An Empirical Study of Data Redundancy for High Availability in Large Overlay Networks

Giovanni Chiola *
Dipartimento di Informatica e Scienze dell’Informazione (DISI)
Università di Genova, 35 via Dodecaneso, 16146 Genova, Italy
chiola@disi.unige.it

Abstract

Redundancy is crucial for high data availability in an environment where computing nodes and/or communication channels are subject to transient failures, such as the Internet. Various techniques ranging from (multiple) replication to more sophisticated erasure correction coding have been studied. To the best of our knowledge, however, no easy rule of thumb has been devised to guide the distributed application designer in the choice of the appropriate technique and the right level of redundancy to guarantee the desired level of data availability. In this paper we present some simulation results that could help clarify the relation between the redundancy technique we adopted and the expected data availability in the context of medium to large size overlay networks, such as a Chord-like DHT.

1 Introduction

A DHT is a self structured collection of independent peers that share the association of data with search keys in a distributed way. Each peer may join or leave the structure independently. When joining the DHT, each peer stores part of the hash table predefined by the key that is statically assigned to it. Any peer can add, update, look-up, or possibly remove information from the DHT. The data access operations are usually called efficient or scalable if they can be performed in logarithmic time and in logarithmic communication complexity with respect to the number of peers. Well known examples of DHT prototypes are Chord [9], Pastry [8], etc. A common characteristic of all DHT proposals is that they are statistically balanced thanks to the properties of the cryptographic hash functions that are adopted [2].

The distributed, decentralized, and self-organizing na-

ture of a DHT suggests that the structure could be resilient to local machine faults and local communication problems, dynamically re-organizing and adapting itself to changing operating conditions. Adopting proper redundancy for data storage, such as erasure correction codes [6, 7] is one of the important aspects, although certainly not the only one, that can guarantee resilience of the DHT against faults and security attacks [1]. Indeed, resilience of the routing protocol must also be considered of crucial importance. [1, 4, 10]. We believe that the critical factors that are required to obtain resilience are:

• the exploitation of redundancy for storing data on more than one independent peer;
• the quick re-organization of the structure that must be completed before other peers possibly fail;
• the intrinsic security and robustness of the look-up/routing algorithms.

In this paper we shall only focus on data redundancy, which should provide the short-term availability of data even when some of the data storing peers become (temporarily or permanently) unavailable. The objective of this redundancy is to re-construct the data that were affected by the loss of some peers. In the longer term, if the faults are permanent, the reconstructed data should be re-allocated to other (functioning) peers in order to restore the original level of redundancy, and be able to withstand subsequent possible failures of other peers. Data redundancy is therefore one of the components that are needed to implement fault-tolerant storage in the long term, and should be complemented by appropriate availability checks and data reconstruction and reallocation, which are, however, beyond the scope of this paper.

Two classic techniques have been investigated and proposed for (short term) data redundancy: data replication, whereby multiple copies are created and distributed over different peers, and the so called “k out-of n” erasure correction codes [6, 7] whereby the original data representation

*Financially supported by the Italian MIUR under the framework of the FIRB 2001 action, WEB-MiNDS Project. http://web-minds.consorzio-cini.it/
is split into \( k \) pieces and additional \( n - k \) redundant pieces are added. The choice between the two techniques has been (and still is [3]) the subject of numerous studies and publications. Despite the abundant literature, no definitive choice has been universally agreed upon, and no easy rule of thumb has been devised to guide the intuition of system designers.

The aim of this paper is to present an empirical study of the problem based on a very straightforward Monte Carlo simulation technique. We evaluated the expected availability of data under various operational and fault scenarios, and we came to the conclusion that the complete replication technique may be worthwhile only in few, specific cases. In many scenarios of overlay network applications, adopting erasure correction codes of the “\( k \) out-of \( n \)” type having the particular \( n \geq 2k - 1 \) constraint appears to be the only reasonable option. Moreover, the “fragmentation” parameter \( k \) should be chosen on the basis of a trade-off between communication efficiency and desired availability.

The rest of the paper is organized as follows; Section 2 outlines our simulation model and simulation methodology. Section 3 presents our simulation results and their interpretation. Section 4 concludes by summarizing the main suggestions to DHT system designers that will allow them to implement data redundancy in an efficient way, based on our simulation studies.

## 2 Simulation model and methodology

We shall start with a brief summary of the shared characteristics of DHTs that have been proposed in the literature, and which should be taken into account when devising a simulation model with sufficiently high fidelity. Let us take into consideration the snapshot of a system taken at an arbitrary time instance \( \tau \) in which the DHT is made up of \( N \) peers connected according to the chosen protocol rules (e.g., a virtual ring structure in both Chord and Pastry). We need not actually discuss the properties of the virtual structure, since they do not affect our model. The generic peer is identified by an index \( p_i \) and by a randomly associated key \( k_i \), which is drawn from a large key space corresponding to the range of a cryptographic hash function (128 or 160 bit).

Data is allocated to peer \( p_i \) depending on the value of its key \( k_i \), as well as on the value of the key of its predecessor or successor \( k_{i-1} \) or \( k_{i+1} \). By assumption, keys are randomly associated to peers, with uniform probability distribution, therefore data pieces are also allocated to peers randomly, with uniform probability distribution as well. This assumption of random, independent, identically distributed data allocation to peers is crucial for the definition of a very simple, yet perfectly accurate, simulation model.

We assume that the topology was stable right before \( \tau \), and that \( F \) out of the \( N \) peers suddenly became faulty at time \( \tau \). At this point, short term data availability (i.e., the possibility to access data right after \( \tau \) and before the state of the DHT undergoes further changes, either due to additional failures or to repairs) is guaranteed only by exploiting data redundancy and contacting some of the still functioning \( N - F \) nodes.

We also have to clarify the type of faults we are dealing with. The simplest fault model one can imagine is the fail-stop model, in which a node can stop functioning and never resume its activity. A more realistic assumption is one of intermittent faults, whereby a node alternates between a functional and a faulty state. In case of intermittent faults, the node itself may also be unaware of some of its temporary failures (such as in the case of temporary disconnection from the other peers due to transient network faults). The latter case is the most difficult to handle as it can create inconsistencies in distributed data storage.

From a very abstract point of view, access to redundant data pieces may be modeled by a triplet of integers \((r, w, a)\). The former identifies the minimal number of non-failed peers storing the required data that must be contacted in order to retrieve data. The second represents the minimal number of non-failed peers storing the required data that must be contacted to store new values. The latter indicates the additional number of (possibly new) peers that must be available to complete the access. The actual values of the triplet \((r, w, a)\) not only depend on the redundancy code that is adopted, but also on the type of access to the data. We analyze a scenario in which the options are chosen from the following matrix of 10 possibilities:

<table>
<thead>
<tr>
<th>access type</th>
<th>redundancy code</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>((0, 0, n))</td>
</tr>
<tr>
<td>read only</td>
<td>((1, 0, 0))</td>
</tr>
<tr>
<td>new version</td>
<td>((1, 0, n))</td>
</tr>
<tr>
<td>cons. update</td>
<td>((1, n, 0))</td>
</tr>
<tr>
<td>delete</td>
<td>((0, n, 0))</td>
</tr>
</tbody>
</table>

In this table, by “create” we mean the initial allocation of a new piece of data into the DHT, by “delete” we mean the removal of the data association to the key, so that a subsequent look-up would not be able to find it, by “new version” we mean the creation of a new version based on a modification of the previous one, and by “cons. update” we mean the update of an already stored piece of data so that the new value replaces the old value (thus the old value can no longer be retrieved). Notice that “new version” can safely be used if the fail-stop fault model is assumed, otherwise inconsistencies may arise. In case of intermittent fault models, “cons. update” should be used in order to avoid inconsistencies. The latter mode can be viewed as the sequence of three other access modes (first read the old value, then delete the old value, and finally create the new one).

From a preliminary analysis of the above matrix, it is
clear that the creation of a new association in the DHT is the least problematic type of access from an availability point of view: it will always succeed provided that $N - F \geq n$. All other access types require the availability of a non-empty subset of the $n$ peers that originally stored the required data pieces, and that could have become unavailable due to failures. In particular, we focus on the “read” and “consistent update” access modes, which would most likely be the ones used in these applications.

By simply comparing the access requirement triplets $(1, 0, 0)$ and $(1, n, 0)$, one can immediately guess that the availability of the consistent update operation is lower than the availability of the read operation when a replication $n > 1$ is adopted. On the other hand, by comparing the requirement triplets $(k, 0, 0)$ and $(k, n - k + 1, k - 1)$, one can observe that there is the opportunity to have the same availability for read and consistent update operations by adopting erasure codes (provided that the total number of non-failed peers $N - F$ is greater than or equal to $n$). This result can be obtained by choosing the particular value $n = 2k - 1$, thus yielding the constraint $(k, k, k - 1)$.

Thanks to the above mentioned random and uniform data allocation, and through the requirement triplets reported in our matrix above, one could try to derive more or less complex analytical formulas (most likely resembling binomial coefficients involving the quantities $n, k, N,$ and $F$), similar, for example, to the ones derived in [5]. Instead, we chose a simpler approach. We set up a very easy to program simulator following the so called Montecarlo simulation approach, in order to derive numerical estimates of the availability level for the various access modes and scenarios.

Our simulation model is almost trivial. First we construct a vector of $n$ boolean variables that represent the failure state of the $n$ peers that we assume were chosen to store the redundant piece of data to be accessed. We use a pseudo-random number generator to assign the value “true” to each vector element with probability $F/N$. Then we verify whether the access requirements defined in the above matrix are satisfied (in which case the data is available to complete the operation) or not. If the check succeeds, we return the availability value $A = 1$, otherwise we return the value $A = 0$. We repeat the whole procedure of “vector generation and check” a great deal of times ($R$ times), and then compute the average value for the availability estimate

$$\bar{A} = \frac{1}{R} \sum_{i=1}^{R} A_i$$

In addition to the average estimate, we also compute the 99% confidence interval (i.e., the range of values within which the “true expected value” of the random variable $\bar{A}$ is expected to be with 99% probability), and keep simulating new instances until the confidence interval becomes less than 1% of the average value. This way our computed results reach a precision in the order of two decimal digits. The meaning of this precision is that with 99% probability, at least the first two digits of our average estimate $A$ are equal to the “true expected value” that would be computed by an exact analytical approach rather than a simulation one. Notice that this simulation procedure is not only very easy to program, but also extremely quick to converge to high precision results. All curves reported in the following section were computed within less than one minute of CPU time on a laptop computer.

3 Simulation results

Let us start with the diagram showing the overall availability behavior as a function of the fraction of failed nodes $F/N$. Figure 1 shows such a diagram for an average/small size DHT containing 1,000 peers, for various redundancy schemes and access modes. The basic curve is the straight segment starting at $A = 1$ for $F = 0$ and going towards the point $A = 0$ for $F/N = 1$, which is, of course, the availability expectation for the non-redundant storage of one copy stored at a single peer. The use of duplication (2 copies on two different peers) implies higher availability for read access type, but lower availability for consistent update access type compared to the non redundant storage, as expected. As is also expected, the availability drastically grows (or decreases) when the number of copies increases to 3 or more for read (respectively for consistent update) access modes. These diagrams clearly show that replication cannot support high availability if data need to be updated from time to time. In our opinion, replication can only be used to provide high availability of persistent data that will never need on-line updating.

Concerning erasure correction codes, comparing the curve for “3 out-of 4” (which happens to be superimposed to the one for “2 out-of 4” consistent change) against the others clearly shows that a choice of $n < 2k - 1$ results in poor availability, even though it guarantees the same level of availability for “read” and for “consistent update” operations like the choice of $n = 2k - 1$. This can obviously be explained by the lower redundancy factor if compared, for instance, to “3 out-of 5.” Therefore, we suggest that the choice of a “$k$ out-of $2k - 1$” is optimal in case data are subject to frequent on-line updates.

The choice of “$k$ out-of $n$” with $n \geq 2k$, as expected, results in higher availability for read and lower availability for consistent update compared to the “$k$ out-of $2k - 1$” encoding. However, one can easily see that the availability offered by the “$k$ out-of $2k$” encoding is substantially higher than what is offered by multiple copies for consistent update operations with peer failure rate below 0.5. For instance, the “2 out-of 4” erasure encoding offers 95% availability for
consistent change access in the face of up to 10% of peer failures, while a 2 copy system would support only 80% availability under the same conditions. Based on these observations, we can conclude that the choice of a “$k$ out-of $n$” encoding with $n \geq 2k$ is a better alternative to replication in cases where read access prevails but in which the consistent update operations must be carried out from time to time.

A further comment that can be made is that the higher the fragmentation into numerous pieces $k$, the greater the availability that can be offered in case of failure ratio below 0.5. However, the fragmentation of data into a high number of small pieces spread out over different peers reduces the efficiency of both read and consistent update accesses since it involves higher communication and synchronization overhead. Therefore, the minimal value $k$ that guarantees the required level of availability under some predefined fault scenario has to be chosen. In order to study such a trade-off, consider the “zoomed-in” version of the same diagram that is depicted in Figure 2. For example, we can observe that if 0.95 availability is considered sufficient for a given application, then the simplest “2 out-of 3” erasure code (which, by the way, could be obtained by using a trivial XOR operation like in a RAID-5 system, rather than using sophisticated and costly polynomial code operations) would suffice to cover up to 13.5% of peer failures, as compared to 5% of peer failures that the non-redundant storage of one copy can withstand. If we want to provide higher availability values, let’s say 0.98, we should probably increase fragmentation to “3 out-of 5,” which (at the expense of adopting a real erasure correction code based on polynomial code operations) would provide similar coverage to what is provided by “2 copy duplication” in case of read operation, withstanding the loss of up to 14% of peers. If we further increase our expectations, requiring for instance 0.99 availability even with more than 22% of peer failures, we would definitely need an “8 out-of 15” erasure encoding, which would outperform the availability of a “3 copy” system even in case of read-only access.

In order to complete our empirical study, let us now discuss the effect of the size of the DHT on the expected availability. Figures 3 and 4 depict the availability curves for a

![Figure 1. Availability for “read” and for “consistent update” data access using various redundancy schemes in a 1,000 peer system for varying fraction of node failures.](image-url)
Figure 2. Zooming the initial part of availability diagram for “read” and for “consistent update” data access using various redundancy schemes in a 1,000 peer system for smaller fraction of node failures.

500,000 peer and a 60 peer system, respectively. Comparing the results shown in Figure 3 against the ones depicted in Figure 2 one can observe that there is virtually no difference. Hence, even a large increase in the total number of peers does not seem to affect the results.

On the other hand, a reduction in the number of peers, such as the one depicted in Figure 4 does make some difference. Indeed, all forms of redundancy coding appear to perform better in a 60 peer system rather than in a 1,000 or a 500,000 peer system, with remarkable improvement in the case of erasure codes with a high level of fractioning. For instance, notice that the “16 out-of 31” encoding yields 0.99 availability in the face of up to more than 35% of peer failures in such a small system, rather than being limited to withstanding less than 30% of peer failures as occurs in larger systems.

Such an interesting phenomenon appears to be difficult to exploit in an attempt to save redundancy overhead for small systems. We believe it makes little sense to adopt an overlay network in which the number of peers is of the same order of magnitude of the fragmentation parameter $k$. However, it can guarantee that an encoding designed to provide high availability for read access with a great deal of peers does not deteriorate if it is operated with fewer peers. This statement, of course, remains true for consistent change access, provided that the total number of non-failed peers $N - F$ remains higher than $2k - 1$. Whenever this condition ceases to hold, availability drops to zero, as can be observed from the diagrams labeled “(16,31)C” and “(24,47)C” reported in Figure 5.

### 4 Conclusions

In this paper we studied the application of well known data redundancy techniques to a particular class (which includes DHTs) of peer-to-peer overlay networks. The distinctive characteristic of the class we studied is the random allocation of data pieces to peers, having uniform probability distribution. The purpose of this redundancy is to provide higher availability in the short-term for accessing data,
even when some of the peers storing those data pieces become (temporarily or permanently) unavailable. As already pointed out, this redundancy complements, but does not replace reconfiguration and reallocation techniques, which could instead provide high availability in the longer term, in the face of permanent faults.

We identified various data access types that we believe are of interest in DHT applications, and we focused on two of them: the read-only access type that could be supported by simple data replication as well as by erasure correcting codes; and the consistent update access type, that requires a particular type of erasure correcting codes for high availability which are characterized by fractioning the original data into $k$ pieces and by adding exactly $k - 1$ redundant pieces. The latter mode guarantees consistency without injecting obsolete data pieces into the system, even in case of intermittent fault models in which a node can resume operation without noticing the occurrence of its own fault period.

By simple stochastic modeling and Montecarlo simulation we were able to estimate the availability offered by the system as a function of the fraction of failing peers, with a high level of accuracy and very low simulation time. Based on these diagrams we discovered that the availability characteristics of the redundant coding techniques are almost insensitive to the total number of peers in case the number of peers is high. On the other hand, availability increases in case the number of peers becomes smaller and smaller, until the number of functioning peers becomes insufficient to support the desired level of redundancy.

From a practical point of view, the results we obtained would suggest the use of “$k$ over $2k - 1$” erasure codes if read and consistent update availability are of equal importance. If, instead, read access types prevail over update access types, higher redundancy “$k$ over $n$” erasure codes with $n \geq 2k$ can be taken into consideration. In any case, the lowest value of fractioning $k$ that can guarantee the desired level of availability under the worst chosen failure scenario must be chosen (of course the solution is feasible only when considering peer failure rate below 0.5, and when the number of still functioning peers is at least $n$). If the availability requirements are not particularly strict, e.g., 95% availability with up to 13% of peer failures, then the simplest RAID-
Figure 4. High availability region for “read” and for “consistent update” data access using various redundancy schemes in a 60 peer system for node failures up to 45%.

Our findings will help us optimize the availability and performance of a prototype middleware whose implementation is currently under way within the framework of the Italian FIRB project WEB-MiNDS. In this framework, we are implementing a peer-to-peer storage prototype that can support personal file systems. Data structures are divided in two levels: first of all, a meta level which is implemented based on a Chord-like DHT, which associates the identity of a user with the list of IP addresses of peers that store data for that user’s file system; secondly, a data storage level in which a small subset of peers takes charge of the actual file storage. The rationale for this two-level design is to combine the efficient search characteristics of DHTs with the flexibility and easiness of data migration of unstructured peer-to-peer systems. Our DHT protocol guarantees the fail-stop semantics for obsolete meta-data by purging the peer’s table before it can re-join the ring. Hence, replication can be adopted for meta-data without incurring into consistency risks. In our case, the complete replication mechanism was chosen (even thanks to the small size of meta-data) in order to keep the time needed to restore the...
initial redundancy after a peer fault is detected to a minimum. Data storage is instead implemented with a $k$ out-of $2k - 1$ erasure correction code in order to guarantee consistency and high availability for the update access mode, as well as for the read-only access mode.

References


