

Extending the VoDKA Architecture with P2P Aggregated Content Management *

Carlos Abalde, Víctor M. Gulías, Laura M. Castro
MADS Group. Computer Science Department
University of A Coruña
Campus Elviña s/n, 15071, A Coruña, Spain
{cabalde, gulias, lcastro}@udc.es

Abstract

This paper presents our results building a decentralized and structured P2P video content distribution network, developed as an extension of an existing on-demand streaming server. It has been designed to improve streaming server capacity, without any infrastructure upgrade, when it must deal with big LAN-networks and highly-correlated server access patterns. A coordination algorithm build on top of a distributed hash table is presented, and its integration inside the streaming server architecture is explained. Experimental results show that the design is an appropriate approach for video content distribution with interesting scalability and availability properties.

1. Introduction

Research on content distribution networks (CDNs) pushes hard to find novel approaches. Powerful, flexible, scalable and fault tolerant architectures are demanded. Two general approaches are identified in [7]: *infrastructure-based content distribution*, improving server infrastructure in a client/server framework, and *peer-to-peer content distribution*, replacing the client/server model by a *peer-to-peer* (P2P) approach. Intuitively, a P2P approach is better suited to deal with mass-scale content distribution. However, this can only be achieved with more effort in terms of coordination, resource management, heterogeneity. . .

In this research, we deal with multimedia content distribution (specifically, video contents). A video content service has large media size contents, read-only sequential access, high service time and low look-up latency. Our goal is to design a video CDN suited to the scenario discussed in section 3, combining both infrastructure-based and P2P approaches, and integrating it with the streaming server VoDKA (Video on Demand Kernel Architecture) [4].

*Partially supported by MEC project TIN2005-08986 and Xunta de Galicia project PGIDIT06PXIC105164PN.

The paper is structured as follows. First, some background knowledge about P2P architectures and content location protocols based on distributed hash tables is introduced in section 2. Section 3 explains the existing problem and section 4 discusses our proposed solution. Next, section 5 briefly sketches the integration with VoDKA. In section 6, some performance evaluation results are presented. Finally, we present our conclusions.

2. Background

Any P2P content distribution system relies on a network of peer computers and connections among them. A survey of P2P material can be found at [1]. The topology, structure and degree of centralization of the overlay network, and the routing and location mechanisms it employs for messages and content look-up, are crucial to the operation of the system. They affect its fault tolerance, self-maintainability, adaptability to failures, performance, scalability, and security. Table 1 classifies P2P content distribution architectures according to their centralization degree and internal structure, and shows some examples for each category.

Fully decentralized structured architectures are the best scalable design and the most suitable approach for our CDN design. Two typical problems are identified in these architectures: (a) *Overlay network creation and maintenance*, how to build peers structure without any centralized servers or supernodes, and how to keep this mechanism scalable and fault tolerant; (b) *Look-up*, how to find an item in a P2P system in a scalable way, without any centralized servers [3]. Both questions are addressed using *distributed hash tables* (DHTs). In a DHT, data items are inserted and looked up using a unique key. In order to implement a DHT, the underlying algorithm must be able to determine the node responsible for storing the data associated with a given key. Hence, each node maintains information (e.g., the IP address) of a small number of other *neighbor* nodes, defining an overlay network, and routes messages to store and retrieve keys [3].

	Centralization		
	Hybrid	Partial	None
Unstructured	Napster	Kazaa	Gnutella
Structured			Chord, CAN, Tapestry, Pastry

Table 1. Some P2P technologies

Up-to-date DHT algorithms stress the ability to scale well to large numbers of nodes, to locate keys with low latency, to handle node arrivals and departures scalably, to ease the maintenance of per-node routing tables, and to balance the distribution of keys evenly among the participating nodes. CAN [8], Chord [10], Tapestry [11], Pastry [9], Kademlia [6] and Koorde [5] are some of the most popular DHT algorithms. Each one suggests a specific overlay network where all the nodes are logically interconnected. Different overlay networks have different advantages and drawbacks in terms of look-up latency, resilience to node failures, etc.

3. Problem description

The target scenario is a network of public *video-information screens* scattered among several *locations* (figure 1). At each location, there may be several video-information screens LAN-connected. Media is primarily stored in a central multimedia server (a VoDKA system), WAN-connected (typically, xDSL link) with each location. Each video-information screen has a *program*, a grid or playout of *media objects* (MOs), that defines the particular media scheduling (usually updated on a daily-base).

In principle, each screen must request to the central server and store in its local cache storage all the scheduled MO before they are required for playback. This deadline is a *timestamp*, a combination of the program start and the actual first appearance of the MO in the program.

This is a naïve approach, as long as several screens in the same LAN may request the same MO (same MO is sent through WAN link several times to the same location), which is a quite common case because, in practice, most of the programs share a great deal of content. Thus, the screens need more time to cache all required MOs and, they might not be able to reach MOs deadlines (WAN link or server overload). In order to improve this situation, screens at the same location should cooperate to avoid unnecessary traffic with central storage, following these requirements:

- Minimize coupling. Screens must operate completely or partially isolated in (probably temporal) adverse conditions.
- Design a screen/location priority framework, managed by the central server, to maximize WAN usage.

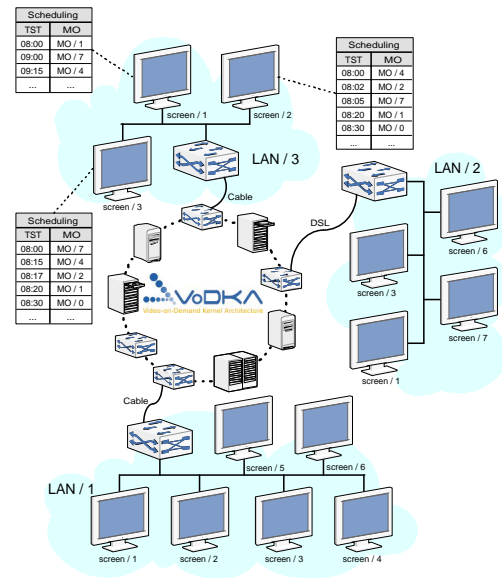


Figure 1. Content distribution scenario

- Avoid single points of failure inside each LAN, in order to reduce management complexity.
- Design a scalable architecture at LAN level.
- Allow efficient node departure and arrivals.
- Keep the server as a central storage point.
- Minimize deployment and maintenance costs.

The problem shown above suggests the definition of a solution that exploits MO sharing and temporal locality among all screen schedulings, and in particular, screens from the same location. Therefore, a coordination scheme for every location is required to improve caching effort while fulfilling the problem requirements. Benefits would be directly proportional to the correlation among screen schedulings. In the ideal case, when all the screens in the same LAN have identical schedulings, each location can be considered –from the VoDKA point of view – equivalent to one single screen, with independence from the number of screens at that location.

4. The coordination algorithm

We propose a design build on top of an structured and fully decentralized P2P architecture based on a DHT. A DHT is defined at each location (local network) to store screens state, using the MO identifier as key and storing information about MO state in the local network (whether it is being loaded in a node from central server or nodes store

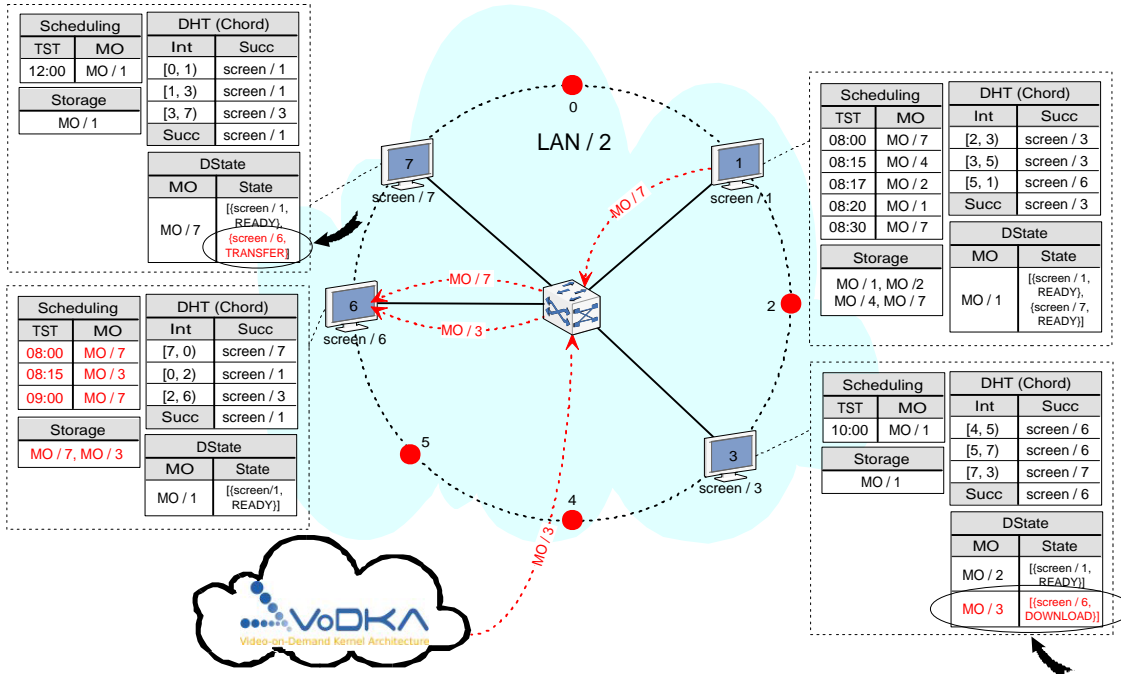


Figure 2. Coordination using Chord-based DHT example

a copy locally). Therefore, every screen in the local network has a (mostly accurate) global vision of location state to implement the coordination strategy.

Figure 2 shows an overlay network on top of a 4-node LAN (screen/1, screen/3, screen/6 and screen/7). Chord has been chosen as DHT algorithm, defining a logical ring that tightens nodes (screens) and keys (MO identifiers). For each screen, the figure shows its MO scheduling (combining current and future programs), the available MOs in its local cache, the routing table (*finger table*, using Chord's terms) for the overlay network, and the key/value pairs stored in the DHT. State and location of all copies are distributed (and replicated, optionally) and can be efficiently retrieved by each screen ($\mathcal{O}(\log N)$, being N the maximum number of screens in the local network). In the example, screen/6 starts with no scheduling and no MO in cache. As soon as its scheduling is updated, the coordination algorithm is triggered to transfer to its local cache MO/3 (from central storage server) and MO/7 (locally transferred from screen/1), updating the DHT information properly.

4.1. Peers coordination

Global state is distributed using the DHT, indexed by MO identifier. The value stored for each MO index is a list of pointers to available copies at the local network. Every pointer has an associated state according with the state

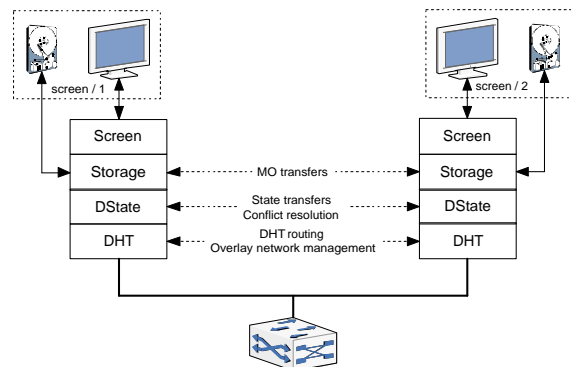


Figure 3. Node architecture

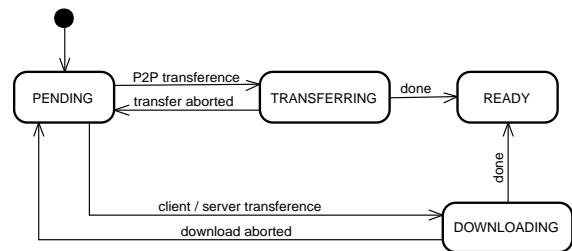


Figure 4. MO states

diagram shown in figure 4:

- PENDING. The MO is being required but it has not been decided yet how to transfer it to the local cache. This is both the initial state and the state of aborted transferences due to external reasons.
- DOWNLOADING. MO is being transferred from central storage.
- TRANSFERRING. MO is being transferred from another screen in the same local network.
- READY. MO is locally available.

LAN nodes have a layered architecture, shown in figure 3. At the storage layer, three events are identified that require the use of coordination algorithm 14 to minimize the interaction with central storage as much as possible: (a) new PENDING MO appears (b) MO deadline update (c) MO transfer cancellation (in both TRANSFERRING or DOWNLOADING state). In this algorithm, $dstate(MO)$ offers access to MO state using DHT layer, and $select_peer(copies)$ chooses one of the possible peers to perform MO transference (from a READY MO copy or, if not possible, from the cheaper TRANSFERRING or DOWNLOADING copy available). Finally, $stg(MO)$ accesses to local MO copy.

Algorithm 1 Coordination algorithm

```

if  $stg(MO) \in \{DOWNLOADING, TRANSFERRING\}$ 
then
     $vodka.notify(MO, tst)$ 
else if  $stg(MO) \equiv PENDING$  then
    if  $copies = dstate(MO)$  then
         $vodka.notify(MO, tst)$ 
         $node = select\_peer(copies)$ 
         $node.transfer(MO)$ 
         $stg(MO) = dstate(MO) =$ 
            {TRANSFERRING,  $node$ }
    else
         $vodka.download(MO, tst)$ 
         $stg(MO) = dstate(MO) = DOWNLOADING$ 
    end if
end if

```

Due to asynchronous behavior, a wrong decision affecting consistency may occur. This must be detected and solved by the DState layer as soon as it tries to update global state, contradicting current state. An example is the situation with several screens downloading the same MO from central storage. The election of the leader node in charge of solving the conflict is performed thanks to DHT MO/node mapping, establishing a dialog with the nodes involved to abort unnecessary transferences. After that, coordination

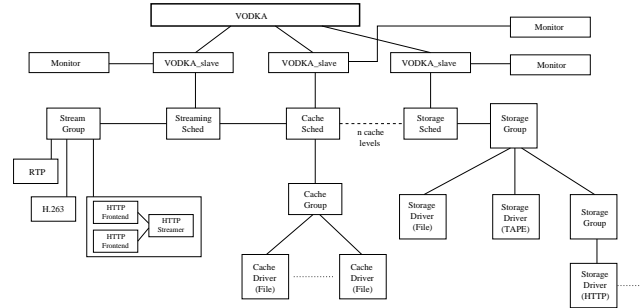


Figure 5. n layers VoDKA configuration

algorithm must be re-executed. Moreover, under some circumstances, the coordination algorithm can generate transference cycles that must be handled (omitted in the algorithm for simplicity).

5. VoDKA integration

The VoDKA [4] system is an extremely powerful and flexible multimedia streaming server. It has been designed since its conception as a distributed system. Logically, VoDKA is an agent-based system where each component –agent– performs a well-defined function: media storage, control, protocol adaptation, caching, supervision, ... These components can be composed in many ways depending on the requirements of the multimedia architecture to be deployed (figure 5).

Therefore, the integration of the peers coordination algorithm with the VoDKA system is done through a new protocol adapter agent. It will attend LANs requests and notifications –coordination algorithm– and schedule server resources based on this information and on a server-managed priority allocation policy.

5.1. Resources and priorities

Let be $L = \{L_1, L_2, \dots\}$ the set of LANs, where each L_i represents screen nodes $L_i = \{L_i^1, L_i^2, \dots\}$. We define $p(L_i)$ ($0 \leq p(L_i) \leq 1$) as a priority value of LAN L_i on the resource scheduling process. Furthermore, let be $bw_{max}(L_i)$ the highest network bandwidth that can be allocated to LAN L_i (usually equal to the maximum LAN link bandwidth).

Both $p(L_i)$ and $bw_{max}(L_i)$ are server-defined parameters managed through the server administration interface. They are the basis for the *resource scheduling algorithm* executed by the new protocol adapter agent every time the *transferences table* needs to be updated. Specifically, this table is updated when, (a) $p(L_i)$ or $bw_{max}(L_i)$ is tuned by a server administrator, (b) a new download request is received, or (c) a timestamp update is received.

L_i	L_i^j	MO	size	pos	deadline	BW
L_1	L_1^2	MO/32	?
	L_1^7	MO/7	?
	L_1^{22}	MO/12	?
L_2	L_2^9	MO/12	?
L_3	L_3^3	MO/4	?
	L_3^5	MO/17	?

Table 2. Transferences table example

LAN	$bw_{min}(L_i)$	$bw_{max}(L_i)$	$p(L_i)$	$bw(L_i)$
L_1	?
L_2	?
L_3	?

Table 3. Compressed transferences table

Once the transferences table is ready, every VoDKA/screen transference bitrate is adjusted in order to fulfill screen scheduling deadlines. Finally, as an additional optimization, and in order to minimize the number of server connections, all the data transferences between a screen and VoDKA are multiplexed in a single connection.

5.2. Resource scheduling algorithm

The resource scheduling algorithm is in charge of calculating $bw(L_i)$ for all LAN L_i . This value is the network bandwidth allocated for all the current transferences to screens at LAN L_i . $bw(L_i)$ depends on LANs priorities ($p(L_i)$), VoDKA network bandwidth (bw_{VoDKA}), transferences deadlines (timestamps), and LANs links bandwidth ($bw_{max}(L_i)$). Summarizing, a maximization problem must be solved by the algorithm.

First of all, a table like table 2 is built. It contains all the active VoDKA-screen transferences, grouped by source LAN, and showing the MO identifier and size (in bytes), the requested starting byte (usually 0), and the notified deadline timestamp. The goal of the algorithm is to allocate a suitable network bandwidth for each transference.

Then, table 2 is compressed as shown in table 3. For all L_i , $bw_{max}(L_i)$ and $p(L_i)$ are well known values, $bw(L_i)$ is the value to be calculated by our algorithm, and $bw_{min}(L_i)$ is the minimal network bandwidth needed by LAN L_i in order to fulfill MOs deadlines. In particular, $bw_{min}(L_i) = \sum \frac{size-pos}{deadline-now()}$, where *size*, *pos* and *deadline* are values from table 2, and *now()* is the current timestamp.

Finally, algorithm 20 is applied using the compressed transferences table as input. The algorithm output will be the optimal values for $bw(L_i)$. The last step is to decom-

press the transferences table and share out $bw(L_i)$ among all the transferences in LAN L_i . If $bw(L_i) > bw_{min}(L_i)$, the remaining bandwidth can be assigned fairly to every transference from L_i . On the other hand, if $bw(L_i) < bw_{min}(L_i)$, some of the MO deadlines in L_i are not going to be achieved and the reserved bandwidth $bw(L_i)$ will have to be assigned to some of the LAN requests under any policy (for example, giving priority to the more urgent requests –less deadline timestamp–).

Algorithm 2 Bandwidth scheduling algorithm

```

 $\forall i, bw(L_i) = 0$ 
 $bw_{remaining} = bw_{VoDKA}$ 
if  $\sum bw_{min}(L_i) \leq bw_{VoDKA}$  then
   $bw(L_i) = \min(bw_{min}(L_i), bw_{max}(L_i))$ 
   $bw_{remaining} = bw_{remaining} - bw(L_i)$ 
end if
 $free = [i | bw(L_i) < bw_{max}(L_i)]$ 
while  $free \neq [] \wedge bw_{remaining} > 0$  do
  for all  $i \in free$  do
     $np = \frac{p(L_i)}{\sum p(L_i) / i \in free}$ 
     $bw(L_i) = bw(L_i) + bw_{remaining} \times np$ 
     $bw_{remaining} = bw_{remaining} - bw_{remaining} \times np$ 
    if  $bw(L_i) > bw_{max}(L_i)$  then
       $excess = bw(L_i) - bw_{max}(L_i)$ 
       $bw(L_i) = bw_{max}(L_i)$ 
       $bw_{remaining} = bw_{remaining} + excess$ 
      Remove  $i$  from  $free$ 
    end if
  end for
end while

```

6. Prototype benchmarking

A prototype implementation has been developed using the concurrent functional language Erlang [2], picking Chord as the algorithm for the DHT. Each screen node is running on different Erlang virtual machines (EVM), and several EVM are deployed on different computers. An additional EVM is in charge of gathering monitoring information from all the screen nodes.

In the experiment, we consider one LAN with a variable number of screens, ranging from 8 to 128, deployed on 4 physical computers. Simulation starts with no scheduling and no media on local caches and an stable overlay network. After that, four bursts of workload are generated with new program versions including a set of fresh MOs, exactly the same for each screen, distributed simultaneously. The size of the program is what we consider as concurrency level, which range in the experiment from 1 to 64.

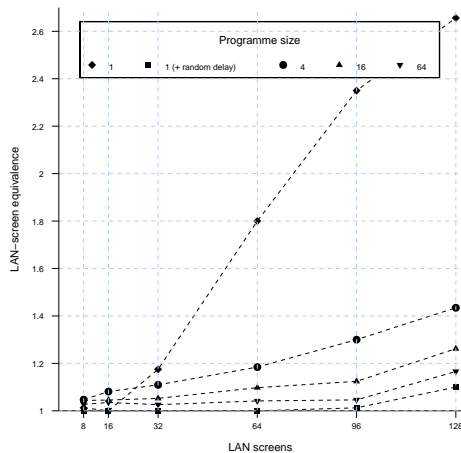


Figure 6. LAN-screen equivalence

Due to space restrictions, we show here the most meaningful result. Considering that 1 is the ideal number of connections per MO, any additional connection is a conflict that must be cancelled. It is interesting to measure the impact of this situation on central storage bandwidth consumption. Figure 6 shows that the whole local network is similar to 1.2 screen nodes working with no coordination at all. As can be observed, system behavior is not affected significantly by high concurrency or large screen populations. Surprisingly, with low concurrency (from 1 to 4 new inserted MOs) and many peers, the average LAN-screen equivalence grows substantially. The cause of this pathological behavior is that, with only a few MOs updated simultaneously on several peers, there is a high possibility of collisions for the first scheduled media (conflicts that are immediately detected and solved). In order to prevent this behavior, it is enough to introduce a small random delay at the very beginning of the coordination process.

7. Conclusions

We have presented a decentralized and structured P2P video content distribution network based on a DHT algorithm and designed to extend an existing on-demand streaming server. The design is especially oriented to big networks of server clients with highly-correlated server access patterns. The proposed coordination algorithm, built on top of a DHT, fulfills our goals since it ensures minimal coupling among screens, removes any single point of failure inside LANs and maximizes LAN scalability.

On the other hand, the P2P networks and the streaming server were easily composed through the VoDKA traders architecture. A screen/location priority framework was defined as the basis for the resource scheduler algorithm. This

algorithm was responsible for optimal resource usage and deadline achievement.

A big network of public information screens was used as case study, and, as shown in the experiment, the design is an appropriate approach for highly-correlated mass-scale content distribution, with interesting scalability and availability features.

References

- [1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, Dec. 2004.
- [2] J. L. Armstrong, M. Williams, R. Virding, and C. Wilkstrom. *Erlang for Concurrent Programming*. Prentice-Hall, Upper Saddle River, NJ, USA, 1993.
- [3] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.
- [4] V. M. Gulías, M. Barreiro, and J. L. Freire. VoDKA: Developing a video-on-demand server using distributed functional programming. *Journal of Functional Programming*, 15(4), 2005.
- [5] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
- [6] P. Maymounkov and D. Mazières. Kademia: A peer-to-peer information system based on the XOR metric. 2429:53–??, 2002.
- [7] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking, 2002.
- [8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, Aug. 2001.
- [9] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *LNCS*, 2218:329–339, 2001.
- [10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Technical Report TR-819, MIT, Mar. 2001.
- [11] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, Berkeley, CA, USA, 2001.