A DECENTRALIZED ARCHITECTURE FOR ADAPTIVE MEDIA DISSEMINATION

Philippe Cudré-Mauroux and Karl Aberer

School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)
1015 Lausanne, Switzerland
{philippe.cudre-mauroux, karl.aberer}@epfl.ch

ABSTRACT

P2P content distribution networks have become extremely popular on the Internet. Due to their self-organization properties, they suffer from the lack of control to balance the load among peers for contents of different popularity. In this paper, we define and analyze a fully decentralized architecture to support replication of popular media content in a peer-to-peer network. We show that by exploiting local knowledge only, this method achieves a near-optimal behavior with respect to response time in a scenario with limited storage capabilities at peers.

1. INTRODUCTION

Although streaming media applications have become commonplace over the internet, the overall infrastructure responsible for delivering media content still lacks performance, scalability and versatility in order to meet today's requirements; exclusive and highly-anticipated media contents (films trailers, album previews or interactive story excerpts) appear regularly on the Internet, while other media files gain sudden popularity in unpredictable manners, generating so-called *flash-crowds* rushing over specific media assets. This leads to millions of users wanting to access thousands of resources asynchronously over short periods of time.

In such situations, the usual client-server paradigm, with its well-known limitations due to the central component responsible for serving all the requests, can not be applied. Though alternative architectures (see Section 5) have been devised to remedy this problem, none of them meets the requirements of a global and scalable dissemination network. In particular:

complete decentralization considered here as a necessary condition for scalability (as central components represent single points of failure and usually do not scale gracefully).

structural self-organization for autonomy and scalability reasons. The system should not require any manual configuration and should manage data in a structured way to support efficient search and replication strategies.

reactiveness as the system has to adapt continually and allocate its resources based on the current situation in order to maximize its utility.

This paper proposes a decentralized architecture for adaptive media dissemination meeting the criteria defined above. We build a completely autonomous and decentralized content distribution network by storing media files in a P-Grid [1, 2] peer-to-peer access structure, presented in Section 2. In Section 3, we introduce a novel mechanism for including reactiveness into the system, facilitating global load-balancing by replicating media files locally. Section 4 evaluates the replication scheme. Related works and conclusions are given in Section 5.

2. DECENTRALIZED SELF-ORGANIZATION

Media files are stored and accessed through a P-Grid, a robust, scalable and decentralized access structure created over an unreliable network (such as the Internet). It is composed around a virtual binary search tree distributed among a community of peers (i.e., the private or public entities forming the distribution network). Peers meet continually, constructing and refining the structure as explained in [1]. Each peer holds part of the overall structure which is built by local interactions only. Names of the assets stored in the system are mapped to binary bit strings as *keys*. Similarly, each peer is virtually positioned in the system by its binary string *path*, indicating the pieces of information the peer is responsible for. For example, the path of *peer4* in the P-Grid shown below (Fig. 1) means that it is responsible for all media items whose keys start with '10', and thus stores them. The path implicitly partitions the search space and defines the structure of the virtual binary search tree. As can be seen from the figure, multiple peers may be responsible for the same path (e.g., *peer1* and *peer6* in Fig. 1), for improving robustness and responsiveness, since peers are not online all the time but with a certain, possibly low, probability. For the rest of this paper, we assume the peers to have reached a stable and unique position in the tree (each of them is located at a leaf in the virtual hierarchy). References to peers situated in the other

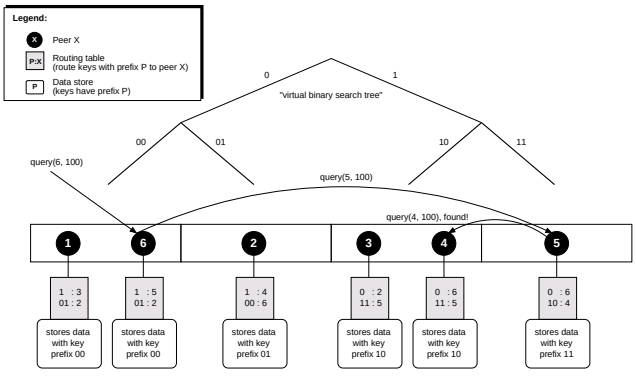

Figure 1: An Example of P-Grid

part of the tree are stored by the peers at each level, so that each peer may serve as an entry point for any search query; the peer first checks for the longest common prefix between its path and the key of the file it is searching for, and eventually forwards the request to another peer it knows at the level corresponding to the end of that prefix. A query for '100' sent to *peer6* would for example be forwarded to *peer5* at the first level (as the key and the path of the peer do not share a common prefix, *peer6* forwards the query to a peer it knows responsible for assets starting with '1'), before finishing at *peer4*, which is capable of answering the request as it stores assets whose keys start with '10'. Note that in a real setting, multiple peers would be listed for each level in the routing tables, and one of them would be selected randomly when forwarding the request.

3. REACTIVENESS

P-Grid provides us with a balanced, distributed, decentralized and self-organizing structure for accessing media assets in a community of peers. On top of this, we build mechanisms for allowing wide-spread dissemination of popular content. A centralized system would typically rank the different assets based on the number of requests they have generated, and replicate the most popular files accordingly. This is obviously not suitable for a decentralized system, where information as well as decision processes are distributed among the peers. Also, we avoid moving the peers from one leaf to another (*unbalancing* the graph *de facto* by clustering numerous peers around paths referring to currently popular assets), as it would imply rewriting most of the routing tables repetitively. Unlike many systems, we are not interested in geographical optimization of caches either; as stated in [3], there is a significant temporal stability in file popularity, but not much stability in the domains from which clients access the popular content. Instead, we take advantage of structural properties of the system to replicate data based on local interactions at strategic locations crossed while processing the query.

This works as follows: Peers maintain statistics about the number of times they serve the assets they are responsible for (i.e., the assets they have to store). An asset is considered as *popular* if this number reaches a certain threshold γ (see next section for methods to compute γ locally), and is immediately granted a *popularity level*; these levels mask the less significant bits of the key used to identify the asset. Due to that mask, the key is considered as shorter, placing the asset virtually higher in the hierarchy, thus implicitly leading to a wider dissemination of the content. For example, Fig. 2a depicts a simplified P-Grid at the introduction of a new asset, represented by key '000'. As explained in the preceding section, the peer responsible for the key (*peer1* here) first possesses the asset. Then, considering the asset as popular, *peer1* grants a first level of popularity to it; the asset, though concretely keeping its original key, is now considered as referring to key '00'. When learning about that fact, *peer2* replicates the content as it is also responsible for key '00' (see Fig. 2b, where *peer1* has transmitted the asset along with the original key and the popularity level to *peer2*). Now, if we slightly modify the query processing algorithm by allowing a peer to check if he stores an asset locally before routing a request any further, *peer2* will start to serve requests for '000' when encountering them. Furthermore, as a request for asset '000' is at the second level routed to *peer1* or *peer2* indifferently, one half of the requests for this asset will statistically be served by *peer2* from this moment. Now, *peer1* estimates again the number

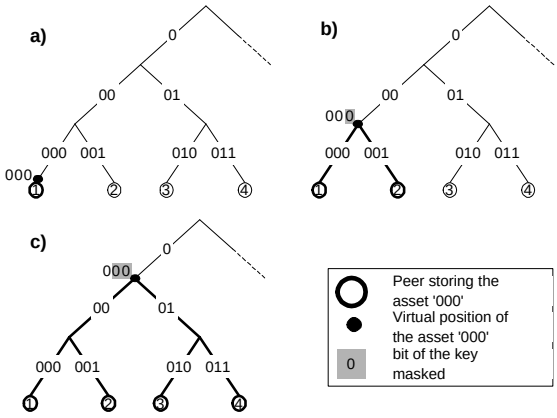

Figure 2: The Replication Mechanism

of requests it has recently answered for asset '000' (this number decreases by the order of two after some time due to the popularity level granted to the asset), and grants the file another popularity level if this number reaches γ again. Treated as if referring to key '0', the asset gets copied to *peer3* and *peer4*. Four peers store the asset, and each of them answers a quarter of the requests asking for asset '000'. Thus, we see that granting a popularity level to an asset multiplies by two the number of peers storing the asset and divides by two the request rate each of them receives for this particular asset. Finally, we force the peers to always route all the requests to their final destination, indicating if needed that the request has already been satisfied (*peer3*, for example, always forwards the request for '000' to *peer1*, even if it has already handled the request). Thus, peers are always capable of maintaining accurate statistics about the number of handled and unhandled requests they receive for an asset they are responsible for.

4. REPLICATION SCHEME EVALUATION

Mean Response Time. In this section, we study the performance of the mechanism presented above on a simplified system where all peers have the same bandwidth and processing power available. We consider a set of peers and a corresponding set of media assets stored by those peers. All assets are of fixed size, but differ in their popularity (some assets generate more requests than others). We consider the popularity of a given asset to be constant over a short period of time. Originally, each peer is located at a leaf in the P-Grid and possesses (and thus is responsible for) one distinct asset assigned randomly. Over time, peers will grant popularity levels and assets will be replicated at other peers.

Let n be the number of peers and let $a_1, \ldots, a_n$ be the assets. The popularity of the assets is given by a Zipf distribution, i.e., requests for the asset a_i arrive at a rate $\lambda_i = i$. Peers serve (i.e., transmit) the assets with a constant rate μ. At first, all assets start without any popularity level, but peers grant levels to the assets they are responsible for as long as the traffic intensity $\rho_i = \frac{\lambda_i}{\mu}$ generated by the asset is equal or greater than γ. As explained above, each of these levels doubles the number of peers storing the assets, thus divides by two the request rate λ_i' a peer receives for the asset a_i. Following this scheme, peers will start copying assets and the system will eventually come to an equilibrium, where asset

a_i will be granted p_i levels of popularity, such that

$$p_i = \lfloor log_2 \left(\frac{\lambda_i}{\mu * \gamma} \right) \rfloor + 1 \quad (1)$$

p_i is limited to the depth of the tree $mindepth = \lfloor log_2 (n) \rfloor$ and cannot be negative. Thus, at equilibrium, 2^{p_i} peers store a_i and the traffic resulting from this asset at one of those peers is $\lambda'_i = \frac{\lambda_i}{2^{p_i}}$. Requests served by a given peer are either requests for its original asset or requests for other assets it has replicated. A given peer, originally storing a_i, stores another asset $a_j \neq a_i$ with a probability $P_{storej} = \frac{2^{p_j}-1}{n-1}$. For the peer storing a_i originally, the overall incoming traffic resulting from all the assets it stores is at equilibrium

$$\lambda'_{oi} = \lambda'_i + \sum_{j=1, j\neq i}^{n} \frac{2^{p_j}-1}{n-1} * \lambda'_j \quad (2)$$

Other peers may serve asset a_i if they have replicated it. We analyze the request traffic for those peers next. The peer possessing originally asset $a_j, j \neq i$ may serve the asset a_i, receiving an overall request traffic of

$$\lambda'_{rij} = \lambda'_i + \lambda'_j + \sum_{k=1, k\neq i, k\neq j}^{n} \frac{2^{p_k}-1}{n-1} * \lambda'_k \quad (3)$$

On average, a peer who did not possess a_i originally but who replicated it has a total incoming request traffic of

$$\lambda'_{ri} = \frac{1}{n-1} \sum_{j=1, j\neq i}^{n} \left(\lambda'_i + \lambda'_j + \sum_{k=1, k\neq i, k\neq j}^{n} \frac{2^{p_k}-1}{n-1} * \lambda'_k \right) \quad (4)$$

Given 2 and 4, we compute the average incoming traffic rate for a peer serving asset a_i by weighting those two values by their respective likeliness:

$$\overline{\lambda'_i} = \frac{1}{2^{p_i}} \lambda'_{oi} + \left(1 - \frac{1}{2^{p_i}}\right) \lambda'_{ri} \quad (5)$$

Then, modelling the arrivals as a Poisson process (i.e., we have a $M/D/1$ queue), we obtain the average response time $\overline{T_i}$ for a request targeting asset a_i:

$$\overline{T_i} = \left(1 + \frac{1}{2} \frac{\frac{\overline{\lambda'_i}}{\mu}}{1 - \frac{\overline{\lambda'_i}}{\mu}} \right) * \overline{S} \quad (6)$$

where $\overline{S} = \frac{1}{\mu}$. Finally, we sum those average response times over all the requests and divide by the total number of requests to find the mean response time $\overline{T}$ of the system for a request:

$$\overline{T} = \frac{\sum_{i=1}^{n} i * \overline{T_i}}{\sum_{i=1}^{i} i} \quad (7)$$

Fig. 3 shows the average response time of the system decreasing as γ gets smaller and assets are granted popularity levels more easily (for $n = 128, \mu = 129$, $\overline{S}$ set to 1 sec). Following the curve from right to left, each discrete step in the function indicates that a popular item is granted one new level of popularity (thus gets replicated, distributing its requests among other peers). The function decreases rapidly when very popular assets enter a previously unoccupied level (at $\gamma \cong 1$, for example). Then, assets less and less popular fill the level, slowing down the decrease (or even increasing the global function, e.g., at $\gamma \cong 0.5$, where more than half of the assets have exactly one popularity level). On the extreme right ($\gamma = 1, \overline{T} \cong 5$), no asset is replicated and the requests for a given asset are always served by the peer to which the asset was assigned originally. In this case, $\overline{\lambda'_i} = \lambda_i$. On the opposite side ($\gamma = 0, \overline{T} = 1.5$), we have a perfectly balanced system where all peers replicate all assets, and where $\overline{\lambda'_i} = \frac{\sum_{i=1}^{n} i}{n}$. The average number of slots a peer needs to store the assets is a function of γ, too, and is given in Fig. 4.

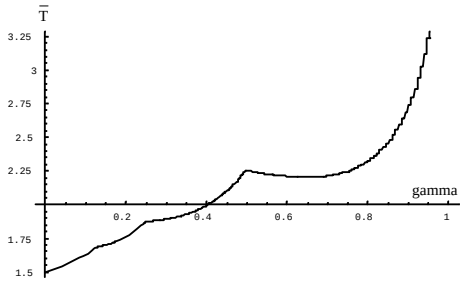

Figure 3: Average Response Time

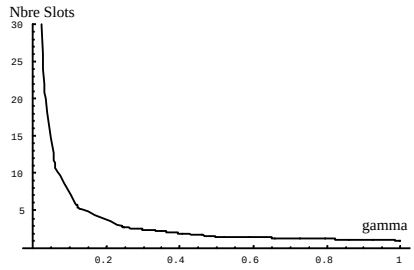

Figure 4: Mean Number Of Slots Needed

Finite Storage Capacity. In practice, peers always have a finite storage capacity; therefore, there is a tradeoff between the amount of space dedicated to asset replication and the overall performance of the system. In this section, we introduce a simple model where peers reserve c slots per level of popularity for replicating files. A peer always accepts to replicate an asset if at least one of the slots corresponding to the popularity level of the asset is still free. If all those slots are taken, the peer randomly chooses one of the $c+1$ assets and abandons it.

For this mechanism, we adapt the equations of the last section. Popularity level granting stays unaltered, and we may easily compute the number of assets $population_i$ having a certain popularity level p_i. The mean request rate generated by asset a_i to a peer storing the asset, if the average number of assets having been granted popularity level p_i is greater than the storage capacity for this level, i.e., $population_i \frac{2^{p_i}}{n} > c$, is

$$\lambda'_i = \frac{\lambda_i}{2^{p_i}} \frac{c}{population_i \frac{2^{p_i}}{n}} \quad (8)$$

Also, we weight each factor of the sum in 2 and 4 with the prob-

ability $P_{storage}(j)$ of keeping the asset a_j knowing that the peer was asked to replicate it. For example, (2) becomes

$$\lambda'_{oi} = \lambda'_i + \sum_{j=1, j\neq i}^{n} \lambda'_j \frac{2^{p_j}-1}{n-1} \frac{c}{(population_j - 1)\frac{2^{p_j}-1}{n-1} + 1} \quad (9)$$

if, again, $(population_j - 1) * \frac{2^{p_j}-1}{n-1} + 1 > c$ (note that we do not take into account the case where $p_i = p_j$ impacts on the average number of assets to be stored by the peer for popularity level p_j). Fig. 5 depicts the average response time of the system with a finite storage capacity as a function of γ (same parameters as before, $c = 2$). This time, the system is optimal for $\gamma \cong 0.2$. Below that point, peers tend to grant popularity levels too easily (therefore increasing the competition between very popular assets and less popular ones trying to fill in the slots of a certain level). Above this value, popular assets are not replicated as widely as would be necessary to balance the load of the system correctly.

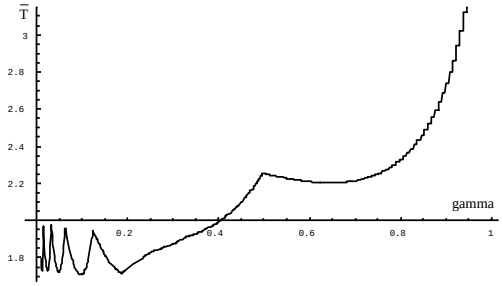

Figure 5: Average Response Time, Finite Capacity

Implementation Issues. The peers may implement the replication scheme by storing a moving average window of the number of requests they receive for the assets they are responsible for. Popularity levels are granted and removed using a local value for γ. An optimal value for this variable is computable in principle, if all system parameters are known. However, this optimal value is related to the load of the system, as performance start dropping when too many assets compete for the slots in a given popularity level. Therefore, a peer can get an estimate of the global system by keeping track of the proportion of the requests satisfied by other peers for its assets over the theoretical possible value for this number: In an unloaded environment, $2^{p_i} - 1$ peers would replicate an asset a_i, but this value decreases as the system fills up and as peers having replicated the asset drop it for replicating some other content. So, we define the emptiness indicator κ_i as this proportion, which is analytically equal to $P_{storage}(i)$ (the probability that a peer effectively stores an asset given that he is supposed to replicate it).

Fig. 6 shows the average κ as a function of γ; as expected, we observe the function decreasing rapidly with $\gamma < 0.2$, as the system fills up beyond its capacity with replicas. Peers may implement the algorithm locally by starting with a high γ and trying to decrease this value over time as long as the emptiness indicator κ is above a predefined threshold ($\kappa > 0.9$ means that less than ten percent of the replicas are dropped, and would yield $\overline{T} \cong 1.8$, which is close to the optimal value of 1.5 for an initial value of 5). Note that γ and κ do not depend on the size of the system, but only on its load.

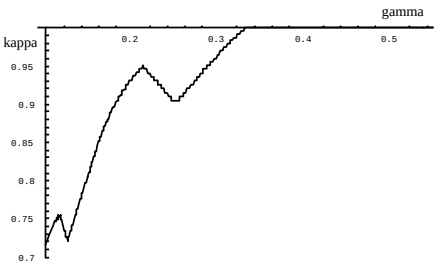

Figure 6: Emptiness Indicator κ

5. RELATED WORK AND CONCLUSIONS

This paper presented and analyzed a new scheme for replicating popular media content on the Internet. Today, *manually configured* systems based on central monitoring (e.g., Akamai FreeFlow at http://www.akamai.com/) or on hierarchical cache trees (such as Squid [4]) are commonly used to distribute content. Those systems are difficult to maintain and do not scale gracefully. To remedy this, decentralized or peer-to-peer systems have emerged recently, but have focused on efficient query routing mechanisms. Bayeux [5] is an application-level multicasting system based on a decentralized layer, but does not replicate the content actively. Freenet [6] is a P2P system protecting anonymity and including a basic replication scheme (the content is copied to every node processing the request) but does not include any analysis of the scheme. Our system is totally autonomous, decentralized, and reacts to the global demand by replicating specific assets at strategic locations, thus achieving near-optimal behavior using local interactions only. We are currently integrating the approach described in this paper into our P-Grid implementation and will evaluate it in large-scale experiments. Additionally, we want to make use of an economic model to provide incentives for collaborative behavior to the peers.

6. REFERENCES

[1] K. Aberer, M. Punceva, M. Hauswirth, R. Schmid, *Improving Data Access in P2P Systems*, IEEE Internet Computing, Jan./Feb. 2002.

[2] K. Aberer, *P-Grid: A self-organizing access structure for P2P information systems*, Proc. of the Ninth International Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy, 2001.

[3] V. N. Padmanabhan, L. Qui, *The content and access dynamics of a busy web site: findings and implications*, SIGCOMM, p. 111-123, 2000.

[4] A. Chankhunthod et al., *A Hierarchical Internet Object Cache*, Proc. of the 1996 USENIX Technical Conference, p. 153-163, January 1996.

[5] S. Zhuang et al., *Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination*, Proc. NOSSDAV 2001, 2001.

[6] I. Clarke et al., *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, ICSIWorkshop on Design Issues in Anonymity and Unobservability, July 2000.