Caching strategies for Personal Content Storage Grids

Niels Sluijs, Koert Vlaeminck, Tim Wauters, Bart Dhoedt, Filip De Turck, Piet Demeester
Department of Information Technology – IBCN – IBBT – IMEC
Ghent University
Ghent, Belgium

Abstract - *One of the latest trends in the Internet is the online sharing of personal files with others. Ideally, an online storage service, enabling access to and sharing of your personal content, should be available anytime, anywhere. To realize such a service, the system should be scalable in the number of users and files, and the delay should be limited. In order to meet the requirements an optimized caching strategy for personal content can be used to increase the efficiency of a network. We discuss a caching strategy and an evaluation of analytical and simulated results in this paper. We extended OptorSim to obtain simulation results. The results of the simulation and analytical calculation closely match, implying that the simulator can be used to observe more complex problems. The presented approach is used for evaluating caching strategies to increase efficiency in deploying a personal content storage service.*

Keywords: Grid computing, caching algorithms, scalability, simulations, personal content.

1 Introduction

Using the Internet, one is able to communicate and share information with others. One of the latest trends is to share your *personal content* with other people. Personal content typically consists of text documents, digital photos, music files, personal movies, etcetera. For instance you can share your personal movies or pictures on central-server architectures like YouTube [22] and Flickr [6].

However, systems that provide the possibility to store personal content for users still have limitations, namely: *scalability* in the number of users and files, and the *delay* caused by central-server architectures. Ultimately users want to have space available to store their personal content, where *access properties* of the online storage is the same as using your own hard disk. The main benefit of such a system is that one can access content fast at *any time* and from *anywhere*. Another advantage of such a system is that users can be relieved from the burden of making backups of their precious files.

In order to overcome the limitations caused by the infrastructure one can use the technology of *Grid computing*. Grid computing offers *computational* and *storage resources* in a transparent way to users. Transparency means that the exact geographical locations of the physical resources are made abstract for users [7]. In this way one tries to increase the utilization of underused resources, in order to enhance the efficiency of a system as a whole.

A Grid that provides the possibility to store personal files is called a *Personal Content Storage* (PCS) *Grid*. Although a lot of research has already been done into Grid technology, there has not been done a lot in dimensioning cache sizes for a Grid that stores personal content. This is due to the fact that Grids, at the time of writing, are mostly used to solve large and computationally complex problems. Most of the research that tries to improve Grid technology tries to increase the efficiency of the utilization of the computational resources, thereby realizing huge savings on execution times of computationally intensive jobs. However, savings can also be obtained by increasing the efficiency of data transfers. Grids that are optimized to transfer data efficiently for computational intensive jobs are called *data Grids*.

A PCS Grid differs from a data Grid, in the sense that the latter is designed to store a set of relatively large data files, which will typically be accessed by a few hundred to a few thousand researchers. In contrast, a PCS Grid will store a large set of relatively small data files and will typically be accessed by thousands to millions of users.

When designing such a PCS Grid, an important question that needs answering is where files are cached in the Grid, in order to meet the user requirements. With data caching in the Grid, frequently accessed files can be brought closer to the user(s) that are requesting that file often. In this way a big part of the Grid can be relieved and the quality of the service of the Grid will remain, even when the number of users and files grows.

At first glance, such a caching strategy seems very similar to caching strategies in *Content Distribution Networks* (CDN). In a CDN 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 [3]. However, CDNs are designed to distribute a limited amount of very popular content, while a PCS Grid will store a huge amount of relatively unpopular content. For such a PCS Grid, where each user adds his/her data, storage requirements are more important. Furthermore, guaranteeing low latency and high bandwidth in an environment where end users each access different files simultaneously, requires data to be cached even closer to the end user.

Nowadays there exists many distributed file systems, ranging from client-server systems (e.g. NFS [2], AFS [9] and Coda [15]) over cluster file systems (e.g. Lustre [4], GPFS [10] and the Google File System [8]) to global scale peer-to-peer file systems (e.g. OceanStore [13], FARSITE [1] and Pangaea [20]). 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 [19]) is being developed, seems a good candidate for this purpose. The OceanStore's core system is composed of a multitude of highly connected pools, among which data is allowed to flow freely [13]. 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 [20], 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.

We describe a caching strategy and an evaluation of the results in this paper. A description of the caching strategy and the test scenario that we use is provided in section 2. The measurements that we did with the discrete event simulator are described in section 3. Finally, we provide a discussion and future work in section 4.

2 Personal content management

As stated in the introduction, this section presents a test scenario for personal content storage. Figure 1 represents a typical (Digital Subscriber Line – DSL) access network with a tree topology, having split s and depth d. Users at the leaf nodes are connected to the level one caches, the server is located at level d. We assume that sufficient capacity is available on the links.

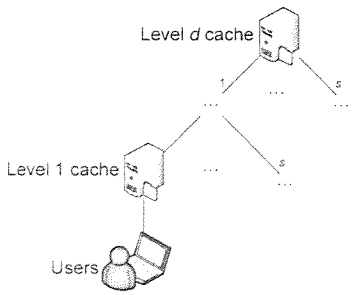

Figure 1: Access network with a tree topology, having split s and depth d. Users are situated at the leaf nodes and connected to a level one cache. The server cache is located at level d. On the links sufficient capacity is available.

In our simulations, users make their personal files available in the network by uploading them to the central server. The uploaded file of a user is cached at the caches on the path to the central server. In total, N files with equal size are uploaded, on average ones every A seconds. The number of uploads is a lot smaller than the number of downloads. The popularity of each file is equal at the time of upload, but decreases *exponentially* afterwards. The popularity distribution of file i at time t, where λ_0 represents the initial request rate, τ is a time constant and $T_{i,0}$ determines the upload time stamp of file i, is given by (1):

$$\lambda_i(t) = \lambda_0 \cdot e^{\frac{-(t-T_{i,0})}{\tau}}, \text{ where } t > T_{i,0} \qquad (1)$$

The function in equation (1) describes an exponentially decreasing popularity for files, which implies that the longer a file is in the system, the less attractive it will be for a user to request it. When a user downloads a file for the first time, each intermediate cache stores that file locally and serves consecutive requests for that file. When a cache is full, older files are deleted according to a *Least Recently Used* (LRU) policy.

Although previous studies on proxy caching techniques [14] or distributed replica placement strategies for CDNs [11], [12], [17] show that greedy algorithms that take distance metrics and content popularity into account perform better than more straightforward heuristics, such as LRU or LFU (Least Frequently Used). We use the LRU algorithm to be able to compare our analytical solution with the simulation results.

2.1 Test scenario

Before we present our analytical solutions our simulation results, we have to define the parameters that we use in our test scenario. For the parameters depth d and split s, we take the value four. This implies that we have 85 cache servers in the topology and in total 64 users, where each user represents the aggregation of an access network. Users are connected to a single level one cache in the network.

Figure 2 shows an example of the popularity distribution for each file, with λ_0 = 0.01/s and τ = 100,000 s.

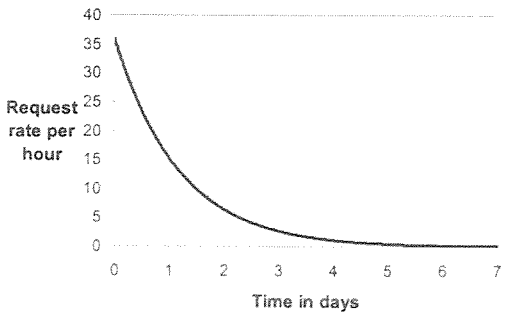

Figure 2: Popularity distribution $\lambda_i(t)$ for a file i, with λ_0 = 0.01/s and τ = 100,000 s, assumed that the file is uploaded at time 0. The popularity distribution is represented as the request rate per hour against the time in days.

The area below the function shown in Figure 2 represents the total number of file requests that are made for each file. This means that a total number of $\int_{T_{i,0}}^{\infty} \lambda_i(t) = \lambda_0 \cdot \tau = 1000$ requests (downloads) are made per file. The number of files N is set to 1,024 and each user has an equal probability to download a file, implying that users have no preference for a certain file. Since we have stated that we assume there is sufficient capacity available on the links, the file size is neglected. Furthermore, we assume that the cache at level d has enough capacity to store all files that will be uploaded. The last parameter that we need to define is the inter-arrival time between uploads A, we assume that every hour a new file is uploaded by a random user.

2.2 Storage dimensioning

First, we present an analytical solution for the content placement that determines the storage capacity on each level of the network, so that the total *cache serve ratio* on each level is equal. *The cache serve ratio is the ratio between the number of files served by a cache layer and the total number of requested files.* An equal cache serve ratio implies that load is balanced for each cache level. Afterwards, these results are compared to those of the discrete event simulator, using the LRU caching algorithm.

2.2.1 Analytical model

When file i becomes available in the network at time $T_{i,0}$, it should be located at each of the caches on level one, closest to the end users, so that the delay and transport cost are minimized.

As its popularity decreases, a file will be relocated to all caches on level two after $T_{i,0} + t_1$ seconds, and so on, until the file is stored in the server at the top after $T_{i,0} + t_{d-1}$ seconds. To achieve an equal total cache serve ratio for this file on each level in the tree, all t_l ($l = 1, 2, \ldots, d-1$) have to be calculated so that the total number of requests made for that file in the intervals $[T_{i,0} + t_0, T_{i,0} + t_1], \ldots, [T_{i,0} + t_{d-1}, \infty]$ is equal, or in other words:

$$\int_{T_{i,0}}^{t_l} \lambda_i(t) = \frac{l \cdot \lambda_0 \cdot \tau}{d}, \text{ or } t_l = -\tau \cdot \ln(1 - \frac{1}{d}) + T_{i,0} \quad (2)$$

When the same procedure is used for all files, each level in the tree serves an equal total number of requests. As we assume that in non-equilibrium steady state new files enter the system at a (nearly) constant rate, the cache serve rate per level is always equally distributed.

2.2.2 Analytical example

When 1,024 files are available, each with the request rate shown in Figure 2, on a tree network with depth $d = 4$ and split $s = 4$, we find that $t_1 = 8.0$ hours, $t_2 = 19.3$ hours and $t_3 = 38.5$ hours. If a constant entry rate of one file per hour is assumed, this means that each cache on level one has to store the eight most recent (i.e. most popular) files, each cache on level two the next eleven most popular files, each cache on level three the next nineteen files and the central server the least popular already available files.

Doubling the entry rate doubles the number of files on each level, the split has no influence. The solution for different values of the depth d is shown in Figure 3.

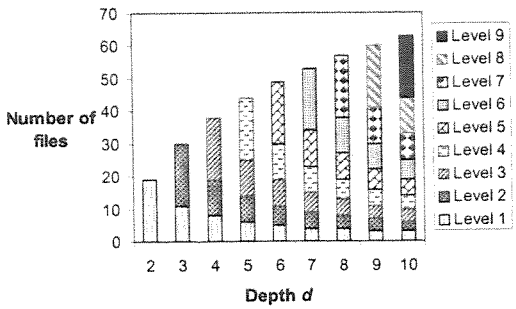

Figure 3: Cache size on each level expressed in number of files, for different tree depths d, to get an equally distributed cache serve rate per cache level. We assume that the inter-arrival time of new files in the system is 3,600 seconds.

Since the cache server at level d is assumed to have enough capacity available to store all files, only the cache sizes at level one till level d-1 have to be calculated. Figure 3 shows that increasing the number of cache levels results in a decrease of the needed cache size at a cache level, in order to have an equally distributed cache serve ratio. However, the sum of the files to be stored over all cache levels increases when parameter depth d increases.

2.3 Content distribution rate

We know, however, that in a more realistic situation, where an LRU caching algorithm is used instead of an optimal dynamic replacement over all caches of the appropriate level, the location of the files is very suboptimal. In this section, we study the time it takes to store one file on as many level one caches as possible, through individual downloads. In section *'3 Simulation and evaluation'* we compare these results to those of the discrete event simulator, using the LRU caching algorithm.

2.3.1 Analytical model

At random, each of the users at the leaf nodes sends one of the M ($= \lambda_0 \cdot \tau$) requests for a file i to one of the J ($= s^{d-1}$) caches located at the lowest level in the tree. We look for the probability $P[k]$ that k of the J caches store the requested file, after request m ($m = 1, \ldots, M$). In the beginning, $P[0] = 1$, $P[k \neq 0] = 0$. After one request ($m = 1$), $P[1] = 1$, $P[k \neq 1] = 0$. The probability that the first k caches store file i, and the other $J - k$ cache do not store file i is given by $P[k] = C_J^k$.

Identify S_j ($j = 1, \ldots, k$) as the set of possible ways to distribute all m requests over k caches so that cache j remains empty. All sets S_j can be combined into

intersections of p subsets, each with cardinality $(k - p)^m$ to distribute m requests over $k - p$ caches, in C_k^p different ways. Following the principle of inclusion and exclusion [21], the number of possible distributions with at least one cache where a file i is not stored is represented by equation (3):

$$\begin{aligned}\#(S_1 \cup ... \cup S_k) &= \sum(\#S_j) - \sum(\#(S_j \cap S_h)) + ... \\ &= C_k^1 (k-1)^m - C_k^2 (k-2)^m + ...\end{aligned} \quad (3)$$

The number of possible distributions where non of the first k caches is empty is then $k^m - C_k^1(k-1)^m + C_k^2(k-2)^m - ...$

In total, J^m distributions (all with an equal probability) are possible, so that the general probability distribution of the number of caches at level one store a file i after m downloads becomes:

$$P[k] = \frac{C_J^k \cdot (k^m - C_k^1 (k-1)^m + C_k^2 (k-2)^m - ...)}{J^m} \quad (4)$$

In the next section we use equation (4) in an analytical example.

2.3.2 Numerical example

For the same parameters values as described in '*2.1 Test scenario*', 64 level one caches are present and file i is requested a thousand times. A plot (see Figure 7) of the probability distribution of the number of level one caches storing file i after one hundred downloads for this example is given in *'3.2 Content distribution rate'*. The analytical result is that after a hundred downloads of a file i on average 51 caches store file i. For the development of the number of filled caches with file i against the number of downloads we refer to Figure 8 in *'3.2 Content distribution rate'*. We notice that the optimal situation (i.e. all caches store the particular file i) in Figure 8 is only (almost) reached after two hundred requests and not immediately, as we presumed in the analytical model described in section *'2.2 Storage dimensioning'*.

3 Simulation and evaluation

Besides solving the problem analytically, we use a *discrete event simulator* to approximate the statistics. In [18] a number of data Grid simulators are described, like: Bricks, SimGrid, GridSim, GangSim and OptorSim. We use the simulator OptorSim [16], since it is an event driven simulator and was originally designed to explore effects of dynamic data replication in the European DataGrid (EDG) project [5].

We use the same parameter values as described in *'2.1 Test scenario'*; this means that there are 64 level one caches, 1,024 different files and each file is upload once and downloaded a thousand times. For the simulation we use the calculated optimal values for cache sizes at each level; the level one caches have a capacity to store the eight most popular files, the caches located at level two can store the next eleven most popular files and the level three caches are able to store the next nineteen most popular files. Since the cache at level four should have sufficient capacity to store all files, this cache can store 1,024 files. In the next two subsections we show that the analytically obtained results and the results obtained with the simulator are similar.

3.1 Storage dimensioning

Since we used the analytical calculated cache size in our simulations, we should get an approximately constant cache serve ratio for each of the cache layers in *non-equilibrium steady state*. In Figure 4 and Figure 5 the convergence of the cache serve ratios for each cache level in relation to the number of downloads of all files, is presented. Figure 4 depicts the first 2500 downloads and Figure 5 shows the convergence over all downloads.

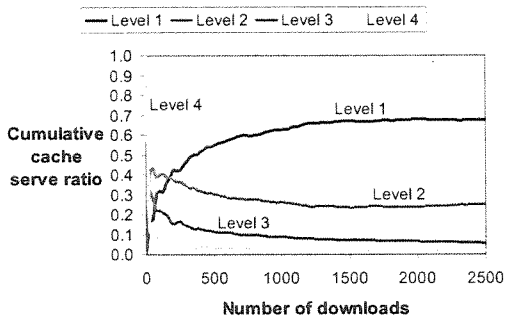

Figure 4: Convergence of the cumulative cache serve ratios for each cache level against the total of number of downloads during the simulation. The number of downloads depicted in this figure, is limited to 2500.

In the first 2500 downloads, you see that the cache serve ratio of cache level four starts at 1.0 and the other cache levels begin at 0.0, after the first download. In the simulation a user does an upload of the file to the cache at level four (i.e. the central server), and on every cache on the path from the user to the cache at level four the file is cached. When a user downloads the file for the first time in this simulation, the closest copy of the file was located at the cache in level four. This explains why the cache serve ratio for level four is 1.0 after one download.

The further developments of the cumulative cache serve ratios in Figure 4, is that the caches at level one mainly serve the users; this agrees with the observation in Figure 8, Figure 8 shows that after approximately two hundred downloads all caches at level one store file i and thus serve the requests to file i.

The reason why the cache serve ratio for cache layer two is higher than the cache serve ratio for cache level three (and four), is that when a user downloads a file for the first time it uses the closest replica of a file in the system. If one of the direct neighbors (i.e. users that share the same cache at level two) of the user already downloaded the file, the file is available at cache level two and cache level two gets a cache hit. The same

explanation can be given for the difference between the cache serve ratios of cache level two and three.

As mentioned above, Figure 5 presents the convergence of the cache serve ratios for all downloads.

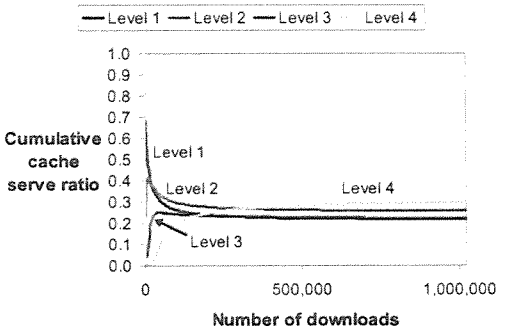

Figure 5: Convergence of the cumulative cache serve ratios for each cache level against the total of number of downloads during the simulation.

When the total number of downloads advances, the cache serve ratio of cache level one decreases, since more new files enter the system. The relative number of older files (files that are served by cache level two, three or four) increases, but users will still produce some requests to these files. The same is valid for the caches at level two and three. For these lower level caches, the drop in cache serve ratio happens after more downloads, since these caches store the next most popular files. Eventually the caches at level four becomes important when the caches at level three have no space left to store the old files. The requests that users make to these old files will all be served by cache level four, so the cache serve ratio of level four will increase.

The cache serve ratio numbers of the caches at the end of the simulation are summarized in Table 6.

Cache level	*Cache serve ratio*
Level 1	0.2196
Level 2	0.2584
Level 3	0.2241
Level 4	0.2978

Table 6: Cache serve ratios for each cache level at the end of the simulation.

According to the analytical example in *'2.2 Storage dimensioning'* cache serve ratio should be equal for each cache level. Since we have four cache levels, the cache serve ratio should be 0.25.

The ratios of Table 6 more or less correspond to the calculated values. The cache serve ratio of cache level one has the lowest cache serve ratio. This is due to the empty caches, when the simulation starts. It will take some time, after a user uploads the ninth file, before the first file is deleted from cache level one and further requests to the first file are served by cache level two. Since all requests of a user that downloads a file for the first time is handled by a cache level other than cache level one, cache level one misses requests that were assigned to cache level one in the analytical calculations. Cache level four profits from this, which explains why the ratio of this cache level is higher. The cache serve ratios of cache level two and three more closely approximate to the calculated value, despite all caches start empty.

3.2 Content distribution rate

In this section we study the time (or number of downloads from file *i*) it takes to store one file on as many level one caches as possible, through individual downloads. According to the analytical calculations after one hundred downloads of file *i* on average 51 caches should store file *i*. This is visualized in Figure 7.

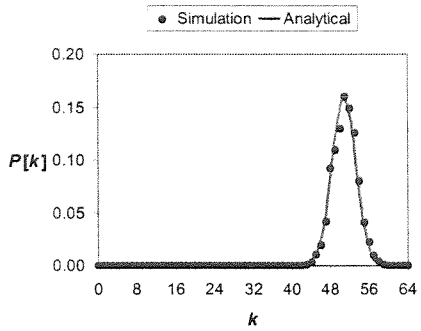

Figure 7: Probability distribution of the number of level one caches storing file *i* after one hundred downloads. The line depicts the analytical solution of equation (4); $m = 100$ and $J = 64$. The dots represent the measured values, obtained from the simulation.

Besides the analytical solution, the measured values of the simulation are also depicted in Figure 7. From Figure 7 we can conclude that the probability distribution obtained with the simulation confirms the analytical probability distribution. The small difference is due to the random number generator in the simulations.

Besides the probability distribution of the filled level one cache with file *i* after one hundred downloads of file *i*, we are also interested in the evolution (i.e. in number of downloads) of the average number of filled level one caches in time. Figure 8 provides this information.

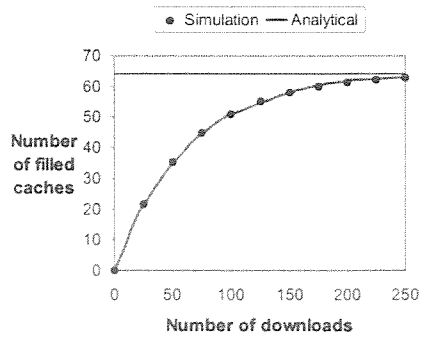

Figure 8: Average number of level one caches that store a file *i* in relation to the total number of downloads of a file *i*.

Both the analytical solution and the measured average number of level one caches that store a file *i* in relation to the total number of downloads of a file *i* are shown. The upper limit in this example is 64, since there are only 64 level one caches present. We can conclude that the measured approximation fits the analytical solution. The small differences can again be explained by using a random number generator in the simulation.

4 Conclusion

To realize a PCS Grid, the Grid should be scalable in the number of users and files, and the delay should be limited. In order to meet these requirements an optimized caching strategy for personal content should be used to increase the efficiency of a Grid. We show with a basic test scenario on an access network with a tree topology, that the results that we obtain with our simulator confirm analytical calculations.

We have determined the required cache capacities at each level of the tree network based on an analytical model, in order to obtain an equal *cache serve ratio* for each cache level. The simulation shows that the cache serve ratios converge closely to the calculated values.

We are aware that in a more realistic situation, where a *Least Recently Used* (LRU) caching algorithm is used instead of an optimal dynamic replacement over all caches of the appropriate level, the location of the files is suboptimal. This is why we also studied the time it takes to store one file on as many level one caches as possible, through individual downloads. The measurements of the simulation of the probability distribution of the number of caches at level one that store a file that is downloaded one hundred times, approximates the analytical calculation closely.

Now that we have a simulator that is able to closely match analytical calculations, we will use it in future work to investigate properties and topologies for which analytical calculations are to complex. Future work will include studying different caching strategies, different topologies where links have a different and limited bandwidth, a more realistic file size distribution, and users having preferences for some common and their own files. Our study will lead to caching strategies where the user experience of the online storage system will be similar to using a local hard disk.

5 Acknowledgment

Filip De Turck acknowledges the F.W.O.-V. (Fund for Scientific Research – Flanders) for their support through a postdoctoral fellowship.

Part of this work has been funded by the IBBT PecMan project.

6 References

[1] Bolosky, W. J., Douceur, J. R., Ely, D., Theimer, M., *"Feasibility of a server-less distributed file system deployed on an existing set of desktop PCs"*, ACM SIGMETRICS, (2002), pp. 34 – 43;

[2] Callaghan, B., Pawlowski, B., Staubach, P., *"RFC1813: NFS version 3 protocol specification"*, 1995, http://www.faqs.org/rfcs/rfc1813.html, last visited in March 2007;

[3] Coppens, J., Wauters, T., De Turck, F., Dhoedt, B., Demeester, P., *"Evaluation of replica placement and retrieval algorithms in self-organizing CDNs"*, SELFMAN'05, (2005), 1 – 5;

[4] Cluster File Systems Inc., *"Lustre: A Scalable, High Performance File System"*, white paper, November 2002, http://www.lustre.org/docs/whitepaper.pdf, last visited March 2007, pp. 1 – 13;

[5] Doyle, A. T., Nicholson, C., *"Grid data management:: simulations of LCG 2008"*, CHEP'06, (2006), 1 – 8;

[6] Flickr™ Gamma a Yahoo! company: http://www.flickr.com/, last visited in March 2007;

[7] Foster, I., Kesselman, C., *"The Grid: Blueprint for a New Computing Infrastructure"*, second edition, Morgan Kaufmann Publishers Inc., 2004;

[8] Ghemawat, S., Gobioff, H., Leung, S., *"The Google File System"*, SOSP'03, (2003), pp. 29 – 43;

[9] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., West, M. J., *"Scale and Performance in a Distributed File System"*, ACM Transactions on Communication Systems, vol. 6, nr 1, (1998), pp. 51 – 81;

[10] Jones, T., Koniges, A., Kim Yates, R., *"Performance of the IBM General Parallel File System"*, IPDPS'00, (2000), pp. 673 – 681;

[11] Kangasharju, J., Roberts, J., Ross, K., *"Object replication strategies in content distribution networks"*, Computer Communications 25 (4), (2002), pp. 376 – 383;

[12] Karlsson, M., Karamanolis, C., Mahalingam, M., *"A Framework for Evaluating Replica Placement Algorithms"*, Technical Report HPL-2002, HP Laboratories, (2002), pp. 1 – 12;

[13] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B., *"OceanStore: An Architecture for Global-Scale Persistent Storage"*, ASPLOS'00, (2000), pp. 1 – 12;

[14] Liu, J., Xu, J., *"Proxy caching for media streaming over the internet"*, IEEE Communications Magazine, vol. 42, no. 8, (2004), pp. 88 – 94;

[15] Mummert, L. B., Ebling, M. R., Satyanarayanan, M., *"Exploiting Weak Connectivity for Mobile File Access"*, SIGOPS'95, (1995), pp. 143 – 155;

[16] OptorSim Release 2.1, October 2006, http://sourceforge.net/projects/optorsim;

[17] Qiu, L., Padmanabhan, V. N., Voelker, G. M., *"On the Placement of Web Server Replicas"*, IEEE Infocom, (2001), pp. 1 – 10;

[18] Quetier, B., Cappello, F., "A survey of Grid research tools: simulators, emulators and real life platforms", IMACS 2005, (2005), pp. 1 – 8;

[19] Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., Kubiatowicz, J., *"Pond: the OceanStore Prototype"*, FAST'03, (2003), pp. 1 – 14;

[20] Saito, Y., Karamanolis, C., Karlsson, M., Mahalingam, M., *"Taming aggressive replication in the Pangaea wide-area file system"*, OSDI'02, (2002), pp. 15 – 30;

[21] Wikipedia: http://en.wikipedia.org/wiki/Inclusion_exclusion, last visited in March 2007;

[22] YouTube™ Broadcast Yourself: http://www.youtube.com/, last visited in March 2007;

PROCEEDINGS OF
THE 2007 INTERNATIONAL CONFERENCE ON
PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND
APPLICATIONS

PDPTA 2007

Volume I

Editor

Hamid R. Arabnia

Associate Editors

Sanjay Ahuja, Jesus Carretero
Ping-Tsai Chung, Jose D. Garcia
Kazuki Joe, Mario Nakamori, Chung Ng
Hiroaki Nishikawa, Zornitza Prodanoff
Christian Rehn, Ashu M. G. Solo

WORLDCOMP'07
June 25-28, 2007
Las Vegas Nevada, USA
www.world-academy-of-science.org

ISBN: 1-60132-020-5, 1-60132-021-3 (1-60132-022-1)
Printed in the United States of America