P2P and Cloud: A Marriage of Convenience for Replica Management

Hanna Kavalionak, Alberto Montresor

P2P and Cloud: A marriage of convenience for replica management*

Hanna Kavalionak and Alberto Montresor

University of Trento

Abstract. P2P and cloud computing are two of the latest trend in the Internet arena. They could both be labeled as "large-scale distributed systems", yet their approach is completely different: based on completely decentralized protocols exploiting edge resources the former, focusing on huge data centers the latter. Several Internet startups have quickly reached stardom by exploiting cloud resources, unlike P2P applications which often lack a business model. Recently, companies like Spotify and Wuala have started to explore how the two worlds could be merged, exploiting (free) user resources whenever possible, aiming at reducing the bill to be payed to cloud providers. This mixed model could be applied to several different applications, including backup, storage, streaming, content distribution, online gaming, etc. A generic problem in all these areas is how to guarantee the autonomic regulation of the usage of P2P vs. cloud resources: how to guarantee a minimum level of service when peer resources are not sufficient, and how to exploit as much P2P resources as possible when they are abundant.

1 Introduction

Cloud computing represents an important technology to compete in the world-wide information technology market. According to the U.S. Government's National Institute of Standards and Technology[1], cloud computing is *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)"*. Usually cloud services are transparent to the final user and require minimal provider interactions. By giving the illusion of infinite computing resources, and more importantly, by eliminating upfront commitment, small- and medium-sized enterprises can play the same game as web behemoths like Microsoft, Google and Amazon.

The rise of cloud computing has progressively dimmed the interest in another Internet trend of the first decade of this century: the peer-to-peer (P2P) paradigm. P2P systems are composed by nodes (i.e. peers) that play the role of both clients and servers. In this architecture the workload of the application is managed in a distributed fashion without any point of centralization. The lack of centralization provides scalability, while exploitation of user resources reduces the service cost. However, several drawbacks exist, like availability and reliability.

* This work is supported by the Italian MIUR Project *Autonomous Security*, sponsored by the PRIN 2008 Programme.

[1] http://csrc.nist.gov

P2P held similar promises with respect to cloud computing, but with relevant differences. While P2P remains a valid solution for cost-free services, the superior QoS capabilities of the cloud makes it more suitable for those who want to create novel web businesses and cannot afford to lose clients due to the best-effort philosophy of P2P.

An interesting "middle ground" combines the benefits of both paradigms, offering highly-available services based on the cloud while lowering the economic cost by exploiting peers whenever possible. Potential applications include storage and backup systems [19, 16], content distribution [11, 15], music streaming [10], and potentially online gaming and video streaming [2].

In all these systems, a generic problem to be solved is related to the autonomic regulation of resources between the cloud and the set of peers. As an example, consider an online gaming system, where the current state of the game is stored on a collection of machines rented from the cloud. As the number of players increase, it would be possible to transfer part of the game state to the machines of the player themselves, solving the problem with a P2P approach. This would reduce the economical costs of the cloud, yet providing a minimum level of service when peer resources are insufficient or entirely absent.

In order to strike a balance between service reliability and economical costs, we propose a self-regulation mechanism that focuses on replica management in cloud-based, peer-assisted applications. Our target is to maintain a given level of peer replicas in spite of churn, providing a reliable service through the cloud when the number of available peers is too low.

The background needed to understand this work is given in Section 2. Section 3 describes the model of the system and provides a more detailed overview of the research problems associated to this work. The description of the solution is given in Section 4. Section 5 presents the evaluation results, while Section 6 discusses related work. The last section concludes the paper and discusses future work.

2 Background

Unstructured topologies are a promising architectural design for cloud-based, peer-assisted applications. Unstructured P2P networks have indeed a number of attracting properties that suit this kind of applications. Links between nodes are established arbitrarily. To join the network a node just have to acquire a set of links from one of the peers of the network. These important properties are able to reduce the network overload caused by churn.

One of the most notable example of unstructured P2P networks is based on *epidemic* (or *gossip*) protocols [4]. The popularity of these protocols is due to their ability to diffuse information in large scale distributed systems even when large number of nodes are continuously joining and leaving the system. Such networks have proven to be highly scalable, robust and resistant to failures.

One of the fundamental problems of epidemic protocols is how to maintain information about the set of peers participating to the network. Ideally, each node should be aware of all peers in the system, and periodically selects one of them for state exchange.

But in large, dynamic networks this is impossible: the cost of updating all peers whenever a new peer joins or leave is simply too high. To solve this problem, each node maintains a partial view of the network composing the overlay [8, 17]. This approach provides each peer with information about a random sample of the entire population of peers in the system, so the network can be represented by a small amount of local information.

As we already mentioned, our work focuses on cloud-based applications. A particular attention is devoted to CLOUDCAST protocol, proposed by Montresor and Abeni [11]. The protocol uses cloud storage to support information diffusion and connectivity in unstructured overlays. The cloud participates in the communication protocol as a normal peer. This heterogeneous architecture adopts CYCLON for peer sampling [17], where the cloud is used to increase reliability of the information diffusion service. The attractive idea from this paper is that the cloud plays the role of static element and the rate of cloud utilization is fixed, no matter what the size of the system is.

In our work, cloud resources are exploited only in the presence of reliability risks. From an operative point of view, our adaptive system is able to switch the behavior between CLOUDCAST and pure CYCLON protocols according to system parameters, reducing the economical effort with respect to previous solutions.

One of the key aspects of autonomic replica management is monitoring of system components. Self-monitoring is vital for self-organized systems, because it allows the system to have a view on its current use and state.

The most popular solutions for autonomous monitoring for unstructured P2P overlays are based on broadcasting algorithms. Updated information about nodes state is broadcast through the system. However, this approach may incur in large network overhead [6].

In this work we adopt the aggregation method proposed by Jelasity et al. [7] to estimate the network size. The protocol continuously provides up-to-date estimation of an aggregate function (in our case the number of nodes in the overlay) computed over all nodes. Time is divided into intervals, so called epochs. At the beginning of each epoch the protocol is restarted. At the end of the epoch each node knows the approximate size the overlay the node had at the beginning of the epoch. The length of a single epoch determines the accuracy of the measure. The convergence rate does not depend on the network size, hence the monitoring cost for each epoch is determined only by the requested accuracy and network churn rate.

3 System model and problem statement

We consider a system composed by a collection of peer nodes that communicate with each other through reliable message exchanges. The system is subject to churn, i.e. node may join and leave at any time. Byzantine failures are not considered in this work, meaning that peers follow their protocol and messages integrity is not at risk. Peers form a random overlay network based on peer sampling service [8].

Beside peer nodes, an highly-available cloud entity is part of the system. The cloud can be joined or removed from the overlay according to the current system state. The cloud can be accessed by other nodes to retrieve and write data, but cannot autonomously

initiate the communication. The cloud is pay-per-use: storing and retrieving data with a cloud is connected with monetary costs.

The problem to be solved is the maintenance of an appropriate redundancy level in replicated storage. Consider a system where a collection of data objects are replicated in a collection of peers. A data object is *available* if at least one replica can be accessed at any time, and is *durable* if it can survive failures.

If potentially all peers replicating an object fail or go offline, or are temporary unreachable, the data may become unavailable or even be definitely lost. A common approach to increase durability and availability is to increase the number of replicas of the same data. Nevertheless, the replicas of the same data should be synchronized among each other and retrieved in case of failure. Hence, there is a trade-off: the higher amount of replicas, the higher the network overhead. Furthermore, a peer-to-peer system may have a limited amount of storage available, so increasing the number of replicas of an object can decrease the number of replicas of another one.

An alternative approach is to store a replica of each piece of data in the cloud [19]. This method improves reliability and scalability. Nevertheless, the economical cost grows proportionally to the size of storage space used.

In order to autonomously support the target level of availability and durability and reduce the economical costs we exploit a hybrid approach. The replicas of the same data object are organized into an overlay. To guarantee reliability (durability and availability), the data object must be replicated a predefined number of times. In case P2P resources are not enough to guarantee the given level of reliability, the cloud resources are exploited, potentially for a limited amount of time until P2P resources are sufficient again.

The main goal is thus to guarantee that the overlay maintains a given size, including the cloud in the overlay only when threats of data loss do exist.

In order to realize such system, the following issues have to be addressed:

- To reach the maximum economical effectiveness of the network services, we have to effectively provide mechanisms to automatically tune the amount of cloud resources to be used. The system has to self-regulate the rate of cloud usage according to the level of data reliability (overlay size) and service costs.
- To regulate the cloud usage, each peer in the overlay has to know an updated view of overlay size. Hence, one of the key aspects that has to be considered is the network overhead caused by system monitoring.
- Churn rate may vary due to different reasons such as time of day, day of the week, period of the year, etc. and average lifetime of nodes can be even shorter than one minute. Hence, the size of the overlay can change enormously. Data reliability must be supported even in the presence of one single replica (the replica stored on the cloud), as well as with hundreds of replicas (when the cloud is not required any more and should be removed to reduce monetary costs).

4 The algorithm

As already mentioned, our goal is to build an unstructured overlay linking the replicas of a data object and maintaining a minimal size in spite of churn. To obtain this, we make

use of a number of well-known protocols like CLOUDCAST [11] and CYCLON [17] for overlay maintenance and an aggregation protocol [7] to monitor the overlay size.

We consider the following three thresholds for the overlay size: (1) *redundant* R, (2) *sufficient* S and (3) *critical* C, with $R > S > C$. The computation of the actual threshold values is application-dependent; we provide here only a few suggestions how to select reasonable parameters, and we focus instead on the mechanisms for overlay size regulation, general enough to be applied to a wide range of applications.

- The *critical* threshold C should be the sum of (i) the minimum number of replicas that are enough for successful data recovering and (ii) the amount of nodes that will leave during the cloud backup replication phase. The minimum number of replicas is determined by the chosen redundancy schema, such as erasure coding or data replication.
- The *sufficient* threshold S can be computed as the sum of (i) *critical* threshold and (ii) the amount of nodes that are expected between two successive recovery phases, meaning that these values depends also on the expected churn rate.
- The *redundant* threshold R is an important parameter that allows to optimize the network resources utilization. The *redundant* threshold depends on the overlay membership dynamics (i.e. peers continuously join and leave the network) and on the application type. For example, backup applications are more sensitive to replica redundancy than content delivery applications. In a typical backup application, users save their data in the network and R is determined according to the adopted redundancy schema and overlay membership dynamics.
  On the other hand, in content delivery applications, popular data is replicated in a high (possibly *huge*) amount of nodes of the network. In this case the *redundant* threshold is not relevant, since peers keep pieces of data based on their popularity. Nevertheless, the *critical* and *sufficient* thresholds are important in order to keep data available during the oscillation of network population.

In general, the size of the overlay is expected to stay in the range $[S, R]$. When the current size is larger than R, some peers could be safely removed; when it is smaller than S, some peers (if available) should be added. When it is smaller than C, the system is in a dangerous condition and part of the service should be provided by the cloud.

Figure 1 illustrates the behavior of our system in the presence of churn. The size $N(t)$ oscillates around threshold S, with $k\%$ deviation. To withstand the oscillation, a system recovery process is executed periodically, composed of *monitoring* phase with duration T_m, a recovery phase with duration T_r and *idle* phase of duration T_i. The length of the monitoring phase depends on the particular protocol used to monitor the system, while the idle phase is autonomically regulated to reduce useless monitoring and hence overhead.

Monitoring is based on the work of Jelasity et al. [7]. The authors proposed a gossip protocol to compute the overlay size, that is periodically restarted. The execution of the protocol is divided into epochs, during which a fixed number of gossip rounds are performed. The number of rounds determines the accuracy of the measure. At the end of each epoch, each peer obtains an estimate of the size as measured at the beginning of the epoch.

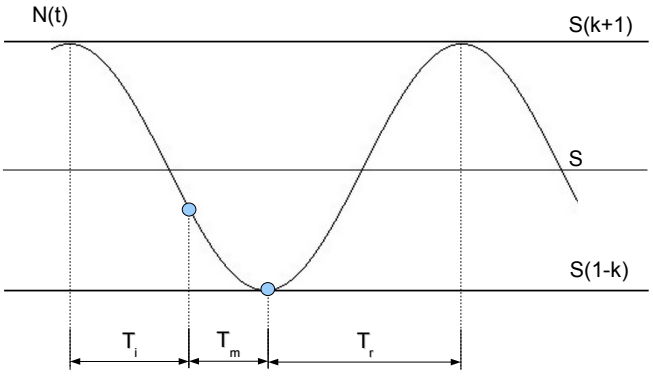

Fig. 1. Overlay size oscillation

We have tuned the aggregation protocol in order to fit our system, with a particular emphasis on reducing network overhead. In our version, each aggregation epoch is composed by monitoring and idle periods. The monitoring period corresponds to the aggregation rounds in the original protocol [7], whereas in idle periods the aggregation protocol is suspended. Our approach allows to reduce network overhead by reducing the amount of useless aggregation rounds.

All peers follow the algorithm in Figure 2. The system repeats a sequence of actions forever. The monitoring phase is started by the call to getSize(), which after T_m time units returns the size of the system at the *beginning* of the monitoring phase. Such value is stored in variable N_b. Given that during the execution of the monitoring phase further nodes may have left the system, the expected size of the system at the *end* of the monitoring phase is computed and stored in variable N_e. Such value is obtained by computing the failure rate f_r as the ratio of the number of nodes lost during the idle phase and the length of such phase.

According to the size of the overlay a peer decides to start the recovering process, delete/create cloud from the overlay or do nothing. If the system size $N_e \geq S$, the peer removes the (now) useless links to the cloud from its neighbor list; in other words, the system autonomously switch the communication protocol from CLOUDCAST to CYCLON. Furthermore, if the system size is larger than threshold R, the excess peers are removed in a decentralized way: each peer independently decide to leave the system with probability $p = (N_e - R)/N_e$, leading to an expected number of peers leaving the system equal to $N_e - R$.

When the number of peers is between critical and sufficient ($C < N_e < S$) a peer invokes function recovery() to promote additional peers to the overlay in a distributed fashion. Each peer in the overlay has to promote only a fraction of additional peers N_a, computed as the ratio of the number of peers needed to obtain the desired $S(1+k)$ nodes and the current overlay size N_e. The ratio is rounded to the upper bound. Function addNode() adds new peers from the underlying overlay to the set of replicated entities. To bootstrap new peers to the overlay, the peer, currently promoting other peers, copies a set of random descriptors from its own local view copyRandom($view$) and sends them

```
repeat
    N_b ← getSize();
    f_r ← (N_b − N_e)/T_i;
    N_e ← N_b − T_m · f_r;
    if N_e ≥ S then
        removeCloud();
        if N_e > R then
            p ← (N_e − R)/N_e;
            leaveOverlay(p);
    else if N_e < S then
        if N_e ≤ C then
            addCloud();
        recovery()
    T_i ← idleTime(N_e, f_r, T_i);
    wait T_i;
```

```
idleTime(N_e, f_r, T_i)
    if N_e ≤ C then
        T_i ← T_min;
    else if f_r > 0 then
        T_i ← S·k/f_r − T_m;
    else if T_i < T_max then
        T_i ← T_i + Δ;
    return T_i;

recovery()
    N_a ← ⌈S(1+k)/N_e − 1⌉;
    for j ← 1 to N_a do
        V ← copyRandom(view);
        if N_e ≤ C then
            V ← V ∪ {cloud};
        addNode(V);
```

Fig. 2. Algorithm executed by peers

as a V to the new peers. A local view (indicated as $view$) is a small, partial view of the entire network [17].

When the overlay size is below critical ($N_e \leq C$), a peer first adds to the local view a reference to the cloud (function addCloud()) and then starts the recovering process. Moreover, the reference to the cloud is also added to a set of random descriptors V that is sent to the new peers. Hence, in case of a reliability risk, the system autonomously switches the protocol back to the CLOUDCAST (i.e. it includes a reliable cloud node into the overlay).

Concluded this phase, the peer has to compute how much it has to wait before the next monitoring phase starts; this idle time T_i is computed by function idleTime().

If the overlay size is lower than C, the idle time is set to a minimum value T_{min}. As a consequence, the monitoring phase starts in a shorter period of time. Reducing of the idle period allows the system to respond quickly to the overlay changes and keep the overlay size in a safe range. In case the amount of replicas has decreased ($f_r > 0$), the idle time is computed as the ratio of the deviation range $S \cdot k$ and the expected failure rate f_r. The ratio is decreased in T_m time entities needed for the next overlay monitoring phase. If the result T_i is less than T_{min}, the final T_i is set to T_{min}. When either the number of replicas is stable or it has increased ($f_r \leq 0$), the idle period is increased by a fixed amount Δ. The duration of the idle phase is limited by the parameter T_{max}, which is decided by the system administrator.

It is important to note that, in order to increase the amount of overlay peers in the network, a peer has to add them faster then failure rate. In fact, promoting rate of new peers depends on several factors, such as network topology and physical characteristics of the network.

Table 1. Parameters used in the evaluation

Parameter	Value	Meaning
n	0-1024	Total number of peers in the underlying network
R	60	*Redundant* threshold
S	40	*Sufficient* threshold
C	10	*Critical* threshold
k	0.2	deviation from *sufficient* threshold
σ_{cyclon}	10s	Cycle length of CYCLON
$\sigma_{entropy}$	10s	Cycle length of anti-entropy
$\sigma_{failure}$	1s	Cycle length of peers failure
c	20	View size
g	5	Cyclon message size
Δ	1s	Incremental addition step for the idle period T_i

5 Evaluation

The repairing protocol has been evaluated through an extensive number of simulations based on event-based model of PEERSIM [12]. Due to limited space, we present here a brief evaluation of the behavior of the system, leaving an extended description to future work. Unless explicitly stated, the parameters that are used in the current evaluation are shown in Table 1. Such parameters are partially inherited from the CYCLON [17], CLOUDCAST [11] and Aggregation protocols [7]. The choice of the thresholds parameters in Table 1 is motivated by graphical representation; in reality, the protocol is able to support both smaller and bigger sizes of the overlay.

The first aspect that has to be evaluated is protocol scalability: is our protocol able to support overlay availability in spite of network size oscillation? Figure 3 and Figure 4 shows the behavior of the overlay when network size oscillates between 0 and 1024 peers in a one-day period. The average peer life time is around 3 hours. When the network size grows, the overlay peers promote new peers to the overlay and its size growths until reaching the sufficient threshold. Then the protocol supports the sufficient overlay size with k deviation. When all peers leave the network, the overlay size falls to zero and contains only the cloud peer. This behavior repeats periodically.

Figure 5 shows the amount of existing links to the cloud in the overlay. The top figure shows the network size oscillation and the bottom one represents the cloud in-degree for CLOUDCAST (thin line) and our protocol (thick line). As you can see, compared with CLOUDCAST, our protocol significantly reduces the utilization of cloud resources without implications on the overlay reliability. When the network size is enough to provide the *sufficient* overlay size, references to the cloud are removed; otherwise the overlay peers include cloud node into the overlay.

For system monitoring we adopt the work of Jelasity et al. [7]. Figure 6 shows the ability of the protocol to support the target overlay size under the various network churn rates and epoch intervals. The X-axis represents the duration of the epoch intervals. The Y-axis represents the deviation of the overlay size from the target *sufficient* threshold. A churn rate $p\%$ for the overlay means that at each second, any peer may abruptly leave

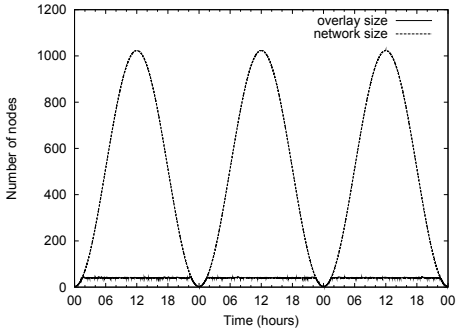

Fig. 3. Overlay size in case of network oscillates daily between 0 and 1024 peers. The single experiment lasted 3 days.

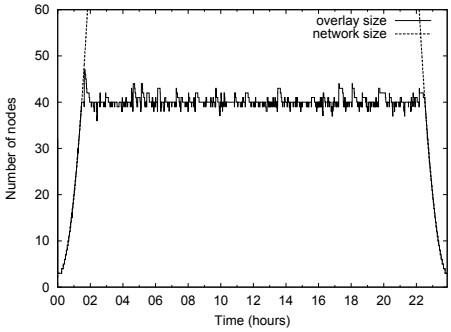

Fig. 4. Overlay size in case of network oscillates daily between 0 and 1024 peers. Zooming over a single day.

the overlay with probability $p\%$. In this evaluation we do not consider that peers can rejoin the overlay.

As we can see from Figure 6, the lines correspond to 0%, 0.1% and 0.01% shows approximately the same behavior. These results proof that the system successfully supports the target deviation ($k = 0.2$). However, with the churn rate 1% the system is not able to support the overlay with an epoch interval larger then 30 seconds. Nevertheless, it must be noted that the churn rate of 1.0% corresponds to an average lifetime of less then 2 minutes.

6 Related Work

One of the most important limitations in P2P-based storage systems is connected with the difficulty to guarantee data reliability. P2P nodes leave or fail without notification and stored data can be temporarily or permanently unavailable.

A popular way to increase data durability and availability is to apply redundancy schemas, such as replication [9, 3], erasure coding [18] or a combination of them [13]. However, too many replicas may harm performance and induce excessive overhead. Therefore it becomes a relevant issue to keep the number of replica around a proper threshold, in order to have predictable information on load and performance of the system.

Two main approaches exist for replica control: (1) proactive and (2) reactive. Proactive approach creates replicas at some fixed rate [14, 13], whereas in reactive approach a new replica is created each time an existing replica fails [9, 3].

The work of Kim [9] proposes a reactive replication approach that uses lifetime of nodes to select where to replicate data. When the data durability is below a threshold the node does replicas in nodes whose lifetime is long enough to assure durability. Conversely, Chun et al.[3] proposes a reactive algorithm, where data durability is provided by selecting a suitable amount of replicas. The algorithm responds to any detected failure and creates a new replica if the amount of replicas is less then the predefined minimum.

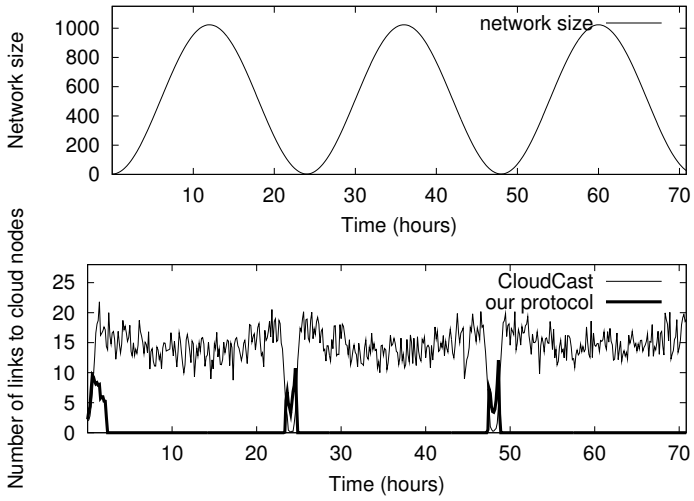

Fig. 5. Cloud in-degree for CLOUDCAST and our protocols in case of network oscillates daily between 0 and 1024 peers.

Nevertheless, these approaches are not taking in account the redundancy of replicas in the network. In fact, increasing the number of replicas affects bandwidth and storage consumption [1].

The problem of replica control is a relevant problem and one of the most complete work in this area is presented by Duminuco et al. [5]. To guarantee that data is never lost the authors propose an approach that combines optimized proactive with pure reactive replication. The approach is able to dynamically measure churn fluctuations and adapt the repair rate accordingly.

Nevertheless, Duminuco et al. consider the estimation of recovering rate, whereas in our work the attention is devoted for the periods of recovery restarting and network overloads caused by system monitoring. Moreover, Duminuco et al in their work do not consider the distributed mechanism of regulation the overlay population.

Finally, recent works in the field do not consider the oscillation of the network size and the situation when P2P resources are not enough or not available. To solve these problems we use the combination of both communication models: P2P and cloud Computing. Moreover, we propose a mechanism that automatically includes or excludes the cloud into the overlay, by adapting the system to the network state.

7 Conclusions

In this paper, we have considered a hybrid peer-to-peer/cloud network, where a replicated service is provided on top of a mixed peer-to-peer and cloud system. We have designed a protocol that is able to self-regulate the amount of cloud (pay-per-use) resources when peer resources (free) are not enough. Future work include considering a

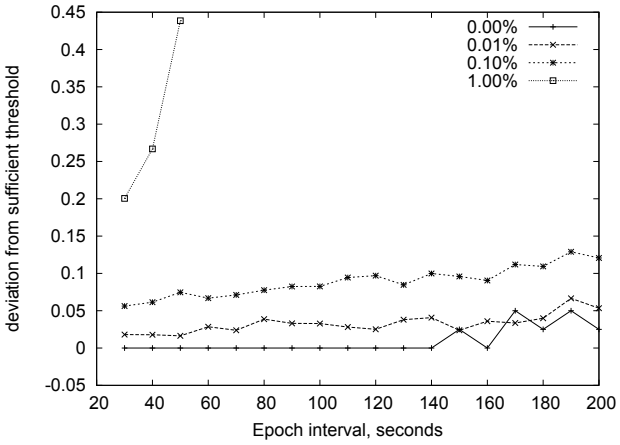

Fig. 6. Deviation of the overlay size from the *sufficient* threshold under different levels of churn with variable epoch intervals.

full-fledged solution for one of the application areas listed in the introduction, such as distributed storage, online gaming or video streaming.

References

1. Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *Proc. of the 9th Conf. on Hot Topics in Operating Systems*, Lihue, Hawaii, 2003. USENIX.
2. Emanuele Carlini, Massimo Coppola, and Laura Ricci. Integration of P2P and clouds to support massively multiuser virtual environments. In *Proc. of the 9th Workshop on Network and Systems Support for Games (NetGames'10)*, Taipei, Taiwan, 2010. IEEE Press.
3. Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proc. of the 3rd Conf. on Networked Systems Design & Implementation (NSDI'06)*, San Jose, CA, 2006. USENIX.
4. Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th ACM Symposium on Principles of Distributed Computing Systems (PODC'87)*, pages 1–12, 1987.
5. Alessandro Duminuco, Ernst Biersack, and Taoufik En-najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Conference on Emerging Network Experiment and Technology*, 2007.
6. Stratos Idreos, Manolis Koubarakis, and Christos Tryfonopoulos. P2P-diet: an extensible P2P service that unifies ad-hoc and continuous querying in super-peer networks. In *Proc. of the 2004 Int. Conf. on Management of Data*, SIGMOD'04, pages 933–934, Paris, France, 2004. ACM.
7. Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252, August 2005.

8. Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25, August 2007.
9. Kyungbaek Kim. Lifetime-aware replication for data durability in P2P storage network. *IEICE Trans. on Communications*, E91-B:4020–4023, December 2008.
10. Gunnar Kreitz and Fredrik Niemelä. Spotify – large scale, low latency, P2P music-on-demand streaming. In *10th IEEE Int. Conf. on Peer-to-Peer Computing (P2P'10)*, pages 1–10, Delft, The Netherlands, August 2010.
11. Alberto Montresor and Luca Abeni. Cloudy weather for p2p, with a chance of gossip. In *Proc. of the 11th IEEE P2P Conference on Peer-to-Peer Computing (P2P'11)*, pages 250–259. IEEE, August 2011.
12. Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.
13. Lluis Pamies-Juarez and Pedro Garcia-Lopez. Maintaining data reliability without availability in P2P storage systems. In *Proc. of the 2010 ACM Symp. on Applied Computing*, SAC'10, pages 684–688, Sierre, Switzerland, 2010. ACM.
14. Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Robert Morris, M F Kaashoek, and John Kubiatowicz. Proactive replication for data durability. In *5th Int. Workshop on Peer-to-Peer Systems*, IPTPS'06, 2006.
15. Raymond Sweha, Vatche Ishakian, and Azer Bestavros. Angels in the cloud – A peer-assisted bulk-synchronous content distribution service. Technical Report BUCS-TR-2010-024, CS Department, Boston University, August 2010.
16. László Toka, Matteo Dell'Amico, and Pietro Michiardi. Online data backup: A peer-assisted approach. In *IEEE 10th Int. Conf. on Peer-to-Peer Computing*, pages 1–10, 2010.
17. Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *J. Network Syst. Manage.*, 13(2), 2005.
18. Chris Williams, Philippe Huibonhoa, JoAnne Holliday, Andy Hospodor, and Thomas Schwarz. Redundancy management for P2P storage. In *Proc. of the Seventh IEEE Int. Symp. on Cluster Computing and the Grid*, CCGRID'07, pages 15–22, Washington, DC, USA, 2007. IEEE Computer Society.
19. Zhi Yang, Ben Y. Zhao, Yuanjian Xing, Song Ding, Feng Xiao, and Yafei Dai. Amazingstore: available, low-cost online storage service using cloudlets. In *Proc. of the 9th Int. Workshop on Peer-to-peer Systems*, IPTPS'10, San Jose, CA, 2010. USENIX.