Prototype of xrootd monitor with hadoop backend

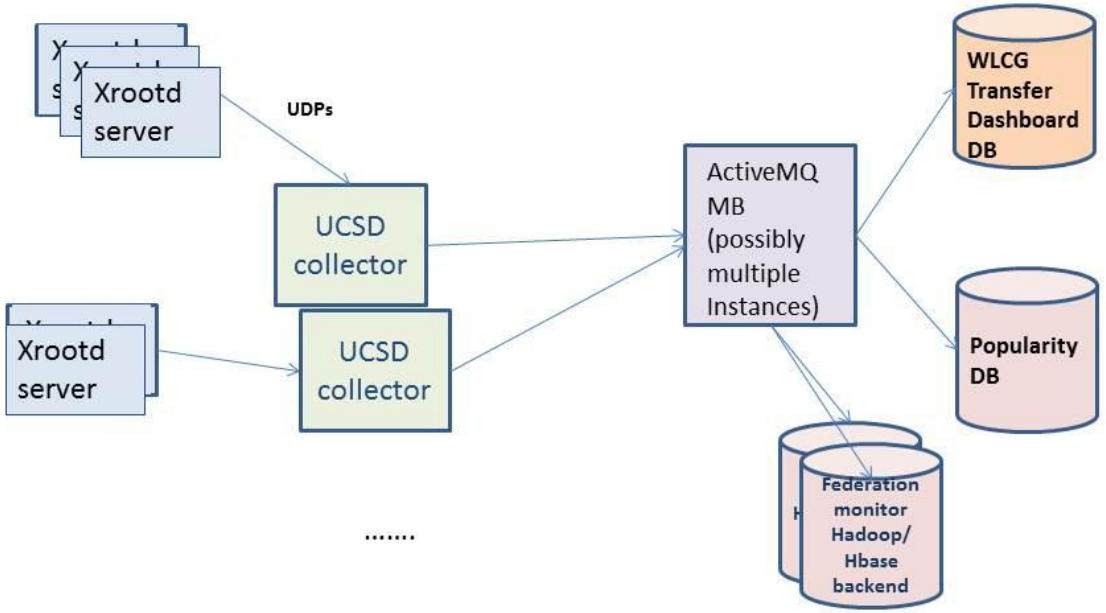
Sergey Mitsyn 2012-10-23

Plan

- 1) xrootd central monitor with Oracle backend intro
- 2) Hadoop and Hbase microintroduction
- 3) Experience and expectations
- 4) Current state of hadoop deployment

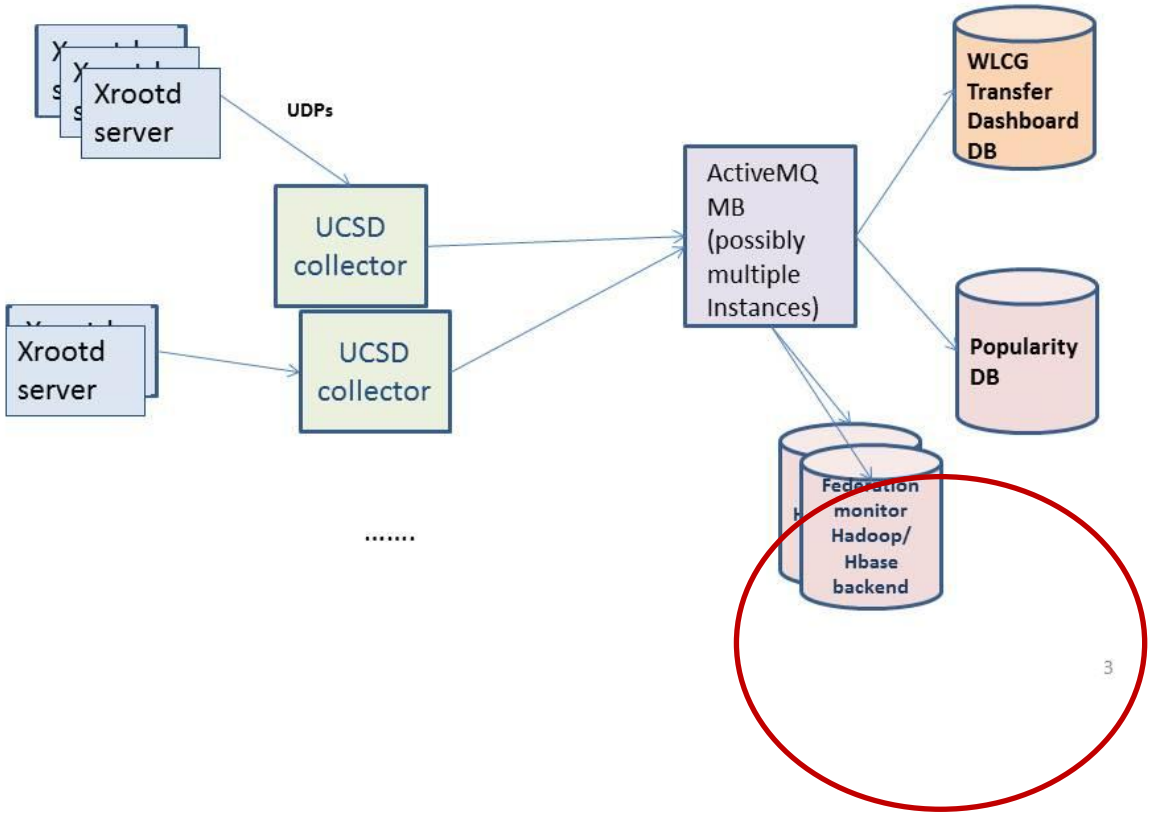
xrootd monitoring/central storage

Data flow for the xrootd monitoring applications

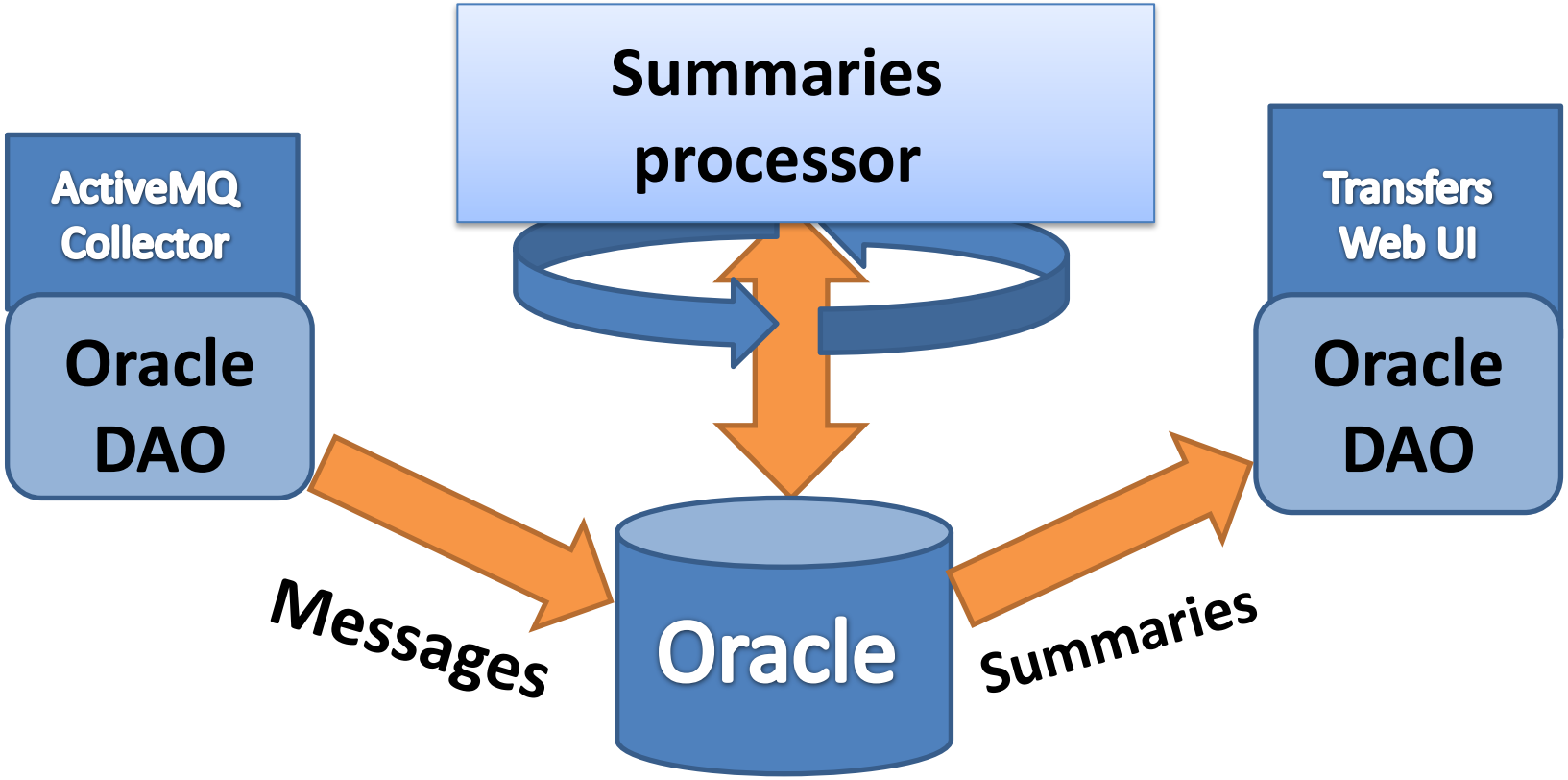


xrootd monitoring/central storage

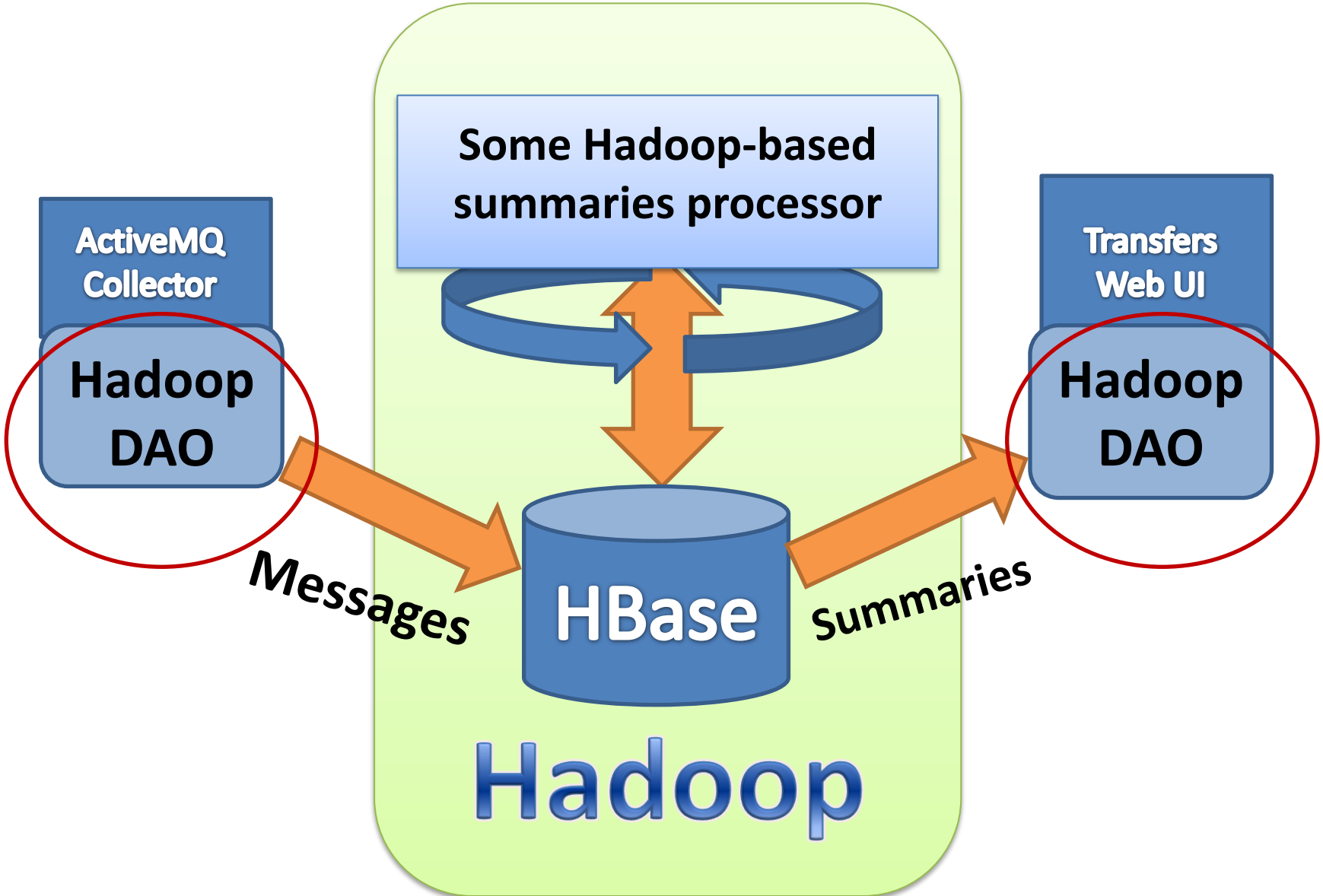
Data flow for the xrootd monitoring applications



xrootd monitoring/central storage



xrootd monitoring/central storage



Hadoop, a microintroduction

- Is an implementation of Google's Map/Reduce framework, a framework for distributed computing.
- Consists of multiple services, including:
  - HDFS – a distributed high-latency file system
  - HBase – a distributed low-latency columns-based database (based on HDFS)
    - Data storage.
  - mapreduce – a framework for Map/Reduce jobs (processing)
    - Data processing backend.

Hbase 1

- Column-based:
  - **Rowkey** – **Table** – **Column** denotes a single value.
    - Slightly more features than key-value based store.
  - All values are uninterpreted bytes;
  - Rowkey: global index, fast lookup;
  - Columns: independent values for single row.
- Operations:
  - Put/get/delete:
    - Single row, multiple columns, *atomic*
  - Batch put/delete:
    - Multiple rows, *non-atomic* across multiple rows
- Naïve usage:
  - Message ~ Row;
  - Message field ~ Column.

HBase 2

- Fully consistent (as in the CAP-theorem, single copy) (more strict than eventual consistency).
  - Though eventual consistency would be fine too:
    - The messages are never deleted, only inserted.
    - Summaries are updated (overwrited) without dependency on old values.
- No joins, no transactions, no global locks.
- No need of transactions:
  - Denormalized data -> no secondary keys:
    - Available atomic operations transfer database from one consistent state to another (in terms of invariant).

Messages table format

Row Key **A concatenation of "end_time" and sha1 hash of json string**	**Single column family 'cf'**			
	Fields/columns *cf:end_time*	*cf:server_domain*	…	*cf:file_size*
1352000001:2e79ac6823b7d64	1352000001	www.com	…	3133700
1352000001:a278399fb980a3cb	1352000001	www.com	…	7654321
1352000002:…	1352000002	cern.ch	…	…

A concatenation of end_time and SHA1 hash of message enables us to:

1) Have unique row key for each message.
2) Have them sorted in "end_time" order for faster processing.

Processing

- Deduce source and destination from *server* and *client* addresses, bytes read and written.
- Group by fields
  - Source, destination, traffic type, etc
- Sum over transferred bytes.
- Write summaries to summaries table.

Processing 2

- Alternatives:
  - Map/Reduce:
    - code in Java ☹
  - External processing: load data to python, process, put summaries back.
    - Does not scale well (but with current load it doesn't matter, also may utilize more sophisticated schemes).
  - Hadoop Hive, Pig:
    - Hadoop services that translate their own code (HiveQL or Pig Latin) to Map/Reduce classes to run locally or on cluster.
    - No to code in Java, yes to code in "strange" languages.
    - Provide ***Join***s! But we don't need them anyway…

Processing 3

- Hive:
  - HiveQL: a language that strongly resembles SQL.
    - Tables, views, SELECT … FROM … GROUP BY …, etc.
    - Supports User-Defined Functions (UDF) written in Python.
- Pig:
  - Pig Latin: a language that resembles SQL a little.
    - More procedural than HiveQL; relations.

Message storage formats

- Column-based:
  - One field – one value:
  - Fits the ActiveMQ message format.
  - Easy to work with Hive, Pig.
  - Big storage waste on metadata.
- “Flat” format:
  - Concatenate all values and put to single column.
  - Tightest storage utilization;
  - Still possible to work with Pig/Hive;
  - Dangerous in case of schema change.
- Json in single column:
  - Messages from collector are also in JSON format.
  - Preserves type info (no need to convert numbers to strings and back).
  - Impossible to work with Pig/Hive (or yes to code in Java).

Hbase external interfaces

2 protocols:

- REST:
  - pycurl/urllib is enough.
  - Cumbersome to use.
  - Buggy.
- Thrift:
  - Has external dependencies (*thrift-lib-python* and hbase bindings rpm, available in dashboard-externals repo for **current** HBase version).
  - Easy to use (happybase in SVN, adapted for python 2.4).
  - Binary (should be faster?).

Implemented

- DAO:
  - For collector: 100% (including 3 message formats)
  - For web UI: partly (transfers matrix and bins are ready)
- Processing:
  - In Python (external processing):
    - ready but not in SVN (will commit soon)
  - Pig, Hive:
    - In very early development stage; currently produce exactly the same results as python summaries processor.
    - To implement as script templates or use packages (PyPig)?
  - Results match these from Oracle backend.
    - … more or less – some jobs are lost due to collector restarts.

More problems

- 1. Increasing keys for transfers are bad:
  - All insertion load is going to be on single node, and possibly summary queries also.
    - But we need timestamp-like keys for queries on time interval.

  http://ikaisays.com/2011/01/25/app-engine-datastore-tip-monotonically-increasing-values-are-bad/
  http://hbase.apache.org/book/rowkey.design.html
- 2. We need ***>= 100 000 000*** of entries for any improvements over RDBMS!
  - Currently no more than 100 000/day

  http://hbase.apache.org/book/architecture.html#arch.overview
- 3. Optimizations ToDo: bloom filter, block caches, local data access…

  http://hbase.apache.org/book/perf.hdfs.configs.html

Using HBase

- To begin learning, one should try HBase shell:
  - 1) Login to any host with HBase available (or install yourself)
    - See next slides…
  - 2) # hbase shell
  - 3) perform all of http://hbase.apache.org/book/quickstart.html p.1.2.3 Shell Exercises.
    - Supports put, get, delete, create/drop table, etc…
    - Don’t drop any of existing tables, please!

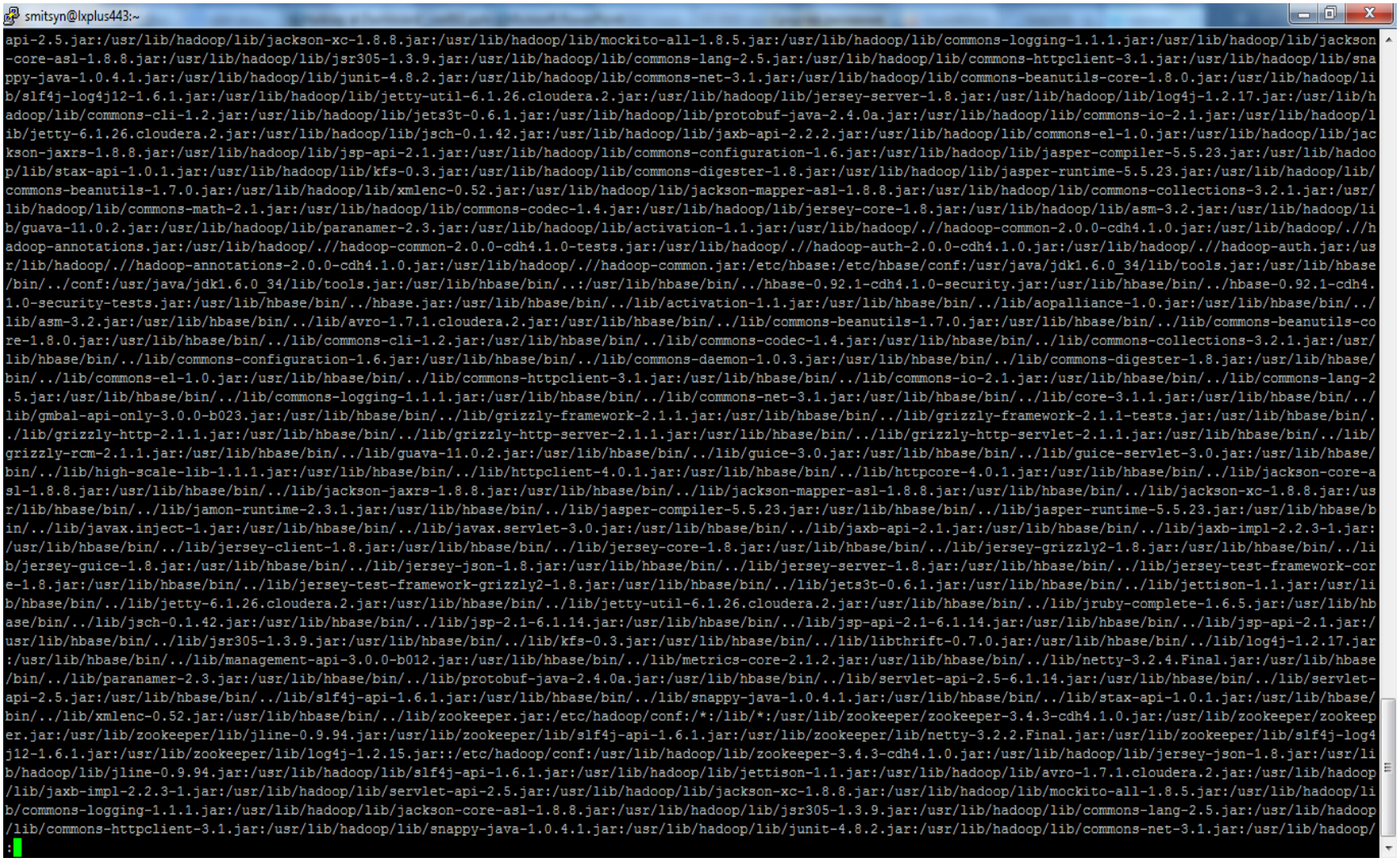
Local Hadoop installation

- Local installations for development:
  - Local or pseudo-distributed: every service on the same node.
  - Easy installation as in the Cloudera QuickStart guide:
    - https://ccp.cloudera.com/display/CDH4DOC/CDH4+Quick+Start+Guide
  - Nice to install for development.
  - Not suitable for application performance measurements (very low!).

The available Hadoop installations:

- CERN's at **lxfssm4401**:
  - *HDFS*: Namenode at lxfssm4401;
  - *Hbase*: master at the same host; *thrift*;
  - *mapred*/*YARN*: look tomorrow!
  - *Gateway* at **dashboard48**.
    - (just a node with software and configuration, but no data or mapred tasks)
- Dashboard's at **dashboard07**:
  - 4 hosts: 07, 08, 64, 65.
  - (currently) limited reliability:
    - Crash of dashboard07 stops everything (no SecondaryNameNode)
    - Crash of any 2 of 3 other machines at the same time is tolerable.
  - *mapred*, *HDFS*, *Pig*, Hive, *HBase* with *thrift* and *REST* interfaces;
  - *Unstable* ☹
  - Future: development only? Deprecate & remove?

Configuring Hadoop: fail



Hadoop Gateway

- Is simply a node without any data or mapred services running,
  - … but may run any data access services like REST or Thrift;
  - may run Java applications for Hadoop.
- All you need are:
  - client packages (e.g. yum install hbase-rest)
  - Copy-paste HBase config (I hope someday you may find it in Dashboard wiki)
- Why you may need that for:
  - An HBase shell without ssh access to the cluster;
  - To run arbitrary service.

The end

- Questions?