A DECENTRALIZED WIKI ENGINE FOR COLLABORATIVE WIKIPEDIA HOSTING*

Guido Urdaneta, Guillaume Pierre and Maarten van Steen
Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands

Keywords: Wiki, peer-to-peer, collaborative web hosting, decentralized.

Abstract: This paper presents the design of a decentralized system for hosting large-scale wiki web sites like Wikipedia, using a collaborative approach. Our design focuses on distributing the pages that compose the wiki across a network of nodes provided by individuals and organizations willing to collaborate in hosting the wiki. We present algorithms for placing the pages so that the capacity of the nodes is not exceeded and the load is balanced, and algorithms for routing client requests to the appropriate nodes. We also address fault tolerance and security issues.

1 INTRODUCTION

The development of the Internet has facilitated new types of collaboration among users such as *wikis* (Leuf and Cunningham, 2001). A wiki is a web site that allows visitors to edit its content, often without requiring registration. The largest and best known example is Wikipedia (Wikipedia, 2006), which is a free content encyclopedia implemented using wiki technology. As shown in Figure 1, Wikipedia's traffic is growing at an exponential rate, which now makes it one of the most popular web sites on the Internet (Alexa Internet, 2006).

Current wiki engines, including the one used by Wikipedia, are based on a centralized architecture that typically involves a web application with all the business and presentation logic, coupled with a central database, which handles all the data. The particular case of Wikipedia deserves special attention because it shows that a wiki can become a very large distributed system, and also how difficult it is to scale a system with a centralized architecture.

Wikipedia consists of a group of wiki projects, with each project typically associated with a specific language edition of the encyclopedia. The biggest

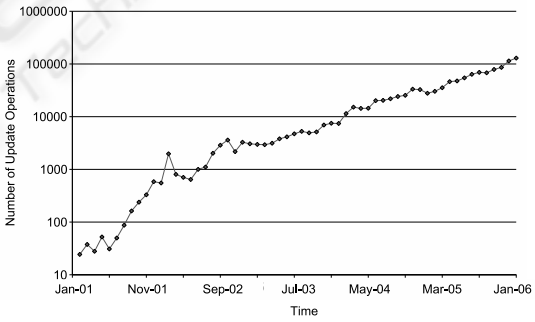

Figure 1: Average number of daily update operations in the English language Wikipedia. Note the log-scale of the y-axis.

Wikipedia edition by far is the English language version, which accounts for over 60% of the total Wikipedia traffic (Alexa Internet, 2006). Currently, Wikipedia is implemented as a set of PHP scripts that access a MySQL database. Wikipedia uses replication and caching in order to scale its architecture. The database is fully replicated to several servers, so that read operations can be spread across a number of replicas. The PHP scripts are replicated to multiple web servers as well. In addition, Wikipedia uses front-end cache servers that can handle most of the

*Supported by the Programme Alban, the European Union Programme of High Level Scholarships for Latin America, scholarship No.E05D052447VE.

Urdaneta G., Pierre G. and van Steen M. (2007).
A DECENTRALIZED WIKI ENGINE FOR COLLABORATIVE WIKIPEDIA HOSTING.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 156-163
DOI: 10.5220/0001268901560163

read requests, which helps reduce the load on the web and database servers. For most Wikipedia editions, including the English version, the database, web and cache servers are located in Tampa, FL (USA), with additional cache servers in Amsterdam (Netherlands) to handle European traffic and Seoul (South Korea) to handle Asian traffic. The total number of servers is around 240.

The centralized architecture of Wikipedia poses both technical and financial problems for its operator. On the technical side, we have one of the most popular web sites in the world experiencing an exponential growth in its activity and relying essentially on only a few access points and a single centralized database for its operations. The resulting quality of service is acknowledged by Wikipedia itself to be occasionally very poor. It is reasonable to expect that, no matter how much it is expanded, this centralized architecture is eventually going to become an even greater bottleneck than it already is. Among the possible reasons we mention that the capacity of the database may not be able to sustain the growth in site activity, and that access from certain locations is subject to delays imposed by network latency and the possible low bandwidth of intermediate links. In addition, power consumption will impose a limit to how much the central location can grow in size.

Several popular web sites have solved their scalability problems using fully distributed architectures. For example, Google maintains an infrastructure of several hundreds of thousands of servers in multiple data centers around the world (Markoff and Hansell, 2006). Amazon.com moved from a centralized architecture similar to Wikipedia's to a completely distributed service-oriented architecture spread across multiple data centers (O'Hanlon, 2006). Others rent the infrastructure provided by Content Delivery Network (CDN) vendors (Akamai Technologies, 2006), which typically consists of thousands of servers distributed across many countries. It is not unreasonable to speculate that Wikipedia will need to move to a similar massively distributed architecture in the near future. However, an architecture based on redundant distributed data centers is economically viable only for businesses where improved quality of service generates extra income that can compensate for the considerable costs involved.

Another option is the utilization of a collaborative CDN (CCDN), which allows multiple independent web site operators to combine their resources collaboratively so that the content of all participants is delivered more efficiently by taking advantage of all the contributed resources (Wang et al., 2004; Freedman et al., 2004; Pierre and van Steen, 2006). The incentive to participate in such a system is that a user who cannot afford to pay for a commercial CDN can get remote resources for hosting his site in exchange for helping other users host their sites. However, the fact that CCDNs keep all updates centralized reduces their relevance for systems like Wikipedia.

In this paper, we take the position that the issues described above can be solved by hosting Wikipedia using a decentralized and collaborative wiki engine. Similarly to the way content is created and updated, we expect that a number of Wikipedia supporters will be willing to share some of their personal computing resources to participate in the hosting of the system. This approach is, in a sense, similar to Seti@Home (Anderson et al., 2002), except that the goal of the system is to host a complex web site instead of performing local computations.

To achieve our goal, we propose to remove the centralized database and distribute the wiki pages in an overlay network formed by the computer contributed to help host the system. Each machine in the system would then be in charge of hosting a number of pages according to its processing and networking abilities. This approach introduces two basic problems: how to decide where to place each page and how to find a node that hosts a page when a client makes a request. In addition to solving these problems, the system must be prepared to handle continuous arrivals and departures of nodes, and it must guarantee that computers outside the control of the operator do not compromise the operational continuity of the web site. Our main contribution is that we explain the design of a system that can solve these problems.

The rest of the paper is organized as follows. Section 2 discusses the functionality provided by Wikipedia. Section 3 presents our proposed architecture. Section 4 discusses fault tolerance. Section 5 outlines our security strategy. Section 6 discusses related work and Section 7 concludes.

2 WIKIPEDIA FUNCTIONALITY

Before motivating our design decisions, we first discuss how Wikipedia currently operates. As shown in Figure 2, the functionality of the wiki engine is composed of page management, search and control.

The page management part is the most important since most of the information provided by Wikipedia (e.g., encyclopedic articles, user information, discussions, documentation) is in the form of wiki pages. Each page has a unique identifier consisting of a character string. Pages can be created, read and modified by any user. A page update does not result in the mod-

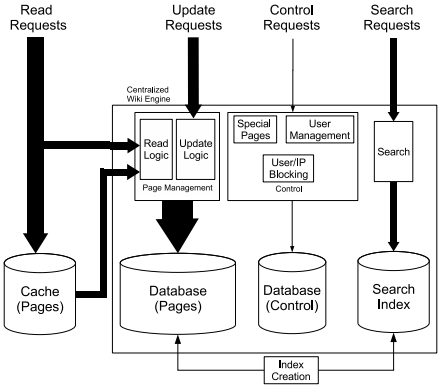

Figure 2: Current Wikipedia Architecture.

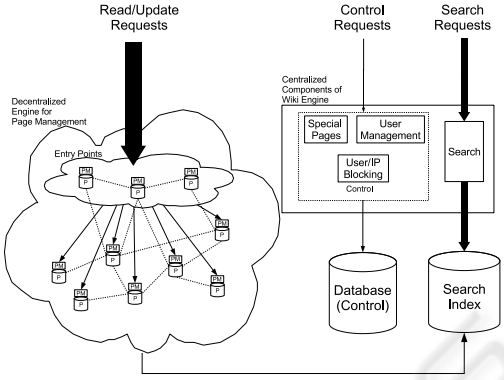

Figure 3: Proposed Wikipedia Architecture.

ification of an existing database record, but in the creation of a new record next to the previous version. It is therefore straightforward for a user to get a list of all editions of a page, read old versions as well as reverting a page to a previous state. A page can be configured to redirect all its read requests to another page, similar to a symbolic link. Privileged users have the option to rename, delete and protect pages from being edited. Part of the load generated by page read operations is handled by a group of external cache servers.

The search part allows users to enter keywords and receive lists of links to related wiki pages as a result. This part of the system is isolated from the rest of the application in that it does not access the centralized database, but a separate index file generated periodically from the text of the pages.

The control part groups the rest of the functions: i) user management, which allows users to authenticate to the system and have their user names stored in public page history logs instead of their IP addresses; ii) user/IP address blocking, which allows administrators to prevent page updates from certain IP addresses or user accounts; and iii) special pages, which are not created by users, but generated by the execution of server-side logic and provide information about the database or specific functions such as uploading static files to be referenced in wiki pages.

The search and control parts handle small or static data sets and do not impose a large load on the system in comparison to the page management part. Page management consumes most of Wikipedia's resources and the centralized database it depends on is a bottleneck that severely limits scalability, especially when considering that the page update rate increases exponentially, as we showed in Figure 1.

To address these scalability problems, we propose a decentralized implementation for the page management functionality relying on a set of collaborative nodes provided by individuals and organizations willing to contribute resources.

3 PROPOSED ARCHITECTURE

As shown in Figure 3, we propose to decentralize the page management functionality by distributing the pages across a network of collaborative nodes. In this system, each page is placed on a single node, such that the capacity of each node is not exceeded, and the load is balanced across all nodes. We assume that the collaborative system contains sufficient resources to host all the pages. It is important to note that end-users of the wiki site are regular web clients that are not necessarily part of the collaborative system.

To build such a system, we first need a decentralized algorithm to determine the placement of pages. In this algorithm, nodes improve global system performance by communicating pairwise and moving pages such that the load is better distributed.

Second, pages can be expected to change location regularly. As a consequence, it is necessary to provide a redirection mechanism to route client requests to the node currently hosting the requested page. Again, this redirection mechanism should be decentralized. Distributed Hash Tables (DHTs) have proven to be effective as a decentralized request routing mechanism.

Third, it can be expected that nodes join and leave the system relatively frequently, possibly as the result of failures and subsequent recovery. It is thus necessary to implement mechanisms that guarantee the integrity of the system in the presence of rapid and unanticipated changes in the set of nodes. Moreover, the nodes participating in the system may come from untrusted parties. Therefore, replication and security mechanisms must be in place to prevent or mitigate

the effects of failures or attacks.

3.1 Cost Function

Before discussing how we move pages from one node to another in order to improve the distribution of the load, we need a method that allows to determine how good a given page placement is. Intuitively, the goodness of a placement depends on how well each node is suited to handle the load imposed by the pages it hosts. To model this, we introduce a cost function, which, when applied to the current state of a node, indicates how well the node has been performing for a given time period. The lower the cost value, the better the situation is. We can also define a global cost function for the whole system, which is the sum of all the costs of individual nodes. The goal of the page placement algorithm is to minimize the global cost.

The cost for a node should depend on the amount of resources it contributes and the access pattern of the pages it hosts. Examples of contributed resources are disk space, network bandwidth, CPU time and memory. To keep our model simple, and considering that the application is fundamentally I/O bound, we decided that the owner of a node can specify limits only on the use of the node's disk space and network bandwidth (both incoming and outgoing).

The cost function for a node should have the following properties. First, its value should grow as the resources demanded by the hosted pages grow. Second, lower client-perceived performance should be reflected in a higher value for the cost. Third, the cost should decrease as the resources provided by a node grow, as this favors a fair distribution of the load. Finally, the cost should grow superlinearly as the resource usage grows, as this favors moving pages to nodes with more available resources, even if all nodes have the same capacity.

This leads us to the following formula for calculating the cost $c(N, P, W)$ of node N hosting the set of pages P over the time interval W:

$$c(N,P,W) = \sum_{p \in P} \left[\alpha \left(\frac{i(p,W)}{i_{tot}(N,W)} \right)^j + \beta \left(\frac{o(p,W)}{o_{tot}(N,W)} \right)^j \right]$$

where $i(p, W)$ is the total size (in bytes) of incoming requests for page p during window W, $i_{tot}(N, W)$ is the maximum number of bytes that can be received by node N during window W, $o(p, W)$ is the number of bytes to be sent out during during window W on account of requests for page p, $o_{tot}(N, W)$ is the number of bytes that node N can send out during window W, α and β are constants that weigh the relative importance of each term, and j is an amplifying constant ($j > 1$).

Note that this function considers only network bandwidth demands and ignores disk space. Indeed, availability of more disk space does not translate into better client-perceived performance. However, regardless of any potential improvements in cost, a node can refuse to receive a page if it does not have enough disk space or if it is unable to execute the page movement without exceeding a predefined time limit.

3.2 Page Placement

Our goal is to find a placement of pages that balances the load across the nodes without exceeding their capacity. By definition, this can be achieved by minimizing the cost for the whole system. However, performing this global optimization presents several difficulties. First, finding an optimal solution is not realistic since the complexity of the problem can be very high, even for relatively small system sizes. Second, trying to directly minimize the total costs would require collecting information from all the nodes. Clearly, this is not a scalable solution. In practice it may even be impossible, since the set of participating nodes can change at any time. A further complication comes from the fact that the only way to improve an initial placement is by relocating pages, but each relocation translates into a temporary reduction of client-perceived performance, since part of the resources normally used to process client requests must be dedicated to move pages.

To overcome these difficulties it is necessary to implement a strategy that reduces the global cost without relying on global information or excessive page movement.

Our cost function has the desirable property that a page movement can affect only the costs for the two nodes involved. Thus, we can reduce the global cost if we execute only page movements that result in a reduction of the sum of the costs of the two nodes involved. This has two main advantages. First, a node only needs to collect information about the potential target peer in order to decide if a page should be moved or not. Second, this procedure can be executed simultaneously by all nodes independently.

However, this simple heuristic is not enough. While it allows to reduce the global cost without relying on global information, it does not prevent excessive page movement. Moreover, it does not provide a method to decide which pages to move or to which peer to send a page to.

3.2.1 Peer and Page Selection

We propose to interconnect the participating nodes using a gossiping protocol. Gossiping has been used to implement robust and scalable overlay networks

with the characteristics of random graphs. This can be exploited to provide each node with a constantly changing random sample of peers. We use the Cyclon protocol (Voulgaris et al., 2005), which has proven to have many desirable properties. Cyclon allows to construct an overlay network in which each node maintains a table with references to n peers, and periodically exchanges table entries with those peers. The result is that at any point in time, every node has a random sample of n neighbors that can be used as potential targets to move pages to.

The next problem is to decide which pages to consider for movement. Checking all possible page-neighbor combinations might be costly due to the possibility that a node hosts a large number of pages. To keep things simple, we consider only a random sample of at most l pages. For each neighbor, the algorithm calculates the costs resulting from moving each page from the sample to the neighbor and stopping when a page suitable for movement to that neighbor is found. We realize that there are page selection strategies that might make the system converge faster to a good distribution of the load, but the trade-off would be significantly increased complexity, which we want to avoid.

We want pages to be transferred between nodes as quickly as possible. For this reason, we move no more than one page at a time to the same neighbor. We also stop trying to move additional pages from a given node if a significant part of the outgoing bandwidth is already dedicated to page movements, as sharing a limited amount of bandwidth between several simultaneous page movements would slow down each such movement.

3.2.2 Excessive Page Movement Prevention

A potential pitfall of the previously described algorithm is that its repeated execution is likely to result in many page movements that produce negligible cost reductions, but also significant reductions in the amount of available resources for processing client requests, thus worsening client-perceived performance.

To overcome this problem, we initiate page movements only when the amount of requested resources approaches the amount of contributed resources from the concerned node over a time window W by a certain margin. The rationale for this is that even though transient resource contention is unavoidable, it should not become permanent to avoid damaging the user-perceived performance.

The load of each contributed resource of node N hosting the set of pages P can be calculated with the following formulas:

$$o_r(N,P,W) = \frac{\sum_{p \in P} o(p,W)}{o_{tot}(N,W)}$$

$$i_r(N,P,W) = \frac{\sum_{p \in P} i(p,W)}{i_{tot}(N,W)}$$

$$d_r(N,P,W) = \frac{\sum_{p \in P} d(p,W)}{d_{tot}(N)}$$

where $o_r(N,P,W)$ is the load on the contributed outgoing bandwidth, $i_r(N,P,W)$ is the load on the contributed incoming bandwidth, $d_r(N,P,W)$ is the load on disk space, $d(p,W)$ is the average amount of disk space used by page p during window W, and $d_{tot}(N)$ is the total contributed disk space of node N.

These values are zero if the node does not host any page and approach 1 as the load on the node approaches its maximum capacity. Note that the only resource whose load can exceed the value 1 is the outgoing bandwidth.

We measure these values periodically and execute the optimization heuristic if any of them exceeds a threshold T $(0 < T < 1)$.

3.2.3 Summary

In our proposed architecture every participating node contributes specific amounts of disk space and incoming and outgoing bandwidth. The wiki pages are spread across the nodes. Nodes continuously try to find a good distribution of the load by relocating pages. We define a good distribution as one in which pages are distributed such that every node hosts a set of pages whose access pattern requires less resources than those contributed by the node. To achieve this, all nodes execute basically three algorithms. The first is the Cyclon protocol, which creates an overlay network that resembles a random graph. This provides each node with a constantly changing random sample of peers with which to communicate. The second is a load measuring algorithm. Every node continuously measures the load imposed by its hosted pages on each contributed resource over a predefined time window. The third algorithm is the page movement algorithm which is built on top of the other two. This algorithm uses the load measurements to determine if page movements are necessary and tries to optimize a cost function by moving pages to peers provided by the Cyclon protocol. Figures 4 and 5 show pseudocode of the page movement algorithm.

3.3 Request Routing

Each participating node should be capable of handling any page request by a client. Moreover, when a node

```
if outgoing_bandwidth_load > To OR
   incoming bandwidth_load > Ti OR
   disk_space_load > Td
    c = cost_func(P)
    S = random_subset(P)
    N = cyclon_peer_sample()
    M = ∅
    for each n ∈ N
        c_n = n.cost()
        x_n = n.cost_func_parameters()
        for each p ∈ S
           if used_bw_page_move() > K
               return
           pot = cost_func(P − M − p)
           pot_n= peer_cost_with_page(x_n,p)
           if pot + pot_n < c + c_n
               if n.can_host_page(p_metadata)
                   n.move_page(p)
                   M=M ∪ p
                   S=S − p
                   break
```

Figure 4: Page movement algorithm executed periodically by all nodes to send pages to peers.

```
n.cost()
    return cost_func(P)

n.cost_function_parameters()
    return {req_incoming_bandwidth(P),
            req_outgoing_bandwidth(P),
            total_incoming_bandwidth(),
            total_outgoing_bandwidth()}

n.can_host_page(p_metadata)
   if size(p) > K_d*avail_disk_space OR
      size(p)*avail_incoming_bandwidth() > K_i
         return false
   return true

n.move_page(p)
    //transfer data of page p
```

Figure 5: Routines invoked by the page movement algorithm on the receiving peer n.

receives a request it is likely not to host the requested page. Therefore, it is necessary to implement an efficient mechanism for routing a request to the node that hosts the requested page.

Several techniques have been proposed to route requests in unstructured overlay networks like the one formed by the gossiping algorithm described above (Cholvi et al., 2004; Lv et al., 2002). However, none of them provides the efficiency and high recall offered by DHTs for simple queries in which an identifier is given. Therefore, we decided to implement a DHT across all participating nodes using a hash of the page name as the key to route messages. The node responsible for a given DHT key is then supposed to keep a pointer to the node that currently hosts the page corresponding to that key.

When a node receives a client request, it executes a DHT query and receives the address of the node that hosts the page. It can then request the data from that node and produce an appropriate response to the client. This approach is more secure than forwarding the request to the node hosting the page.

Page creation requests constitute an exception. In this case, if the specified page indeed does not exist, it may be created in any node with enough capacity to host a new page. A new key must also be introduced in the DHT. Note that initially, any node will be acceptable, but after some time, the page placement algorithm will move the page to a better node.

The DHT introduces a simple change in the page placement algorithm. Every time a page is moved, the nodes involved must route a message in the DHT with information about the new node hosting the page.

Any DHT protocol can be used for this application (Stoica et al., 2003; Rowstron and Druschel, 2001; Ratnasamy et al., 2001).

4 FAULT TOLERANCE

In a collaborative system such as ours, membership is expected to change at any time because of the arrival of new nodes and the departure of existing nodes, either voluntary or because of failures. Node arrivals are easy to handle. A new node simply has to contact an existing node, and start executing the application protocols. As a result, the new node will eventually host a number of pages and DHT entries.

Voluntary departures are easy to handle as well. We assume that whenever a node leaves, the rest of the system maintains enough resources to take its load. It is thus sufficient for the leaving node to move all its pages to other nodes that can host them, move all its DHT keys to the appropriate nodes, and refuse to take any page offered by other nodes while it is in the process of leaving.

Node departures caused by failures are more difficult to handle because a failed node is unable to move its data before leaving the system. To simplify our study, we make several assumptions. First, we only consider node crashes which can be detected by other nodes through communication errors. Second, the DHT protocol tolerates failures so that the data it hosts are always available. Third, the overlay network created by the gossiping protocol is never partitioned into disconnected components. Fourth, a crashed node that recovers and rejoins the system

does so as a new node, that is, it discards all data hosted previous to the crash.

We focus our discussion on the problem of ensuring the availability of the pages. The key technique to prevent data loss in case of failures is replication. We thus propose to use a replication strategy that keeps a fixed number of replicas for each page. To achieve this first we need to guarantee the consistency of the replicas. Second, we need to guarantee that, when failures are detected, new replicas are created so that the required number r of valid replicas is maintained.

The system will keep track of the replicas using the DHT. Instead of a pointer to a single node hosting a page, each DHT entry associated with a page name will consist of a list of pointers to replicas, which we call a *hosting list*.

Read requests may be handled by any of the replicas. The node handling the client request can process it using a randomly selected replica. If the selected replica fails, it suffices to retry until a valid one is found. The failed node must be removed from the hosting list and a new replica must be created.

It can happen that different nodes accessing the same DHT entry detect the failure at the same time and try to create a new replica concurrently. This would result in more replicas than needed. This can be solved by introducing a temporary entry in the hosting list indicating that a new replica is being created. Once the page is fully copied the temporary entry can be substituted by a normal one.

Update requests must be propagated to all the replicas. The node handling the client request must issue an update operation to all nodes appearing in the hosting list for the page being updated, with a simple tie breaking rule to handle conflicts caused by concurrent updates. In addition, all replicas have references to each other and periodically execute a gossiping algorithm to exchange updates following an anti-entropy model similar to the one used in (Petersen et al., 1997). This approach has a number of advantages. First, it guarantees at a very low cost that all replicas will eventually converge to the same state even if not all replicas receive all the updates because of unexpected failures. Second, it does not depend on a coordinator or other similar central component, which could be targeted by a security attack. Third, it does not require clocks in the nodes handling client requests to be perfectly synchronized. If a failed replica is detected during the update operation, it must be removed from the hosting list and a new replica must be created.

5 SECURITY

Any large scale collaborative distributed system such as ours carries with it a number of security issues. We concentrate here on issues specific to our application and not on general issues related to the technologies we propose to use. In particular, security in gossiping algorithms and DHTs are active research areas that have produced many important results (Castro et al., 2002; Jelasity et al., 2003).

We assume that there are two types of nodes: trusted nodes, which are under the control of the operator of the wiki and always act correctly, and untrusted nodes which are provided by parties outside the control of the operator. Most untrusted nodes behave correctly, but it is uncertain if an untrusted node is malicious or not. We finally assume the existence of a certification authority that issues certificates to identify all participating nodes, with the certificates specifying if the node is trusted or not.

The most obvious security threat is posed by the participation of malicious nodes that try to compromise the integrity of the system, for example, by deleting or denying access to a specific page. Our first security measure is preventing untrusted nodes from communicating directly with clients, as there is no way to determine if the responses provided by an untrusted node are correct or not. Therefore, all nodes that act as entry points to the system must be trusted and must not redirect client HTTP requests to untrusted nodes.

Due to space limitations, we do not discuss all specific security measures, but our general strategy is based on digitally signing all operations in the system, so that all nodes are accountable for their actions. Certain operations, such as reading or updating the DHT, are restricted to trusted nodes and other operations performed by untrusted nodes, such as moving pages, require approval by a trusted node. This approach based on digital signatures has already been applied successfully in (Popescu et al., 2005).

6 RELATED WORK

Several systems for collaborative web hosting using peer to peer technology have been proposed (Wang et al., 2004; Freedman et al., 2004; Pierre and van Steen, 2006). However, all these systems apply simple page caching or replication algorithms that make them suitable for static Web content only. To our knowledge, no system has been proposed to host dynamic content such as a wiki over a large-scale collaborative platform.

7 CONCLUSIONS

We have presented the design of a decentralized system for hosting large scale wiki web sites like Wikipedia using a collaborative approach. Publicly available statistics from Wikipedia show that centralized page management is the source of its scalability issues. Our system decentralizes this functionality by distributing the pages across a network of computers provided by individuals and organizations willing to contribute their resources to help hosting the wiki site. This is done by finding a placement of pages in which the capacity of the nodes is not exceeded and the load is balanced, and subsequently routing page requests to the appropriate node.

To solve the page placement problem we use a gossiping protocol to construct an overlay network that resembles a random graph. This provides each node with a sample of peers with which to communicate. On top of this overlay, we try to balance the load on the nodes by executing an optimization algorithm that moves pages trying to minimize a cost function that measures the quality of the page placement. Routing requests to the nodes hosting the pages is done by implementing a Distributed Hash Table with the participating nodes. We further refine the system by replicating pages such that failures can be tolerated.

We also outline our security strategy, which is based on the use of a central certification authority and digital signatures for all operations in the system to protect the system from attacks performed by untrusted nodes acting maliciously.

In the future, we plan to evaluate our architecture by performing simulations using real-world traces. We will also carry out a thorough study of its security aspects and work on the issue of providing incentives to motivate potential collaborators to participate.

REFERENCES

Akamai Technologies (2006). `http://www.akamai.com`.

Alexa Internet (2006). Alexa web search - top 500. `http://www.alexa.com/site/ds/top_sites?ts_mode=global`.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61.

Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. (2002). Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314.

Cholvi, V., Felber, P., and Biersack, E. (2004). Efficient search in unstructured peer-to-peer networks. In *Proc. SPAA Symposium*, pages 271–272.

Freedman, M. J., Freudenthal, E., and Mazires, D. (2004). Democratizing content publication with Coral. In *Proc. NSDI Conf.*

Jelasity, M., Montresor, A., and Babaoglu, O. (2003). Towards secure epidemics: Detection and removal of malicious peers in epidemic-style protocols. Technical Report UBLCS-2003-14, University of Bologna, Bologna, Italy.

Leuf, B. and Cunningham, W. (2001). *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional.

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proc. Intl. Conf. on Supercomputing*, pages 84–95.

Markoff, J. and Hansell, S. (2006). Hiding in plain sight, Google seeks more power. *New York Times*. `http://www.nytimes.com/2006/06/14/technology/14search.html?pagewanted=1&ei=5088&en=c96a72bbc5f90a47&ex=1307937600&partner=rssnyt&emc=rss`.

O'Hanlon, C. (2006). A conversation with Werner Vogels. *Queue*, 4(4):14–22.

Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and Demers, A. (1997). Flexible update propagation for weakly consistent replication. In *Proc. SOSP Conf.*

Pierre, G. and van Steen, M. (2006). Globule: a collaborative content delivery network. *IEEE Communications Magazine*, 44(8):127–133.

Popescu, B. C., van Steen, M., Crispo, B., Tanenbaum, A. S., Sacha, J., and Kuz, I. (2005). Securely replicated Web documents. In *Proc. IPDPS Conf.*

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. (2001). A scalable content-addressable network. In *Proc. SIGCOMM Conf.*, pages 161–172.

Rowstron, A. I. T. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. Middleware Conf.*, pages 329–350.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32.

Voulgaris, S., Gavidia, D., and Steen, M. (2005). CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217.

Wang, L., Park, K., Pang, R., Pai, V. S., and Peterson, L. L. (2004). Reliability and security in the CoDeeN content distribution network. In *Proc. USENIX Technical Conf.*, pages 171–184.

Wikipedia (2006). Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Wikipedia`.