. So the system needed to be able to recover quickly and smoothly. There are finite resources: processing time, storage, bandwidth; the system should be able to optimize the resources for maintenance and core processing. And when you have so many systems, data need to be synchronized.

Hadoop is created to solve these problems. Data is replicated across nodes and monitored to ensure integrity. Data processing is quick because of MapReduce, which we will dive into more details later. Hadoop system has high scalability because it is easy to add new machines with very minimal modification of MapReduce code.

We'll take a plunge into Hadoop and find out what it is. Hadoop is made up of two core technologies, the Hadoop Distributed File System, hereby known as HDFS and MapReduce. HDFS is a file system that is designed to be highly fault-tolerant and spread out across multiple clusters of computers. It comprises of a single master server, known as the NameNode, and one or more slave computers, known as the DataNodes. NameNodes store metadata and coordinates client access to files, while DataNodes stores files. HDFS is kept up via Heartbeats

that DataNodes send to NameNode to indicate that it is up and running. The way HDFS achieves fault-tolerance is via replication. Imagine a single file made up of blocks 1 to 5. These blocks are replicated and stored on the DataNodes across the cluster.

The next core technology of Hadoop is MapReduce. This is a simple flow of the MapReduce process. Yes this is already simplified. Basically, MapReduce allows for the processing of large amounts of data in parallel across multiple computers in a fault-tolerant manner. It achieves that by splitting up the input into various portions which are processed by Mappers in parallel. Mappers take in a key-value pair, and outputs another key-value pair. The results are consolidated via the partitioner and shuffler, and finally passed on to the Reducers. Reducers also take in a key-value pair, to output another key-value pair.

If you didn't understand that flow I created, check out this more visual shape-counter example.

Hadoop meets a few of our requirements, but there is one critical problem that I discovered, it had a single point of failure at the NameNode. If the NameNode went down, it wouldn't recover smoothly, human intervention will be required to restart the server or even modify code if the damages are too much. So what can be done?

Interestingly, there is another project by Apache called ZooKeeper. Which is a coordination service for distributed applications that helps with synchronization and configuration maintenance. This seems to be helpful in keeping our Hadoop system up even when the NameNode goes down.

Thanks to Apache, I did not have to handle this integration, as there was another project called HBase, that solves this problem using ZooKeeper.

HBase is a column-oriented database that is built on top of HDFS. It provides real-time real/write access, and more importantly, because of its reliance on ZooKeeper, it does not have a single point of failure. Lets talk about column-oriented database for a moment. Compared to legacy Relational Databases, data is now stored row by row, rather, an entire column is stored together physically. One main advantage of column-oriented databases is that they tend to handle read requests faster than relational databases due to the nature of most queries. For example, in a usual relational database, employees details would probably be a single row for one employee, with many many columns consisting of the details of the employee. If you wish to find the job title of a person, you would retrieve the entire row, then filter to get the job title. Sounds like a lot of needless reads for data you don't need!

In a column-oriented database, the same query requires only reading the job title column, and the employee id column.

This is what a HBase table looks like. It is made up of rows, each single row is identified by its row key. It has multiple column families, which are groups of columns. So 1 column family is made up of multiple columns, and in a single table you can have many column families. The way to identify a column family is by the colon. A cell is uniquely identified by a { row, column, version } tuple. The table shows multiple version of the address of john.

So to see this in action we will look at a demonstration of using HBase. What I am doing here is that I have some test data stored into HBase, and this java application I am running now is coded by me. It is basically a layer built on top of the basic table operations that HBase provides, namely GET SCAN PUT DELETE.

This application is not the only outcome of my internship. Another thing that I've done is a set-up guide for Hadoop, HBase and ZooKeeper that is extremely comprehensive. This includes hyperlinks for novice users to learn more about UNIX commands, tips to follow to ensure a smooth installation, warnings that aims to help new users steer clear of the mistakes that I have made, and a treasure trove of links for users who require more details. There are a few documents I have done as preliminary research on databases, and also heavy documentation on the Hadoop, HBase and ZooKeeper systems itself, which I heavily referenced from books by O'Reilly Media. In developing for Java, I have also taken a step into UML, and here are the Class Diagrams and System Sequence Diagrams I have created to help users understand my software.

To sum up this presentation, this is how the entire Hadoop ecosystem looks like. You have the underlying file system: HDFS, HBase as a column-oriented database implemented on HDFS, MapReduce as a programming framework for data analysis, and ZooKeeper as the coordination system.

My work doesn't stop here, there is a lot more knowledge to absorb, a lot more mistakes to make and definitely a lot more ways we can adapt the ecosystem to suit the organization's needs. For example Pig and Hive can be integrated into the system to increase useability. Pig is a platform for analysing large data-sets using a high-level language. Essentially it wraps MapReduce into easy to use keywords. Hive is like an SQL for Hadoop, its strong points being data-summarization and adhoc queries.

Another project worth exploring is Lily, which is based on HBase and Solr, providing storage, index and searching capabilities.