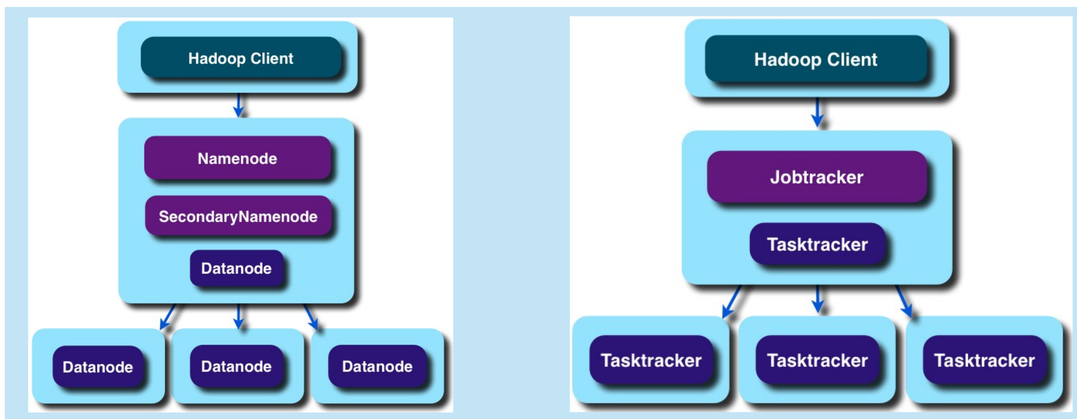
dCache billing data analysis With Hadoop.

Andreas Knoepke (HTW Berlin, Berlin, Germany)
Kai Leffhalm (DESY, Zeuthen, Germany, contact: kai.leffhalm@desy.de)

Hadoop



Hadoop – Framework for data intensive distributed computing

Apache Hadoop is developed as open-source to provide data intensive applications with a reliable and scalable file system (HDFS) and an computational paradigm: MapReduce

HDFS

With one name node and one to many data nodes it can handle terabyte sized files, but reduces it to a WORM model. Data is stored in blocks over all data nodes, number of replicas of each block is configurable per file.

The name space is hierarchical organized and located on the name node, but no user data is streamed through the name node
.
Data distribution is done by the name node. It receives a heartbeat from the data nodes to identify missing nodes and to restore the number of replicas in case of failure.

MapReduce

A framework written in java for processing parallel analysis on big amount of data.

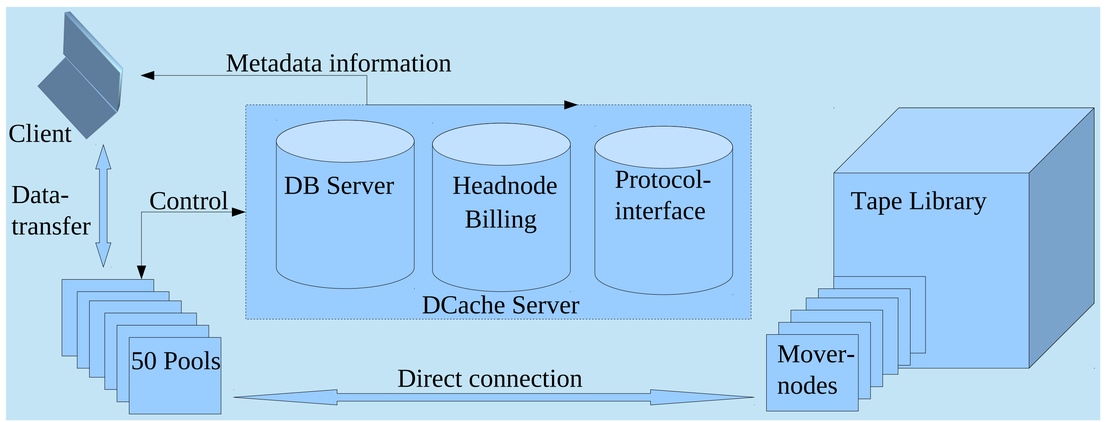
It is a two step algorithm starting with a **Map** procedure sorting and listing data on every single data node and a **Reduce** procedure performing a summary operation on the mapping results.

Several MapReduce algorithms can be called in succession to split up problems in smaller portions before processing the data.

An algorithm composed of a sequence of MapReduce processes has to be created for each „query“ to the data according to a sql command in a relational database or a grep command piped through several bash commands used on files.

The MapReduce processes can be derived from the master classes, each having a setup and clean function that can be overwritten.

DCache – Setup at DESY, Zeuthen



dCache – large capacity file system

dCache is a distributed file system for storing and retrieving data on disk and tape.

It can handle petabytes of data distributed over many different machines containing different hardware and enabling hierarchical storage management.

The name space is realized as a PostgreSQL DB. It supports many different protocols: WebDAV, xrootd, NFS4.1, direct I/O with its own protocol dcap.

HSM - Integration

The access to tape libraries is possible and used at many sites for storing large amount of data for later access and long term archival. dCache is storing the place returned by the HSM system and can trigger restore and pre-staging if a file is not on disk anymore.
.

dCache – Setup in DESY, Zeuthen

There are three instances:
> Atlas (mid-size Tier2)
> LHCb
> Astro-Particle physics experiments (data needed by users located at our site) which is linked to an HSM (via OSM).

Altogether the three dCache instances provide 2.5 PB spread over 110 machines and are filled by more than 75%.
The attached Tape Library has currently a capacity of 2.5 PB (shared with normal backup).

Standard storage servers are used, connected via 2 x 1GE or 10GE, older machines having a JBOD attached.

There are more than 900,000 entries per day in the log files, resulting in 250 MB of logging files per day created by 310,000 files read and 95,000 written in average.

Billing information – Motivation for using Hadoop for storing the logging information

Billing example

```
10.09 00:00:02 [pool:ice-rw-obst82-1@obst82Domain:transfer] [0000A6AC26D2037842629A920AEC0E2591AB,105447073] [Unknown]
neutrSouth:icecube13@osm 105447073 1203 false {DCap-3.0,bladef5.ifh.de:38292}
 [door:DCap-puck-<unknown>-784904@dcap-puckDomain:1381269601031-1268665] {0:""}

10.09 00:00:01 [door:DCap-puck-<unknown>-784904@dcap-puckDomain:request] ["":-1:-1:unknown]
[0000A6AC26D2037842629A920AEC0E2591AB,0] [/net/pnfs/acs/<path to file>] <Unknown> 1225 0 {0:""}

10.09 00:54:29 [pool:ice-rw-obst82-1@obst82Domain:store] [0000A6AC26D2037842629A920AEC0E2591AB,105447073]
[Unknown] neutrSouth:icecube13@osm 211265 268701 {0:""}
```

Billing data in dCache

Actions logged by dCache as billing information:

> Writing a file
> Reading a file
> Deleting a file
> Storing to HSM
> Restoring from HSM
> Requesting a file not available
> Error messages from storage pools

Billing data in files

> Actions are logged at the moment the action is completed.
> Every action has two procedures (other numbers are possible) stating the request and the actual transfer, eventually an error.
> Entries are ordered by end time but recorded is the start time.
> Every day starts with a new billing file giving the possibility to compress and archive files.

Billing data in DB

Using PostgreSQL each action has one entry updated when the action is completed. Faster but DB might reach big size (> 40GB in half a year at our site)

Expected advantage using Hadoop

Processing capabilities of billing files is important

> for error detection and analysis
> collecting accounting information
> anlysing data usage
> … by user
> … by group/experiment
> … by data organisation (eg per directory)

Up to now there are two possibilities:
> sql queries to DB
> searching through files

Disadvantage of DB is data cannot easily be phased-out, they grow fast and splitting and partitioning is difficult to maintain.

Disadvantage of files is they cannot be searched easily with floating time intervalls, they may take quite long time for long intervalls.

Both have problems with scaling as data is increasing.
Hadoop should be better scalable as a DB (promised to be linear with machin numbers) and give faster access to data than file approach.

Setup and implementation

Constraints

Several features had to be controlled and tuned during setup, implementation and testing:

> Correctness of data
> Performance of requests
> Scalability

Test requests

Focus on two main Information requests from billing information:

> All transfers of one pool in a given time intervall
  > differentiated by read, write
> All files not deleted but never read
  > Interesting in discussions with experiments about needed space
> Atlas instance billing files are used as it has the most entries and no store/restore actions (no HSM link)

System

> Data nodes: 4 real and one virtual machine
> heterogeneous environment
  > different file systems
  > different CPU
> Name node: installed on one of the real machines

Data

Data has to be transferred from billing files to Hadoop

> Search for lines concerning one transfer and combine them.
> Store the files either per day or per month in the HDFS (when doing migration).
> Data has to be added continuously in daily operation (next step).

MapReduce strategies

Applying a MapReduce algorithm starts with tuning of the stored data. Every query to the data has to be converted to an appropriate key – value pair during the Map process.

Number of transfers and data-rate of one pool

On every data node the Mapper creates key – value pairs out of the data:
> Key is the kind of transfer: read or write
> Value is text with duration of transfer, size of requested data and number of requests

<write> <t s c>
<read> <t s c>

One Reducer for every key is created and accumulating the data given by the Mappers:

<write> <<t1 s1 c1>...<tn sn cn>

This is parsed and accumulated to the final result

Number of files never accessed

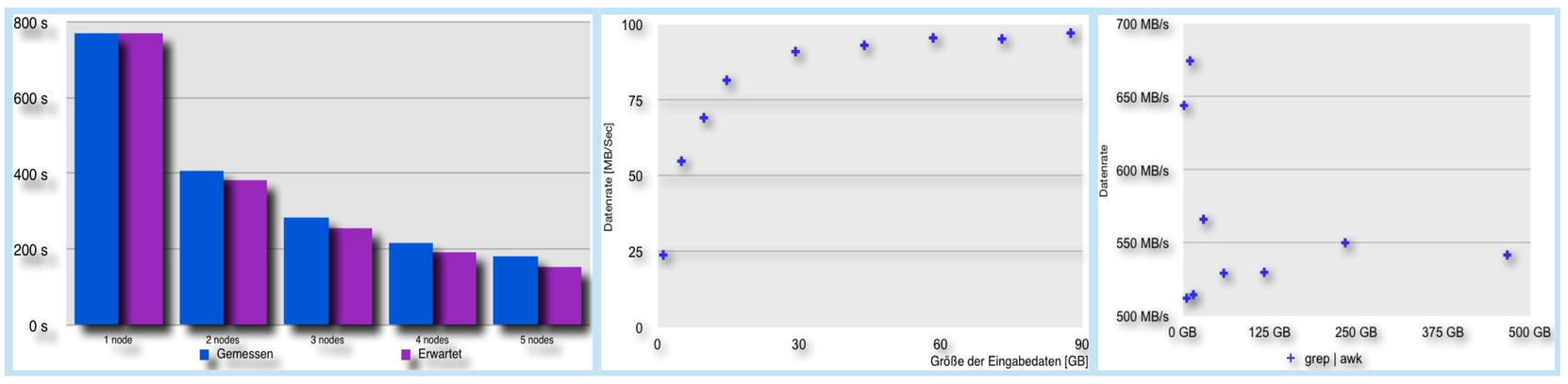
For this query the key is the id of the file. The Reducer might get the following input for a file read three times and deleted:

<id> <<r> <r> <r> <w> <d>>

Software

> Java classes derived from the generic classes Mapper and Reducer have been developed for the two use cases.
> jar files can be added to the name node and will be distributed on runtime by the framework to the data nodes.

Results



General results

During installation and testing several points attracted our attention:

> Open standard Hadoop tcp ports to accelerate intercommunication by factor of 12.
> Optimized max number of Map and Reduce tasks running parallel:
  > 6 on the real machines with quad core cpu
  > 3 on the virtual machine
> Real machine performance was varying by 35% between the nodes (slowest slower than fastest).
> The underlying file systems used at the site (ext3, xfs) were compared and differences of 2.5% found (xfs is faster than ext3).
  > As ext3 is in addition very slow for setting up big file systems xfs is recommended and used.

Performance measurements analysis rate by number of data nodes

Theoretical expectation was that increasing the cluster from 4 to 5 machines should decrease time by 20%.
The first plot shows (blue) the measured and (purple) the expected rate for differnet number of data nodes.

Scalability is quite good, keeping in mind that machines are not equally fast, still every additional machine is increasing the performance.

Performance measurements analysis rate by billing size

Time needed for analysis by different sized billing files (second plot)

> As expected scalability is good
> Bad performance for small amount of billing data
  > Due to framework overhead
  > Effect is decreasing fast with increasing size of billing data

Performance measurements grep/awk as alternative

Grep/awk are the standard way to query text file billing information. On the third plot some data rate measurements have been done.
For the small amount of data used they can compete and outrun the small Hadoop setup tested, still it is obvious that with increasing amount of data analysis is getting slower.

Conclusion drawn

It has to be considered if only new data is interesting or if large amount of old data should be taken into account too before switching to Hadoop as billing system.

Important for the decision is also how many data nodes are available as it seems with high volume of data the data rate to analyze the billing information is highly dependent only on the number of nodes.

Future plans

dCache pool integration test

Hadoop is at its most impressive when distributing data over many data nodes like dCache. It seems natural to combine both on the same machines using different partitions for dCache data and Hadoop file system.

We will install Hadoop in the existing dCache test system to measure the impact of the MapReduce algorithm on the pool data rate if any is measurable.

Integration with other frameworks

Many frameworks and projects have evolved around Hadoop. Some can maybe ease installation, integration or configuration.

These have to checked on their usability to our use case
> Cassandra
> Pig

New version of MapReduce (MapReduce2, YARN) has some features (limit of number of files in HDFS).

dCache logging integration

> First tune system to have data added automatically on regular base.
> Realtime adding is aimed for later on.
> The optimal solution for admins would be to have a direct configuration parameter for dCache logging to be written in a Hadoop cluster. This integration needs much more testing with different setup szenarios like having not many single pools but big storage hardware split up between different applications.

Links and Literature:

http://hadoop.apache.org/
http://dcache.org
Bachelor work of Andreas Knoepke:
„Vergleich der Billing-Datenbank im dCache
mit parallelisierter Abfrage in Hadoop“

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences