Observing Reddit's Interaction Network

A stream-based approach for large scale Network Analysis on Reddit

Andreas Huber

Thesis submitted for the degree of
Master in Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

Observing Reddit's Interaction Network

A stream-based approach for large scale Network Analysis on Reddit

Andreas Huber

Observing Reddit's Interaction Network

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The area of misinformation is attracting considerable interest since the 2016 presidential elections in the United States of America [5] or when conspiracy theories linked 5G with the coronavirus. Misinformation can have drastic implications in the real world. Analyzing social media data offers a way to detect where misinformation originates from and how it spreads. The Understanding and Monitoring Digital Wildfires (UMOD) project at Simula Research Laboratory focuses on monitoring digital wildfires and developing improved prevention and preparedness techniques to counteract misinformation. This work aims to add Reddit as an additional data source to the UMOD project. We, therefore, demonstrate how to build a massive graph between subreddits on Reddit from a parallel streamed dataset. We, therefore, introduce the Reddit Dataset Stream Pipeline (RDSP), a processing pipeline for the Reddit dataset based on Akka Streams. We then show methods to create a graph between the subreddits on Reddit and weigh them. We investigate the generated graphs and present the experiments' results, including a visualization of the graph. This thesis provides the foundation for the UMOD project to investigate further how misinformation spreads on Reddit.

Contents

List of Figures

List of Tables

Acknowledgments

I want to take this opportunity to thank the people who supported and helped while shaping this thesis.

First, I want to thank my supervisor Daniel Thilo Schröder for supporting me in defining the outline of this thesis and shaping this thesis, as well as providing me his expertise in various fields, especially in graph theory. Furthermore, I want to thank my supervisors Johannes Langguth and Carsten Griwodz for their general support, counseling, critically reading this thesis and many practical suggestions for improvements. I want to thank all my supervisors for the time they spent in countless meetings and the valuable insights they provided me.

I want to thank the Simula Research Laboratory for providing me the project and a place to work.

The research presented in this paper has benefited from the Experimental Infrastructure for Experimental Infrastructure for Exploration of Exascale Computing (eX^3), which is financially supported by the Research Council of Norway under contract 270053.

I am thankful to my fellow student Haseeb Rana, who spent his time at the office with me and was always there to discuss new approaches.

Finally, I want to thank my friends and family, especially my wife Aleksandra, for her mental support and for making life more enjoyable.

Part I

Introduction

Chapter 1

Introduction

Fake news and misinformation are becoming a severe worldwide concern. Harmful misinformation spread on social media can have drastic implications in the real world. In recent years the impact of missinformation came into the discussion surrounding the 2016 presidential elections in the United States of America [5] or when conspiracy theories linked 5G with the coronavirus, which resulted in arson and vandalism against phone masts in the United Kingdom [1].

Analyzing social media data offers a way to detect where misinformation originates from and how it spreads.

The Understanding and Monitoring Digital Wildfires (UMOD) project at the Simula Research Laboratory focuses on the monitoring of digital wildfires and developing improved prevention and preparedness techniques to counteract misinformation [25].

For understanding how misinformation spreads, it is crucial to obtain data from various sources. The project previously focused on data collected from Twitter and GDELT. The FACT framework provides capabilities to capture and analyze Twitter graphs [43]. The introduction of Reddit as an additional data source allows a different perspective on the world of missinformation. In contrast to Twitter, it is easier to obtain textual content from Reddit but harder to establish connections between individual users.

Reddit is organized in topical sub-communities. In Reddit's terms, a sub-community is called a subreddit. Those communities are primarily user-generated and user-moderated and come in various sizes [45]. Subreddits are independent, meaning there is no direct connection to link or group subreddits together. A user subscribing to a subreddit indicates that the user has an interest in that topic. However, Reddit does not disclose which users subscribe to which subreddits. What is visible is which users commented or posted (contributed) in which subreddit by looking at all posts and comments. We believe that these expressions of interest in combination with the interaction in specific subreddits open the possibility to observe missinformation in a "map of ideas".

Why is Reddit crucial? Reddit is a social news aggregator, where information from all over the internet is aggregated and discussed in decentralized topics. It is particularly relevant because the discussions on Reddit are open and cover a huge number of topics with different opinions from all around the world. Not having a character limit (unlike Twitter) and the possibility to see a discussion tree also enable more profound and richer conversations. Furthermore, like other social media platforms, Reddit has been targeted by disinformation campaigns before [51].

Why Reddit over time? Misinformation does not appear out of nowhere. Instead, it spreads over time before it becomes visible. To see how misinformation spreads, and not only the finite state is relevant to track the evolution of the spread. Therefore it is necessary to investigate the previously mentioned "map of ideas" over time. Only then dynamics over time will show how misinformation evolves, spreads, and might behave in the future.

This thesis, therefore, aims to build a framework for massive temporal Reddit graph analysis, which allows us to investigate the world of subreddits from its beginning.

1.1 Problem definition

The topic of this thesis is the design of methods to process, create, and understand Reddit graphs in order to be able to analyze the behavior of Reddit over time. The research question of this thesis is:

Given a Reddit datastream, how to build a large-scale data processing framework that allows examining Reddit's interaction graph over time?

We conceive this process as a triad involving data acquisition, massive data analysis, and graph building. Whereby the latter includes the development of a model for discovering and weighing the relationship between subreddits. In the course of the process, we are faced with the following challenges:

Massive Data On Reddit, information from all over the web is gathered and discussed. Considering it claims to have 52 million daily users in October 2020 [30] and according to the Alexa traffic rank the 19^{th} most popular website on the internet [38], we need to deal with one principal problem: The dataset size is enormous, and working with this amount of data is a challenge.

Graph Building Another problem is the data structure. Reddit is organized in topical communities called subreddits, and these communities are independent. They do not have categories, neighbors and not relatable via

the data structure itself. So there is no obvious way of grouping subreddits together. In order to visualize and measure relationships between subreddits, a method to connect subreddits has to be developed. Then it could be possible to find out which subreddits relate to and influence each other.

Graph size With a developed method, there is still the challenge that the graph has to be built. With the high number of users on Reddit, it is to be expected that the graph between subreddits is highly connected. The number of edges in a fully connected undirected graph with n verticies is $\frac{n(n-1)}{2}$. Storing the adjacency matrix thous takes $\mathcal{O}(n^2)$ space. Memory consumption and compute time are therefore a concern.

Temporal Graphs Considering a social network, it is interesting to inspect changes in structure over time. Therefore the time dimension should be considered while generating the graph. Theoretically, we could generate a graph for every change, but this approach is not feasible given the massive data problem. As a solution, we propose to create time slices of the graph, giving us multiple views in time and the possibility to compare them.

Prepare for streaming The UMOD project focuses on the monitoring of digital wildfires and developing improved prevention and preparedness techniques to counteract misinformation [25]. For the project to benefit from the monitoring in its final stage, newer data than the one from archived datasets is required. The created processing pipeline should be built with the possibility of streaming in mind. It then can be extended later so the archive dataset can be combined with live data from Reddit. Processing streams can be more challenging than simple batch processing because new data continuously arrives, and streams are not bounded in size [48].

1.2 Contributions

This thesis consists of three parts. The first part focuses on the Reddit Dataset Stream Pipeline (RDSP) and building a processing pipeline for the Reddit dataset based on Akka Streams. The second part focuses on developing methods to create a graph between subreddits on Reddit and weigh them. The third part discusses the generated graphs and presents experimental results on these graphs.

Reddit Dataset Stream Pipeline In order to tackle the problem of file size, this thesis introduces an approach to read, extract, filter, analyze and transform the Reddit dataset with Akka Streams in parallel. "Akka Streams implements the Reactive Streams standard within the larger Akka concurrency project." [9] Reactive Streams are a way to provide highly responsive applications that can handle multiple requests per second while

also handling backpressure [10]. Since the resulting pipeline is stream-based, it can be extended to function with live data in the future.

Graph Building Method Considering the independence of subreddits, a method to overcome it by building a graph with edges between subreddits from user contributions is shown. Furthermore, a weighting method that takes the importance of neighbors into account is applied to those edges.

Paralell Score Calculation With expanding graph sizes, the time to complete the computation of all scores on the graph single-threaded consumes too much time - and therefore also blocks compute resources for longer than necessary. Considering that it is necessary to test multiple iterations of the algorithm and that the results have to be checked promptly, a single-threaded calculation is not an option. Therefore the scores and weight of edges and vertices are calculated in parallel over multiple iterations to build the graph in a timely manner.

Graph Visualization Understanding if the created graph is meaningful and resembles the landscape of Reddit, a visual representation is needed. Therefore, this thesis shows a visual representation of the subreddit landscape of the top 10.000 subreddits. To get a helpful visualization, edges with low edge weights are filtered, and the graph layout is based on the developed edge weight. This is necessary because otherwise, the rendered graph would show more edges than one can comprehend.

Experiments and Distributions Furthermore, the thesis gathers and discusses the distributions of the scores in the graph and provides a more detailed overview of the landscape of Reddit.

Raw Graphs and Time Slices To make the results more accessible and to make it possible for others to base their work upon these results, with this thesis we release the graphs as DOT files and as edge and vertex CSV lists. Further, pre-generated time slices from 2005 to 2019 are released as well - even though the results are not discussed in this thesis. The time slices are released as DOT files and as edge and vertex CSV lists.

Project Contribution This thesis is the first contribution to the UMOD project concerning Reddit. It makes Reddit as a data source more accessible and provides prepared graphs that can be further explored.

1.3 Outline of the Thesis

Chapter 2 provides a necessary overview on topics concerned in this thesis. Starting from a quick overview about Reddit, why it is relevant, and what the data structure looks like, to the Pushshift Reddit dataset that acts as the main data source. The reader gets a quick introduction to Actors,

Akka, and Akka Streams. Concerning the graph creation, the background chapter provides a short overview of the java library JGraphT. Finally, the chapter presents the eX³ platform [1] and the SLURM workload manager [50] used to schedule work on eX³.

Chapter 3 illustrates the ideas and methods behind the developed software. It mainly focuses on two parts—the dataset processing part with the Reddit Dataset Stream Pipeline and the graph creation part. The dataset processing part provides a short overview of operations and transformations performed on the dataset and how parallelism is solved with Akka streams. It also provides general insight on design choices around the data flow within the actor system. The graph building stage discusses how the data provided by the Reddit Dataset Stream Pipeline is aggregated to build the first graph. Given the first graph, the approach discusses multiple iterations of improving the graph model to finally get a useful weighted graph representing the subreddit landscape of Reddit. The last part discusses the tooling, like the slurm workload manager, to process the datasets on multiple nodes in the eX³ cluster.

Chapter 4 describes the implementation process and gives deeper insight on how to understand and use the programs. As in Chapter 3, the implementation chapter focuses on mainly two parts - the Reddit Dataset Stream Pipeline and the graph building. The Reddit Dataset Stream Pipeline section discusses the whole feature set of the program and the flow of data within Akka Streams in detail. Moreover, it explains all the necessary steps to get the data ready for the graph creation phase. The graph building section discusses building the graph with detailed insight on important design choices during the implementations. Furthermore, the technical details of exporting and loading the graphs section is elaborated. Finally, the section explains how to use the command line interface of the program.

Chapter 5 Based on the Reddit graph, this chapter discusses experiments performed on the graph and our findings. We present basic statistics on the top n subreddits. Further, we present metrics we calculate on the graph and show our attempt at visualizing the graph.

Chapter 6 shows a perspective on how the gained knowledge and toolset can be used and how it will integrate with the UMOD project. The outlook chapter discusses future changes, proposes new changes, and where to go from there. It primarily focuses on what steps have to be undergone to create a system that supports live streaming. Finally, we conclude what has been achieved and what potential there is for future improvement.

[1] https://www.ex3.simula.no/

Chapter 2

Background

Chapter 2 is concerned with providing relevant base knowledge to get a better understanding of this Thesis scope. In this chapter, first, we illustrate what Reddit is, how it is structured and why it is relevant. We present possible Reddit data sources and the Reddit data structure. We then introduce utilized concepts, frameworks, and tools and discuss related work.

2.1 What is Reddit?

Reddit is a social news site that aggregates a continuous flow of user-submitted posts. According to Reddit's CEO Steve Huffman, "Reddit is a platform where users can discuss anything imaginable" [21]. The relevance of a post or comment is determined by its age and how users rank the post by voting it up or down. Reddit shows the most popular posts on its homepage [49]. To better understand how Reddit works, it is essential to discuss its building blocks first.

Subreddits. Reddit is comprised of thousands of primarily user-generated, and user-moderated topical sub-communities of various sizes. In Reddit's terms, a community is a so-called subreddit [45]. A subreddit is identified by its unique URL in the scheme of `/r/subredditname`. The subreddit Politics, for example, is accessible under the URL www.reddit.com/r/politics. A subreddit and its discussions are usually open accessible to be read by everyone that visits the site [2]. Still, there is the possibility to create private subreddits [11], and some subreddits might be quarantined, which hides them from new users [21]. Reddit reported more than 130.000 active communities as of December 4, 2019. [20].

Posts. All registered users can submit posts to a subreddit whether they subscribed to the corresponding subreddit or not. Users can post new content as "self-posts" and links. Self-posts are text posts that are saved to Reddit directly where link posts point to various external sources such as articles, images, or videos [45].

Comments. All registered users can comment on the initial post or in reply to other comments. The result is a hierarchical discussion tree where users answer to previous comments or create an entirely new comment in answer to the post [45].

Voting. Posts and comments can be upvoted or downvoted by registered users. Reddit uses a ranking function based on time and votes. Thereby votes have a direct influence on the order in which Reddit displays the posts on the page.

Karma. Users receive karma points when other users upvote their posts or comments, and they lose karma points when they receive downvotes. For example, if a user's comment receives five upvotes and one downvote, the user gains four karma points for the post. Reddit counts post karma and comments karma independently. Users do not earn karma on self-posts. Users with a large amount of karma are considered more active or valuable members of the community. As a reward, users with a high amount of karma are allowed to post more frequently [49]. The amount of karma a user has earned can be seen by hovering over their username or accessing their profile page [2].

2.2 Why is Reddit relevant to detect misinformation?

The objective of the thesis is to gather social network data from sources other than Twitter and predict information spreading within this data. Although further evaluation of data sources is part of the thesis, my preparations have already shown that Reddit is a potential candidate for a data source due to its open nature, easy to handle API, and the existence of substantial historic datasets [4]. Furthermore, Reddit's spreading data is relatively easy to extract using existing libraries such as The Python Reddit API Wrapper (PRAW) [1] compared to, e.g., Twitter spreading graphs which require one to overcome massive technical obstacles.

Data from Reddit is relevant for research because "...it arguably enables richer user interactions than other sites, such as Facebook and Twitter" [8]. Moreover, the conversations on Reddit are not restricted to one topic per se. Due to its open API, the data is more accessible and not as limited as the data that other social media sites provide through their API. Reddit also claims to be bot and crawler friendly, as long as they do not exceed usage limits or lie about their identity [37].

For the UMOD research project, Reddit is one of many possible data sources. Undoubtedly the project can benefit from as many sources as possible. Fortunately, Reddit itself aggregates various sources on a large scale. It is relevant to see how misinformation spreads from and to multiple sites to analyze how misinformation spreads. Since Reddit uses links, it would be possible to link tweets to posts and vice versa across those two

[1] PRAW: The Python Reddit API Wrapper https://praw.readthedocs.io.

data sources and many more in the future. There is also the hypothesis that some fake news stories or ideas might originate on Reddit. Furthermore, many popular fake news stories will land on Reddit eventually.

2.3 Possible data sources

2.3.1 Official Reddit API

Reddit provides an extensive Rest API that allows developers to integrate third-party applications with Reddit. Many pages on Reddit can also not only be requested as an HTML page but also as JSON, XML, or RSS file, by adding ".json", ".xml" or ".rss" to the URL. If you wanted to view the platforms newest posts from www.reddit.com/r/all/new/ in JSON format, the modified URL would be www.reddit.com/r/all/new/.json.

Reddit has an API rate limitation of 60 requests per minute. How many requests your application has remaining and when the time window resets is provided via the "X-Ratelimit-Remaining" and "X-Ratelimit-Reset" HTTP headers. API clients applications must also be authenticated using OAuth2 and offer a unique User-Agent string. By requesting multiple resources at a time, it is possible to increase the queried data by a certain amount without circumventing the request limit. For example, instead of requesting each submission at a time, we could request the 100 newest submissions by increasing the limit to 100 www.reddit.com/r/all/new/.json?limit=100 [37]. But there is still a limit of how much data can be requested. In this example, 100 results are the maximum number of submissions the API returns.

Third-party clients for the Reddit API exist for various programming languages [36]. The implementations and features vary from project to project. PRAW for example, provides functionality to query the API but also more intelligent features such as submission streams and automatic rate limit handling.

The Reddit API provides all the desired data, but scraping all of Reddit while respecting their limits does not seem to be feasible. Also, aggregating and storing the results would undoubtedly require substantial resources.

2.3.2 The Pushshift Reddit Dataset

Pushshift is a five-year-old platform that collects Reddit data and exposes it through an API. Compared to the Reddit API, the Pushshift API allows researchers to query historic Reddit data more easily, including full-text search against submissions and comments. They also provide a higher API limits than Reddit [4]. There are three queryable API endpoints to obtain information about subreddits, submissions, and comments:

- `/reddit/subreddit/search`
- `/reddit/submission/search`
- `/reddit/comment/search`

Dataset	Compressed	Extracted	Ratio
Submissions August 2019	6507 MB	68490 MB	1:10.53
Comments August 2019	16789 MB	186005 MB	1:11.7

Table 2.1: Size comparison between the compressed and extracted Reddit submissions and comments August 2019.

Dataset	Compressed	Extracted
Submissions	200.03 GB	2.11 TB
Comments	653.96 GB	7.25 TB
Total	854.04 GB	9.35 TB

Table 2.2: Dataset size estimation using the compressed sizes and the compression ratios from the August 2019 sample in Table 2.1.

The API provides extensive functionality to filter and search based on the content and simply restrict the result ranges by ID or date. Because Reddit's IDs are sequential, this is a simple way to query, e.g., missing comment ranges.

Compressed monthly archives. In addition to the searchable API, the Pushshift Reddit dataset also provides its data as monthly dumps. The dumps are available as compressed Newline Delimited JSON (NDJSON) files, where each line represents an object (e.g., a single comment or an individual submission). The archives are available under files.pushshift.io/reddit. The compressed size is around 200 GB for all the submissions and 654 GB for the comments from June 2005 until September 2019. To estimate the uncompressed size of the whole dataset, samples from the submissions and comments were taken. Table 2.1 shows the sample's compressed and uncompressed sizes as well as the compression ratio. To estimate the extracted size of all the dumps together, we took the compression ratios from the sample above and applied them to summed up file sizes in Table 2.2.

This method is far from precise because it does not take the different compression algorithms, content types, and file sizes into account, but it provides a solid basis to understand the amount of data that we have to deal with.

The advantage of these archives is that they are easy to obtain since no rate limit or login is required. They are files, so we can store the archives and read them by a magnitude faster than the Rest API. The disadvantages are that the dumps are only created when the Pushshift team triggers the export. Therefore the data available in the dumps can lag several months behind. The Pushshift dumps also do not provide time-series information – meaning they only offer one snapshot for every submission or comment at the time that it has been crawled [35].

Type prefix	Data type
t1_	Comment
t2_	Account
t3_	Link
t4_	Message
t5_	Subreddit
t6_	Award

Table 2.3: List of all possible full name prefixes the Reddit API uses.

Pushshift Elastic Search API. Pushshift also provides access to the Elastic Search API. Cloudflare currently blocks access to the API due to misuse. It is possible to get whitelisted upon request [34].

2.3.3 Pusher real-time Reddit API

Pusher provides a commercial real-time API that lets you subscribe to a subreddit and get notifications for new posts. In theory, this could provide the data we need, but it does not seem practical to maintain a subscription to each of the over 130.000 subreddits [33].

2.4 Reddit data structure

Every object in Reddit's API is a *thing*. A *thing* is a universal base class that has an ID, a *kind* that specifies the underlying data type and a full name. A full name combines the *kind* of a *thing* (e.g. a comment) and the unique ID to generate a compact, globally unique object ID on Reddit. Full names have the scheme `type_id`, for example for a comment `t1_1ac81d`. Table 2.3 lists all the possible full name prefixes and their corresponding data types [39].

Reddit IDs are regular ascending numeric IDs that are Base36 encoded. The created interface simply provides a unified way to determine when an entity has been created. Moreover, the votable interface provides commonly used properties for voting. The subreddit entity holds the details about a subreddit. Most importantly, its unique URL `/r/subredditname`. A link is the data type for a post. Based on whether the self text or link field is filled, the post is a self-post or a link. The comment data type is associated with a link via the `link_id`. Suppose the comment is a reply to a parent post, then the `parent_id` points to the comment in response. Therefore the `parent_id` allows hierarchical comments. Figure 2.1 shows a simplified class diagram of Reddit's object model.

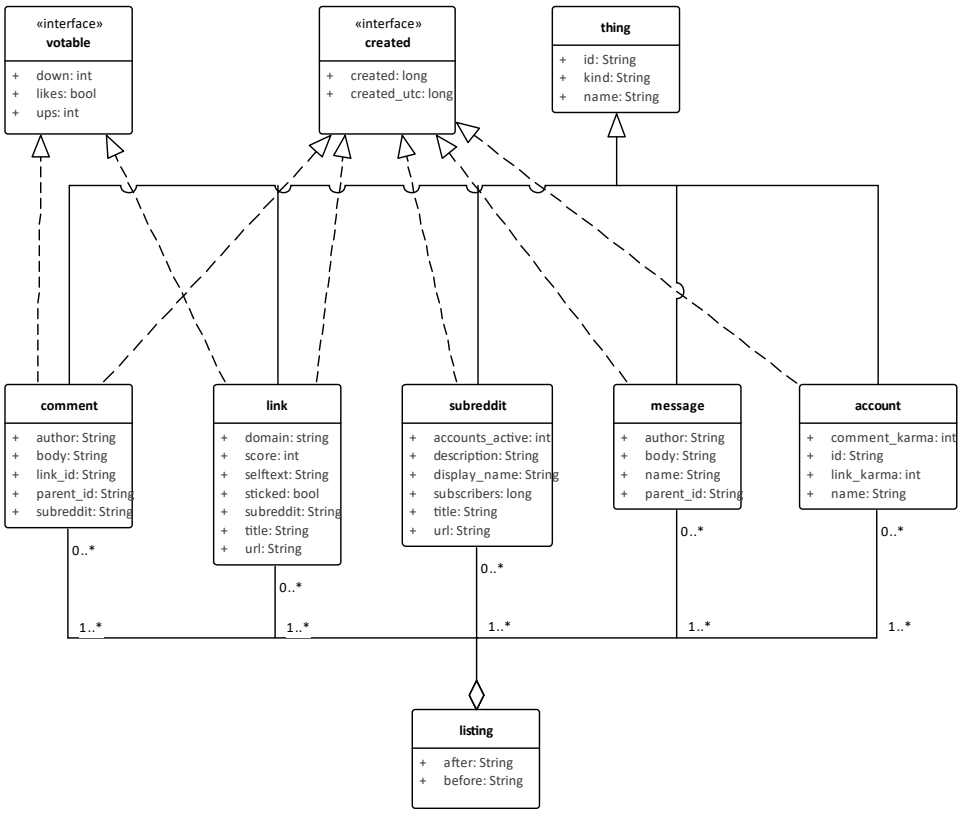


Figure 2.1: Simplified class diagram of Reddit's object model used for the API. Only the relevant classes and properties are displayed.

2.4.1 Pushshift data structure

The dataset is split up into monthly dumps.

Submissions are stored as a flat list (one submission object per line). The relationships to the author and subreddit entities can be generated by matching the unique author name and the subreddit ID. If a submission is a link, there might be additional metadata within the object (e.g., thumbnails or HTML code to embed a youtube video).

The dataset organizes comments in a flat list (one comment object per line); there is no hierarchy within the dataset. The recursive comment structure has to be recreated by matching the ID with other comments parent IDs as described in Chapter 2.4 as well with their subreddits. Pushshift provides a single file containing all the users, including metadata, and another one for all the subreddits with metadata. All the entities in the dataset represent the state at the time they were scraped by the Pushshift ingest script. Therefore each object contains a `retrieved_on` field – a Unix timestamp with the time and date it was retrieved. The relationships between the different entities must be recreated during processing or a database import to get a relational data structure.

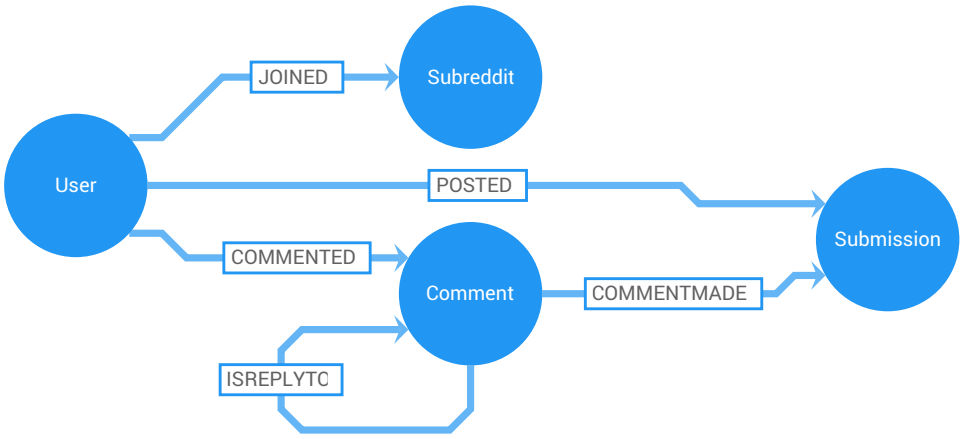


Figure 2.2: Scheme of the proposed exemplary graph, illustrating the different possible relationships between users, submissions, subreddits and comments.

2.4.2 Exemplary graph structure

Figure 2.2 depicts an exemplary and partial graph scheme that visualizes relationships between the most important entities. The graph model describes which users JOINED which subreddit, which User POSTED a submission, and which user posted a comment. It also associates in which subreddit a submission or comment was posted and to which external domain they link. This graph structure can be modified and extended based on the needs of a specific experiment. Figure 2.3 shows a sample of what evaluations we could do with the model. The red dots represent subreddits, and the blue dots the users that posted into the subreddit. The graph illustrates that the users in the central cluster were active in both subreddits.

2.5 Actor Model

To support the current development workflows of developers writing for concurrent and distributed systems, the Actor Model, defined in 1973 in a paper by Carl Hewitt, provides a higher abstraction level that lets developers avoid complicated thread management and working with locks to write concurrent and parallel systems. Ericsson has used the model to build highly concurrent and reliable telecom systems and has benefited immensely.

In computer science terms, actors are computational entities that encapsulate state and behavior and can additionally do the following units of work, in no expected sequence, in response to the messages they receive [23]:

- send a finite number of messages to other actors

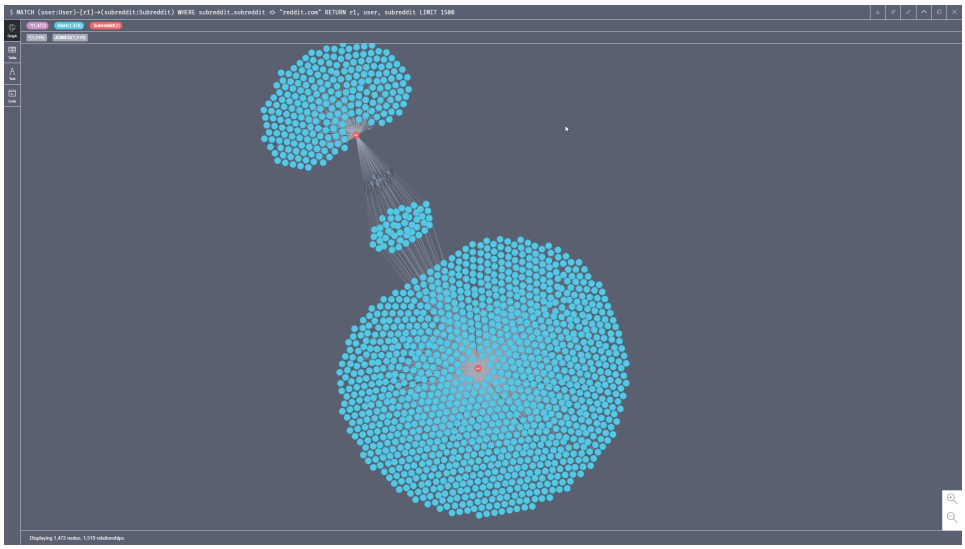


Figure 2.3: Sample graph displaying users (blue) that posted in subreddits (red). The cluster in the middle is composed of users that posted in both subreddits.

- create a finite number of new actors
- select behavior for next time a message is received

The model enables asynchronous communication by allowing each recipient of a message to be identified by a "mailing address," which can only be obtained by the sender actor if it received it in a message or was the actor that spawned the recipient. The actor system makes sure that each actor processes a single message at a time. When the actor processes a message, it can modify its state, switch to a new behavior for the following message or send a message to another actor. Figure 2.4 illustrates this behavior where the actors send messages to a recipient actor from whom they have an address with no restriction on message arrival order.

In summary, the actor model is characterized by concurrency of computation within and among actors, dynamic creation of actors, actor addresses in messages, and interaction only through direct asynchronous messages [18] [17].

2.6 Akka Streams

Most of the popular services on the web require the streaming of data. Typical examples are downloading Netflix movies or uploading youtube content. Thus streaming data becomes necessary as most datasets are too large to be processed as a whole. In order to spread these computations, the datasets are broken down, fed sequentially, and processed, as streams, through CPUs [22].

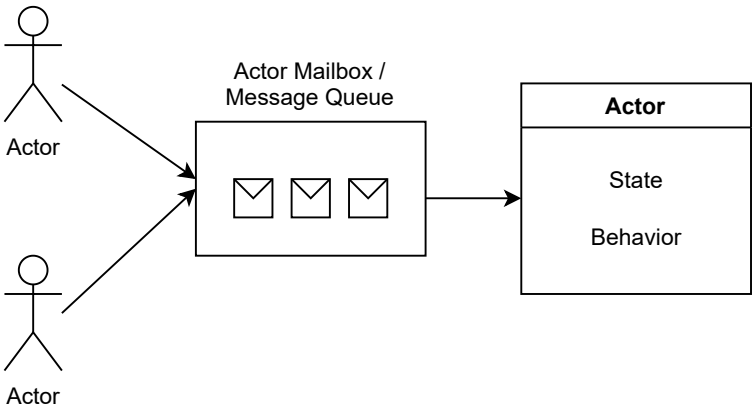


Figure 2.4: Illustration of an actor with a state and behaviour and the message queue.

Akka streams API offers an intuitive and secure way to create stream processing setups that can efficiently execute and limit the usage of systems resources. The API uses a feature called "back-pressure", a buffering technique that slows down the producers if the consumers cannot keep up. Back-pressure is a core characteristic of the "Reactive streams" initiative, a standard for asynchronous stream processing with non-blocking back pressure [2].

Akka streams API and Reactive streams. Akka Streams API is decoupled from the Reactive Streams interface specifications. Reactive streams primary purpose is to provide interfaces that allow different streaming implementations to interoperate and mechanisms to move data across asynchronous boundaries without losses, buffering, or resource exhaustion [22]. In contrast, Akka Streams API provides an end-user API that allows the transformation of data streams by implementing the above-mentioned Reactive Streams interfaces.

2.7 JGraphT

Over the last few decades, a wealth of large-scale societal, economic network data has been collected and been available to different actors. At the same time, the emergence of graph-theoretical problems in the fields of network security, computational biology, chemistry, logistics, among others, has been observed. Thus, there has been an increased interest in mathematical graph analytical solutions that efficiently allow modeling, analysis, and querying of large graphs. JgraphtT is a programming library that does just that. The library is written in Java and consists of highly efficient and generic graph data structures and algorithms implemented to provide stability and performance while working with large-scale graphs.

[2]Reactive Streams https://www.reactive-streams.org/

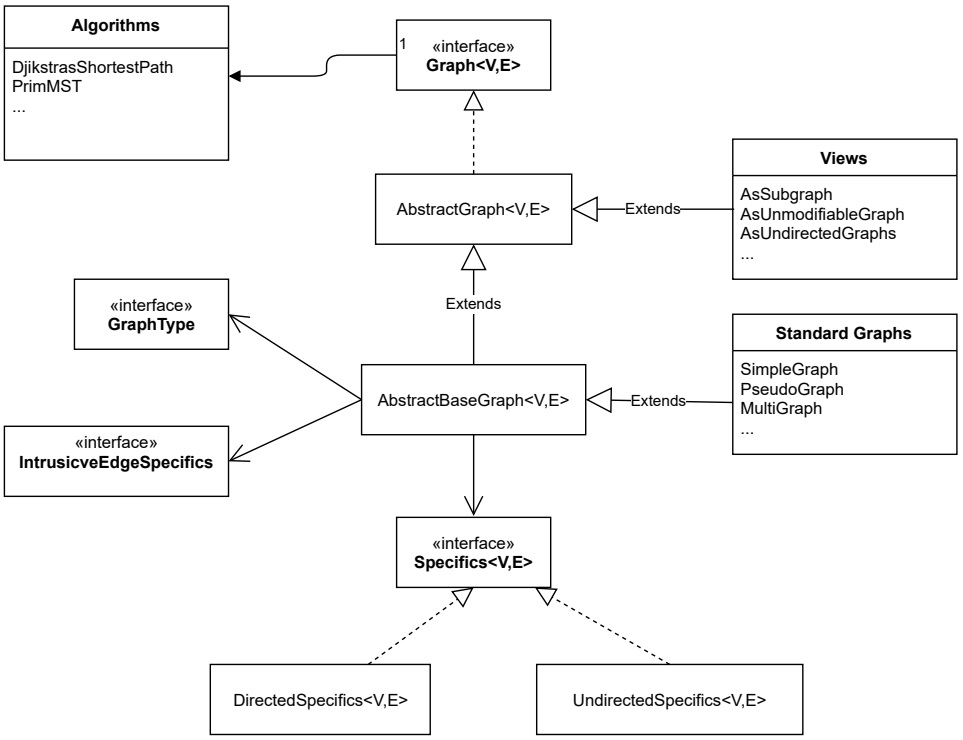


Figure 2.5: JGraphT class diagram that shows the core structure of the library [26].

One of JgraphT's distinguishing features is that it allows users to use custom objects as vertices and edges of a graph. The generic design and versatility it provides make it easy to represent standard real-world networks, such as social networks. Figure 2.5 shows the overall design of the JgraphT library. The `Graph<V, E>` interface is central to the library and provides the necessary operations for construction and accessing the elements of the graph. The interface receives two generic parameters `<V>` and `<E>`, which can be any Java object, used as the vertices and edges of the graph. A simple use case is "CampingSite" objects, where each object contains information about the capacity and amenities of a particular spot, used as vertices of a graph. Similarly, creating "Trail" objects containing information about the route between two campsites, such as trail type, length, trail condition, and other applicable properties, adds to the versatility of the resulting graph. The library's ability to model any combination of objects as vertices and edges allows users to express any relationship in a graph, increasing the versatility and making its application endless.

Furthermore, the library is designed such that all interactions with the graph happen only through the `Graph<V, E>` interface, thus providing a standard interface. All predefined classes implement the interface, and the graph algorithms included in JgraphT expect an instance of `Graph` to work. Commonly used graph types such as simple, multi,

and pseudographs, where each graph can be undirected or directed and weighted or unweighted, are included in the library for quick setup and use. JgraphT also allows users to construct graphs using builder pattern functionality, by which the library automatically chooses the most suitable graph implementation from the user's requirements.

Moreover, JgraphT provides built-in algorithms that can be run on the graph. The library contains algorithms for various problems such as connectivity, least common ancestor, cycles, shortest paths, node centrality, spanning trees and spanners, recognizing graphs, matchings, cuts and flows, isomorphism, coloring, cliques, vertex cover, and tours. Furthermore, it comes with multiple graph generators, has the capability to import and export graphs.

Thus, JgraphT is used in several commercial and scientific projects, one notable example being Apache Cassandra [**michail_jgraphtx2014java_2020**].

2.8 Gephi

Gephi is an open-source software developed to provide interactive graph exploration and network analysis capabilities to its users. These capabilities facilitate the perceptual abilities of its users to identify peculiar network features and improve the overall exploratory process by providing a comprehensive visual and tuneable toolbox. In order to achieve these goals, the maintainers of Gephi, therefore, developed the software in separate, interoperable modules, with extensibility in mind. Furthermore, the software has custom plugin support, allowing anyone to write custom filters and layouts.

The user interface (UI) is designed around the real-time graph visualization module while the exploration modules such as the layout algorithms, data-filtering, annotation, clustering, and statistics surround it. The layout took inspiration from editors like Adobe Photoshop, making it familiar to new users, as illustrated in Figure 2.6.

The visualization module in Gephi uses a custom 3D rendering engine to render networks in real-time and provides real-time data-filtering, layout algorithms, clustering, and more. The 3D renderer uses the graphics card to render the graphs, allowing the CPU to handle other computational tasks such as layout algorithms and handling multiple Gephi projects simultaneously. In addition, Gephi also has extensive functionality for exporting graphs in different formats and customization of how the nodes, edges, and labels should be colored and sized.

Finally, Gephi's support for handling dynamic networks, where the nodes and edges appear and disappear through time or change properties,

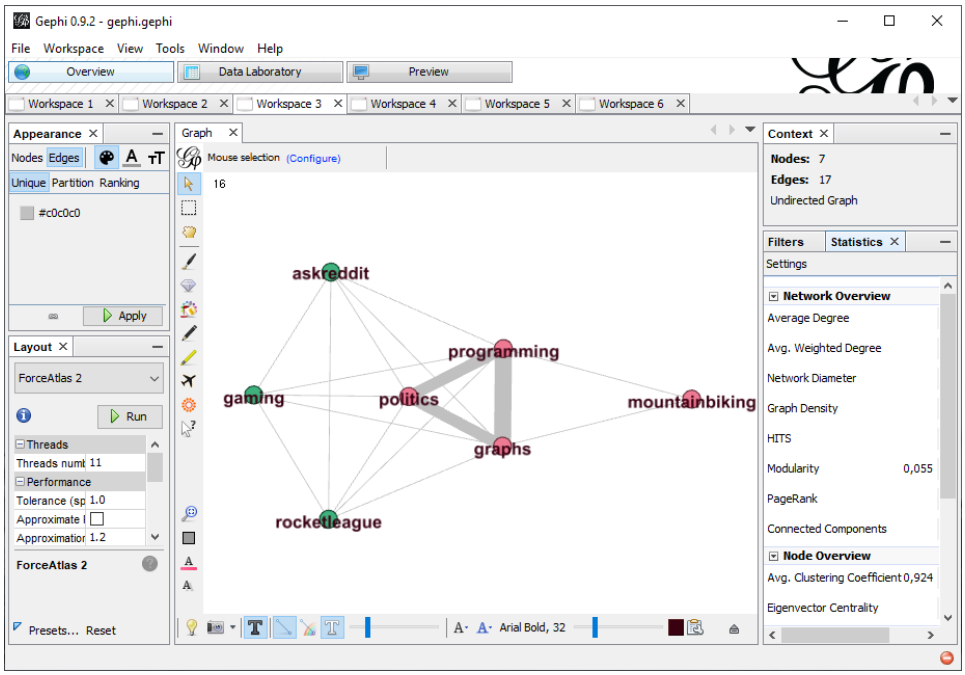


Figure 2.6: Gephi user interface. In the center of the UI is the graph. Currently, it shows a sample subreddit graph, where the nodes are colored based on their modularity class. The toolbar on the left holds the configurations for the graph appearance and the layout algorithms. The right toolbar shows the graph context and the panels to filter the graph and execute statistics.

is an essential and valuable feature. This feature allows network propagation, where new actors and relationships appear, to be visualized or, in the case of a diffusion model, see how a rumor spreads across time [3].

2.9 Gradle

Modern-day software builds requirements exceed the much simpler compiling and packaging requirements of the past. Today's projects require tools that incorporate multiple languages, various software stacks, and the capacity to apply a range of testing strategies, and at the same time, support early and frequent delivery of code into test and production environments [28].

Gradle is JVM native build tool that has these capabilities and resembles established tools like Maven and Ant. Gradle follows a build-by-convention approach and allows declaratively modeling the problem domain using a domain-specific language (DSL) implemented in Groovy. Gradle is Java Virtual Machine (JVM) native, allowing one to write custom build logic in either Java or Groovy. Gradle also includes a highly configurable dependency management tool, allowing artifacts downloads

from remote repositories. The dependency management is not restricted to external sources; it supports the definition and organization of multi-project build and modeling dependencies between projects [14].

Google uses Gradle as Android's build tool since its design and model allow extensibility and support for native C/C++ development and expansion to other ecosystems. Gradle also performs twice as fast as Maven in most build scenarios and has received consistent support from IDEs, which has greatly improved usability and adaption [14].

2.10 sbt

sbt is an open-source build tool created for Scala and Java projects. It is the build tool of choice for over 90 % of Scala developers as of 2019 [41]. Some of the main features of sbt are native support for compiling Scala code, continuous compilation, testing, and deployment, support for mixed Java/Scala projects, dependency management, and others. Other Scala-specific abilities that sbt has, which are among the main reasons for its wide adoption, is that sbt allows users to cross-build their project against multiple Scala versions, incremental compilation, and the interactive shell. Similar to Gradle, sbt has support for DSL used by writing build description files in Scala [42].

2.11 Named Pipes

Standard pipes are a mechanism for Inter-process communication (IPC). The output of a process/program can be "piped" as an input of another to chain the processing; these pipes live temporarily in the kernel [23].

FIFO stands for first-in-first-out and is a queuing technique where the oldest queued item or job is processed first. Unix named pipes, also known as FIFO, use this concept to create files in secondary storage. Named pipes contents reside inside the main memory and are not saved to the disk. For the contents of a named pipe to be passed/processed, both the input and output end needs to be in use. This capability allows named pipes to be used in ways that standard files would not be efficient [16]. Named pipes also require surprisingly little resources for writing to them.

2.12 Experimental Infrastructure for Exploration of Exascale Computing (eX^3)

The exponential and ever-increasing need for computing power for scientific and commercial interests have sparked research and development of technologies capable of performing billion billion (10^{18}), also know as exascale, floating-point operations per second. The high-performance

computing (HPC) machines of the future will consist of sophisticated, heterogeneous, tailored processing nodes of large scale, with deep memory hierarchies and complex infrastructural topologies each targeted, for example, a specific set of algorithms or machine learning techniques [27].

One such example is the eX³ project, a collaborative effort between Simula Research Laboratory and their partners to further the national resource and prepare researchers for exascale computing in Norway. The ex3 infrastructure is not an exascale computer but allows researchers to experiment with upcoming hardware and software-based HPC technologies [27].

Users of eX³ can choose to develop for and use different processing nodes, each consisting of processors from various manufacturers with either x86 or ARM architectures, and set them up with primary and secondary storages in unique configurations. A specific example from the eX³ cluster is the AMD EPYC nodes configured as dual processors with 2 TB DDR4 main memory and 2.8TB local NVMe scratch storage. Other nodes, namely the Intel Xeon and ARM-based Cavium Thunder X2, are similarly configured with varying local memory solutions. In addition to local secondary storage hardware, all nodes in the cluster have access to a 500TB NetApp Enterprise hybrid, i.e., combined HDDs and SSDs, storage unit set up through the BeeGFS parallel file system. This storage is intended as a temporary place to collect results from large-scale experiments and simulations [40].

Numerous ongoing projects using the eX³ cluster and multiple journals and papers published amplify the usefulness and benefit to future research of the project [32].

2.13 Related work

Baumgartner et al. [4] describe in their article "The Pushshift Reddit Dataset" how they overcame technical barriers to collect and archive social media data from Reddit and what Data and APIs they provide for researchers.

Holec and Mack [19] discuss "How new scholars navigate research on digital platforms while remaining grounded in traditional disciplines" based on two cases on the Reddit platform.

Tsugawa and Niida [47] conclude that several social network features have significant effects on the survival of communities. The social network features they investigated do not have a substantial influence on the growth rate.

Singer et al. [45] show how Reddit has grown and changed over time, based on an earlier version of the Pushshift Reddit dataset.

Part II

The project

Chapter 3

Approach

The approach discusses the concept of how to build a graph between subreddits on Reddit.

Constructing graphs with the size of Reddit is a challenge - The most obvious reason why is because of the sheer size of the dataset (over 900 GB compressed). There are constraints in the available compute resources like CPUs, Memory, and Storage. Then there is the data structure. Reddit subreddits are independent topics. The dataset does not directly indicate which subreddits are in relation to each other. A method of identifying relationships between subreddits is required. However, just building a graph would result in a huge graph, which most likely hides the information we actually want to look at. So a method to weigh and filter relationships between subreddits needs to be developed. A fundamental challenge of building a comprehensive graph is to find a edge weighting function that promises comprehensive results.

In order to tackle this task, the problems can be broken down into manageable size steps. The results of each step can be directly piped into one run, but to cut down on compute time, they also shall be saved to reuse.

Overall the processing pipeline can be divided into three stages and programs, as illustrated in Figure 3.1. One to read and transform the dataset and make it reusable for graph building and other projects. Another one to create the graph, work with it—furthermore, the last one for analyzing the graph and experiments.

In the first stage , a method to efficiently read the dataset needs to be developed in order then to determine the top n subreddits. Knowing the top n subreddit makes it possible to create subsets of the dataset containing those subreddits. This will limit the number of subreddits and, therefore, reduce the complexity and memory usage. This is ok because subreddits with just a few contributors will not tell much about the overall structure of Reddit. The focus is set on the top 10k subreddits. The CSV results will be written to named pipes or zstandard compressed archives. To perform experiments over time, yearly subsets will be created using the same steps but filtered by each year.

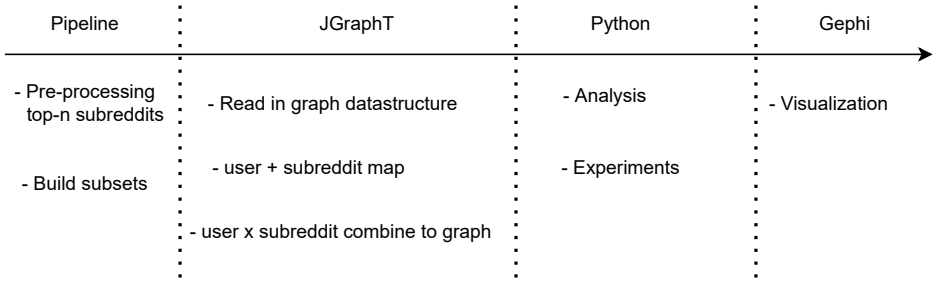


Figure 3.1: Illustrates the necessary steps to generate the graph from the dataset. The fist step is to use the Reddit Dataset Stream Pipeline on the left for reading, prepossessing and filtering. The second step to build the graphs with JGraphT and calculate the scores. With those graphs we can perform further experiments and build visualizations in step 3.

In the second stage , the actual graph data structure is built from the "subreddit, user" CSV. The first step is to read the CSV and create a distinct list of subreddits for each user. We can then combine all possible distinct combinations of subreddits to create an edge between them from that list. Adding each of these edges to the graph distinctively creates the simplest form of the graph structure. It depicts only which subreddits are in relation to each other. The next step is to extend this implementation to create a simple edge weight by counting users that posted in both subreddits. Furthermore, precalculate common scores like degree, degree-degree, and the local clustering coefficient in a reasonable time. From there, the weight needs to be improved, to take neighbors into account. Furthermore, functionality to export and import graphs in a universal format needs to be implemented.

The third stage focuses on working with the graph. With the edge lists, experiments such as ovserving distributions can be performed. Furthermore, a way to visualize the graph is another challenge. To visualize the graph visually and performant, the data structure has to be reduced from a highly connected graph to the essential edges only.

3.1 Understanding the Pushshift Reddit Dataset

Pushshift is a data collection service for Reddit and other social media sites that makes data available to researchers. They provide an API to query the dataset as well as a downloadable archive used in this thesis.

The dataset is split up into two entities/subsets - submissions and comments. Each of these datasets contains

- NDJSON (one JSON object per line)
- Compressed using xz, BZ2, ZSTD (Google names)

- Dataset is split into monthly archives, on submission, and one comment file per month.
- Only using data from 2005-2019.

Submissions The submissions dataset contains one JSON object per submission/post that has been made on Reddit. Table 3.1 lists the fields the submissions dataset provides. The most interesting fields to create the graph are the subreddit and the author. These are the only fields required to identify which user posted in which subreddit since the author field contains a unique user name and the subreddit names are unique.

Field	Description
id	The submission's identifier, e.g., "5lcgjh" (String).
url	The URL that the submission is posting. This is the same with the permalinl
permalink	Relative URL of the permanent link that points to this specific submission, e
author	The account name of the poster, e.g., "example username" (String).
created_utc	UNIX timestamp referring to the time of the submission's creation, e.g., 148
subreddit	Name of the subreddit that the submission is posted. Note that it excludes t
subreddit_id	The identifier of the subreddit, e.g., "t5 2qh1i" (String).
selftext	The text that is associated with the submission (String).
title	The title that is associated with the submission, e.g., "What did you think of
num_comments	The number of comments associated with this submission, e.g., 7 (Integer).
score	The score that the submission has accumulated. The score is the number of u
is_self	Flag that indicates whether the submission is a self post, e.g., true (Boolean)
over_18	Flag that indicates whether the submission is Not-Safe-For-Work, e.g., false
distinguished	Flag to determine whether the submission is posted by moderators or admi
edited	Indicates whether the submission has been edited. Either a number indicati
domain	The domain of the submission, e.g., self.AskReddit (String).
stickied	Flag indicating whether the submission is set as sticky in the subreddit, e.g.,
quarantine	Flag indicating whether the community is quarantine, e.g., false (Boolean).
hidden_score	Flag indicating if the submission's score is hidden, e.g., false (Boolean).
retrieved_on	UNIX timestamp referring to the time we crawled the submission, e.g., 1483
author_flair_css_class	The CSS class of the author's flair. This field is specific to subreddit (String).
author_flair_text	The text of the author's flair. This field is specific to subreddit (String).

Table 3.1: Pushshift submissions dataset description. Copied from the paper The Pushshift Reddit Dataset [4].

Comments The comments object contains one JSON object per comment that has been made on Reddit. It can be a comment directly to a submission or a comment in reply to another comment. Table 3.2 lists the fields the comments dataset provides. The most interesting fields to create the graph are the subreddit and the author. As in the submissions dataset, those two fields contain unique names.

[4]

Field	Description
id	The comment's identifier, e.g., "dbumnq8" (String).
author	The account name of the poster, e.g., "example username" (String).
link_id	Identifier of the submission that this comment is in, e.g., "t3 5l954r" (Str
parent_id	Identifier of the parent of this comment, might be the identifier of the su
created_utc	UNIX timestamp that refers to the time of the submission's creation, e.g
subreddit	Name of the subreddit that the comment is posted. Note that it excludes
subreddit_id	The identifier of the subreddit where the comment is posted, e.g., "t5 2q
body	The comment's text, e.g., "This is an example comment" (String).
score	The score of the comment. The score is the number of upvotes minus th
that	Reddit fuzzes the real score to prevent spam bots. E.g., 5 (Integer).
distinguished	Flag to determine whether the comment is made by the moderators or a
edited	Flag indicating if the comment has been edited. Either the UNIX timesta
stickied	Flag indicating whether the submission is set as sticky in the subreddit,
retrieved_on	UNIX timestamp that refers to the time that we crawled the comment, e
gilded	The number of times this comment received Reddit gold, e.g., 0 (Integer
controversiality	Number that indicates whether the comment is controversial, e.g., 0 (Int
author_flair_css_class	The CSS class of the author's flair. This field is specific to subreddit (Stri
author_flair_text	The text of the author's flair. This field is specific to subreddit (String).

Table 3.2: Pushshift comments dataset description. Copied from the paper The Pushshift Reddit Dataset [4].

Size The compressed size of the dataset is currently around 900 GB including parts of 2020. We are using using data from 2005-2019.

3.2 Reddit Dataset Stream Pipeline

The Reddit Dataset Stream Pipeline is proposed as a program to read, filter and transform the Reddit dataset in parallel. The shall be able to provide a stream for other programs that use the data provided. The pipelin shall be created with keeping the possiblity of live streaming data in mind.

Challages the dataset brings. Dataset is large, just unpacking it single-threaded takes time, and reading it multiple times also takes time. Storing it uncompressed takes space, and in some cases, longer to read depending on whether the Bottleneck lies with IO or CPU time.

We therfore consider the following features as neccessary:

- Process the dataset in parallel
- Create intermediate results
- Make sure to only read the dataset it once per iteration

Idea The idea is to create a program that can perform the reading and provide access to the dataset via UNIX named pipes. Also, it sthall restrict the information flow to a minimum, by A: only providing the fields

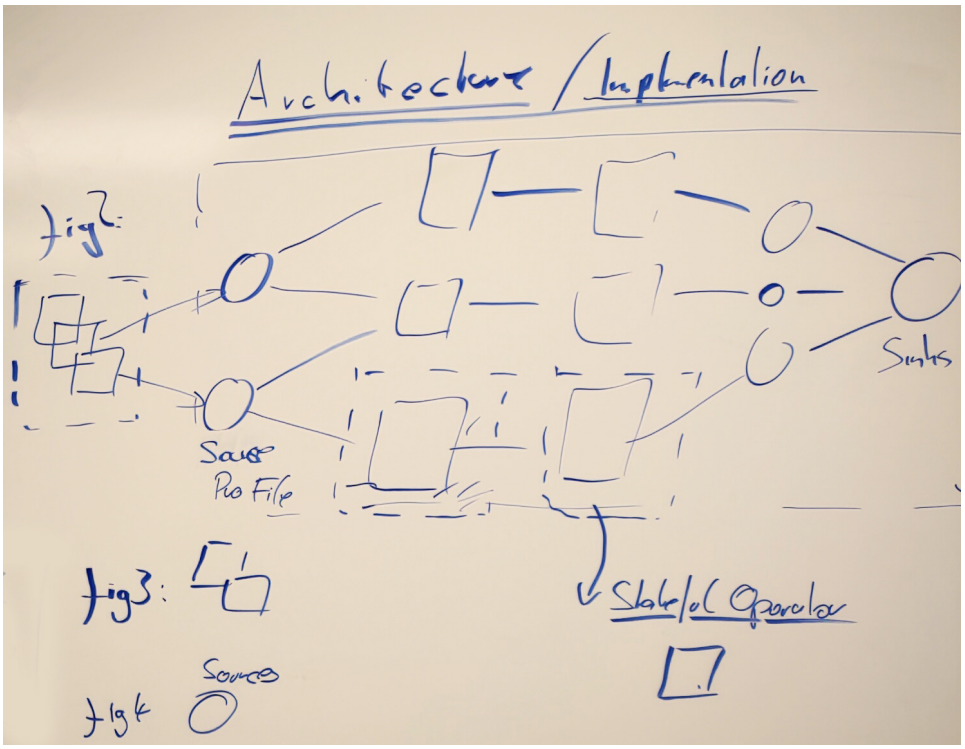


Figure 3.2: Architecture concept overview of the Reddit dataset stream pipeline.

required. B: Filtering deleted users for the graph creation because it is impossible to know if one post was made by the same deleted user or different deleted users. The dataset just contains the string

> [deleted]. We should C: Create a possibility only to filter the top n subreddits. To solve this we propose to create a stream processing system, so it would be possible to process live data with it in the future. We propose the use of Akka streams to utilize features such as backpressure and parallel execution and to extract the dataset as fast as needed by the client for now. The result stream shall be made available for other programs by using files or named pipes.

> **Concept** The program shall read the files in parallel and merge the results. The order does not matter four our cases, and therfore there is no need to preserve it. We suggest to read the comments before the submissions because the files are larger, and we estimate to get a better CPU core usage by doing so.
>
> The implementation shall filters lines that contain a corrupted JSON object from the result stream. As well as lines that only contain null bytes.

Proposed features Commandline interface with a few settings:

- dynamically specifies the location of the dataset
- Filter for files that contain a specific string. This is used to, e.g., select just the dataset for the year 2012 by passing the `--filter 2012` to the program.
- Exclude files that contain a specific string. To, e.g., exclude 2020, the parameter `--exclude 2020` can be passed to the program.
- Compress option to compress the output using ZSTD. This can be helpful to reduce the amount of data written to disk for larger datasets.

3.2.1 Statistics mode

The statistics module shall be used to gather basic metrics about the whole dataset or subsets of it. Currently, two different experiments have been implemented.

Number of user contributions in subreddit experiments. We propose to create the User contributions in subreddit experiments. It shall be a experiment to count the number of contributions users made in subreddits. A contribution is a post or comment. To parallelize reading and counting the contributions, the program shall maintain an intermediate contribution per subreddit count for each file in a $hashmap[subreddit, count]$. Once the file is completely read, the hash map is passed to a merge step that aggregates all the intermediate counts into one big hashmap. Once all files are read, this hashmap is then converted to CSV and written to disk.

Number of users per subreddit. An Experiment to count the users that made at least one contribution in a subreddit. A contribution is a post or comment. Like in the contributions per subreddit, to parallelize the counting of the users, the program shall maintain an intermediate subreddit hashmap per file. The hashmap holds an inner hashmap per subreddit used to list the users that contributed to the subreddit and the number of contributions for each user. $hashmap[subreddit, hashmap[user, count]]$ Once a file is completely read, the hash map is converted to a simpler map that contains the subreddit and the number of distinct users and passed to the merge step. $hashmap[subreddit, count]$ The merge step then aggregates all the intermediate counts into one big

hashmap. Once all files are read, this hashmap is then converted to CSV and written to disk.

The number of users per subreddit implementation shall filter-filters lines where the author is not set or has been deleted. Since posts of deleted users are not distinguishable, because the username is "deleted" they are ignored. If they were not ignored, the count of every subreddit with $1 - n$ deleted comments or submissions would be increased by one. This means small subreddits with only one deleted post would get a proportionally higher count than large subreddits with many deleted posts.

3.2.2 Passtrough mode

This mode shall read the dataset files and provide them as a single output stream, therefore passes them through. The main use case is to make the dataset easier readable by third-party tools. Therefore it can break down the large dataset to a single CSV stream without writing it to disk and utilizing multithreading for extraction. Initially, it was intended to use it via Unix Named Pipes and read the data from another program, e.g., Python and Pandas. Nevertheless, it shall also be possible to write the file to disk - e.g., with the compression option enabled to save disk space.

It shall hande two streams if requested. One for comments one for submissions. Possible to return two streams at the same time. The number of parallel read files is split between the two streams.

There shall be multiple multiple ways the result stream can be returned.

1. User in Subreddit CSV with he flag `--only-user-in-sr`. A simple line with "subreddit,author" is written to the output for every comment and submission. That format is the format used to create the graphs because submissions and comments are treated equally.
2. Submission and Comment CSV. Default - without any flags. The fields "subreddit,id,author,title" are passed to the CSV stream for the submissions. The comments stream provides the fields "subreddit,id,author,body" . The fields can be extended when necessary by extending the data structures in the scala code.
3. Original JSON. Using the flag `--keep-original-json` the original JSON format is passed to the output stream, and

the program mostly acts as a filter and simplified way to read the files as one single file.

The implementation needs to filters lines where the author is not set or has been deleted. The reason deleted authors shall be filtered is that they would falsify the resulting graphs. Since the graphs will be built on "which user contributed in which subreddit", all the posts for deleted users would be seen as one user with the name "[deleted]", which is connected to many subreddits. That could create edges between subreddits that would otherwise have no connection at all. The impact for smaller subreddits would be higher because, with one deleted post, they suddenly would be connected to many more subreddits.

3.2.3 Streams Flow Implementation

For each mode we propose a seperate flow, since the implementation is differen. To parallelize reading and counting the contributions, the program shall use a stateful map-concat within the flow for each file. For the pass trough this is not neccessary. Though shared logic like reading the dataset shall be used by both flows.

3.2.4 Steps to get the data ready for the graph stage

In order to build the Graph, a list for every comment and submission a user made into a subreddit is needed. For example a CSV in the form of `subreddit,author`. The goal is to create a graph from that list, considering limited RAM, CPU time, and storage. Considering these limits, we take only up to 10.000 Subreddits to build the graph and strip all unnecessary data from the result.

1. Count the number of user contributions per subreddit. We can utilize the Reddit Dataset Stream Pipeline statistics mode first to count all the users per subreddit. That run shall create a CSV file with the subreddit name and the number of users that contributed to that subreddit. To exclude the incomplete year 2020.
2. Filter the top n subreddits from the previously generated `subreddit,count` CSV file with a python script. The script can use Pandas to load the CSV into a data frame. Then use the `DataFrame.nlargest` function to find the top n subreddits with the highest user count. The names of the top n subreddits shall be then written to a new-line delimited text file.

3. We then create a subset of the dataset that only contains submissions and comments of the top n subreddits. With the passthrough mode a filter parameter `--filter-by-sr <filterlist.txt>` pointing to the newline delimited text file of subreddits to filter for, created in the previous step. To only export the fields required for the graph building (`subreddit,author`), we can supply the option `--only-user-in-sr` to the program. We exclude the incomplete year 2020 by adding the parameter `--exclude 2020`. To keep the size down, use the `--compress` parameter. It shall be possible to either process both submissions and comments in one run by supplying both the `--comments` and `--submissions` parameter. It can also make sense to split it up, for example, to run it on multiple computers in parallel. In any case the `--comment-out <file>` and `--submission-out <file>` parameters should be specified. Otherwise, the output is written into the execution directory as `comments.csv` and `submissions.csv`.

In order to build the yearly graphs, yearly subsets are required. We have to repeat the three steps per year to create those subsets per year $y = [2005 : 2019]$. The Reddit Dataset Stream Pipeline shall supports filtering for years indirectly by filtering for filenames that contain the year. To filter the subreddits for 2014, we porpose the parameter `--filter 2014`. In summary the process could look the following:

1. Find the number of user contributions for year y with the `--filter y` parameter.
2. Find the top n subreddits for year y. The export user count script includes exports for the years 2005-2019.
3. Create a subset of the dataset that only contains submissions and comments of the top n subreddits for year y.

We can repeat thet process to crete the time slice

$$\{2005 : 2019\} \times \{Top\ 5, 10, 100, 1.000, 10.000\}$$

3.3 Graph Building

The Reddit Dataset Stream Pipeline (RDSP) provides large datasets that tell us which users posted in which subreddit. In this section, we elaborate how the data provided by the RDSP is used to create the graph. We discuss the evolutionary steps that went into constructing a graph.

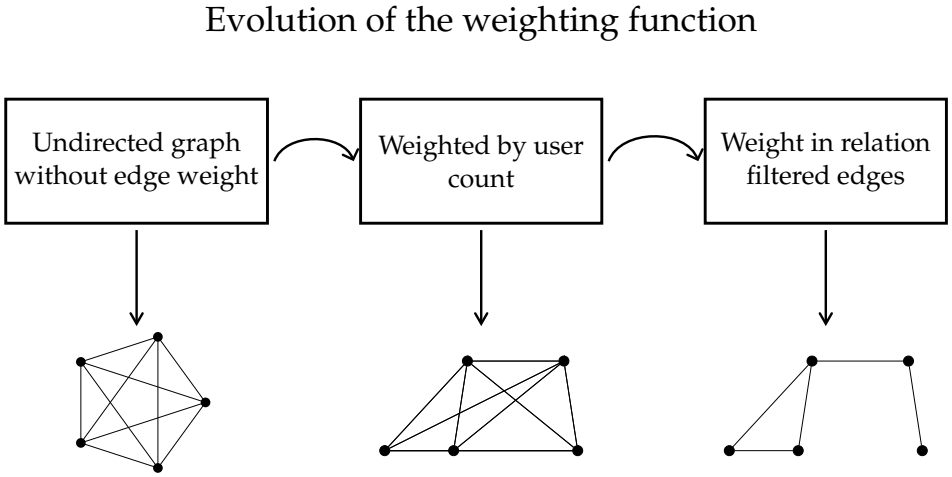


Figure 3.3: Evolution of the weighting function. The first stage illustrates the simplest form of an undirected graph subreddit graph. For every user that posted in two subreddits, an edge is created. For a situation like the top 10000 subreddits with many more users, this usually results in a fully connected graph since it is likely that at least one user posted in two subreddits. Therefore this is a first step towards building the graph. In the next step, the edges are weighted by counting the users that posted in the two subreddits an edge connects. This provides us with a simple edge weight that gives a first indication of which subreddits belong together. The final weight function considers the importance of neighbors. Once we filter out edges with low weights, we can visualize a graph that indicates which subreddits belong together.

3.3.1 Reddit Graph tool

The goal is to develop a program that can create the graphs from the user CSV stream the RDSP creates. Furthermore, the program should be able to read the graphs and then later perform experiments on them. To solve this problem, we built a Java program that does exactly that. It is leveraging the algorithms and data structures the JGraphT Java library provides. The Java project is built with Gradle, so it can be built and packaged with one command without worrying about dependencies. Moreover, the program uses various third-party libraries for compression/decompression, CSV handling, creating the command line interfaces. Since the program serves two purposes, it provides two modes that can be selected via a command-line parameter.

Create the Graph In order to build a Graph from the "user, subreddit" CSV, the program can be called with the `--mode CreateFromUserSubredditCsvAndExport` command line parameter. The program then reads the CSV, creates distinct

vertices, and builds distinct maps of subreddits each user has contributed to. From there, it creates the degrees and calculates scores and weights for the graph. After the graph creation is finished, the results can be exported as edge and vertex csv files and a dot file. The exported results include all the calculated scores.

Load the Graph When the program is called with the `--mode LoadFromVertexEdgeList` command line parameter, it is set into the load mode. This mode creates the graph in memory, but rather than reconstructing it in from the "user, subreddit" CSV, it loads the graph from the vertex and edge list which is more efficient. These lists are simple CSV files that contain the vertices, plus the precomputed parameters such as a degree or the weight. The program reads and deserializes these lines into the same data structures. First, the vertices are loaded, then the edges. The advantage of this approach is that it is faster by a magnitude. Having the vertex and edge lists in a simple format makes them reusable for importing them into other tools such as a Graph database or a Dataframe in Python with Pandas or in R.

3.3.2 Evolution of the graph creation

The Reddit Dataset Stream Pipeline provides a "user, subreddit" CSV for the top n subreddits. This section will conceptualize how to build a subreddit graph representing a landscape of subreddits in multiple iterations. In order to create the graph, the workload will be broken down into manageable smaller workloads shown in Figure 3.3. The first challenge is to create an undirected subreddit graph from the list of user engagements. The next challenge is to create a simple weighting function. The final step is to develop a more advanced weighting function and filter edges by a threshold to study the network of strong connections between subreddits. This base graph can then be used for further experiments.

Unweighted graph. The Reddit Dataset Stream Pipeline provides a list of users in subreddits. The first step towards a meaningful graph is creating an undirected unweighted graph from the list of users in subreddits in Figure 3.4. Where one user posting in any pair of subreddits results in an edge between those subreddits. Whether one or n users post in those two subreddits has no impact on the resulting graph.

fun, andreas
java, andreas
java, daniel
berlin, daniel
fun, haseeb
berlin, haseeb

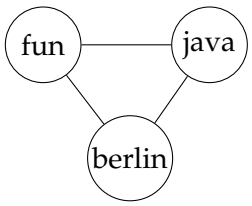


Figure 3.4: Creating a simple unweighted graph. Given the CSV list, which indicates which users contributed in which subreddit at the left, we can create a simple undirected graph. Andreas posted in the subreddits fun and java, so we can create an edge between fun and java. Daniel posted in java and berlin, so an edge between java and berlin is created. Last haseeb posted in fun and berlin, so we make an edge between fun and berlin. This results in a simple graph without any edge weights taken into account yet.

CSV	**Hashmap[Hashset]**	**Graph**
fun, andreas	andreas:[fun, java]	
java, andreas	daniel:[java,berlin]	
java, daniel	haseeb:[fun, berlin]	
berlin, daniel	john:[fun]	
fun, haseeb		
berlin, haseeb		
fun, andreas		
fun, john		

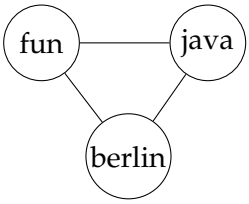


Figure 3.5: From CSV to Hashmap to Graph. todo: extend description

Since the whole graph will be built with JGraphT, we create a new graph JGraphT graph object first. We then parse the CSV file line by line. For each subreddit, we add a vertex if it does not exist yet. The current list contains one entry per post a user has made. Meaning the same subreddit could be listed n times. For the graph, a distinct list of subreddits per user is required. To create the distinct list, a `HashMap[User, HashSet[Subreddit]]` is used. If the hashmap has no entry for the current user, it will be added, and the HashSet in that slot will be created with the current subreddit. If now another line for the same user is added, we add the subreddit to the HashSet. When the whole CSV is processed, we have a distinct list of users and a distinct list of subreddits per user, which will be the basis for creating the graph. An example of such a hashmap can be seen in Figure 3.5.

	A	B	C
A	0	0	0
B	1	0	0
C	1	1	0

$\{AB, AC, BC\}$

Figure 3.6: Unique subreddit combinations per user in matrix representation.

All unique combinations of subreddits per user have to be determined to create the edges between subreddits a user was active in. The exemplary matrix representation in Figure 3.6 illustrates that if a user is in the subreddits A, B, C, all possible unique combinations would be AB, AC, BC. Formally this can be defined as:

Let be X a set consisting of n subreddits assigned to one user.

$$X = X_1, X_2, ..., X_n$$

Then we define a set M consisting of all unique subreddit pairs in X without equivalent combinations.

$$M = \{\{x, y\} | x, y \in X, x \neq y\}$$

The result is a set of unique subreddit pairs M, without combinations where subreddits would be paired with themselves, such as $(AA), (BB)$ and without inverse combinations $(AB) = (BA)$.

Programmatically this can be solved by accessing the subreddit set by index. The following two loops illustrate how to create all possible unique index combinations. Note that the start index of the inner loop is dependent on the outer loop.

```
For i = 0 to MaxIndex
    For j = i + 1 to MaxIndex
            log(i, j)

Resulting combinations:
i=0, j=1
i=0, j=2
i=1, j=2
```

For each subreddit pair, an edge is added if it does not already exist in the graph. It could already exist because another user could be associated with the same two subreddits.

There is no parallelization in this step, as exclusive write access would not be ensured.

Figure 3.7a illustrates an example of such a graph. In this almost fully connected graph, all the subreddits are close to each other. Meaning this tells us nothing other than there is some relationship between those graphs.

Weighting by user count The next logical step to create a graph with weights. The most simple edge weight is the number of users that posted at least once in both subreddits U_{ij}. To create a graph and calculate U_{ij} for each edge, the previous attempt has to be modified. The vertices and `HashMap[User, HashSet[Subreddit]]` can be created exactly as in the previous step. Previously if an edge had already been present because another user posted in both subreddits already, the new edge was discarded. In this approach adding the edges has to be handled differently.

The edge will get a property U_{ij} (numberOfUsersInBothSubreddits). We check for each subreddit pair we got from the previous step if there is already an edge present. If there is no edge present, we create a new edge between those two subreddits and set U_{ij} to one because now one user-contributed to those two subreddits. If the edge is already present, the property U_{ij} is increased by one.

Weight in relation Create a weight in relation to its neighbors. Figure 3.7c shows the graph using the ... based weight. A clearer distinction between the groups. In this case, between the programming section and the rest.

- Todo: explain weight
- Todo: read scale-free network, small-world network
- Todo: explain weight.
- Todo: explain why weight is better. (Not such a high range as Uij)
- Todo: formula weight
- Todo: explain all calculated metrics

Filter Edges by threshold After filtering the edges by weight, the resulting graph only displays connections of importance. Many quite low edge weights. Not so many high edge weights. If we filter out the unimportant low edge weights, we get a graph with meaningful edges. In this state, the graph can be

easier rendered, and neighbors should be visible more clear than in a fully connected graph.

Function W die jeden Touple auf eine natürliche Funktion abbildet - und alles auf 0 setz, was unter dem Threshold liegt.

$$f(m) : M \rightarrow \mathbb{R}$$

t the threshold.

$$W_M(m) = \begin{cases} 0, & weight(m) < t \\ weight(m), & otherwise \end{cases} \quad \forall m \in M$$

The result is an in-memory graph that can be used as-is for the experiments. Achievement: Through the Weight, the real network structure of subreddits will be visible. As an intermediate result, it will be written to disk as edge and vertex CSV lists to be loaded from there fast.

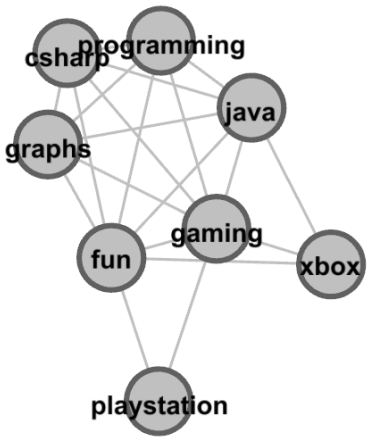
3.4 Supporting Tools

Some challenges are surrounding the development process that a set of tools can solve. One problem is that many single batch jobs need to be executed on eX³ to compute all the results. Therefore we propose the use of SLURM to schedule batch jobs on the eX³ cluster. Another problem, that the developed code needs to be deployed to eX³ before we can run it. Therefore we elaborate on the CI / Continuous Deployment (CD) system we propose as a solution.

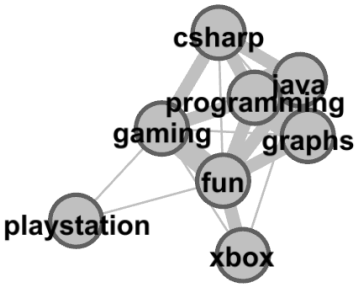
3.4.1 Scheduling with SLURM

SLURM is a cluster resource manager that allocates cluster resources, provides a framework for running and monitoring tasks on the cluster nodes. Furthermore, SLURM maintains a queue for conflicting work [50].

In order to generate all the Reddit graphs with multiple parameters over time, a cluster resource manager like SLURM is essential. Manually queueing all the tasks after each other would be tedious, and utilizing multiple nodes in a cluster provides faster results. However, given enough time, it would be possible to run all those scripts on one machine sequentially. SLURM has the capability to perform array jobs, meaning one script call can be parameterized. We use it to create the time slices by supplying the years from 2005-2019 as an array job

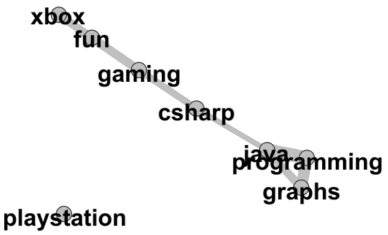


(a) Without weights



(b) User count U_{ij}

(c) Weight



(d) Removed around 1/3 low weights

Figure 3.7: The four stages of building the graph. Graph rendered in Gephi using the Force Atlas 2 layout algorithm [24] with gravity set to 1 and scaling set to 50. Graph 3.7a shows the graph without weights, which means the graph is layouted only based on the number of edges each node has. Graph 3.7b uses the user count U_{ij} as a weight, which results in a graph that gets pulled close together. Graph 3.7c depicts the same graph with the custom weight function. The groups of nodes are more distinct than in the previous graph. Graph 3.7d shows the graph with the custom weights, but around 1/3 of the edges with low edge weights are filtered due to their low importance.

parameter. Instead of creating 15 individual tasks, SLURM groups those tasks together as one array job. Nevertheless, the individual runs can still be split over multiple nodes. We also generate multiple graph graphs for the top $n = 5, 10, 100, 1000, 10000$ subreddits by the number of contributing unique users. For the yearly graphs alone, 75 individual runs are required per step of the process.

To store the dataset and the generated results, we will use the 500 TB BeeGFS storage that is mounted in most of the eX³ nodes. BeeGFS is a network filesystem as well as a parallel file system [15]. Furthermore, the results are synchronized with Google Drive to make them accessible locally.

3.4.2 CI builds, tests, and automatic updates

Dealing with such a vast dataset requires testing the codebase frequently on a larger machine than the local desktop. Therefore it has to be frequently deployed to eX³. Since there is no sbt installed to compile the scala code on the machine directly, and copying the Java Archives (JARs) manually is a tedious process, the deployment process is using CI/CD. In general, CI is a good practice that helps to keep the code in a compilable state and helps to notice bugs early in the process [13]. Therfore we propose the following CI workflow, which is also illustrated in Figure 3.8:

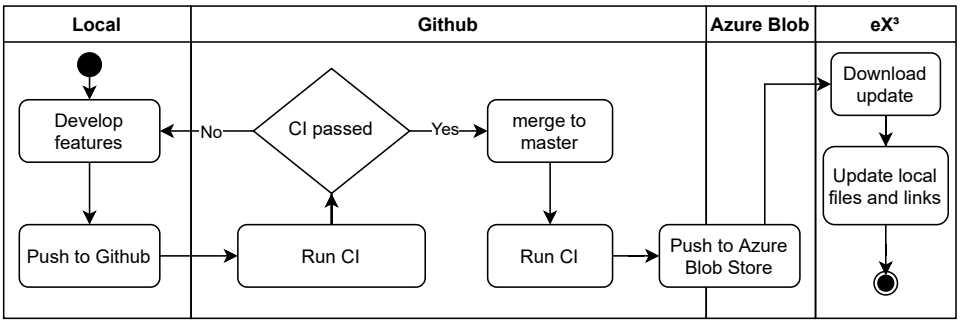


Figure 3.8: **CI stages**. The application is developed locally. Once a new feature is ready, it is then pushed into a feature branch on Github. On Github, the CI Build verifies that the code is compilable and tests pass. If the CI build passes, the code can be merged to the master branch. From there, the automatic CI Build is started again, but this time the generated package is uploaded to an Azure Blob Store. From there, it can be downloaded to eX³ by the update script, which extracts and overwrites the old version with the new one.

1. We develop features in a feature branch and commit them to the git repository. Push the changes to Github.
2. Once the feature is done, we create a pull request to merge the feature into the master branch. The CI Build automatically restores dependencies, compiles the code, and executes unit tests to ensure the code is working. If the CI Build fails, the code cannot be merged. This mechanism ensures that the code merged into the master branch can be compiled at all times and that tested logic is not broken.
3. We merge the feature branch into the master branch and close the pull request.
4. Github actions build is triggered automatically for the master branch. Now the CI Build again automatically restores the dependencies, compiles the code, and executes unit tests. Other than in the feature branch build, the code is packaged and uploaded to an Azure Blob store where it can be downloaded as well. Finally, the build automatically creates a pre-release on the Github page.
5. On eX^3, the software now can be updated to the latest version via a simple shell script that downloads the program from the Azure Blob Store. To update the program, it automatically extracts the zip file and replaces the old software with the new version. It also creates soft links and wrappers to use the script via the `rdsp` and `rgraph` command, without the need of manually supplying commands to the Java Runtime Environment (JRE). We chose not to automatically update the software because it would interfere with experiments that are still running.

The process will be the same for the Reddit Dataset Stream Pipeline (RDSP) as well as the graph building tool. The only difference is, that the RDSP is built and tested using sbt, and the graph building tool is built and tested using Gradle.

Chapter 4

Implementation

In this chapter, we illustrate how the Reddit Dataset Stream Pipeline (RDSP) and the Reddit Graph tool are implemented. We present every step of the programs and explain essential design decisions. Furthermore, we discuss how to use the programs.

4.1 Reddit Dataset Stream Pipeline

The idea behind the Reddit Dataset Stream Pipeline (RDSP) is to create a program that could provide the dataset as a stream for other programs via UNIX named pipes. The RDSP has been implemented using Akka Streams. We chose Akka streams because it implements the Reactive Streams standard [23]. Furthermore it is extensible and provides connectors for systems like Apache Kafka and seems therefore like a promising solution for processing the dataset now and in the future.

4.1.1 Command-line interface

The Reddit Dataset Stream Pipeline comes with a command-line interface that is implemented using the scopt command-line options parsing library [44]. Scopt provides a functional description language that can be used to describe all possible parameters. The parsed command-line arguments are written to a case class. Scopt supports arguments, options, and commands. It can specify rules for input validation. Furthermore, it can automatically generate a help text from the specification. If we call the program using the `--help` option, it prints formatted text output of all the possible options to the command line.

```
Reddit Dataset Stream Pipeline 0.1
Usage: redditdatasetstreampipeline [passtrough|statistics] [options]

  -i, --dataset-dir <dir>  Dataset directory that contains the submissions and comments folder. Default value: 'redditdataset'
  -p, --parallel <n>       Number of how many files should be read concurrently.
  --filter <filter>        File name contains filter.
  --exclude <exclude>      File name not contains filter.
  --compress               If enabled, the output will be compressed using ZSTD.
  --help                   prints this usage text
Command: passtrough [options]
Reads the datasets files and provides them as a output stream.
  -s, --submissions        Enables the submission output stream.
  -c, --comments           Enables the comments output stream.
  -a, --authors            Enables the Authors output stream.
  --count                  If enabled, the program counts the number of elements on the stream.
  --submission-out <file>  File or named pipe where to write the submissions csv to. Default value: 'submissions.csv'
  --comment-out <file>     File or named pipe where to write the comments csv to. Default value: 'comments.csv'
  --author-out <file>      File or named pipe where to write the authors csv to. Default value: 'authors.csv'
  --filter-by-sr <file>    Filter by a newline separated list of subreddits.
  --keep-original-json     If enabled, the program writes the original json to the stream.
  --only-user-in-sr        If enabled, the program writes only a subreddit,user csv.
Command: statistics [UserContributionsInSubreddits|UsersInSubreddits] [options]
Runs the program in statistics mode.
Command: statistics UserContributionsInSubreddits
Experiment to count the number of contributions users made in subreddits. A contribution is a post or comment.
Command: statistics UsersInSubreddits
Experiment to count the users that made at least one contribution in a subreddit. A contribution is a post or comment.
  --experiment-suffix <suffix>
                           Experiment out file suffix, e.g. to append a distinct value like a year.
  --statistics-out <dir>   Directory where the results of the experiments shall be written to. Default value: '~'
```

Figure 4.1: Caption

In summary, the command line interface of the RDSP provides basic settings for all commands: Dataset directory, concurrency settings, file inclusion, exclusion filters, and an option to compress the output. Then the user has to choose if the mode shall be run in statistics mode (documented in Section 4.1.5) or passthrough mode (documented in Section 4.1.4)

Statistics arguments. After the statistics mode is selected, the user has to specify an experiment as well. Currently, the experiments to count users and user contributions in subreddits are implemented. Since the program labels the experiments, the user can only specify a file suffix and an output directory.

Pass-through arguments. For the passthrough mode, the user can select if the submissions, comments, and/or author dataset shall be passed through. For the passthrough mode, there is the option to filter the dataset by a list of subreddits provided in a newline delimited text file. There is an option to count the elements in the stream if desired. The user can optionally specify the output files for each dataset individually. The user can also select if the original JavaScript Object Notation (JSON) objects are written to the output instead of the CSV.

4.1.2 System setup

Except for the command line parsing, the whole program runs in one Akka `ActorSystem`. The Actor System is initialized at the

beginning of the program. In the default configuration Akka only utilizes only 64 threads. This resulted in low performance on our test system.

The AMD EPYC nodes on eX³ are equipped with dual AMD EPYC 7601 processors with 32-cores and therefore 128 total threads and 2 TB DDR4 main memory. The ARM CAVIUM nodes come with dual ARM Cavium ThunderX2 CN9980 processors with 32-cores and 256 total threads, and 1 TB DDR4 main memory [40].

In order for Akka to take full advantage of all CPUs on these larger eX³ nodes, we have to adjust the maximum thread count in the configuration. Currently, the maximum thread count in the fork-join-executor is set to 1024. Moreover, the thread pool size for the blocking io is set to 128. With the adjusted settings, Akka can utilize all the 256 CPU threads. Figure 4.2 shows the resource usage after the thread count has been adjusted using htop on one of the eX³ nodes with 256 threads and 1 TB DDR4 main memory. Using all the available threads results in much faster program execution. The CPU usage was documented during the "Count users in subreddits" processing at the beginning of the run. In the end, fewer cores are used because the stateful map concat takes one intermediate result per file.

4.1.3 Akka streams architecture

For both the statistics mode and the passthrough mode, the architecture for processing the dataset is similar. The processing time for every element in the stream is low, but billions of entries are in the dataset. A fundamental design decision was not to distribute the sequential operations on the stream among multiple actors since this would result in extensive message passing and consequently low performance.

Figure 4.3a illustrates the abstract dataflow graph of how the processing is organized. Based on the command line config, we create a Source that emits all file names that make up the dataset. The dataset is organized in one submission file and one comment file per month. This source is connected to a flat map merge operator. The flat map merge operator creates a new Source of output elements for every input element. It is then flattened into the output stream by merging, meaning the original order is not preserved. The maximum breadth of

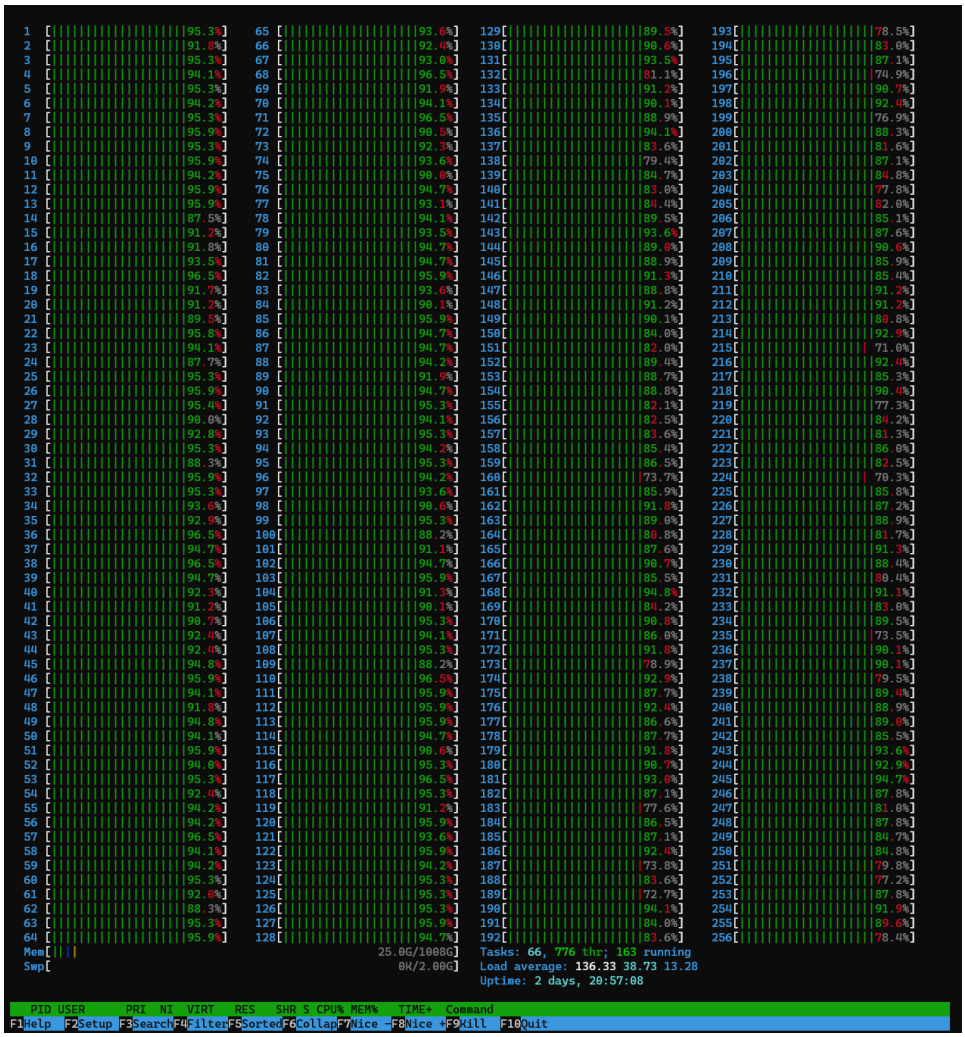
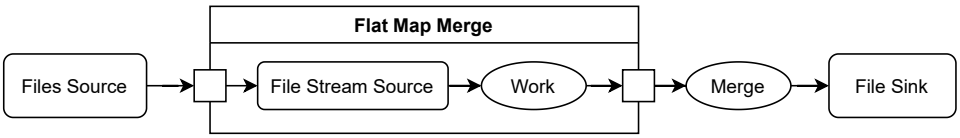


Figure 4.2: CPU usage in htop on 256 threads on two ThunderX2 CN9980 - Cavium 32 core ARM processors. Most of the 256 threads show a more than 90 % usage. We documented the CPU usage during the "count users in subreddits" processing at the beginning of the run.

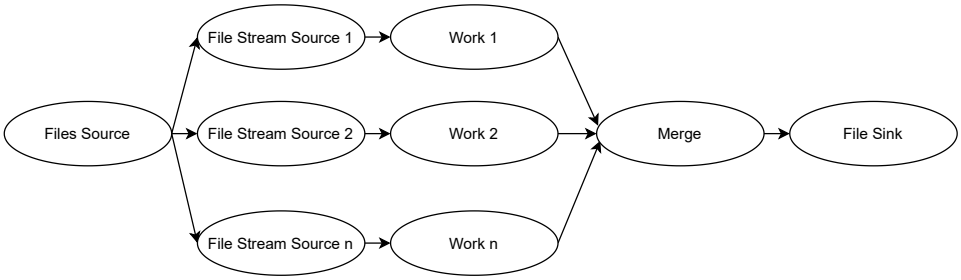
substreams to be consumed at any given time has to be defined [12].

The flat map merge executes n substreams in parallel. By default, the number of parallel processed files is set to match the number of CPU threads on the machine. The data flow diagram in Figure 4.3b, illustrates the parallel execution within the flat map merge.

Within the flat map merge, we create a Source from a decompression stream for the provided filename. The dataset contains XZ, bzip2, and Zstandard compressed archives. The



(a) Flow implementation



(b) Data flow

Figure 4.3: Graph 4.3a shows the concept of how the Akka Streams Graph is built. More text describing the graph. The Data flow diagram 4.3b illustrates how Akka Streams parallelize the defined graph. More text describing the process.

type Source is therefore created dynamically, based on the type of archive. This is done via Apache Commons Compress, which has built-in compression streams for bzip2 and wraps the third-party libraries for Zstandard and XZ [6]. In order to create the Source, we create a Java Compressor Input Stream that takes care of the decompression. We then use the Akkas Stream Converters to wrap the Java Stream into a Source.

After the source, there is the actual work step, which operates on the Byte String stream we get from each step. This varies from stage to stage.

The modified result stream is merged into one stream, without consideration of the order. After the merge, there is the possibility of executing operations that require operating on one stream. Nevertheless, most notably, the aggregated results are written into one Sink. For the Sink, there is the option to use a Sink that compresses the output with ZStandard.

4.1.4 Pass-through mode

The pass-through mode was created to easily make the whole dataset available as a stream for other programs via UNIX named pipes. With the pass-through mode, it is possible to pass the whole dataset as one stream to another program that works

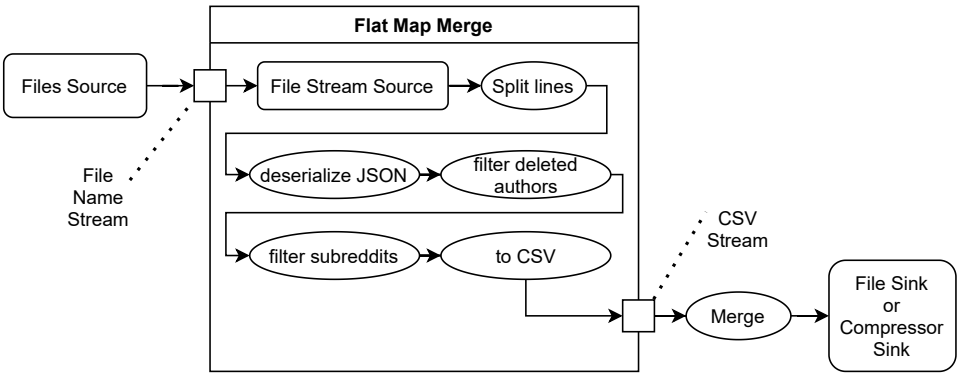
on the dataset. The Reddit dataset is decompressed on the fly and uses all available CPU cores to do that efficiently.

The pass-through only emits only small subsets of the fields as CSV, but there is the option to emit the original NDJSON. Further, the pass-through mode implements filters to filter only for certain subreddits and to ignore deleted authors. A list of subreddit names to include in the output stream can be provided as a new-line delimited text file. Apart from filtered rows, the pass-through mode emits the entities in the dataset without any further aggregation.

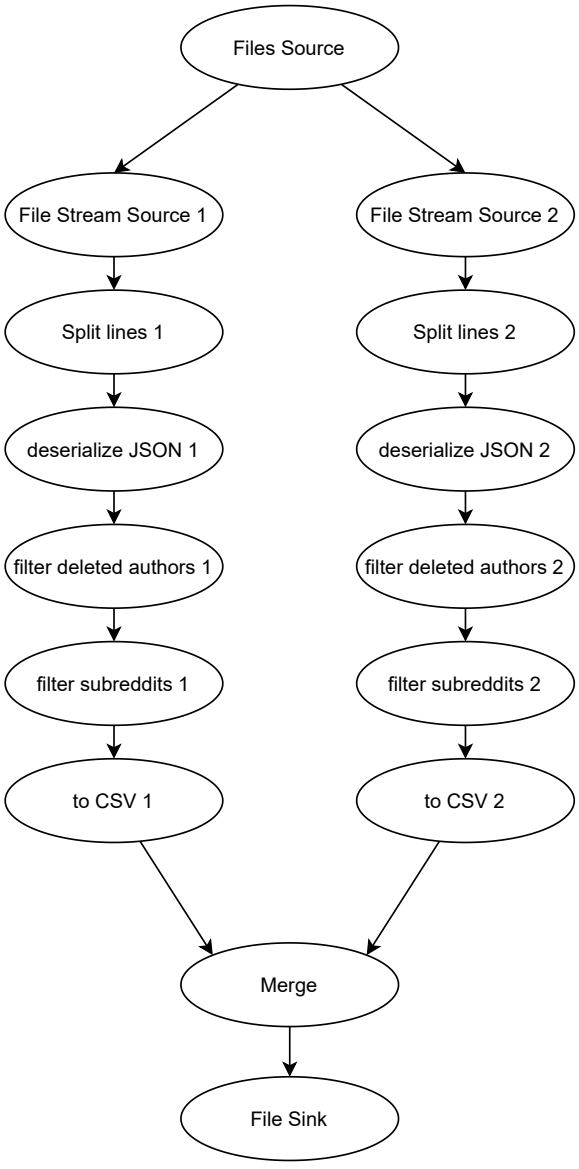
For the pass-trough, the user can select which entities (submissions, comments, authors) shall be processed. If the user selects multiple entities, the multiple streams are started in the Actor System. The but parallelization of the flat map merge through the number of streams.

Source. Figure 4.4a shows the architecture of such a flow. First, a source that emits all the files that shall be processed. The files are filtered only to include those that match the processed entity (submissions, comments, authors). Further, the function to create the source consideres optional exclusion and inclusion filters for the file names. With those filters, it is possible to include or exclude all files that contain a certain string, for example, a year.

Flat map merge. The file is passed into the flat map merge, where the compressor input stream source is created. The source emits a steam of unstructured byte chunks and can span over multiple lines or less than one line. To create messages of whole entities to pass on, the stream is framed into a stream of bytes that belong into one line by looking for the new line byte sequence.
These lines of bytes contain one JSON object each. Therefore the next step is to deserialize the JSON into the corresponding case class with spray-json [46].
Since now objects are passed along the stream, we can operate on them. First, we filter deleted users since we cannot use those for graph building. Then we optionally filter only to let entities pass further onto the stream associated with subreddits specified in the filter list.
The last step within the flat map merge is to convert every entity to CSV.
We did not include any async boundaries within the flat map



(a) Flow implementation of pass-troguh



(b) Data flow

Figure 4.4: Graph 4.4a shows More text describing the graph. The Data flow diagram 4.4b illustrates, how . More text describing the process.

merge that would split the subsequent operations onto multiple actors since we are already utilizing all the available CPU threads by streaming many files in parallel. Async boundaries introduce in this context unnecessary overhead due to message passing.

Sink. The steam that leaves the flat map merge flow is a stream that contains all the processed CSV byte strings from all dataset files. Depending on the settings, this CSV stream flows in one of two possible Sinks. Either into an IO Sink that writes the bytes directly to a file or a **unix!** (**unix!**) named pipe. Alternatively, into a Sink that compresses the stream with ZStandard before writing it to a file.

Data flow. Figrue 4.4b illustrates how the stream is processed in a data flow diagram. Every step within the flat map merge is split onto n actors, and therefore most of the processing is done in parallel. The stream is only merged to write it into a single Sink.

4.1.5 Statistics mode

Since we have to limit the graph size, the statistics mode was created to determine which subreddits are important. The statistics mode currently implements two methods of counting the size of subreddits. One method is to count the total number of user contributions per subreddit. The other to count the number of users that contributed at least once to that subreddit. When creating the "subreddit, user" CSV with the pass-through command, we can use a list of top n subreddits to shrink down the dataset. This list is created from the results of the statistics mode. Therefore the statistics mode acts as a preprocessing step for the pass-through.

The statistics mode reads the whole dataset and creates a CSV list of all subreddits and out the number of users or contributions in that subreddit. While reading the data, the submissions and the comments dataset are combined since we only use the subreddit name and the author name of every entity to create the list. Compared to the pass-through mode, there are fewer user options the user can make since the counting follows a specific implementation.

Source. Figure 4.5a illustrates the architecture of the statistics flow. At the beginning of the flow, a Source emits the names of

the files to be processed. The Source lists both the submissions and the comments archives. By default, it takes all the submissions and comments, but the optional filter to include and exclude certain files applies here.

Flat map merge - Users in subreddits. The file is passed into the flat map merge, where the compressor input stream source is created. The source emits a steam of unstructured byte chunks and can span over multiple lines or less than one line. To create messages of whole entities to pass on, the stream is framed into a stream of bytes that belong into one line by looking for the new line byte sequence.
These lines of bytes contain one JSON object each. Therefore the next step is to deserialize the JSON into the corresponding case class with spray-json [46].
Since now objects are passed along the stream, we can operate on them. First, we filter deleted users since we cannot associate them with a specific account. Leaving the deleted users in the stream would increase the final count of each subreddit by 0 or 1
The "count Users in Subreddit per File" operator contains the logic for counting the users. It is implemented as a stateful map concat. It being stateful means that the flow can hold an internal state accessible for every element that is processed on the stream. In this internal state, we hold a mutable `HashMap[String, HashMap[String, Int]]`. Subreddit names index the outer Hashmap. In the inner Hashmap, we hold another Hashmap that contains the users in this subreddit, plus an integer counter with the number of contributions per subreddit.
For each UserInSubredditEntity on the incoming stream, the flow checks if the subreddit is present in the hashmap. If the subreddit is not present, a new child Hashmap is created, including the current user with a count of 1. If the subreddit is present, the user is added or updated in the child Hashmap. In the update operation, the counter is increased by one.
It is to note that the "count Users in Subreddit per File" does not emit an output element towards the stream for every incoming element. It buffers until the processed archive file reaches the end of the file. Once the last entity is processed, a list of subreddits and users in that subreddit is created from the Hashmap. All the elements that `List[CountPerSubreddit]` are emitted to output.

Flat map merge - User contributions per subreddit. The user contribution per subreddit count is similar to the users in subreddits count. The only differences are that the deleted users

are not filtered because they still count as a contribution by some user. Since we do not take the user names into account, the Hashmap is simpler `HashMap[String, Int]`. The index contains the subreddit name, and the value holds a counter for the subreddit that is increased according to the entity's subreddit. The User contributions per subreddit directly emits this list to the stream once the current file is completely read.

Merge. The counts per subreddit coming from the flat map merge are unique per file but not unique considering all files. We, therefore, have to merge the results of the parallel processing stage. This was a conscious design decision to avoid synchronizing access to one Hashmap while processing the files. The merge operation is a stateful map concat that holds an `HashMap[String, Int]`. In this Hashmap, we aggregate the results we get from a single stream. Figure 4.5b illustrates the merge operation. All the counts for the subreddits are written into the stateful Hashmap. If an entry for the currently processed subreddit already exists, we add up the count of the entity from the stream to the count that is already in the Hashmap. Once the results from all archive files are combined into one Hashmap, the content of this Hashmap is emitted to the stream and converted to CSV.

Sink. The writes the resulting CSV stream to disk. The option to compress the output stream is there but not necessary since the resulting file is rather small compared to the initial dataset.

Data flow and performance The data flow diagram in Figure 4.5b illustrates how the flow is parallelized. All the files are processed and counted in parallel. However, the following merging process is only running in one actor. This is a bottleneck. Especially when most of the files are processed, fewer and fewer CPU threads are utilized by the program because combining all the intermediate results in one enormous hashmap takes time. Nevertheless, this is better than working in one enormous hashmap during the whole process. Another reason why this step seems to be lagging because it does not receive a message before the first dataset file is fully processed.

Memory usage. This method of in-memory aggregation using Hashsets was built around the idea of leveraging the capabilities of eX³ nodes with 1 TB or 2 TB main memory. Therefore we were running the Java Virtual Machine (JVM) with the maximum memory allocation pool set to 1.8 TB and the initial memory allocation pool set to 1 TB. The maximum memory usage

we logged from within the JVM was during the "user contributions per subreddit" run, with a total of 340 GB at the time of logging.

4.1.6 Unit Tests

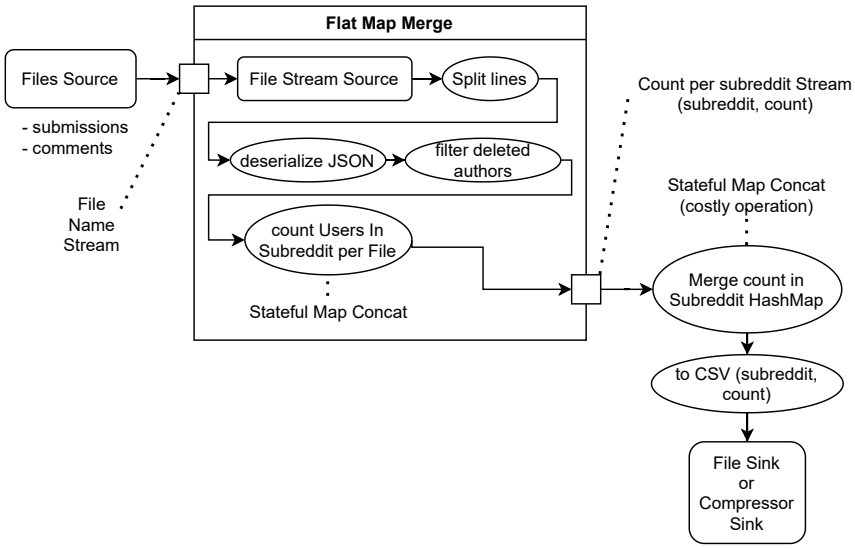
To ensure that the base implementation is working and test certain program features isolated, we use unit tests. The tests were implemented using ScalaTest. ScalaTests specification-based approach makes it easy to write meaningful tests and clarify what the implemented tests are for. The tests are organized into three parts.

With ScalaTest, we can test classes and objects like with any other unit test framework. However, it is impossible to execute a Flow and get the results as it would be with a method. In order to test flows, we have to run them in their own Actor system. To do that, we create a simple Akka Streams Graph that provides data via a source to the flow we want to test and then submits the results in a sink. After running the whole Graph, we have to verify the results from the Sink.

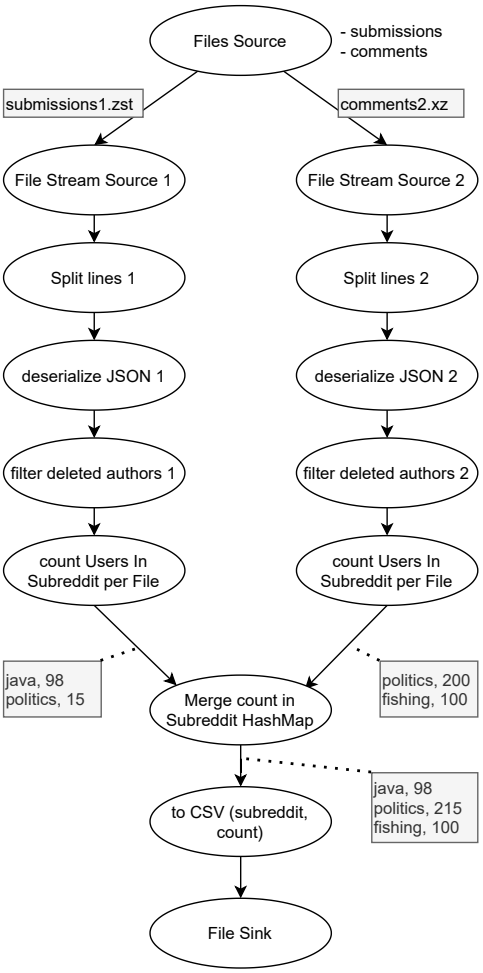
Statistics tests. The statistics specs verify that the counting of users in subreddits is done correctly. They test that the flow that counts users in subreddits per file `countUsersInSubredditsPerFile` ignores multiple user entries per subreddit and does not increase the count. Furthermore, it tests that the subreddits created from the CSV are carried over correctly.

Flow tests. The flow tests ensure that the isolated sub-flows operate as intended. The sub-flow `ndJsonToSubmission` should convert JSON byte strings to Submission objects. We test that by providing a simple JSON line and verify the results, and then we do the same thing with a JSON line taken from the live dataset. Furthermore, we test that the more generic `ndJsonToObject` flow can deserialize a line from the Reddit submissions dataset and convert it to a Submission. We also assure that the `ndJsonToObject` ignores lines filled with null bytes since they occur in the dataset. The other way around, we also test that the `objectToCsv` flow can convert entities that implement the `ToCsv` trait can be converted to CSV byte strings.

Model tests. The model tests verify that submission objects are converted to the correct CSV lines and that the number of CSV headers matches the number of fields. Currently, only



(a) Flow implementation of the statistics mode (count users in subreddits)



(b) Data flow

Figure 4.5: Graph 4.5a shows More text describing the graph. The Data flow diagram 4.5b illustrates, how . More text describing the process.

the submission objects are under test because this helped with detecting errors that occurred because of frequent changes to the model during the implementation. The other model is lacking a test.

4.2 Graph Building

The Graph Building application is used to construct the Reddit Graph from the `subreddit,user` CSV that the Reddit Dataset Stream Pipeline provides. It utilizes the JGraphT library as its underlying data structure and utilizes Graph algorithms provided by JGraphT. Moreover, it can import previously generated graphs from edge and vertex lists to avoid regenerating graphs for experiments.

The Graph Building application is implemented in Java. It is using Gradle as its build automation tool and dependency manager.

4.2.1 Building the graph

The graph is implemented as a JGraphT undirected weighted graph `DefaultUndirectedWeightedGraph<Vertex, Edge>`. For the edges and the vertices, we used custom implementation. These custom classes implement serializable properties and custom functions we use for calculating the scores on the graph.

In order to build the Graph, several steps are necessary. After adding the vertices, we build a user subreddit map, from which we determine the edges between the vertices. Once the edges are added, we can calculate the scores and weights on the graph over multiple iterations.

Reading the CSV stream. The Reddit Dataset Stream Pipeline provides a `subreddit,user` CSV stream for the top n subreddits. From this stream, we are going to create the graph. The CSV stream is read from a file stream. Depending on whether the input stream is compressed, an additional Compressor Input Stream is used to handle the decompression. Since we are using Open CSV [29] to parse the CSV, we can wrap the input stream into a CSV Reader. The CSV reader implements `Iterable<String[]>`, which means we can just use a loop to read the CSV stream line by line. Each string array represents one line of the `subreddit,user` CSV file. The input can be split over n sequentially read files. In our case, we had one stream for the submissions and one for the comments.

Adding vertices. While reading the input stream, we can already add the vertices to the JGraphT Graph object. If a vertex already exists, we skip it.

Subreddits per user map In the same loop, we will create a `HashMap[User, HashSet[Subreddit]]`. When we read the CSV, we find or add the user in the HashMap and add the subreddit to the HashSet if it is not already present. We get a list of users and, per user, a distinct list of subreddit the user contributed to. We that later to determine where to create the edges. Reading the files is done now. The next steps will be performed in memory.

Determine edges In Section 3.3.2 we discussed the evolution of the graph creation. In Figure 3.6 we illustrated how to combine the list of subreddits associated with a user to get all possible unique combinations of subreddits per user. For example for the subreddits A, B, C, all possible unique combinations would be AB, AC, BC. We implemented the proposed method while iterating over the `HashMap[User, HashSet[Subreddit]]`. For each of these subreddit combinations, an edge is added between the two vertices. In our example, we would add an edge between AB, AC, and BC as illustrated in Figure 4.6.
Suppose an edge is already present in the graph, another user-contributed to those subreddits. Since we want to show the connection strength between subreddits, we increment U_{ij} by calling the method `incrementNumberOfUsersInBothSubreddits` on the edge. U_{ij} is defined as the number of users that posted at least once in both subreddits.
Once we added all the edges for every user, we no longer need the user subreddit HashMap. In order to reduce memory consumption, the user entries from that hashmap are removed once the edges for one user are added.

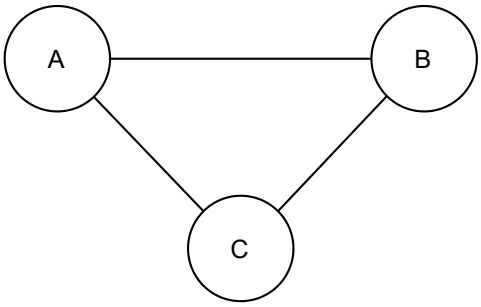


Figure 4.6: Example graph between the subreddits we get if we draw an edge for every unique combination (AB, AC, BC) between the subreddits A, B, C.

Score calculation. In order to compute the graph scores faster, this is done multi-threaded via Javas `parallelStream()` feature. We also save the results of our calculations to reduce iterations - especially for recursive operations. A downside of making the computation parallel is that we now have to consider that dependent values might not have been computed. We, therefore, split up the calculation into three parts. Calculating the vertex scores that do only depend on U_{ij} and are therefore independent. Calculating the edge scores that depend on these vertex scores. Moreover, finally, the calculation of the vertex scores that depend on the previously calculated edge scores.

Independent vertex score calculation When calculating the independent vertex scores, we currently only calculate one score. The Sum of edge weights U_{ij} connected to the vertex. This is done by looping through the edges connected to the current vertex and adding the edge weights.

Edge score calculation We depend on the `sumOfEdgeWeightsConnectedToVertex` calculated in the vertex score calculation step when calculating the edge scores. For each edge, we cache the source and target degrees by using the `graph.degreeOf()` provided by JGraphT. Further, we cache the weighted source and target degree, the average weighted source, and target edge weight. Finally, we calculate the edge weight.

Vertex scores dependent on edge weight. As the last step, we iterate over the vertices in parallel again to calculate:

Degree degree
Weighted degree degree
And the local clustering coefficient

4.2.2 Exporting the graph

Once the graph is built, we can export the graph. We will export the graph as a DOT file to visualize it, and we will also export the graph as a universal edge and vertex CSV list.

DOT file JGraphT has an integrated DOT exporter. We manually have to specify which attributes are written into the dot file. Dot files can be used to visualize the graph in tools like Gephi.
The vertices in the dot file are exported with the following attributes by adding a custom attribute provider.

- degree
- degree degree
- weighted degree
- weighted degree degree
- local clustering coefficient

Gephi uses the attribute "weight" for the edge weight. Therefore, add an attribute that uses the edges weight property. We only export the weight to keep the file size down since there will be many more edges than vertices in the graph.

Vertex and Edge lists Vertex and edge lists are written as CSVs since it is a straightforward format. They can be reused, for example, by loading them into a data frame with Pandas in Python. They also can be read fast to recreate the graph in JGraphT. Therefore, it is not necessary to wait on the graph generation to experiment within JGraphT. Writing the edge and vertex list is straightforward. Both classes implement the CSV interface that returns a string array. This string array can be converted to a CSV line with Open CSV and then written to a file. The file names can be specified by command-line arguments.

4.2.3 Technical details of loading the graph

Load vertices from the vertex and edge CSV lists are simple. The vertex and edge files are read with a File Reader. The file reader is passed into an Open CSV `CsvToBeanBuilder`, which can be used to provide us with an iterable stream of Vertex and Edge objects. These objects are added to the graph object, and our graph is recreated.

4.2.4 Command-line interface

Commandline interface is implemented using picoli, [31]. Currently, there is the option to decide between two modes - creating the graph and loading the graph.

Graph Creation This mode builds the graph from the "subreddit, user" CSV files. It is possible to specify one or more CSV files as arguments. Furthermore, the user can specify files for the DOT export as well as the locations of the vertex CSV and edge CSV.

Load Graph This mode loads the graph from the vertex and edge lists. Compared to building the graph again, it is much more efficient. Arguments specify edge and vertex list CSV to load the graph from. This mode also performs a DOT export, which makes it easier to modify the DOT files for visualization.

4.3 Python scripts

During various steps of the project, we used an assortment of Python scripts. Whether to take a quick look into the dataset or to generate plots.

Filter lists The Reddit Dataset Stream Pipeline (RDSP) provides us with a list of subreddits and a count of how many users are in that subreddit. We used a Python script to create the filter lists for the RDSP pass-trough mode. We used Pandas to load the dataset into a data frame. Then filtered and sorted the dataset for the largest 5, 10, 100, 1000, and 10000 subreddits. Then we export the subreddit names to a text file. Now we have a list of the top 5, 10, 100, 1000, and 10000 subreddits we can use to filter with the pass-through mode.

Experiments We used Python, Pandas, Mathplotlib, and Jupyter Notebooks for many of the experiments described in Chapter 5.

4.4 Supporting Tools

In Chapter 3.4, we proposed a few tools to support the development process.

SLURM We used SLUM as proposed to schedule all the runs on eX³. We, therefore, created an assortment of bash scripts that made it easier to label all the resulting files, interpret parameters and run the Reddit Dataset Stream Pipeline and the Graph Building tool. Especially for creating the yearly graphs, a scheduler like SLURM is helpful to run the batch jobs and monitor the progress. Figure 4.7 shows the SLURM queue on eX³, with 14 nodes working on different tasks in parallel.

Continuous Integration (CI) We implemented the CI / CD pipeline using Github Actions as proposed in Figure 3.8.

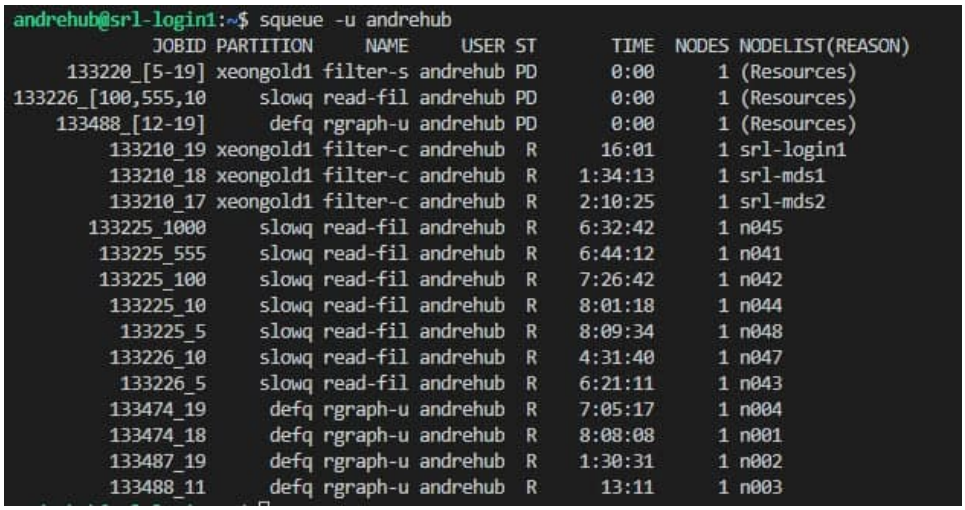


Figure 4.7: The SLURM queue on eX³. In the queue, there are Various tasks to filter and build the graphs. Currently, we are running the tasks on three different queues using a total of 14 nodes in parallel. The active tasks are for graph building, filtering and creating subsets of the dataset.

Google Drive and rclone Furthermore, we used Google Drive to store a copy of the dataset and a copy of the results. Due to the dataset size and the size of the results, we used rclone to synchronize the files stored on eX³ with Google Drive.

Chapter 5

Experiments

This Section discusses experiments and statistics performed on the Pushshift Reddit Dataset and the generated Graph. For creating the graph, we determined the top n subreddits. Furthermore, we show important metrics calculated for the vertex and edge lists. Finally, we present a visualization of the generated graph.

5.1 Top n subreddits

To get a deeper understanding of the Reddit landscape, we had to develop a method to determine which subreddits are of most importance. We, therefore, created the statistics mode conceptualized in Chapter 3.2.1 and implemented it in Chapter 4.1.5. The statistic mode implements two ways of counting. One that counts the number of contributions in subreddits and one to count the number of unique users in subreddits. For building the graph, we decided to use the "number of users in subreddits" as a base to filter because we are also building the graph based on the number of users in both subreddits and not based on the number of contributions—nevertheless, the contributions where helpful to get important insights on the size of subreddits.

Creating a graph in the size of Reddit is a challenge. Due to the vast number of users on Reddit, it is to be expected that the graph between subreddits is highly connected. Storing such a graph requires $\mathcal{O}(n^2)$ space. Therefore, we have to face constraints in time and memory. We, therefore, decided to restrict the graph to the arbitrary limit of the top 10000 subreddits.

5.1.1 Number of unique users in subreddits between 2005 and 2019

When determining the number of unique users in subreddits between 2005 and 2019, we count the number of unique users that contributed at least once to a subreddit using the Reddit Dataset Stream Pipeline (RDSP). A contribution can be either a post or a comment.

The resulting "subreddit, count" CSV is sorted and filtered with the *pandas* Python package, so we can get a list of the top n subreddits we use as a base for generating the graph. We created filter lists for the top 5, 10, 100, 1000, and 10000 subreddits based on this method. These are primarily used to create graphs of different sizes. Furthermore, we created yearly count lists, which are used to create yearly time slices of the graph.

The ten largest subreddits by unique users are presented in Table 5.1 in descending order. The counts revealed a significant difference between the largest subreddit, "AskReddit" with around 71 million unique contributors, and the second-largest subreddit, "funny" with around 29 million unique contributors. Further down the list, the differences between the subreddits become less noticeable.

The histogram in Figure 5.1, shows how the number of unique users is distributed over the subreddits. It is noteworthy that the number of subreddits on the y-axis is represented in a logarithmic scale, and the x-axis is scaled in steps of 10 million users. To the right end of the x-axis, we can observe one isolated bar representing the subreddit "AskReddit" with 71 million unique contributors as listed in Table 5.1. Moreover, above the 20 million user mark, there are only the subreddits in 2nd and 3rd place from the same table. The vast majority of subreddits are concentrated in the leftmost bar. In summary, these results show that there are very few large subreddits but many small subreddits.

5.1.2 Number of contributions in subreddits between 2005 and 2019

To determine the number of contributions in subreddits between 2005 and 2019, we count all submissions or comments per subreddit using the Reddit Dataset Stream Pipeline (RDSP). A contribution is a single submission or a single comment, and

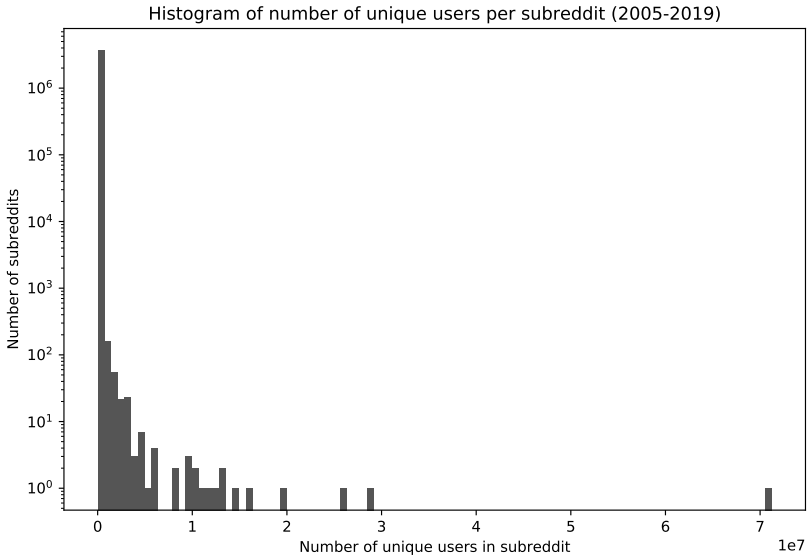


Figure 5.1: Distribution of uniqe users per subreddit. TODO: Explain numbers 1e8. Log scale

they are weighted them equally. Meaning for every post or comment in a subreddit, the count was incremented by one.

The resulting "subreddit, count" CSV is sorted and filtered with *pandas* so that we can get a list of the top n subreddits. We created filter lists for the top 5, 10, 100, 1000, and 10000 subreddits based on this method. These can be used to create graphs. Furthermore, we created yearly count lists, which can be used to create yearly time slices of the graph.

Table 5.2 lists the ten largest subreddits by contributions in descending order. A significant difference was found between the first subreddit, "AskReddit" with around 508 million contributions, and the second-largest subreddit, "politics" with around 121 million contributions. Further down the list, the gap between the subreddits becomes less prominent. Compared to the user contributions in Table 5.2, the most popular subreddit "AskReddit" stays the same, but politics comes in 2nd and pushes funny to the 3rd place.

The histogram in Figure **??**, shows how the number of contributions is distributed over the subreddits. It is noteworthy that the number of subreddits on the y-axis is represented in a logarithmic scale, and the number of contributions in the x-axis

subreddit	count
AskReddit	71233827
funny	28882078
pics	25886189
gaming	19312574
videos	16283626
todayilearned	14796435
WTF	13238354
worldnews	12908501
aww	12354192
politics	11478928

Table 5.1: Top 10 subreddits by unique users between 2005-2019

is shown in steps of 100 million. To the right end of the x-axis, we again can observe one isolated bar representing the subreddit "AskReddit" with 508 million contributions as listed in Table 5.2. Moreover, above the 100 million contributions mark, there are only the subreddits in 2nd and 3rd place from the same table. The vast majority of subreddits are concentrated in the leftmost bar. This confirms what we saw in Section 5.1.1. We can observe that there are very few large subreddits but a vast majority of small subreddits.

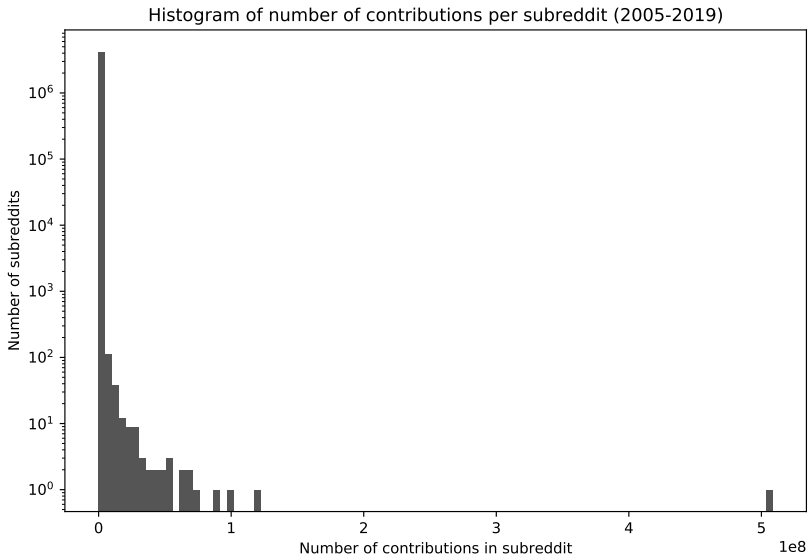


Figure 5.2: Distribution of user contributions per subreddit. TODO: Explain numbers 1e8. Log scale.

subreddit	count
AskReddit	508529584
politics	121041987
funny	100436096
pics	88154864
leagueoflegends	72562127
gaming	68271941
worldnews	67099490
nba	66082257
nfl	63448568
news	53710557

Table 5.2: Top 10 subreddits by contributions between 2005-2019

5.2 Graph scores

While creating the graphs, we calculated various scores to gain more insight into the overall graph and to determine the edge weight in multiple iterations. In this Section we will look into those scores in detail. We used *pandas* to load the edge and vertex lists we exported after creating the graph into a data frame to analyze the scores.

5.2.1 Vertex scores

Figure 5.3 shows distributions for all the calculated vertex scores.

U. in Figure 5.3a is the sum of all edge U_{ij} (users in both subreddits) connected to a vertex. Thus, U_{ij} is the number of users that posted in both subreddits. Interestingly we can still see the same decline as we saw when counting the users per subreddit in Section 5.1.1, just on a more detailed scale.

Local clustering coefficient. The local clustering coefficient in Figure 5.3b indicates that the graph is almost fully connected and that it is tightly coupled as suspected. This can be explained by the fact that we are using only the top 10000 subreddits as a basis, meaning if there is only one user that posts in two subreddits, there will be an edge - which is quite likely for so many users.

Degree. The degree in Figure 5.3c is the sum of edges connected to this vertex. Many vertices are connected to

all other vertices. The least connected subreddits are still connected to over 8400 neighbors, confirming that this graph is highly connected.

Weighted degree. The weighted degree in Figure 5.3d puts the degree in relation with the sum of users that posted in this and another subreddit U. The weighted degree is defined as $weighteddegree = U/degree$.

Degree degree. The degree degree in Figure 5.3e is the sum of edges connected to the neighbors of this vertex. As with the degree, the degree degree indicates the same trend towards a fully connected graph.

Weighted degree degree. The weighted degree degree in Figure 5.3f puts the degree degree in relation with the sum of users connected to neighbors of this subreddit. The weighted degree is defined as $weighteddegreedegree = sum(U)/degreedegree$.

5.2.2 Edge scores

Figure 5.4 shows distributions for all the calculated edge scores, including the weight W_{ij}. i and j stand for the subreddits connected by the edge.

Number of contributors in both subreddits. The number of users that contributed in both subreddits i and j at least once is defined as U_{ij}. Figure 5.4a shows the distribution of U_{ij}. When looking at the user counts associated with the edges, we can see the same behavior as with the users. A few edges with many users posting in two subreddits and many edges with fewer contribution users. This indicates that not all edges are equally strong.

Source and target degree. The degree distribution for i in Figure 5.4c and the degree distribution for j in Figure 5.4dshows the degree of the connected vertices. Here we can see as well that many vertices are connected to all other 10000 vertices.

Weighted source and target degree. The weighted degree in Figure 5.4e and Figure 5.4f is the sum of all U connected to the source vertex i and respectively the target vertex j. In other words, the sum of all edge U_{ij} connected to the vertex i or j.

Average weighted source and target weight. The average weighted edge weight is the number of users connected to the source or target vertex divided by the source or target degree. Those can be defined as:

$$a_i = \frac{U_i}{degree(i)}$$
$$a_j = \frac{U_j}{degree(j)}$$

Figure 5.4g and Figure 5.4h show the distributions of the average weighted edge weights.

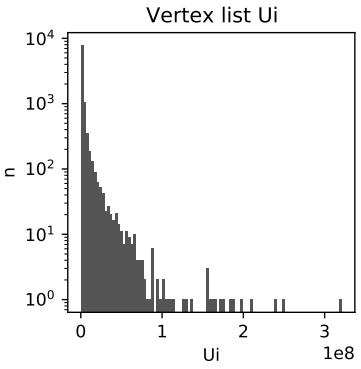
Edge weight. The edge weight W_{ij} sets the amount of contributors hat contributed in the connected subreddits in relation to amount of users connected all the neigbours of the source and target. The edge weight W_{ij} is be defined as:

$$W_{ij} = \frac{U_{ij}}{(weightedSourceDegree + weightedTargetDegree)/2}$$

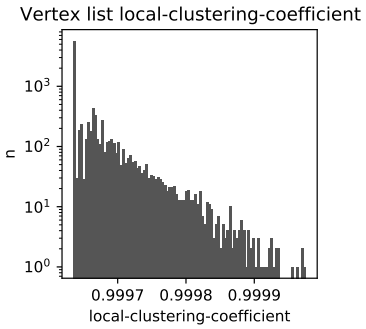
Figure 5.4b shows the distribution of the edge weight. As with U_{ij} we find that there are many low edge weights and few high edge weights. We can use that when representing the graph by filtering edges with low edge weights to show the structure of the graph.

5.3 Graph visualization

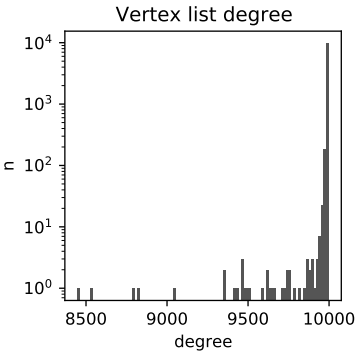
In order to investigate the generated graph in detail, we create a visual representation of the graph. Visualizing the fully connected graph is challenging because there are 49966785 edges in the graph. We therefore opted to filter edges with a weight lower than about one third. Figure 5.6 shows the graph for the top 1000 subreddits from 2005 until 2019. The different colours mark different clusters as identified by Gephis modularity rank. Figure shows the top 10 thousand subreddits divided into communities. An intersting observation is how mainstream anime and gaming have become.



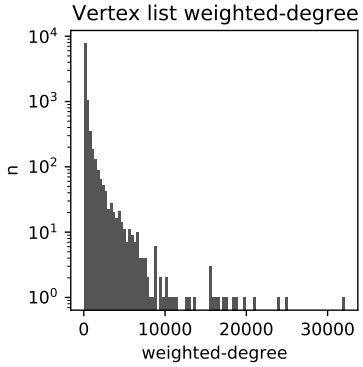
(a) U



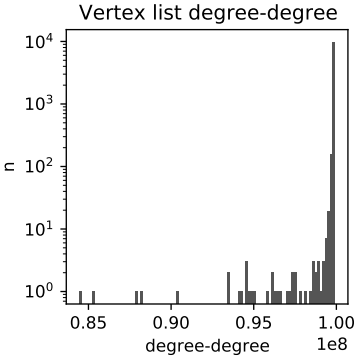
(b) local clustering coefficient



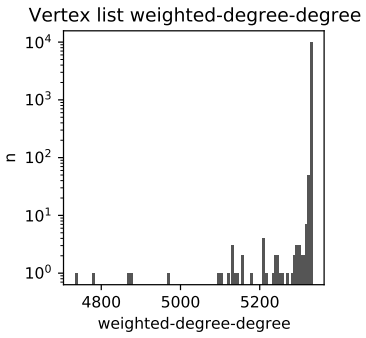
(c) degree



(d) weighted degree

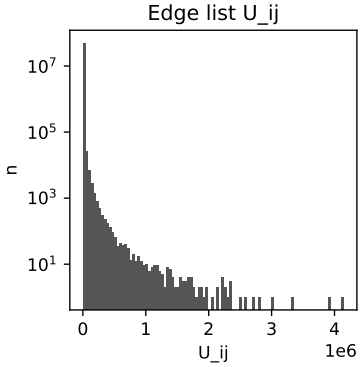


(e) degree degree

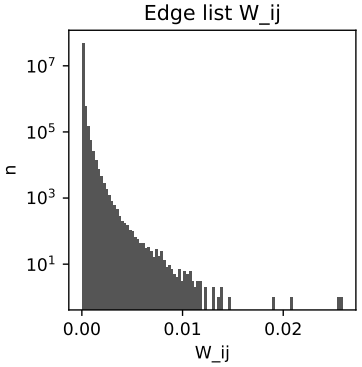


(f) weighted degree degree

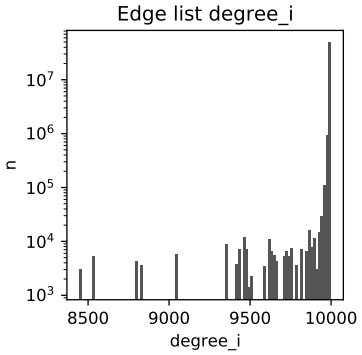
Figure 5.3: Vertex list top 10k 2005-2019 basic metrics



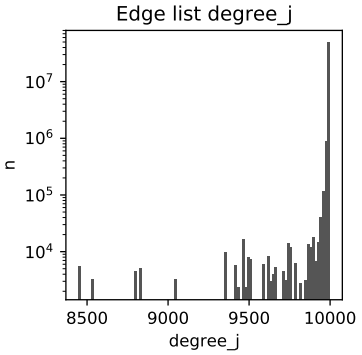
(a) U_{ij}



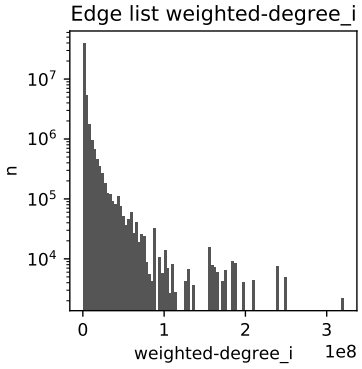
(b) W_{ij}



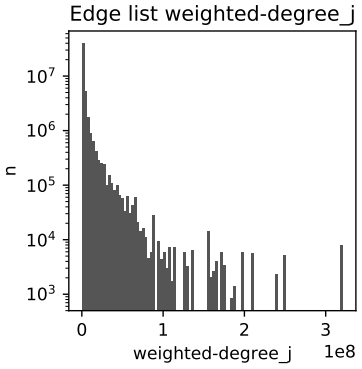
(c) degree i



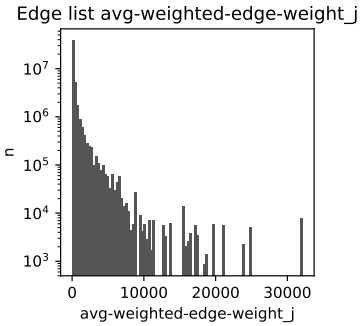
(d) degree j



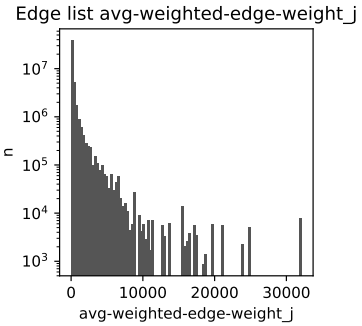
(e) weighted degree i



(f) weighted degree j



(g) avg weighted edge weight i



(h) agv weighted edge weight j

Figure 5.4: Edge list top 10k 2005-2019 basic metrics

Figure 5.5: The final graph of subreddit relations from Gephi for the top 1000 subreddits.

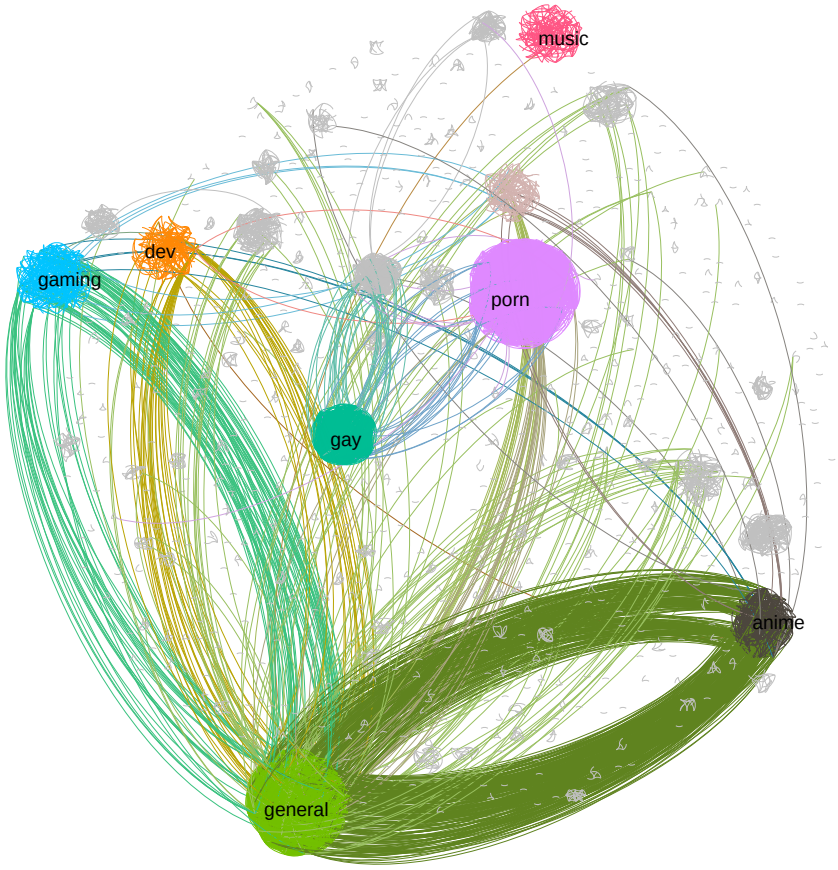


Figure 5.6: The final graph of subreddit relations from Gephi for the top 10000 subreddits.

Part III

Conclusion

Chapter 6

Outlook

This thesis has demonstrated how to build a massive graph between subreddits on Reddit from a dataset transformed into a parallel stream. We, therefore, created the Reddit Dataset Stream Pipeline (RDSP), a processing pipeline for the Reddit dataset based on Akka Streams. We then had to develop methods to create a graph between the subreddits on Reddit and weigh them. We investigated the generated graphs and presented the results of the experiments, including a visualization of the graph.

Reddit Dataset Stream Pipeline. When creating the RDSP we had to overcome the problem of file size. We, therefore, introduced an approach to read, extract, filter, analyze and transform the Reddit dataset with Akka Streams in parallel. Since the resulting pipeline is stream-based, it can be extended to function with live data in the future.

Graph building method. When creating the graph for Reddit we had to work with independent subreddits. We developed a method to create edges between subreddits by identifying and aggregating users that contributed to two subreddits. Furthermore, we developed a weighting method that took the importance of neighbors into account and applied it to those edges. The thesis gathers and discusses the distributions of the scores in the graph and provides a more detailed overview of the landscape of Reddit.

Parallel score calculation. Calculating all the metrics on a graph of this size was a noteworthy challenge. In this thesis, we showed how we could split up the workload into multiple parallel executable workloads. The edge and vertex scores are calculated in parallel over multiple iterations. Therefore we

could deal with expanding graph sizes and build the graph promptly. It drastically reduced the overall compute time and helped not consume more resources than necessary.

Graph visualization. Creating a visual representation of the discovered graph helped us better understand if it is meaningful and resembles the landscape of Reddit. In this thesis, we have shown our representation of the subreddit landscape of the top 10.000 subreddits. We introduced a custom edge weight that takes neighbors into account. We filtered edges with low edge weights to only show important edges. This was necessary to keep the graph visually comprehensible.

Raw Graphs and Time Slices. To make the results more accessible, we created a set of pre-processed graphs for the top 5, 10, 100, 1000, and 10000 subreddits. The created graphs are stored as DOT files and as edge and vertex CSV lists. Moreover, we pre-generated time slices from 2005 to 2019 – even though the results are not discussed in this thesis. The time slices are stored as DOT files and as edge and vertex CSV lists. Overall a dataset of around 151 GB plus another 509 GB containing compressed subsets of the Pushshift Reddit dataset were used for creating the graphs.

Project Contribution. This thesis makes Reddit a data source more accessible and provides prepared graphs that can be further explored. It is the first contribution to the UMOD project concerning Reddit.

In summary, we have shown a first approach of processing the Reddit dataset as a stream and creating a graph between independent subreddits based on user interaction.

6.1 Challenges

While creating the approach for this thesis, we faced several challenges.

Dataset size. The most obvious challenge was the dataset size. Working with such a vast amount of data requires time and resources. Without the computing power of eX^3 it would not have been possible to handle the dataset in reasonable time.

Akka streams. The main challange wiht Akka Streams is that it differs from standard programming models. It took some time to think in flows and form proper streams that performed well. A significant effort was neccessary to determine where to place async boundaries to improve performance.

Visualizing the graph. When visualizing such a large graph, we ran into limitations on what software like Gephi can do on a notebook. The main problem was the large number of edges that require substantial amounts of memory. However running visualization programs on the server comes with perfromance problems on its own.

6.2 Limitations

Live data. Initially, it was intended that the system would scrape and continuously refresh the dataset with live data from Reddit. During our research, we found out about the Pushshift Reddit dataset and concluded to use that dataset as a basis for the thesis. The main reason against scraping the data ourselves was that it would not be feasible for us to scrap all the information we need with Reddit's described rate limitation.

Limited to the top n subreddits. We found an approach for building a graph between the most important subreddits. However, our method is currently still limited by memory and CPU resources. With the shown method, it is therefore not feasible to create a graph containing all the subreddits.

6.3 Future Work

Livestream data. One future improvement would be to extend the RDSP, so it can steam data from a live source, such as a continuously running crawler. First, this requires the use of a different Source that replaces the Files Sorce. Second, it is necessary to create a stateful operation that determines the top n subreddits on the stream. To not be bound by memory, we suggest creating time slices. The stateful operation must be limited to either a sliding or a rolling window to not consume a possibly infinite amount of memory.

Create graphs on stream. Another improvement would be to create graphs while streaming. We propose to use the same windowed approach to create time slices.

Akka cluster. In order to handle the possibly growing amount of data Reddit provides, it would be interesting to look into how Akka cluster could help split the workload onto multiple machines – for a continuously running stream-based system.

Measure the graph. We have performed basic experiments on the created graph. We propose to conduct further experiments in order to get a better understanding of the graph. It could be interesting to better understand how two subreddits connect, what the shortest path between two subreddits is, and how subreddits cluster together.

Experiments over time. With the created time slices, there is the possibility to look at how Reddit changes over the years. We suggest looking at the distributions over time and investigate if Reddit as a community grows together or vice versa. Furthermore, it can be interesting to visualize the development of the Reddit graph over time. We suggest visualizing how different subreddits grow and gain importance by increasing the vertex size dynamically.

Predictions. When looking at the graph over time, it would be interesting to see if one could predict how certain aspects of the graph will behave. However, it would require more fine grained time slices and possibly more metadata to train a learning algorithm.

Directional Graph. In this thesis, we created an undirected graph of Reddit. We recommend that future research examines how to create a directional graph instead.

Bibliography

[1] *5G, coronavirus and contagious superstition*. en. Section: World news. Apr. 2020. URL: http://www.theguardian.com/world/2020/apr/26/5g-coronavirus-and-contagious-superstition (visited on 05/19/2021).

[2] Katie Anderson. "Ask me anything: what is Reddit?" In: *Library Hi Tech News* 32 (July 2015), pp. 8–11. DOI: 10.1108/LHTN-03-2015-0018.

[3] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks." en. In: *Proceedings of the International AAAI Conference on Web and Social Media* 3.1 (Mar. 2009). Number: 1. ISSN: 2334-0770. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/13937 (visited on 05/28/2021).

[4] Jason Baumgartner et al. "The Pushshift Reddit Dataset." en. In: *Proceedings of the International AAAI Conference on Web and Social Media* 14 (May 2020), pp. 830–839. ISSN: 2334-0770. URL: https://www.aaai.org/ojs/index.php/ICWSM/article/view/7347 (visited on 07/26/2020).

[5] Alexandre Bovet and Hernán A. Makse. "Influence of fake news in Twitter during the 2016 US presidential election." en. In: *Nature Communications* 10.1 (Jan. 2019). Number: 1 Publisher: Nature Publishing Group, p. 7. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07761-2. URL: https://www.nature.com/articles/s41467-018-07761-2 (visited on 05/19/2021).

[6] *Commons Compress – Overview*. URL: http://commons.apache.org/proper/commons-compress/ (visited on 05/28/2021).

[7] *Configuration • Akka Documentation*. URL: https://doc.akka.io/docs/akka/2.5.32/general/configuration.html (visited on 01/11/2021).

[8] Helana Darwin. "Doing Gender Beyond the Binary: A Virtual Ethnography." en. In: *Symbolic Interaction* 40.3 (2017), pp. 317–334. ISSN: 1533-8665. DOI: 10.1002/symb.316. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/symb.316 (visited on 07/08/2020).

[9] Adam L. Davis. "Akka Streams." en. In: *Reactive Streams in Java: Concurrency with RxJava, Reactor, and Akka Streams*. Ed. by Adam L. Davis. Berkeley, CA: Apress, 2019, pp. 57–70. ISBN: 978-1-4842-4176-9. DOI: 10.1007/978-1-4842-4176-9_6. URL: https://doi.org/10.1007/978-1-4842-4176-9_6 (visited on 05/21/2021).

[10] Adam L. Davis. "Introduction to Reactive Streams." en. In: *Reactive Streams in Java: Concurrency with RxJava, Reactor, and Akka Streams*. Ed. by Adam L. Davis. Berkeley, CA: Apress, 2019, pp. 1–3. ISBN: 978-1-4842-4176-9. DOI: 10.1007/978-1-4842-4176-9_1. URL: https://doi.org/10.1007/978-1-4842-4176-9_1 (visited on 05/21/2021).

[11] Michael Fire and Carlos Guestrin. "The Rise and Fall of Network Stars: Analyzing 2.5 million graphs to reveal how high-degree vertices emerge over time." In: *arXiv:1706.06690 [physics]* (Oct. 2018). arXiv: 1706.06690. URL: http://arxiv.org/abs/1706.06690 (visited on 06/29/2020).

[12] *flatMapMerge • Akka Documentation*. URL: https://doc.akka.io/docs/akka/current/stream/operators/Source-or-Flow/flatMapMerge.html (visited on 05/28/2021).

[13] Martin Fowler. *Continuous Integration*. May 2006. URL: https://martinfowler.com/articles/continuousIntegration.html (visited on 05/26/2021).

[14] *Gradle | Gradle vs Maven Comparison*. en-US. URL: https://gradle.org/maven-vs-gradle/ (visited on 05/30/2021).

[15] Jan Heichler. "Introduction to BeeGFS." en. In: (), p. 11.

[16] Sandra Henry-Stocker. *Why you should use named pipes on Linux*. en. Jan. 2018. URL: https://www.networkworld.com/article/3251853/why-use-named-pipes-on-linux.html (visited on 05/31/2021).

[17] Carl Hewitt. "Viewing control structures as patterns of passing messages." en. In: *Artificial Intelligence* 8.3 (June 1977), pp. 323–364. ISSN: 0004-3702. DOI: 10.1016/0004-3702(77)90033-9. URL: https://www.sciencedirect.com/science/article/pii/0004370277900339 (visited on 06/15/2021).

[18] Carl Hewitt, Peter Bishop, and Richard Steiger. "A universal modular ACTOR formalism for artificial intelligence." In: *Proceedings of the 3rd international joint conference on Artificial intelligence*. IJCAI'73. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1973, pp. 235–245. (Visited on 06/15/2021).

[19] Victoria Holec and Amy Mack. "Researching Reddit: On the Affordances and Challenges of Social Media Data Collection and Analysis." In: *The Journal of Communication and Media Studies* 5.1 (2020), pp. 15–30. ISSN: 2470-9247, 2470-9255. DOI: 10.18848/2470-9247/CGP/v05i01/15-30. URL: https://cgscholar.com/bookstore/works/researching-reddit (visited on 05/14/2020).

[20] *Homepage - Reddit*. Library Catalog: www.redditinc.com. URL: https://www.redditinc.com/ (visited on 07/07/2020).

[21] *In the Elevator With Reddit CEO Steve Huffman*. en. The Wall Street Journal: Video. URL: https://www.wsj.com/video/series/in-the-elevator-with/in-the-elevator-with-reddit-ceo-steve-huffman/76E47E38-773F-4E77-9EFE-3A6DE234523F (visited on 07/07/2020).

[22] *Introduction • Akka Documentation*. URL: https://doc.akka.io/docs/akka/current/stream/stream-introduction.html (visited on 06/15/2021).

[23] *Introduction to Named Pipes | Linux Journal*. URL: https://www.linuxjournal.com/article/2156 (visited on 05/31/2021).

[24] Mathieu Jacomy et al. "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software." en. In: *PLOS ONE* 9.6 (June 2014). Publisher: Public Library of Science, e98679. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0098679. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0098679 (visited on 05/26/2021).

[25] Johannes Langguth. *UMOD: Understanding and Monitoring Digital Wildfires*. en. Dec. 2017. URL: https://www.simula.no/research/projects/umod-understanding-and-monitoring-digital-wildfires (visited on 05/20/2021).

[26] Dimitrios Michail et al. "JGraph - A Java Library for Graph Data Structures and Algorithms." In: *ACM Transactions on Mathematical Software* 46.2 (May 2020), 16:1–16:29. ISSN: 0098-3500. DOI: 10.1145/3381449. URL: http://doi.org/10.1145/3381449 (visited on 06/09/2021).

[27] *Mission*. en-US. URL: https://www.ex3.simula.no/mission (visited on 05/29/2021).

[28] Benjamin Muschko. *Gradle in action*. OCLC: ocn870650386. Shelter Island, NY: Manning, 2014. ISBN: 978-1-61729-130-2.

[29] *opencsv –*. URL: http://opencsv.sourceforge.net/ (visited on 05/30/2021).

[30] Sahil Patel. "Reddit Claims 52 Million Daily Users, Revealing a Key Figure for Social-Media Platforms." en-US. In: *Wall Street Journal* (Dec. 2020). ISSN: 0099-9660. URL: https://www.wsj.com/articles/reddit-claims-52-million-daily-users-revealing-a-key-figure-for-social-media-platforms-11606822200 (visited on 05/19/2021).

[31] *picocli - a mighty tiny command line interface*. URL: https://picocli.info/ (visited on 05/31/2021).

[32] *Publications*. en-US. URL: https://www.ex3.simula.no/publications (visited on 05/29/2021).

[33] *Pusher Realtime Reddit API*. en-US. Aug. 2014. URL: https://blog.pusher.com/pusher-realtime-reddit-api/ (visited on 07/09/2020).

[34] *r/pushshift - Download-able dumps of comments as they appear at time of post?* en-US. Library Catalog: www.reddit.com. URL: https://www.reddit.com/r/pushshift/comments/dig2u3/downloadable_dumps_of_comments_as_they_appear_at/ (visited on 04/22/2020).

[35] *r/pushshift elastic.pushshift.io is currently blocked*. en-US. Library Catalog: www.reddit.com. URL: https://www.reddit.com/r/pushshift/ (visited on 07/09/2020).

[36] *reddit-archive/reddit Wiki API*. en. Library Catalog: github.com. URL: https://github.com/reddit-archive/reddit/wiki/API (visited on 07/08/2020).

[37] *reddit-archive/reddit Wiki JSON*. en. Library Catalog: github.com. URL: https://github.com/reddit-archive/reddit/wiki/JSON (visited on 07/08/2020).

[38] *reddit.com Competitive Analysis, Marketing Mix and Traffic - Alexa*. URL: https://www.alexa.com/siteinfo/reddit.com (visited on 05/19/2021).

[39] *reddit.com: API Dokumentation*. URL: https://www.reddit.com/dev/api (visited on 07/13/2020).

[40] *Resources*. en-US. URL: https://www.ex3.simula.no/resources (visited on 05/27/2021).

[41] *sbt - The interactive build tool*. URL: https://www.scala-sbt.org/ (visited on 05/30/2021).

[42] *sbt Reference Manual — sbt Reference Manual*. URL: https://www.scala-sbt.org/1.x/docs/ (visited on 05/30/2021).

[43] Daniel Thilo Schröder, Konstantin Pogorelov, and Johannes Langguth. "FACT: a Framework for Analysis and Capture of Twitter Graphs." In: *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. Oct. 2019, pp. 134–141. DOI: 10.1109/SNAMS.2019.8931870.

[44] *scopt/scopt*. original-date: 2012-03-18T19:27:17Z. May 2021. URL: https://github.com/scopt/scopt (visited on 05/27/2021).

[45] Philipp Singer et al. "Evolution of reddit: from the front page of the internet to a self-referential community?" In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14 Companion. Seoul, Korea: Association for Computing Machinery, Apr. 2014, pp. 517–522. ISBN: 978-1-4503-2745-9. DOI: 10.1145/2567948.2576943. URL: https://doi.org/10.1145/2567948.2576943 (visited on 07/06/2020).

[46] *spray/spray-json*. original-date: 2011-05-06T21:31:38Z. May 2021. URL: https://github.com/spray/spray-json (visited on 05/28/2021).

[47] Sho Tsugawa and Sumaru Niida. "The impact of social network structure on the growth and survival of online communities." In: *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ASONAM '19. Vancouver, British Columbia, Canada: Association for Computing Machinery, Aug. 2019, pp. 1112–1119. ISBN: 978-1-4503-6868-1. DOI: 10.1145/3341161.3343526. URL: https://doi.org/10.1145/3341161.3343526 (visited on 06/29/2020).

[48] Scott Wares, John Isaacs, and Eyad Elyan. "Data stream mining: methods and challenges for handling concept drift." en. In: *SN Applied Sciences* 1.11 (Oct. 2019), p. 1412. ISSN: 2523-3971. DOI: 10.1007/s42452-019-1433-0. URL: https://doi.org/10.1007/s42452-019-1433-0 (visited on 05/14/2021).

[49] Tim Weninger, Xihao Avi Zhu, and Jiawei Han. "An exploration of discussion threads in social news sites: A case study of the Reddit community." In: *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*. Aug. 2013, pp. 579–583. DOI: 10.1145/2492517.2492646.

[50] Andy B. Yoo, Morris A. Jette, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management." In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Gerhard Goos et al. Vol. 2862. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer

Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-20405-3 978-3-540-39727-4. DOI: 10.1007/10968987_3. URL: http://link.springer.com/10.1007/10968987_3 (visited on 05/26/2021).

[51] Savvas Zannettou et al. "Who Let The Trolls Out? Towards Understanding State-Sponsored Trolls." In: *Proceedings of the 10th ACM Conference on Web Science*. WebSci '19. New York, NY, USA: Association for Computing Machinery, June 2019, pp. 353–362. ISBN: 978-1-4503-6202-3. DOI: 10.1145/3292522.3326016. URL: https://doi.org/10.1145/3292522.3326016 (visited on 05/19/2021).