Optimizing Replica Placement in Peer-Assisted Cloud Stores

Ahmed Ali-Eldin
Cloud Research Group
Umeå University
Umeå, Sweden
Ahmed.Ali-Eldin@cs.umu.se

Sameh El-Ansary
P2P Networking Group,Center of Informatics Science
Nile University
Cairo, Egypt
sansary@nileuniversity.edu.eg

***Abstract*—Peer-assisted cloud storage systems use the unutilized resources of the clients subscribed to a storage cloud to offload the servers of the cloud. The provider distributes data replicas on the clients instead of replicating on the local infrastructure. These replicas allow the provider to provide a highly available, reliable and cheap service at a reduced cost. In this work we introduce NileStore, a protocol for replication management in peer-assisted cloud storage. The protocol converts the replica placement problem into a linear task assignment problem. We design five utility functions to optimize placement taking into account the bandwidth, free storage and the size of data in need of replication on each peer. The problem is solved using a suboptimal greedy optimization algorithm. We show our simulation results using the different utilities under realistic network conditions. Our results show that using our approach offloads the cloud servers by about 90% compared to a random placement algorithm while consuming 98.5% less resources compared to a normal storage cloud.**

***Keywords*-Clouds; Peer to peer computing; Data storage systems;**

I. Introduction

Data storage and backup is a tedious computer problem. In order to protect backup data, the system must distribute replicas of the data across multiple sites. The backup process should be autonomous with minimum user intervention. Any lost replica on any site should be recreated autonomously. Building such a distributed backup system comes at a very high cost in terms of bandwidth, hardware and cooling.

P2P storage systems were proposed as a solution to the high cost problem [1]. A user of the system contributes part of his bandwidth and storage for replicating the data of other peers in the system. In return, the user utilizes resources from other dispersed peers around the world for replicating his data. P2P backup and storage systems solve some problems of traditional backup systems like the replica distribution but they introduce some different problems e.g. a peer on the network expects to have his files available and durable. Availability is the ability of a peer to always retrieve his data from the network at any time. Durability is the ability of a peer to retrieve his data in the future but not necessarily at a certain instance. It was suggested later that a P2P storage network is infeasible [2].

A recent solution for backup is to use cloud storage systems [3]. Cloud storage systems store data in virtual disks owned by a service provider instead of dedicated servers owned by the data owner relieving the owner from managing the stored data. The provider assigns resources for a user according to the user's *current* resource requirements. Using a cloud storage service has many advantages for the service users [3]. On the other hand, there are many challenges facing cloud storage systems, of which cost is the most significant [4]. The resources of the provider must scale with the number of subscribers. The bandwidth cost and the datacenter cooling costs [5] constitute major portions of the total costs.

Another approach that has been studied recently was to combine the previous two approaches to build a peer-assisted cloud storage system [4]. In peer-assisted cloud storage, peers in the network contribute part of their network and storage resources to offload the provider's servers. The service provider distributes replicas of the data of the subscribers through the network reducing the infrastructure and the cooling costs. Whenever a peer needs to download his data, the data is fetched from the network. If no copy of the data is available on the network, the data is fetched from the servers of the provider. This way the service provider provides a cheap and reliable service at a fraction of the cost. In this work, we focus on the replica placement problem in peer-assisted clouds which addresses the problem of where to place the replicas created by the system to maintain high levels of durability and availability. We extend on our previous work [6] and design five different utility functions to be used for optimizing the replication process. We show the advantages of our approach and discuss the different limitations.

This paper is organized as follows; in Section II, we describe our proposed system. Section III explains the design of the profit functions. Section IV shows the results of our simulations and the analysis of these results. In Section V, we give a brief overview of the literature. We conclude in section VI.

II. System Design

A. An economical framework for Replica placement

The problem of replica placement in peer-assisted cloud storage and its relation to the economic problem was discussed previously [6]. Replica placement is similar to the economic problem. There are scarce resources (bandwidth and storage) that can be allocated in many ways (different alternatives) between the different peers. We need to find an allocation that maximizes the utility of the system. Replica management economies are systems where the peer acquires real or virtual money for hosting replicas of other peers [7]. The peers use this money to buy resources for their own replicas from other peers. Similarly, we consider the problem of replica placement in peer-assisted storage clouds as an economical problem and we solve it using a mixture of two types of auctioning; first-price sealed-bid auctioning and double auctioning [8]. Our system design is robust to economical problems that might arise such as inflation.

1) Auctioning using task assignment: We define a data block to be a generic unit of data of constant size. Data blocks are the units used for trading between different peers. In a peer-assisted cloud storage, a peer replicates data blocks on the resources of several peers. The placement of these replicas affects both availability and durability. The goal is to utilize the contributed resources optimally. Optimality criteria might change between different deployment scenarios.

The framework we propose transfers the replica placement problem to a task assignment optimization problem. In task assignment problems, we have a group of workers and a group of tasks. A worker can be assigned to a task with a profit x. The number of tasks and workers do not have to be equal. The problem is solved optimally by an assignment that obtains the maximum total profit.

We view replica placement as a task assignment problem where we have a group of peers who need to replicate their data and a group of peers who contribute their unutilized resources. An auction is held between the different peers to assign each replica source to a replica host while maximizing the total profit of the assignment. A peer can be a source (bidder) and a host (seller) or both. A profit function is calculated between each source and host based on the bids sent.

B. Notations and Definitions

Let N be the number of peers available in the network. Let $P = \{p_1, p_2, ...p_N\}$ be the set of online peers in the system. There are K distinct data blocks hosted by the peers in the system. Each block is replicated in order to ensure its availability and durability. We define the set of distinct data blocks $B^{global} = \{b_1, b_2, ...b_K\}$ such that $b_i \neq b_j \ \forall \ i \neq j$. Every peer p_i has a group of data blocks that he hosts. We define the set of blocks that a peer p_i hosts to be $B_i = \{b_{ik} | b_{ik} \in B^{global} \ \forall \ b_{ik} \ hosted \ by \ p_i\}$. In other words, $B_i \subset B^{global}$. The replication level of a data block $R(b_k)$ is the number of replicas of the data block in the system. We want every distinct block b_k to be replicated r times on the available peers. We define the set of hosts or owners of the replicas of a block b_k to be $O(b_k) = \{p_i | b_k \in B_i \ \forall p_i \in P\}$. The system replicates a data block b_{ik} if $R(b_{ik}) < r$ only.

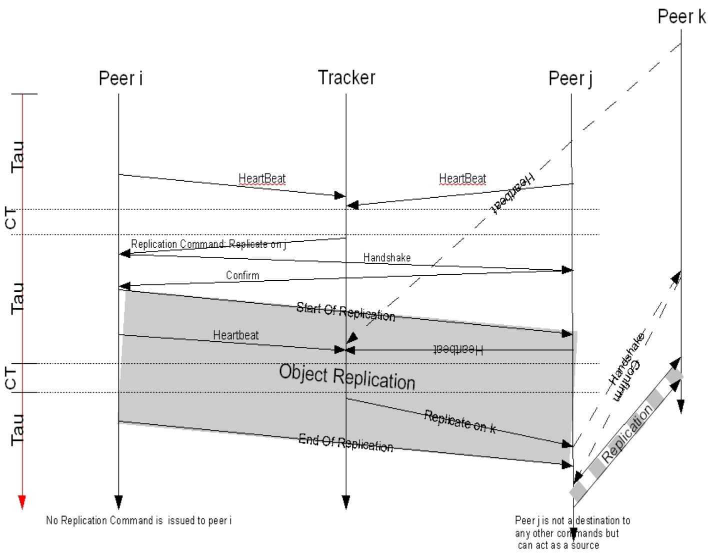

Figure 1. Long Replication commands that takes more than τ

We define the set of under-replicated data block on a peer p_i to be $\hat{B}_i = \{b_{ik} | R(b_{ik}) < r \ \forall \ b_{ik} \in B_i\}$. We define the set of unique data blocks between p_i and p_j to be $B_{ij}^{transf} = \{b_{ik} | b_{ik} \in \hat{B}_i \ and \ b_{ik} \notin B_j\}$.

Every peer p_i has an upload bandwidth u_i, a download bandwidth d_i and free space F_i. We define the unhealthiness of a peer p_i to be the average number of replications that its blocks need. The higher the unhealthiness of a peer, the more the risk the files he hosts can be lost. Thus,

$$H_{un}(p_i) = \frac{\sum_{\forall b_{ik} \in \hat{B}_i} r - R(b_{ik})}{|\hat{B}_i|}. \tag{1}$$

C. Design Goals

There are two main agents to design; the storage tracker who acts as an auctioneer and optimization decision maker and the peer agent who acts as both a source and host. We wanted our design to incorporate the following features and capabilities:

1) Bidding is done on the global scale. This is one of the main differences between our design and what was proposed by Geels et al. [7]. Using global optimization does not result in the best selfish choice, but gives the best overall system performance.
2) NileStore can be deployed by a storage service provider or for internal use within an organization. If the system is deployed by a service provider, fairness must be insured. The quality of service a peer gets is proportional to his contribution to the network. While

in an organization, a peer in can consume resources for storing data while contributing nothing.

3) The system transparently reintegrates the peers rejoining the network after a period of temporary failure. Peer reintegration saves considerable bandwidth. Eventually, the system comes to a stable state and temporary failures are masked[9].
4) Data blocks of the same owner are spatially replicated on the same machines if possible. This is one innovation in our system which brings down the management and communication overheads to be proportional to the number of peers rather than the number of data block. To the best of our knowledge, this approach is novel.
5) If the same data block is owned by two different peers initially, the system should be able to avoid making extra replicas.

D. The peers

Algorithm 1: Peer Sending Protocol

```
Report: Peers report their status through heartbeats periodically.
Listen: Peers listen for messages.
Receive
if message==Replication Command then
  if $UploadFlag$==FALSE then
    Handshake: Replica owner sends handshake messages to the potential host containing the list of blocks the tracker ordered for replication on the destination host.
    Receive Confirmation
    Raise $UploadFlag$ TRUE
    Replicate: replicate blocks disregarding any block that can not be accepted by the destination
    if $\tau$ passed then
      Report: Peer reports status through heartbeats
    end if
    End: Owner raises an $EndOfReplication$ flag in the message of the last replicated block
    Clear Flag: Set the $UploadFlag$ to FALSE
  else
    Ignore: Ignore the replication
  end if
end if
```

When acting as a replica source, the peer uses Protocol 1 while when acting as a destination the peer uses Protocol 2. A peer p_i sends a heartbeat to the tracker every τ minutes. The heartbeats contains five fields; identifier of the peer p_i, list of hashes of the owned and hosted blocks on p_i , the free space F_i to be contributed, the upload bandwidth u_i, the download bandwidth d_i and $UploadFlag$ and $DownLoadFlag$.

The tracker replies with a replication command indicating the address of the replica host (seller) p_j. The replication command has the three fields; p_i's identifier, p_j's identifier, list of hashes of the blocks on p_i to be replicated and the number of missing replicas to threshold of each block. This message can be digitally signed if needed.

To reduce the load on the tracker's network, this command in sent only to the block source p_i. p_i forwards the message to p_j. p_j confirms that he is able to host the blocks advertised by p_i. If any of the peers becomes unavailable, the replication command is discarded. p_i transfers the blocks that p_j agreed upon starting with b_{ik} with minimal $R(b_{ik})$. p_j checks the hashes of the blocks that p_i is sending.

In the next round, peers sending heartbeats include the list of hashes of hosted blocks with the owned blocks allowing the tracker to accurately track the replica count of each block. Peers replicate both their hosted and owned blocks helping the system converge faster to a steady state. Thus a peer consumes his bandwidth to help replicate other peers' blocks in exchange of the service he gets from all the peers hosting his blocks. This design eliminates the economic problem of inflation.

Algorithm 2: Peer Receiving Protocol

```
Listen: Peers listen for messages.
Receive
if message==handshake then
  if $DownloadFlag$==False then
    Confirm: Confirm the suggested list. The message contains also a list of blocks that can not be accepted from the suggested list.
    Receive Replicas and Raise flag: when the replicas start to come, raise $DownloadFlag$ to TRUE and store the blocks.
    if $\tau$ passed then
      Report: Peer reports status through heartbeats
    end if
    End: Receiver receives an $EndOfReplication$ flag in the message of the last replicated block.
    Clear Flag: $DownloadFlag$ set to FALSE
    Report: Destination reports his state after the replication operation to update the server's snapshot
  else
    Ignore: Ignore the handshake
  end if
end if
```

When peers send their heartbeats, a peer involved in an upload process raises the $UploadFlag$ to True while a peer involved in a download process raises the $DownLoadFlag$ to True. A peer engaged in an upload operation only is considered by the tracker as a worker (a replica destination) but no replication commands are issued to him as shown in Figure 1.

E. The Tracker

The tracker runs the *tracker protocol* shown in protocol 3. The tracker is managed by the cloud service provider and is responsible for calculating the assignment decisions. The tracker has three main jobs:

1) Tracking B^{Global} in the system and determining $\hat{B}_i \; \forall \; i \in P$ for replication.
2) Converting the peer placement problem into a task assignment problem.
3) Solving the task assignment optimization problem using either an optimal or a suboptimal algorithm. The resulting assignment represents peer coupling commands. The tracker sends to each peer the address of the host to which he is assigned.

Algorithm 3: Tracker Protocol

```
while 1 do
  Collect: tracker collects Heartbeats from the peers in the $peersTable$ and $blocksTable$.
  if $\tau$ passed then
    Collect delayed: Tracker accepts any delayed heartbeats in a buffer for concurrency reasons.
    Transform: Tracker transforms the replica placement problem into Task assignment optimization.
    Profit function: for each table entry calculate the utility of assignment
    Compute: Tracker Solves the Task Assignment problem using optimal or suboptimal algorithms
    Issue Commands: Tracker sends the results of the assignment as replication commands
  end if
end while
```

The tracker receives the heartbeats from every peer as shown in Figure 1. A peer who did not send a heartbeat for some time is considered unavailable. The heartbeats are stored in the $peersTable$ as shown in Table I. The heartbeats are used to generate the $SourcesTable$ shown in table II which tracks the peers with under-replicated blocks, the set of blocks that need replication $\hat{B}$ and H_{un}. Since the heartbeats contain a list of hashes of the data blocks, the tracker can accurately track similar blocks. The tracker waits for a period τ in order to collect as many heartbeats as possible before starting a computation round. During the computation round, the tracker constructs a task assignment problem from the data collected using one of the utility functions explained later. Any heartbeats received during the computation round are cached for the next computation round. The task assignment problem is solved using the sub-optimal algorithm explained in [6]. Each peer is sent a message indicating the address of the host who will store the peer's replicas and the objects to be replicated. This cycle is repeated.

Table I
THE STRUCTURE OF THE PEERS TABLE IN NILESTORE

Peer Name	blocks list	free space	Upload	Download
i	B_i	F_i	u_i	d_i

Table II
STRUCTURE OF THE SOURCES TABLE IN NILESTORE

Peer Name	$\hat{B}$	$\Sigma(1 - R(b_k)) \; \forall \; b_k \in \hat{B}$	H_{un}
p_i	$\hat{B}_i$	Value	$H_{un}(p_i)$

III. THE DIFFERENT UTILITY FUNCTIONS

The utility functions consist of three multiplied terms. The terms taken into consideration are storage feasibility S_i, transfer or bandwidth feasibility T_i and H_{un} indicating the rareness of the blocks on a peer. The utility is designed to be flexible and more parameters can be added e.g. a term for the ping time latency.

A. Profit function I

Algorithm 4: Profit function I

```
for $p_i \in SourcesTable$ do
  for $p_j \in peersTable$ do
    Calculate Feasibility of Storage $S_{ij} = \frac{\min(|\hat{B}_i|, F_j)}{\max(|\hat{B}_i|, F_j)}$
    Calculate Feasibility of Bandwidth $T_{ij} = \frac{\min(u_i, d_j)}{\max(u_i, d_j)}$
    Profit function $U(p_i, p_j) = S_{ij} * T_{ij} * H_{un}(p_i)$
  end for
end for
```

The Feasibility of storage S_{ij}: responsible for capturing the feasibility of storing the data of p_i of size $|\hat{B}_i|$ on the free space F_j of p_j as shown in Algorithm 4. When coupling two peers, we try to minimize the fragmentation of data and free space by keeping the blocks owned by a peer spatially on the same machines. If p_j has free space F_j, the assignment finds the best fit allocation source p_i having data to fill F_j. For example, suppose four peers p_A, p_B, p_C and p_D join the system. Both p_A and p_B have blocks to be replicated of sizes 40 MB and 500 MB respectively. p_B joins the network at a later time than peer p_A . Peers p_C and p_D are in the system and contribute 50 MB and 530 MB of free space respectively. If p_A is coupled with p_D, p_D is left with only 490 Mb of free space. When p_B joins the network, he has to fragment his data on two peers in the network. Using our definition for S_{ij}, $S_{AC} = 0.8$, $S_{AD} \approx 0.075$, $S_{BC} = 0.1$ and $S_{BD} \approx 0.94$. The engine increases the profit by choosing $S_{AC} = 0.8$ and $S_{BD} \approx 0.94$.

If a peer contributes at least r times the data size he wants to replicate, we guarantee fairness i.e. a peer hosts data equal to the storage he uses. If the initial data size is large, he has to pay the price by contributing large free space to the system. If the peer chooses to increase his storage contribution, he is able to host more blocks and will eventually get a higher utility and better assignment. Initially, the system favors peers whose initial block size is large. This way we prevent economical inflation.

One disadvantage of calculating S_{ij} this way is that it equalizes between a peer p_q having free space enough to host the source's data $|\hat{B}_i|$ and a peer p_w that can not host the source's data but the F_w to $|\hat{B}_i|$ ratio equals to the ratio $|\hat{B}_i|$ to F_q.

The Feasibility of transfer T_{ij}: This term adds the bandwidth consideration to the utility calculations. We want to couple p_i with upload bandwidth u_i with a peer p_j with download bandwidth d_j such that we try to completely utilize the bandwidth of p_i and p_j in order to reduce the total transfer time of all the peers. If all peers are coupled this way, we guarantee that every peer will finish the replication process as fast as he can.

Fairness can be imposed by enforcing a restriction on the ratio between the upload bandwidth of p_i and the download bandwidth he has to contribute to the system. A peer who wants to upload his blocks faster by increasing his upload bandwidth has to increase his contributed download bandwidth.

The $H_{un}(p_i)$ parameter indicates the rareness of the blocks hosted by a peer and thus gives the peer a weight based on that value. Each of the above parameters is calculated for any peer as shown in Algorithm 4 versus any potential host and fed as input to the assignment engine.

B. *Profit function II*

Algorithm 5: Profit function II

```
for $p_i \in SourcesTable$ do
  for $p_j \in peersTable$ do
    if $|\hat{B}_i| <= F_j$ then
      Compute $S_{ij} = \frac{|\hat{B}_i| * Constant}{F_j}$
    else
      Compute $S_{ij} = \frac{F_j}{|\hat{B}_i|}$
    end if
    Compute $T_{ij} = \frac{\min(u_i, d_j)}{\max(u_i, d_j)}$
    Profit function $U(p_i, p_j) = S_{ij} * T_{ij} * H_{un}(p_i)$
  end for
end for
```

This profit function was designed to overcome the problem of non-linearity introduced by Profit function I. When calculating S_{ij}, the feasibility of storage for all peers having free storage equal to or greater than that required by the source is increased by multiplying the utility by a constant as shown in Algorithm 5. The highest utility is given to peers with free space equal to the required storage by the source as the constant is multiplied by one in this case and by a fraction otherwise. We do not change T_{ij} the same way as such a change will have a negative effect on the assignment decisions that results in a waste of resources [10].

C. *Profit function III*

This utility function is a simple extension to Profit function I explained previously but at a higher computational cost. Instead of calculating the utilities based on B_i, we calculate the utilities based on B_{ij}^{transf}, the blocks that are not common between the two peers . This way we ensure that the utility between two peers does not take into account any common data. This comes at the cost of calculating the difference between the two peers' blocks. The rest of the algorithm remains the same as in Algorithm 4 .

D. *Profit function IV*

Similar to Algorithm 5 but we modified the function to only include the blocks that are not common between the two peers during the utility calculation B_{ij}^{transf}. This also comes at an increased computational cost.

E. *Profit function V*

The last utility calculating function is shown in Algorithm 6. First, we find B_{ij}^{transf}. We then calculate the maximum number of blocks that can be transfered between any two peers p_i and p_j. The system calculates the maximum transferable amount of data between the two peers in the time period τ. This value is the the minimum of the upload bandwidth of the source peer and the download bandwidth of the host multiplied by τ. Next, the system calculates the maximum data storage units that can be transfered. This value is the minimum between the size of data that the source needs to replicate and the destination free storage. The minimum of these two values represent the maximum number of blocks that can be transfered in τ and is donated by M_{ij}. M_{ij} is then multiplied by $H_{un(ij)}^{trans}(p_i, p_j)$ which is defined as follows :

$$H_{un}^{trans}(p_i, p_j) = \frac{\sum_{\forall b_{ik} \in B_{ij}^{trans}} r - R(b_{ik})}{|B_{ij}^{trans}|} \qquad (2)$$

$H_{un}^{trans}(p_i, p_j)$ represent the total unhealthiness of the blocks that can be transfered between p_i and p_j. As this value increase, the transfer between the two peers results in an increase in the total system health.

Algorithm 6: Profit function V

```
for $p_i \in SourcesTable$ do
  for $p_j \in peersTable$ do
    Find $B_{ij}^{transf}$
    Compute
    $M_{ij} = \min(\min(B_{ij}^{transf}, F_i), \min(u_i, d_j) * \tau)$
    Compute $H_{un}^{trans}(p_i, p_j) = \frac{\sum_{\forall b_{ik} \in B_{ij}^{trans}} r - R(b_{ik})}{|B_{ij}^{trans}|}$
    Compute $U(p_i, p_j) = M_{ij} * H_{un}^{trans}(p_i, p_j)$
  end for
end for
```

IV. Simulation and Results

We built a discrete event simulator to simulate a peer-assisted cloud storage network. The storage network has on average 1000 peers. Initially all the peers in the simulated network have unique blocks. The simulation's time step is one minute.

We used a group of P2P workload studies to come to a workload model that fits our problem. We are not aware that there are any studies that measures all the parameters we are using in NileStore from a single P2P network. Peers' upload and download rates were obtained from [11]. Unfortunately, we have not come across any study that measures the average amount of free storage a peer contributes and the average amount of replication a peer consumes in a P2P storage network or a peer-assisted cloud storage service. This led us to initially generate for each peer a number of unique blocks of sizes between 10 MB and 1 GB and a maximum contributed free storage between r and $2r$ the size of his data.

There are a number of P2P network studies that try to quantify the peer Join/Leave rates. We mixed a group of these studies in order to obtain rates for temporary failures, permanent failures and join rates. According to [12], peers Join and Leave rates exhibit a negative binomial distribution in the Kademlia P2P network (KAD) with $n = 3$ and $p = 0.127$. These are the aggregate rates, that is the Leave rate is the sum of the temporary and permanent failures, while the join rate is the sum of the new peers joining and the returning peers that have been temporarily unavailable. There is no way to differentiate between permanent failures and temporary failures though due to the problem of KAD aliasing [12] so we used another study [9] in order to get the ratio of temporary failures to permanent failures with time. We assume that the average number of new peers joining is equal to the average number of peers permanently failing . We conducted our experiments to evaluate NileStore versus the random placement approach. Random placement is the only approach available in the literature for replica placement in peer-assisted clouds. In this approach, the cloud provider couples the peers randomly.

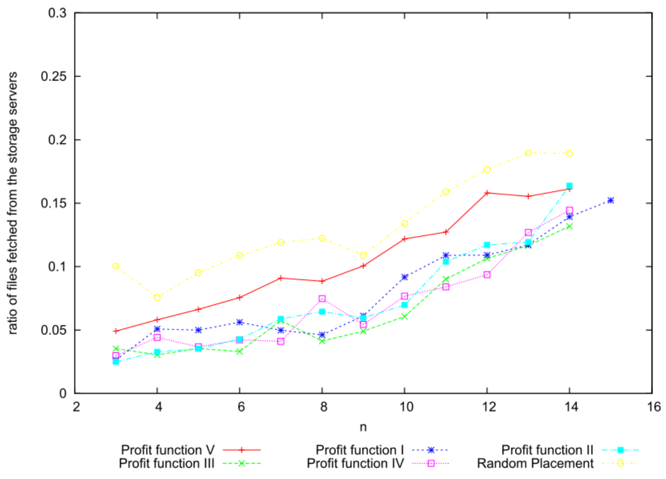

Figure 2. Comparing the different Utilities proposed with random placement when the churn is changed

In the first experiment we try to quantify the effect of increasing the failure rates on the performance of the system using the different utility functions. As the failure rate follows a negative binomial distribution, we changed the n parameter of the negative binomial distribution increasing the failure rate to orders of magnitude of the failure rate seen in the Planetlab network [10]. We simulated a period of 20000 minutes with $\tau = 2$ minutes . Figure 2 shows the percentage of files that needed fetching from the cloud servers after 20000 minutes. The figure shows that NileStore using any of the utility functions proposed shows a significant improvement between 50% and 75% compared to a random placement approach at low churn rate by offloading the cloud servers by 97.5%. As the churn rate increases, all the utility functions converge to the random placement approach because the system dynamics are two high that most of the replication commands are not fullfilled.

Figure 3 shows the effect of changing the number of replicas on the percentage of data fetched from the provider's servers. Our results conform with previous results that a replication degree beyond 3 is of minimal advantage [13].

We next quantified the effect of increasing τ on the system performance as shown in Figure 4. The best performance is achieved when τ is set to 5, 6 or 7 minutes with a performance gain of more than 90% compared to the random placement approach. The performance starts to decrease when τ is increased more. If τ is small, many peers will not have time to finish the previous replication commands. The assignment algorithm will have less assignment options and ends up coupling peers that have low utilities as there is a lack of other options. If τ is set to be large, NileStore will react to failures slowly risking the loss of the objects that need replication. A balanced τ allows the system to react in time to permanent failures while at the same time allowing

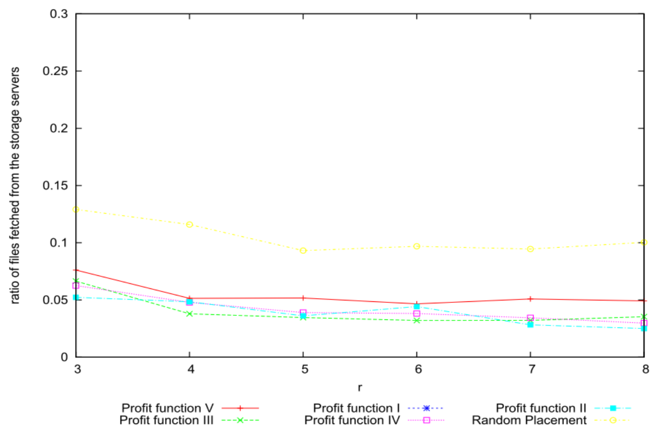

Figure 3. Effect of changing the replication degree on the percentage of data fetched from the servers

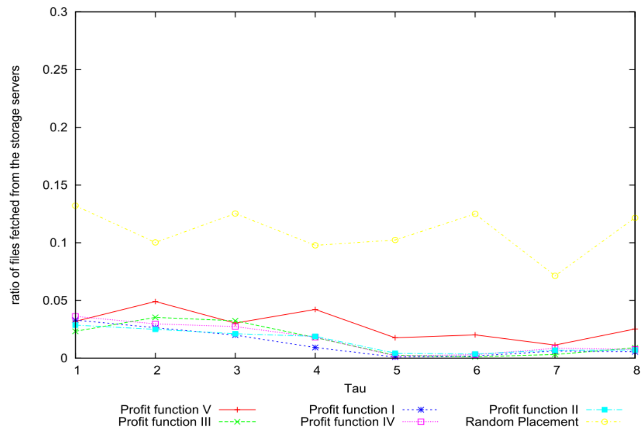

Figure 4. Effect of changing τ on the percentage of data fetched from the servers

as many peers as possible to finish the replication process between two computation rounds.

The worst performance among the different utilities is the one shown by Profit Function V. There is an inherent problem between profit function V and the algorithm we use for solving the task assignment problem as we have discovered. Profit function V in many cases gives equal utility for an assignment between a peer p_i and several other peers wasting valuable system resources. For illustration, assume a peer p_i has $u_i = 1$ and $|B_i| = 50$ with 9 unique blocks , also assume that peer p_j contributes to the system $d_j = 1\ MB/min$ and $F_j = 50\ MB$, peer p_k contributes to the system $d_k = 8\ MB/min$ and $F_k = 60\ MB$ and peer p_l contributes to the system $d_l = 20\ MB/min$ and $F_l = 1000\ MB$. Let $\tau = 9\ minutes$ and assume that all the 50 objects on p_i are not replicated on the system and that $r = 9$. Calculating the costs for each of these functions using profit function V, we find that all of these peers have the same cost because the upload rate of the peer is quite low. Thus, the upload bandwidth of peer p_i dominates the calculation of $M_{i*} = 1 * 9 = 9\ MB$ while $U(p_i, p_*) = 9 * 9 = 81$. This means that all the three peers have the same cost giving all three peers equal probability of pairing with peer p_i. In reality peer p_i should be coupled with peer p_j in order to minimize the resources wasted.

our simulations show that NileStore requires the following computation times: 7 seconds on average for profit function I and between 18 and 20 seconds for the other profit functions compared to on average 0.7 seconds for a random placement approach. Since the performance gain of the first 4 utility functions is almost equal, using profit function I is natural as it is the simplest profit function with the lowest computational cost. Although the assignment time is longer for NileStore, it is still much shorter than τ making NileStore a feasible solution to the replica placement problem.

V. Related work

Peer-assisted cloud storage systems has recently received some attention. In [14], FS2You, a peer assisted cloud storage system deployed in china is introduced. The authors describe their system design and show some measurements taken from their system which has more than one million subscribers. FS2you does not consider more advanced replica placement algorithms. Our work can be considered as an extension to their system. Toka *et al.*[4], study the benefits of a peer assisted approach to online backup. They prove that using a hybrid approach provides performance comparable to that of a centralized cloud scenario at a fraction of the cost. They compare a randomized placement for replicas versus an approach where peers are clustered depending on their online behavior to impose fairness. Our system takes into account the contributed storage, bandwidth and the scarcity of the data when doing data placement while satisfying the needs of all the customers. Amazingstore [15] is a peer-assisted storage system that is still in development phase. In the initial design a random placement approach is used. More advanced replica placement strategy and algorithms are kept as future work. Our work can be considered as an extension to AmazingStore.

There is a huge body of work on P2P storage systems. Oceanstore [16], Farsite [13] and Tahoe [17] are examples of P2P storage systems. In [2], the authors argue that large-scale P2P storage is limited by P2P dynamics and cross-system bandwidth. They conclude that when redundancy, data scale, and dynamics are all high, the needed cross-system bandwidth is unreasonable. A similar conclusion was presented in a study that sums up the lessons learned by the designers of Farsite [18]. Peer-assisted cloud storage systems are more feasible as the storage nodes in the cloud are less dynamic compared to the P2P nodes.

Volley [19] is a system used for choosing a datacenter for data placement based on the distance between the center and the weighted centroid of the data users in order to reduce the access latency. Volley was designed for data placement between datacenters and is not suitable for use with a P2P network with dynamic membership.

VI. Conclusion and Future work

We introduced NileStore, a Peer-assisted cloud storage protocol that offloads the resources of the cloud storage servers using the unused resources of the peers subscribed to the storage service. The unused free storage and bandwidth of the peers is contributed to the system to replicate the data owned by each peer. The replica placement problem is converted into a task assignment optimization problem. Replication of a data block takes place until the number of replicas of the object reaches a certain threshold. A copy of each data block is stored on the provider's servers to ensure durability and availability. A peer always tries to retrieve his data from the peers hosting his data on the network before contacting the storage servers. We designed five different utility functions for the optimization problem.

The main metric we use for measuring the performance of the system is the ratio of data that needs fetching from the servers to the data stored on the network. This ratio using our utility functions is around 0.9 less compared to a random placement approach and around 0.99 less compared to a normal storage cloud and will thus result in huge savings to the storage cloud provider. In the future we plan to distribute the allocation process, use more robust assignment engine, and consider security issues such as the trust between the different peers and the cloud provider.

References

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[2] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *Proceedings of the 9th conference on Hot Topics in Operating Systems-Volume 9*. USENIX Association, 2003, p. 1.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[4] L. Toka, M. Dell'Amico, and P. Michiardi, "Online Data Backup: A Peer-Assisted Approach," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE, 2010, pp. 1–10.

[5] H. Amur, J. Cipar, V. Gupta, G. Ganger, M. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 217–228.

[6] A. Ali-Eldin and S. El-Ansary, "Replica placement in peer-assisted clouds: An economic approach," in *Distributed Applications and Interoperable Systems*. Springer, 2011, pp. 208–213.

[7] D. Geels and J. Kubiatowicz, "Replica management should be a game," in *In Proc. of the 10th European SIGOPS Workshop*. ACM, 2002.

[8] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and computation: practice and experience*, vol. 14, no. 13-15, pp. 1507–1542, 2002.

[9] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 4–4.

[10] A. Ali-Eldin, "Replication in nilestore: A peer-to-peer assisted cloud storage system," Master's thesis, Nile University, 2010.

[11] D. K. Correa, "Assessing Broadband in America: OECD and ITIF Broadband Rankings," *SSRN eLibrary*, 2007.

[12] M. Steiner, T. En-Najjary, and E. W. Biersack, "Analyzing peer behavior in kad," Institut Eurecom, October 2007, Tech. Rep.

[13] W. J. Bolosky, J. R. Douceur, and J. Howell, "The farsite project: a retrospective," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 2, pp. 17–26, 2007.

[14] Y. Sun, F. Liu, B. Li, B. Li, and X. Zhang, "Fs2you: Peer-assisted semi-persistent online storage at a large scale," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 873–881.

[15] Z. Yang, B. Zhao, Y. Xing, S. Ding, F. Xiao, and Y. Dai, "AmazingStore: available, low-cost online storage service using cloudlets," in *Proceedings of the 9th international conference on Peer-to-peer systems*. USENIX Association, 2010, p. 2.

[16] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5, pp. 190–201, 2000.

[17] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*. ACM, 2008, pp. 21–26.

[18] W. Bolosky, J. Douceur, and J. Howell, "The farsite project: a retrospective," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 2, pp. 17–26, 2007.

[19] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, pp. 2–2.