Cloud Storage for Mobile Users Using Pre-Positioned Storage Facilities

Benjamin Baron, Prométhée Spathis, Marcelo Dias de Amorim, Mostafa Ammar

Cloud Storage for Mobile Users Using Pre-Positioned Storage Facilities

Benjamin Baron[†], Prométhée Spathis[†], Marcelo Dias de Amorim[†], and Mostafa Ammar[*]
[†]UPMC Sorbonne Universités [*]Georgia Institute of Technology
{bbaron, spathis, amorim}@npa.lip6.fr, ammar@cc.gatech.edu

ABSTRACT

We propose a cloud-like file storage and sharing system designed for mobile users. Our system relies on a collection of strategically pre-positioned repositories within which files are replicated without relying on a conventional infrastructure-based network. Once stored in the first encountered repository, copies of the files are carried by the initial uploader or subsequent users and distributed among the other repositories. Having multiple copies available at different repositories thus increases the likelihood of finding the requested files in a timely fashion. Files can later be retrieved by other users at different locations. We are interested in processing user storage and retrieval requests before their deadlines expire. We design an algorithm to place the repositories such that they serve a maximum number of requests before their deadlines expire. We evaluate our system using mobility traces of San Francisco city buses. We show the impact of the number and placement of repositories on request success rate. We also show the benefits of mobility-leveraged file distribution.

Keywords

Vehicular cloud storage and sharing system, opportunistic networks, placement algorithm

1. INTRODUCTION

Cloud storage services have shown their great potential given the popularity of services such as Dropbox or Google Drive. Mobile users accessing such services through wireless networks including cellular 4G LTE or Wi-Fi are often faced with high cost, low bandwidth, and limited coverage.

In this paper, we design a cloud-like file storage and sharing system specifically targeting mobile users. The system deploys a collection of strategically located pre-positioned data storage facilities acting as file repositories. Mobile users upload files that they would like to archive or share, or retrieve files uploaded by them or by other users from this set of repositories as they meet them. We leverage the existing mobility of the initial uploaders and subsequent users to limit the dependency of our system on conventional infrastructure-based networks.

We consider techniques that use the mobility of either the initial file uploader or subsequent users to replicate files. The distinctive feature of our approach lies in the opportunistic use of mobile users as data shuttles between the file repositories. Copies of the files are transferred to users when in the vicinity of a repository and transported to other repositories along user routes. To increase the likelihood of replication, copies of the user files can be loaded on multiple users traveling among the repositories.

One of the main challenges in implementing our system is handling user requests to either store a file or retrieve one in a timely fashion. A request is delayed depending on how close the user is from the nearest repository. Each request comes with a deadline indicating when it expires. Past the deadline, the request is considered as failed. Our goal in the design of the system is to bound the request deadline to minimize the failure rate of requests.

Two aspects of the system can have a significant effect on the request failure rate. The first is the *degree of replication* for files in the repository. The second is the *number* and *placement* of file repositories in the mobile environment. Our work considers both of these challenges.

We propose an algorithm to place the repositories at strategic locations where they can capture a maximum number of user requests before they expire. To enable the distribution strategy of the user files across the repositories, the placement algorithm also takes into account the flows of users traveling between the repositories. The placement of the repositories then faces two different objectives. On the one hand, the repositories must be spaced far enough from one another to avoid starving user requests and limit the overlap of user requests they serve. On the other hand, the repositories must be close enough and well connected to one another by the movements of the users to create a connected network that enables the distribution of user files in all repositories. We instantiate our system using bus stops as potential locations for repositories and buses as mobile users. Buses also behave as data shuttles between repositories.

The repositories we use to build our system are somehow similar to throwboxes [10] and offloading spots [1]. Repositories also enable data transfers by combining the trajectories of multiple mobile users. The mobile users that we use to distribute the files across the repositories act as data

[*]This author's work was supported in part by NSF grants NETS 1409589 and NETS 1161879.

MULEs [9]. In our system, the repositories store files in a distributed manner and help share them with mobile users. Content storage and retrieval was also an issue studied in the context of Delay-Tolerant Networking (DTN) [4]. Examples include TierStore [3] which is a Network File System that relies on very low latency links that intermittently connect the repositories. This makes TierStore well suited in the context of DTNs. Ott and Pitkänen [8] proposed a system based on repositories to cache user content, thus decreasing the latency when retrieving content. However, both of these approaches do not study the impact of a placement of storage repositories that can guarantee the availability and distribution of files.

To summarize, our contributions are the following:

- **File distribution.** We leverage the users traveling between repositories to distribute files. With copies of the files available at multiple repositories, user requests are more likely to be successful.
- **Repository placement.** We design an algorithm that places a target number of repositories over a geographical area to satisfy a maximum number of user requests. The placement of the repositories depends on the mobility patterns of mobile nodes, the deadline for user requests, and the vehicle flows connecting the repositories together.

The rest of this paper is organized as follows. We describe the problem statement in Section 2. In Section 3, we detail the algorithm to place the repositories. Finally, we evaluate our cloud storage system with traces of San Francisco buses in Section 4 and conclude in Section 5.

2. PROBLEM STATEMENT

We consider a geographical area with n mobile users that move according to a known mobility pattern and m fixed repositories placed over the geographical area according to a placement strategy. Both are equipped with wireless capabilities and with local storage to store user files. We enable communications only between mobile users and repositories through wireless interfaces (*e.g.*, IEEE 802.11).

2.1 Repositories

The repositories are stationary and located at specific locations. They feature high availability, large storage and fast wireless transmission capabilities to handle large data rates in parallel with multiple mobile users. We assume a central controller is in charge of monitoring the files available at each repository. Whenever a repository receives a new file from a user, it notifies the controller of the availability of this file. Likewise, the mobile users can be notified of the availability of the files through the controller. We assume that the communications between the controller and the repositories and mobile users are handled by an inexpensive control channel (*e.g.*, a cellular connection).

2.2 User requests

There are two types of user requests: `put` requests to store user files and `get` requests to retrieve them. All requests have an associated deadline. Once a mobile user generates a request, the request becomes pending for the user until they encounter a repository. When in the transmission range of the repository, the user seamlessly transfers all of its pending requests to the repository.

`put` request. When a mobile user generates a `put` request, the user waits to be in the transmission range of the first repository it encounters. On an encounter with a repository, the user transfers the `put` request message along with the file. Once the file is received, the repository updates its local storage and notifies the availability of the file to the central controller. We define the two behaviors of mobile users when they have a `put` request to execute:

- "First" `put` policy: Mobile users immediately delete the request once they transferred it to the first repository they encounter or if its deadline expires, whichever occurs first.
- "All" `put` policy: Mobile users delete the request once the deadline associated with the request has expired. This allows the users to potentially replicate the files to all repositories they encounter from the moment the request is generated to the moment it expires.

`get` request. Users share the files they store on the repositories with other users. That is, any user can generate a `get` request to retrieve any available file. The user then transmits the `get` request to repositories it meets until the request is satisfied or until its deadline expires, whichever occurs first. In turn, the repository processes the request and transfers the requested file to the mobile user if the file is available in the repository's local storage. Otherwise, the repository notifies the mobile user that the requested file is not available. The mobile user will then try to retrieve the requested file from the next repository it will encounter. If the mobile user does not encounter any repository with a copy of the requested file before the request's deadline expires, the request fails.

2.3 File distribution with user mobility

We incorporate a file distribution strategy that exploits user mobility to move copy files among repositories. Increased file distribution will reduce `get` failure rates. To accomplish this, we introduce `sync` requests for the files that need distribution. Repositories generate `sync` requests once files are stored by their initial uploader, following `put` requests. A `sync` request results in transferring copies of a file from the repository to passing mobile users in charge of distributing these copies to repositories along their trajectories. A later encountered repository updates its local storage with the corresponding file and notifies the central controller of the new availability of the file. If the file is yet to be available in every repository, the new repository starts distribution the file it just received by generating corresponding `sync` requests. Otherwise, the central controller informs the repositories of the distribution completion.

The three types of requests `put`, `get`, and `sync` are shown in Figure 1. In this figure, the mobile nodes are buses that aggregate requests from the passengers riding the bus. The first mobile user on the left carries three requests: `get` for file A, `sync` for file B, and `put` for file C. The repository, which only has a local copy of file A, executes all the requests and adds files B and C to its local storage. When a subsequent mobile user encounters the repository, the repository generates `sync` requests and transfers them to the mobile node,

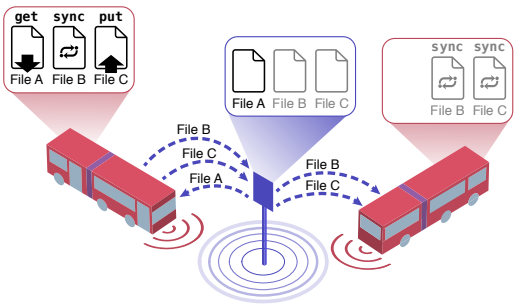


Figure 1: Requests and data exchanges between a repository and two mobile users when they encounter one another.

along with the copies of the files. The mobile node then carries the `sync` requests to the next repository it encounters.

2.4 Performance objectives

We consider the following metrics to evaluate the performance of our storage and sharing system:

- **Request success ratio.** We aim to maximize the successes and minimize the failures of user requests. A `put` or `get` request is successful if it completes before its deadline expires. The success ratio is the number of successful requests divided by the number of all requests issued during the time period of interest.

- **File distribution duration.** File distribution relies on the movements of the mobile nodes to distribute copies of files across the repositories. The distribution duration of a file is the time it takes for it to be available in all the repositories of the system. The repository placement algorithm takes this metric into account when allocating the location of the repositories to minimize the distribution duration of the files.

3. REPOSITORY PLACEMENT

We detail the repository placement algorithm that determines the locations of a target number of repositories for a given user request deadline. The goals of the placement algorithm are twofold: (*i*) determine locations such that the allocated repositories serve the maximum number of user requests before they expire and (*ii*) connect the repositories together by the mobile users' movements to create the file distribution network. The goals of the placement algorithm are represented in Figure 2 in the case of a bus transit system. The bottom layer shows the trajectories of the buses in the Financial District of San Francisco, with allocated bus stops. Each bus trajectory is represented by a color and a width indicating the frequency of buses. The middle layer shows the discretized demands generated by the bus passengers. This layer shows the demands that are allocated to the repositories placed at the bus stops. Finally, the top layer shows a logical graph where nodes correspond to allocated repositories and edges to the flows of buses running between two repositories. As in the bottom layer, the width of the edges corresponds to the frequencies of buses.

We derive the placement of the repositories from a set cover problem, in particular, the Maximal Covering Location Problem [2]. Given a set of candidate locations for the repositories, the placement problem consists of selecting the candidate locations that maximize the success ratio of the

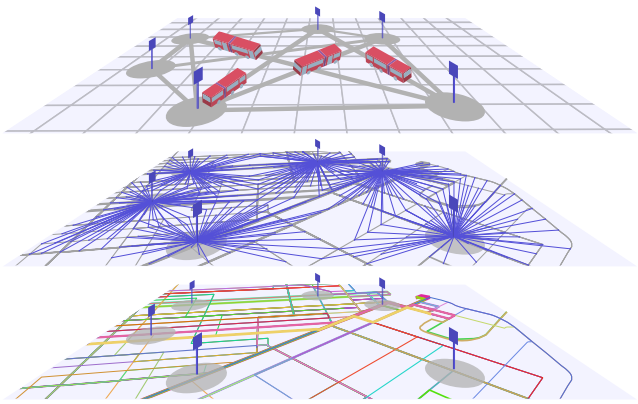


Figure 2: Different layers to represent the goals of the placement algorithm in the case of a bus transit system.

requests issued by the users. This problem was shown to be NP-Hard [7], so we adapt known heuristics to solve it. To this end, we consider the Greedy Adding with Substitution (GAS) heuristic [2] that determines the optimal locations for each iteration of the algorithm.

3.1 Candidate and demand locations

The placement algorithm takes both a set $\mathcal{C}$ of candidate locations where the repositories can be placed, as well as a set $\mathcal{D}$ of demand locations to be served by the allocated repositories. The candidate and demand locations refer to geographical places (typically circles and square cells) visited by the user trajectories. The algorithm outputs a set $\mathcal{A}$ of allocated candidate locations for the repositories.

Demands. Users can generate a request at any time. Each successive location visited by the trajectories of the users is a demand location with equal probability of being the starting location of a user request. To this end, we divide the geographical area into a grid of cells of a given size (*e.g.*, 100 m × 100 m). Each cell aggregates the discretized user trajectories that go through the cell. The greater the number of trajectory aggregates, the higher the probability for a cell to be the starting location of user requests.

Candidates. Candidate locations for the repositories correspond to relevant locations the mobile users will often visit for long periods of time, allowing the transfer of significant amounts of data. For instance, candidate locations may be road intersections, or bus stops (*e.g.*, in the case of a bus transit system). In the evaluation section, we use available data from bus transit systems and consider bus stops as candidates for the location allocation of the repositories.

To determine the locations of the repositories, we quantify the relations between two locations l_1 and l_2 (either a demand or candidate location) by computing the following spatial statistics:

- The mean visit frequency $f(l_1, l_2)$ of mobile users that visited location l_1, then location l_2.

- The median inter-visit duration $v(l_1, l_2)$ of two consecutive visits of mobile users at location l_1 that visited location l_2 later in during their trajectory.

- The median travel time $t(l_1, l_2)$ of the time it takes for a mobile user to travel from location l_1 to location l_2.

We then measure the weight $w(l_1, l_2)$ of a pair of locations (l_1, l_2) using the visit frequency $f(l_1, l_2)$ and the median inter-visit duration $v(l_1, l_2)$ as follows:

$$w(l_1, l_2) = \frac{f(l_1, l_2)}{v(l_1, l_2)}.$$

In the following, we denote by $\{w_{l_1 l_2}\}$ the *weight matrix* such that $w_{l_1 l_2} = w(l_1, l_2)$, and by $\{t_{l_1 l_2}\}$ the *travel time matrix* such that $t_{l_1 l_2} = t(l_1, l_2)$.

Relations between demands and candidates. The deadline of the user requests imposes a bound δ on the travel time $t(d, c)$ between a demand location d and a candidate location c. A demand location d is allocated to a candidate point c if the demand is within the bound δ and the weight w_{dc} is not null (*i.e.*, there are flows of mobile users that travel from the demand to each of the candidates). The goal is to maximize the sum of the weights of the demands $d \in \mathcal{D}$ allocated to a candidate c: $\sum_{d \in \mathcal{D}} \{w_{dc} \mid t_{dc} \leq \delta_{dc}\}$. Note that a single demand cell can be allocated to multiple candidates, each within the given bound δ with non-null weights.

Relations between pairs of candidates. We characterize the connectivity between two candidates in terms of flows of mobile users travelling from c_1 and c_2 with both weights $w_{c_1 c_2}$ and $w_{c_2 c_1}$. For each candidate, we maximize the sum of the weights between this candidate c and the chosen candidates $k \in \mathcal{A}$: $\sum_{k \in \mathcal{A}} w_{kc} + w_{ck}$.

3.2 Location allocation algorithm

The algorithm derives from a greedy-adding with substitution (GAS) heuristic proposed to solve the Maximal Covering Location Problem [2]. For each candidate location, we divide the iterations into two parts: selection and substitution.

Selection. We first pre-select the set of candidates from the set of remaining candidates such that each candidate is connected to the chosen repositories by the flows of mobile users. This pre-selection is necessary, as it guarantees the creation of the distribution network and increases the availability of files. From these pre-selected candidates, we choose the candidate that maximizes the sum of demand weights within the bound δ as we defined previously.

Substitution. The second part of the algorithm goes back on the previous iterations and examines each allocated candidate and tries to find a better candidate that maximizes the demand weights, while guaranteeing that the new candidate will be connected to the chosen candidates. If a better candidate is found, it replaces the candidate under examination. The substitution part allows to replace the candidates allocated in the early iterations that are not justified in the later iterations due to subsequent allocated candidates.

We adapt the algorithm for our system as follows. The first candidate is chosen such that it maximizes the sum of demand weights within the bound δ. Note that, since the demand weights are maximized for this candidate, the candidate is often visited by mobile users, making it a good intermediary node to be connected to the other candidates by the flows of mobile users. There is no substitution part, as no other candidate was already chosen. In our implementation, once a candidate is chosen, we remove all the candidates in the vicinity of the chosen candidate (in terms of travel time and distance from the chosen candidate, *e.g.*, half the deadline and the distance corresponding to this duration with average speed). This allows us to better spread the repositories over the geographic area and avoids cluttering denser regions with too many repositories.

4. EVALUATION

We run our evaluation in two parts. The first part corresponds to the allocation of a given number of repositories for the given deadline. The second part is the simulation of the system with the repositories placed in the area.

Mobility model for public transit vehicles. Firstly, we simulated the movements of the San Francisco MUNI buses[1] using the ONE simulator [6]. The ONE simulator is a time-discrete and event-based simulator of delay-tolerant networks. We used the `MapRouteMovement` mobility model implemented in ONE to simulate the movements of mobile nodes and their stops on a street map. This mobility model is particularly well suited to simulate the movement of buses that are part of a transit system. We derive the movements of the buses from GTFS (*General Transit Feed Specification*)[2] feeds given by the MUNI transit authority that gather the buses' schedules. We develop generic tools to convert the GTFS feeds into a collection of `MapRouteMovement` mobility models compatible with the ONE format.

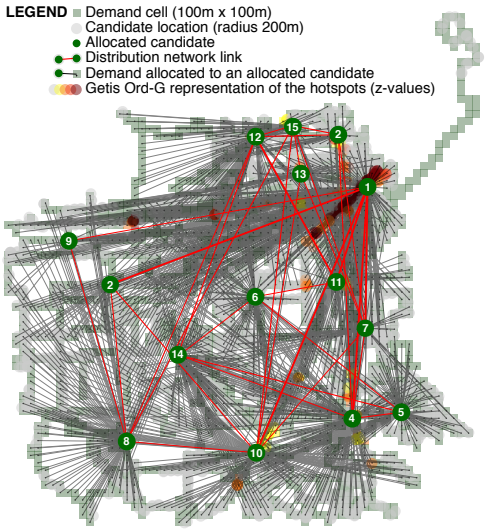
Repository placement. Secondly, we developed a framework to determine the locations of the repositories. The framework implements the algorithm described in Section 3. It takes the bus movements we derived from the GTFS feeds. We divide the geographical area into a grid of cells of size 100m $\times$ 100m. We take bus stops as candidates to host a repository, since the buses stop for a long enough duration to transfer large amounts of data. The framework then allocates the locations of the target number of repositories among the candidate locations, given user request deadline. We represented in Figure 3a the output allocation of 15 repositories' locations out of the 4,590 bus stops in San Francisco using the movements of the MUNI buses. In this figure, we show a Getis Ord-G hotspot analysis of the spatially significant bus stop candidates with regards to the visit frequencies of the buses in their vicinity [5].

After running the tools to convert the GTFS feed, we obtained the movements of 493 buses running on 130 different trajectories, which represents the traffic of buses running between 10am and 4pm on a typical weekday. Both the mobile users and the repositories are equipped with WiFi network interfaces with a range of 250m. We ignore wireless interference and assume enough wireless capacity to accommodate the full exchange of files between the repositories and the mobile users. The candidate locations are the bus stops given by the GTFS bus feed projected onto the street map. The wait time of the buses at the bus stops is picked at random between 10 seconds and 30 seconds. The mobile users send requests independently every 10 minutes following a Poisson distribution, without any assumptions on the popularity and geographical locality of the files. We assume the mobile users generate the `put` and `get` requests as follows: 10% of the requests are `put` requests and 90% of the requests are `get` requests. The simulation duration is 20,000 seconds.

Success ratio. As a first step, we perform the simulations to evaluate the benefits of the `put` policies and the distribution network in the success ratio of the user requests. To

[1] https://www.sfmta.com/

[2] https://developers.google.com/transit/gtfs



(a) Allocation of the repositories. The color variations of the candidate locations represent hotspots resulting from the Getis Ord-G spatial analysis with regard to the bus visit frequencies at these locations [5].

15	51 (7)	9 (3)	28 (5)	35 (8)	54 (12)	52 (9)	31 (6)	40 (5)	87 (13)	49 (5)	34 (7)	6 (3)	9 (3)	47 (8)	0 (0)
14	55 (7)	51 (8)	63 (9)	32 (12)	36 (10)	69 (12)	40 (12)	63 (12)	108 (17)	40 (12)	48 (9)	41 (9)	57 (9)	0 (0)	49 (10)
13	50 (6)	21 (5)	39 (6)	42 (5)	58 (10)	50 (7)	35 (8)	52 (6)	96 (13)	44 (6)	31 (5)	18 (5)	0 (0)	59 (9)	10 (3)
12	46 (5)	15 (4)	23 (5)	38 (6)	56 (12)	46 (9)	37 (5)	35 (4)	81 (14)	54 (4)	27 (7)	0 (0)	16 (5)	41 (10)	7 (3)
11	19 (3)	40 (6)	49 (5)	17 (4)	31 (9)	19 (6)	25 (6)	62 (6)	84 (18)	48 (7)	0 (0)	27 (6)	33 (7)	47 (10)	33 (6)
10	31 (4)	52 (12)	40 (8)	45 (15)	59 (19)	61 (6)	30 (17)	24 (8)	69 (12)	0 (0)	47 (7)	54 (4)	45 (6)	41 (13)	50 (5)
9	70 (20)	95 (11)	61 (13)	92 (19)	108 (19)	96 (20)	83 (16)	44 (14)	0 (0)	71 (14)	84 (17)	84 (14)	98 (14)	108 (17)	90 (13)
8	52 (6)	50 (7)	14 (4)	64 (10)	80 (15)	78 (7)	53 (13)	0 (0)	46 (14)	25 (8)	59 (5)	34 (4)	50 (6)	63 (12)	41 (6)
7	28 (12)	23 (8)	54 (6)	9 (7)	27 (11)	44 (7)	0 (0)	59 (14)	89 (18)	32 (17)	26 (6)	38 (6)	38 (9)	39 (12)	35 (9)
6	33 (4)	61 (8)	67 (6)	36 (7)	39 (8)	0 (0)	43 (7)	80 (6)	99 (19)	62 (6)	19 (6)	47 (8)	53 (10)	67 (12)	55 (8)
5	44 (12)	43 (11)	72 (9)	12 (9)	0 (0)	41 (12)	23 (10)	75 (12)	104 (20)	50 (14)	32 (11)	54 (10)	54 (9)	36 (12)	52 (8)
4	28 (6)	28 (9)	59 (6)	0 (0)	14 (8)	36 (7)	9 (6)	65 (12)	94 (19)	40 (15)	16 (4)	40 (6)	42 (6)	30 (12)	39 (6)
3	41 (8)	39 (6)	0 (0)	59 (6)	79 (12)	67 (7)	55 (6)	16 (3)	64 (16)	41 (8)	49 (6)	23 (4)	39 (6)	63 (8)	30 (6)
2	50 (11)	0 (0)	40 (6)	26 (9)	45 (12)	58 (8)	24 (9)	53 (6)	96 (14)	51 (12)	40 (6)	17 (5)	21 (5)	50 (7)	12 (4)
1	0 (0)	49 (11)	40 (8)	27 (5)	44 (11)	32 (5)	28 (12)	52 (6)	68 (20)	31 (4)	17 (3)	45 (8)	50 (8)	60 (12)	50 (7)
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

(b) Origin-destination matrix of the repositories (identified by the numbers represented on the Figure 3a showing the mean travel time in minutes between each pair of repositories. The values in parentheses are the corresponding standard deviation. The simulations were done for 15 repositories placed as shown in Figure 3a.

Figure 3: Results for the allocation of 15 repositories in San Francisco using the bus movements of the MUNI bus transit system.

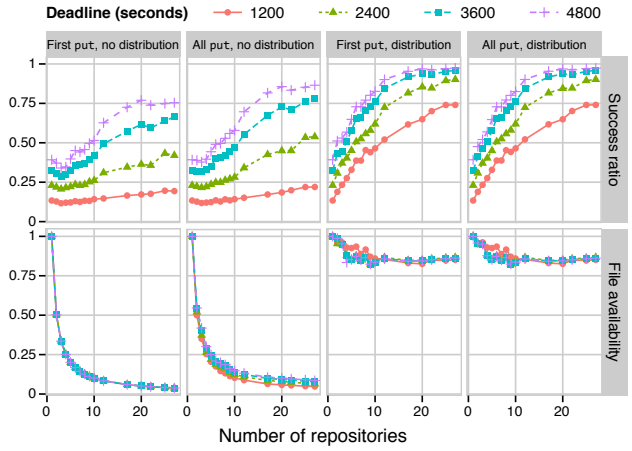


Figure 4: Comparison of the impact of the different `put` policies and the existence of the distribution network in the user request success ratio and the file availability in the repositories.

this end, we simulate the system under different assumptions regarding the `put` policy ("first" or "all") and the existence of the distribution network. We represent the average success ratio of the `get` user requests and the file availability in the repositories in Figure 4. In the scenario with the "first" `put` policy and no distribution network, we notice that the more repositories there are, the more user requests are satisfied. Some buses that have pending `get` requests are not able to reach any repository because of their routes and the few repositories available. Adding more repositories allows more buses to reach repositories, resulting in better success ratio of the `get` user requests. However, since the data is scattered across multiple repositories, the success ratio of the `get` requests still remains low for short deadlines. For long deadlines, the buses visit more repositories, which increases their chances to visit one with a copy of the requested file. Taking advantage of the "all" `put` policy helps increase the success ratio of the `get` requests by 9.98% on average for all the deadlines. With this policy, the buses distribute copies of the files to different repositories, thus increasing the availability of the files. Contrary to the "first" `put` policy where user files are available at only one repository, the "all" `put` policy increases the chance of a mobile user to find a repository with a local copy of the requested file.

The distribution network led to increase the user request success ratio by 104.3%. Note that this improvement is greater for lower user request deadlines (*e.g.*, the improvement for the user requests of 1200 seconds is of 201.4% on average). The distribution network distributes copies of the files in several repositories. Since the repository placement algorithm guarantees that all the repositories are connected together, the files can be distributed to each repository of the system. Hence, a mobile user with a pending `get` request is more likely to encounter a repository that has a local copy of the requested file, which increases the success ratio of the `get` requests.

File availability. We compare the benefits of the distribution network with the "all" `put` policy using the "File availability" plots in Figure 4. The plots show the proportion of files available at the repositories, where "1.0" of file availability corresponds to all the files being available at each repository. We clearly see the benefits of the distribution network to distribute the files to the repositories. The "all" `put` policy also distributes the files to several repositories, however, it fails to distribute the files to repositories beyond those located on the routes of the mobile users that generated the `put` requests. The plot further shows that, even with the distribution network, the files are not available at every repository (only 85% of them). In fact, the data represented accounts for the files that were not fully distributed by the end of the simulation.

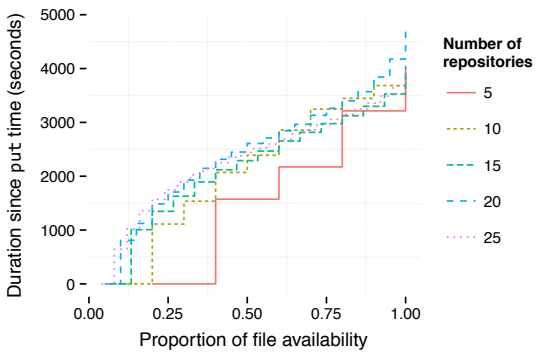


Figure 5: Distribution duration as a function of the file availability for different numbers of repositories.

Distribution duration. In the second step of our evaluations, we study the distribution duration of the copies of user files among the available repositories. In Figure 5, we show the average duration to distribute a file throughout the distribution network since the time the file was stored in the first repository (following a `put` request). We represent the distribution duration of the copies of user files as a function of the proportion of the file availability for different numbers of repositories. For example, with 10 repositories, 40% of file availability corresponds to the file being available in four repositories. The first repository where the file is stored can be any of the repositories available. We notice that the average time needed to distribute the user files to every repository, regardless of the number of repositories available, is 4000 seconds, or a little more than one hour. It takes more time to distribute the copies of the files to the repositories at the beginning and end of the file distribution. At the beginning of the file distribution, only one copy of the files is available at the first repository. It takes on average 700 seconds to distribute the first copy of the file to the second repository. As more repositories distribute copies of the files, the distribution of the copies becomes faster. By the end of the file distribution, copies of files are available at most repositories. It takes on average 500 seconds to reach the last repository, as it is the farthest away from the first repository where the original copy of the file was stored.

In Figure 3b, we give an origin-destination matrix that shows the average travel time in minutes between any pair of repositories. This translates to the time needed to distribute the file to different repositories from the time the file was stored in a first repository (following a `put` request). These values give the average duration for a file to be available in a repository. In the figure, the repositories are identified by the same numbers as the ones in Figure 3a. The connectivity between two repositories depends on the number and frequency of the buses going from one repository to another. For instance, repositories 1 and 15 are very well connected to the rest of the repositories. However, repository 9 is not as well connected since it is farther away from the rest of the repositories. This further explains the longer time it takes to distribute copies of the user files from repository 9 to every other repository. This goes to show that the further away the repositories are, the more time it takes on average to reach them and distribute copies of the files to them.

5. CONCLUSION

In this paper, we presented a cloud-like file storage and sharing system. Users share files on the system by seamlessly storing and retrieving files on pre-positioned repositories within a bounded deadline. A challenge we identified in this paper is the processing of user requests in a timely fashion. To this end, we leverage the movements of the mobile users between the repositories to distribute copies of the user files throughout the system. Additionally, we designed an algorithm to place a given number of repositories such that they serve the maximum number of user requests. We evaluated the storage system with simulated movements of San Francisco buses. We showed how the number of storage nodes impacts the success of the user requests to retrieve user files. We also evaluated the distribution duration of the copies of the files with the movements of the users between the repositories. Regardless of the number of repositories available, it takes a little more than an hour to distribute a file to each repository placed in San Francisco using traces of the MUNI buses.

As future work, we plan to implement distribution strategies that replicate copies of the files to some of the repositories. We also intend to take into account the limited capacity of wireless transmissions between the mobile users and the repositories, as well as the contention when multiple nodes try to exchange files with a repository. This creates more challenges with regard to the scheduling of the requests executed at each repository.

References

[1] B. Baron, P. Spathis, H. Rivano, and M. D. de Amorim. Offloading massive data onto passenger vehicles: Topology simplification and traffic assignment. *IEEE/ACM Transactions on Networking*, 2015.

[2] R. Church and C. R. Velle. The maximal covering location problem. *Papers in regional science*, 1974.

[3] M. J. Demmer, B. Du, and E. A. Brewer. Tierstore: A distributed filesystem for challenged networks in developing regions. In *FAST*, 2008.

[4] K. Fall. A delay-tolerant network architecture for challenged internets. In *ACM SIGCOMM*, 2003.

[5] A. Getis and J. K. Ord. The analysis of spatial association by use of distance statistics. *Geographical analysis*, 1992.

[6] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation. In *ACM Simutools*, 2009.

[7] N. Megiddo, E. Zemel, and S. L. Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 1983.

[8] J. Ott and M. J. Pitkänen. Dtn-based content storage and retrieval. In *IEEE WoWMoM*, 2007.

[9] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Elsevier Ad Hoc Networks*, 1(2):215–233, 2003.

[10] W. Zhao, Y. Chen, M. Amma, M. Corner, B. Levine, and E. Zegura. Capacity enhancement using throwboxes in dtns. In *IEEE MASS*, 2006.