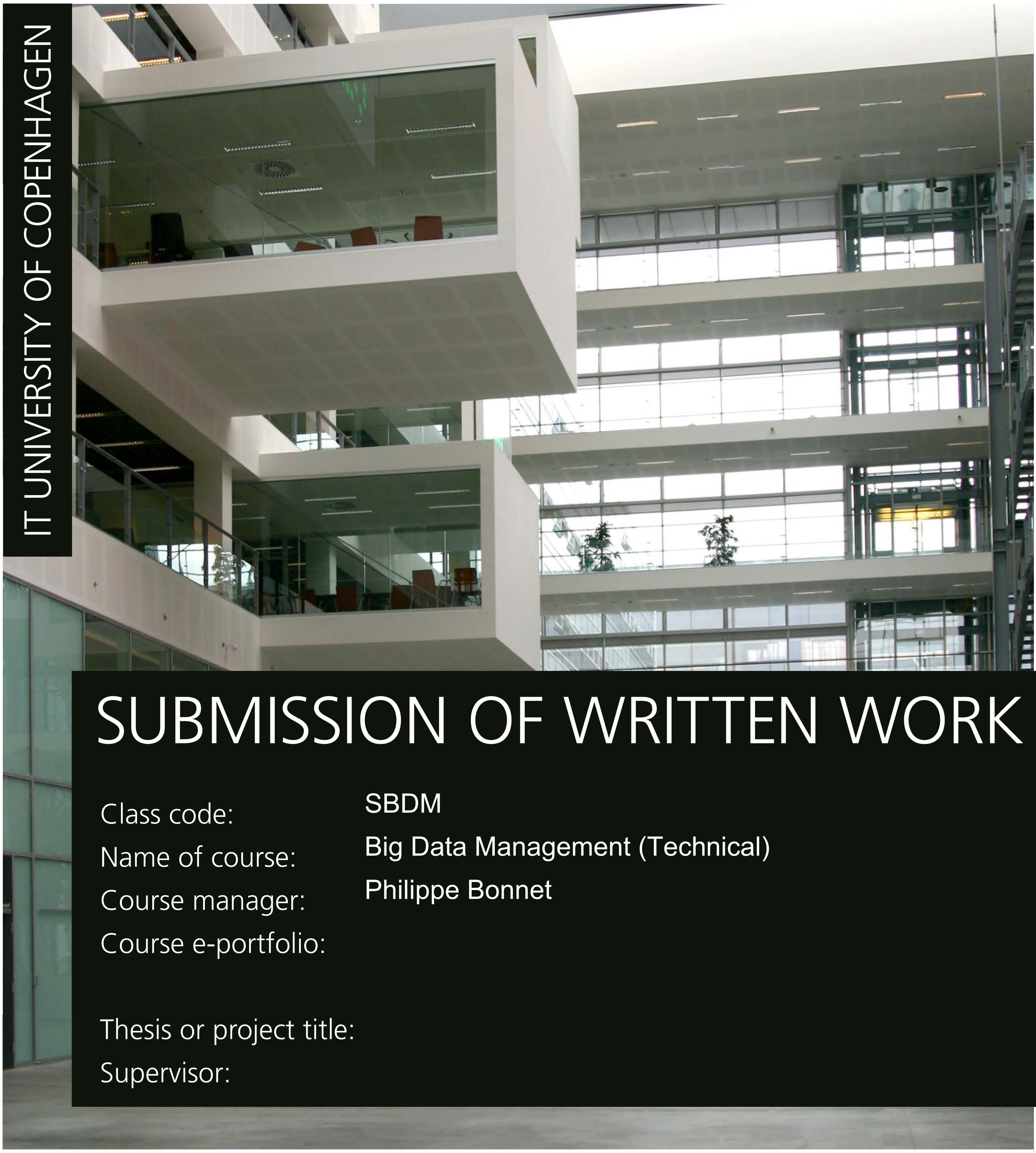

IT UNIVERSITY OF COPENHAGEN

SUBMISSION OF WRITTEN WORK

Class code:	SBDM
Name of course:	Big Data Management (Technical)
Course manager:	Philippe Bonnet
Course e-portfolio:	
Thesis or project title:	
Supervisor:	

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Anders Wind Steffensen	10/02-1993	awis@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Big Data Exam - Autumn 2016

Anders Wind Steffensen

December 13th 2016

Contents

Contents **3**

1 Question 1 **4**

2 Question 2 **6**

3 Question 3 **7**

4 Conclusion **8**

1 Question 1

1.1 A)

"Consider Apache Flink: `https://flink.apache.org`*. You should characterize this system, describe how it can be used in the context of the Lambda architecture and compare it with systems you have used during your projects."*

INTRO Apache Flink(from here just Flink) is a streaming dataflow engine. It works in a distributed setting and makes analysis of streaming data(data in motion), and batch data(data at rest) analysis easier. It incorporates multiple other systems, for machine learning, graph-analysis, and more. To further characterize Flink I will use the characterization model presented in the course.

Datamodel: Flink works on event-based streams of data. The specific format it works in is Java and Scala embedded objects, but in some cases also works with Python objects. By allowing objects of defined languages it is easy to incorporate Flink with already existing systems, that uses these languages.

Partition Management: To be able to scale, Flink builds upon Map-Reduce

check

and therefore uses the same abstraction of being able to divide work among a collection of nodes, which can be placed on the same server or distributed on multiple machine on a network. Map-Reduce uses a hashing

check

function for sharding.

Flink supports replaying of a stream to be able to recover from failures, that is if a failure occurs the stream is replayed from the last checkpoint. At a checkpoint the nodes have stored the incoming information and the job. This also implies that if the stream is infinite, some threshold of how long a node should store information must be provided to the framework. Flink uses Kapfka, which is a stream collecting system, to store the data of the checkpoints.

Batch and Stream Processing: Flink provides two APIs, one for batch analysis and stream analysis. Since Map-Reduce streams HDFS files to do batch processes, Flink has implemented batch processing as a special case of stream-processing, greatly simplifying the process. The only difference is that while streaming data is infinite, batch data is finite. The two API's can be used from Java or Scala, and provide a Java-Streams-Like interface, where it is easy to do typical SQL commands, such as where, grouping, sum and so on. Furthermore Flink has made it possible to easily define a window of the stream to allow for more sophisticated analysis.

Flink provides Pipelining to make nodes able to concurrently work on different tasks, even on different machines.

Throughput:

LAMBDA

IN REALATION TO PROJECTS

1.2 B)

"You are asked to store a master data set of 80 GB given to you as an XML file. Why is the XML data format problematic when working with Map-Reduce? Would a format transformation from XML to JSON be helpful? Would a transformation from XML to CSV be helpful? How would you store this master data set? Explain your answers"

The XML format is in a tree structure, and might not be easily partitioned into smaller parts, which can be distributed among the data nodes of the storage system that Map-Reduce works on. Furthremore XML is also a very verbose format and therefore data will take up more space which will make the computations somewhat slower and require more space on the server, even though hadoop of course handles this quite fine, less space use is always good to be preffered when the available information is the same. Transforming the data to JSON would mostly help on the amount of data, since JSON is less verbose but JSON is still a tree-structure language and therefore partitioning would still be a bit difficult. Because Json objects do not specify a start and an end tag it can actually be more difficult to split up than xml. CSV on the other hand is flat data structure and therefore is easily partitioned per line and therefore allow map-reduce to work on multiple processes, greatly increasing performance.

It is important to notice that as soon as one transforms data, it per definition becomes derived data. Therefore, the master data set will be derived, which puts a question on what happens to the primary data. I will argue that a transformation can be done from XML to CSV, which allows one to go through a similar process which converts the resulting CSV back into XML data which, even though is not the same 1's and 0's, is equal to the primary data, and therefore just keeping the CSV and not the primary data is enough.

1.3 C)

"Describe pros and cons of using the Hadoop ecosystem, based on the lessons you learnt from project 2 and project 3."

The Hadoop ecosystem, has changed the industry, and how it looks at data in general. By going from a restricted structured boxed view, the big data movement tries to break these boundaries but it is still in its youth. The relational databases go back to the 1970s and have had many years to polish its rough edges and making it easily available to developers. The big data movement is still trying to do this and most frameworks in the Hadoop data system exactly tries to sell it self as easy to use, but in my experience most of these systems still have a high learning curve. Furthermore setting up a server with a Hadoop ecosystem requires a lot of time. Systems like Horton tries to make this process easier by creating a single entrypoint for organizing and managing a large amount of the different hadoop frameworks.

Another con is that integrating different frameworks is often difficult. Since there does not exist a standard often times frameworks are made to integrate well with other specific frameworks, but if it is desired to integrate with another system then the developers are often left to figure out how and if it is possible themselves.

For small systems, or embedded systems where it is known that the amount of data will never surpass a low limit, the overhead of using big data technologies is also often not worth it. Then using relational database systems, can be enough

A lot of the frameworks have a lot of overhead on what they do, even though they scale better. If you know you will never store a lot of data, or want it to be available on the device of the user, going with a SQLlite or a system specific Relational Database would be smarter.

That said, the Hadoop ecosystem really shines when it comes to large amounts of data. A lot of businesses saw a huge rise in the amount of data they save through the 2000s and now that processors were nearing their clock speed limit, being able to to scale systems vertically were very important. The Hadoop ecosystem is build around concepts of being able to abstract the distributiveness of the data away and allow developers to write code which automatically would scale to an arbitrary amount of machines. Even in the case of machine breakdowns the frameworks of Hadoop will handle it and be able to replay, reroute, or abandon the process, and the developers are able to specify which approach should be taken as to how to restore the data of that node.

2 Question 2

2.1 A)

"Consider the data set from project 3. How much of the work you did in project 2 to clean data could be reused to clean the data set from project 3? Explain your answer."

From a code perspective it would be difficult to reuse the source code of Project 2 to clean the data from Project 3, mostly because we used the serialization framework AVRO, which then requires the data to be in a certain format to use as input and outputs it in a certain way as well. One could have very generic mappers and reducers which in combination could have had the same effect and could have been put together differently to match this project.

We used streaming to clean the data in Project one so in some sense it would be possible to reuse the streaming part, since when streaming it is possible to handle the 80GB of data in Project 3, but with some other logic on what data to remove, label or ignore.

One of the kinds of cleaning that would need special development for Project 3 is the fact that sometimes, when cars have been staying still for too long they are randomly teleported to other parts of the map. If some analysis would be done on location, some cleaning process needs to handle this.

2.2 B)

"Describe a cleaning process for the data set in project 3. Describe the design of a system that implements this cleaning process."

A cleaning process over this data could include checking for valid values, such as speed values that are positive or 0. Then checking whether or not each value has a type, such as Vehicle-type or Person-type. Another step in the cleaning process would be to check for missing information or information which should not exist for an entity. Another more difficult part of cleaning the data would be to check for the teleporting vehicles. This would require some table of information on where each car was last measured and if the distance from that point to the current point was to far, then the next entity should be labelled something saying which would make it possible to handle this in the analysis.

To create such a cleaning process, I would create a Map-Reduce program such that it can handle the large amounts of data. Then I would create a mapper for each of the different procedures, and checks, which each output to the next mapper in a pipelining fashion. By doing this it is possible for map reduce to parallize as much of the process as possible. A reducer could then be placed at the end, aggregating all the results into two lists of entitites, one for vehicles and one for people. At the end the results could be stored on HDFS, ready for batch or streaming analysis.

3 Question 3

3.1 A)

"Assume that the data from project 3 is not a massive data set, but a data stream. Every time step, a large collection of vehicles and persons is generated (based on the attributes contained in the ¡vehicle¿ and ¡person¿ elements of the XML file given in project 3). How would you proceed to characterize such a data stream?"

The data of the stream contains structured data since it is in XML format. The structure is as follows:

```
<Timestep>
    <vehicle, id, x, y, angle, type, speed, pos, lane, slope/>
</Timestep>
```

or

```
<Timestep>
    <person, id, x, y, angle, speed, pos, edge/>
</Timestep>
```

depending on which type the input has. This information, since it is in XML is easily obtained since the structure of the data is part of the data itself. Had the data been in JSON format it would be more difficult, albeit not impossible to find this structure, and had the data been unstructured it would have been even more difficult, since one should try to create and fit a schema at the same time.

Furthermore to describe the data stream one could examine the number of discrete elements that are present over timesteps in total or an average of that. Given these values it becomes obvious that we need big data systems to handle the stream. This could be done either with batch jobs, but it could also be done by making windowed stream analysis. By examining a window in the stream one could approximate the average amount of entities pr timesteps.

3.2 B)

"Describe a meaningful view based on the data set from the Project 2 data set. How do you obtain that view? Describe the problems you faced obtaining such views in project 2 and how you fixed them."

I have choosen to showcase the third view from our Project 2 as I find that the most interesting. Furthermore we did not describe

4 Conclusion