Analyzing the Twitter Data Stream Using the Snap! Learning Environment

Andreas Grillenberger and Ralf Romeike

Friedrich–Alexander–Universität Erlangen–Nürnberg (FAU)
Department of computer science, Computing Education Research Group
Martensstraße 3, 91058 Erlangen, Germany
{andreas.grillenberger,ralf.romeike}@fau.de

Abstract. In the last few years, tremendous changes have occurred in the field *data management*, especially in the context of *big data*. Not only approaches for *data analysis* have changed, but also *real–time data analyses* gain in importance and support decision–making in various contexts. One of the most exciting approaches for processing and analyzing large amounts of data in nearly real–time are *data stream systems*.
In this paper, we will demonstrate how such developments in CS can be introduced in CS education by using data stream systems as an example. We will discuss these systems from a CS education point of view and describe an approach for carrying out data stream analysis by using the Twitter stream as data source. Also, we will show how the programming tool Snap*!* can be extended for supporting teaching in this context.

Keywords: Big Data · Data Management · Data Stream Systems · Twitter · Snap*!* · Real–Time Data Analyses · CS Education

1 Introduction

In modern computer science, a major challenge is to process and analyze large amounts of data. Such *data analyses* are central to *big data*—a topic that is frequently being discussed nowadays, not only in CS and in the economy, but also in politics, society and daily life. Especially, the impact of *real–time data analyses* is increasing tremendously. At the same time, data analyses are hard to notice at all in everyday life, but will become even more important with emerging technologies, like the *Internet of Things* or *Cyper-Physical Systems*, as they will provide many additional data and use cases.

While discussions on data are often focused on storing large amounts of them, for example generated by early data retention projects or intelligence agencies, this aspect is a minor challenge today. Instead, the main difficulty is to process and analyze these growing amounts of data. This leads to a new view on data processing: while traditional data analyses are especially focused on relatively static data, typically stored in a database, today data are rather dynamically changing. Also, traditionally it was sufficient to generate results eventually after capturing the data, but today's analyses are often focused on

immediate reactions, like in a tsunami warning system based on seismic sensors. However, CS education in this context mainly focuses on storing data in a proper way, often using databases and the relational data model as example, while the aspect of analyzing data is typically left out or only considered marginally (cf. [4]).

In the context of such developments, CS education is confronted with the challenge of keeping track with them and incorporating the basic principles into teaching. However, the complexity of such topics makes this a difficult task. An example is big data: despite its relevance for the student's daily life (cf. [6]), this topic is highly complex and hence appears difficult to include in teaching. In this paper we will discuss one of the emerging approaches for handling big data, *data stream systems*, concerning the new view it brings to data management, as well as its main working principles. Thereafter, we will describe an example of how to introduce this topic in teaching and how to arrange teaching in this context in a student–oriented way using active learning, while also fostering competencies that can be used for overcoming today's flood of information. Additionally, by using Snap*!* [7] as an example, we will show how an universal programming tool can be extended in order to support teaching of the principles of such new developments.

2 Data Stream Systems

2.1 Function of Data Stream Systems

In modern information systems, the challenge of handling large amounts of data in nearly real–time is becoming increasingly important. While data are typically stored and processed using *Database Management Systems* (DBMS), this approach is being challenged by increasingly large amounts of varying data in short time–spans (*big data*), because traditional approaches can hardly fulfill modern requirements like real–time analyses. Instead, they are designed for use cases in which immediate reactions are not required, such as analyzing business data. But when direct reactions are essential, immediate processing of data is inevitable, e. g. when measuring high values of tectonic movements in a tsunami warning system. While in some cases this challenge might be overcome by accelerating data processing using more powerful machines, this cannot solve the problem in general. A fundamental question in this context is: "*Why do we store* ***all*** *the data?*" In modern *data management*, one approach to address this problem is not to store all the data but only the uncommon ones which have a higher self–information, while presuming that if nothing was stored, everything was as usual. Coming back to the tsunami warning system, it does not make sense to save values of nearly no tectonic movement. While the traditional approach tries to gather as much data as possible on an object/model in order to enable any desired further analysis of them (cf. fig. 1), the approach of *data stream systems* (DSS) is only suitable when there is a clear analysis goal and when criteria can be defined before starting the analysis: they analyze a data stream by filtering out the relevant data on–the–fly (cf. fig. 2). In an analogy, we can describe

the database approach as a hamster who collects food on stock, while DSS are characterized by a bear who catches fishes only when he is hungry.

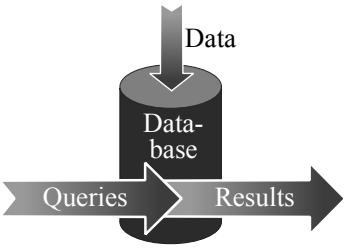

Fig. 1. Function of a database system

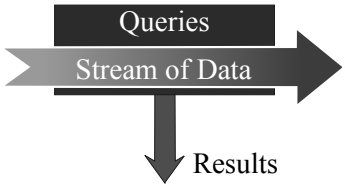

Fig. 2. Function of a data stream system

The main working principle of data stream systems is to execute queries on "*[. . .] a real–time, continuous, ordered (explicitly by time stamp or implicitly by arrival time) sequence of items*" [3], called *data stream*, instead of one–time queries on stored data. A basic assumption is that when data are arriving in a specified order, each new datum adds new information to the previously received ones or revises them. In consequence, a main characteristic of DSS is that they also produce a continuous stream of results instead of occasional results only when executing a query.

Both, DBMS and DSS, are generally suitable for processing large amounts of data, but they are optimized for different tasks: as databases store all the data for a longer period of time, it is possible to analyze them for correlations or patterns in the data, even such that were unexpected. Such analyses are summarized under the term data mining: "*the process of discovering interesting and useful patterns and relationships in large volumes of data*" [1]. Instead, digging for hidden information without having concrete criteria is not possible using DSS, because the analysis criteria must be defined before the data arrive. For combining the advantages of both systems—immediate reactions, but also long–time availability of all data—using a combination of both types is a promising approach.

2.2 Usage Examples of Data Stream Systems

A typical domain of DSS is monitoring data streams for defined events/criteria. In the following, we will characterize the use of these systems by describing Twitter analysis as example, but most of the described characteristics can also be found when considering other services. While social media play an important role for most students today, they are also a rich data source, e. g. for predicting upcoming trends or for product marketing. We decided for using Twitter as example, as up to 6,000 tweets are posted per second [8] and can be easily accessed using the Twitter API. With each tweet containing not only up to 140 characters

as message, but also about 150 additional attributes [2] (e. g. unique ID, author, followers, time stamp, geographical origination, language, information on the profile page of the author), Twitter is a rich information source.[1] When only considering the tweet text (assuming on average 70 characters) and storing it in UTF–8 encoding, this makes about 200 bytes per tweet, in combination with the metadata, a conservative estimation would be around 500 bytes per tweet, which means 3 MB per second or 259.2 GB per day. With these large amounts of data, and especially metadata, various interesting and meaningful analyses can be done: analyzing trends, like Twitter does directly on its start page, reviewing the success of newly released products (even grouped by countries, for example) or generating accurate election forecasts (e. g. [9]). A concrete example is product marketing: when releasing a new product, a typical task is to analyze its success. While such analyses can deliver important results on how to improve the next product, another important aspect is to react to discussions on it. So, an exemplary task is to find regions in which the product needs to be advertised more intensively in order to ensure better success.

Another example for data stream analyses is monitoring of sensors or services like websites. Especially, website monitoring is also highly versatile: e. g. periodically checking if a website is online or offline, monitoring its performance or checking if the price of an item in an online shop has changed. In all these examples, the data source is a continuous stream of data, of which most of the data are not interesting, but the really interesting values can be filtered out efficiently by defining appropriate criteria. A suitable way for depicting the data flow of such analyses is using data flow diagrams. In fig. 3, we depicted such a diagram for monitoring changes on a website.

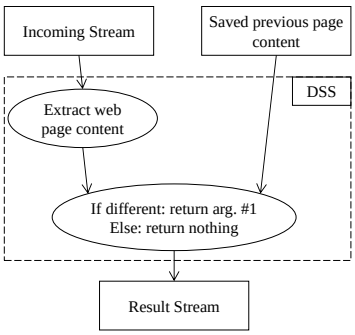

Fig. 3. Data flow of a continuous query for monitoring website changes

2.3 Principles of Data Stream Analyses in Daily Life

The same principle that was used in the previously described examples is also present in everyone's daily life, as they describe the monitoring of data sources, something that everyone does regularly: before refueling our car, we monitor the gasoline price and react as soon as we recognize a cheap price, and when buying goods, we watch the price and buy them when they are on sale. Also, we can use sensors to capture information that is not originally available digitally. This is the case for various innovations in home automation: measuring sunlight for automatically controlling the window shades or measuring temperature for controlling the heater. Even when working in the garden, there are

[1] A complete overview of which metadata a tweet contained in 2010 can be found at http://online.wsj.com/public/resources/documents/TweetMetadata.pdf (last checked: 2015-07-20, created by Raffi Krikorian)

various monitoring aspects, like pouring water on the flowers or the lawn only if they are becoming too dry. Systems that are used for the automation of such tasks are often referred to as *Cyber–Physical Systems* (CPS): they perceive their environment using complex sensor structures, react to changes, influence their environment and also communicate with other systems. Such systems do not only enable everyone to monitor and to control their own environment, but they also create new information sources by providing almost every item with its own digital identity—one of the main principles of the *Internet of Things*. This paradigm change in the relationship between physical and virtual objects and practices illustrates the importance of competencies that are needed to cope with the new requirements arising therefrom. Supporting the formation of such competencies is an important task of CS education, which also requires some fundamental knowledge on the basic working principles of such systems.

3 Using Snap! for Twitter Analysis

For CS education, introducing modern approaches like DSS into teaching is a complex task. In particular, topics like big data also cause changes in the relevance of various other concepts, set new emphases and require new examples (cf. [5]). On the other hand, incorporating such topics into teaching enables students to recognize the broadness of CS as well as the chances of using modern CS methods. In this sense, DSS are prototypical for the characteristics of modern data management, e. g. the difference between data and information, the value of data and metadata, standardized data interchange formats, and data analysis methods. With DSS as example, we will illustrate how CS education can support understanding the basic working principles of such modern developments and the acquisition of competencies in this context. For understanding the principles of DSS, students need to be able to use such a system by themselves in order to ensure student–oriented teaching and facilitate active learning. So, in the following we want to demonstrate, how the principles of these systems can be taught using an easy–to–use data stream analysis tool. As typical professional data stream systems are too complex for discovering the main working principles without having in–depth knowledge on the software, we decided to implement a school–focused working example of a DSS based on the easy–to–use programming tool Snap*!*. In the following we will first describe how this universal programming environment can be used for demonstrating the principles of DSS, in this example using the Twitter stream as data source. Thereafter, we will present the central aspects of our implementation. Our main reason for choosing Twitter is that it offers large amounts of data in an easily accessible way: there are two APIs (one for discrete data, one for streaming data)[2], which are opening up various possibilities. For the described use, mainly the streaming API is of interest: it provides three different endpoints, which means three different data streams.

[2] The Twitter APIs are described in detail at https://dev.twitter.com/overview/documentation (last checked: 2015-07-20)

In this example we will access the so–called “filter–stream”, which in our tests provided about 16 tweets per second, each enriched with location metadata.

3.1 Possibilities and Usage

Some examples for analyses that can be done using the Twitter data at school are coming from the previously mentioned context product marketing. For doing such analyses, we need to access the tweet’s text, its geographical location, perhaps hashtags (if provided), retweets or likes and so on. All these information are provided by the Twitter stream, so we only need to make them accessible in Snap*!*. Therefore, we implemented various blocks for accessing and processing the Twitter data. In the following, we will present a more detailed view on these blocks, while technical details will follow in section 3.2.

The “get next full tweet” block (fig. 4) reads the next tweet from the helper via a HTTP request and returns all attributes as JSON formatted string. If no value is returned by the helper, it likely means that the helper is not running.

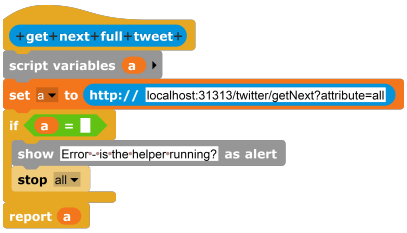

Fig. 4. Implementation of the “get next full tweet” block

The “read attribute from tweet” block (fig. 5) typically gets one tweet in JSON format as argument, which it processes via a simple JavaScript function, which parses the JSON string and returns the requested attribute.

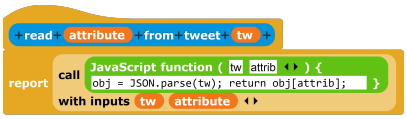

Fig. 5. Implementation of the “read attribute from tweet” block

The “for each tweet do” C–shaped block (fig. 6) is implemented using a forever loop, in which it reads the next tweet, determines if its text has at least one character and then executes the given lambda function (which is in the user–view represented as all blocks inside the C–shape), with the JSON formatted tweet as input.

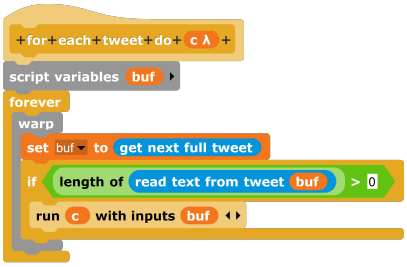

Fig. 6. Implementation of the “for each tweet” block

Using these blocks, students can perform various analyses on tweets, for example by keyword, language, countries, length of tweet, hashtags or the color of the author’s profile page. Because only getting numerical and textual results is not a very motivating and interesting outcome, we also implemented a block for showing the location of tweets on a map. As most simple example, in fig. 3.1, we have depicted all tweets that arrived during a time frame of 5 minutes on a map without doing any additional steps. Another visualization that we have implemented, in particular for

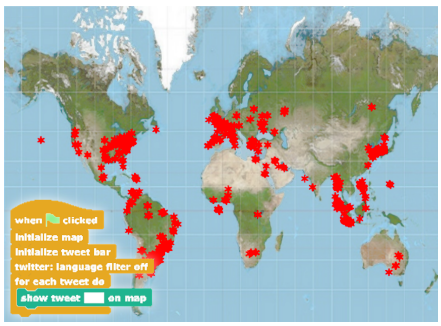

Fig. 7. Map visualization of tweets. Map: ©2011 Strebe, CC BY–SA 3.0

Fig. 8. Bar chart showing the amount of tweets in different languages.

doing simple statistical analyses, is a bar chart that can be generated out of a list of integer values. In fig. 3.1, we depicted another simple example: determining the language of the tweet and generating a chart showing the relative frequency of the languages English, German, Spanish, French, Japanese as well as all others. These visualizations and analysis blocks can be used by students to do analyses depending on their own interests, for example they could analyze which country likes which colors most (determined by the author's profile background color), where a current topic seems to be most discussed or which country favors which stars.

A concrete example for an analysis task in the context of product marketing that can be done at school using our tool is measuring the spread of products in different countries. So, students can e. g. analyze how much the smartphone platforms Android, iOS and Windows as well as their environment are discussed by doing a keyword analysis. By mapping three sets of keywords for these three categories in different colors, the students can get a good overview. However, deeper insights are only possible by using the bar chart and, for example, by restricting the counted tweets to specific countries or regions.

3.2 Realization and Technical Aspects

For implementing the described functions in Snap*!*, we have chosen an approach that can also be transferred to other data sources (like RSS feeds, website data or sensor data). As both, our approach and Snap*!* in general, can be used, adopted and extended in a versatile way, in this section we will describe essential details of the implementation that are also relevant for further extensions, using other data analysis approaches and for connecting to other data sources, as well as for transferring our extension to other programming environments. In addition, we will clarify the limitations of our implementation of which the teacher should be aware when using this tool.

As Twitter offers various options for accessing its rich data sources, the first decision was which one to use. We are using the streaming API, which provides three different data streams: the whole stream of data, the so–called fire hose, which is only accessible with special permissions, the sample stream providing a 1% sample of all tweets (about 10 to 15 tweets per second in our tests) as well as the filter stream which lets the user define some criteria (like geolocation or keywords) and returned about 12 to 16 tweets per second. So, not only the slightly higher amount of tweets let us choose the filter API, but especially the fact that we could restrict the results to only tweets containing location data.

After this decision, when implementing our tool we were faced a main challenge: directly connecting from Snap*!* to Twitter it not possible because of security measures for preventing cross–site scripting attacks in all typical browsers. Hence, we implemented a helper app which acts as proxy between Snap*!* and Twitter and forwards all tweets to Snap*!* in a suitable way (JSON format). On the one side, the helper app connects to Twitter using its streaming API, on the other side it offers Snap*!* the ability to connect to it by running a small web server and allowing Snap*!* access to it[3]. In Snap*!*, all functions are implemented using custom blocks and the provided Javascript block, but without modifying the Snap*!* source code. So, these blocks can be used in the official Snap*!* installation after importing them.

As a consequence of using a helper application, our implementation cannot fully preserve the data stream character: while there is a typical data stream between Twitter and the helper app, as it is only requested once and then continuously filled by Twitter until the connection ends, we cannot send the data from the helper app to Snap*!*. So, the "get next full tweet" block breaks up the stream character by requesting every single item on its own. However, as this behavior is hidden from the students, this is only a minor restriction. Also, in order to imitate the stream character as good as possible, we do not cache tweets in the helper app, but instead discard them if there is no incoming request from Snap*!* in the short time until a new tweet is being received, so if it is not being processed in time. Nevertheless, teachers using this Snap*!* extension should be aware of this restriction in order to avoid building up misconceptions.

The helper app can also be used with other programming languages and environments that support HTTP requests and parsing text as well as the JSON format, as it provides a universal REST interface and uses JSON for data interchange. So, transferring our solution to e. g. Scratch or AppInventor is possible.

4 Summary

Data stream systems involve various important aspects of CS and in particular of data management. As shown in chapter 2, DSS do not only implement an effective yet easy to understand approach for handling large amounts of data, but they can also serve as an example for data flow modeling, show the principles

[3] This technique is known as cross–origin resource sharing (CORS): the target sites allows remote access to its resources by setting a special HTTP header.

of real–time data processing and point out the necessity for defined interfaces and exchange formats between systems. Discussing the principles of DSS at school can hence not only show some important principles of CS, but in CS teaching, it also helps with understanding topics of current interest, like the chances and risks of data analyses.

With the growing importance of data analysis and the large amounts of data that are being generated today, understanding the fundamental concepts and principles in this field becomes increasingly important for handling own data and for understanding common topics in the modern information–driven society. DSS can help students understanding popular data analysis and discussions on data–driven topics, but they also help recognizing the threats accompanying these possibilities. The described tool also gives them the chance to carry out own basic data stream analyses based on the Twitter feed without the need to understand the Twitter API and without possessing in–depth programming skills. Also, this example of analyzing the Twitter data stream can be transferred to many other examples related to the students' daily life: there is only a slight difference to analyzing RSS feeds or other data sources instead of the Twitter stream. As many use cases of DSS are focused on monitoring, this topic addresses another perspective on CS, as students can relate this to their own activities and understand how to automate tasks by using such systems. So, they can also take advantage of this, e. g. by transferring this knowledge to use cases like analyzing the prices of a concrete flight and alerting you when a defined limit is exceeded.

Additionally, incorporating aspects of modern data management into teaching can also foster the formation of various key competencies that are needed for handling own data in an appropriate and responsible way: for example, in the context of DSS, students need to make decisions on whether to store data in a temporary or permanent way, "*understand the purpose of metadata*" and "*combine data in order to gather new information*" [6]. In particular, with the described Twitter analyses, students would also be able to recognize the value of metadata, as most analyses would be relatively meaningless when only considering the tweet text but none of the additional information like the location.

Data stream systems can hence function as an example for involving the ongoing developments and emerging topics of CS into CS education. While current CS curricula do not or only marginally cover such topics, in future the relevance of modern data management topics in CS education is likely to increase: for example, in the context of the web, networking, protocols and so on. In addition, this article demonstrates that considering common tools of CS education from a broader view and using them in a wider context is a promising approach: in this case, the programming environment Snap*!* could easily be extended to cover aspects of data management and data analysis and to clearly show the main working principles of data stream systems.

References

1. Data Mining. Encyclopdia Britannica http://www.britannica.com/EBchecked/topic/1056150/data-mining

2. Dwoskin, E.: In a Single Tweet, as Many Pieces of Metadata as There Are Characters. The Washington Journal (2014), http://blogs.wsj.com/digits/2014/06/06/in-a-single-tweet-as-many-pieces-of-metadata-as-there-are-characters
3. Golab, L., Özsu, M.T.: Processing Sliding Window Multi-joins in Continuous Queries over Data Streams. In: Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29. pp. 500–511. VLDB '03, VLDB Endowment (2003)
4. Grillenberger, A., Romeike, R.: A Comparison of the Field Data Management and its Representation in Secondary CS Curricula. In: Proceedings of WiPSCE 2014. ACM, Berlin (2014)
5. Grillenberger, A., Romeike, R.: Big Data Challenges for Computer Science Education. In: Glbahar, Y., Karata, E. (eds.) Informatics in Schools. Teaching and Learning Perspectives, Lecture Notes in Computer Science, vol. 8730, pp. 29–40. Springer International Publishing (2014)
6. Grillenberger, A., Romeike, R.: Teaching Data Management: Key Competencies and Opportunities. In: Brinda, T., Reynolds, N., Romeike, R. (eds.) KEYCIT 2014 – Key Competencies in Informatics and ICT. Commentarii informaticae didacticae, Universitätsverlag Potsdam (2014)
7. Harvey, B., Mönig, J.: Snap! Reference Manual (2014), http://snap.berkeley.edu/SnapManual.pdf
8. Krikorian, R.: New Tweets per second record, and how! (2013), https://blog.twitter.com/node/2845
9. Tumasjan, Andranik, Sprenger, Timm O., Sandner, Philipp G., Welpe, Isabell M.: Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. In: Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media. The AAAI Press, Menlo Park, California (2010)

All electronic sources were at last retrieved on 2015-07-20.