Computer Networks 56 (2012) 2001–2013

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

S-CLONE: Socially-aware data replication for social networks

Duc A. Tran *, Khanh Nguyen, Cuong Pham

Department of Computer Science, University of Massachusetts, Boston, MA 02125, USA

ARTICLE INFO

Article history:
Received 24 September 2011
Received in revised form 18 February 2012
Accepted 20 February 2012
Available online 3 March 2012

Keywords:
Online social networks
Distributed storage
Data replication

ABSTRACT

Online social networking has become one of the most important forms of today's communication. While an online social network can be attractive for many socially interesting features, its competitive edge will diminish if it is not able to keep pace with increasing user activities. Deploying more servers is an intuitive way to make the system scale, but for the best performance one needs to determine where best to put the data, whether replication is needed, and, if so, how. This paper is focused on replication; specifically, we propose S-CLONE, a socially-aware data replication scheme which can significantly improve a social network's efficiency by taking into account social relationships of its data. S-CLONE's performance is substantiated in our evaluation study.

1. Introduction

Evidenced by the success of Facebook, Twitter, and others alike, online social networks (OSNs) have become ubiquitous, offering novel ways for people to access information and communicate with each other. Nielsen recently published stats [1] showing that three of the world's top 10 popular brands online are social-media related and, for the first time ever, social network or blog sites are visited by 75% of global consumers who go online. Among mobile users, social networking would surpass voice as the most popular form of mobile communication by 2015, according to Airwide Solutions [2].

The increasing popularity of social networking is undeniable, and so scalability is an important issue for any OSN that wants to serve a large number of users. A typical way to cope with scalability is adding servers, as it results in expanded storage capacity as well as lower server traffic. In a distributed storage system, where the data is partitioned across a number of servers, the data can also be replicated to provide a high degree of availability in case of failures. This paper considers the aspect of data replication for OSNs, with the following motivations:

- While data partitioning and replication is a well-known problem in the literature of distributed database systems [3–9], OSNs represent a novel class of data systems. In an OSN, a data read for a user often requires fetching the data of her neighbors in the social graph (e.g., friends' status messages in Facebook or connections' updates in LinkedIn). This *social locality* should be taken into account when determining which servers to store these data so that, given a read query, all of its relevant data can be returned quickly and efficiently. The concept of social locality does not exist in conventional storage systems.
- The importance of social locality in data storage for OSNs has been substantiated in earlier work [10,11]. It is suggested that efficiency can be improved by reducing the number of servers required to answer each read query, and, therefore, the data of socially connected users should be located on the same server if at all possible. This preservation of social locality, unfortunately, does not hold for today's OSNs which rely on DHT to partition the data across the servers [12]. For the best efficiency, an ideal replication scheme designed for such a data partition should be *socially-aware*, meaning that it should try to preserve social locality in replicating the data.

Our problem is to devise a socially-aware replication scheme that can run on top of any given data partition.

* Corresponding author.
E-mail addresses: duc@cs.umb.edu (D.A. Tran), knguyen@cs.umb.edu (K. Nguyen), cpham@cs.umb.edu (C. Pham).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.02.010

OSNs of our interest are those that want a fixed budget for the disk space and update cost required for replication. Furthermore, although users are not equally active in the network, we want the same degree of data availability for every user so that everyone has an equal chance to successfully access her data under any failure condition. In the context of our problem, the number of replicas therefore is identical for every user. We propose a replication scheme called S-CLONE as the solution to this problem. The efficiency and performance of S-CLONE are substantiated in our evaluation study.

The remainder of this paper is structured as follows. We discuss the related work in Section 2. We define the problem formally in Section 3. The details of S-CLONE are presented in Section 4. The evaluation results are reported in Section 5. The paper is concluded in Section 6.

2. Related work

There are two main approaches to improving a data system's scalability: vertical scaling and horizontal scaling. While vertical scaling scales "up" the system by adding more hardware resources to the existing servers, horizontal scaling scales "out" the system instead, by adding commodity servers and partitioning the workload across these servers. Vertical scaling is simple to manage, but horizontal scaling is more cost-effective and avoids the single-point-of-failure and bottleneck problems. The latter has been a de facto standard when it comes to managing data at massive scale for many OSNs today.

The most prominent distributed storage scheme for OSNs is Cassandra [12] which is based on horizontal scaling. Cassandra, originally deployed for Facebook to enhance its Inbox Search feature, has been used by other OSNs such as Twitter, Digg, and Reddit. While there exist well-known distributed file and relational database systems such as Ficus [5], Coda [6], GFS [7], Farsite [8], and Bayou [9], these systems do not scale with high read/write rates which is the case for OSNs. Cassandra's purpose is to be able to run on top of an infrastructure of many commodity storage hosts (possibly spread across different data centers), with high write throughput without sacrificing read efficiency.

Cassandra is a key-value store resembling a combination of a BigTable data model [13] running on an Amazon's Dynamo-like infrastructure [14]. The data partitioning scheme underlying both Cassandra and Dynamo is based on consistent hashing [15], using an order-preserving DHT. The idea is to organize the servers as nodes in a circular space, called a ring, where each server is given a random value in this space representing its position on the ring. Each data item, identified by a key, is assigned to a coordinator node by hashing this key to yield its position on the ring, and then walking the ring clockwise to find the first node right after the item's position. Thus, each node becomes responsible for the data items hashed to the region in the ring between it and its predecessor node. A read query or a write query of a user is always sent to its coordinator node. For replication, Cassandra allows the application to choose its replication policy on top of the data partition. One policy provided by Cassandra is to replicate each data item on the successor nodes of its coordinator node on the ring. Other policies are also provided taking into account the load balancing across the servers within a data center, as well as across multiple data centers.

The drawback of DHT is that hashing data to random servers does not preserve social locality. Data queries in OSNs are usually light-load and it has recently been shown in [11] that network I/O can substantially be improved at the server side by keeping all of the relevant data of each query local to the same server. The objective of [11] is to maintain social locality *perfectly*, i.e., every two neighbor users must have their data co-located, which may result in some users having more replicas than the disk space can afford. In contrast, we attempt to preserve social locality under a fixed space budget for replication. In our case, there may be two neighbor users having data stored on different servers, but we try to avoid this case if possible. We also aim to provide every user with equal chance to successfully access data under any failure condition.

It is noted that besides mainstream online social networks such as Facebook and LinkedIn, there are efforts to design an OSN as a decentralized system, such as Diaspora,[1] where a user is free to choose its own hosting server. Decentralization not only addresses the DDoS security problem but also provides more privacy and freedom to the users. The research in this paper is not directly applicable to this interesting direction. However, our proposed concept of social locality in replication can be useful to designing a replication scheme for such decentralized OSNs.

3. Problem formulation

We consider an online social network of N user nodes whose data is distributed across a set of M servers. The data of our interest is the data belonging to each user that must be downloaded *by default* when she spends time online in the network. For example, in the case of Facebook, where a user's data includes her profile information, wall messages, links, pictures, and video clips, we are interested in the messages which must be displayed by default on the user screen; these messages, consequently, are the type of contents most frequently downloaded from the servers. The contents such as pictures and video clips are downloaded much less often and only on demand; hence, not our focus in this paper.

We assume an existing partition of the N users' data to the M servers, which is represented by a boolean notation p_{is} where $p_{is} = 1$ if and only if user i's data is stored at server s, and $\sum_{s=1}^{M} p_{is} = 1 \; \forall \, i$. In our replication problem, we need to find an efficient way to store K replicas for each user's data on the M servers ($K < M$). The value for K is chosen depending on the replication budget of the system and its desired availability. We use a boolean notation, x_{is}, to represent the replica assignment, where $x_{is} = 1$ if and only if user i's data is replicated at server s.

[1] http://diasporafoundation.org/.