BigData
An Overview of Several Approaches

David Mera

Masaryk University
Brno, Czech Republic

16/12/2013

Table of Contents

Table of Contents

Introduction

- There are huge datasets of heterogeneous data available which are growing fast
- Most of world's data were created in the last 2 years (IBM source)
  - 2.5 exabytes are created every day
  - Wallmart collects 2.5 petabytes of data each hour
  - 340 millions of tweets are sent every day

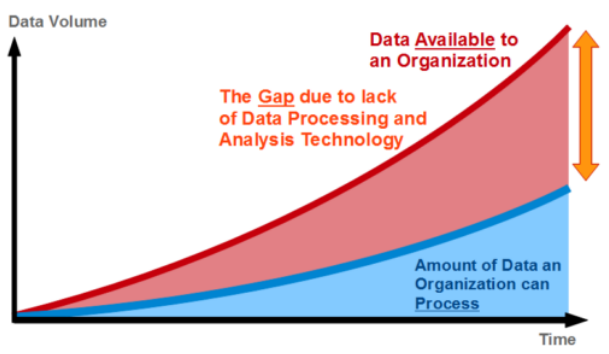


Table of Contents

Terminology

Batch data

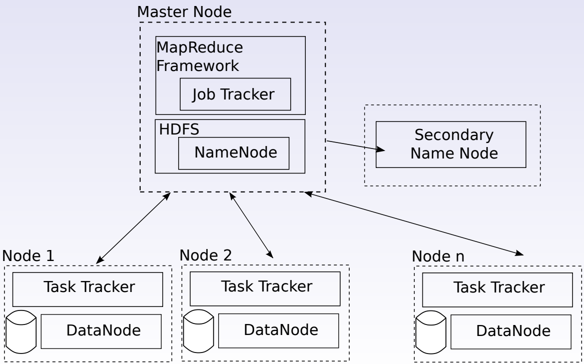
- Static snapshot of a data set
- Batch computation has a ‘start’ and an ‘end’
- Fast datasets processing

Stream data

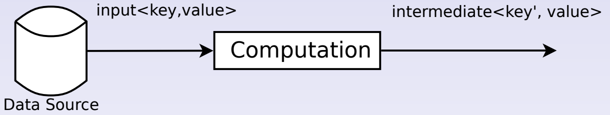
- Stream of events that flows into the system at a given data rate over which we have no control
- Stream computation ‘never’ ends
- The processing system must keep up with the event rate or degrade gracefully
- Near-real time answers

Table of Contents

- MapReduce is a framework for paralleling processing of massive data sets.
- Hadoop implementation is highly optimized for batch processing
- Hadoop attempts to run Map and Reduce tasks at the machines were the data being processed is located



- MapReduce Job
  - Map function (mandatory)



  - Reduce function (optional)

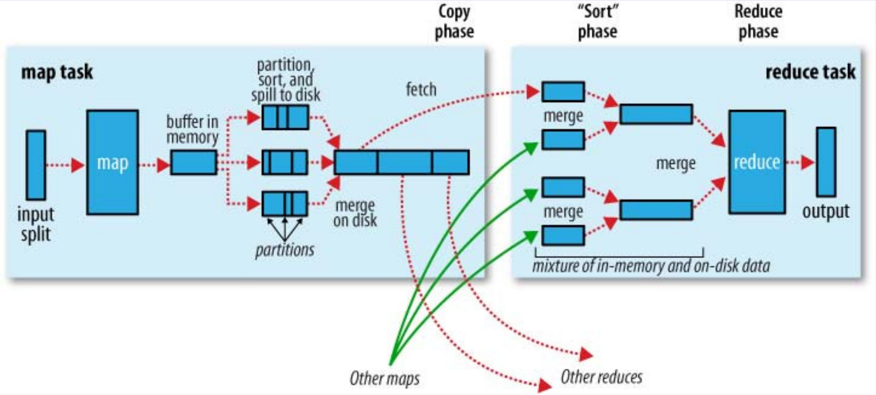
- Characteristics that the developer gets without the need to write any code
  - Machine communication
  - Task scheduling
  - Scalability
  - Ensuring availability
  - Handling failures
  - Automatic partition of the input data

- Data placement
  - Data are split in storage blocks
  - First replica is located in the same node as the client
  - Second replica is placed on a different rack chosen at random
  - Third replica is placed on the same rack than the second but in different node
  - 'Balancer' daemon
- Input Reader
  - Input data can be retrieved from several datasources (file system, database, main memory)
  - Data are split in FileSplits
    - The unit of data processed by a map task
    - Storage blocks (by default)

- Map function
  - Mandatory function
  - A new map task is created per FileSplit (block)
  - The user can not manage the number of mappers
  - Each FileSplit is divided into records and the map processes each record <key,value> in turn
  - Map function outputs the result as a new <key,value> pair.

- Combiner function
  - It does partial merging of data before sending them over the network
  - It is executed on each machine that performs a map task
  - Same code than the reducer function

- Shuffle and Sort phase



- Reduce function
  - To merge map outputs
  - The number of reducers can be managed by the user
  - The Reduce function is invoked once for each distinct intermediate key
  - Pairs with the same key will be processed as one group
  - The input to each reduce task is guaranteed to be processed in increasing key order
- Output writer
  - It is responsible for writing the output to stable storage
  - Data storage could be modified

- Large files optimization
  - How to deal with images?
  - HIPI
- Data format management
  - Optimized for text inputs
  - HIPI
- Selective access to data
  - Hadoop++ provides indexing functionality
    - Non intrusive
    - Indexes are created at data load time and thus have no penalty at query time
    - We must know the schema and MapReduce jobs
- High communication cost
  - CoHadoop
    - Related data are stored in the same node
    - HDFS is extended with file-level property

- Redundant processing
  - Restore
    - Workflows of MapReduce jobs
    - To manage the storage of intermediate results
    - To reuse intermediate results
- Early termination and quick retrieval of approximate results
  - Reduce functions cannot start before all map functions are finished
  - 'MapReduce online'
- Lack of iteration
  - Iterative data analysis cannot be processed efficiently by the framework
  - MapReduce sequences are complicated to write.
  - A performance penalty is paid in every iteration (data reload and data reprocessing)
  - 'MapReduce online'

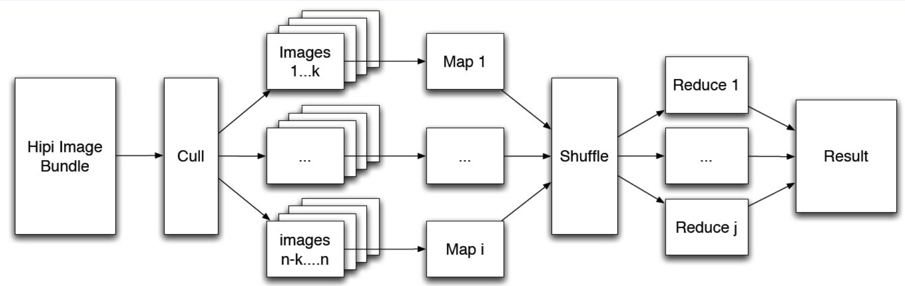
- Load Balancing
  - The runtime of the slowest machine will easily dominate the total runtime.
  - Plain partitioning schemes that are not data-aware don't get good results
  - Even when the data is equally split to the available machines, equal runtime may not always be guaranteed
- Real-time processing
  - MapReduce runs on a static snapshot of a data set
  - The input data set cannot change.
  - No reducer's input is ready to run until all mappers have finished
  - A MapReduce computation has a 'start' and an 'end'
  - 'MapReduce online'

HIPI

A Hadoop Image Processing Interface for Image-based MapReduce Tasks

- Specific framework to deal with image processing and computer vision applications
- HIPI goals
  - Providing an open, extendible library for image processing
  - Storing images efficiently
  - Filtering images
  - Hiding Map-Reduce details
  - Optimizing applications to be executed in MapReduce

- HIPI Image Bundle Data Type stores many images in one large file
- HIPI has a filter based on image properties
- HIPI processes individually each image
- Images are stored as standard data types. The HIPI library encodes and decodes images



- Main goals
  - Online aggregation (Incremental outputs)
  - Continuous queries (streaming processing)
- Large modification of Hadoop
- Data are pipelined between operators
  - Reducers begin processing data as soon as they are produced by mappers
  - Increasing opportunities for parallelism
  - Resource utilization improvement
  - Response time reduction

- Map tasks were modified to push data to reducers
  - Map buffer
    - Fixed threshold
    - Combiners are applied over buffer data
    - Buffer data are sorted
    - Data are written into the disk
    - Files are registered in the TaskTracker
    - TaskTracker sends files ASAP to the reducer

- Online aggregation
  - Reduce function is applied over the pipelined map outputs
  - Snapshots are stored in HDFS
  - Snapshots can be used as inputs for the next task
- Iteration
  - Reducers can pipeline their output to the next map operator
    - To avoid HDFS storage
  - JobTracker was modified to accept a list of jobs

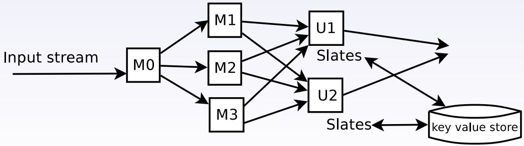
- Continuous queries
  - Mappers and reducers are fixed
  - Reducers are configured to be executed periodically
  - Map outputs are maintained in a buffer with unique id
  - Reducer informs to the jobTraker when its task is finished
  - Jobtracker informs mappers that data are no longer necessary

- Main goal
  - To treat a streaming computation as a series of deterministic batch computations on small time intervals
- Data are received and stored in intervals
- Model advantages
  - It is easy to unify with batch systems
  - Users only need to write one version of their analytic task
  - Fault tolerant. Similar recovery mechanisms to batch systems
  - Consistency is well-defined since each record is processed atomically with the interval in which it arrives

Table of Contents

- Framework specifically developed for fast data
- Components
  - Event<stream_id, timestamp, key, value>
  - Stream is a sequence of events with the same ‘stream_id’ and increasing order of timestamp
  - Map function: map(event)=event*
    - Memoryless
  - Update function: update (event,slate)=event*
  - Slate
    - A slated is determined by the tuple <update U,key k>
    - $SLATE_{uk}$ is an in-memory data structure which summarizes all events with key ‘K’ that an update function ‘U’ has seen so far
    - Time-to-live parameter

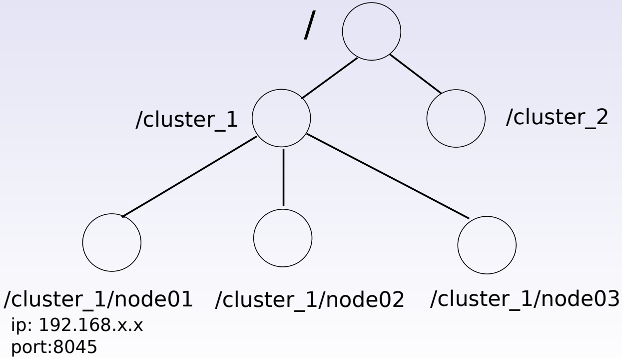
- The work flow is modeled as a direct graph
- Muppet starts up a set of workers on each machine
  - A hash function is used to distribute events
  - A special mapper is used to read from the input stream
- Slates
  - All events with the same key will go to the same update
  - Key-value storage - Cassandra
    - Slates may outgrow the memory
    - Persistent slates help recovering the application from crashes
    - Slates could be queried long after the termination of the application



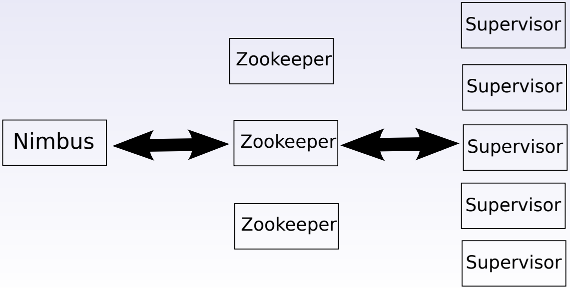
- A worker ‘A’ determines the worker ‘B’ to which to send an event by hashing the key and destination updater function of the event
- If ‘A’ cannot contact ‘B’, then it assumes the machine has failed, and ‘A’ contacts the master to report
- The master broadcasts the machine failure to all workers
- Hash function is updated
- If updater fails then temporary slate data are lost.

ZooKeeper
Wait-free coordination for Internet-scale systems

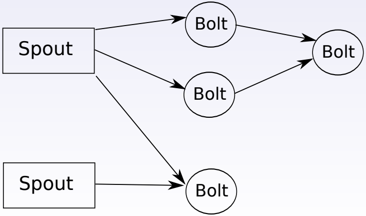
- Centralized service to coordinate distributed processes
- Shared hierarchical name space of data registers (znodes)
- Data are kept in-memory
- Znodes are limited to the amount of data that they can have
- The service is replicated over a set of machines



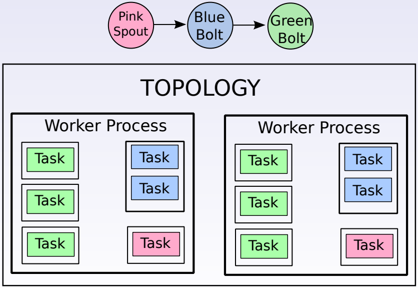
- Storm cluster
  - Master node
    - The Nimbus daemon is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures
  - Worker nodes
    - The Supervisor daemon listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.
  - Communication - Zookeeper

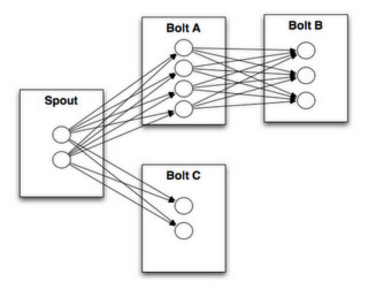


- Storm runs topologies
  - Graph of computation
  - Each node in a topology contains processing logic
- Stream
  - Unbounded sequence of tuples
- Spout
  - It reads input data from an external source and emits them as a stream
  - It is capable of replaying a tuple
- Bolt
  - Input streams –> some processing –> new streams.



- Topologies execute across worker processes (JVM)
- Tasks are spread evenly across all the workers
- The parallelism for each node is defined by the user
- User can also specify tasks for each node
- Stream grouping - How a stream should be partitioned
  - i.e.Shuffle grouping
- Scalability in processing time

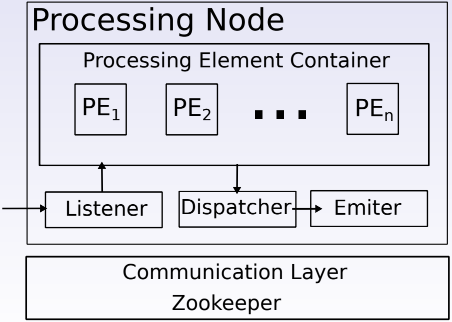




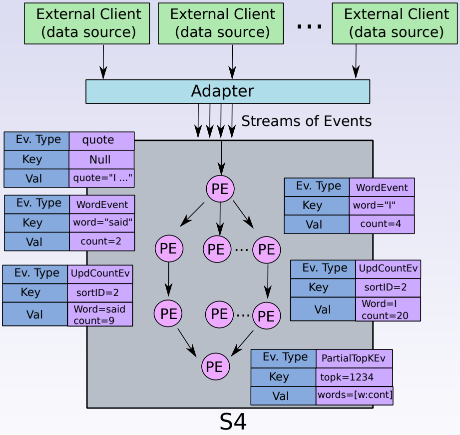
- If a worker process dies then the supervisor will restart it
- If a node dies then Nimbus will reassign those tasks to other machine
- If a daemon dies (Nimbus or Supervisor) then they restart
  - State of Nimbus and workers is saved on Zookeeper
- Storm guarantees that each message will be fully processed.
  - A tuple is considered "fully processed" when the tuple tree has been completely processed.
  - User must specify links in the tree of tuples
  - User must specify when an individual tuple is done

- S4 goals
  - Simple programming interface for processing data streams
  - Language neutrality
  - Commodity HW
  - High availability and Scalability
  - Decentralized architecture
  - To avoid disk access
- S4 assumptions
  - Lossy failover is acceptable
  - Nodes cannot be added or removed from a running process

- Stream: sequence of events <Key,Value>
- Processing Elements (PEs) are the basic computational units
- Processing Nodes are the logical hosts to PEs
- S4 routes each event to PNs based on a hash function
- Communication layer: Zookeeper



- Example: “I meant what I said and I said what I meant.”



- The processing of an event is not guaranteed
- The network is used heavily
- User must consider carefully how to split the data (keys) in terms of performance

Table of Contents

Conclusions

- Typically, systems are developed to solve an specific problem
- Lack of heterogeneous systems

“Attempting to build a general-purpose platform for both batch and stream computing would result in a highly complex system that may end up not being optimal for either task”