SPLAD: scattering and placing data replicas to enhance long-term durability

Véronique Simon, Sébastien Monnet, Matthieu Feuillet, Philippe Robert, Pierre Sens

SPLAD: scattering and placing data replicas to enhance long-term durability

Véronique Simon , Sébastien Monnet, Mathieu Feuillet, Philippe Robert, Pierre Sens

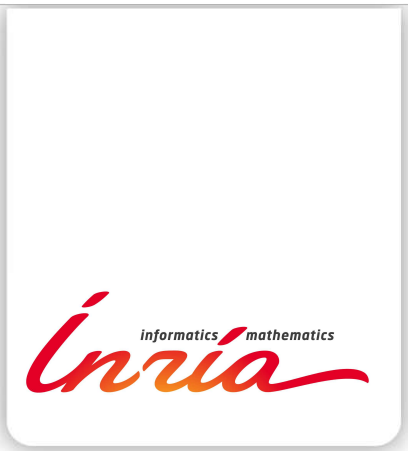
RESEARCH REPORT

N° 8533

May 2014

Project-Teams RAP and REGAL

ISSN 0249-6399 ISRN INRIA/RR--8533--FR+ENG



SPLAD: scattering and placing data replicas to enhance long-term durability

Véronique Simon[*] [†] [‡], Sébastien Monnet[*†‡], Mathieu Feuillet[§], Philippe Robert[‡], Pierre Sens [*†‡]

Project-Teams RAP and REGAL

Research Report n° 8533 — May 2014 — 20 pages

Abstract: Distributed storage systems have to ensure data availability and durability despite the occurrence of failures. To do so, many of them rely on replication mechanisms: for each piece of data, several replicas are stored. We show that the layout of the data block copies on the nodes, chiefly the way the copies are scattered, has a major impact on the reparation speed and thus on the data loss ratio. In this paper, we propose SLPAD, an approach that provides the ability: (i) to finely tune the proportion of common content stored by the nodes; and (ii) to control the storage load distribution while creating new data block copies. We propose a simulation model that allows us to present a long-term study of the impact of the data block copies layout and the system load on the data loss ratio.

Key-words: distributed data storage; replication; data durability; data placement; load distribution; backup

[*] Sorbonne Universités, UPMC Univ Paris 06, F-75005, Paris, France
[†] CNRS, UMR_7606, LIP6, F-75005, Paris, France
[‡] Inria, Paris—Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France
[§] ANSSI, 51 boulevard de la Tour-Maubourg, 75700 Paris 07 SP, France

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

SPLAD: dispersion et placement des copies des données répliquées pour l'amélioration de la durabilité

Résumé : Les systèmes de stockage distribués doivent assurer la disponibilité des données et leur durabilité malgré l'occurrence de défaillances. Pour ce faire, beaucoup d'entre eux utilisent des mécanismes de réplication: pour chaque donnée, plusieurs copies sont stockées. Nous montrons que la disposition des copies des données sur les nœuds, surtout la façon dont elles sont dispersées, a un impact majeur sur la vitesse de réparation et donc sur le taux de perte. Dans ce papier, nous proposons SLPAD, une approche qui offre la possibilité: (i) de régler finement la proportion de contenu commun stocké par les nœuds; et (ii) de contrôler la répartition de la charge de stockage lors de la création nouvelles copies. Nous proposons un modèle de simulation qui nous permet de présenter une étude à long terme de l'impact de la disposition des copies des données et de la charge du système sur le taux de perte.

Mots-clés : stockage de données distribué; réplication; durabilité des données; placement de données; distribution de charge; sauvegarde

1 Introduction

Distributed storage systems such as Hadoop Distributed File System (HDFS) [1], Google File System (GFS) [2], and Windows Azure [3] ensure data availability and durability using replication. Persistence is achieved by replicating the same data block on several nodes, and ensuring that a minimum number of copies are available on the system at any time. Thus, whenever the contents of a node are lost, for instance due to a hard disk crash or a destructive operating system reinstall, the system regenerates the data blocks stored before the failure by transferring them from the remaining replicas.

There are many research efforts on data replication mechanisms. However, they generally focus on how to maintain data copies consistency; on the number of copies needed to serve popular contents; on the placement of copies to reduce access latency; or on the data availability regarding the connectivity on the nodes. Furthermore, most of the time, the configurations used to evaluate the new approaches are not well sized: the mean time between failures (MTBF) is usually very small according to the amount of stored data and the network speed. The goal is to have an under-sized system in which data losses (i.e., pieces of data that can not be retrieved nor repaired) occur. This provides the ability to compare different systems or configurations over a short period of time. Research work studying the long-term behavior of data storage systems is usually theoretical, generally based on mathematical models [4, 5]. In this paper, we study the impact of the data block copies layout on the storing nodes on the long-term data durability. In order to study this impact, for sake of simplicity, we do neither take into account data updates (and thus consistency concerns), nor user data accesses. We propose SPLAD that provides the ability to tune how the data block copies are scattered and placed among the storage nodes. To do so, we rely on a discrete event simulation engine. We propose a simulation model that is: (i) simple enough to be able to simulate the behavior of hundreds of nodes during years within several hours; (ii) accurate enough to take into account the precise data distribution among nodes and the behavior of the maintenance protocol; and (iii) highly configurable in order to finely tune data block copies placement parameters and being able to experiment various configurations.

We claim that the distribution of the data block copies among nodes is a key criterion. With a same failure pattern, the more the failure healing process is fast, the less data will be lost. A node stores many different data block copies, in case of failure each of them has to be recreated using a node storing another copy of the same data block. As the available bandwidth is fixed, our approach consists in increasing data transfer parallelization by shuffling nodes content. This allows the maintenance system to use both more sources and more destinations to repair a failure.

In this study, we use the RelaxDHT [6] approach to maintain metadata information. RelaxDHT provides a mean to locate and maintain data block copies.

The main contributions of this paper are:

- SPLAD, an approach that provides the ability to finely tune the way data block copies are mapped to the storage nodes;
- a simulation model that permits to accurately study the long-term behavior of a distributed data storage system under various different configurations;
- a long-term study (over two simulated years) of the system behavior under failures. We focus on the impact of the load of the system, and of the data distribution layout among the nodes, over the data loss ratio.

The following of this paper is organized as follows. Section 2 describes the related work. Section 3 describes our model, explains how our approach provides the ability to tune the way

the data copies are scattered among the nodes. Then Section 4 presents our evaluation parameters and metrics before analyzing the simulation results. Finally, Section 5 concludes the paper.

2 Related work

Data replication in large-scale distributed data storage systems has been widely studied. A large part of the literature focuses on consistency models and on how to maintain consistency of replicated data. In recent work, Scatter is noticeable [7]. However, in our work, we do not take into account mutable pieces of data. We consider immutable data blocks; therefore we do not consider data consistency problems.

Another category of research work tackles the problem of user access latency [8]. Data copies should be placed close to the users that access them to offer performant access. Furthermore, some pieces of data will never be accessed while others have a high popularity, this should be taken into account to set the replication factor: popular data may need many sources to be served properly [9]. The content delivery networks (CDN) such as Akamai [10] that are used to speed up data access highlight concern. In this paper, we focus on data durability; we do not take user access into account.

If we put aside replication mechanisms that place copies according to nodes online predictions [11] (trying to place copies on nodes that are not up at the same time to provide service continuity for instance), in the context of large scale peer-to-peer data storage, there are two main basic replica placement strategies, contiguous (e.g., leafset-based) and fully scattered (e.g., multiple-key-based).

Using contiguous placement strategies, the data block copies are stored by contiguous nodes (nodes are given identifiers which are used to sort them). The peer-to-peer distributed hash tables (DHT) such as DHash [12] and PAST [13] use this technique. This data placement strategy provides an efficient implicit data localization, even in case of failure, the data, stored by immediate neighbors is easy to locate. However, it induces useless data movements to maintain the placement invariant: in case a new node arrival, many pieces of data have to be move to respect the contiguous placement invariant. Furthermore, using this strategy, node contents are very similar, which may impact the reliability, as shown in the evaluation section.

Using a fully scattered placement strategy, like multiple key replication which relies on computing k different storage keys corresponding to different storage nodes for each data block (where k is the replication factor), the data block copies are shuffled among all the system nodes. This solution has been implemented by CAN [14] and Tapestry [15]. Google file system (GFS) [2] uses a variant based on random placement to improve data repair performance. *Path* and *symmetric replication* are variants of multiple-key-based replication [16, 17]. This replication method has the drawback that its maintenance protocol is costly and has to be done on a "per data block" basis, which means that the number of messages linearly grows with the number of data blocks.

Lian *et al.* propose a hybrid stripe replication scheme where small objects are grouped in blocks and then randomly placed [18]. Using an analytical framework, they show that their scheme achieves on near-optimal reliability.

Contiguous and fully scattered placement policies are two extremities, in our work we propose to experiment and evaluate intermediate approaches (from contiguous to fully scattered).

Finally, several studies have focused on the placement strategies based on availability of nodes. Van Renesse [19] proposes a replica placement algorithm on DHT by considering the reliability of nodes and placing copies on nodes until the desired availability is achieved. To this end, he proposes to track the reliability of each node such that each node knows the reliability information about each peer. In FARSITE [20], dynamic placement strategies improve the

availability of files. Files are swapped between servers according to the current availability of these latter. With theses approaches, the number of copies can be reduced. However, such approaches may lead to a high unbalanced distribution whereby highly available nodes contain most of the replicas and can become overloaded.

Cidon *et al.* [21] propose the concept of copysets in which the copies of a data will be stored. They aim at reducing the frequency of data loss events (not necessarily the actual number of data losses), while correlated failures occur. Even if this work is different from ours, it also mentions the benefit of scattering data to enhance reparation speed.

In this paper, we do not propose a new replication strategy. Based on a previous work [6] we propose a mechanism that provides the ability to vary the way the data copies are placed among the nodes. We provide a study of the impact of the data placement on the data loss ratio, on a long term.

3 SPLAD overview

This section details our system model and explains how we set the way the data block copies are scattered among the nodes.

3.1 System model

Our model is simple. Our goal is to study the impact of the data copies placement on the data-loss ratio. A complex system may bias the analysis, or makes it difficult. Furthermore, we aim to study the long-term durability; a simple model offers the possibility to have fast simulations, allowing us to simulate the system's behaviour among years.

The system is composed of n nodes storing m data blocks replicated k times[1]. Therefore, there are $m * k$ data block copies spread among the n nodes.

We do not take into account new data blocks: all the data is added at the initialization of the system, and then we observe its behavior in presence of failures. We consider that the nodes are homogeneous; they have the same network characteristics both in terms of latency and bandwidth. We study both the symmetric and the asymmetric bandwidth cases.

Each node is assigned a unique identifier. We consider that the nodes are organized according to their unique identifiers to form a logical ring like in the case of the ring-based distributed hash tables (DHT, like Past [13] or DHash [12]). A routing layer, able to route a message to the node having a given identifier is assumed. However, the results we present in this paper are independent on the way nodes or data blocks are being indexed/localized. We do not require the use of a DHT-like key-based routing layer. The only requirement we impose is to have a set of connected nodes, able to locate, and communicate with any other node in the set. This can also be achieved in a datacenter by using a global index/monitor service, for instance.

The failure model is also simple: node failures follow a Poisson distribution. Each node has the same probability to fail. Failures are crash (fail silent model): a node that fails will never come back; the data block copies it uses to store are lost. Furthermore, a new empty node having a random identifier replaces each faulty node immediately. Thus, a new node arrives at a random position in the logical ring, independently from the faulty node position. The system thus has a constant number of nodes. However, as the new node arrives empty, a failure implies the loss of the data block copies stored by the faulty node.

[1] We do not focus on the variation of the replication factor, we use 3 in our evaluations, which is a standard replication factor used in Hadoop or Amazon.

We also consider that we have a perfect failure detector. That means that a node is able to know if any other node is up and running or not. However, a false positive will only result in the creation of extra copies while a false negative will just differ the failure reparation. In practice, the period of the failure detection mechanism (which can be performed by the routing layer of a peer-to-peer DHT, a heartbeat based distributed failure detector like [22] or a centralized monitoring service for instance) is much shorter than the storage maintenance period (by orders of magnitude).

3.2 Metadata management

SPLAD replication mechanism metadata management is based on our previous work, RelaxDHT [6]. In RelaxDHT, a data block is assigned a unique identifier in the same namespace as the nodes unique identifiers.

The node having the closest identifier is called the `root` of the data block[2]. It is responsible for the replication of this particular data block: the `root` periodically checks that there exist k copies of the data block in the system. When it is not the case, it has to choose a node to store a new copy of the data block. The nodes storing a copy of a data block are called `storers` for this data block. For a particular piece of data, both the `root` and the k `storers` maintain the associated metadata which consist in: (i) the data identifier; (ii) the identifier of the `root` node responsible for this piece of data; and (iii) the identifiers of all the `storer` nodes hosting a copy of the data. As there are more data blocks than nodes in the system, each node is both `root` for many data blocks, and stores many data block copies.

In RelaxDHT, the replication mechanism relies on the notion of leafset: a `root` node chooses `storer` nodes within its fixed-size leafset. The leafset of a node is the small and fixed set of direct neighbor nodes in the DHT ring. RelaxDHT is based on Pastry [23], in which the leafset is maintained by the routing layer, it is used to route messages to the last hop. Each RelaxDHT node periodically and locally (using the local leafset information) checks whether the `storers` hosting copies of the data for which it is `root` are still alive or not. It also checks if the `root` nodes of the data for which it is a `storer` are still alive. In case of a `storer` failure for a particular piece of data, the `root` node has to choose a new node in the leafset to store the lost data copy. In case of a `root` failure, a `storer` node, has to compute which node is the new `root` (i.e., the node having the closest identifier, by definition). A message is then sent to all of its neighbors within its leafset, either to tell them to keep storing the data for which it is `root`[3] or to ask them to store a new piece of data in the case they have been chosen to replace a faulty `storer`.

Thanks to the use of a local knowledge, each node sends only *one* message to all the other nodes in their leafset each maintenance period. Therefore, whatever the number of blocks to store, the number of messages will remain constant.

SPLAD implementation is based on the RelaxDHT metadata management mechanism. RelaxDHT provides a distributed solution to index data block copies. However, depending on the targeted scale and architecture, this mechanism could be replaced by a centralized service that monitors `storer` nodes and places new replicas.

3.3 Replication mechanism and transfer parallelization

When a failure occurs, data block copies are lost. The system should be healed by recreating the missing copies while other copies still remain in the system. Otherwise, the data blocks could

[2]This is usual in the context of peer-to-peer DHTs.

[3]RelaxDHT storage was based on the notion of lease to allow data deletion, it is not the case anymore in SPLAD.

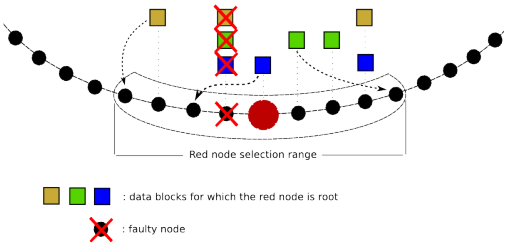


Figure 1: Red node is root for the blue, green and gold data blocks. We only consider data block copies of these blocks. Replica are represented just above the nodes that store them. When a failure occurs, the red node can choose among its selection range nodes to store a new copy of each lost replica of a data for which it is root.

be lost for ever. As each node stores many data blocks, many data block copies are lost while a node failure occurs. The remaining data block copies, that can be used to recreate the lost ones, are hosted by a subset of the remaining nodes. The larger is this subset, the more sources can be used to regenerate the lost data block copies. To study this impact, we propose an approach based on a *selection range* with a tunable *width*.

We keep using the notion of `root` node: each data block is assigned a `root` node to store it. The `root` node of a piece of data can choose a `storer` among a subset of nodes composed by its closest neighbors clockwise and counterclockwise, as illustrated by Figure 1. This subset, centered by the `root`, is called the *selection range* for the `root` node. Each node has its own selection range, centered by itself. All the node's selection ranges have the same width (i.e., the selection range width is a system parameter).

Varying the size of the selection range allows the system to tune the amount of data that two nodes have in common and thus to impact the number of available sources to repair a failure. It provides the ability to finely tune how much the data block copies are scattered in the system. Obviously, the larger the section range, the bigger is the number of available sources, but also, the bigger is the number of possible destinations (destinations have to be chosen within the selection range, by definition). In our evaluations, we study the impact of the selection range size on the transfer parallelization while repairing failures and thus, on the data loss ratio.

Each node having its own selection range, like a sliding window among nodes, a node may store data for a `root` located at the borders of its selection range. There is a non-null probability that this root has chosen its other selection range extremity to store another copy of the same data block. Therefore a node can share some common pieces of data with nodes that are not in its selection range, but it may have common data with approximately[4] two times the selection range width, as illustrated by Figure 2.

However, it is necessary to keep in mind that the size of the selection range may impact the maintenance overhead: each node needs to maintain an up-to-date state of all the nodes in its selection range. This overhead depends on the system size and on the kind of architecture that is used (peer-to-peer system, data center, etc.). It may be interesting to tune the selection range

[4]Even if a `storer` is originally chosen within the `root`'s selection range, while node insertions occur, it is possible that it leaves the selection range, however, in practice, nodes do not roll away to far from there original position.

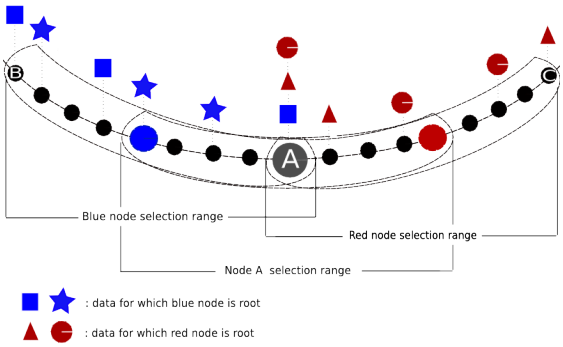


Figure 2: The copies of the data blocks stored by a node can be spread among $2*|selectionrange|$ nodes. Here, we only represent four data blocks, replicated three times, for which either the red or the blue node is root. Node A is in both red and blue node selection ranges, it stores common data with nodes B and C located outside its selection range (the blue square and the red triangle). In case of node A's failure, nodes B and C can be involved in the reparation.

size according to the effective benefit/cost. Our evaluations may help for this task.

3.4 Target node choice

The selection range size is not the only placement criterion that can be tuned in our approach. For each data block copy lost while a failure occurs, its `root` node has to choose another node in its selection range to store a new copy. Any node that does not already store a copy of this particular data block can be chosen. This choice can be done following different policies. We will show that the impact of the chosen policy has an influence both on the data loss ratio, but also on the data load distribution among `storers`.

We study three different policies: random, less charged and power of choice.

3.4.1 Random

This is the simplest policy. The root node choses a new storer randomly (using a uniform distribution) among its selection range nodes that do not already store a copy of the data block. This solution is easy to implement, it does not require any extra knowledge on the nodes, like their current storage load. It provides the ability to uniformly use all the possible nodes in the selection range, which gives a high number of transfer destinations. However, this strategy induces a huge imbalance in the node storage load distribution: old nodes may store many data block copies. A node never removes a copy unless it fails. Indeed, while using the random strategy, it will always be candidate for new data block copies. Thus the older a node is, the more data blocks it is expected to store.

3.4.2 Less charged

To counter the imbalanced load distribution effect of the random strategy, a solution consists in choosing the less charged node among the selection range nodes that do not already store a copy. This second policy is harder to implement: it requires maintaining a relatively up-to-date

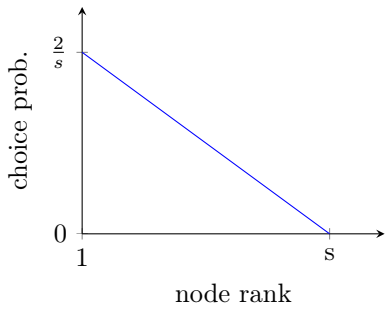


Figure 3: Probability for a node to be chosen according to its load in the power of choice policy

knowledge of the storage load of all the nodes in the selection range. It provides a good load distribution among all the system's nodes. Our implementation is optimized: the data blocks being transferred are taken into account in the choice. As we use a simulator, we can do this easily, in practice it can be done with a central coordinator for small-scale systems. However, we show in our evaluations that this strategy may have a bad side effect impacting the data loss ratio when the selection range is wide.

3.4.3 Power of choice

The last policy we study is simple and has been studied for years. Mitzenmacher*te al.* [24] presents a detailed study of many applications of the "power of two random choices paradigm" (that we simply call power of choice here). Using this strategy, the `root` of a data block picks randomly (using a uniform distribution) two nodes that do not already store a copy of the data block in the selection range, it then chooses the less loaded of them (breaking ties arbitrarily). This does not necessarily require maintaining information for all the selection range nodes; the `root` can probe the two chosen nodes when required. A good property is that this policy provides the ability to choose a node with a probability that decreases with its storage load.

Here is an explanation of the probability for a node to be chosen according to its current storage load. Considering the s selection range nodes that do not already store a copy of the data block to repair, we assign an increasing rank to the nodes, growing with the amount of blocks they store. Thus, the less charged node is assigned rank 1 and the most charged one rank s (to simplify, we do not consider ties).

The probability p_k that a node of with rank k is chosen can be computed as follows. There are $\frac{s(s+1)}{2}$ possibilities to build distinct pairs of nodes. There are $s-k$ pairs of nodes in which the rank k node is the less charged of the couple. We deduce that $p_k = \frac{2(s-k)}{s(s-1)}$.

In particular $p_1 = \frac{2}{s}$ and $p_s = 0$. The probability to be chosen decreases linearly with the rank, as illustrated by Figure 3.

3.5 Simulation model

Our simulator is based on the PeerSim simulation engine [25]. To be able to simulate long-term executions, we have designed a simple simulation model. For instance, we do not simulate the routing layer; each node is able to communicate with any other one in the system. Furthermore, the simulator simulates only the maintenance; it does neither take into account the addition of new pieces of data to the system, nor the access to existing data blocks by users. We consider the

homogeneous set of n nodes storing m replicated pieces of data of a fixed size, having the same network and storage capacity, and the same probability to crash following a Poisson distribution.

We have paid a particular attention to the bandwidth management. In case of the occurrence of a failure, many data blocks have to be transferred among a subset of nodes, it is thus important to take into account bandwidth limitation and network contention. In our simulator, each node manages two waiting queues: one for its outgoing link and one for its incoming link. Each link has latency and a bandwidth. While a packet transmission is done, it is placed in the incoming waiting queue of the destination node. This roughly emulates a star network with a central virtual node storing packets before forwarding them to the destination. This allows our simulator to efficiently simulate the network while taking into account bandwidth limitations.

4 Evaluations

The goal of our evaluations is to study the impact of (i) the global storage load, and (ii) the placement of the data copies upon (a) the data loss ratio, and (b) storage load distribution, on a long term. All the results presented in this section have been obtained using the discrete event simulator presented in Section 3 based on the PeerSim simulation engine.

For all the simulations, we fix the number of nodes (n) to 200, the data block size to $10MB$, the replication factor (k) to 3, the mean time between failure (MTBF) of a node to 7 *days*. We simulate two different network models, one with symmetric and one asymmetric bandwidths. In the first case, the downlink is set to $10Mb$ and the uplink to $1Mb$, in the later, both up and down links are set to $5.5Mb$. In both cases, the latency is constant, set to $0.1s$[5].

The parameters that are going to vary are:

- *the global system storage load* trough the number of data blocks to store;
- *the data block copies placement* through the variable size of the selection range and the policy used for the choice of a target node within the selection range.

For each set of parameters we run 210 experiments.

4.1 Impact of the load

The first thing we have done is to size our system. The load, in terms of number of blocks to be stored, has an important impact on the system's behavior. We vary the number of data blocks to store from 2 000 to 20 000 (so that the size varies from $20GB$ to $200GB$) replicated 3 times and spread among the 200 nodes. For this first set of experiments, the power of choice strategy is used and the bandwidth is asymmetric. Figure 4 shows the results we obtain using several different selection ranges.

Obviously, we can observe that the more the system is loaded, the bigger is the data loss ratio. However, the loss ratio increase is not linear.

While the system is well sized/not overloaded, the loss ratio is low. It first increases slowly until the system becomes overloaded, then increasing the load has a great impact on the data loss ratio. There is no real break in the figure, the point where the system can be considered has overloaded depends on the selection range width, it also depends on the "accepted" loss risk. Theoretically the risk to lose data is never null in presence of failures.

Notice that even in our case, the nodes MTBF is relatively high: 7 *days*. That means that the whole system MTBF is about 50 *minutes* (a failure occurs in the system every 50 *minutes*

[5]To have a fast simulator, the time granularity we use is $0.1s$, it is thus the smallest non-null latency we can set.

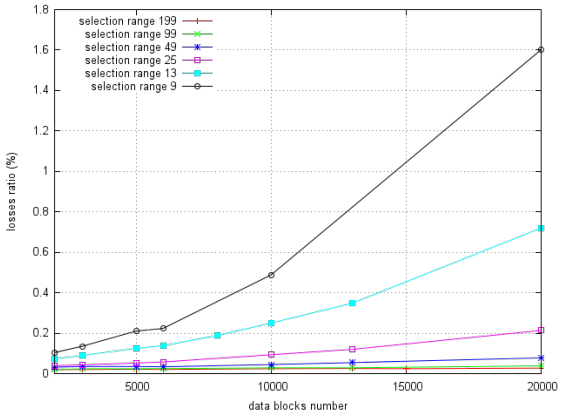


Figure 4: Impact of the load on the loss ratio.

on average). Furthermore, as new nodes come empty, that means that we only consider crash failures. This is equivalent to either considering hard drives with a 7 *days* MTBF, or considering an in-memory only storage using nodes having a 7 *days* MTBF.

On Figure 4, we can also observe that the impact of the load on the loss ratio increases while the selection range width decreases. The load amplifies the differences between the placement strategies (the curves diverge while the storage load increases). For this reason, many research works on replication strategies evaluate their approach using an under-sized configuration, either by increasing the storage load, or by increasing the failure rate. It provides the ability to have the occurrence of many failures in a relatively short period of time, which permits to compare the performance of different approaches. In this paper, even if we study the system's behavior over a long period of time (2 years), we use a slightly under-sized configuration for the same reasons.

In the following of this section, all the results we present are obtained with a storage load of 10 000 data blocks replicated 3 times, which seems reasonable for a wide variety of selection ranges considering our configuration parameters.

4.2 Impact of the selection range width

One of the main advantages of our approach is its ability to tune how much common data blocks two nodes will share by varying the selection range width. Figure 5 presents how the data loss evolves while varying the selection range width, using the three different strategies. The selection range width has a huge impact on the data loss ratio. If the selection range is really small, e.g., 9 in our simulations, tens of data losses occur. This is due to the fact that there are few sources that can be used to re-create the lost data block copies.

In the case of a small selection range the number of possible destinations for the transfers is also limited. This limits the possibility to parallelize data transfers, and thus increases the healing time, leading to a greater risk of loosing data blocks.

However, placing 3 copies within a 9 nodes selection range is still more efficient than using a contiguous data placement (which is close to use a 3 nodes selection range) as it is usually the case in some reference peer-to-peer DHTs implementations like PAST [13] or DHash [12]. Growing the selection range from 9 nodes to 25 nodes brings a significant gain: the number of losses is divided by more than 4.

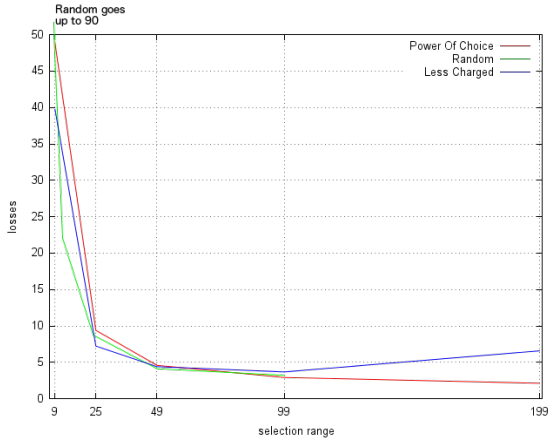


Figure 5: Impact of the selection range width on the loss ratio using an asymmetric bandwidth network.

If we keep growing the selection range, from 25 nodes to 49, the gain is smaller, but it is still significant: the number of losses can be divided by 2 with the random and power of choice strategies (a detailed comparison of the target node choice strategies is given below).

Then, while continuing to grow the selection range, given our system parameters, the benefits are more limited, except in the case of the *random* strategy for which significant benefits can still be observe until 99 nodes selection ranges. For the *less charged* strategy, getting a too large selection range is even worse (the loss ratio increases while the selection range width varies from 99 to 199). This particular case in explained in Section 4.3.

These results confirm that, in the general case, a larger selection range allows the system to scatter the data among more nodes which provides the ability to parallelize more the transfer during failure healing, which in turn leads to fewer data losses. However, we learned from our experiments that the gain does not grow linearly with the size of the selection range. It is important to notice that maintaining larger selection ranges may lead to a bigger overhead, as nodes have to maintain an up-to-date knowledge of the nodes in their selection ranges. Therefore, depending on the system parameters (number of nodes, latencies and bandwidths, etc.) the selection range should be set to a good tradeoff.

For instance, in our system, 49 seems to be a reasonable selection range width.

4.3 Impact of the choice of a new storer node

The data loss ratio depends on the system's load in terms of number of data blocks and on the selection range's width, but as Figure 5 shows, the strategy used by `root` nodes to choose a `storer` (within its selection range) also has an impact.

This impact is really significant, in terms of data losses, for small selection ranges. For a selection range of 9, the *random* strategy induces more than twice the number of data losses than the *less charged* one. For selection ranges wider than 49, the impact of the placement strategy on the data loss is still visible but less significant.

The main difference between these three strategies is the distribution of the storage load among the nodes they induce.

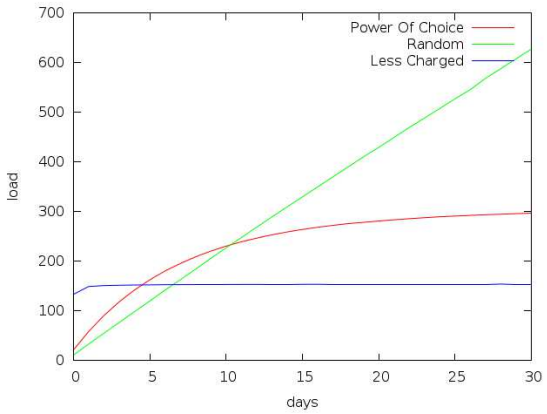


Figure 6: Storage load evolution using a 99 nodes selection range.

While using the random strategy, a node that is in the system for a long time has more chance to store many pieces of data: a copy of a data block is never removed from a node (except in case of failure), however, using the random strategy, all the nodes have the same probability to store new data block copies. When an old node, storing a large number of data block copies crashes, the time to rebuild all the lost copies can be long. It will even be longer if the selection range is short, due to the poor transfer parallelization.

Figures 6 and 7 show the evolution of the mean load of a node depending of its age in days. Using the random strategy, the load increases linearly until the node's death.

The *less charged* strategy solves the load distribution problem. By systematically choosing the less loaded node within the selection range to store a data block copy, the storage load tends to be uniformly distributed among nodes.

However, this solution presents two main drawbacks. Firstly, it requires that the nodes maintain an up-to-date knowledge of the load of all the nodes within their selection range. This can bring a big overhead; chiefly to enable an optimized *less charged* strategy, taking into account ongoing transfers (like it is the case in our implementation). Secondly, this strategy breaks the way the data is scattered among the nodes. The number of possible target nodes is reduced, leading to a network contention on the transfer destinations. The more the selection range is wide, the more this second drawback is important. The "less charged" notion is relative to a selection range indeed. With small selection ranges, there are many local less charged nodes on which the load is going to be spread, while using a maximal selection range every `root` node is going to choose the same less charged destination.

Figure 6 shows that, using this strategy with a 99 nodes selection range, the first day, a node acquires 90% of its load; it reaches 100% the second day. Once loaded, a node will rarely be chosen: the chance that a new node is less charged in its selection range is high. This brutal filling of new nodes explains the bad behavior of the less charged strategy.

With a 199 nodes selection range, the filling is even faster: a new node already has 100% of its load at the end of the first day, like illustrated by Figure 7 for which the selection range was set to 199 nodes. To obtain these figures, we let the system warm up, then, for each new node we record its load every day. An average for each age is then computed.

We can conclude that the more the selection range is wide, the more there will be contention on new, empty nodes incoming bandwidth. This explains why in Figure 5 the number data losses increases while growing the selection range, using the less charged strategy.

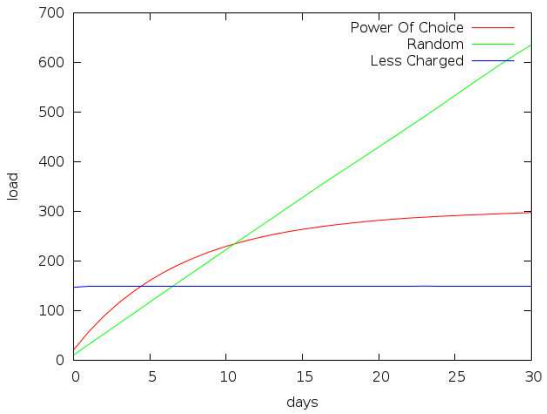


Figure 7: Storage load evolution using a 199 nodes selection range.

In all our experimentations, the power of choice strategy behaves well. Figure 5 shows that except for small selection ranges, for which the less charged strategy behaves slightly better, it induces less data block losses than the other strategies. Furthermore, it is important to recall that this strategy does not impose to maintain an up-to-date knowledge of the entire selection range nodes load, which is the case for the less charged one. The load of the two randomly chosen nodes can be probed when needed.

By selecting, for each data block to repair, the less charged node between two randomly chosen ones: (i) the power of choice strategy spreads the load among many destinations, contrarily to the case of the less charged strategy; and (ii) it distributes the load better than the random strategy.

4.4 Impact of bandwidth symmetry/asymmetry

All the results presented above have been obtained using an asymmetric bandwidth network, modeling classical Internet connections. However, datacenter networks are symmetric, and event for domestic Internet connections, the networks tend to be more and more symmetric. We thus study the impact of using a symmetric bandwidth network.

Figure 8 gives the number of data block losses according to the selection range width for the three strategies using a symmetric network. There are less data block losses using the random strategy or the power of choice strategy than in the asymmetric bandwidth case. The benefits are very important with small selection ranges for all the strategies. The number of losses can even be divided by 8 in the case of the random strategy. These results were expected: in the general case, using SPLAD, while a failure occurs, there are approximately the same number of sources and destinations for the healing transfers. In the case of a network with asymmetric bandwidths, the outgoing bandwidth of the source nodes was the limitation. The outgoing bandwidth, in the symmetric case is more than five times higher than in the asymmetric one.

However, in case of wide selection ranges, the less charged strategy drawback (i.e., the contention it induce on the destination nodes) is even more important. This is explained by the fact that in this particular case the contention is on the incoming bandwidths of the less charged nodes. They are favored in the case of asymmetric bandwidth networks: the incoming bandwidth of a node is almost divided by two in the symmetric bandwidth network configuration compared to the asymmetric case ($5.5Mb$ versus $10Mb$). Using a selection range of 199 there is twice the

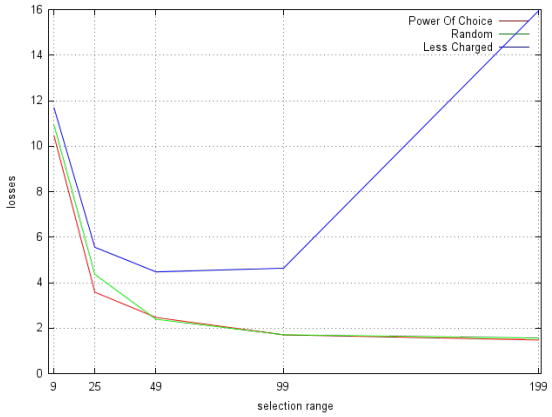


Figure 8: Comparing strategies using a symmetric bandwidth network.

number of losses compared to the asymmetric bandwidth network case.

4.5 Storage load distribution

We have taken the data block loss ratio as the main metric. We have observed that the storage load distribution has an impact on the data block losses. However, the storage load distribution is itself an interesting metric. For a same number of losses, the losses can occur uniformly among time or by bursts depending on the distribution of the storage load. In practice, one may prefer to have to handle few big data loss events than having to handle many small ones, regardless the total number lost pieces of data [21]. Part of the cost induced by a data loss, for the storage provider, is fixed. However, we adopt the users point of view and we focus on the total number of data block losses. Furthermore, the needed storage capacity of the nodes depends on the storage load distribution.

Figure 9 presents the distribution of the storage loads at initialization, and after two simulated years. It presents, for each load, the average percentage of the nodes having this load (among the 210 runs). At the beginning, for each strategy, the data block copies are placed using the right strategy. In all the cases, at day zero, the load is uniformly distributed: approximately 100% of the nodes store the same load (150 data block copies). As all the strategies give similar results, we've plotted only one curve on the figure for the load distribution at day 0.

However, after two years of failures and reparations, only the less charged strategy still gives a uniform distribution.

The random strategy, as expected, presents a highly non-uniform distribution with old nodes being overloaded. Notice that for clarity, the figure has been truncated, Table 1 gives the maximum loads that have been observed for each strategy over 132 090 samples: after day 100, the maximal load has been measured and recorded each day, this for the 210 runs. We can see that the mean maximum load of the random strategy is already high (more than five times the average), and furthermore, the load varies a lot, the maximum measured load being 2 188 data blocks. This implies either to buy over-sized storage devices for each node (the average load is 150 data blocks), or to maintain a fallback complex method, e.g., storing a forwarding link pointing the actual storer node instead of the actual data.

The power of choice policy gives a remarkable result: each load is represented by the same amount of nodes; the storage load is far from being uniformly distributed among nodes. Figure 9

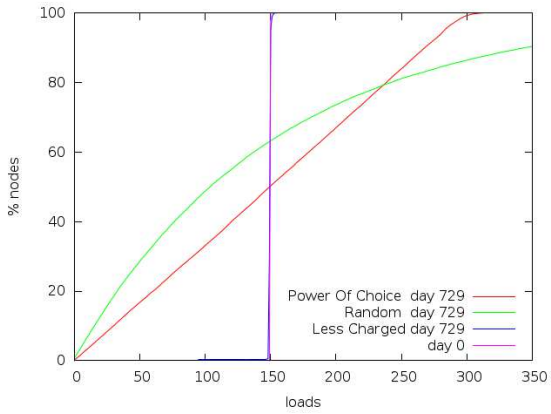


Figure 9: Impact of the target node choice policy on load distribution.

strategy	mean	minimum	maximum
Less Charged	153	150	165
Random	864	465	2188
Power Of Choice	300	269	328

Table 1: Maximum loads statistics

shows that there is approximately the same number of nodes having 0 data blocks, than nodes having 150 data block or nodes having 300 data blocks. Moreover, this is a stationary state. Furthermore, the variation is low, we can observe in Table 1 that upon the 132 090 samples, the most charged node was never above 328. From a practical point of view, that means that it is possible to oversize the nodes storage devices: a bit more than twice the average should be enough.

The fact that the power of choice policy maximum storage load is twice the average load is predictable; it is induced by the power of choice policy and the probability that a node has to be chosen according to its load (as illustrated by Figure 3). It can be proved if we assume that the distribution of the average number of nodes according to the storage load is uniform, as plotted in Figure 9 (the distribution of the storage load among the nodes is not uniform).

Proof:

Let $Maxl$ be the mean of the observed maximum load ($Maxl = 300$ in our evaluations), let m be the total number of data blocks ($m = 10\ 000 * 3 = 30\ 000$ in our evaluations) and let n be the number of storers ($n = 200$).

Let $MeanL = \frac{m}{n}$, the average number of blocks nodes would store if the load was evenly distributed among the n nodes. Then the result $Maxl = MeanL * 2$ comes from the fact that the mean ratio of nodes for every observed load is a constant c.

We have

$$m = \sum_{i=0}^{Maxl} (c * i) = c * \sum_{i=0}^{Maxl} i.$$

Furthermore

$$\sum_{i=0}^{Maxl} i = \frac{Maxl * (Maxl + 1)}{2} \text{ and } c * (Maxl + 1) = n$$

Therefore

$$m = \frac{n}{Maxl+1} * \frac{Maxl*(Maxl+1)}{2} = \frac{n*Maxl}{2}$$
So $Maxl = 2 * \frac{m}{n} = MeanL * 2$.

■

We are currently working on the analysis of this stationary particular storage load distribution.

4.6 In a nutshell

We have learned several lessons from our evaluation campaign.

First of all, a distributed storage system should not be overloaded. For a given number of storage nodes, a given network capacity and a given failure pattern, a distributed storage system can only store a limited amount of data without losing too much data. If a distributed storage system is overloaded, the replication strategy used is even more important than in the well-sized case: bad strategies behave even worse in overloaded systems.

Second, it is interesting to scatter the data block copies among many nodes to allow the system to parallelize data transfers while healing failures. However, this may have a significant cost, and the benefit of scattering decreases for big selection ranges. Thus, a tradeoff has to be found between the maintenance cost and the scattering benefit.

Finally, the way the data is scattered, the choice of the node that stores data block copies, is also important. If the selection range is small (i.e., the data is scattered among few nodes), the less charged strategy is interesting: it offers good results both in terms of data durability and storage load distribution. However, as mentioned above, small selection ranges do not give good results in terms of data losses in general.

For larger selection ranges, the power of choice strategy is a good choice. It permits to have fast reparations by distributing the new copies load on many destinations while keeping a reasonable storage load distribution among nodes, and this, without requiring a costly maintenance overhead. As already pointed out by Cidon *et al.* [21], random data block copies placement does not give good results in general.

Acknowledgment

This work as been partially done within the context of the French ODISEA FUI project funded by région Ile-de-France.

5 Conclusions

Distributed storage systems use replication mechanisms to support failures. In such systems, a storer node stores many different pieces of data. In this paper we have studied the impact of the way the data block copies are distributed among the storer nodes. We have shown that if nodes share a high number of common data blocks, failure healing will not be efficient, leading to many data losses. On the opposite, if the data block copies are shuffled among many nodes, failure reparation will involve many transfer sources and destinations, speeding up the system healing and diminishing the data loss risk.

We have also observed that the storage load distribution has an important impact. It is a hard task to distribute evenly the storage load without having to acquire and maintain a large knowledge on the neighbor nodes and risking network contention. We outlined that the power of choice strategy behaves well for such a task.

Finally, our SPLAD model allows us to study the behavior of a distributed data storage system on a long term. SPLAD relies on the notion of adjustable-width selection ranges to tune the way the data block copies are scattered among the system's nodes. It also permits to choose among different strategies to select storer nodes within a selection range.

References

[1] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP '03: Proceedings of the 9th ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, October 2003, pp. 29–43.

[3] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci *et al.*, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[4] S. Ramabhadran and et al., "Analysis of long-running replicated systems," in *The 25th IEEE Annual Confrence on Computer Communication (INFOCOM)*, 2006.

[5] F. Picconi, B. Baynat, and P. Sens, "An analytical estimation of durability in dhts," in *4th International Conference Distributed Computing and Internet Technology,(ICDCIT)*, 2007, pp. 184–196.

[6] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "Relaxdht: A churn-resilient replication strategy for peer-to-peer distributed hash-tables," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 2, pp. 28:1–28:18, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2240166.2240178

[7] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Scalable consistency in scatter," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043559

[8] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Grid Computing—GRID 2001*. Springer, 2001, pp. 75–86.

[9] G. Silvestre, S. Monnet, R. krishnaswamy, and P. Sens, "AREN: a popularity aware replication scheme for cloud storage," in *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2012.

[10] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[11] A. Pace, V. Quema, and V. Schiavoni, "Exploiting node connection regularity for dht replication," *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, pp. 111–120, 2011.

[12] F. Dabek, J. Li, E. Sit, J. Robertson, F. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *NSDI '04: Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, San Francisco, CA, USA, March 2004.

[13] A. I. T. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *SOSP '01: Proceedings of the 8th ACM symposium on Operating Systems Principles*, December 2001, pp. 188–201.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM*, vol. 31, no. 4. ACM Press, October 2001, pp. 161–172.

[15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A global-scale overlay for rapid service deployment," *IEEE Journal on Selected Areas in Communications*, 2003.

[16] A. Ghodsi, L. O. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," in *DBISP2P '05: Proceedings of the 3rd International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Trondheim, Norway, August 2005, p. 12.

[17] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, "Performance evaluation of replication strategies in DHTs under churn," in *MUM '07: Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*. New York, NY, USA: ACM Press, December 2007, pp. 90–97.

[18] Q. Lian, W. Chen, and Z. Zhang, "On the impact of replica placement to the reliability of distributed brick storage systems," in *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, June 2005, pp. 187–196.

[19] R. van Renesse, "Efficient reliable internet storage," in *WDDDM '04: Proceedings of the 2nd Workshop on Dependable Distributed Data Management*, Glasgow, Scotland, October 2004.

[20] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "Farsite: Federated, available, and reliable storage for an incompletely trusted environment," in *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, December 2002.

[21] A. Cidon, S. M. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 37–48. [Online]. Available: http://dl.acm.org/citation.cfm?id=2535461.2535467

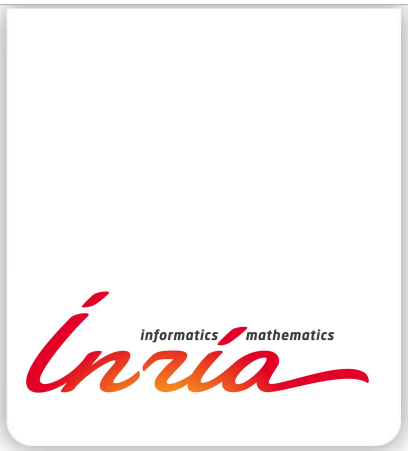
[22] M. Bertier, O. Marin, P. Sens *et al.*, "Performance analysis of a hierarchical failure detector." in *DSN*, vol. 3, 2003, pp. 635–644.

[23] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[24] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *in Handbook of Randomized Computing*. Kluwer, 2000, pp. 255–312.

[25] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," http://peersim.sourceforge.net/.

Contents



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399