The Big Digger & Puzzler System for Harvesting & Analyzing Data from Social Networks

Einar J. Holsbø, Phuong H. Ha, Otto J. Anshus
University of Tromsø
{einar.j.holsbo, phuong.hoai.ha, otto.anshus}@uit.no

Abstract

The Big Digger & Puzzler is a distributed system for harvesting and analyzing data from social networks. The system is distributed on users' PCs around the world. A Digger collects data as specified by the user; a Puzzler analyzes data collected by the Digger, and can contact other Puzzlers to request analytics data on-demand. Puzzlers don't share the raw data with each other, only analytics: this cuts down on bandwidth- and storage requirements, and effectively distributes the compute workload across users interested in approximately the same topics and analytics. On a large enough scale, enough relevant analytics will be available to serve many different requests, statistically speaking.

1 Introduction

Big Data is typically characterized by large values along three dimensions: volume, velocity, and variety. Online social networks produce Big Data: The volume is large, rapidly growing, and frequently updated and accessed. The data has large variety, representing a range of information including text, images, and video. The data is created and contributed to by billions of users around the world: Twitter users produced "a billion Tweets every four days" in early 2012 [2]; there were over a billion monthly active Facebook users in early 2013 [5]; and there are no statistics about how many people use implicit online social networks (e.g., online forums, diverse comment sections), but it is reasonable to assume many millions. Most of the data is collected, stored, and partly made available by a few large companies.

Such data is a rich source for journalists, social scientists, businesses, etc., explicitly and implicitly: IBM showed how steampunk is the coming thing in fashion by use of their trend analysis system[12], and their Smarter Cities project attempts to improve infrastructure flow by integrating social media data, traffic reports, weather data, and more, into a comprehensive view of the current traffic [4]. Google offers startlingly accurate flu statistics based on how people are searching [7]. President Obama based his 2012 re-election campaign heavily on gathering the data and doing the math; in fact, both sides were scouring the Web, doing their utmost to glean as much information as possible about the electorate [1, 3].

This paper was presented at the NIK-2013 conference; see http://www.nik.no/.

There are several reasons why a do-it-yourself analyst would have trouble similarly leveraging these data sets: (i) the data may be in private control, which may restrict their availability; (ii) their size and remote location may pose latency, bandwidth and storage challenges; (iii) the volume and complexity of the data may be beyond what a single machine can analyze, and it can at least be hard to do so interactively fast; and (iv) it may be hard to tell how to interpret and analyze the data.

The Boston Marathon bombing is a fairly recent event that caused large Twitter activity. Twitter allows you to stream data produced right now in a continuous stream, but if you want something from earlier, you have to use the restricted REST[14] interface, which limits you to some small number of lookups per day. Let's say that in the time following the bombing, a journalist opens a Twitter stream on relevant keywords and starts collecting several gigabytes (GBs) of data about the event. The data is processed with MapReduce on a cluster to look for interesting patterns. Although interesting information can be found in data from right before and during the explosion, the deadline for the news clip is in a few minutes, so there is no time to wait for the bandwidth-limited on-demand harvesting. If other people had harvested data about the event during the day, they could already have done some analytics that could be almost instantly used for the on-demand analytics.

The Big Digger and Puzzler system presented in this paper is an infrastructure enabling individual users to harvest and analyze data from social networks on their own PC. The Digger part of the system collects data from social networks according to the user's specifications. The Puzzler then does analytics on the collected and locally stored data. The Puzzler can also locate and contact other Puzzlers, and receive analytics to use to enhance the local analytics. This is done in a *symbiotic* way. There is no central control of- or coordination between Puzzlers.

Experimental evaluations of a system prototype run on an Apple Mac Mini and a cluster of 57 Dell workstations show that the system achieves fair lookup scalability (increasingly large networks require only logarithmic increase in message hops), the Digger consumes little bandwidth (worst case 146kilobytes per second (KB/s)), and the cost of doing symbiotic analytics is very low at 0.8KB/s network bandwidth usage, and $<$ 10% memory- and cpu utilization. The system prototype includes data harvesting, simple analysis of collected data, and a network for sharing analytical results.

The rest of this paper runs as follows: Section 2 explains the assumptions, concepts, and architecture of this system. Section 3 describes the design and implementation of a prototype system. Section 4 outlines the experiments performed to evaluate the prototype, while section 5 provides a discussion of the experimental results and the system in general. The related work is discussed in section 6, and section 7 concludes.

2 Concepts and Architecture

A data harvester should be *accurate*, be *flexible*, and have *low latency*. Accurate in providing the right kind of data; flexible to allow for on-demand harvesting, to reduce the need for planning and pre-fetching. Finally, it is naturally desirable that data is provided with low latency. Having all three of these properties implies owning the data, thus being exempt from retrieving it. If the data needs to actually be fetched from elsewhere, the harvester should have a high bandwidth, low latency access path into the social network. If money is no issue, data can be purchased from the social networks, but these approaches are surely unfeasible for many users.

One of the ideas behind the Big Digger and Puzzler system is for users to pool

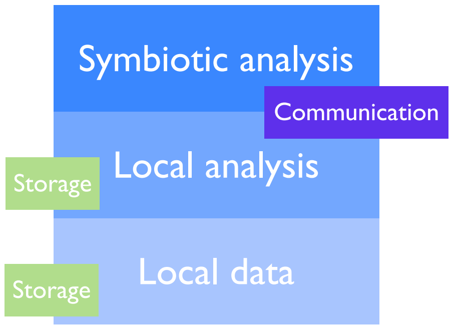


Figure 1: The system architecture.

their resources in order to achieve a tolerable trade-off between flexibility, latency, and accuracy without paying for it. The system lets people gather the data they want, and perform whatever analysis they want, whenever they want to do so. In order to increase bandwidth, many users harvest data. This is done with no coordination or assignment of what data a user should collect. The users share their analytics, if no other user has done relevant analytics, the only option is to wait for the on-demand slow harvesting to provide data. The volume of this data may make for limited local results. However, as the user base grows, there will statistically be more and more analytics available that can be combined on-demand. This can be leveraged for low latency analytics, as well as for lowering the uncertainty in the local analytics results.

Having many Diggers around the world will provide for a large bandwidth from social networks. The data from the social networks will be fragmented and most likely partly duplicated many times onto many PCs, which enables sharing of work load by doing local analytics. Sharing the results of the analytics instead of sharing social network data between users reduces the bandwidth need. Latency is reduced because relevant analytics have already, statistically speaking, been done elsewhere. Robustness is increased because analytics are available from many sources.

The Digger & Puzzler system architecture comprises three abstractions (cf. Figure 1): the local data, the local analytics, and the symbiotic analytics:

- **The local data abstraction** defines the functionalities of locating, gathering and locally storing data from social networks. The user specifies what should be collected, the frequency of collection, where to collect the data from, and how much of the data should be stored. None of the local data is shared with any other user of the system.

- **The local analytics abstraction** defines the functionalities of doing local analytics, and storing the results locally. Information is extracted from the local data by means of some user-defined computation, or some other kind of data processing. The information is computed, stored, and used locally, but also made available for symbiotic analytics. The user presumably needs to possess domain knowledge of the local data to produce high-quality analytics.

- **The symbiotic analytics abstraction** defines the functionalities of locating and gathering remote analytics results, and doing symbiotic analytics. Through this abstraction, the user can request analytics results from other users, and will by the same token share their own results. This abstraction is at the minimum the union of results from the local analytics of other users with relevant results.

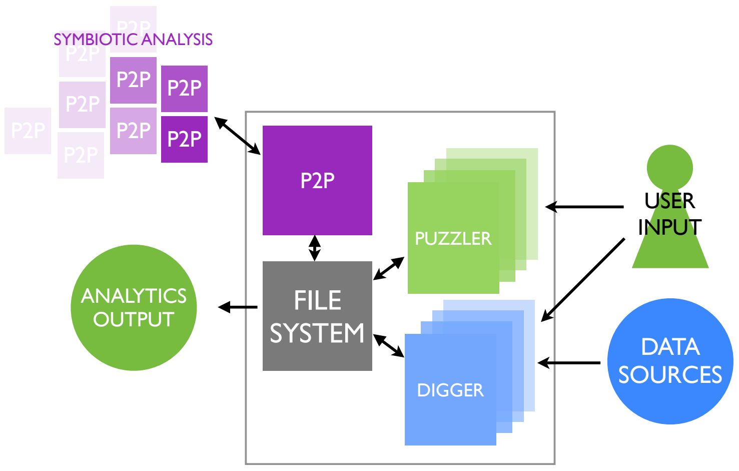


Figure 2: System design.

3 Design and Implementation

The design is comprised of multiple processes encompassing the functionality of the architecture, see figure 2. Diggers provide local data by gathering data and writing it to disk; puzzlers provide local analytics by reading the local data, performing computations, and writing results back to disk. Symbiotic analytics are provided by contacting other users by means of a peer-to-peer (P2P) network. Finally, the user reads analytics results as stored on disk.

Diggers

Diggers realize the local data abstraction: conceptually, each digger is an autonomous process that produces data. A digger can produce data in many ways, including crawling the web, polling the Facebook social graph API, fetching data from some local application, or producing data of its own accord.

A prototype digger was made for the Twitter social network. Topics to dig for are defined as a Python class with a topic name, a language, and relevant keywords that should be submitted to Twitter for monitoring. For e.g., the goat herding topic would be named “goatherding”, and would perhaps monitor the keywords “goats”, “goat herding”, “goat feed prices”, etc. Its language would be “en” for English, which would cause the Digger to discard Tweets that don’t specify their language as “en”.

A Digger reads its topic from a configuration file, and a request is sent to the Twitter stream API[1]. A stream is opened, and all data received over this stream is stored on disk. Twitter will send data as fast as suits Twitter, and the receiving application is responsible for being able to process the incoming data quickly enough. One stream is allowed per Twitter account, so if a user wants to run several streams in parallel, several accounts are needed.

All tweets received are stored in files per topic, full date, and hour of the day. For instance:

[path to data directory]/goatherding/2013-06-01/15

is the file containing all tweets on the topic of goat herding made between 15:00 and 15:59 GMT on the 1st of June, 2013.

[1]https://dev.twitter.com/docs/streaming-apis

Puzzlers

Puzzlers realize the local analytics abstraction. Each puzzler is an autonomous process that processes the harvested data, producing analytics. The output is assumed to be associated with enough semantic meta-data to be understandable when shared.

A prototype Puzzler was made to analyze the data from the prototype Digger. The puzzler applies a polarity classifier, which attempts to classify a message as either *positive* or *negative* in sentiment. For example, a positive message might be “my goats are really basking in this fine weather”, while a negative one might be “working with these goats all day has ruined my best church going clothes”. This implementation features a Bayesian spam filter[24] for polarity classification. The classifier outputs a negativity indicator from 0.0 to 1.0. This provides a mechanism for dealing with the fact that not all texts are polar (“there are five goats” does not imply sentiment). Messages in the lower 33rd percentile are counted as negative; messages in the upper 33rd percentile are counted as positive. Everything between is defined as neutral.[2] The aggregate classification results are calculated on a per-file basis and stored as per the scheme above, with a prefix describing the analytic’s name (“polarity”). The training data for the spam filter is built and pre-processed as described in [17]: in broad terms, Twitter searches based on emoticons (e.g. “:)”, “:(”, etc.) are used as noisily labeled training data.

Symbiosis

The symbiotic analysis abstraction is realized through an unstructured P2P overlay network, a client for which runs as a separate process with access to the local analytics.

A participant in the symbiotic analysis joins the P2P network by contacting a peer that is already participating in the network and using this peer as an entry point. This is done by either knowing of such an entry point beforehand, or by contacting a tracker that knows of possible entry points.

Once the network is joined, connections to other peers are created semi-randomly by a pull-based approach: the newly joined peer obtains a list of neighbors-of-neighbors to the entry point and randomly chooses a few of these to have as its own neighbors. A peer tries to keep its number of neighbors in a set interval, for e.g. between 5 and 15. If there are too many neighbors, a few are dropped; if neighbors are unresponsive to periodic ping messages, they are dropped; if there are too few neighbors, some new ones are pulled in from two hops away as described above. NB that the neighbor relation is not mathematically symmetric: a peer’s neighbors do not necessarily know of said peer, which results in a directed network graph.

To locate peers that may have relevant analytics, a recursive flood search is performed with a list of topics that the searching peer is interested in: the searching peer sends this list of topics to its neighbors along with its contact info. A hop count is attached to the message to prevent the network buckling under infinite messages. The receiving peers check whether they have analytics relevant to the search, and reply directly to the searching peer by means of the provided address. A separate channel can then be opened to negotiate for- and transfer analytics. If the hop count of a received message is > 0, the system forwards the message to all neighbors after decreasing the hop count by 1. The searching peer specifies how long it is willing to wait for responses, giving up if none arrive within this interval. Replies that arrive after this time are simply discarded.

The prototype implementation searches for topics automatically every so often based

[2]NB that these limits are chosen arbitrarily; tuning the classifier is outside the scope of this work.

on the topics about which the user has already gathered data. The prototype is primarily implemented in Python, it relies on three outside sources for code: the Python module *networkx*[3] used for graph simulation; the Python module *tweepy*[4] used for interfacing with the Twitter application programming interface (API); and the JavaScript library *D3*[5], used for visualizing the overlay network during tests and monitoring.

4 Experiments and Results

A set of experiments was conducted to measure the performance characteristics of the prototype, documenting the scalability of the peer-to-peer overlay network, the Digger bandwidth usage, the Puzzler resource footprint, and the resource cost of sharing data symbiotically.

The following metrics were used: central processing unit (CPU) utilization, memory utilization, classifier throughput, network bandwidth usage, and average path length.

- *CPU utilization* is the amount of time a process spends running on the CPU. If the process has been running for 500 milliseconds the last second, the CPU utilization is 50%. NB that CPU utilization is measured *per core*: on a four-core system, the maximum attainable CPU utilization is defined as 400%, not 100%.

- *Memory utilization* is the percentage of the total addressable memory space that a process is allocated. Memory- and CPU utilization are both measured using the external monitoring tool *ps* for Unix-based systems. This tool is accurate enough to for ballpark figure.

- *Network bandwidth usage* is the amount of data transferred over the network per time unit. It will be measured in KB/s by measuring the total amount of data transferred using the Unix network monitoring tool *nettop*, which monitors network usage per process. We divide this measure by the wall-clock time passed to reach KB/s.

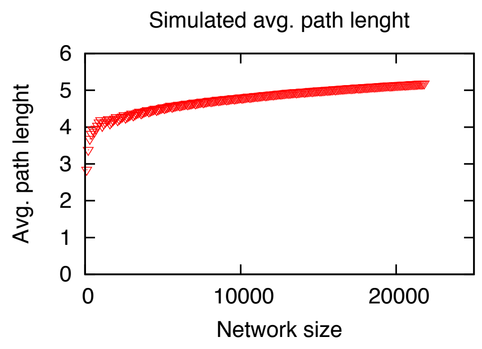
- *Classifier throughput* is defined here as the amount of raw data a classifier can at best process. This is measured in KB/s similarly to bandwidth usage.

- *Average path length* in a directed graph is the average number of jumps from a given node to all other reachable nodes in the graph, for all nodes. This has been measured with the Python module *networkx*.

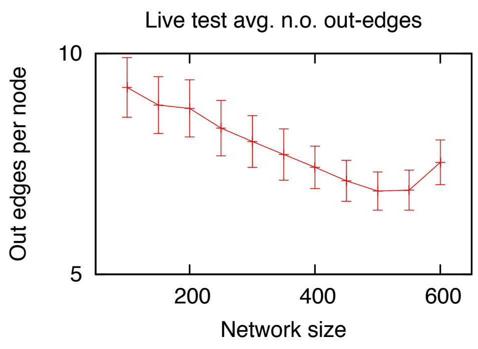
Measurements were made on an Apple Mac Mini with an Intel Core i7 processor, 16GBs of random access memory (RAM), and running 64-bit OS X version 10.8.2. In the data sharing CPU utilization- and overlay live test benchmarks, measurements were made on a cluster of 57 DELL Precision WorkStation T3500s running 64-bit CentOS 6.4 (kernel 2.6.32-358.11.1.el6.x86_64) on Intel Xeon E5520 processors with 12GBs of RAM.

Overlay scalability The first benchmark measures the viability of a basic flood search in terms of how far, on average, a message has to travel to reach all of the network. The benchmark was performed by simulating the random neighboring scheme as outlined in Section 3 for increasingly large networks, and measuring the average path length.

[3]http://networkx.github.io/
[4]http://tweepy.github.io/
[5]http://d3js.org/



(a) Average path length



(b) Average n.o. out-edges per vertex

Figure 3: Overlay network characteristics

Figure 3a shows the average number of hops a message must make to reach every node from every other node in the overlay graph. For a typical situation with a few thousand nodes, the average path length is around four. The trend seems to be toward logarithmic growth, and adding new nodes does not significantly increase the average path length after a certain point. This is a well-known characteristic of random graphs, see for e.g. [16]; the logarithmic increase in path length is known as the *small world phenomenon*. The plotted network had an average neighbor count of 25. This number will obviously influence the average path length, and will also influence how many messages are generated per lookup.

The second benchmark performs a live test of the overlay by running the symbiosis experiment described farther below for increasing network sizes, and having each peer log its number of neighbors (i.e., out-edges) periodically.

Figure 3b shows the overlay network growing sparser, indicating that peers drop neighbors under stressful load. This is corroborated in figure 4b below, where we see an initial higher load, which evens out after a while once the machines find their comfort zones.

Digger bandwidth usage The Digger's network bandwidth usage was measured by opening a very general search stream to Twitter. The Twitter stream API sends messages informing you how many tweets were dropped due to Twitter's limit on how much you are allowed to stream. These messages denote that the stream is transferring data as fast as it's going to. Such a search was run overnight for 12 hours.

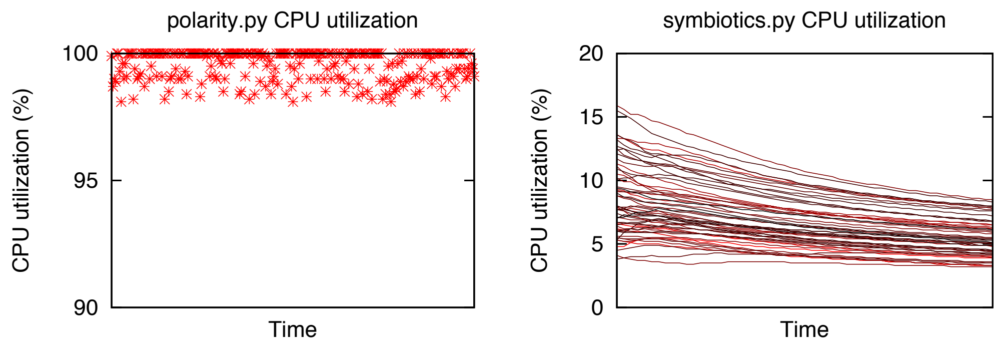
The results show that the maximum attainable download bandwidth usage before reaching Twitter's cap is $\sim$ 146 KB/s, on average.

Puzzler resource footprint This benchmark was performed by feeding 11 GBs of pre-harvested data through a Puzzler process and measuring throughput, memory utilization, and CPU utilization.

Figure 4a shows CPU utilization at more-or-less 100% for the duration of the experiment. Memory utilization, not pictured, was at a constant 0.6%.

The puzzler processed all of the data in $\sim$ 790 seconds, which makes the throughput $\sim$ 14600 KB/s.

Cost of sharing data for symbiotic analytics This benchmark was performed on the 57-node cluster described above. Many access patterns on the internet, including the



(a) Puzzler (b) Symbiotic process

Figure 4: CPU utilization over time

popularity of websites, have been shown to fit Zipf-like distributions[9]. With this in mind, each node was assigned to serve four out of 60 random topics, according to a Zipf distribution. Each node then issued lookups every few seconds on similarly selected topics for the duration of the experiment (4 hours).

Figure 4b shows CPU utilization over time for all the machines in the experiment. The darker the line, the more popular topics the machine in question hosted. The CPU usage is low at $5-10\%$, and it doesn't look as though hosting more popular topics affects load one way or the other, which suggests that the cost is not in transferring analytics, but in serving lookup-related requests, which is more closely tied to number of in-edges. Memory usage, not shown, stayed at 0.3% for the duration of the experiment.

Average bandwidth usage was measured to ~ 0.8 KB/s, both down and up, on the Mini.

5 Discussion

The system design allows the implementation to keep different parts of the system parallel and mostly independent, which allows easy be extension to support different diggers and puzzlers.

With regards to **overlay lookup scalability**, the experiments show that up to about 22000 peers, 5-hop messages will reach most peers most of the time. However, if a flood search keeps a hop count of 5, and each peer keeps 25 neighbors, a single lookup will generate 25^5 messages. The pigeonhole principle tells us that most of the messages will be repeated messages that nodes have seen before ($25^5 >> 22000$). In general, a message will generate $O(2^n)$ messages, with n being the number of hops, which obviously gets expensive fast. A method for making a flood search significantly cheaper is to have the searching peer do an iterative graph walk instead of a recursive flood. It is also possible to have nodes drop messages they have already seen.

The graphs generated for the lookup scalability experiment exhibited a very high in-degree for nodes that joined the network early. This is because a node that has been in the network for a longer time has had more opportunities to be selected as a neighbor. This is likely to fix itself (see figure 3b) because neighbors are dropped if they don't respond fast enough. A peer with high in-degree in the real world would thus be one with the resources to respond more rapidly.

The experiments show that **bandwidth usage of a Digger** on a Twitter stream is low. Consequently, many diggers can be run concurrently from a single PC (if many Twitter accounts are available).

The **cost of symbiotic analytics** is very low. The bandwidth usage is low primarily because the analytics data volume is low. If instead the raw data collected by diggers were to be shared, significantly higher bandwidth usage can be expected.

The implementation has scalability issues, most notably the inability to handle communication (remote procedure calls) efficiently enough. However, the experiments do not seem to suggest fault in the architecture or design.

The **resource footprint of a Puzzler** running at full pelt is small in memory usage, but large in CPU utilization. However, the throughput of the Puzzler is much higher than the speed at which a Digger can fetch data, which means that several puzzlers can be run without a problem. NB that as the implementation is done in Python, the puzzler effectively runs as a single thread on a task that is embarrassingly parallel. NB also that the Bayesian classifier has not been tested for accuracy on manually labelled test data. This should be done before the system is put to actual use.

Disk input/output (IO) is not measured. However, the disk IO of a Digger can be inferred from the amount of data received from the social network (approx. 145 KB/s). The data is buffered in memory and written once every 5 minutes, so the disk IO is sparse. A Puzzler uses the disk by a similar pattern only for reads, polling every now and again for new data to puzzle over.

The system has no policies or mechanisms to ensure the quality of the analytics data from other users. However, the system does not mix outside data with the data from the local analytics, so the user is at least guaranteed to not have the local data poisoned. A group of users who trust each other can set up their own installation of the system and agree upon a common goal and common methodology for quality assurance.

Apart from the question of whether other users have performed analytics correctly, there is also a question about whether the analytics performed by other users will prove useful, i.e., has anyone performed interesting computations on the right kind of topic. As mentioned above in section 4, the distribution of interests is likely to be Zipf-like, some topics ubiquitous, some obscure. This is likely to be true for analytics performed as well (as a sort of second-order interest). Take for e.g. Twitter: The aggregate number of messages about Rihanna is probably freely available, whereas the n.o. messages in Standard Written English vs. other dialects about leveling your turntable is probably hard to come by.

Were very many users to adopt the system, more obscure analytics are increasingly likely to to be available according to the law of large numbers, but it is an open question how many armchair analysts it is possible for such a system to attract. Perhaps the system should allow Joe Average to participate in the system not only by purchasing analytics with other analytics, but by providing compute time or Digger bandwidth that other systems can use in exchange for access to the symbiotic knowledge pool.

If there are no peers in the network, or if no peers have useful information, the system still makes forward progress, though slower and possibly less accurate, through the local harvesting and analytics.

The measure of success in a system such as this is adoption of the system itself, a more polished implementation of this system should be made available to the public to gauge whether the idea has real merit.

Security issues are largely ignored in the Big Digger and Puzzler system. The P2P network will make no guarantees whatsoever about anything.

6 Related Work

The Great Internet Mersenne Prime Search[8] has been a **collaborative and volunteer** approach to computing since 1996. Users lend their spare computation cycles toward finding new Mersenne prime numbers. The Berkley Open Infrastructure for Network Computing (BOINC)[10] is a platform for large-scale public resource/grid computing used in “@home” computational projects. Both approaches have central control over what should be computed and how. The Big Digger & Puzzler system differs from these in in its complete lack of coordination: users do their own digging and puzzling, and may just happen to benefit from others doing the same.

Fjukstad et al.[15], do collaborative weather forecasts according to a model similar to that of the Big Digger & Puzzler system. One difference is that the Big Digger & Puzzler lets the user customize the digging out of data. Another difference it that it does not prescribe which computations (analytics) should be performed on which data. It is entirely up to each user to choose which analytics to do, and how to do them.

Sentiment classification is to classify documents by overall sentiment, e.g., whether the text is positive or negative toward its subject matter (also known as polarity classification). Pang, et al., [22], did sentiment classification in movie reviews, showing that standard machine learning techniques greatly outperform human guesswork-based approaches. Go, et al., [17], did sentiment classification in Twitter, identifying a novel approach towards building training data quickly, to which the classifier in this project owes its existence.

For **user behavior analysis**, Benevenuto et al. [11] gathered click-stream data from several social networks. They found that users spend most of their time browsing other users. Nazir, et al. [21], analyzed user interactions in Facebook apps, finding that users interact over large time frames and large geographical distances.

For **social graph topology analysis**, Mislove et al.[20] examined the network graphs of four different social networks and found several similarities to real-world social networks. Kwak, et al.[19] did a similar analysis of Twitter and found that Twitter differs from other social networks in that it has a non-power-law distribution of followers. They also show that the majority of Twitter’s trending topics are news stories, and as such shows Twitter’s usefulness as a news dissemination media and potential source for interesting information. Ediger et al.[13] demonstrates a system for finding the more influential actors in the social graph of Twitter, allowing analysts to focus on a smaller data set.

There are two main approaches to **P2P networks**: structured and unstructured overlays. The structured overlay network is designed so that lookups will be routed more directly to the relevant peer by keeping strict rules for neighboring and for where a particular request should be routed. An example is the Chord ring[25] and other distributed hash tables. These arrange peers in a circular structure based on consistent hashing [18]. Lookups are routed by assigning a hash value to the item looked up, and assigning responsibility for this item to the node that has the closest preceding hash value to the item.

The prototype Puzzler featured an unstructured overlay. Unstructured overlays, including Gnutella [6], have a flat structure where neighbors are assigned randomly, or according to some simple heuristic. This makes the network itself scale well, and maintaining routing information is cheap. However, a lookup has to be flooded to everyone (or more realistically, everyone within some number of hops from the requesting peer), which generates orders of magnitude more messages than a structured lookup would. Modern unstructured systems, such as Tribler [23], keep extra routing information in order to route lookups to peers that are more likely to have what is needed.

7 Conclusion

The amount of data created by social networks is huge, and the information hidden in the data is enormous. However, good approaches, systems, and tools for scalable harvesting and analytics are not available to most users and contributors of the social networks. Unless social networks apply a model where users can easily extract and analyze data, a collaborative and symbiotic approach is needed where each user harvests on-demand until the limits set by the social networks is reached.

This paper has described the architecture, design, and implementation of an infrastructure enabling collaborative harvesting and analysis of data from social networks in general, and Twitter in particular. This infrastructure documents that it is realistic, resource-wise, to let users do harvesting and local analytics, and to share the results from the analytics with other users. The symbiotic analytics allow the system to scale well because very little data is exchanged. However, quality assurance of the data is needed if the system is used between users with no trust in each other.

While the experiments documented some scalability issues, we believe these are due to specific flaws of the implementation, not to flaws of the architecture or design. Therefore, an improvement of the communication used by the peer-to-peer network should be sufficient to achieve better scalability as the number of users sharing analytics data increases.

Interesting future work would be the exploration of other data sources for the diggers, developing more advanced analytics for the puzzlers, including using visualization, and applying the system in select usage domains.

8 Acknowledgements

Thanks to Jan-Ove Karlberg for allowing himself to be bribed into proofreading the final draft of this paper, and for his help with setting up later experiments. Thanks also to Michael Kampffmeyer for help with setting up earlier experiments.

References

[1] Inside the secret world of the data crunchers who helped obama win. http://swampland.time.com/2012/11/07/inside-the-secret-world-of-quants-and-data-crunchers-who-helped-obama-win/, 2012. [Online; accessed 30-April-2013].

[2] Twitter blog — tweets still must flow. http://blog.twitter.com/2012/01/tweets-still-must-flow.html, 2012. [Online; accessed 30-April-2013].

[3] Wrath of the math: Obama wins nerdiest election ever. http://www.wired.com/dangerroom/2012/11/nerdiest-election-ever/all/, 2012. [Online; accessed 30-April-2013].

[4] Bbc news technology: Why city transport is set to become 'smarter'. http://www.bbc.co.uk/news/technology-22311689, 2013. [Online; accessed 30-April-2013].

[5] Facebook newsroom — key facts. http://newsroom.fb.com/Key-Facts, 2013. [Online; accessed 30-April-2013].

[6] Gnutella - a protocol for a revolution. http://rfc-gnutella.sourceforge.net/, 2013. [Online; accessed 31-May-2013].

[7] Google.org flu trends. http://www.google.org/flutrends/, 2013. [Online].

[8] Great internet Mersenne prime search — GIMPS. http://mersenne.org/, 2013. [Online; accessed 29-May-2013].

[9] ADAMIC, L. A., AND HUBERMAN, B. A. Zipf's law and the internet. *Glottometrics 3*, 1 (2002), 143–150.

[10] ANDERSON, D. P. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (2004), IEEE, pp. 4–10.

[11] BENEVENUTO, F., RODRIGUES, T., CHA, M., AND ALMEIDA, V. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), IMC '09, pp. 49–62.

[12] DAVIS, D. T. Trend spotting moves to a third dimension: Crossing the cultural divide. http://asmarterplanet.com/blog/2013/01/22562.html, 2013. [Online; accessed 30-April-2013].

[13] EDIGER, D., JIANG, K., RIEDY, J., BADER, D. A., CORLEY, C., FARBER, R., AND REYNOLDS, W. N. Massive social network analysis: Mining twitter for social good. In *Parallel Processing (ICPP), 2010 39th International Conference on* (2010), IEEE, pp. 583–593.

[14] FIELDING, R. T., AND TAYLOR, R. N. Principled design of the modern web architecture. In *Proceedings of the 22nd international conference on Software engineering* (2000), ACM, pp. 407–416.

[15] FJUKSTAD, B., BJØRNDALEN, J. M., AND ANSHUS, O. Embarrassingly distributed computing for symbiotic weather forecasts. In *Third International Workshop on Advances in High-Performance Computational Earth Sciences: Applications and Frameworks, IN PRESS* (2013).

[16] FRONCZAK, A., FRONCZAK, P., AND HOŁYST, J. A. Average path length in random networks. *Physical Review E 70*, 5 (2004), 056110.

[17] GO, A., BHAYANI, R., AND HUANG, L. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford* (2009), 1–12.

[18] KARGER, D., LEHMAN, E., LEIGHTON, T., PANIGRAHY, R., LEVINE, M., AND LEWIN, D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (1997), ACM, pp. 654–663.

[19] KWAK, H., LEE, C., PARK, H., AND MOON, S. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web* (2010), ACM, pp. 591–600.

[20] MISLOVE, A., MARCON, M., GUMMADI, K. P., DRUSCHEL, P., AND BHATTACHARJEE, B. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (2007), ACM, pp. 29–42.

[21] NAZIR, A., RAZA, S., AND CHUAH, C.-N. Unveiling facebook: a measurement study of social network based applications. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), ACM, pp. 43–56.

[22] PANG, B., LEE, L., AND VAITHYANATHAN, S. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10* (2002), Association for Computational Linguistics, pp. 79–86.

[23] POUWELSE, J. A., GARBACKI, P., WANG, J., BAKKER, A., YANG, J., IOSUP, A., EPEMA, D. H., REINDERS, M., VAN STEEN, M. R., AND SIPS, H. J. Tribler: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience 20*, 2 (2008), 127–138.

[24] SAHAMI, M., DUMAIS, S., HECKERMAN, D., AND HORVITZ, E. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop* (1998), vol. 62, pp. 98–105.

[25] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review* (2001), vol. 31, ACM, pp. 149–160.