Integrated Tools for the Simulation Analysis of Peer-To-Peer Backup Systems

Olivier Dalle
Université Nice Sophia Antipolis
Laboratoire I3S UMR CNRS 7271
INRIA Sophia Antipolis - Méditerranée
Sophia Antipolis, France
olivier.dalle@inria.fr

Emilio P. Mancini
Mascotte Project,
INRIA Sophia Antipolis - Méditerranée
Laboratoire I3S UMR CNRS 7271
Sophia Antipolis, France
emilio.mancini@inria.fr

ABSTRACT

In order to evaluate the performance and estimate the resource usage of peer-to-peer backup systems, it is important to analyze the time they spend in storing, retrieving and keeping the redundancy of the stored files. The analysis of such systems is difficult due to the random behavior of the peers and the variations of network conditions. Simulations provide a unique means for reproducing such varying conditions in a controlled way. In this paper we describe a general meta-model for peer-to-peer backup systems and a tool-chain, based on SimGrid, to help in their analysis. We validated the meta-model and tool-chain through the analysis of a common scenario, and verified that they can be used, for example, for retrieving the relations between the storage size, the saved data fragment sizes and the induced network workload.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Miscellaneous; D.4.8 [**Software Engineering**]: Performance—*simulation*

1. INTRODUCTION

In the last decade, new trends geared toward distributed topologies for persistent and volatile memory have emerged. Improved performance and cost of non-volatile memory can take a better advantage from distributed models and from modern interconnect as InfiniBand [9], for building new generations of distributed memorization systems. However, the performance characterization of such systems becomes challenging due to the network conditions and to the perturbations of cooperating nodes.

A range of solutions already provide commercial service for networked storage. For example, Amazon offers on line storage with S3 services [16], and Microsoft with the Azure system [12]. While traditional storage systems, like NAS or distributed file systems, are very reliable, and became cheaper as the technology advances, there is a growing interest for the peer-to-peer distributed storage solution. Oceanstore [13], for example, is one such attempt to offer better dependability and performance than centralized architectures using a peer-to-peer architecture.

Theoretically, they can reach better performance, reliability and scalability; however, the optimization of a peer-to-peer dynamic network poses great challenges. For example, given a known network architecture, it is important to make an estimation of the time required to store a file's fragment. Due to the number of hosts involved and their ability to join or leave the network unpredictably, simulation offer a practical means for studying the performances of such systems in a controlled experimental environment.

Peer-to-peer storage systems require some redundancy on order to compensate for the failure or loss of peers and ensure the reliability: for example, they may split the incoming data in fragments, and dispatch them on different peers. The redundancy may either consist in replicating the fragments or by computing new additional fragments using erasure code techniques [5]. The accent on the reliability makes such systems interesting for backup purpose. However, the reliability comes at a cost, because data fragments are continuously lost: when a peer suffers a permanent failure, or when it simply leaves the system for too long, the system needs to compensate and regenerate the missing data from other peers in the system. The creation and re-creation of lost pieces, requires computational and bandwidth resources, that can slow down the whole system. The peers, moreover, tend to send the fragments on the network concurrently, which may result in bursty bandwidth usage patterns. On these premises, can we estimate the required time to store a fragment, on a peer-to-peer architecture, using a specific fragment size, and with a specific amount of load, comparing the results of different choices? To answer this question, we present an integrated test-bed for peer-to-peer backup simulation models. Our design has the goal to be easily adapted to different peer-to-peer networks and storage models, using a set of run-time parameters, or through the extension of some implementation module. We propose also a set of tools to automate and coordinate the execution of multiple simulations.

The paper is structured as follow. In Section 2 we present our contribution: the methodology and tool-chain to build and run simulations. In Section 3, we deal with the simulation meta-model. In Section 4, we analyze the model's performance in different scenarios. In Section 5, we discuss the related work. At the end, in Section 6, we draw the conclusion and the future work.

Simutools 2012, March 19-23, Desenzano del Garda, Italy
Copyright © 2012 ICST 978-1-936968-47-3
DOI 10.4108/icst.simutools.2012.247783

2. THE METHODOLOGY AND THE TOOL-CHAIN

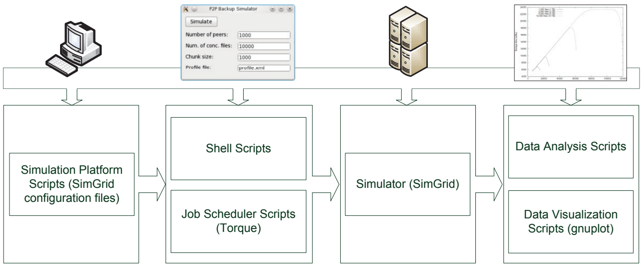


Figure 1: The tool-chain provides support for all the steps of the experiments: the design of the platform, the execution, the simulation and the data analysis.

The simulation analysis of complex systems requires the coordinated execution of several steps, as, among others, the choice or the implementation of a simulator, the modeling, the validation, the parameterized execution of simulation sets, and the data analysis. In this section, we describe the proposed tool-chain to analyze a peer-to-peer backup system. We implemented it using the SimGrid toolkit [2], and a combination of Perl and shell scripts. Figure 1 sketches the steps and the tools we propose:

1. The generation of the simulator's configuration files, using templates and the simulator's tools;

2. The choice of the simulation sets, and the submission for the execution using batch scheduler frameworks; this step is coordinated by a graphical user interface;

3. The simulation, using a simulator built on Simgrid framework;

4. The post-processing and the data analysis, using scripts for exportation to external software.

The automation of the configuration generation is not trivial, due to the number of possible combinations. However, in the case of peer-to-peer backup systems, we can identify some more interesting hardware configurations, and then we can parameterize it to be used with the chosen models.

SimGrid allows the simulation of very large models using a little amount of computational resources; however, huge simulations still require time. For this reason it is important to use an efficient system to run them. We implemented two tools for local and remote execution. Usually, the local execution is preferred when the simulation time is limited, and the users need to perform some interaction. To facilitate this step, we developed a simple graphical user interface (see Figure 1). It allows an easy choice of the parameters, to run quickly a set of simulations, adjusting manually the critical values between the executions. We implemented the current GUI prototype in Perl, but we planned a tighter integration in Eclipse [8], to improve the usability and the extensibility.

The graphical interfaces are a good choice when running on a machine with direct access, but often the best way to run heavy computations is through a batch system/job scheduler. In this way it can take full advantage of cluster and Grid infrastructures. Our current tool-chain interfaces Torque [1] to automatically schedule simulations on a computational cluster. A future implementation will directly use the Grids [7]. We built the batch components in a modular way, so the users can make small scale experiments on their own machine, before they move to larger scale ones with minimal changes.

For example, if users wants to study the influence of failure rates on a peer-to-peer backup system performance, they may be interested in running multiple simulations, each of which with an increasing percentage of failing hosts. The batch scripts can easily dispatch the simulations on the available machines. They are more useful when the analysis is multidimensional, when the user wants to explore the combined effects of the variation of two or three parameters. For example, as illustrated by the use case presented in section 4, users may want to study the failure of different number of hosts on systems using different data fragment sizes, ($r + s$ fragments of size z, with $z \in \{2\text{ MB}, 4\text{ MB} \ldots 256\text{ MB}\}$). In this case, it is convenient to run several different simulation groups, one for each possible z value, each set exploring the failure effects for that z.

As earlier mentioned, the actual simulator is built on SimGrid. SimGrid is a framework for the simulation of large-scale systems. The current version (3.6), offers four APIs [2]:

- *MSG*: a simulation interface for distributed programming;

- *GRAS*: an interface that allows to execute the simulation models as real applications (for example, substituting simulated socket connection with real ones);

- *SMPI*: for the simulation of MPI [18] applications;

- *SimDag*: for the scheduling of task graphs.

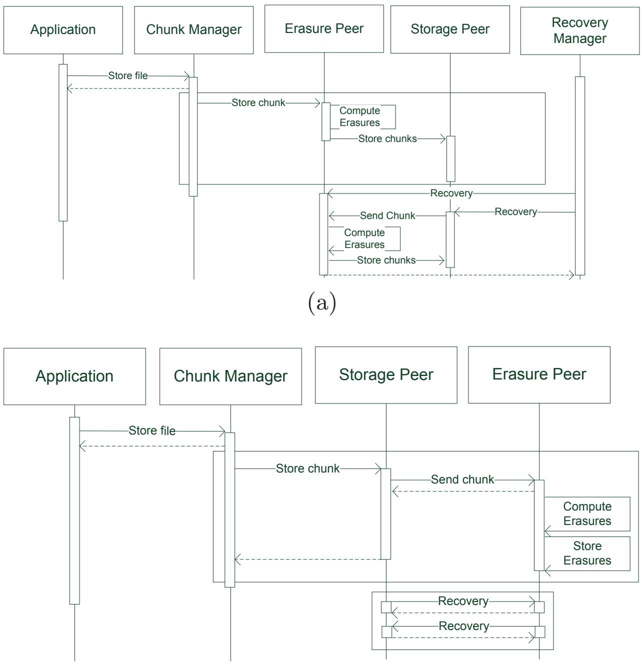
We used the MSG interface, to implement our generic model of a peer-to-peer backup system. It is a binary executable, integrated with the mentioned scripts, that simulates the storage of huge files on a peer-to-peer system. The executable is driven by configuration files and command line parameters and creates trace files with raw data and statistical results collected during the simulation. The executable modules follow the schema that will be discussed in the next section.

The last elements of the tool-chain convert the simulator's data into common format files, which are usable with graphical programs (e.g., Gnuplot's data files) or data analysis tools (e.g., *csv* text files).

3. THE SIMULATION MODEL

This section deals with the modeling of peer-to-peer storage systems.

Compared to the popular file sharing service, peer-to-peer reliable storage have many different characteristics. For example, in a reliable storage system, all data must be ensured the same (high) chances of surviving while in file sharing, only the most popular data will end-up surviving. Therefore backup systems require a closer cooperation of peers to reach better performance and ensure higher fault tolerance. The overlay network structure does not generally evolve synchronously with the service usage, because the join and the exit of the peers may not be decided by the final users. In addition, the system's usage may stress some operation more



(a)

(b)

Figure 2: Two alternate simulations model sequence diagram using the same tool-chain.

than others. For example, in the case of backup systems, the reads and the writes are asymmetrical. Indeed, the restore operation, which triggers the reads, is supposed to occur much less frequently than the backup operation that produces the writes. On the contrary, in most file sharing systems, the reads exceed significantly the writes.

The optimization and performance prediction of the systems studied in peer-to-peer backup scenarios are challenging: analytical methods often fail to capture important implementation details, while real experiments involving large scale distributed applications require a lot of preparations, and often fail to scale to more than a few hundred nodes. Hence, simulation offers a good trade-off, allowing to take into account a significant amount of details while still allowing to run (in silico) experiments involving up to several thousands of nodes. For this purpose, various models have been proposed (see Section 5). In this work, we implement a meta-model that can be modified to reflect the various architectures potentially build around the same elements, so that the simulation results can be compared.

The model we present splits the files to store in fragments in order to allow for the parallelization of the memorization on several peers. It relies on erasure codes to ensure the required fault tolerance level. Our model's schema has five actors:

- *The Chunk Managers* (CMs), that receive the files, split them in fragments of size L, and sends them over the network to other peers;
- *The Recovery Managers* (RMs), who are responsible to identify the faulty hosts, and assign reconstruction tasks to reconstruction peers;
- *The Storage Peers* (SPs), that receive the fragments and store them on disk. SPs also respond to RMs requests to send some of their stored fragments to reconstruction peers in charge of rebuilding lost redundancy blocks;
- *The Erasure Peers* (EPs), who receive the fragments from the SPs or CMs to compute the erasure codes (e.g., computing a linear combination of them);
- *The Overlay Network Module* (ONM), who implements the peer-to-peer policy.

The sequence diagram of Figure 2 show how such actors interact with each other. The CMs receive a set of n files of size d_i. We can hypothesize two types of sources for the files: an external source, or a local source that is located on the same peer. This latter case results in increased network usage and added perturbations. Then the CMs split each incoming file into s_i chunks. This operation allows for the parallelization of I/O, and makes them safer and faster. Concurrent dispatches, however, tend to saturate the network. As their third task, CMs send the fragments to the Storage Peers using the overlay network features [14]. Different overlay networks may chose different destination peers, which may lead to different performance. Our model allows for the substitution of the ONM in order to compare the possible performance gains. The SPs store the fragments and possibly send them to EPs, depending on the architecture chosen, using the ONM. The amount of redundancy and the number of peers involved depends on the coding schema used, which in turn impacts the performance of the system.

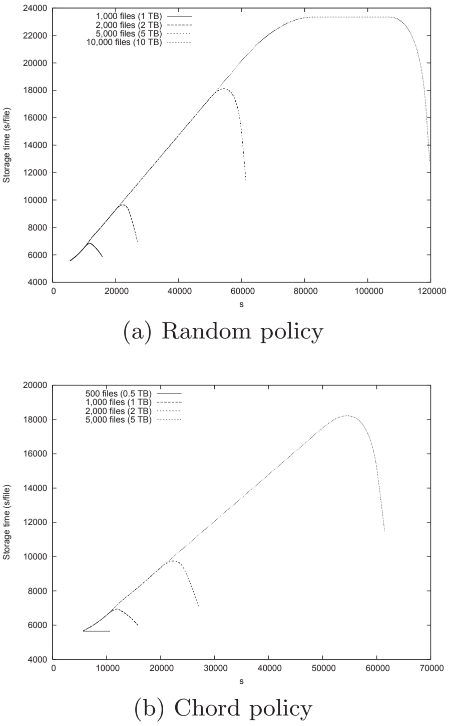
The dependability relies on the EPs. They collect remaining fragments from the surviving SPs (or in some cases initially from the CMs), compute their erasure codes, and store the rebuilt fragments on new SPs [5]. For this purpose, a monitoring process (possibly distributed) detects peer failures, decides when a redundancy level becomes critical and assigns recovery tasks to EPs. Several reconstruction policies exist to enable the recovery process; Dalle *et al.* report some example in [3].

We can easily change this model. The classic distributed computing one consists in assigning the exact same functions to each peer whereas a classic networking layered model could build a segmentation layer using CMs on top of a reliable storage service using RMs for mintoring and EPs for reconstruction, which could in turn be build on top of an unreliable storage layer built using SPs.

This work focuses on the network and on the peers' interaction more than on the physical data storage. For this reason, we used a trivial disk model where the storage time depends only on the disk's access time and data transfer: $t = t_a + sd_t^{-1}$. Several papers describe more accurate models, also making predictions on the failures (e.g., Strom *et al.* [20]). We plan to extend this study including an improved disk model.

4. THE VALIDATION

In this section we analyze the impact of the variation of some parameter, as the fragment size or failure rate, on the performance forecasted by our model, in order to validate it. The case study simulates a set of clients sending 1 GB files to the CS. On reception, they split every file in fragments and dispatch them to the storage sites. The SP are clustered in couples, in order to simulate the computation of a simple



(a) Random policy

(b) Chord policy

Figure 3: Storage time for increasing load.

$(3, 2)$ MDS (Maximum Distance Separable) code [5]. The notation (n, k) with $n > k$, means that k packets are encoded in n fragments: any set of k elements of the n, suffices to restore the original data. Therefore, in our example, every member of a couple of peers receives a fragment, stores it on the disk, and sends it concurrently to the erasure peer. This last one receives two chunks, simulates the computation of their linear combination, and then simulates the storage of the result in his local disk. Figure 4 illustrates the model we used in this case study.

In a first scenario, we analyze how the system responds to load, by sending an increasing amount of data for storage. In a second scenario we investigate the impact of the fragment size on the system performance. Finally, in a third one, we show how to study the effect of the failures using our meta-model. The experiments compare the performance when using two different overlay networks based on Chord [19] and a completely random placement strategy. The latter one is only used as a reference and has not practical interest. It should be noted that in this section we report the results using a flat network model that makes the validation easier: links are assumed to be equally shared and the maximum bandwidth is always fully available (no overhead or contention effect). This model ca easily be replaced by one of the more realistic models provided by SimGrid (see Section 2).

To evaluate the effect of the load, a new file storage is initiated every t seconds, which produces a piecewise linear increasing charge. As expected, the results show that as long as the resource usage stays below the available limit, the storage time is proportional to the amount of data to be stored. However, this analysis is important when dimensioning a peer-to-peer storage. This is particularly true in a data center, with a limited input bandwidth. In this case, input links are bottlenecks that limit the system performance, altogether with intrinsic hardware characteristics. Figure 3a shows the results of this experiment with a fixed fragment's size L of 256 MB, and 2,000 hosts, each of them hosting a chunk server, a storage peer, and an erasure peer. This analysis uses a random policy for the choice of the peers. In this case, a new client is started every 10 seconds and each client stores 1 GB of data. The simulation measures the storage time. In this experiment, we also assume that the data are produced localy by each client. In other words, we did not consider the perturbation of them on the network used by the peers. It may be the case, for instance, where the data are stored on a network file system, on different interconnect interfaces. Figure 3b shows the same experiment, but using Chord to chose the storage and erasure peers. A good estimate of the reported curves is crucial when dimensioning actual systems. It should also be noted that using real network topologies may lead to different results choosing a policy that can better exploit the locality principle, minimizing the switch hops.

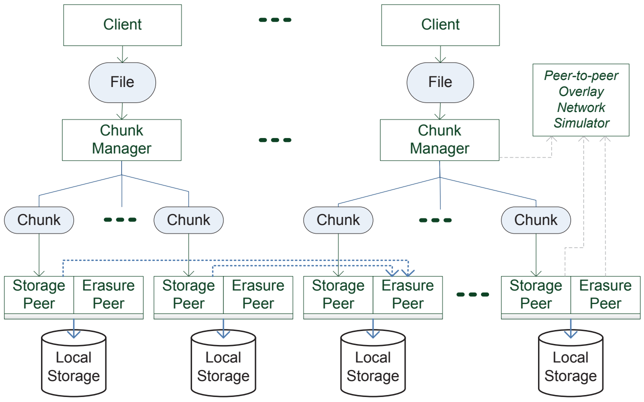
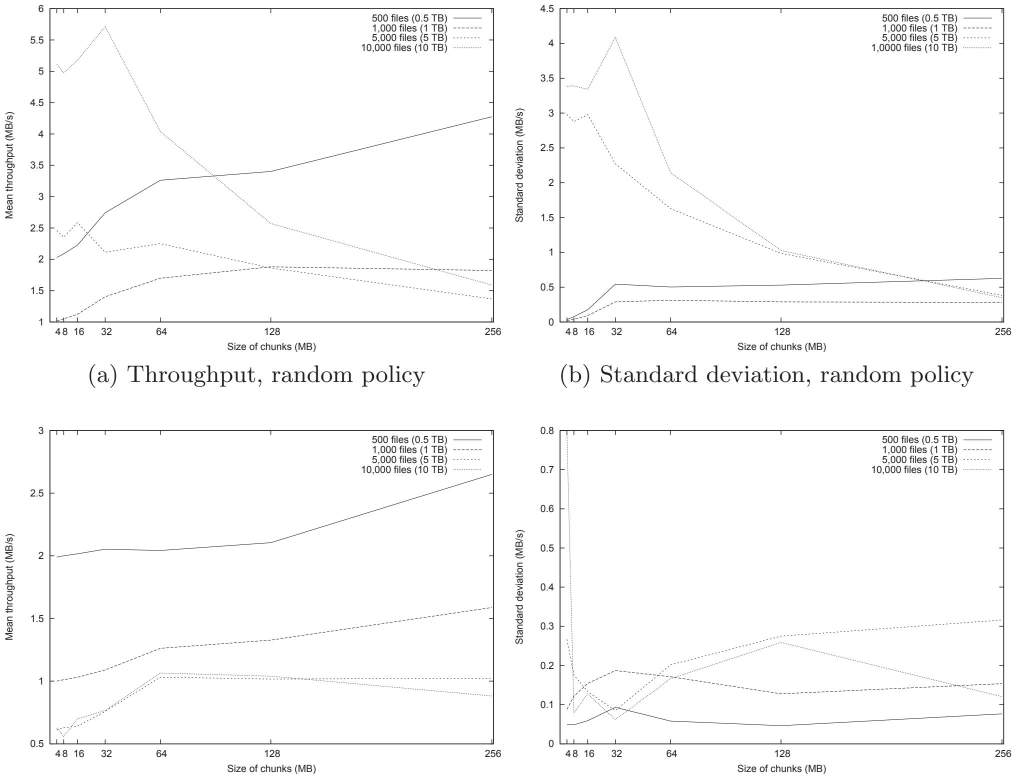


Figure 4: The simulation model's architecture.

Our simulations show that the influence of the size of the fragments on the performance is not obvious. A bigger or smaller size can affect in different ways the network, the CPUs (e.g., for the erasure codes computation) and the disks. Figure 5a shows the results of an experiment where we analyze the performance impact of the fragment size variations with $L \in \{4, 8, 16, 32, 64, 128, 256\}$, using a random host selection policy. Figure 5c replicates this experiment, using Chord to choice the peers. The measured throughput quantifies the fragment's memorization time, the actual storage time of the whole file is longer than the one computed using this throughput and the file size (See Figure 3). The peers send the chunks concurrently over the network, so, a smaller fragment's size increases the network traffic, but improves the memorization on local disks. Figure 5b presents the standard deviation, that evidence a wider dispersion of the measured sampled for the higher loads, but just for small fragment sizes. This effect is not present in the simulation using Chord (Figure 5d). The figures reveal two different trends when the system is under-loaded and overloaded, this may suggest a dynamic choice of the chunk size in function of the forecasted load.

The analysis of the failures impact is another critical point when studying the distributed storage systems. In this ex-



(a) Throughput, random policy

(b) Standard deviation, random policy

(c) Throughput, Chord policy

(d) Standard deviation, Chord policy

Figure 5: Throughput and standard deviation for different fragment sizes and files/peers ratiosusing a random and Chord policy for the choice of the peers.

periment, we ran the simulation asking to store a new 1 GB file every seconds. After some time we made unavailable a set of hosts. Figure 6a shows the results using a random policy for the peers' choice. At 500 s one third of the hosts crashes, then a recovery process restores the lost fragments on new hosts. We assume that we are able to recover all the fragments, but obviously, in a real case this is not true and will be a smaller load due to the recovery process. Figure 6b show the same analysis using Chord, to choice the storing and erasure peers. The analysis shows two trends in the storing performance. The set of files not involved in the crashes see a small increment to the storage time (it affects only the fraction not yet stored). The other files have a big discontinuity in their performance, due to the retransmission of lost data.

5. RELATED WORK

This section describes some interesting work related to the presented research. The reader can find some general information about the Peer to peer network in several surveys, as for example the one proposed by Lua *et al.*, in [14]. Other surveys describe the network codes, as Dimakis *et al.* did in [5]. In our work, we implemented the model using SimGrid [2], but many interesting simulators for peer to peer network have been proposed, as Naicken *et al.* state in [17], for example PeerSim [15].

In [3], Dalle *et al.* discuss about the failures impact on peer-to-peer storage systems. Here the authors propose some analytical models based on a fluid approximation, to assess the resource consumption and the probability of data losses when using erasure codes.

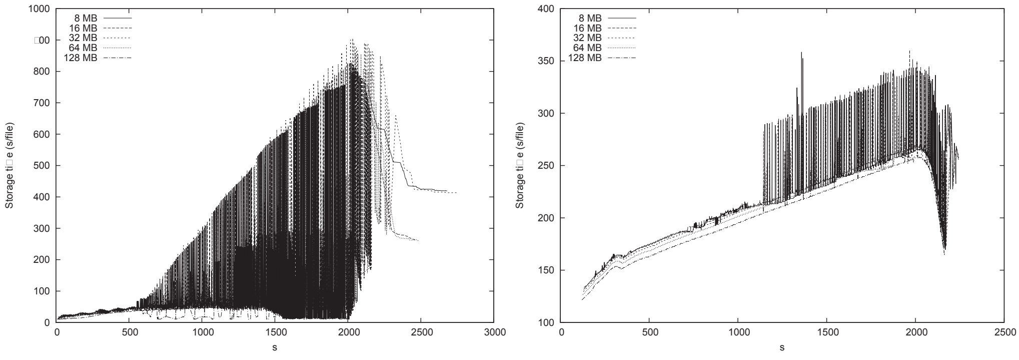
Many interesting papers presents models about peer-to-peer storage and backup systems. For example, Dandoush et al. in [4] describe a model for P2P storage systems that simulates in detail the real distribution of recovery processes. They implemented the model using the NS-2 network simulator [11]. The goal of our paper, more than describe a new P2P model, is to provide the tools and some reference result, to make quicker the creation of new models and easier the result comparison. In that way, researchers can implement only the parts of their interest and measure immediately how they differ from other models.

To assure a good level of fault tolerance the storage systems use redundancy. Unfortunately the erasure codes require an amount of resources (storage, bandwidth, and computational) proportional to the required reliability. While in our experiments, we explored just the linear erasure code, more sophisticate techniques exist, as the one proposed by Duminuco and Biersack in [6].

6. CONCLUSION

In this paper, we presented a preliminary version of an integrated tool-chain for the simulation peer-to-peer backup systems, a meta-model and their application to the performance evaluation of various scenarios.

We plan to improve the presented research in several directions. While the implemented tools are functional, a future integration in an environment like Eclipse [8] could greatly



(a) Random policy

(b) Chord policy

Figure 6: Storage time in case of disasters: after 500 s, one third of the nodes crashes, and at 1,000 s the recovery process starts, restoring the lost fragments on peers chosen using a random or a Chord policy.

improve them, enforcing the usability and the robustness. Most of the model's modules can benefit from a smarter design, for example, a better disk model, taking in account more parameters, and describing a non-linear behavior. We plan also to analyze some common host configuration, involving, for example, multi-level switches, and exploring the effect of locality and the network heterogeneity.

7. ACKNOWLEDGMENTS

This work is partly funded by the French Agence Nationale de la Recherche (ANR), in the USS-SimGrid project [10], and by the INRIA Mascotte project.

8. REFERENCES

[1] Adaptive Computing. *TORQUE Admin. Guide*, 2011.

[2] H. Casanova, A. Legrand, and M. Quinson. SimGrid: A generic framework for large-scale distributed experiments. In *Proc. of Int. Conf. on Computer Modeling and Simulation*, pages 126–131, Los Alamitos, CA, USA, 2008. IEEE Comp. Soc.

[3] O. Dalle, F. Giroire, J. Monteiro, and S. Perennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *IEEE Int. Conf. on Peer-to-Peer Comp. (P2P '09)*, pages 184 –193, 2009.

[4] A. Dandoush, S. Alouf, and P. Nain. A realistic simulation model for peer-to-peer storage systems. In *Proc. of the 4th Int. ICST Conf. on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 7:1–7:11, Brussels, Belgium, 2009. ICST.

[5] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *CoRR*, abs/1004.4438, 2010.

[6] A. Duminuco and E. Biersack. Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems. In *Proc. of Int. Conf. on Peer-to-Peer Comp. (P2P '08)*, pages 89–98, 2008.

[7] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In H. Jin, D. Reed, and W. Jiang, editors, *Network and Parallel Computing*, volume 3779 of *LNCS*, pages 2–13. Springer, 2005.

[8] S. Holzner. *Eclipse*. O'Reilly Media, 2004.

[9] InfiniBand Trade Association. InfiniBand Trade Association. Web Site, 2011. `http://www.infinibandta.org`.

[10] INRIA. USS-SimGrid project. Web Site, 2011. `http://uss-simgrid.gforge.inria.fr`.

[11] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer, 2008.

[12] S. Krishnan. *Programming Windows Azure: Programming the Microsoft Cloud*. O'Reilly, 2010.

[13] J. Kubiatowicz et al. Oceanstore: an architecture for global-scale persistent storage. In *Proc. of the 9th Int. Conf. on Architectural Support for Prog. Languages and Operating Systems*, ASPLOS-IX, pages 190–201, New York, NY, USA, 2000. ACM.

[14] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[15] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of IEEE Int. Conf. on Peer-to-Peer Comp. (P2P '09)*, pages 99–100, 2009.

[16] J. Murty. *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly, 2008.

[17] S. Naicken et al. Towards Yet Another Peer-to-Peer Simulator. In *Proc. of Int. Working Conf. on Performance Modelling and Evaluation of Heterog. Net. (HET-NETs)*, Ilkley, UK, 2006.

[18] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the Conf. on Applications, Technologies, Architectures, and Protocols for Computer Comm.*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[20] B. Strom, S. Lee, G. Tyndall, and A. Khurshudov. Hard disk drive reliability modeling and failure prediction. *IEEE Transactions on Magnetics*, 43(9):3676 –3684, Sep. 2007.