An introduction to Big Data

Presentation by Devesh Sharma, Zubair Asghar & Andreas Aalsaunet
About the presenters

Andreas Oven Aalsaunet

- Bachelor’s Degree in Computer Engineering from HiOA (2011 - 2014)
- Software Developer Associate at Accenture (2014 - 2015)
- Master’s Degree in Programming and Networks at UiO (2015 - 2017)
- Master’s Thesis with Simula Research Laboratory: “Big Data Analytics for Photovoltaic Power Generation”
- Interests: Big Data, Machine Learning, Computer games, Guitars and adding as many pictures to presentations as I can fit in.
About the presenters

Devesh Sharma

- Bachelor’s Degree in Computer Science Engineering.
- Master’s Degree in Programming and Networks at UiO
- Master’s Thesis with Simula Research Laboratory: “Big Data with high performance computing”
About the presenters

Zubair Asghar

- Bachelor’s In Information Technology (2002 - 2006)
- Master’s Degree in Programming and Networks at UiO
- Worked with Ericsson for 6 years
- Currently working as a developer with DHIS2.
- Master’s Thesis with DHIS2 Disease Surveillance Program
- Passion For Technology
About the presenters

Zubair Asghar

- Bachelor’s Degree in IT.
- Worked in Ericsson for 6 years
- Master’s Degree in Programming and Networks at UiO (2015 - 2017)
- Master’s Thesis with DHIS2: “Disease Surveillance Program”
- Passion for technology
This presentation is based on the following articles:


What is Big Data, and where did it come from?

Some definitions:

- Wikipedia: “data sets so large or complex that traditional data processing applications are inadequate”
- McKinsey: “datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze”
- Oxford English Dictionary: “data of a very large size, typically to the extent that its manipulation and management present significant logistical challenges.”
What is Big Data, and where did it come from?

Arose due to the confluence of three major trends:

1. Cheaper to generate lots of data due to increases in:
   a. Inexpensive storage (e.g. increase in storage capacity in commodity computers)
   b. Smart Devices (e.g. Smart Phones)
   c. Social software (e.g. Facebook, Twitter, Multiplayer games)
   d. Embedded software, sensors and Internet of Things (IoT)
What is Big Data, and where did it come from?

Arose due to the confluence of three major trends:

1. Check!

2. Cheaper to process large amounts of data due to advances in:
   a. Multi Core CPUs
   b. Solid state storage (e.g SSD)
   c. Inexpensive cloud computing (e.g SaaS, PaaS and IaaS)
   d. Open Source Software

3. Democratization of data:
   a. “The process of generating, processing, and consuming data is no longer just for database professionals”, [1].
   e.g. more people have access to data through dashboards etc
What is Big Data, and where did it come from?

“Due to these trends, an unprecedented volume of data needs to be captured, stored, queried, processed, and turned into [actionable] knowledge” [1]

Big Data thus produces many challenges in the whole Data Analysis Pipeline

Most of them related to the four V’s:

- **Volume** - Handling the storage and organization of massive data amounts
- **Velocity** - Capturing data and storing it at live time
- **Variety** - Handling unstructured data, with a variety of sources, formats, encoding, context etc
- **Value** - Turning that data into actionable knowledge (e.g making the data “useful”)
Big Data comes with Big Challenges

“Big Data also brings enormous challenges, whose solutions will require massive disruptions to the design, implementation, and deployment of data management solutions.” [1]

The Beckman Report identifies a few overarching challenges with Big data:

- Scalable big/fast data infrastructures
- Coping with diversity in data management
- End-to-end processing of data
- The roles of people in the data life cycle
Scalable big/fast data infrastructures

Parallel and distributed processing:
- Seen success in scaling up data on large number of unreliable, commodity machine using constrained programming models such as MapReduce (more on that later!).
- The Open Source platform Hadoop, with its MapReduce programming model, soon followed.

Query processing and optimization:
- To fully utilize large clusters of many-core processors more powerful, cost-aware and set-oriented query optimizers are needed.
- Query processors will need to integrate data sampling, data mining, and machine learning into their flows
Scalable big/fast data infrastructures

New hardware:
- In addition to clusters of general-purpose multicore processors, more specialized processors should be considered.
- Researchers should continue to explore ways of leveraging specialized processors for processing very large data sets. Examples of this are:
  - Graphics processing units (GPUs)
  - Field-programmable gate arrays
  - Application-specific integrated circuits (ASICs)
- Adapt parallel and distributed query-processing algorithms to focus more on heterogeneous hardware environments
Scalable big/fast data infrastructures

High-speed data streams:
- High-speed data sources with low information density will need to be processed online, so only the relevant parts are stored.

Late-bound schemas:
- For data that is only used once or never, it makes little sense to pay the substantial price of storing and indexing it in the database.
- Requires query engines that can run efficiently over raw files with late-bound schemas.

Cost-efficient storage
- SSDs are expensive per gigabyte but cheap per I/O operation.
Scalable big/fast data infrastructures

Metrics and benchmarks:
- Scalability should not only be measured in petabytes of data and queries per second, but also:
  - Total cost of ownership (e.g. management and energy costs)
  - End-to-end processing speed (time from raw data to eventual insights/value/actionable knowledge)
  - Brittleness (e.g. the ability to continue despite failures or exceptions, such as parsing errors)
  - Usability (e.g. for entry level users).
Coping with diversity in data management

No one-size-fits-all:
- Variety of Data is managed by different softwares (programming interfaces, query processors & analysing tools).
- Multiple classes of systems are expected to emerge each addressing a particular need like data deduplication, online stream processing.

Cross-platform integration:
- Due to diversity in systems, Platforms are needed to be integrated for combining and analysing data. By doing this it will:
  - Hide the heterogeneity of data formats and access languages.
  - Optimize the performance of data access from and data flow in, big data systems.
Coping with diversity in data management (Cont.)

Programming model:
- More Big data analysis languages are needed in addition to: Julia, SAS, SQL, Hive, Pig, R, Python, a Domain specific language, MapReduce, etc.

Data processing workflows:
- To handle data diversity, Platforms are needed that can span (bridge) both raw and cooked data.
  - Cooked data, eg: tables, matrices, or graphs.
  - Raw data: unprocessed data.
- Big data systems should become easily operable.
  - Eg: Cluster resource managers, such as Hadoop 2.0’s YARN.
End-to-end processing of data

“Few tools can go from raw data all the way to extracted knowledge without significant human intervention at each step.”

Data-to-knowledge pipeline:
- Data tools should exploit human feedback in every step of analytical pipeline, and must be usable by subject-matter experts, not just by IT professionals.

Tool diversity:
- Multiple tools are needed for solving a step of the raw-data-to-knowledge pipeline and these must be seamlessly integrated and easy to use for both lay and expert users.
End-to-end processing of data (cont.)

Open source:
- Few tools are open source, Most other tools are expensive. Therefore, existing tools cannot easily benefit from ongoing contributions made by data integration research community.

Knowledge bases (KBs):
- Domain knowledge, which is created, shared and used to understand data, such knowledge is captured in KBs.
- KBs are used for improving the accuracy of the raw-data-to-knowledge pipeline.
- Knowledge centers provide the tools to query, share and use KBs for data analysis.
- Goal: More work is needed on tools for building, maintaining, querying, and sharing domain-specific KBs.
Roles of people in the data life cycle

“An individual can play multiple roles in the data life cycle. People’s roles can be classified into four general categories: producers, curators, consumers and community members.”

Data producers:
- Anyone can generate data from mobiles, social platforms, applications, wearable devices.
- Key challenge for database community is: To develop algorithms that guide people to produce and share most useful data, while maintaining the desired level of data privacy.

Data curators:
- Data is not just controlled by DBA or by an IT department, but variety of people are empowered to curate data. Approach: crowdsourcing.
- Challenge is to obtain high-quality datasets from a process based on imperfect human curators.
- Solution can be by building platforms and extending relevant applications to incorporate such curation.
Roles of people in the data life cycle (cont.)

Data consumers:
- People want to use messier data in complex ways.
- Common man may not know how to formulate a query. Therefore require new query interfaces.
- We need multimodal interfaces that combine visualization, querying, and navigation.

Online communities: (people interact virtually)
- People want to create, share and manage data with other community members.
- Challenge: Build tools that help communities to produce usable data as well as to share and mine it.
Community Challenges

Database education:
- Old technology taught today: memory was small relative to database size, servers used single core processors.
- Knowledge of key-value stores, data stream processors, MapReduce frameworks should also be given with SQL & DBMSs.

Data science:
- Data scientists can transform large volumes of data into actionable knowledge.
- Data scientists need skills not only in data management, but also in business intelligence, computer systems, mathematics, statistics, machine learning, and optimization.
Introduction to MapReduce

A bit of history first:

MapReduce was created at Google in the early 2000’s to simplify the processes of creating application that utilized large clusters of commodity computer. It simplifies this by handling the error-prone tasks of parallelization, fault-tolerance, data distribution and load balancing in a library.

“A typical MapReduce computation processes many terabytes of data on thousands of machines” [2]
Introduction to MapReduce

What exactly is MapReduce then? In short terms, it is:

- A programming model with a restrictive functional style
  - No mutability simplifies parallelization a great deal as mutable objects can more easily lead to nasty stuff like starvation, race-condition, deadlocks etc.
- The MapReduce run-time system takes care of:
  - The details of partitioning the input data
  - Scheduling the program’s execution across a set of machines
  - Handling machine failures by re-execution
  - Managing the required inter-machine communication.
- The user specifies a map- and a reduce function (hence “MapReduce”), in addition to optional functions (like a combiner) and number of partitions.
Introduction to MapReduce

More on Mapping and Reducing

- The Map- and Reduce functions operate on sets of key/value pairs.
- The Mapper takes the input and generates a set of intermediate key/value pairs
- The Reducer merges all intermediate values associated with the same intermediate key.

```java
map (k1, v1) → list(k2, v2)
reduce (k2, list(v2)) → list(v2)
```

```java
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += parseInt(v);
    Emit(AsSTRING(result));
```
Introduction to MapReduce

Let’s take a simple example:

Let’s say we have a text and we want to count the occurrence of each individual word. The original input (1), “Hello class and hello world”, is first split by line (2). Each split is then sent as a single key/value pair to separate mappers, on separate nodes, where it is “mapped” to several key/value pairs. In our simple example the key is the word itself, and the value is set to 1. We shuffle and sort (4) the keys/values pairs and send them to the reducers. Our reducers (5) is sent different maps and merges the values that shares a key. The different partitions are merges to an output file (6).
Implementation

Different implementations are possible, it depends on the environment. This environment has large clusters of commodity PCs connected together with switched Ethernet. (wide use at Google)

Execution overview:

- User specifies the input/output files, number of Reduce tasks(R), Program code for map() and reduce() and submit the job to scheduling system.
- One machines becomes the master and other workers.
- Input files are automatically splitted into M pieces. (16-64MB)
- Master assigns map-task to idle workers. Worker read the input split and perform the map operation.
- Worker stores the output on local disk which is partitioned into R regions by partitioning function and informs about the location to the master.
- Master check for ideal reduce worker and gives the map output location.
- Reduce worker remotely reads the file and performs the reduce task and stores the output on GFS.
- When the whole process is completed and the output is generated, the master wakes up the user program.
Figure 1: Execution overview
Fault Tolerance:
MapReduce library is designed in a way that it handles machine failures gracefully.

Worker failure:
- Master pings every worker periodically > if (noReply) > mark worker fail.
  - Any completed/in-progress map task (because output on local disk) or in-progress reduce task > Reset > scheduled to other worker.
- Completed reduce tasks > not re-executed > Because output stored on global file system (GFS).
- if any map task executing worker fails, All reduce task executing workers are informed.

Master failure:
- Master write periodic checkpoints of master data structure.
- if master task dies > new copy can be started from last checkpoint.
- if master fails (rarely happens) > MapReduce computation fails.

Locality:
- For saving network bandwidth, GFS divides input data into 64 MB blocks and stores several copies of each block (typically 3) on different machines.
Backup Tasks mechanism:

**Straggler**: In a MapReduce operation, it is a machine that takes an unusually long time to complete one of the last few map or reduce tasks.

**Why stragglers arise?**: Many reasons, eg. Machine with bad disk (slow read), Moreover other scheduled tasks waiting in queue.

**How to reduce this problem**: To save time, when MapReduce is near to completion, Backup task mechanism backup the remaining *in-progress* tasks. And the task is marked as *completed* whenever either the primary or backup execution is complete.
“These are few useful extensions, which can be used together with the Map & Reduce functions.”

Partitioning Function:
- A partitioner function partitions the key-value pairs of intermediate Map-outputs.
- It partitions the data using a user-defined value, which works like a hash function.
- The total number of partitions is same as the number of Reducer tasks for the job.
- Default partitioning function >>> hash(key) mod R

Combiner function:
- Used for partial merging of data.
- Combiner function is executed on each machine that performs map task.
- Same code is used to implement both combiner and reduce functions.
- Difference is: How MapReduce library handles output of these functions.
- Combiner function output > Intermediate file. Reduce function output > Final output file.
Input and Output types:

- MapReduce library provides support for reading input data and writing output data in several different formats.
- User can add support for new input type by implementing Reader interface.

Skipping bad records:

- If there are Bugs in user code or in a third party library that causes map and reduce functions to crash on certain records, the records are skipped. Because sometimes, it is not feasible to fix bugs.
Performance

Factors affecting performance

- Input/Output (Harddisk)
- Computation
- Temporary Storage (Optional)
- Bandwidth
Performance

Traditional Approach Vs Big Data Approach

Data is fetched

Data is transferred

Code is fetched

Code/Result is transferred (That too on local Disk)
Performance

Performance Analysis based on Experimental Computation

Running MapReduce for GREP and SORT.

Figure 2: Data transfer rate over time
Performance

Effect of Backup tasks and Machine Failure.
Performance

Improved Performance cases

Google search engine indexes.

- Smaller Code
- Separating unrelated concepts
- Self Healing of failures
MapReduce Conclusions

- Easy to use
- Simplifies large-scale computations that fit this model
- Allows user to focus on the problem without worrying about details.
- Commodity Hardware

Any questions?