View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Ghent University Academic Bibliography

Low Delay Personal Content Storage in a Multi-service Access Network

Koert Vlaeminck, Filip De Turck, Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IBBT - IMEC
Gaston Crommenlaan 8 bus 201, B-9050 Gent, Belgium.
Tel: +32 9 33 14942, Fax: +32 9 33 14899
E-mail: koert.vlaeminck@intec.ugent.be

Abstract

A current important trend is the introduction of new services in the access and aggregation network, close to the end user. A major opportunity for such service enabled access networks is providing users with fast and reliable storage, allowing them to transparently access and share their personal data anytime, anywhere, while guaranteeing data retention. An important issue in deploying such a service is where to put the storage servers, minimizing the deployment cost without sacrificing performance. This paper presents and evaluates an optimization algorithm for solving this storage server placement problem, respecting the bandwidth constraints of the access and aggregation network while guaranteeing a low delay for accessing the storage service from any access node. It will be shown that the algorithm produces close to optimal results relatively fast.

Introduction

The implementation of the "broadband for all" objective and the large-scale introduction of new services, such as (interactive) digital television, has triggered an evolution of the access network infrastructure [1]. Where current access networks are often separated per service, telecom research envisions offering a myriad of broadband services over a converged access network, as depicted in Figure 1, bringing IP-awareness closer to the end user and increasing functional intelligence of the access and aggregation nodes [2].

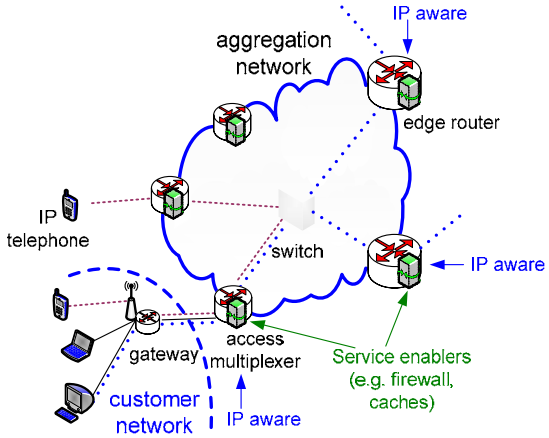

Figure 1: Evolution towards a converged, IP aware, full service access network.

A major opportunity of such multi-service access networks is providing anytime, anywhere access to fast and reliable storage. End users have an increasing amount of digital data (digital photo albums, digital home videos, a digital music and movie collection, etc), while harddisks are failure prone and recordable optical media only have limited archival lifespan [3]. A high speed network storage service, enabling a user to access his personal data at all times, wherever he is, while relieving him of the burden of meticulously backing up all his precious photos, home videos and other important files, would make life a lot easier. When designing such a storage service, an important question that needs answering is where to put the storage servers.

This paper presents and evaluates an optimization algorithm for determining the best locations for placing storage servers, minimizing the deployment cost while guaranteeing fast access from any access node. Both access and aggregation nodes are candidate storage server locations. A user can be served from any storage server. This requires a pervasive replication mechanism, as introduced in [13], creating a replica of a user's data wherever and whenever it is accessed. In this paper, it is assumed that server nodes have enough processing power and storage capacity to handle all requests. Furthermore, a read mostly data storage service is assumed. In many typical end user applications, such as a digital media library, data is read much more often than it is written. Read mostly data implies no strong consistency is required between the different replicas. Updates can be propagated periodically, at the same time deciding whether a replica should be retained or deleted (e.g. based on last access time, access frequency, etc), taking into account that a minimum set of replicas should be maintained at all times in order to ensure resilience. Since this kind of read mostly data storage service puts most strain on downstream traffic, only downstream traffic measures are considered when determining the best locations for placing a storage server.

The remainder of this paper is structured as follows: first, related work is presented. Then, the next section formalizes the problem description, after which the storage server placement algorithm is detailed. Next, the algorithm is evaluated using two typical access and aggregation network topologies. Finally concluding remarks are presented and some future work is discussed.

Related work

Many distributed filesystems exist today, ranging from client-server systems such as NFS [4], AFS [5] and Coda [6] over cluster filesystems (e.g. Lustre [7], GPFS [8] and the Google File System [9]) to global scale peer-to-peer filesystems (e.g. OceanStore [10], FARSITE [12] and Pangaea [13]).

Deployment of a transparent, fast, scalable and reliable storage service in the access and aggregation network is a new idea, triggered by the increased functional intelligence

in emerging service aware access networks. At first glance, such a service is very similar to content distribution networks (CDNs), where streaming content, which is very sensitive to jitter and packet loss, is replicated to so-called surrogate servers at the edge of the network in order to tackle the performance issues of the classical client-server approach [14]. However, CDNs are designed to distribute a limited amount of very popular content, while a personal content storage service stores a huge amount of relatively unpopular data. For such a service, where each user adds his own data, storage requirements are far larger and replica management a lot more complex. Furthermore, guaranteeing low latency and high bandwidth in an environment where end users each access different files simultaneously, requires data to be replicated even closer to the end user.

None of the distributed file systems enumerated above, were designed for large-scale deployment in an access and aggregation network environment. However, OceanStore, for which a prototype (Pond [11]) is being developed, seems a good candidate for this purpose. The OceanStore core system is composed of a multitude of highly connected pools, among which data is allowed to flow freely [10]. A pool could for instance be associated with an access and aggregation network. Most of the time, data will be accessed from within the pool, but when a user is traveling, his data is still accessible. Pangaea [13], with its pervasive replication mechanism that replicates data based on user activity, also seems a good candidate. Data that is only accessed from within the access and aggregation network will be kept locally. Users on the move will trigger replication of their data in other access and aggregation networks.

Before trying to deploy such a storage service and selecting the appropriate technology, however, the storage placement problem should be studied: how can a storage service be deployed in the access and aggregation network, using a minimum set of servers while guaranteeing fast access from any access node. This is the topic of this paper. The placement problem was introduced in [15] in a web server environment.

Model description

Before presenting the storage server placement algorithm in the next section, this section introduces some definitions and formalizes the problem description by presenting it as a Binary Integer Linear Programming (BILP) problem [16]. The goal is to find a minimum set of storage server locations that can service storage requests from users connected through any access node within a specified time (maximum delay constraint), respecting the maximum available bandwidth of each network link (bandwidth constraint).

Define $N=\{N_1,\ldots,N_n\}$ as the set of all nodes and $A=\{A_1,\ldots,A_m\}$ as the set of access nodes ($A \subset N$). d_{ij} represents the total delay from N_i to A_j. The maximum allowable delay for accessing a storage server is defined as d. Furthermore, define D_j as the bandwidth demand of access node A_j. Then $D = \sum_{j=1}^{m} D_j$ is the total demand in the access and aggregation network. Finally, define $E=\{E_1,\ldots,E_o\}$ as the set of all edges. The cost (delay) of edge E_l is defined as c_l and its bandwidth is b_l.

The problem can be formalized as a BILP problem as follows: define n boolean variables s_i ($s_i = 1$ if node N_i represents a server location, 0 otherwise). Furthermore introduce a set of continuous *path* variables: define P_{ij} as the set of all paths from node N_i to access node A_j and f_{ijk} as the flow on the k^{th} path P_{ijk} from N_i to A_j. Then f_{ij} represents the total flow from N_i to A_j.

Minimize:

$$z = \sum_{N_i \in N} s_i$$

subject to:

$$\sum_{E_l \in P_{ijk}} c_l \le d, \forall N_i \in N, \forall A_j \in A, \forall P_{ijk} \in P_{ij}$$

$$f_{ij} = \sum_{P_{ijk} \in P_{ij}} f_{ijk} \le s_i \times D_j, \forall N_i \in N, \forall A_j \in A$$

$$\sum_{N_i \in N} f_{ij} = D_j, \forall A_j \in A$$

$$\sum_{\forall P_{ijk} \ni E_l} f_{ijk} \le b_l, \forall E_l \in E$$

The first set of constraints ensures that only paths with a total delay $\le d$ are considered. The second set of constraints makes sure s_i is set to 1 if N_i provides flow to at least one access node A_j. The third set of constraints ensures that the demand of each access node is met. The final set of constraints makes sure the bandwidth constraint of each edge E_l is respected.

Storage server placement algorithm

For determining the minimum set of servers required for providing all users with a fast storage service from any access node within the bandwidth constraints of the access and aggregation network, a three-phase algorithm, was designed. This algorithm is described in pseudo-code in Figure 3 and explained below.

Phase 1: Transformation to a minimum cost flow problem.

As stated in the introduction, a pervasive replication algorithm ensures all user requests can be handled by any server. Furthermore, due to the nature of the storage service, not all requests coming from a single access node have to be handled by the same storage server. Therefore, the problem can be transformed to a single-commodity minimum cost flow problem [17]. This transformation, which is the first phase of the algorithm, is illustrated in Figure 2 and detailed below. First connect all access nodes A_j to a virtual *super target* node T with edge cost 0 and edge capacity D_j. In the example in Figure 2, A_1 has a demand of 2 Mbps and A_2 has a demand of 3 Mbps, so edges E_{A_1T} and E_{A_2T} have capacity 2 and 3 respectively.

Next connect a virtual *super source* node S to all candidate server locations (i.e. all nodes N_i that can reach at least one access node A_j with delay $\le d$) with infinite edge capacity. Since each edge will carry at most D flow, the total demand, this value can be used to represent infinity.

The total demand in Figure 3 is 5 Mbps, so edges from S have capacity 5. Note that S is not connected to N_1 in the example, since the total delay from N_1 to access node A_1 or A_2 is 5 ms, which is larger than the maximum allowable delay of 3 ms.

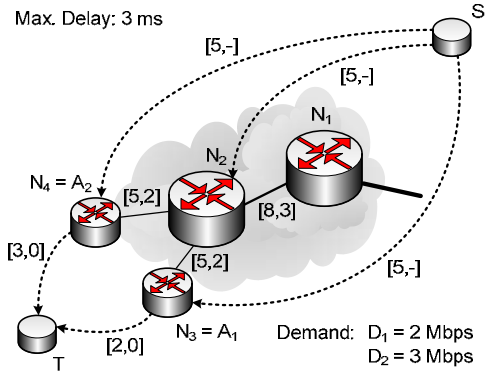

Figure 2: Transformation to a minimum cost flow problem. Access node A_1 has a demand of 2 Mbps, A_2 has a demand of 3 Mbps. The maximum delay is 3 ms, hence no connection from S to N_1 is available. The edge labels represent [residual bandwidth (Mbps), delay (ms)], a dash represents a dynamic value.

Now determine the minimum cost flow with demand D (5 in the example) from S to T. Since edges from an access node A_j to T have bandwidth D_j, the only way to solve the minimum cost flow problem with demand D is by routing the appropriate demand through each access node. The cost of the edges from S is adapted dynamically, making sure a server is used to its maximum, once a server location is chosen, as will be explained in the next subsection. In Figure 2, the dynamic costs are represented by a dash.

The transformation to a minimum cost flow problem is done in lines 7 to 18 of Figure 3. As shown in line 16, the initial candidate server edge cost for node N_i is a function of B_i (the outgoing bandwidth of N_i), C_i, (the number of access nodes N_i has in range) and d (the maximum delay). The function $f(C_i, B_i, d)$ is defined such that candidate server edges have an initial cost between d and $2d$. Higher values of B_i and C_i result in lower values for the initial cost of E_{SN_i}. This initial cost of E_{SN_i} represents the cost of installing a server at node N_i.

Access nodes are treated differently from other candidate server nodes, cf. line 11 of Figure 3. The initial cost of an access node A_j also depends on the demand D_j and the total incoming bandwidth (B'_i, where $A_j = N_i$). If the demand of an access node exceeds its total incoming bandwidth ($D_j > B'_i$), a server must be installed at that location. In that case:

$$f'(C_i, B_i, B'_i, D_j, d) = d$$

```
 1: S2 ← empty set
 2: S3 ← empty set
 3: F ← 0
 4: P ← empty set
 5: calculate Ci and Bi, ∀ Ni ∈ N
 6: calculate B'i, ∀ Ni ∈ N ∩ A
 7: add S and T to N
 8: for ∀ Ni ∈ N do
 9:     if ∃ j: Ni = Aj (Ni ∈ A) then
10:         add E_SNi to E
11:         b_SNi ← D, c_SNi ← f'(Ci, Bi, B'i, Dj, d)
12:         add E_AjT to E
13:         b_AjT ← 0, c_AjT ← 0
14:     else if Ci × Bi > 0 then
15:         add E_SNi to E
16:         b_SNi ← D, c_SNi ← f(Ci, Bi, d)
17:     end if
18: end for
19: while F < D do
20:     p ← shortest augmenting path from S to T
21:     s ← Ni : E_SNi ∈ p
22:     if Σ_{El∈p} cl > d + c_Ss then
23:         c_Ss ← +∞
24:     else
25:         if s ∉ S2 then
26:             add s to S2
27:             c_Ss ← 0
28:         end if
29:         f ← min{min{bl, El ∈ p},(D - F)}
30:         bl ← bl - f for edges El along p
31:         F ← F + f
32:         add p to P
33:     end if
34: end while
35: while P not empty do
36:     p ← last p from P
37:     if p does not contain s ∈ S3 do
38:         Aj ← access node where E_AjT ∈ p
39:         select s ∈ S2 closest to Aj on p
40:         add s to S3
41:     end if
42:     remove p from P
43: end while
```

Figure 3: Storage server placement algorithm: 3-phase algorithm for determining the minimum set of servers providing a storage service with guaranteed delay to all access nodes, respecting the available bandwidth of the network links. $S_2 \subset N$ is defined as the set of server locations selected in phase 2 of the algorithm and $S_3 \subset S_2$ as the set of server locations that remain after phase 3 of the algorithm. C_i is defined as the number of access nodes A_j that can be served from N_i with delay $d_{ij} \leq d$, i.e. $C_i = \sum_{j=1}^{m} C_{ij}$, and c_l as the cost (delay) of a single edge E_l. B_i is defined as the total outgoing bandwidth of node N_i and B'_i as the total incoming bandwidth of node N_i. F represents the total flow that has already been realized by the servers currently in the set S_2. P is the set of successive shortest augmenting paths that are computed during the second phase of the algorithm.

If $D_j \leq B'_i$ and the access node also has other access nodes in range ($C_i > 1$, since C_i includes the access node itself), it is treated as any other candidate server:

$$f'(C_i, B_i, B'_i, D_j, d) = f(C_i, B_i + D_j, d)$$

Else, selecting the access node as server location should be avoided when possible:

$$f'(C_i, B_i, B'_i, D_j, d) = 3d + 1$$

The algorithm always finds a solution respecting the maximum delay constraint and bandwidth constraints: worst case a server is installed at each access node. Since the initial cost of all edges E_{SN_i}, hence the cost of installing a server at N_i, is at least d, an algorithm solving the minimum cost flow problem will first add flows to access nodes in range from already installed servers (additional cost $\leq d$), before installing a new server.

Phase 2: Solving the minimum cost flow problem

The second phase of the algorithm is solving this minimum cost flow problem, using a (modified) successive shortest path algorithm, as introduced by Busacker and Gowen [18]. The cost of candidate server edges is adapted dynamically in order to promote already installed servers and to ensure a maximum delay for accessing a server from any access node.

At each iteration of the modified successive shortest path algorithm, a flow along the current shortest path from S to T is added. Since S is only connected to candidate server locations and only access nodes are connected to T, this implies a flow is added from the current best candidate server node N_i (lowest cost of E_{SN_i}), to the access node A_j closest to it. Candidate server nodes with a lot of access nodes in range and a large outgoing bandwidth will be selected first, since they have the lowest initial cost for edge E_{SN_i}.

When a node N_i is selected as server location, the cost of E_{SN_i} is set to zero, making sure this server is used as much as possible: the cost of adding a flow from an installed N_i to an access node A_j in range from N_i ($\leq d$) will always be lower than the cost of installing a new server, which is at least d augmented with the cost of the path from this new server to the closest access node. The only exception is when the network contains access nodes that have a demand larger than their incoming bandwidth. The total cost of a flow from S to T through such an access node is d. The minimum cost flow problem will start by installing servers at those access nodes.

Once no more access nodes A_j can be reached from N_i with delay $d_{ij} \leq d$, the cost of the edge from S to N_i is set to infinity, forcing the successive shortest path algorithm to select another server location.

Lines 19 to 34 of Figure 3 show the modified successive shortest path algorithm, where line 27 sets the cost of a candidate server edge to zero, once a server is installed at that location and line 23 sets the cost of a candidate server edge to infinity when a server at that location no longer has any access nodes in range. After each iteration, the residual bandwidth of the edges along the path is decreased with the amount of flow (f) that was added (cf. line 30).

Phase 3:Redistributing the flow

It is possible that at some iteration of the successive shortest path algorithm, a new server is added that is closer to some access node A_j that was already assigned a server in an earlier iteration. It is even possible that already selected server locations are obsoleted by some later server location additions. This is illustrated in Figure 4 using a simple tree topology access and aggregation network. Assume each access node has a demand of 5 Mbps, with a maximum allowable delay of 5 ms. The modified successive shortest path algorithm will first select node N_1 as server location, since it has the highest total outgoing bandwidth and all access nodes in range. However, the 8 Mbps links from N_1 can not carry all demand and nodes N_2 and N_3 are also added as server locations (access nodes are avoided), obsoleting N_1.

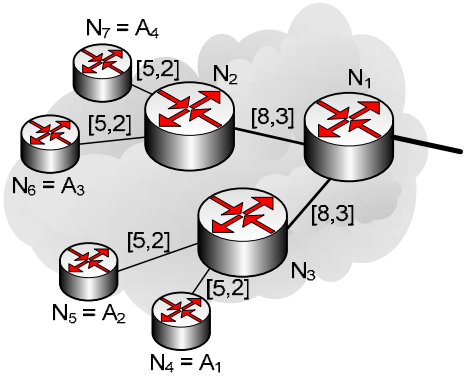

Figure 4: Each access node has a demand of 5 Mbps, with a maximum allowable delay of 5 ms. The successive shortest path algorithm will first select node N_1 as server location. However, the 8 Mbps links from N_1 can not carry all demand, resulting in the addition of N_2 and N_3 as server locations, obsoleting N_1. Again, the edge labels represent [residual bandwidth (Mbps), delay (ms)].

To resolve this problem, the third and final phase of the algorithm redistributes flows among the selected server locations in such a way that each access node is served by the closest server. Server locations that provide zero remaining flow are removed from the list. This can easily be done by traversing the set of shortest augmenting paths P, selected during the second phase, in reverse order. At each iteration, the server closest to the access node is selected, unless the path contains a server previously selected during phase 3. This is shown in lines 35 to 43 of Figure 3.

Implementation and evaluation

The algorithm was implemented using the Telecom Research Software library (TRS) [19], developed at the broadband communications research group of Ghent University. It is a Java network library, aiding in the representation of (multilayer) networks and related concepts.

Use cases

For evaluating the algorithm, the number of storage servers required for providing a fast storage service was computed for different demands and maximum allowable delay in two typical access and aggregation network topologies: a mesh of trees topology, typically used in DSL deployment, and a ring of rings topology, used in cable Internet access deployment. Both topologies were downscaled to 250 access nodes in order to keep the simulation results verifiable. For simplicity, the hopcount was used as a delay measure.

The example mesh of trees topology is depicted in Figure 5. The 250 access nodes are at the leaf nodes of 5 trees of depth 2, aggregated per 10 at depth 2 and per 5 at depth 1. The available bandwidth between the access nodes and the first aggregation node is 600 Mbps. Those aggregation nodes are connected to the root of a tree by a 1.2 Gbps link each. The five trees are interconnected by a full mesh of 3 Gbps links.

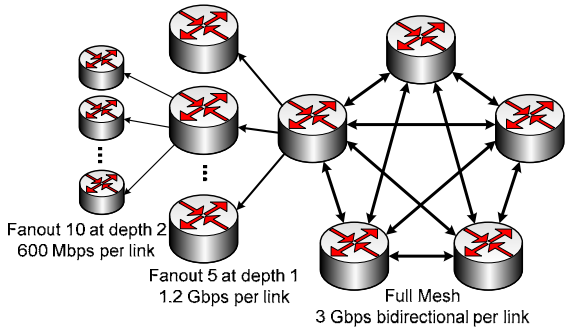

Figure 5: Use case 1: a downscaled mesh of trees access and aggregation network topology, used for DSL deployment.

The example ring of rings topology, is depicted in Figure 6. It consists of five unidirectional 6-node secondary rings of 2 Gbps. One node connects the secondary ring to a primary ring of 10 Gbps (also unidirectional). The other 5 secondary ring nodes each connect to 10 access nodes trough a shared medium of 2 Gbps, resulting in a total of 250 access nodes.

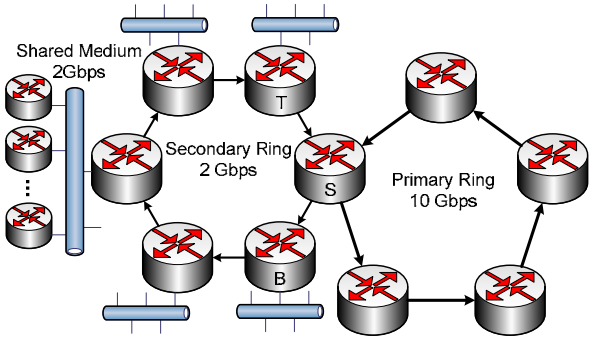

Figure 6: Use case 2: a downscaled ring of rings access and aggregation network topology, used for cable Internet access deployment. The markings *B*, *S* and *T* are used in the discussion of the simulation results.

Simulation results

Results from the simulations of the example mesh of trees access and aggregation network topology are summarized in the Table 1. Due to the full mesh, a root node of one of the trees can reach any access node in at most 3 hops. Up to a demand of 60 Mbps per access node, the access and aggregation network has sufficient bandwidth available to handle all requests and the number of servers required, is determined by the maximum delay constraint. The table clearly shows that for higher demands per access node, the benefit of allowing a larger maximum delay decreases. This is caused by the statistical multiplexation of bandwidth on upstream links (e.g. in the topology of Figure 5, ten 600 Mbps access node links are multiplexed on a single 1.2 Gbps link). For demands greater than 600 Mbps per access node, a server is to be installed at each access node, due to bandwidth constraints of the access nodes' uplinks. It can easily be verified that the algorithm finds the optimal result on this rather simple topology. As a reference, the optimal delay only results, calculated by hand using Figure 5, were added to Table 1.

Table 1: Simulation results for the mesh of trees access and aggregation network topology, depicted in Figure 5. The table gives the required number of servers for a varying demand per access node and increasing maximum delay. As a reference, the optimal delay only results were added to the table.

	max. 1 hop	max. 2 hops	max. 3 hops
0 - 60 Mbps	25	5	1
61 - 120 Mbps	25	5	2
121 - 600 Mbps	25	25	25
> 600 Mbps	250	250	250
delay only	25	5	1

Results from the simulations on the example ring of rings topology are depicted in Figure 7. For comparison, the optimal delay only results of the server placement problem were added to the graph. These can easily be calculated using Figure 6: a server installed at the primary ring node following *S* can reach all access nodes (the access nodes connected to *T* are furthest away) in at most 10 hops, a maximum delay of 8 or 9 hops requires at least two servers on the primary ring, etc.

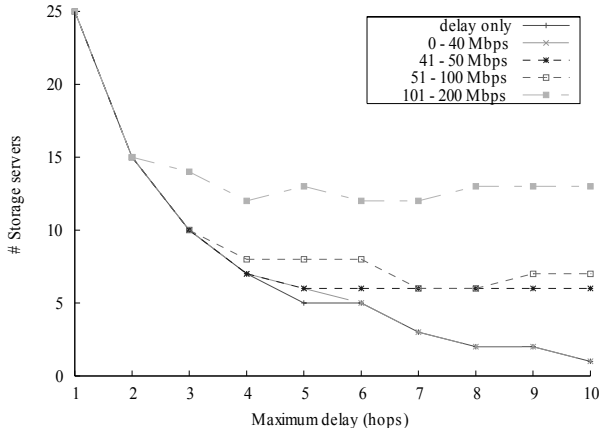

Figure 7: Simulation results for the ring of rings access and aggregation network topology, as depicted in Fig. 6. The delay only results can easily be obtained by hand. For demands > 200 Mbps per access node, the shared medium connecting the access nodes to the secondary rings becomes a bottleneck and multiple servers per secondary ring aggregation node are required, i.e. $10-\lfloor 2Gbps/D_A \rfloor$ server nodes, with D_A the average demand per access node connected to the shared medium.

Again, the simulation results clearly show that for higher demands per access nodes, the benefit of allowing a larger maximum delay decreases. The example ring of rings network topology has enough bandwidth available to handle demands of up to 40 Mbps per access node before bandwidth constraints of the links begin to influence the result.

However, for a maximum delay of 5 hops, the graph in Figure 7 shows a small deviation between the calculated result for low bandwidth demands (≤ 40 Mbps) and the optimal delay only result. This is caused by the nature of the heuristic: the storage server placement algorithm assigns a quality to each candidate server node, based on the number of access nodes it has in range and its total outgoing bandwidth. While the node marked by *B* in Figure 6 can serve all nodes connected to its secondary ring, resulting in a total of 5 servers, the algorithm will first select the node marked by *S*. *S* has a total of 100 access nodes in range, while *B* only has 50 access nodes in range. Furthermore, since it's a primary ring node, *S* has a larger outgoing bandwidth than node *B*. However, access nodes connected to *T* are out of range from *S*, resulting in a suboptimal solution. To allow the algorithm to break free from such suboptimal solutions, a stochastic parameter was added to the node quality, producing close to optimal results.

The graph of Figure 7 plots the result after 10 iterations, each iteration taking under a minute on modern PC hardware[1]. For a demand > 200 Mbps per access node, the shared medium connecting the access nodes to the secondary ring becomes a bottleneck, requiring multiple servers per secondary ring aggregation node, i.e. $A_S - \lfloor b_S / D_A \rfloor$ servers. In this equation, A_s is the number of access nodes connected to the shared medium, D_A is the average demand of those access nodes and b_s is the bandwidth of the shared medium. In the example topology, this results in $10 - \lfloor 2Gbps / D_A \rfloor$ access nodes per secondary ring aggregation node, for D_A > 200 Mbps. In order to keep the graph readable, this is not plotted in Figure 7.

Conclusions and future work

This paper presented and evaluated an optimization algorithm for determining a minimum set of storage server locations in the access and aggregation network that can service the *read-mostly* storage requests of for instance a digital media library service from users connected through any access node within a specified time. This algorithm takes into account a maximum delay constraint for guaranteeing fast access and respects the bandwidth constraints of the network links.

The two typical access and aggregation network topologies were used for evaluating the algorithm: a mesh of trees, used in DSL deployment, and a ring of rings, used for cable Internet access deployment. The algorithm is shown to produce close to optimal results relatively fast. An important trend that can be noticed from the simulations is that, due to the statistical multiplexation of bandwidth on upstream links, the benefit of allowing a larger maximum delay decreases for increasing demand.

Work is in progress to extend the storage server placement algorithm to finding a minimum set of storage servers for servicing *read-write* storage requests from any access node within a specified time. In this case, read and write requests from a user in a single session are to be redirected to the same storage server. Future plans include extending the presented algorithm to deal with additional constraints, more accurately modeling storage server limitations.

References

1. T. Wauters, K. Vlaeminck, *et al*, IPTV Deployment: Trigger for Advanced Network Services!, FITCE'06, Athens, Greece, August / September 2006.
2. T. Stevens, K. Vlaeminck, *et al*, Deployment of service aware access networks through IPv6, 8th ConTEL'05, Zagreb, Croatia, June 2005, Vol. 1, p. 7-14.
3. T. Vitale, Digital Imaging in Conservation: File Storage, AIC News, Vol. 31, Nr. 1, January 2006.
4. B. Callaghan, B. Pawlowski, *et al*, RFC1813: NFS version 3 protocol specification, June 1995, http://www.faqs.org/rfcs/rfc1813.html.
5. J. Howard, M. Kazar, *et al*, Scale and performance in a distributed file system, ACM TOCS, Vol. 6, Nr. 1, 1988.
6. L B. Mummert, Maria R. Ebling, *et al*, Exploiting weak connectivity for mobile file access, SOSP'95, Copper Mountain, CO, USA, Dec.ember 1995, p. 143-155.
7. Cluster File Systems Inc, Lustre: A Scalable, High-Performance File System, white paper, November 2002, http://www.lustre.org/docs/whitepaper.pdf.
8. T. Jones, A. Koniges, *et al*, Performance of the IBM general parallel file system, IPDPS'00, May 2000, p. 673-681.
9. S. Ghemawat, H. Gobioff, *et al*, The Google File System, SOSP'03, Bolgon Landing, New York, USA, October 2003, p. 29-43.
10. J. Kubiatowicz, D. Bindel, *et al*, OceanStore: An Architecture for Global-Scale Persistent Storage, ASPLOS'00, November 2000.
11. S. Rhea, P. Eaton, *et al*, Pond: the OceanStore Prototype, FAST'03, March 2003.
12. W. J. Bolosky, J. R. Douceur, *et al*, Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs, ACM SIGMETRICS, June 2000, p. 34-43.
13. Y. Saito, C. Karamanolis, *et al*, Taming aggressive replication in the Pangaea wide-area file system, OSDI'02, December 2002, p. 15-30.
14. J. Coppens, T. Wauters, *et al*, Evaluation of replica placement and retrieval algorithms in self-organizing CDNs, SELFMAN'05, Nice, France, May 2005.
15. L. Qiu, V. Padmanabhan, *et al*, On the placement of web server replicas, InfoCom'01, Anchorage, Alaska, April 2001, p. 1587-1596.
16. E. L. Johnson, G. L. Nemhauser, *et al*, Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition, INFORMS Journal on Computing, Vol. 12, Nr. 1, January 2000, p. 2-23.
17. R. K. Ahuja, T. L. Magnanti, *et al*, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.
18. R. G. Busacker, P. J. Gowen, A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns, Technical Paper ORO-TP-15, Operations Research Office, The Johns Hopkins University, Bethesda, Maryland, 1961.
19. K. Casier, S. Verbrugge, *et al*, Using aspect-oriented programming for event-handling in a telecom research softwave library, ICSR-8, Madrid, Spain, July 2004.

[1] The simulations were executed on an AMD64 3000+ with 512 MB of memory, running Linux and the Sun J2SE 5.0 Runtime Environment