Peer-to-Peer Wikipedia

Rain Gu and Nick Bradley

October 18, 2016

1 Introduction

Wikipedia is a free online collaborative encyclopedia hosted by Wikimedia and funded by the non-profit Wikimedia foundation. We propose a peer-to-peer version of Wikipedia for two primary reasons. First, decentralization helps eliminate proprietary interests in the system's infrastructure; instead of trust being placed in dedicated servers, trust is diffused over all participants. Second, the need for administration is diminished, since there is no dedicated infrastructure to manage.

Our project addresses two major issues that arise in a peer-to-peer encyclopedia: article replication and article discovery. Replication allows articles to be available despite node failures but requires consistent versioning across replicas. Discovery of articles is done via a search of the network. The simplest approach is to traverse the entire network until the required article is found but this results in a search time that is proportional to the number of nodes and generates a large quantity of traffic. An alternative approach is to maintain a structure (overlay) on top of the physical network to bound the search time. These issues are discussed in the next section.

2 Background

A peer-to-peer network consists of distributed nodes that make some of their resources available to others without the use of central coordination. Nodes are both consumers and suppliers of resources and may leave/fail or join the network at any time. Nodes can be organized using overlays. An unstructured overlay is "an overlay in which a node relies on its adjacent nodes for delivery of messages to other nodes in the overlay" [4]. A structured overlay is "an overlay in which nodes cooperatively maintain routing information about how to reach all nodes in the overlay" [4]. There is a trade-off between search efficiency and routing maintenance between the two overlays.

Replication of content across nodes in any type of distributed system is important for ensuring availability. However, replication increases complexity by requiring that content be successfully distributed across a sufficient number of nodes while also maintaining consistent versions. That is, the same content must be shown to the requestee regardless of the node hosting the article.

To ensure that versions are consistent across nodes we will employ Interval Tree Clocks (ITC) [3]. We choose ITCs over simpler version vectors [5] or version stamps [1] because it does not require global ids nor global coordination to manage versions and unlike version stamps, it is suitable for practical use [2]. We will use the core operations: `fork`, `event` and `join` that are used to model causal tracking mechanisms in ITC. These operators act on stamps (logical clocks) whose structure is a pair (i, e) where i is an id and e is a version vector. Thus, for our application, causality is the pointwise partial order: $e \leq e'$ iff $\forall k, e[k] \leq e'[k]$. We will use the following definitions to implement the operators:

- `fork` Clone the causal past of a stamp, outputting a pair of stamps that have identical copies of the event component and distinct ids: $\texttt{fork}(i, e) = ((i_1, e), (i_2, e))$ such that $i_2 \neq i_1$.
- `event` Increments a counter associated to the identity in the stamp: $\forall k \neq i, e'[k] = e[k]$ and $e'[i] = e[i] + 1$.
- `join` Merge two stamps, producing a new one: $\texttt{join}((i_1, e_1), (i_2, e_2)) = (i_3, e_3)$ where e_3 is the pointwise maximim of e_1 and e_2.

To be a useful service, our peer-to-peer encyclopedia will fulfill two requirements

(SG1) The latest version of every article will be available with high probability, and

(SG2) Versioning of articles is consistent across all nodes storing the article.

We explain how we plan to achieve these requirements in the next section.

3 Proposed Approach

3.1 Assumptions

We make the following assumptions about the system:

- Nodes fail (app isnt running) independently and with the same probability p.
- No Byzantine faults or application bugs.
- Messages will not be dropped or corrupted.
- Messages may be delayed but will be ordered correctly.

3.2 Article Discovery

TODO: Will just be a flooding search.

3.3 Article Replication

An article must be replicated to a sufficient number of nodes to guarantee (up to some probability) that the content will be available any time it is modified.

Choosing replica nodes. To guarantee an article is available with probability α, we send the article to $n \geq \lceil \frac{\log(1-\alpha)}{\log p} \rceil$ nodes. We keep sending the article to different nodes until we receive n responses (there is no harm if the article is on more than n nodes).

Determining version number. If a client wishes to modify an article, then the client must first find the latest version of the article using the method described in the article discovery section. If no article is found, then the client is creating a new article (this is true with high probability because of SG1). Otherwise, the client modifies the content of the latest version.

We use ITC [1] to guarantee consistent versioning across replicas. For forking operations we will use the MAC address of the node to generate the new id.

Creating an article

Assign a seed version stamp to the newly created article

Fork the version to the chosen replica nodes

Modifying an article

Fork the latest version from some node

Modify the content of the article

Sync [2] the new version to the chosen replica nodes

Deleting an article (not implemented: can't guarantee that article is removed from all nodes)

[1] We will use the Go ITC library https://github.com/fgrid/itc

[2] A `sync` is the atomic composition of `join` followed by `fork`.

4 Evaluation Methodology

To test the algorithms and protocols we designed and implemented, we will build an application in Go with simple user interface (i.e. HTML). Since p2p Wikipedia is targeting to be used over internet, we will build several nodes using proxy IP to simulate the real world scenarios. The top one attribute of the application to test is SG2: Versioning of articles is consistent across all nodes storing the articles, since SG1 is achieved mostly by assumption.

The cases to test are:

1. a client read an article.
2. a client update an article while no other clients updating the same articles in a close time interval.
3. multiple clients are updating the same article in a close time interval.
4. a new client(node) joins the system and bootstrap.

For each cases, we will also test some cases with the faulting nodes or network failure. The other technique we are using would be the GoVector Library to trace the message orders.

5 Timeline

Set up Node Template and Data Structures (i.e. ITC).	Oct 27
Implement Search & Read operations and protocols.	Nov 11
Implement Update operation and protocols.	Nov 20
Design a simple UI	Nov 23
Final Encapsulation & Report	Nov 30

References

[1] P. S. Almeida, C. Baquero, and V. Fonte. Version stamps-decentralized version vectors. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 544–551, 2002.

[2] P. S. Almeida, C. Baquero, and V. Fonte. Improving on version stamps. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, volume 4806 of *Lecture Notes in Computer Science*, pages 1025–1031. Springer, 2007.

[3] P. S. Almeida, C. Baquero, and V. Fonte. *Interval Tree Clocks*, pages 259–274. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] J. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications.* Morgan Kaufmann, 2008.

[5] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Softw. Eng.*, 9(3):240–247, May 1983.