UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Harvest

A Collaborative System for Distributed Retrieval of Social Data

Tor Kreutzer

Inf-3990
Master's Thesis in Computer Science
May, 2012

Abstract

In recent years, social network providers has become one of the largest industries in the world. These networks created a new arena for sharing information over the Internet, and thus changed the way people interact with each other. Hundreds of millions of social network users are updating statuses and sending messages to each other every day. These interactions produce vast amounts of social data. This data is the core of the social network providers business model, and it is sold to large companies to perform personalized advertisement, brand monitoring and viral marketing. The price of this data can be intimidating, and some might be unable or unwilling to pay for it because of its price. If the data was freely available, research that could benefit from this data would be derived more freely, leading to new knowledge.

This thesis presents Harvest, a collaborative system for retrieving social data. Harvest is a peer-to-peer system consisting of contributing social network users, inspired by public resource computing. Harvest shares social network account-bound resources to retrieve large social data sets. Contribution is achieved by running an application on the contributors computer like other public resource computing system such as the @home systems.

The system implements retrieval of data from Twitter. Experiments on real Twitter data show that the system scales with increased contribution. The data retrieval bandwidth per contributing user is quite low, and the number of contributors needed to achieve a considerably large data retrieval bandwidth is high, but there are no associated financial costs with the system. Harvest would benefit greatly by retrieving data from more sources as this would increase its data retrieval bandwidth, in addition to offer more abundant data.

Acknowledgements

I would like to thank my adviser Professor Otto Anshus and my co-advisers Associate Professor John Markus Bjørndalen and Associate Professor Phuong Ha Hoai for their constructive feedback and support. Their help and advice has been invaluable.

I would like to thank my fellow student Andreas for his constant (and sometimes constructive) criticism, and for six awesome years of study.

Finally, I would like to thank my family, my friends and my beloved girlfriend for their support.

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

1.1 Problem Statements 2

1.2 Motivation 3

1.3 Contributions 3

1.4 Limitations 4

1.5 Lessons Learned 5

1.6 Organization 5

2 Modern Social Networks **7**

2.1 Explicit Social Networks 7

2.1.1 Data Access 8

2.2 Implicit Social Networks 8

2.2.1 Data Access 8

3 Related Work **11**

3.1 Data Mining 11
3.1.1 Twitter 11
3.1.2 Facebook 12
3.2 Public Resource Computing 12
3.3 BOINC 12

4 Harvesting the Social Network **15**
4.1 Collaboration 16
4.2 Distributed System 16
4.2.1 Data Retrieval 16
4.2.2 Work Distribution 17

5 Architecture **19**
5.1 Harvest Architecture 19
5.1.1 Peer to Peer System 19
5.1.2 Harvest Nodes 20
5.1.3 System Discovery 22
5.2 Social Networks and Social Data 22
5.3 Consumers 23

6 Design **25**
6.1 Harvest Design 25
6.1.1 Consumer Interface 25
6.1.2 Collection Management 28
6.1.3 Data Storage 29

6.1.4 Collection Interface 29
6.1.5 Network Interface 30
6.2 Tracker Design 31
6.2.1 Tracker Interface 31

7 Implementation **33**
7.1 PyRpc 33
7.1.1 PyRpc Architecture 33
7.1.2 Network Communication Design and Implementation . 34
7.1.3 PyRpc Design and Implementation 36
7.2 PyChord 37
7.2.1 Chord 37
7.2.2 PyChord Design and Implementation 40
7.3 PyRest 41
7.3.1 PyRest Architecture 41
7.3.2 PyRest Design and Implementation 41
7.4 Harvest 43
7.4.1 Consumer Interface 43
7.4.2 Collection Management 43
7.4.3 Data Storage 44
7.4.4 Collection Interface 44
7.4.5 Network Interface 44
7.5 Harvets Tracker Implementation 45

8 Experiments **47**

8.1 Methodology 47
8.1.1 Metrics 48
8.2 Benchmarks 50
8.2.1 Experimental Setup 51
8.3 Performance Benchmarks 51
8.3.1 System Scaling 51
8.3.2 Network Utilization 54
8.3.3 System Activity Distribution 56
8.3.4 Memory Usage 57
8.3.5 CPU Usage 57
8.3.6 Data Loss 58
8.4 Sensitivity Benchmarks 59
8.4.1 CPU Sensitivity 59
8.4.2 Disk IO Sensitivity 60

9 Discussion **63**

9.1 Discussion of Experimental Results 63
9.1.1 System Performance 63
9.1.2 Harvest Node Inactivity 64
9.1.3 Data Loss 65
9.1.4 Application Sensitivity 65
9.2 Motivation for Harvest Architecture and Design 65
9.3 Evaluation of Implementation 66
9.3.1 Motivation for Implemented Artifacts 66

9.4 Evaluation of Problem Statements 68

10 Conclusions and Future Work **69**

10.1 Contributions 69

10.2 Concluding Remarks 70

10.3 Future Work 70

References **73**

List of Figures

4.1 Harvest Overview 15
4.2 Data Retrieval Process 17

5.1 Harvest Architecture 20
5.2 Harvest Node Architecture 21
5.3 System Discovery 22

7.1 PyRpc Architecture 35
7.2 PyChord Architecture 39

8.1 Systems Research Methodology 48
8.2 System Scale Benchmark Results 53
8.3 Network Utilization Benchmark Results 55
8.4 CPU Sensitivity 60
8.5 Disk IO Sensitivity 61

List of Tables

6.1 Harvest RESTful Interface. 26

6.2 Harvest Tracker RESTful Interface. 32

List of Abbreviations

API	application programming interface.
BOINC	Berkley Open Infrastructure for Network Computing.
CPU	central processing unit.
DHT	distributed hash table.
IO	input/output.
KB	kilo bytes.
Kbps	kilo bits per second.
MB	mega bytes.
Mbps	mega bits per second.
P2P	peer-to-peer.
PRC	public resource computing.
REST	representational state transfer.
RPC	remote procedure call.
SN	social network.
SNBW	social network bandwidth.
SNP	social network provider.
TCP	transmission control protocol.

Chapter 1

Introduction

Social interaction has changed much in the last few years. The introduction of web based social network (SN) created a new arena for sharing information over the Internet, and thus revolutionized the way people interact with each other. As a consequence, large amounts of social data are generated within these networks every day. This social data can in turn be collected and analyzed, giving previously intangible knowledge.

Social network provider (SNP) generally have price tags on their collection of social data. A single SN user on his own has only access to a smaller part of this data for free. This makes the access to large amounts of social data limited for those who do not have the means to afford it. This is a problem for individuals interested in large collections of social data, but are unable to extract it, because of the limitations of the social network providers. By introducing a platform where users can collaborate on retrieving data, their combined resources will grant access to a potentially much larger amount of social data without financial cost. This will lower the threshold for conducting research on social data, in turn lead to more results in this field of research.

This thesis presents: *Harvest, A Collaborative System for Distributed Retrieval of Social Data*. This system allows regular SN users to contribute the resources made available from their SN account to collaboratively retrieve large amounts of social data. Contribution is achieved by contributors running client software on their computers. The contributor will then be part of a distributed system retrieving data on his behalf. This software is an application that will run *in the background*, having little affect on other

applications.

Harvest offers an interface for retrieving data using the resources of the contributors. Those interested, the consumers of the data, can define their data set of interest, and request it from Harvest. In turn, Harvest will start a distributed retrieval session, retrieving the requested data from the SNs.

Experiments on real social data are run to benchmark and explore the system behaviour. Public data is retrieved from Twitter using the Twitter REST API[1]. The experimental results show the limitations in data retrieval rate of a single user, and the that the system presented in this thesis is a scalable alternative to retrieving large amounts of data from social networks with no financial costs associated with it. But it is also clear from the experiments that the number of users needed to achieve a relatively large data retrieval bandwidth is very high for it to have obvious practical use.

Should the quotas of the SNPs increase however, and should Harvest include other sources for retrieving data, the above limitations can be surpassed.

1.1 Problem Statements

The social data on today's modern SNs are a part of their business model. As a consequence, their huge amount of data is generally reserved for those with the means to afford it, for instance large research projects and advertisement companies. The single SN user is limited to only a handful of this data within some time frame with no cost. This means he has no chance to retrieve large amounts of data on his own from these networks without having to pay for it. It would be interesting to see the performance of many users as part of a system where they could collaborate with many others, combining many hands to retrieve larger amounts of data without cost.

This thesis proposes the following hypothesis:

> **Social data can be retrieved from social networks in a scalable manner, increasing in data retrieval rate with an increasing number of contributing users, using a collaborative system.**

[1] Twitter REST API - http://dev.twitter.com

Following the hypothesis above, thesis states the following:

> **Collaborative retrieval of social data using such a system will avoid the financial costs generally associated with such data extraction.**

The statement and hypothesis is evaluated through Harvest. The system provides an interface for collaborative retrieval of large amounts of social data. The architecture, design, and implementation of Harvest is evaluated on its ability to scale with increasing contribution, and experimental results are discussed in Chapter 9.

1.2 Motivation

Social data has been useful in research topics such as social network analysis and sociology for a long time. Many interesting references on such topics can be found in [14].

Another field of research that greatly benefits from the modern social networks is *information diffusion*; the theory on how, why, and at what rate new ideas or actions spreads through communication channels [13]. This area is extensively researched with relation to viral marketing and epidemiology [9, 11, 4].

With the new modern SNs, the size of this data is in a totally different magnitude than earlier, and this makes such work much more exiting than before. The motivation for this work is to grant access to large data sets from SNs to those who are interested but discouraged by their financial costs. With more accessible data, new and innovative work and research could spawn more freely, leading to applicable results in many fields.

1.3 Contributions

The following scientific contributions are made:

- The architecture, design and implementation of a scalable collaborative system for retrieving social data.

- Experimental results showing properties of the system.

The following artifacts where also developed along with this thesis. Regarding the artifacts that are *open source*. They were developed by the author during this thesis, and later opened to the public. Their entire development is done by the author.

- Harvest, an implementation of a scalable collaborative system for retrieving social data.
- An open source Python Chord protocol module.[2]
- An open source Python RPC module.[3]
- An open source Python module for building RESTful interfaces.[4]

1.4 Limitations

In the work on this thesis, some areas are not taken into account.

Harvest does not take into consideration the side effects of arbitrary disconnects from the system. Although the system overlay network handles arbitrary node failures, the system itself does not handle the data loss that comes with node departure. When Harvest nodes retrieve data, this data will not be available if the node either disconnects or fails. To avoid this behaviour, a separate scheme for either replication of retrieved data, or overlapping of data retrieval must be designed.

There has been no focus on security aspects of the system. As this is a distributed system running on arbitrary users personal machines there is always the need to look for potential exploits. But as this thesis is about the scale of data retrieval rate based on increasing contribution, this area has been left out.

Harvest does not consider user specific privileges when coordinating data retrieval. As some users might have access to some private material, there is potential for a small optimization by coordinating this.

[2] http://github.com/TnaK/PyChord.
[3] http://github.com/TnaK/PyRpc.
[4] http://github.com/TnaK/PyRest.

Another potential optimization is reuse of data between sessions. This is not handled by Harvest, and could possibly provide huge optimizations for certain requests with overlapping data.

1.5 Lessons Learned

This thesis presents a method for retrieving social data from social networks using collaboration. Using this method, it is possible to retrieve larger amounts of data without associated financial cost, with increasing data retrieval bandwidth based on contribution. It is clear from the experiments that the contribution needed for this to be reasonable is quite high. Combining data retrieval from different social networks would be highly beneficial, as this would both increase the data retrieval bandwidth as well as offering more abundant data.

1.6 Organization

The remainder of the thesis is organized as follows. Chapter 2 will describe preliminary details about the current state of modern social networks. Chapter 3 will discuss related work.

The next chapters will describe the state of Harvest in a dogmatic fashion. The system is described as is, saving the discussion and evaluation for later chapters. Chapter 4 presents the main idea behind the work of this thesis. The architecture of the system presented in the thesis is detailed in Chapter 5. The design is given in detail in Chapter 6. Chapter 7 gives the implementation details of the system.

The system is evaluated in Chapter 8 and Chapter 9 with experiments and discussion. The thesis is concluded in Chapter 10, and outlines for future work is presented.

Chapter 2

Modern Social Networks

People and the relationships between them can be represented as a graph (or network), with individual persons representing nodes in the graph, and the relationships between them as edges in the graph. This is called a *social network*.

The modern meaning of the term SN refers the explicit social networks that belong to web applications of the Internets SNPs such as Facebook, Twitter and Myspace. There are also other, implicit SNs such as web forums and blogs.

2.1 Explicit Social Networks

These web applications create concrete representations of SNs, and map all social interactions between users within them. This creates social networks that are more detailed and more accessible than before, and gives new possibilities for work on such networks since there is almost unlimited amounts of data associated with them.

The amounts of users on these SNs has grown to vast numbers in the past few years. Facebook alone has over 901 million active users [6] as the end of March 2012. The amount of social data created within Facebook was close to the exa-scale around 2010.

This social data is the core of the SNPs business model, and comes with a price tag. To support the cost of providing a social service at this scale, their

social data is sold as a product to those of interest. As a result, not everyone can afford this data.

2.1.1 Data Access

The SNPs generally make their data available through web application programming interfaces (APIs), commonly using representational state transfer (REST), or RESTful APIs. Though many of these APIs are meant for social apps that interact with the networks, some also offer APIs for data harvesting and analysis of social data.

The regular APIs are generally free, but limited by the amount of API calls allowed within a certain time frame. After this time frame has passed, the limit is reset. These APIs are designed for creating web applications and mobile application on top of the SNs, but can be used for data harvesting of small data sets.

Data harvesting APIs give access to more data, but this data is not free, and the price increases with the amount of data requested. For example, the two main providers of twitter data (Gnip[1] and DataSift[2]) have prices ranging up to 15.000$ per month for their service.

2.2 Implicit Social Networks

These are social networks that are created by social behavior on the Internet. For example, if two Internet users discuss the same topic thread in a public forum, there is an implicit social connection between them since they share a common interest on the topic.

2.2.1 Data Access

Implicit SNs were not designed for data mining in the same way as for explicit SNs. Because of this, data retrieval from such networks is rather limited, and

[1]Gnip - http://gnip.com/twitter
[2]DataSift - http://datasift.com

generally done by web crawling techniques, as there are seldom specific APIs for retrieving data from public forums.

Chapter 3

Related Work

3.1 Data Mining

Data mining is the extraction of knowledge from data. In this context the data has origin from social networks; data mining social data. This is usually done by crawling the web, or by public APIs designed for such purposes offered by SNPs. In most of the work done on data extraction there has been an emphasis on the processing of this data and the analysis associated with that processing. This is in most cases different from the work of this thesis as only the method of data extraction is relevant.

Below are work related to data extraction from social networks.

3.1.1 Twitter

In an experiment done by Kwak et. al [10] used a setup of 20 computers that collaboratively harvested the entire Twitter social graph, and executed analysis on that data set. This differs from the work of this thesis in that it was not built as a system for retrieving social data. Rather this was a setup to collect that specific data for the analysis experiment.

3.1.2 Facebook

In another work on social data mining [12], data was harvested from users through social apps developed using Facebook Development Platform[1], including a gaming app. In this experiment, data was extracted from the usage of these applications using different techniques, such as retrieving user information from the Facebook users that

Harvest is similar in the fact that it also uses a SN app for the data retrieval. But the Harvest application is designed explicitly for this purpose, and offers no functionality beyond that. Instead of retrieving info about the users that use the app, Harvest use the quota of the SN account to retrieve public data from other users.

3.2 Public Resource Computing

Public resource computing (PRC) has been a research field since the mid 1990's starting with GIMPS[2] and Distributed.net[3]. There have since then been a number of systems that take advantage of *volunteer computing* [1, 2, 7]. The major difference between Harvest and PRC is how they define shared resources. In PRC, resources is generally referring to computational resources such as central processing unit (CPU) and memory. Although these resources are contributed, Harvest focuses on the ability to retrieve data as its resource. It is this resource that is limited, and that is shared within Harvest.

3.3 BOINC

Berkley Open Infrastructure for Network Computing (BOINC)[4] [1] is a framework for creating large-scale public resource computing system. BOINC takes advantage of the computing and storage resources of participating users to create scalable systems with high demands for computing, storage and communication. Many systems (60+) use BOINC to achieve their computational resources, including SETI@home, folding@home, MilyWay@home to mention

[1] Facebook Development Platform - http://developer.facebook.com
[2] Great Internet Mersenne Prime Search - http://www.mersenne.org
[3] Distributed.net - http://www.distributed.net
[4] http://boinc.berkeley.edu/

a few. These systems have grown largely in popularity the last years, and has proved the effectiveness of a distributed model for performing large scale computing.

BOINC takes advantage of the computational hardware of participating users to perform computation on data. Harvest on the other hand uses the participating users' download limit, or quota, of his SN account. Also, Harvest is all about acquiring data, where as BOINC is a framework for data processing.

Harvest has a different architecture than BOINC. BOINC uses a centralized server [3] for task distribution. Harvest on the other hand is a decentralized system, and handles this task distribution distributed on every participating node.

Chapter 4

Harvesting the Social Network

The limitations of cost (financial) free data retrieval from SNs are described in Chapter 2. The idea behind this thesis is to provide a way to provide higher data retrieval bandwidth using collaboration. This is achieved by a distributed and decentralized system of collaborating peers that combine their resources for the greater good. Illustrated in Figure 4.1, this system is the middle layer between those interested in the data; the data consumers, and the holders of the data; the SNs.

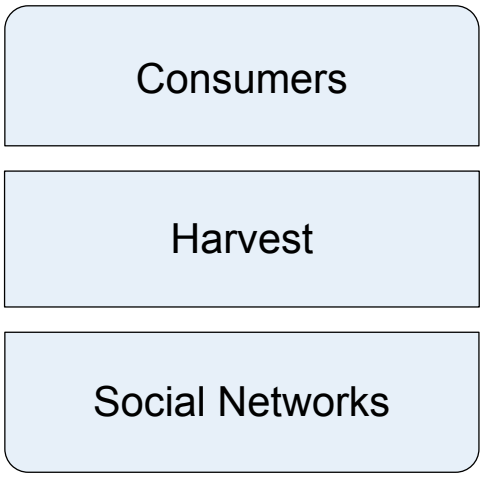

Figure 4.1: *Harvest as the middle layer between the data and their consumers.*

4.1 Collaboration

All users of SNs have certain resources associated with their accounts. The resource of interest is their capacity to retrieve data from a social network; their social network bandwidth (SNBW). As mentioned in Chapter 2 the free APIs of the social network providers are limited by the number of API calls within some time frame. This means that the SNBW of a single SN user is the amount of data he can retrieve within that time frame.

In order to achieve increased data retrieval rate, the SNBW of several users (*contributors*), is combined by having them collaborate on retrieving the social data. With more users, one would achieve higher SNBW, thus getting high network bandwidth. The result is the ability to retrieve large data sets within a reasonable amount of time.

4.2 Distributed System

This collaboration is achieved through a distributed system that connects all contributors together. In order to be granted access to this system, contributors run client software on their computers that are authenticated with their SN accounts. The system handles all communication within the network, and there is no need for manual administration. The core functionality of the system is to coordinate the collaborative data retrieval. This includes harvesting data from the SNs and work distribution.

4.2.1 Data Retrieval

The data of interest is the social data. The information described by SN users on their profiles, their social links (friends and followers), and their status updates. It is this data that is retrieved by Harvest.

To initiate data retrieval, Harvest offers an interface for external users (*consumers*). This is an asynchronous interface, letting consumers request the desired data, check the status of their request, and download the data when the task is completed.

The data retrieval process of harvest is illustrated in Figure 4.2. It starts by a consumer initiating a request for a data set. This requested data set is

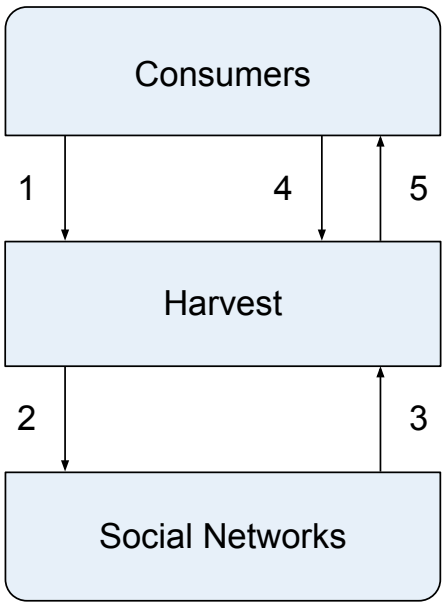

Figure 4.2: *Data retrieval process of Harvest in steps 1 to 5. 1: Request to retrieve data set. 2. Data set requested from social network. 3: Distributed data retrieval from social network. 4: Data requested from Harvest. 5: Requesed data set downloaded and merged from Harvest.*

then converted into SN API calls, and requested from the SN by the contributors. Partial data is then retrieved and stored at each contributor. Upon request from the consumer, the data is downloaded from each contributor, and merged into a final result; the requested data set.

Harvest is is only responsible for the retrieval of the requested data, and does not handle the merging of acquired data. Data is merged by the consumer that has requested this data.

4.2.2 Work Distribution

In order to coordinate data retrieval, Harvest distributes work within the system. The work that is done in harvest is API calls to the SNs, and these must be distributed evenly in order to fully utilize the API call limit of each contributing user. Harvest divides the data request into a number of SN API calls that retrieves that data. These API calls, or tasks, can then be sent to nodes within the network that are responsible for executing that specific

task.

Chapter 5

Architecture

Harvest is a distributed system for retrieving large amounts of data from social networks. The system consists of SN users that run software on their computers, which are the nodes in the system. The illustration in Figure 4.1 from Chapter 4 shows an overview of the main idea of the system. This chapter details this idea into a system architecture, and describes the purpose and functionality of the social networks, of Harvest, and of the consumers.

5.1 Harvest Architecture

5.1.1 Peer to Peer System

Harvest is a distributed and decentralized peer-to-peer (P2P) system of contributing SN users. Each contributor is represented as a node in the system; a *Harvest node*. The architecture is illustrated in Figure 5.1. These nodes present the system equally in terms of interfaces they provide. This means that all requests to Harvest should result in the same result regardless of what node handled the request. This is the intended behaviour, and no guarantees are set to make it absolute.

Harvest users contribute their resources to the system in the form of data retrieval bandwidth. This contribution is achieved by contributors running a Harvest application on their computer. This application acts as a peer in Harvests P2P network, collaborating with other peers within the system to

retrieve data.

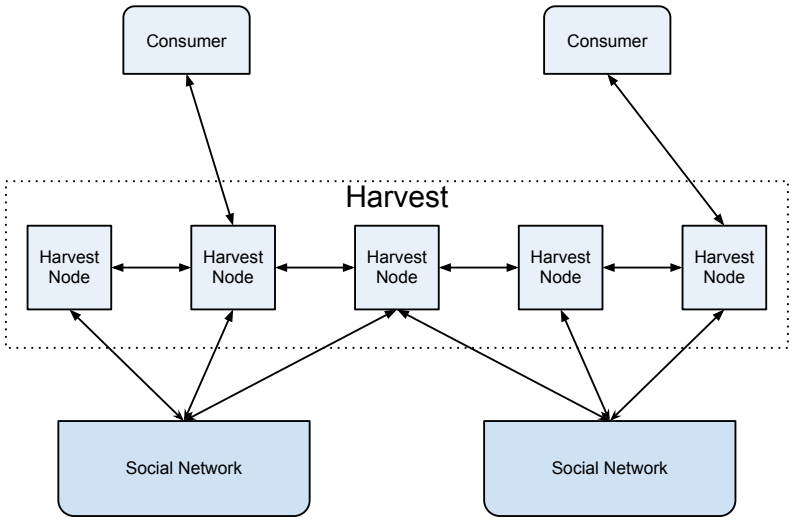

Figure 5.1: *Harvest peer-to-peer system. An elaborated illustration of Figure 4.1. The Harvest nodes within the dotted lines represent the Harvest box in Figure 4.1. The same data retrieval process applies here.*

Harvest is a decentralized P2P system. New nodes can enter the network from any node currently in the system, and will find their place with no manual administration. Likewise, consumers can send requests to any node in the system, and the data retrieval process will be distributed internally within the system autonomously.

5.1.2 Harvest Nodes

The architecture of the Harvest nodes is illustrated in Figure 5.2. The node has a layered architecture, where each layer only communicates with the layer just above or just below it.

The top layer is the consumer interface for the node. This interface handles all interaction with consumers and lets consumers access the data collection management and the data storage of this node. The interface support requesting data harvesting, as well as defining the data to be retrieved. In addition it has methods for downloading retrieved data. Retrieving data is done per node, meaning the entire data for a data retrieval session must be partially retrieved from every node.

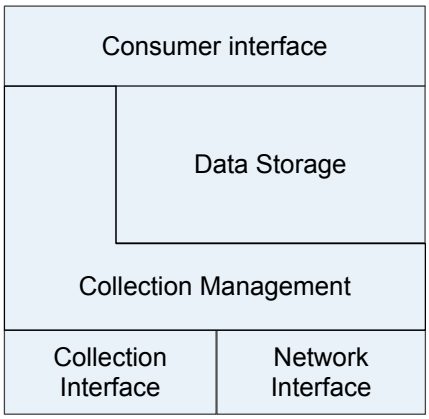

Figure 5.2: *Harvest node architecture. Each layer only communicates to the layer directly above or below it.*

The collection management layer is responsible for the retrieval of the requested data. This includes coordinating collaborative data retrieval, work distribution, and load balancing. The collection management at each node harvests data according to the consumers request utilizing Harvests collection interface, and stores retrieved data using local data storage.

The Harvest nodes data storage persistently stores all retrieved data. Data stored in the data storage is accessible for consumers through the consumer interface. Retrieved data is handed to the data storage by the collection management.

The collection interface directly interacts with the interfaces of the SNs. This is the layer that does the dirty work, and retrieves all the data. It holds specific SN collection interfaces and is authenticated with the SN account credentials of the contributing user.

The network interface layer is responsible for all internal communication between the Harvest nodes, along with maintaining overlay network structure and placing new nodes in the P2P network. The network interface is used by the collection management to perform work distribution between Harvest nodes.

5.1.3 System Discovery

Harvest has a point of entry into the system to simplify system discovery for the Harvest application. Similar to BitTorrent and other file sharing P2P system, a tracker to track nodes currently in the system is used to achieve system discovery. The tracker will try to have a updated list of Harvest nodes in the system, but sets no guarantees of consistency to achieve this.

This tracker is used both by contributors entering the system and consumers finding a system node to request data from. The system discovery is illustrated in Figure 5.3. The tracker contains the address of all nodes in the system. To find an entry point into the system, both consumers and contributors need to contact the tracker and get the list of nodes currently in the system. Any node in the system can then give entry to the system for contributors, and respond to any requests from the consumers.

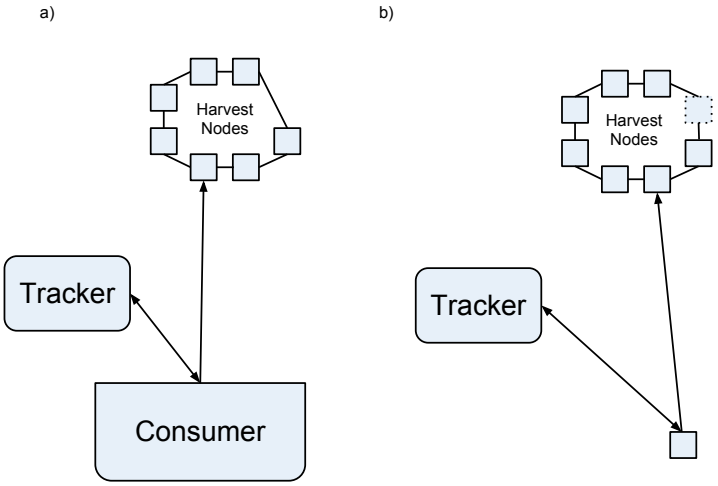

Figure 5.3: *System discovery for consumers and contributors. a) A consumer finds a Harvest node by first contacting the tracker. b) A Harvest node entering the system finds an entry point by contacting the tracker.*

5.2 Social Networks and Social Data

The SNs are holders of social data. This is where social data is created and stored. The public part of this data is offered by Harvest, and contains

information such as user information, social links, and status updates. SNs make their social data accessible through web based interfaces that let users control their SNs accounts and view public data.

Harvest utilizes these web based interfaces for retrieving the public data from the social networks.

5.3 Consumers

The consumers are the external users of Harvest. Consumers are interested in the social data, and will use Harvests consumer interface for acquiring that data.

Consumers interact with Harvest by starting a data retrieval session. Within this session, a consumer can define the data set to be harvested, and acquire it at a later time when it is retrieved. To define the data, Harvest offers a interface to describe a data set. This interface lets consumers define data something like the following: *the information of user x and his followers*, or *the timeline of user y and his friends*. This is greatly detailed in Chapter 6.

Chapter 6

Design

This chapter takes the architecture from Chapter 5 and defines the design for the Harvest system. Both the design of the Harvest nodes and the tracker is detailed.

6.1 Harvest Design

Illustrated in Figure 5.2, harvest has its main responsibilities divided into several parts, or layers; *consumer interface*, *collection management*, *data storage*, *collection interface*, and *network* interface. This section will detail these separate responsibilities, and describe how these layers interact with one another.

6.1.1 Consumer Interface

Harvest provides consumers with a RESTful interface for social data retrieval. This interface is provided by each node within the system. The interface consists of the two RESTful resources: *sessions* resource and *data* resource. These RESTful resources and their supported methods are detailed in Table 6.1.

Resource	Description
GET /sessions	Returns a list of all sessions located at the Harvest node.
POST /sessions	Creates a new session at the Harvest node. Returns the session id for this new session.
GET /sessions/session_id	Returns information about the session at the Harvest node associated with the session id.
POST /sessions/session_id	Upload new collection definitions to the session associated with the session id.
DELETE /sessions/session_id	Delete all session data belonging to the session associated with the session id.
GET /data/session_id	Return all retrieved data from the session associated with the session id.
DELETE /data/session_id	Delete all retrieved data associated with the session id.

Table 6.1: *Harvest RESTful Interface.*

Sessions Resource

The sessions resource defines operations to create and interact with data retrieval sessions in Harvest. Sessions define the context of data retrieval of a data set. The session consists of a session id, and the definition of what data is to be retrieved for that session.

The sessions resource is divided into a sessions collection resource and a session specific resource. The sessions collection resource (*/sessions*) offer methods for view ongoing sessions, and creating new sessions at a Harvest node. The specific sessions resource (*/sessions/session_id*) is accessed using the session id. This resource offer methods for defining data sets to be retrieved, getting status of a given session, and deleting the session entirely.

Data Resource

The data resource is for accessing session specific data. One must specify the data of interest with a session id. This resource offers methods for down-

loading the retrieved data for the given session, and for deleting it.

The data has no timeout associated with it. This means that if the session owner does not delete the data, it will stay there until deleted.

Data Definition Interface

Harvest retrieves data according to the consumers request. These data sets are defined using *collection definitions*. A collection definition is essentially a data structure consisting of the following fields:

- **type_f*
- **user_f*
- *timeline_f*
- *friends_f*
- *followers_f*
- *recursive_range_f*

Some fields are mandatory in order to define a data set. These are denoted by an asterisk. Other fields are optional or situational, depending on other fields.

The *type_f* field indicates what type of data is of interest and is a mandatory field. The type can be one of the following:

- *user_t*
- *timeline_t*
- *friends_t*
- *followers_t*

The type *user_t* defines that user specific data for a given user is to be retrieved. What data is retrieved depends on what SN API is used, commonly including user name, friends count, and description. *The timeline_t* type will

retrieve the timeline, or list of public statuses of a given user. The types *friends_t* and *followers_t* retrieves a list of friends and followers respectively for a given user.

The *user_f* field define what specific user to retrieve data from. This can be specified either by the users alias or SN id. This field is also mandatory.

The *timeline_f* field is linked with the user type, and only has effect when used with that type. If the timeline field is set, then a users timeline will be retrieved in addition to the user info of the given user.

The *friends_f* and *followers_f* fields are also linked with the user type. If the friends or followers fields are set, then the given users list of friends or followers respectively will be retrieved as well.

The field *recursive_range_f* denotes whether the given request should be repeated in a recursive manner, and how many recursive steps to take. For instance, to retrieve the friends and followers of a given users, and then in turn retrieve the friends and followers of those users, the *recursive_range_f* field can be used to achieve this in a single collection definition.

6.1.2 Collection Management

The collection management of Harvest is done on a per session basis. This means that every data retrieval session is treated separately. Whenever a session is started by a consumer via the consumer interface, a session object is created and stored at the node, and a session id is returned to the consumer. This session id will then be used for all future reference to the session.

As mentioned, data retrieval is defined by collection definitions. Collection definitions define data sets in a coarse grained manner. A single definition can define a very large data set, and in many cases the data sets defined by a collection definition will be translated into a lot of SN API calls. In order for the collection interface to retrieve the defined data from the SNs, these definitions must be converted into collection definitions that correspond to a single SN API call.

It is the responsibility of the collection manager to distribute these fine grained collection definitions through the network to achieve collaboration in the system. This in turn means that work for the collection interface is generated at the Harvest nodes, rather than by the consumers.

Work Distribution and Load Balancing

The collaboration of the system is achieved by two techniques: Work distribution, and work stealing. Both of these methods are coordinated by the collection management, by performed by the network interface.

Work distribution is a pushed based approach for distributing collection definitions to other nodes in the network. The collection management receives collection definitions from the consumer interface or by converting coarse grained definitions into fine grained definitions. The definition is run through the network interface to determine if this definition belongs to another node in the system (this is described in a later section). The network interface will send the definition to the collection management of the node responsible for that definition.

If a Harvest node is out of collection definitions but has not reached its data retrieval limit by the SN, the collection management will steal collection definitions from other nodes. The collection management will poll the network for excess work using the network interface. This is a pull based approach, and is an optimization of the work distribution to achieve more even load balancing.

6.1.3 Data Storage

Retrieved data is stored persistently at each node in a file system storage. The collected data is stored immediately after it is retrieved to avoid high memory consumption. This data is made accessible through the consumer interface, and can be downloaded or deleted by the consumer. Data is stored in separate directories for each session, and separate sub-directories for each data type. All individual units of data retrieved from the SNs are stored in a separate file.

6.1.4 Collection Interface

A specific collection interface for each SN handles the retrieval from that specific SN API. Each SN API call is derived from the collection definitions received from the collection management. These fine grained collection definitions correspond to a single SN API call. The data from these API calls

are retrieved and returned to the collection management for storage.

6.1.5 Network Interface

Harvest has an object based network interface for internal network communication. This is achieved using a remote procedure call (RPC) interface. Network nodes are represented as proxy objects. Remote methods are called on these methods to perform the network protocol, and to send data between nodes.

The system uses Chord [15] as the overlay network protocol. Chord is a structure P2P network protocol providing the following features to Harvest (as described in the original paper):

- "**Load balance:** Chord acts as a distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance." [15]
- "**Decentralization:** Chord is fully distributed: no node is more important than any other. This improves robustness and makes Chord appropriate for loosely-organized P2P applications." [15]
- "**Scalability:** The cost of a Chord lookup grows as the log of the number of nodes, so even very large systems are feasible. No parameter tuning is required to achieve this scaling." [15]
- "**Availability:** Chord automatically adjusts its internal tables to reflect newly joined nodes as well as node failures, ensuring that, barring major failures in the underlying network, the node responsible for a key can always be found. This is true even if the system is in a continuous state of change." [15]
- "**Flexible naming:** Chord places no constraints on the structure of the keys it looks up: the Chord key-space is flat. This gives applications a large amount of flexibility in how they map their own names to Chord keys." [15]

By utilizing the Chord protocol, Harvest is a completely decentralized system, with minimal need for manual administration. Chord simplifies overlay structure maintenance by automated failure detection, node entry and node departure.

Work Distribution and Load Balancing

The network interface handles distribution of collection definitions around the network. The interface will determine where the collection definitions belong by performing a Chord lookup based on a key from the collection definition. The collection definitions *user_f* field will be used as key. The Chord lookup will find the Harvest node responsible for the given key, thus responsible for the collection definition.

Work stealing is done using a different approach. The collection management requests excess work to steal using the network interface. The network interface will iterate network as a linked list, stealing excess work along the way up to some limit. This limit is calculated based on the amount of work the stealing node can steal and still be within his retrieval limit. If there is nothing to steal (the stealing nodes target has no definitions himself, or can perform all his definitions within his retrieval limit), the node will continue.

6.2 Tracker Design

The tracker is a simple HTTP server with a RESTful interface. It is essentially a name server containing enough information about each node in the network in order to either join the network in the case of contributors, or send requests to the system in the case of consumers.

6.2.1 Tracker Interface

The tracker holds information about Harvest nodes connected to the system. This information is used by the Harvest application in order to join the distributed system, and also for consumers to locate system nodes in order to start data retrieval sessions and request data. The tracker provides a RESTful interface for this interaction. The interface is described in Table 6.2.

The Interface of the tracker consists of a single resource; the names resource. This resource is divided into a name collection resource (*/names*) and a specific name resource (*/names/name_id*). The names collection resource has methods for listing all names, or nodes, currently in the system and for posting a new entry into the list of names. The specific name resources is used

to get information about certain nodes in the system such as its connection information. It also has a method for removing a entry from the list of names.

Resource	**Description**
GET /names	Returns a list of all Harvest nodes currently on line in the system as a list of name ids.
POST /names	Register as an active node in the system. Return a name id corresponding to the registered node.
GET /names/name_id	Returns information about the Harvest node associated with the name id.
DELETE /names/name_id	Removes the name entry associated with the name id from the tracker.

Table 6.2: *Harvest Tracker RESTful Interface.*

Along with being a single point of entry to the system, it is also useful in other cases. The tracker holds information about all nodes in the system. Using this information, the information of the entire system is easily accessible. This makes it possible to create useful tools and application for monitoring and control the system.

Chapter 7

Implementation

This chapter will present the implementation details of Harvest. The chapter will start by giving a brief architecture, design, and implementation of the most significant artiface modules created in addition to Harvest. Then continue to present the implementation of Harvest itself, and how the aforementioned modules are used to achieve this implementation.

7.1 PyRpc

PyRpc is a RPC module for Python. It is an object based RPC modules handling method calls to be called on remote objects with arbitrary input parameters. Only restriction is that the argument must be serializeable using Pythons pickle module.

7.1.1 PyRpc Architecture

The PyRpc module has a client/server architecture. It consists of proxy objects and remote objects. The proxy nodes act as clients, and remote objects as servers.

Upon receiving a call to a remote method, the proxy node will serialize that method call, send it to the remote node and wait for a serialized return value from the remote object. This return value is then de-serialized and returned to the caller.

Remote objects act as server, waiting for remote method call as requests from proxy objects. The remote objects will receive serial methods, de-serialize them and execute them call locally. The return value is then serialized and returned to the requesting proxy object. This interaction is illustrated in Figure 7.1.

7.1.2 Network Communication Design and Implementation

The network communication between proxy and remote objects are achieved by using high abstraction network interfaces. There are three layers of network abstractions that is used to achieve both network communication as well as serialization and de-serialization of method calls; *streams*, *channels*, and *portals*.

Streams

Streams create a simple file-like abstraction over sockets for sending and receiving data over a network connection. The streams are represented as stream objects. They have a *read/write* interface. Reading and writing from the stream takes input the number of bytes to read or write.

The network communication is done using transmission control protocol (TCP) to achieve reliable data transfer. This is implemented using the TCP sockets of the Python socket library.

Channels

Channels are a higher level of abstraction compared to streams. The channels support the sending of frames over the network. Frames are continuous series of bytes with a header describing the length in bytes of the frame. aChannels have a *send/receive* interface which allow sending and receiving single frames at a time, and supports frames of arbitrary sizes.

The channels are represented as channel objects and are instantiated using stream objects. The channels are implemented using the stream module for network communication, and uses the Python struct module to create a

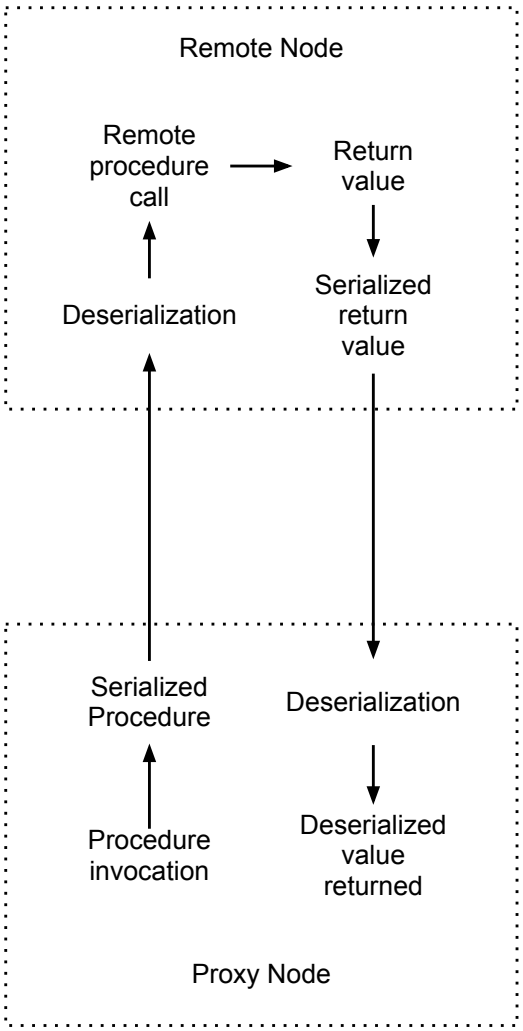

Figure 7.1: *Client server architecture. Proxy nodes send requests to remote nodes in the form of serialized procedures.*

frame header.

Portals

Portals are a higher level of abstraction compared to channels. Portals support sending and receiving of arbitrary Python objects over a network connection. The portals offer *send/receive* interface, and can take an arbitrary number of Python objects as input.

Portals are essentially channels that support serialization of input and output parameters so that Python objects can be sent through them directly. Serialization of Python objects is done using Pythons pickle module.

Portals network communication is implemented using the channel module. Serialized objects are put into frames and sent using the channels send interface.

7.1.3 PyRpc Design and Implementation

The network communication between the proxy and remote nodes are done using aforementioned network interfaces. The serialization is achieved using the Pickle protocol.

Proxy Objects

Proxy objects use portals to achieve network communication and serialization of remote methods.

Proxy objects are initialized with an address to its corresponding remote object. This is used to connect to the remote object when executing remote calls.

In Python, instance methods are attributes to objects. PyRpc takes advantage of this by treating all method calls that are not named attributes of the object as remote calls. Upon such method calls, the proxy object stores the name of the method as well as the arguments of that method in a method container class, connects to the corresponding remote object and sends the method container to the remote object through a portal. The proxy will then

receive the return value from the remote object through the same portal and return it to the caller.

PyRpc supports exception handling of remote method calls. If the remote method should throw an exception, this exception will be the return value from the portal. Exceptions will be raised instead of returned by the proxy object.

Remote Objects

Remote objects are the remote resources that call the method calls locally to produce the results. To achieve this they are implemented as threaded TCP servers taking connection requests from proxy objects. Upon connection, all network communication is done using portals, and the remote objects will start by receiving a method container object from the proxy object. The method name and arguments are then extracted and the named method is called locally

The remote objects are instantiated with an address to set up a TCP server so that proxy objects can connect to them and perform remote calls.

7.2 PyChord

PyChord is an open source Python implementation of the Chord protocol [15]. The implementation is inspired by the Open Chord[1] project, but is a standalone implementation developed as part of this thesis.

7.2.1 Chord

This section give a brief description of Chord, as is heavily based on the original paper. For further details the reader is directed to [15].

The Chord protocol is a structured P2P overlay network protocol designed for distributed hash table (DHT) networks. Chord specifies how to perform key based lookup, how to handle new nodes joining the system, and how to recover from node departure of existing nodes (failure and planned).

[1] http://open-chord.sourceforge.net/

Chord provides fast distributed computation of a hash function mapping keys to nodes. Chord assigns keys to nodes with *consistent hashing* [8] which grants properties such as high probability load balancing, and scalable overhead related to key transfer when a node joins (or leaves) the network.

Chord scalability comes from the nodes needs to only contain a small amount of routing-information about other nodes. This is information is in the form of a lookup table, or finger table, which results in every node in a N-node network needs only know about $O(\log N)$ other nodes.

Consistent Hashing

Using a consistent hash function each node and key is assigned an m-bit *identifier* using SHA-1 [5] as hash function. A nodes identifier is determined by hashing the nodes address. This address can be the nodes IP address. A key's identifier is determined by hashing the key.

> "Consistent hashing assigns keys to nodes as follows. Identifiers are ordered in an identifier circle modulo 2^m. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the *successor node* of key k, denoted $successor(k)'$. If identifiers are represented as a circle of numbers from 0 to $2^m - 1$, then $successor(k)$ is the first node clockwise from k." [15]

The illustration in Figure 7.2 shows a three node DHT with a key range of 8. The nodes 1, 4, and 6 are responsible for the keys from and including them selves and down to their predecessor. For example will node 4 be responsible for keys 4, 3, and 2.

Scalable Key Location

The basic lookup scheme of Chord is described as a list traversal, iterating through node successors until the node responsible for the given key is found. Chords scalable alternative involve storing additional routing information.

Given an m-bit identifier space, each node contains a routing table with up to m entries, called the *finger table*.

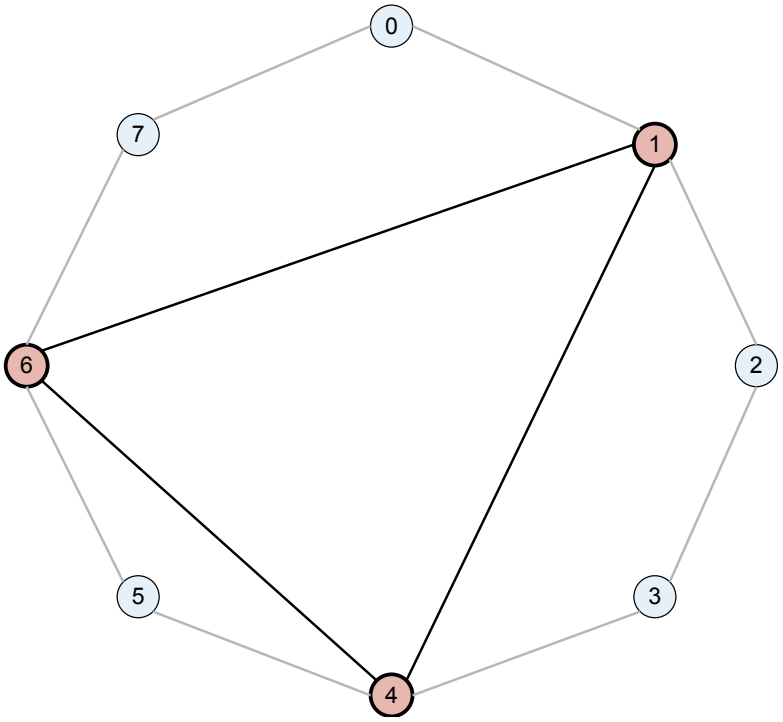

Figure 7.2: *A three node Chord DHT with a key range of 8 keys, with nodes indexed by keys 1, 4, and 6.*

> "The i^{th} entry at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle, i.e., $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$ (and all arithmetic is modulo 2^m). The node s is referred the i^{th} finger of node n." [15]

Dynamic Operations and Failures

In order to ensure that lookups execute correctly, Chord must ensure that finger table entries and successor pointers are correct even in the presence of failures and nodes joining the network. This is achieved by Chord "stabilization" protocol which runs periodically in the background. This protocol consists of the following methods: *stabilize*, *notify*, *fix_fingers*, *check_predecessor*.

Stabilize is an algorithm that is periodically called to verify the nodes position in the ring. If the position is not correct, the node will *notify* the neighbours about the new structure of the ring; stabilizing the network. *Fix_fingers* periodically goes through the nodes lookup table and update their relevant information about the network structure. *Check_predecessor* is called periodically to check if the nodes predecessor has failed. If so, the network will be updated the next time the ring is *stabilized*.

7.2.2 PyChord Design and Implementation

PyChord fulfills the Chord protocol algorithms and supports key lookup according to Chords scalable lookup scheme.

A PyChord network consists is represented as a PyChord object, and each instance of this object will represent a node in the network. A PyChord object supports joining, creating and leaving the network in addition to performing key lookup within the network.

PyChord implements the Chord "stabilization" protocol algorithms as asynchronous tasks in separate threads. These algorithms will run periodically with configurable intervals, and starts to run as soon as the PyChord object joins a network.

PyChord is designed to be customized and configurable. This is achieved by sub-classing the PyChord object, and implementing new methods, creating interfaces for network interaction. These methods can then be accessed by

other nodes through RPCs.

A small difference with between Chord as described above and PyChord is that both the nodes IP address as well as its port number will define its key. This way a single computer can support multiple instances of PyChord.

7.3 PyRest

PyRest is an open source Python module for developing RESTful interfaces for web applications in Python. It was developed prior to Harvest as a solution to a mandatory assignment in a separate university course, and is used for creating Harvests external interfaces, along with the Harvest Tracker interface.

The idea behind PyRest is to create a framework that simplifies the process of creating RESTful interfaces. The framework is similar to that of Webpy[2], Flask[3], and Bottle[4], only less complex, as PyRest only handles the setup of interfaces, rather than a framework for development entire applications.

7.3.1 PyRest Architecture

PyRest follows a standard client-server architecture, where PyRest is the framework for setting up interfaces for clients at the servers. PyRest offers a simple way of defining the interfaces, and a server handling the requests of clients accessing this interface.

7.3.2 PyRest Design and Implementation

PyRest features methods for setting up RESTful HTTP interfaces, and to implement the HTTP commands to support for each resource defined. PyRest also features a simple way of supporting mime type responses with formatted data.

[2]The Webpy project - http://webpy.org
[3]Flask - http://flask.pocoo.org/
[4]Bottle: Python Web Framework - http://bottlepy.org/

Interfaces

PyRest interfaces are set up using nested classes. Classes (representing resources) within classes build up the paths for the defined resource. The support for specific HTTP methods are added by implementing them. For example, the following class definition will produce the resource */api/resource* supporting the HTTP method GET that does nothing.

```
class api(PyRest.RestResource):
    class resource(PyRest.RestResource):
        def GET(self):
            pass
```

In this example the name of the resource matches the name of the class resource. PyRest also has support for regular expressions to support arbitrary input paths. These expressions are set by defining a *__pattern__* attribute at the specific resource, and then naming this pattern in the python regular expression notation of Pythons *re* module. The following class definition produces the interface */api/ < numeric >* where any numerical character pattern is matched to this path. Again the resource *TheResource* supports only the method GET that does nothing.

```
class TheApi(PyRest.RestResource):
    __pattern__ = "api"
    class TheResource(PyRest.RestResource):
        __pattern__ = "\d+"
        def GET(self):
            pass
```

Mime Types

PyRest uses Python decorators to support different MIME types in return values. By decorating the HTTP method definitions with a MIME-decorator the return value of the method sets and converts the content type of the return value. The following class definition creates the interface */jsonobject* that returns an empty JSON object on HTTP GET.

```
class jsonobject(PyRest.RestResource):
    @PyRest.mime_json
    def GET(self):
        return {}
```

7.4 Harvest

This section will cover the implementation of Harvest. The implementation of Harvest is done in Python. Harvest is implemented as an event-based system, where most interaction between modules are done using events. Events are controlled by a separate event handler running in the background in a separate thread. Modules can subscribe to specific events by registering callback methods at the event handler.

The Harvest application runs a total of 11 threads (including the main thread).

7.4.1 Consumer Interface

The consumer interface is implemented using the PyRest module described above. It implements the interface described in Sub-section 6.1.1 by creating the appropriate class hierarchy, and implementing the methods corresponding to the HTTP commands each resource supports.

The consumer interface is handled by a RESTful server. This server is put in a separate thread running in the background. Upon read requests from consumers, e.g. getting session status or downloading retrieved data, the interface has direct access to the data and will return it to the requester. Upon requests that alter session state, the consumer interface will generate appropriate events that are handled by subscribing handlers.

7.4.2 Collection Management

The collection management consists of several classes that manage different aspects of Harvests data retrieval.

A part of the collection management is the session management. This is done by a session store that maintain track of all sessions. The session management subscribes to the events generated when a new session is initiated, and also if the session properties (such as collection definitions) are updated.

Although executed by the network interface, the session store manages the work distribution and work stealing of Harvest. Upon new collection definition events, the session store will pass the definitions through the network

interface to determine the Harvest node responsible for that definition. When out of definitions to process, the collection management will steal work from the network using the network interface.

The retrieval of data is handled by a collector object. The collector will interact with the session management to retrieve the collection definitions of the currently active session and executing them using the collection interface, as well as storing retrieved data in the Harvest nodes local storage.

7.4.3 Data Storage

The Harvest nodes local storage is a file system storage. It stores files in the JSON[5] data format. All data retrieved from the social networks such as users information about a user or a users timeline, is stored in separate files. All data that belongs to individual sessions will be stored in separate directories.

7.4.4 Collection Interface

The collection interface implements a wrapper between the SN APIs and Harvest. The support for different SN APIs is achieved by implementing the mapping from a collection definition to single SN API calls. In the current implementation of Harvest only Twitter is supported using the Twitter REST API.

This API mapping, along with twitter authentication, is achieved with Tweepy[6], an open source Python Twitter module that implements the Twitter REST API.

7.4.5 Network Interface

The network interface is implemented using the aforementioned PyChord module. The interface is implements a subclass of a PyChord object. The object interface is extended to support methods for work distribution and

[5]JavaScript Object Notation - http://json.org
[6]Tweepy - http://tweepy.github.com

work stealing. These methods are implemented using RPC methods that are callable from any node in the system.

7.5 Harvets Tracker Implementation

The Harvest tracker is a RESTful server implemented using the aforementioned PyRest module. The server hold a list of current Harvest nodes in the system in a list in memory.

The tracker periodically checks the status of every node in the system to verify that it has not failed. This is done by performing an HTTP connection to the nodes consumer interface. If the connection cannot be achieved, the node is assumed failed, and removed from the list. This is run in a separate thread.

Chapter 8

Experiments

Harvest is benchmarked, and several experiments give documentation of system performance. This chapter details the experiments run, as well as their results.

This chapter will first go through the methodology used when conducting experiments as well as define the metrics of performance. Then it will detail the experiments that benchmark the system and discuss their results individually. A more rigorous discussion of joint results is given in Chapter 9.

8.1 Methodology

The evaluation done in this thesis follow a systems approach. In this approach the performance of the system are evaluated by experimentation, and the different stages of the system are iteratively, based on experimental results, improved. This process is illustrated in Figure 8.1

The research is based on an idea. This idea defines the main goal of the project. It is an abstract description of some need or problem, and perhaps a solution to this problem. It is this idea that will shape the project. Based on this idea, a solution to the problem at hand is then is then devised in the form of a system.

The first stage of creating such a system is defining its architecture. This architecture define all the components of the system. It defines their functional properties and their responsibilities. These properties include their

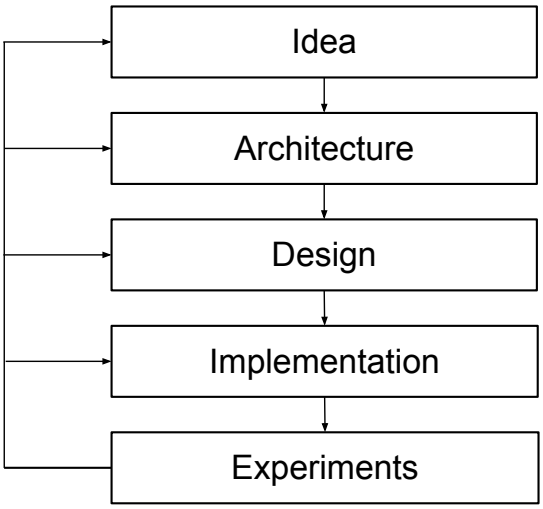

Figure 8.1: *The systems research methodology.*

functional purpose and their dependencies on other components.

The second step is to create a concrete design of the architectural components, and define how they aught to achieve their properties. This will produce a system design that define how the system and all of its components interact with each other on a more detailed level, containing a detailed description of the internal components of each architectural component, and the interfaces between them.

The third step in the process is creating a system implementation. The implementation is itself the actual system. This implementation is used for conducting experiments that will define the performance of the system. These experiments produce results, and these results grants knowledge. This newly acquired knowledge will shed light on strengths and weaknesses in the system at various levels, and may in turn lead to new ideas, architectures, designs and implementations, repeating the process from that level, continuously improving the system.

8.1.1 Metrics

In this thesis, the following metrics measure performance: (i) network bandwidth usage, (ii) memory utilization, (iii) CPU utilization, (iv) active time distribution, (v) system scale.

Network Bandwidth Usage

Network bandwidth usage is the amount of network traffic used for retrieving data from social networks. This is noted in bits per second, with prefix multipliers *Kilo* or *Mega*. In short kilo bits per second (Kbps), and mega bits per second (Mbps).

Network bandwidth is measure using an external tool; Wireshark[1]. Wireshark lets users measure all network packets going in and out from a computer, and has the ability to create filters to separate useful data from other network noise.

Memory Utilization

Memory utilization of a process is defined as the memory used by a process during the execution of that process. The Memory utilization is measured in bytes with prefix multipliers *mega* or *kilo*. In short mega bytes (MB) and kilo bytes (KB)

Memory utilization is measured using external monitoring tools, namely the Activity Monitor of OS X. This is measured using observation. Although this might be inaccurate, it will give the order of magnitude of the memory usage, and will be accurate enough for the experiments.

CPU Utilization

CPU utilization is defined as the amount of CPU used by a given process. CPU utilization is measured in percent of total CPU capacity.

CPU utilization is measured using external monitoring tools, namely the Activity Monitor of OS X. This is measured using observation. Although this might be inaccurate, it will give the order of magnitude of the CPU utilization, and will be accurate enough for the experiments.

[1]Wireshark - http://wireshark.org

Active Time Distribution

The active time distribution is the ratio *active time* over *inactive time* of a Harvest node. Active time denotes the time spent doing useful work, including: retrieving data from a SN, distributing work within the network, handling requests, and storing data.

This active time distribution is measured internally within each node individually. The measurements does not represent the active time distribution of the system, but for each node. The measurements are measure using Pythons *time* module. Exactly where this is timed is detailed for the specific experiment.

System Scaling

System scaling defines Harvest ability to increase data retrieval bandwidth with increasing number of contributing users. This is measured as the ratio of execution time of a data retrieval session of n users over that of 1 user.

System scaling is measured internally for each node. The execution time of each session is then calculated by taking the earliest start time and latest finish time of each nodes individual sessions.

8.2 Benchmarks

Harvest benchmarked on a number of experiments to define the properties of the system. The benchmarks are divided into two types; i) performance benchmarks and ii) sensitivity benchmarks.

The performance benchmarks measure the performance of the system. This includes its hardware resource utilization and scale with increased user contribution.

Sensitivity benchmarks measure the Harvest applications influence on other applications running side by side. These experiments show the affect it has on other applications, and determine whether the Harvest application behaves like low profile background application.

When running any of the experiments, the computers that are part of the

benchmarks have been left without any external influence. All processes that are not part of the operating systems background processes have been terminated to achieve more reliable results.

8.2.1 Experimental Setup

Some of the experiments are run on a single computer, and some are run on a collection of computers.

Benchmarks that are run on a single computer are run on a Macbook Air running 64-bit OS X 10.7.3 on an Intel i5 processor with 4GB or RAM and a SSD hard drive.

Benchmarks that are running on several computers are run on DELL Precision WorkStation T3500s running 32-bit Cent OS 5.7 on an Intel Xeon E5520 processor with 12 GB RAM with a 7200rpm S-ATA hard drive.

All experiments are run retrieving Twitter data using the Twitter REST API version 1.

All experiments are run using Python version 2.7.2.

8.3 Performance Benchmarks

The following benchmarks measure the system performance with respect to hardware resource utilization as well as measuring the system scaling with increasing number contributing users.

8.3.1 System Scaling

One of Harvests main measurements of performance is its ability to scale. The system should have an increasing data retrieval rate with increasing number of contributing nodes.

The System scaling benchmark has the following factor: The number of Harvest nodes contributing to a data retrieving session. The parameters of this benchmark is the data set that is retrieved.

By increasing the number of Harvest nodes contributing to the data retrieval of the same data set, the execution time of each factor is compared to that of a single Harvest node. This will show the systems ability to scale.

The execution time of the data retrieval session was measured using Pythons *time* module, more precise the function *asctime*. This function returns the current time on the format "Sun Jun 20 23:21:05 1993". As soon as the consumer interface of Harvest receives a request to initiate a data retrieval session, a session object is created, and the object is timestamped in its constructor like so:

```
class Session(object):
    def __init__(self):
        self.initiated = time.asctime()
```

Then, upon finishing the session, the session object is timestamped again, but this time denoting when it ended. The session is considered finished by the Harvest node when it no longer has any collection definitions left. End time time stamp is done in a session object method like so:

```
class Session(object):
    def end(self):
        self.ended = time.asctime()
```

When every Harvest node finishes their local sessions, the earliest start time and the latest end time are extracted and will denote the execution time of that session.

This experiment was also run on a various number of computers to test Harvests sensitivity to the underlying platform; centralized (single computer) vs. decentralized and distributed (multiple computers). The experiments was run with varying factors on one computer, and on the number of computers matching the number of Harvest nodes.

System Scaling Results

A Harvest retrieval session was started on a relatively static data set, namely the retrieval of user information of the Twitter user "Bashiok", all of his friends and followers, and their timelines. The total size of this data set was small enough to perform repeatable experiments within a reasonable amount of time (approximately 39,3 MB, takes about 29.3 hours to retrieve the data

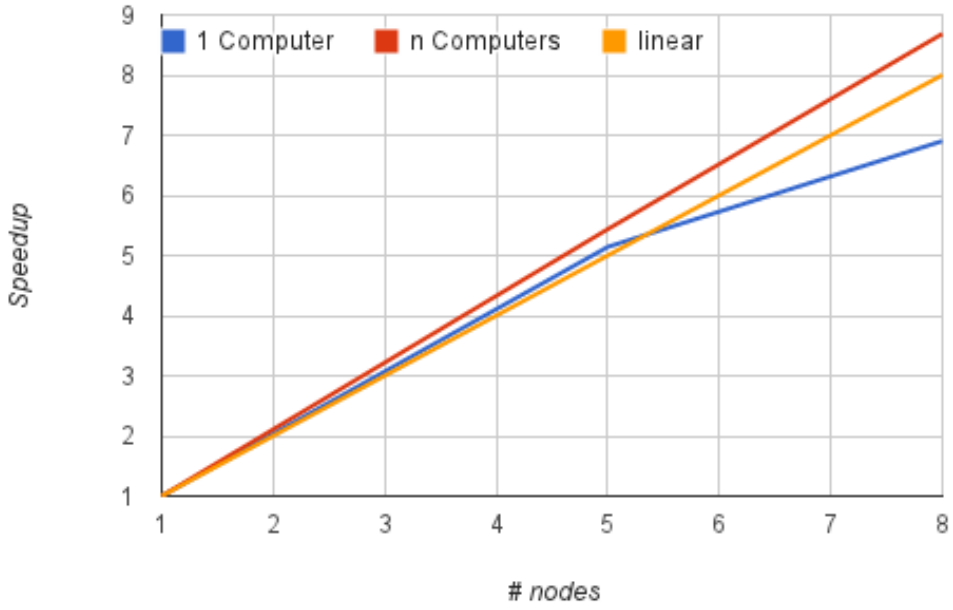

Figure 8.2: *System scale with an increasing number of contributors on a single computer and increasing number of contributors on equally increasing number of computers. The measured points are for 1, 5 and 8 Harvest nodes. The yellow line represents linear speedup, the blue line represents varying number of Harvest nodes on one computer, the red line represents varying number of Harvest nodes on equal number of computers.*

set for 1 Harvest node 1 time), and big enough to show the system scale with increased contribution.

The data retrieval was performed with the factors 1, 5 and 8 Harvest nodes. The results of this experiment is shown in Figure 8.2. The blue graph shows the benchmark for a single computer running a various number Harvest nodes. The red graph shows the benchmark of multiple computers running on the same number of Harvest nodes.

The yellow graph is a reference to linear speedup. The blue graph shows increasing performance with increasing number of Harvest nodes. The speedup is just above linear for 5 contributing Harvest nodes, but shows signs of diminishing returns as the contribution increases. The red graph shows speedup

just above linear, but with no obvious signs of diminishing returns.

The total data retrieval bandwidth for each of the experiments were approximately 3Kbps, 15Kbps, and 25Kbps for 1, 5, and 8 contributing nodes respectively.

System Scaling Discussion

The speedup achieved when increasing the number of contributing users is clearly best when running on several computers, rather than on one computer. This shows that this type of data retrieval fits the chosen architecture.

The super-linear results from the benchmark run on several computers are probably due to the way Harvest nodes retrieve data. A Harvest node will retrieve data until their limit is reached. Then they will wait for the limit to be reset by the SNP and continue. If the data harvesting was completed early or late within that data harvesting period, this would affect the results when the data size is relatively small.

8.3.2 Network Utilization

Each individual Harvest node must respect the limitation determined by the SNP of the data retrieval session. As a consequence of this the system will spend a lot of time inactive, waiting for the limit to be reset. To measure the effect of this on network bandwidth usage, data traffic is measured for system nodes while executing a data retrieval session. This benchmark is run on one computer running one Harvest instance.

Wireshark[2] is used to monitor the network usage. Wireshark has real time packet monitoring, and supports filtering of packets to filter out irrelevant packages. To create a graph over packets going between Twitter and Harvest, a filter on destination and source host names was created. This gives an overview of network bandwidth usage by the Harvest application.

[2]Wireshark - http://wireshark.org

Network Utilization Results

The graph in Figure 8.3 is a sample of the network utilization of a Harvest node during execution of a data retrieval session. The figure is a screen dump created by Wireshark, and shows only relevant network traffic by using filters as described above.

The units on the Y-axis are in bytes per tick, and the tick is set to 1 minute. The units on the X-axis are in minutes. The green line indicate the outgoing traffic to Twitter, and the red line is the incoming traffic from Twitter.

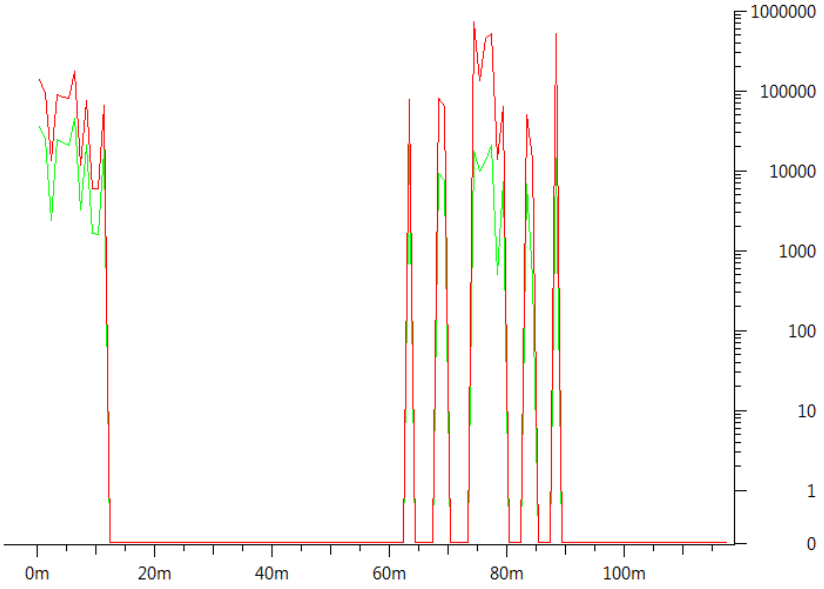

Figure 8.3: *Network utilization over time for Harvest nodes. The X-axis is time in minutes, the Y-axis is Bytes per minute. Red line denotes incoming data traffic, green line denotes outgoing traffic.*

The average network bandwidth usage of a Harvest node was 3 Kbps and was derived from the system scaling benchmark results. From the graph one can observe that the network peak usage is below 200 Kbps. Also, from the graph it is clear that Harvest has a bursty network usage. The network usage of Harvest is flat when the limit of the SNP is reached, and it jumps back up again as soon as it resets.

Network Utilization Discussion

One can conclude that Harvest is not a bandwidth intensive application. Harvest requests data from the SN sequentially, and the latency between these requests dominate the download time of the data.

8.3.3 System Activity Distribution

This benchmark measures the total time a Harvest application is in an active state vs. the time the application is in an inactive state. Harvest is in an active state when it is retrieving social data; Either performing SN API calls, load balancing, work distribution, data retrieval coordination, or data storage. Harvest is inactive the rest of the time; this could be either because it has not yet received any work to execute, or it is waiting for the limit of the SNP to be reset.

To measure the active time distribution of a Harvest application, the active time of a data retrieval session is measured and compared to the total run time of the same session.

System Activity Distribution Results

The time of several data retrieval sessions are measured and averaged. The measured time in an active state compared to an inactive state for a data retrieval session is 0.25. This means that for every unit of time spent on useful work, a Harvest application will do nothing for four.

System Activity Distribution Discussion

It is important to note that this result is only valid for one Harvest instance. The entire system will in practise not show this activity distribution, as there will at most times be at least one Harvest node retrieving some data, depending on the number of Nodes currently in the system.

8.3.4 Memory Usage

During the runtime of a data retrieval session execution, the memory utilization of the nodes are measured to give an indication on how much memory is consumed by a Harvest node. The memory utilization is measured on a single computer running a single Harvest application instance.

The memory usage is not measured accurately, but observed using OS X Activity Monitor. This does not give low accuracy, but it will show the order of magnitude of the memory consumption during runtime.

Memory Usage Results

The memory consumption of the Harvest application ran stable with a relatively constant usage of 22.0MB while the application was active retrieving data. The memory usage when the application was inactive was 10.3MB.

Memory Usage Discussion

The reason for the relatively constant memory consumption is due to Harvests storage policy. Since Harvest stores data to disk immediately after retrieving data, the memory consumption does not go up any more than the amount of data to be stored just after it was harvested.

8.3.5 CPU Usage

During the runtime of a data retrieval session execution, the CPU utilization of the nodes give an indication on the computational demands of the Harvest application. This experiment is run on a computer running a single Harvest instance.

The CPU usage is not measured accurately, but observed using OS X Activity Monitor. This does not give low accuracy, but it will show the order of magnitude of the CPU usage during runtime.

CPU Usage Results

The CPU utilization was when active retrieving data observed at a stable rate of 2.8% with high and low peaks at values 1.5% and 3.8%. When not active the application was measured a stable average rate of 1.5% with high and low peaks at values 1.1% and 1.8% respectively.

CPU Usage Discussion

Although the observations of CPU usage are inaccurate, it clearly shows that the CPU usage of the Harvest application is quite low.

8.3.6 Data Loss

To loss of the data harvesting, session statistics for several data retrieval sessions are gathered. As each collection definition is processed, they are stored in the session object as either completed or failed. This is then used to determine the average fail rate of a definition. Also, the failure reason for failed definitions is also stored. This will give indication to why the definitions fail.

Data Loss Results

The data was retrieved from 4 data retrieval sessions. The results from the data retrieval sessions show that the amount of failing requests compared to the total amount of requests done is 16.3%, meaning that 1 in every 6th request failed. It was also almost exclusively timeline requests that filed.

Data Loss Discussion

According to the documentation of the Twitter REST API, failures can come from a number of different reasons. Extracting the relevant reasons based on API responses results in the following: failures at the server handling the request, the requested data was private, or the request for the data took too long time to process. From this it was clear that the reason for the failing requests was almost exclusively due to requests taking too long.

8.4 Sensitivity Benchmarks

The following experiments measure the affect Harvest has on other applications. The experiment is done by running Harvest during a data retrieval session side by side of another application, and measure how the performance decreases. This experiment is run both as Harvest is active retrieving data, and while Harvest is inactive, waiting for the limit of the SNP to reset. Both of these runs are then compared to a reference run measured without the Harvest application.

8.4.1 CPU Sensitivity

To test the affect Harvest has on the CPU performance of other applications, a CPU intensive benchmark calculating prime numbers is run side by side Harvest.

The performance of the benchmark is measured both execution time, or real time, and CPU time. CPU time measures the amount of time the CPU has spent on the benchmark. Differences in this measurement would indicate the cache effect of the Harvest application. The real time measurements measure the total amount of time spent on the benchmark. This measurement would indicate the effect Harvest has on CPU utilization and CPU scheduling.

The benchmarks are written in C, and uses headers *time.h* and *sys/time.h* for measurements. Real time is measured using the function *clock* found in *time.h*, CPU time is measured using the function *gettimeofday* found in *sys/time.h*

All experiments are run at least 60 times and averaged.

CPU Sensitivity Results

The measurements are shown in Figure 8.4 with relative performance to the reference run with respect to execution time. The results suggest that there is little affect by the Harvest application on CPU utilization as the performance decreased with less than 1% to the reference run in both the active and inactive runs.

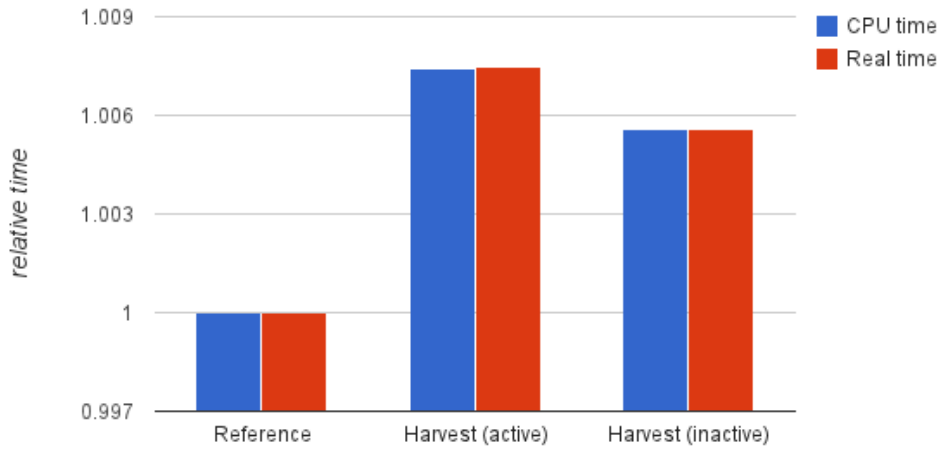

Figure 8.4: *Relative CPU performance comparing real and CPU execution time. Lower is better. Note that the Y-axis does not start on 0.*

CPU Sensitivity Discussion

The CPU time and real time measurements are fairly equal. This can indicate that the side effects from Harvest is probably due to cache invalidation, and less that Harvest is hogging CPU resources.

8.4.2 Disk IO Sensitivity

The disk input/output (IO) benchmark will measure Harvests affect on disk IO performance. This is measured by running a disk IO intensive benchmark side by side the Harvest application. A benchmark writing large amounts of data to a file will measure write performance, and a benchmark reading a large file into memory will measure read performance. The performance is measured in execution time. The time is measured in real time.

Disk IO Sensitivity Results

The measurements of disk IO sensitivity are shown in Figure 8.5. From the figure it is clear there is little effects from Harvest with respect to disk IO as the performance decrease with respect to the reference run is less than 1% for both read and write operations.

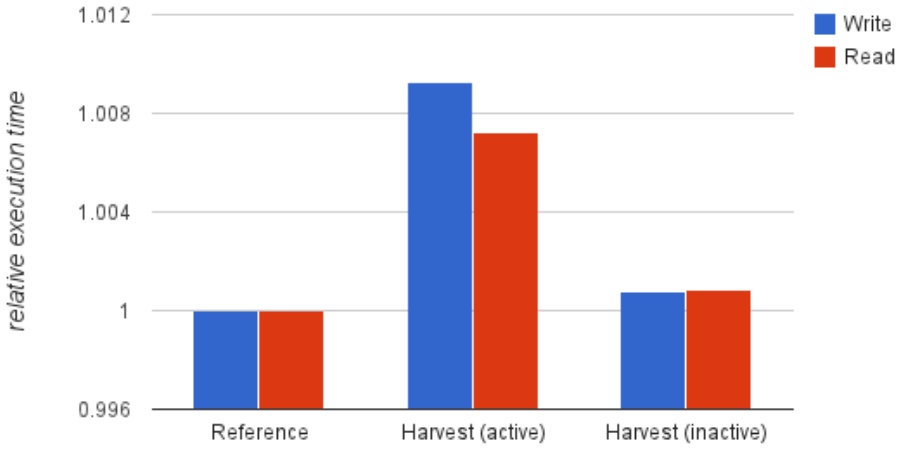

Figure 8.5: *Relative disk IO performance comparing real execution times of the reference run against running the benchmark with Harvest active and inactive. Lower is better. Note that the Y-axis does not start on 0.*

Disk IO Sensitivity Results

Running this benchmark on a SSD might be the reason for the sensitivity results. Running this on a disk drive might show different results, ans the mechanical elements of disk drives have very low performance compared to SSDs.

Chapter 9

Discussion

This chapter will try to argue for the decisions made during the work of this thesis, along with discussing the results of the experiments of Chapter 8.

9.1 Discussion of Experimental Results

9.1.1 System Performance

Unfortunately, the experiments were not run with a particularly large amount of contribution, and the performance of the system in its intended form was not tested fully. This could only be achieved by having a tremendous amount of Twitter accounts, which is simply not feasible without launching the system. As it is not mature enough for such a launch at the time of writing, this was not achieved. Ideally there should have been thousands or millions of accounts participating, giving a data retrieval rate several orders of magnitude greater.

The results of the system scale benchmark show 1:1 scaling performance with increasing number of contributing users. The experiments showed linear speedup. But it is clear from the experimental results that the data retrieval bandwidth per user is quite low. On average, each Harvest node give a bandwidth contribution of 3Kbps. With an assumption of linear speedup (which would be the best case), to achieve a relatively high bandwidth (10Mbps) one would need over 300.000 contributing users. This is also assuming there

is only one user having an active session within the system at any one time. Given 100 users having active sessions within the system, the bandwidth has to be divided among all of those users.

From these results one can draw the conclusion that Harvest in its current state not a reasonable alternative for harvesting social data if the financial costs is not a considered factor. The motivation for using Harvest is that it is an alternative to retrieving social data without any associated financial costs, even though the current data retrieval bandwidth is low.

Should the rules change, and the limits be eased, the potential increase in data retrieval bandwidth is large. The increased data retrieval rate will increase proportional to the number of contributing users, ass all users will benefit from it. This show potential in Harvest. The same would also be true if data could be harvested from more locations. This is one of the powers of distributed data retrieval such as Harvest. The potential can much higher than its centralized counterpart.

9.1.2 Harvest Node Inactivity

Harvest nodes shows from experiments that it is mostly in an inactive state; meaning it is not doing any useful work due to limitations from the SNPs. Performance benchmarks show that the overall utilization of hardware resources (CPU and memory) is quite low, not only because the Harvest node isn't doing any work most of the time, but low in general. The conclusion to draw from these results is that Harvest could be doing more.

By supporting more SN APIs, Harvest could be retrieving data from different locations while waiting for others to reset. The idea of combining data harvest from several different SN APIs was there from the beginning of the development. But for simplicity, and to show a proof of concept only such API was implemented. Adding support for more APIs is not a very hard task, but the synchronization of the semantics of data from different SNs would be a challenge.

There is also the alternative to retrieve data from implicit social networks such as public forums and blogs. As they have no limitations to how much data can be retrieved from them. This would introduce new challenges as there are no or few APIs to retrieve that data. In addition the data from these networks are structured differently, or not structure at all.

As Harvest nodes are doing nothing large portions of their time, there is the alternative to let them perform other types of useful work such a computation of retrieved data. This was experimented with early in the thesis work, but was quickly discontinued as it produced too many challenges. Mainly in what type of computational model to use for such distributed computation.

9.1.3 Data Loss

The reason for the data loss is due to the implementation of the retrieval of user timelines. When requesting a users timeline, the entire timeline is to be retrieved. As some of this data is fairly old, Twitter probably does not store it in a cache for fast retrieval. Rather it is most likely stored in a slower storage, making the request take too long.

To fix this problem, the collection interface should limit the retrieval to only recent tweets. This would give lower data losses. It would also result in more updated data, as only newer timelines are retrieved. The only downside would be that less is retrieved, but considering the pros is may be worth it.

9.1.4 Application Sensitivity

Results from memory utilization, CPU utilization and network bandwidth usage show that Harvest node application is not a performance intensive application. CPU and disk IO intensive benchmarks showed minuscule degradation in performance, and should be able to run on any modern computer in the background without ruining the performance of other applications.

9.2 Motivation for Harvest Architecture and Design

The main idea behind Harvest is to have a collaborative data retrieval to harvest large amounts of social data without any financial cost associated with it. In order to achieve this, there needs to be a way for people to contribute their resources to the collaborative data retrieval. The common way of addressing this is to have a small application run on the contributors computer, and this method is also used by Harvest. The advantage of this

approach is to distribute the load among many peers, using computational resources that are already there, and that is seldom used at full efficiency anyway. To address this, Harvest is a completely decentralized system, with no need for manual administration. All data is retrieved at the peers, and sent to the consumer upon request. It coordinates all work distribution and load balancing in a distributed fashion.

Another key point to Harvest is that is should scale to be able to achieve high data retrieval rates. This is achieved by choosing a scalable P2P architecture; a DHTs, or more specifically Chord. The DHT was chosen as it seemed fit to distribute work the same way it uses consistent hashing to distribute keys. With a large amount of contribution the distribution would be quite even. Chord was chosen for its simplicity, and as its details was well known to the author.

9.3 Evaluation of Implementation

The entire implementation of Harvest is done in Python. As Harvest is not a computationally demanding application, Python is a reasonable choice as it is a very productive language. Pythons relatively low computational performance will hardly be noticed as Harvest would idle most of the time, which is backed up by experimental results. In addition, web based applications do not suffer as much from low computational performance as network latency will dominate the execution time in many cases.

9.3.1 Motivation for Implemented Artifacts

PyRpc

The idea behind PyRpc is to simplify network communication between network nodes in the implementation of the PyChord module.

The inspiration for this module came from the pseudo code of the original Chord paper. The goal was to achieve a RPC implementation that would be as similar as possible. The pseudo-code of the Chord method *find_successor* goes like this:

n.find_successor(id)

if $(id \in (n, successor])$ **then**
  return $successor$
else
  $n' = closest\_preceding\_node(id)$
  return $n'.find_successor(id)$
end if

Compared to the PyChord source below for the same method, it is very similar. This makes implementing the protocol from the pseudo-code simpler. Note that the PyChord source is stripped from error handling and optimizations to more clearly illustrate the resemblance.

```
def find_successor(self, nodeId):
    if nodeId.inInterval(self, successor):
        return successor
    else:
        n0 = closestPrecedingNode(id)
        return n0.findSuccessor(id)
```

PyChord

The decision to do a implementation of Chord was because of two things. First, there was no easily accessible open source solution for such an implementation. Second, such an implementation would be educational, and no harm will come from more learning. It also made it possible to provide tailored interfaces for the future implementation of Harvest, which is harder to achieve with third party libraries.

PyRest

PyRest is a module for setting up RESTful interfaces in Python. It was primarily implemented as a possible solution to an assignment in a different course at the University of Tromsø. As the design of Harvest was to use RESTful interfaces for its interaction, PyRest was further developed to be more robust and useful.

Webpy is an alternative to PyRest, which is an entire framework for creating web applications. Other examples are Bottle, and Flask. These are larger and more complex compared to PyRest, and offers no extra functionality to

Harvest. Because of this PyRest was chosen. It also served as an educational bonus.

9.4 Evaluation of Problem Statements

In Chapter 1 the following hypothesis was presented:

> **Social data can be retrieved from social networks in a scalable manner, increasing in data retrieval rate with an increasing number of contributing users, using a collaborative system.**

Experimental results point to the validation of the hypothesis. Although the achieved data retrieval bandwidth is low for each contributing user, the performance of Harvest increases with increased contribution. The data retrieval rate of several users is linearly, with a 1:1 rate, scaling with the number of Harvest nodes.

The statement of this thesis from Chapter 1 was:

> **Collaborative retrieval of social data using such a system will avoid the financial costs generally associated with such data retrieval.**

By utilizing a decentralized and distributed P2P system, Harvest is able to do all data retrieval with no associated financial cost. Assuming the system will continue to scale beyond that of the experimental coverage, this will continue even when reaching a number of contributing users in the thousands, or even millions.

Chapter 10

Conclusions and Future Work

This chapter concludes the work of this thesis. The chapter will give a summary of the thesis and its contributions, concluding remarks, and future work for continuing this research.

10.1 Contributions

This thesis describes the architecture, design, and implementation of a collaborative for retrieving social data. The thesis hypothesis and statement defined in Chapter 1 are stated below:

> **Social data can be retrieved from social networks in a scalable manner, increasing in data retrieval rate with an increasing number of contributing users, using a collaborative system.**
>
> **Collaborative retrieval of social data using such a system will avoid the financial costs generally associated with such data retrieval.**

Harvest was created to evaluate this hypothesis and statement. The system has been evaluated by experiments on real Twitter data, and the results of the experiments have shown linear scaling in data retrieval rate with increasing number of contributing users to the system.

In addition to Harvest, this thesis has contributed with several open source Python modules. PyChord is a Python implementation of the Chord DHT protocol. It creates a simple interface for building P2P systems with easily expendable protocols. PyRpc is an Python RPC module that allows for network communication at a high level of abstraction. Finally, PyRest is a Python framework for setting up RESTful interfaces for web applications.

10.2 Concluding Remarks

As the amount of social data in social networks continue to grow, the potential for new/meaningful knowledge within this data is ever increasing. By making this data more accessible to the public (by removing financial costs associated with it), this potential knowledge could be extracted.

The goal of this work was to create a system where users of social networks could share their resources to the greater good, and collaborate on the retrieval of social data without having to pay for it. This thesis has focused on the creation of a scalable architecture and design that can support a large amount of collaborating users. This resulted in Harvest.

Harvest allows users to collaborate on data retrieval, avoiding any financial cost. But the data retrieval bandwidth achieved by Harvest is very low. It is clear that in its current state it is not a reasonable alternative to retrieve large amounts of data from such networks. One can conclude that Twitter has thought of this when designing their limitations. But if Harvest retrieves data from more social networking channels, and given a large contributing user base, the above limitations can be surpassed.

Harvest nodes have a large portion of their time spent inactive, or idle. This means they have the possibility to perform other types of useful work, such as data processing.

10.3 Future Work

Harvest is a working system, and its design and implementation is a solid foundation for conducting experiments and evaluating the thesis statement. But it lacks certain features and has some flaws in its design that need to be

addressed before it can be a fully functional system.

Data loss/unavailability due to offline clients is currently not handled. This means that not all requested data will be accessible at any time, in turn resulting in incomplete data sets. Consequences of this has not been tested. This can be handled by some replication scheme, but that leads to another problem; consistency. For Harvest to be useful in a real-life scenario, this should be addressed.

The system has also not been tested at a very large scale. Limited to a small number of Twitter accounts, the system has not gone through experiments that completely verifies its ability to scale.

The current interface for defining data sets is limiting. A richer interface supporting keyword searches and live streams of public status updates would make the system more usable.

To achieve more abundant data set, the support for more SNs would help. As the current implementation only supports retrieval of Twitter data, it is limited to the users of twitter, and the type of data found there; mini-blogs, follower/friend relations, and trending topics. By including more explicit SNs like Facebook and Myspace, and also implicit social networks such as public forums, blogs or Reddit, a much wider data set could be harvested. This will introduce new challenges such as data formats and representations between these networks.

Harvest nodes are being idle, or inactive large portions of the time. There is also the potential for performing other types of useful work in their spare time. If Harvest nodes for instance perform desired processing on the retrieved data, only the results needed to be downloaded by the consumers. Evaluating the characteristics of adding support for such processing would be interesting. It could also prove useful, as it takes more advantage of the underlying architecture.

References

[1] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[2] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, November 2002.

[3] David P. Anderson, Eric Korpela, and Rom Walton. High-performance task distribution for volunteer computing. *e-Science and Grid Computing, International Conference on*, 0:196–203, 2005.

[4] N.A. Christakis and J.H. Fowler. *Connected: Amazing Power of Social Networks and How They Shape Our Lives*. HARPERCOLLINS UK, 2009.

[5] Donald E. Eastlake and Paul E. Jones. US Secure Hash Algorithm 1 (SHA1). `http://www.ietf.org/rfc/rfc3174.txt?number=3174`.

[6] Facebook. Facebook statistics. http://newsroom.fb.com/.

[7] M. Grivas and D. Kehagias. A multi-platform framework for distributed computing. In *Informatics, 2008. PCI '08. Panhellenic Conference on*, pages 163 –167, aug. 2008.

[8] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *In ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[9] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM*

SIGKDD international conference on Knowledge discovery and data mining, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.

[10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.

[11] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. In *Proceedings of the 7th ACM conference on Electronic commerce*, EC '06, pages 228–237, New York, NY, USA, 2006. ACM.

[12] Atif Nazir, Saqib Raza, and Chen-Nee Chuah. Unveiling facebook: a measurement study of social network based applications. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 43–56, New York, NY, USA, 2008. ACM.

[13] E.M. Rogers. *Diffusion of Innovations*. The Free Press, New York, 5th edition, 2003.

[14] John Scott. Social network analysis. *Sociology*, 22(1):109–127, February 1988.

[15] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31:149–160, August 2001.