File Survivability in P2P Networks based on Stochastic Swarm Guidance

Francisco Barros
francisco.t.barros@tecnico.ulisboa.pt
IST Taguspark, Av. Prof. Dr. Cavaco Silva, 2744-016 Porto Salvo, Portugal

***Abstract*— With the growing importance of IT devices and solutions in governments, companies, and individuals' lives, cloud services, in particular cloud storage, have become increasingly desired alternatives to safeguard important files. Traditional approaches range from centralized architectures, where multiple nodes continuously report to highly reliable monitoring servers, to decentralized Peer-to-Peer (P2P) networks, in which nodes gossip users' queries to find and store their items. All paradigms use supplementary techniques to improve some performance metric of the system. In this paper, we propose the use of Probabilistic Swarm Guidance (PSG) to increase the reliability of a system and the durability of stored files. We focus on finding out if the approach brings benefits to this type of service. We create a custom simulator where we test Markov Chains (MCs) generated with different procedures and find that while we do not provide guaranteed durability with the solution, the P2P-based Distributed Backup System (DBS) outperforms Hadoop Distributed File System (HDFS).**

***Index Terms*— File Durability; Markov chains; Peer-to-Peer Storage; Swarm Guidance;**

I. Introduction

We have entered an era in which devices with computing capabilities are ubiquitous. Individuals generate more data than ever before, some of which they might want to backup remotely, e.g., photos. Organizations recognize IT and the internet as a vital part of their business and governance. The law often requires them to store critical documents for long periods, with the risk of penalization when they fail to comply. Regarding digital storage, one choice is to use Distributed File Systems (DFSs); they facilitate the distribution of documents to multiple clients, which can collaboratively and transparently modify them. Continuous availability and session-guarantees are fundamental properties of DFSs. Alternatively, DBSs are a sub-category of DFSs. Their priority is to ensure that uploaded files become durable, perhaps at the cost of losing shareability or editability. At the extreme, two paradigms emerge as pillars for these systems.

Supervisor Prof. Carlos Silvestre is with the Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Macau, Macau, China, on leave from Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal, csilvestre@umac.mo

Co-supervisor Daniel Silvestre is with the Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Macau, Macau, China, and with the Institute for Systems and Robotics, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal., dsilvestre@isr.ist.utl.pt

This work was partially supported by the project MYRG2018-00198-FST of the University of Macau; by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UIDB/50009/2020.

On the one hand, we have P2P networks, in which equally privileged peers contribute with a portion of their resources to achieve common goals. This approach is popular due to, among others, its self-organized behavior, lack of centralization, and low cost, e.g., HandyBackup. On the other hand, Cloud platforms offer unmatched, on-demand, self-served, availability, and reliability at higher price points. The latest approach is trendy, with DropBox, Google, and Microsoft offering diversified solutions. Cloud-based systems are often centralized architectures, in which a large number of computers are clustered and managed by master entities, which may become bottlenecks. While P2P implementations are cheaper for both companies and their clients, they have a hard time achieving performance levels seen in Cloud implementations. P2P approaches have a higher inherent risk of permanent file loss resulting from the fact that contributors may leave at any time for no particular reason (churn), making them unappealing for clients who seek to store critical data. However, as aforementioned devices disks have ever-increasing mean-time-to-failure values and with allocation spaces growing disproportionally faster than the generality of file-sizes, we propose a P2P-based DBSs as a possible solution to the problem.

This body of work aims to assert the viability of PSG in a DBS environment. The reasoning is twofold. First, due to the widespread adoption of this method in robotics and control fields, where results demonstrate that it is possible to gift autonomous agents, working independently of each other, with simple probabilistic rules and limited knowledge of the environment, and have them achieve complex tasks as a group. Furthermore, PSG also enables recovery from predicaments without human interference. Second, to the best of our knowledge, no one attempted the use of PSG as an underlying algorithm for file durability in DBSs. Consequently, the topic represents an exciting opportunity to observe the behavior of the technique outside of its typical application. Our main contributions can be summarized as follows:

- An open-source cycle-based simulator for Python users.
- First implementation of a PSG algorithm in a DBS.
- Evaluation and comparison against abstracted HDFS.

The document has the following structure. We survey the literature concerning different topics in Section II. In Section III, we define our problem more objectively. We outline our solution in Section IV and a synthesis of the used algorithms are given in Section V; Results are provided in

Section VI followed by our conclusions in Section VII.

Symbolic Notation

Miscellaneous

- Epochs ..... $T = \{t_0, t_1, \ldots, t_n\}, n \in \mathbb{N}$
- Function ..... $f(x)$
- Parameter or variable x
- System States ..... $S = \{s_0, s_1, \ldots, s_n\}, n \in \mathbb{N}$

Algebra

- Vector ..... $\underline{v}$
- Vector at epoch ..... $\underline{v}^{(t)}, t \in T$
- Matrix ..... $\mathbf{M}$
- Matrix or vector transpose ..... $x^\intercal$
- Matrix element ..... $\mathbf{M}_{ij}, i, j \in \mathbb{N}$
- Matrix row ..... $\mathbf{M}_{i\star}, i, j \in \mathbb{N}$
- Matrix column ..... $\mathbf{M}_{\star j}, i, j \in \mathbb{N}$
- Matrix Eigenvalue ..... $\lambda(\mathbf{M})$
- Identity Matrix ..... $\mathbf{I}$
- Zeros Matrix, Ones Matrix ..... $\mathbf{0}$, $\mathbf{1}$
- Ones Vector ..... $\underline{\mathbf{1}}$
- Equilibrium vector ..... $\underline{e}$
- MC at epoch ..... $\mathbf{M}^t, t \in T$
- MC event chance at epoch .. $\mathbf{M}^t_{ij}, t \in T \wedge i, j \in \mathbb{N}$

II. Literature Review

A. Swarm Guidance

The control and robotics fields of research are packed with complex problems. One such problem is how to control effectively groups of robotic agents. Swarm Guidance (SG) is a bio-inspired technique [1], [2] that emerges as a straight forward solution that acts at the agent level rather than on the group level. Thus, SG permits the decentralization of control systems by gifting each agent with the capability of carrying their functionality and contributing to the group mission without necessarily interacting with the remainder of the agents. Consequently, this avoids intricate algorithms that are hard to validate. Research by B. Açıkmeşe *et al.* [3]–[5] shows PSG is a viable form of SG. Summarizing, SG is a compelling technique that may be applied to a wide array of topics [6], [7]. We study its effectiveness in the safekeeping of files stored in P2P networks.

B. Distributed File Systems

We can classify DFSs according to their architectural centralization. To say that a system is centralized is to say that at least one component plays a central role in its operation, e.g., metadata server that stores file locations or monitors that decide what nodes are operational. P2P approaches are often decentralized. Classical DFSs like Google File System (GFS) [8] and HDFS [9] are examples of centralized architectures. In fact, their modus operandi is the same save for the approach to security and permission handling. In these systems, clients talk to master servers to know which storage servers they need to contact to read or write files. These same masters are also responsible for receiving heartbeats from storage servers and controlling replication levels within them. Despite the disregard for centralized architectures in the scientific community, these systems have proven to offer unmatched service guarantees, concerning reliability and durability, and operational performance. Ceph File System (CFS) [10], [11], another state-of-art DFS, uses an algorithm to provide fast and precise localization without using an indexing server and leverages P2P behavior within clusters of storage servers, to mask and recover from faults. CFS, however, still relies on consensus performed by some of the cluster's dedicated nodes to decide on which storage servers are up. On the other end of the spectrum, we have a novelty system, called Gluster File System (GlusterFS) [12], a completely decentralized solution, in which indexing is done through algorithms similar to those in CFS. However, GlusterFS does not have any metadata dependency, which makes it exceptionally fast at handling small file operations and acceptably fast, albeit slower than GFS and HDFS at handling large file operations. Finally, recognizing that there is no one-system-fits-all solution, Hybrid File System (HybridFS) [13], creates an abstraction layer over HDFS, CFS and, GlusterFS allowing users to use them as if they were one single DFS, ensuring the best storage and access performance by respectively using dynamic file migration mechanisms and artificial intelligence to select which subsystem will receive a certain file.

C. Overlay Networks

Since we implement PSG algorithm over a P2P-based DBS, it is essential to understand overlays. They exist because it is often unfeasible or undesirable for each peer to know and interact with every other entity in the network. Thus, an overlay is a logical abstraction of the physical network where single-hop edges represent links between pairs of peers. There are two main categories of overlays, *structured* and *unstructured*. In the foremost [14], peer placement follows rigid rules to speed up read operations, commonly using Distributed Hash Tabless (DHTs) or tree implementations, e.g., decision trees. These overlays have higher maintenance costs since one change in the topology can cause a cascade of modifications concerning peer organization, the data they hold, or both, causing them to be unideal for highly dynamic environments. In the latter [15], [16], information dissemination is likely to occur in gossip or broadcasting fashions; this makes reads and writes slower than their structured counterparts, but the maintenance is often easier because peer placement is arbitrary – a new peer entering or leaving the network requires few rearrangements. Consequently, they tend to be more scalable and robust in the advent of failures but have a higher likelihood of peer isolation. In the last decade, some multi-layer overlays emerged. They are a combination of two or more conventional overlays concurrently abstracting the same physical network. When properly combined, these multi-layer types can satisfy a broader range of requirements for their applications and off-set some weaknesses. However, network and computational resources may also deplete faster and, their complexity can lead to more service failures. Alternatively, bio-inspired overlays [17], [18] have shown promising results, and their objective is also to diminish or eliminate the disadvantages associated with structured

or unstructured overlays, which they extend by introducing dedicated agents that accomplish complex behaviors (in similarity with SG) that would otherwise require time or space consuming algorithms. These agents usually roam the network to aggregate data and rearrange file locations or peer connections. The downside is: results of agent actions are not always easy to evaluate.

III. Problem Statement

Let us consider a set of storage nodes, S, participating in a P2P network to hold B file block replicas or encoded fragments (parts) belonging to a file F. Their objective is to achieve the durability of F. Assume that message integrity, confidentiality, and authenticity are not at risk, that there are no malicious attacks on the system and that nodes do not deviate from the defined algorithms intentionally or otherwise. Admitting that message loss may occur when nodes communicate with one another; that they may suffer from disk errors or disconnect at any given time. We must introduce a distributed PSG algorithm into P2P-based DBS. The problem involves generating an overlay topology, represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, that defines edge-connections between pairs in S or alternately by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{S \times S}$. Topology rearrangements are only allowed when nodes in S leave or join the system, and they must be connected, i.e., network partitions are not allowed except due to failures. There are no constraints with whom peer nodes may communicate as long as any first message, in a sequence of exchanges, must result from a probabilistic event. Our algorithm must also use a policy that diminishes or delays the likelihood of losing parts from B. There is space for some centralized components; however, these must-have lightweight functions, since the cost benefits of P2P approaches lies in maximizing the utility of participating machines, S. In other words, centralized components can not: i) store data files; ii) directly monitor participating peers; iii) perform computationally intensive operations.

IV. Proposed Solution

A. PSG from Robotics to Distributed Systems

We study PSG in a DBS environment and see if files can survive in the system, i.e., we want to know if files uploaded to a remote set of peers using the PSG algorithm become durable. From a PSG perspective, our proposal is a mixture of the problems studied in B. Açıkmeşe *et al.* [3], [5] and the FMMC problem in Boyd *et al.* [19]. We want to create a MC that guides agents to the desired formation in space, defined by the equilibrium vector $\underline{e}$. The space is a logical overlay, our regions are nodes in the overlay, and our agents are files and their replicas. Since files are motionless, and the network links do not exert wind-like forces on them, we must account for this difference and adapt B. Açıkmeşe *et al.* ideas. We also need to be aware that overlay topologies change due to churn or other machine malfunctions. Consequently, we want the fastest possible MC to increase the odds that $\underline{e}$ (desired file distribution) is reached before such occurrences, thus increasing the system reliability, the files availability, and fair load-balancing among clusters' nodes. Boyd *et al.* Semi-definite programs (SDPs) are not directly applicable because our $\underline{e}$ is not uniform and even if it was, heuristically created MCs are not always slower than SDP ones. The non-uniformity of $\underline{e}$ can also result in a non-convex problem.

Algorithm 1 Creation and selection of the fastest MC.

```
function CreateMC(c)
    ▷ Random, symmetric and, connected, with self-loops.
    K ← NewTopology(c)
    e ← NewEquilibrium(c)
    ▷ Eq. (1)
    K_opt ← SDP.Solve(K)
    choices ← ∅
    ▷ Algorithm 2.
    choices ∪ MetropolisHastings ⟨K, e⟩
    choices ∪ MetropolisHastings ⟨K_opt, e⟩
    ▷ Eq. (2)
    choices ∪ GO.Solve(K, e)
    μ* = ∞
    M* = ∅
    for all M in choices do
        if μ(M) < μ* then
            μ* ← μ(M)
            M* ← M
        end if
    end for
    return M*
end function
```

B. Markov Chain Optimization

In order to attain a good performing DBS service that ensures the durability of uploaded files, when using PSG, two aspects are related to the algorithm itself. The first is the selection of a suitable $\underline{e}$; this means appropriately filling the entries of the vector accounting for past node behavior, their hardware capabilities, and perhaps usual working hours to minimize the probability of losing parts. We do not explore this subject. The second aspect is to use a MC that converges quickly to the $\underline{e}$, to minimize the time the file-hosting cluster spends in unideal states. Since optimal solutions are hard to produce, we employ multiple techniques, including relaxed constraint global optimization, to deal with non-convex cases, producing sets of feasible MCs, and choose the one with the fastest Second Largest Eigenvalue Modulus (SLEM) as depicted in Algorithm 1.

Our first technique is Metropolis-Hastings (MH) (Algorithm 2), using symmetric, connected adjacency matrices, with obligatory self-loops as a proposal matrix ($\mathbf{K}$) targeting $\underline{e}$. Self-loop enforcement is done for all methods since by experience, the self-loop property revealed a tendency to create faster chains and to reduce the algorithm bandwidth footprint, but is not obligatory. The second technique leverages the fact that uniform distributions, $\underline{u}$, are likely to create clean SDPs. We first optimize an adjacency matrix, $\mathbf{A}$, to $\underline{u}$ by minimizing the matrix eigenvalues (Eq. (1)); then, we

Algorithm 2 Metropolis-Hastings by B. Açıkmeşe *et al.*

```
function METROPOLIS-HASTINGS(K, e)
    R ← 0
    F ← 0
    M ← 0
    n ← Length(e)
    for i ← 0 to n do
        for j ← 0 to n do
            R_ij ← (v_i K_ji) / (v_j K_ij)
            F_ij ← min(0, R_ij)
            if i ≠ j then
                M_ij ← K_ij F_ij
            end if
        end for
    end for
    for i ← 0 to n do
        M_ii ← K_jj + Σ_{k≠j} (1 - F_ij) K_kj
    end for
    return M
end function
```

use the output as being the $\mathbf{K}$ that targets $\underline{e}$ in MH. The final technique uses global optimization (Eq. (2)), where our constraints force the output matrix to converge to $\underline{e}$. The model directly targets $\underline{e}$ because we allow the optimization variable to be asymmetric. Note that neither of the mathematical optimization models guarantees a globally optimal SLEM. The former technique is local-optimization-based, and, conversely, the latter is global-based but, the objective function minimizes the norm of the output matrix, which is a relaxed approximation of eigenvalue minimization.

$$\begin{aligned} \underset{\mathbf{K}_{opt},\, t}{\text{minimize}} \quad & t \\ \text{subject to} \quad & \mathbf{K}_{opt} = \mathbf{K}_{opt}^{\intercal}, \\ & \mathbf{K}_{opt} \geq 0, \\ & \mathbf{K}_{opt} \cdot \underline{\mathbf{1}} = \underline{\mathbf{1}}, \\ & \mathbf{K}_{opt} \odot (\mathbf{1} - \mathbf{K}) = \mathbf{0}, \\ & -t \cdot \mathbf{I} \preceq \mathbf{K}_{opt} - \frac{1}{n} \cdot \mathbf{1} \preceq t \cdot \mathbf{I}, \quad t \in \mathbb{R} \end{aligned} \tag{1}$$

$$\begin{aligned} \underset{\mathbf{M}}{\text{minimize}} \quad & \left\| \mathbf{M} - \frac{1}{n} \cdot \mathbf{1} \right\|_2 \\ \text{subject to} \quad & \mathbf{M} \geq 0, \\ & \mathbf{M} \cdot \underline{\mathbf{1}} = \underline{\mathbf{1}}, \\ & \mathbf{M} \odot (\mathbf{1} - \mathbf{K}) = \mathbf{0}, \\ & \underline{e}^{\intercal} \cdot \mathbf{M} = \underline{e}^{\intercal}, \quad \underline{e} \in \mathbb{R}^n \end{aligned} \tag{2}$$

While the optimizations may not result in the fastest chains, the speed-ups may be significant, as can be deduced by looking at Fig. 1. For the presented box plots, we ran a Monte Carlo simulation with 1000 $\langle \mathbf{K}, \underline{e} \rangle$ pairs, and for each of those pairs, we created a MC using the described methods for different-sized clusters.

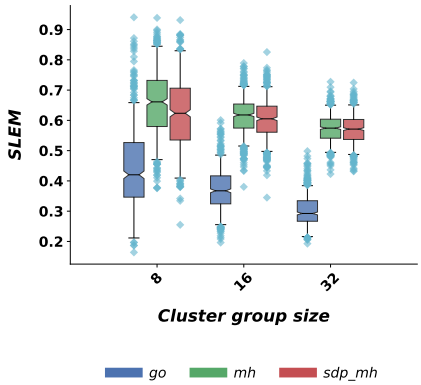


Fig. 1: Comparing Markov chains' mixing rates with our heuristic and optimization implementations.

C. Swarm Guided Distributed Backup System (SGDBS)

Our proposal is composed of five entities. Master servers, H, are responsible for handling client and contributor registration and authentication, and mapping clients' files to clusters, and keeping a record of each clusters' members. Clusters are groups of storage nodes who contribute with their private storage space to the DBS. A cluster maintains one single file with a predefined replication level, r, on behalf of precisely one client. Consequently, whenever a client, C, wishes to upload a file F, to the system, he first sends F's metadata to H_k, who will reply with a subset S of storage node identifiers with size n, along with the ideal equilibrium vector $\underline{e}$. Upon receiving H_k's response, C divides F into multiple blocks, B, and creates a random uniform symmetric and connected topology, $\mathbf{A}$, and uses it along with $\underline{e}$, to generate Markov matrix, $\mathbf{M}$, using Algorithm 1. C then slices $\mathbf{M}$ into n row vectors, each being sent to the respective S_n storage node, to be used as a routing table for Fs' blocks. C also sends one replica of each B_i to the r closest storage nodes. From this moment onwards, at every epoch, which does not need to be synchronized, each S_n uses F's routing table to send one or more B_i to other members of the cluster. This process ensures the synthesis of $\mathbf{M}$ and the eventual distribution of parts according to $\underline{e}$. Monitors, O, are a set of high-reliability servers but could be sets of dedicated commodity network nodes instead, like in CFS [10]. Either way, they do not contribute with storage, but rather, receive complaints from storage nodes about their direct cluster neighbors. When monitors receive a quorum of complaints respecting a node, the complaining cluster's members evict the complainee (i.e., the suspicious or failed node). Eventually, another storage node will replace the ousted one. When a node leaves or joins a cluster group, the Monitors create and broadcast a new MC. Ideally, this replacement would be direct so that the MC could remain the same or only be slightly change to avoid a complete block density distribution restart. For simplicity, we restart the entire algorithm on membership changes and assume that disconnected nodes will never rejoin the system, which degrades the performance of our solution. We note that

Erasure Coding (EC) is likely a better pairing for PSG, however, we favor the use of block-level replication to provide fair comparisons against HDFS in Section VI-B.

V. Implementation

A. Hives simulator

We implement our solution using the Python 3.7[1] programming language, since it gives us access to powerful open-source mathematical and data science packages. To evaluate the proposed solution performance, we utilize Hives[2], an open-source simulation framework, with a GPLv3 license, for Python, developed by us. The simulator was developed because there are few well-known alternatives for this language other than NS-3 [20], which focuses on low-level problems related with the Network layer of the OSI model. Ours, on the other hand focuses on the application layer and favors quick prototyping for the generality of DFSs by implementing server, group, and individual node behaviors. The bundled classes and mechanisms may be modified or extended effortlessly using the language's inheritance and polymorphism features. Apart from providing base behaviors, the simulator provides researchers logging tools that store the simulations state on an epoch basis to disk in a JSON format for easy post-processing and analysis. All-in-all, the simulation framework decreases the number of hours researchers have to put into programming; however, it lacks the time-efficiency seen in famous competitors such as PeerSim [21] due to the shortage of data structures and procedures that favor speed. It supports at most 10^4 network nodes and has a base time complexity of $\mathcal{O}(e \times c \times n)$, which becomes $\mathcal{O}(e \times c \times n \times r)$[3] when using our abstracted PSG implementation. Other functionality provided by the simulation framework includes a module for setting up environment properties, such as message loss and disk error probability, replication level, the minimum and maximum times to recover from replica loss, among others. There is also a way of defining available nodes in the network, their uptime, and the cluster groups' size that will persist at most one file each, through the use of what we call simulation files. The documentation is available on our project's website[4].

B. Swarm Guidance based Distributed Backup System

Our Master server implementations adhere to the used simulator specifications. In particular, they act as record-keepers that indicate which storage nodes are currently online, such that clusters may find replacements for their faulty nodes. We pretend that, during the simulation, any available storage node has gone through a prior authentication process. Our storage nodes, *SGNodeExt*, inherit from the default *Node* class offered by the simulator. The same holds true for our cluster class (*SGClusterExt*), which apart from performing the simulator described roles, also emulates the initial client write to the r closest storage nodes, as well as the role of a set of Monitors who registers node complaints, deciding which suspicious members to expel from the complainers' cluster, the resulting member substitution and chain recalculation. In Algorithm 3 we provide the routines carried by these entities.

[1] https://www.python.org/downloads/
[2] https://github.com/FranciscoKloganB/hivessimulator
[3] e: epochs; c: #clusters; n: (#nodes)/cluster; r: (#blocks)/node
[4] https://www.hivessimulator.tech/

Algorithm 3 DBS algorithm implemented in Hives

upon event SIMULATE $\langle json \rangle$ **do at** SGMASTER : m
  cSize, fid, blocks ← IO.Read($json$)
  c ← m.NewClusterGroup(cSize, fid)
  trigger event SPREADFILES ⟨c, blocks⟩.
upon event SPREADFILES $\langle B \rangle$ **do at** SGCLUSTER : c
  M ← CREATEMC ⟨c⟩ ▷ Algorithm 1
  for all s **in** c.members **do**
    if s ∈ NEAREST(r) **then**
      s.files ∪ {c.fid, $\{b_1, \ldots, b_k\}$}
    end if
    trigger event DOWORK ⟨s, c, $M_{s\star}$⟩
  end for
upon event DOWORK ⟨c, $\underline{s}$⟩ **do at** SGNODE : s
  every Δt **do**
    for all b **in** s.files[c.fid] **do**
      d ← SelectNextState($\underline{s}$)
      trigger event RECEIVEPART ⟨d, c, b⟩
      async wait r **then**
        if r ∈ {OK} **then**
          s.files[fid].Delete(b)
        else if r ∈ {BADREQUEST} **then**
          trigger event REPLICATE ⟨c, b⟩
          s.files[fid].Delete(b)
        else if r ∈ {NOTFOUND, TIMEOUT} **then**
          trigger event ERR ⟨c, s, d, Δt⟩
        end if
    end for
upon event RECEIVEPART ⟨c, b⟩ **do at** SGNODE : s
  if ¬ Sha256(b.data) **then**
    return BAD_REQUEST
  else if $b \in$ s.files[c.fid] **then**
    return NOT_ACCEPTABLE
  else
    s.files ∪ {c.fid, b}
    return OK
  end if
upon event ERR ⟨n_1, n_2, Δt⟩ **do at** SGCLUSTER : c
  cid ← n_1 | n_2
  if cid ∉ c.epochComplaints(Δt) **then**
    c.epochComplaints[Δt] ∪ cid
    c.complaints[n_2] + 1
    if c.complaints[n_2] > $\frac{c.\text{size}}{2}$ **then**
      c.members.Replace(complainee)
      M ← CREATEMC ⟨c⟩
      for all s **in** c.members **do**
        trigger event DOWORK ⟨s, c, $M_{s\star}$⟩
      end for
    end if
  end if

VI. Results Discussion

We divide our study into two parts. In Section VI-A, we make the environment as easy as possible; the idea is to assess the baseline performance of the algorithm in perfect situations. In the Section VI-B, we put away the environment's easiness to understand how real-world problems may come to affect the algorithm and compare our work against a simulated HDFS. All test scenarios lasts a day with 480 epochs, i.e., an epoch occurs every three minutes, and a file is considered durable if all epochs are played. We summarize all of the different played scenarios in Table I. The table identifies scenarios by prefixing the tested system acronym[5], followed by the number of storage nodes in the simulated cluster, suffixed by a short, acronym-like, description that summarizes the key property or features that distinguish the scenario from the remainder, e.g., one hundred parts would equate to the suffix of **100P**, a test where messages can be lost in transmission would equate to the suffix of **ML**, likewise a system using optimizations would be tagged with **Opt**. We also use these identifiers in the figures that follow. From now on, any time we say that instantaneous convergence is verified at some epoch, we mean that, during the logging stage of a simulation, Eq. (3) held, where $\underline{v_s}$ is the current part density within a cluster's storage node. We may also imply that a cluster reached a goal; in this case, we consider Eq. (4) instead. Associated with the latter case, we also measure the cluster's distance to that goal, as in Eq. (5).

$$c_t \Rightarrow \left|\underline{v}_s^{(t)} - \underline{e}_s\right| \leq \min\left(\frac{1}{dim(\underline{v})}, 0.05\right) + 0.05 \times |\underline{e}_s| \quad (3)$$

$$c_{avg} \Rightarrow \left|\frac{\sum_t \underline{v}_s^{(t)}}{t_d} - \underline{e}_s\right| \leq a_{tol} + 0.05 \times |\underline{e}_s| \quad (4)$$

$$c_{dm} = \sqrt{\sum_s \left|\frac{\sum_t \underline{v}_s^{(t)}}{t_d} - \underline{e}_s\right|} \quad (5)$$

For the first batch of tests, each scenario plays 100 times. We set up the simulator environment with disk error and message loss chances to zero. The replication factor is $r = 3$, but we sometimes vary the block size. We also activate or deactivate the optimizations to MCs to investigate if they produce any practical effects. Storage nodes never fail; consequently, we run a Monte Carlo simulation with a predefined pool of $\langle \mathbf{K}, \underline{e} \rangle$ pairs designated as challenges, which can be solved by any of our three methods. This is useful, as it ensures that different scenarios run challenges of equal difficulty at the same respective playthrough $p \in [1, \ldots, 100]$. Ultimately, through the simulation file, we vary cluster sizes.

For the second part disk errors and message loss chances are possible. The replication factor is kept at $r = 3$ and we fix $b(F) = 1$MB for both *SGDBS* and *HDFS*. While the theory argues that using more parts reduces the distance to $\underline{e}$ in the long run, it does not indicate that such a desired state is reached faster. As a matter of fact, using extra parts might increase the time it takes for the first convergences to occur due to the increased distance to the goal at the start of the simulations. Consequently, using smaller block sizes and thus, having a greater number of parts in Swarm Guided Distributed Backup System (SGDBS) is not necessarily favorable for us. Also, for similarity, more resilient storage nodes will receive a bigger quota of parts to safeguard, without bounds, hence, we do not use random equilibrium vectors. We base our notion of machine resiliency purely on the time a node remains online throughout a simulation. For both SGDBS and the HDFS system, we test three different scenarios that differ with respect to storage node uptimes, $u_k(s), \forall s \in S$, hence we distinguish them by tiers. For the first scenario and tier we have $u_{T1}(s) \in [4, 32]$, correspondingly, we have $u_{T2}(s) \in [32, 64]$ and $u_{T3}(s) \in [64, 100]$. The optimizations in Eqs. (1) to (2) are always executed in these scenarios for the SGDBS simulations, meaning that, we pick the fastest $\mu(\mathbf{M})$ from the pool of available MCs when a membership change occurs. Also, due to membership changes, our Monte Carlo simulations have 500 samples rather than 100, to reduce result variance. Finally, fault detection is not immediate and depends on the running protocol, i.e., complaints *vs.* heartbeats. Once detected, if they concern disconnected storage nodes, t_{snr}, an immediate replacement takes place; otherwise, if they concern lost file block replicas, these will take three to nine minutes, i.e., $t_{brr} \in [1, 3]$ epochs, to be possibly restored to no more than r.

A. Swarm Guidance on Hives

One issue associated with the use of PSG in a DBS would be the bandwidth consumption, comparing with other approaches that do not regularly change the location of file parts as a feature of the underlying algorithm. Fig. 2 presents the bandwidth consumption for simulations with different number of parts. Irrespective of this choice, the members exchange $\approx$ 80MB on average in messages containing file block replicas at every epoch, excluding Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) headers, message fields like block identifying data, response and complaint messages. A clear disadvantage compared to HDFS, which may be relevant in some scenarios. When implemented with storage nodes (peers) spread worldwide, link saturation is unlikely. However, if all nodes are in the same building and multiple clusters exist, issues may arise not only in the DBS but also in other systems not associated with the backup service. On the other hand, HDFS and likely GFS, only perform checksum verifications when a client retrieves a block from a remote DataNode. This means that file corruption due to faults in a storage device, the network, or the software is unknown until a client accesses his files, which can be infrequent. By verifying checksums whenever forwarding the parts, the chance of losing files to corruption is minimized. Another advantage is the self-

[5] SG: Swarm Guidance algorithm tested on a perfect environment;
SGDBS: Swarm Guided Distributed Backup System;
HDFS: Hadoop Distributed File System;

healing property of PSG, which allows clusters to recover from faults transparently. Depending on the used topology, permanent faults are also detected and dealt with quicker (Fig. 3). Tweaks to reduce bandwidth consumption include: a Gia-like satisfaction metric; adjusting epoch-length statically or dynamically; not allowing parts received at some storage node at some epoch to be considered for routing before the next epoch; using EC instead of block-level replication.

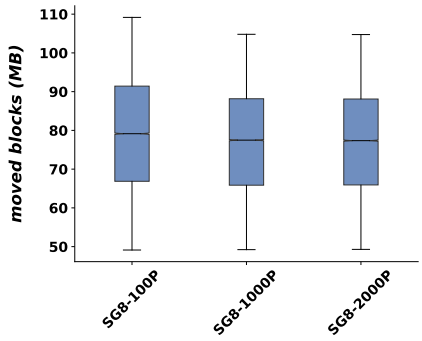


Fig. 2: Bandwidth consumption, per epoch, with 45MB files.

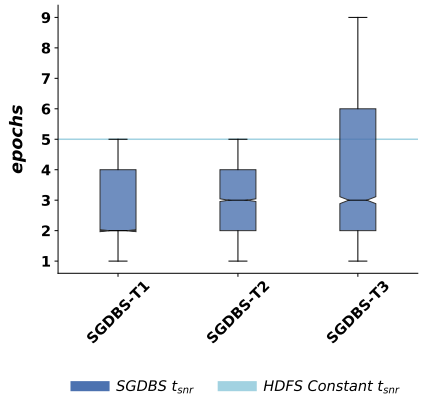
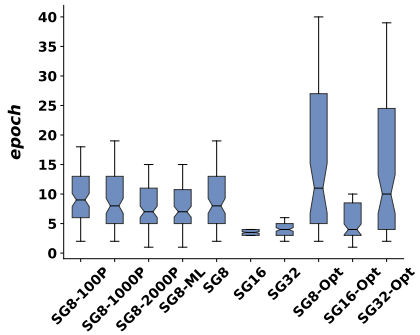
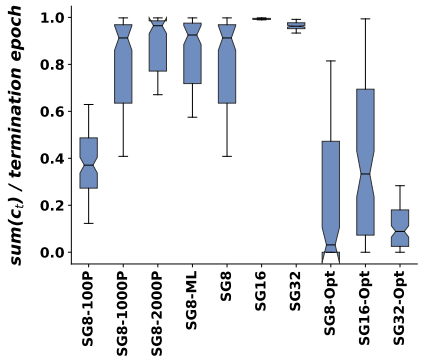


Fig. 3: Epochs required to detect/replace faulty nodes.

Since the number of parts in the system does not appear to influence bandwidth consumption, the next step is to determine how many parts should exist in the cluster. An increased number of parts implies that somewhere in the system, centrally or otherwise, more metadata will exist. Our solution states that master servers map clients' file identifiers to clusters containing the blocks without tracking which storage nodes have which blocks, thus eliminating a big chunk of queries the masters would otherwise receive. However, the masters or some other entity in the system must maintain for each existing file the hash values associated with its blocks, to check for disk and message corruption. Consequently, increasing the number of parts without bounds is undesirable, as the metadata file would occupy more space. Furthermore, when using EC to minimize bandwidth and storage node disk usage, the reconstruction of lost replicas could become slow (more parts to collect and decode), which endangers durability. Our results indicate that the number of parts does



(a) epoch at which c_t boolean condition was first verified.

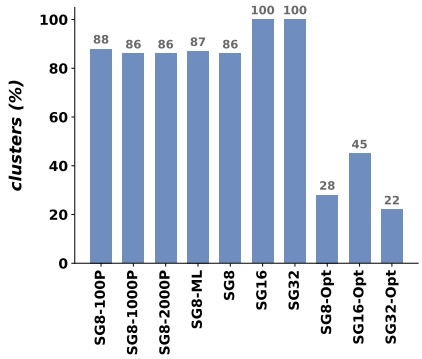


(b) clusters lifetime (%) where c_t boolean condition was verified. In this case termination epoch is fixed for all cases.

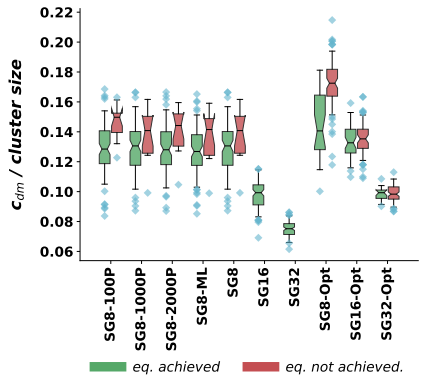
Fig. 4: Overview of instantaneous convergence (c_t) behavior.

not affect the time it takes to witness the first convergences (Fig. 4) in a cluster. Truthfully, all scenarios had very satisfying results taking what equates to something between 30 to 90 minutes to achieve the desired configuration for the first time. Conversely, using approximately ≈ 1000 parts produces the best balance between the number of blocks that require tracking, the witnessed instant convergences, and the number of times the goal equilibrium is achieved, on average, as well as the distance to that goal (Fig. 5), including the cases where the goal remains unachieved.

As expected, there is a tendency for the first convergences to occur relatively early in a cluster lifecycle and for the number of observed instantaneous convergences to increase as simulations progress. Fig. 6 shows that the number of occurrences tends to become bounded after a certain point in time. In the aforementioned figures, the inequalities used to declare convergence Eqs. (3) to (5) cause clusters with 16 and 32 members to be faster as the entries in the distribution vector are smaller values. If other functions are used results could differ. An example would be using only the relative tolerance in Eq. (3), i.e., considering only the amount of parts in the cluster, never considering the number of nodes. From these simulations, the use of bigger networks contributes to better load-balance for the same replication factor as well as fewer storage node isolation situations, i.e., they make for more resilient swarms, greatly reducing the number of

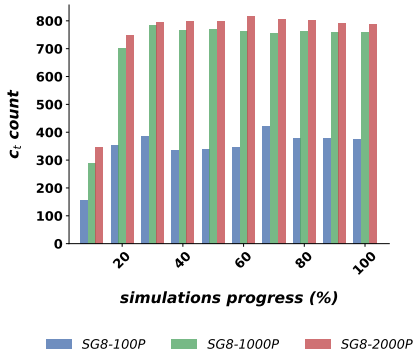


(a) percentage of clusters that verified the boolean condition c_{avg} at the end of a playthrough.

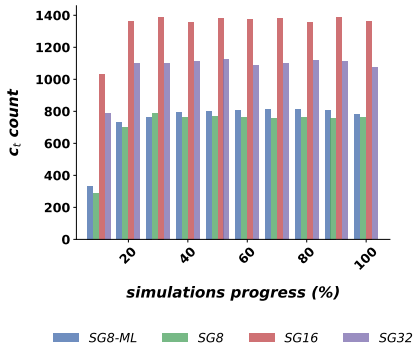


(b) clusters distance to the goal at the end of a playthrough, based on Eq. (5) (c_{dm}).

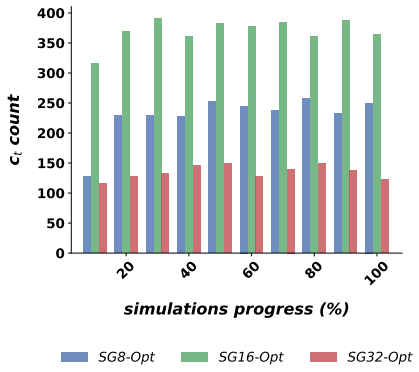
Fig. 5: Clusters (%) achieving or not achieving the desired equilibrium and registered distance to that goal.



(a) varying only the number of parts



(b) varying cluster size, or alternatively allowing for messages to be lost.



(c) varying cluster size with M optimizations activated.

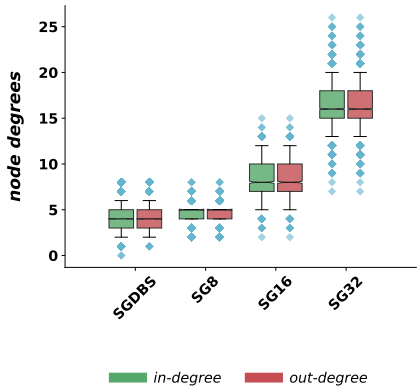
Fig. 6: Simulations' instantaneous convergence distribution.

outliers in Fig. 3. Another unexpected behavior regards the optimization algorithms used to produce MCs with better SLEM. We expected results to be better than the ones obtained using MH when it comes to reaching and stabilizing around the desired equilibrium. However, they were strictly worse. This is an issue that requires further research. Lastly, message-loss seems irrelevant concerning the studied properties, including time taken to converge, the number of instantaneous convergences, and goal achievement. In a real-world scenario, they may impact system availability and reliability and even more so the durability of the files, but we do not concern ourselves with that in this subsection.

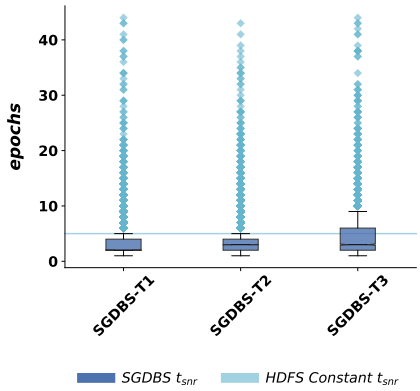
B. SGDBS vs HDFS

Our solution outlines a DBS made of commodity nodes that are not guaranteed to remain online for long periods, where the system users either contribute to it as storage nodes or use the available space as file uploaders. Files hosted in the service are not intended to be modified frequently, nor in parallel; rather, the central purpose of the system is to persist critical user files for arbitrarily long periods in dynamic networks, at a low cost, with little centralization. Consequently, we focus on file survivability metric.

In P2P overlays, churn, and its consequences are one of the most significant dangers to file durability; that explains why so much overlay research focuses, among other topics, on keeping the networks connected and timely failure detection. We extrapolate from surveys cited in Section II that robustness of overlays is tied node degrees. In our case, we do not need to concern ourselves with node clusters or hubs because our solution is not gossip-based. However, we still want to avoid weakly connected storage nodes, particularly regarding the number of receive channels they have. We explain the outliers in Fig. 7(b), essentially with the node degrees seen in Fig. 7(a). Even though our system often beats HDFS concerning the time it takes to detect and replace failed nodes, a median in-degree of four, sometimes as low as two,

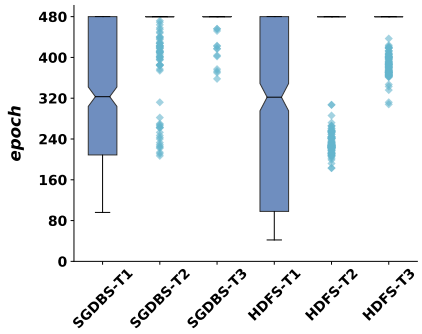


(a) Node degrees with and without optimizations.

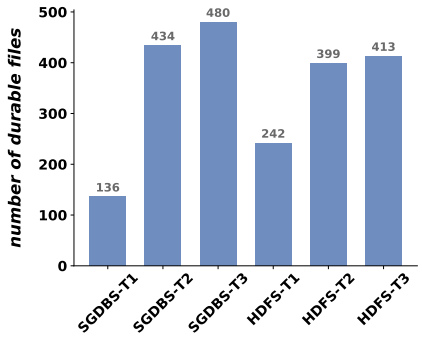


(b) Time to replace faulty nodes, including outliers.

Fig. 7: Node degrees resulting from different configurations and their impact on t_{snr}.



(a) simulations termination epochs distribution.



(b) simulations reaching the maximum number of epochs, thus guaranteeing durability.

Fig. 8: Data respecting termination epochs and playthroughs which successfully ensured file durability.

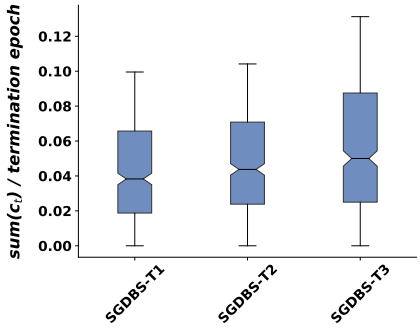
implies that it takes only three node failures for some other to become isolated, a grim scenario when our main objective is to guarantee the durability of uploaded files regardless of peer uptime. Apart from that, new topologies give no guarantee that a failed, yet undetected, node will have a good in-degree in the next chain, which in turn, might postpone its detection even further sometimes, indefinitely. This situation is particularly relevant when clusters suffer a high churn. Our approach, contrary to the expected, was vastly inferior to the one employed by HDFS on T1 machines, even when they do not proactively defend against disk errors and file block corruption. The termination epoch value for HDFS-T1 is more dispersed than in SGDBS-T1; the fact remains that the former still ensured the durability of files 1.77 more times than the latter. SGDBS-T2 and SGDBS-T3 had better results but did not win versus their HDFS counterparts by astoundingly large margins. Ultimately both systems failed in providing durability for all of the 500 tests.

Finally, we reinforce the value of adaptively changing the topology and the goal, if at all, when cluster membership changes occur, something we did not do as expressed in Section IV. The result of such hindsight is visible in Fig. 9. For the same number of file block replicas in a cluster, the time clusters spent in instantaneous convergence is approximately ten times worse than the clusters that used optimizations and whose members had 100% uptime. The distances to the goal also increased slightly because every member's departure meant restarting the Markov process from a system state with a possibly lousy initial file distribution and finally, due to the reduced time the storage node swarms had to achieve the goal, only $\approx 4.33\%$ of the clusters achieved desired equilibrium, on average.

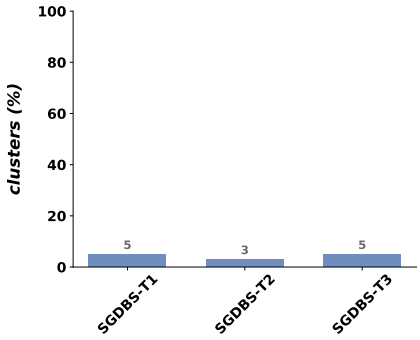
VII. Conclusion and Future Directions

This paper addresses the introduction of PSG to DBSs. Our proposal imposes a high tax on network bandwidth, and not all of the results were in line with the expectations. The sought purpose of ensuring 100% durability was unachieved, but we still outperformed HDFS [9]. Thus, the solution merits further investigation. Some issues remain open. In particular, future research endeavors may include:

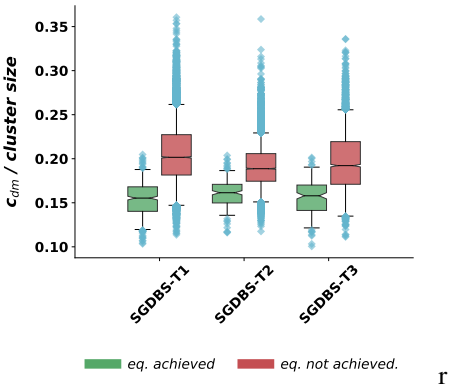
- Study the effectiveness of the solution using fountain-code EC as the file redundancy model.
- Given a set of available storage nodes, form a group that minimizes the probability of file loss.
- Adapt the solution such clusters' goals and MCs are not altered on a membership change occurs; or alternatively, minimize the magnitude of alteration.
- Further decentralize the algorithm giving more responsibility to storage nodes, i.e., collectively or independently decide MCs to follow.



(a) clusters lifetime (%) where c_t boolean condition was verified.



(b) percentage of clusters that verified the boolean condition c_{avg} at the end of a playthrough or after a membership change.



r

(c) distance to the equilibrium goal whenever c_{avg} condition was verified, using Eq. (5)

Fig. 9: Impact of changing MCs on node replacement.

REFERENCES

[1] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[2] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[3] B. Açikmeşe and D. S. Bayard, "A markov chain approach to probabilistic swarm guidance," in *2012 American Control Conference (ACC)*, June 2012, pp. 6300–6307.

[4] B. Açıkmeşe and D. S. Bayard, "Probabilistic swarm guidance for collaborative autonomous agents," in *2014 American Control Conference*, June 2014, pp. 477–482.

[5] N. Demir and B. Açıkmeşe, "Probabilistic density control for swarm of decentralized on-off agents with safety constraints," in *2015 American Control Conference (ACC)*, July 2015, pp. 5238–5244.

[6] J. M. Lien and E. Pratt, "Interactive planning for shepherd motion," *AAAI Spring Symp. Agents Learn Human Teachers*, March 2009.

[7] E. Masehian and M. Royan, "Cooperative control of a multi robot flocking system for simultaneous object collection and shepherding," *Studies in Computational Intelligence*, vol. 577, pp. 97–114, January 2014.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 29–43, October 2003. [Online]. Available: https://doi.org/10.1145/1165389.945450

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, June 2010.

[10] Red Hat. (2016) Architecture — ceph documentation. Accessed 1st July 2020. [Online]. Available: https://docs.ceph.com/docs/master/architecture/

[11] X. Zhang, S. Gaddam, and A. T. Chronopoulos, "Ceph distributed file system benchmarks on an openstack cloud," *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, November 2015.

[12] M. Selvaganesan and M. A. Liazudeen, "An insight about glusterfs and its enforcement techniques," in *2016 International Conference on Cloud Computing Research and Innovations (ICCCRI)*, 2016, pp. 120–127.

[13] L. Zhang, Y. Wu, R. Xue, T. Hsu, H. Yang, and Y. Chung, "Hybridfs — a high performance and balanced file system framework with multiple distributed file systems," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2017, pp. 796–805.

[14] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," *Proceedings of first international workshop on peer-to-peer systems*, vol. 55, p. 53–65, April 2002.

[15] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," *Computer Communication Review*, vol. 33, August 2003.

[16] B. Cohen, "Incentives build robustness in bittorrent," *Workshop on Economics of Peer-to-Peer systems*, vol. 6, June 2003.

[17] A. Forestiero, C. Mastroianni, and M. Meo, "Self-chord: A bio-inspired algorithm for structured p2p systems," *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, June 2009.

[18] A. Brocco, A. Malatras, and B. Hirsbrunner, "Enabling efficient information discovery in a self-structured grid," *Future Gener. Comput. Syst.*, vol. 26, no. 6, pp. 838–846, June 2010.

[19] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM Review*, vol. 46, March 2003.

[20] U. of Washington NS-3 Consortium (nsnam). (2011) About — ns-3. Accessed 14th July 2020. [Online]. Available: https://www.nsnam.org/about/

[21] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 2009, pp. 99–100.

APPENDIX A - TABLES

TABLE I: Tested scenarios' configurations.

Scenario	System	Optimized	Cluster Size	Blocks	Replication	Link Loss	Disk Error	Node Uptimes	Plays
SG8-100P	SG[6]	No	8	33		No	No	100	100
SG8-1000P	SG	No	8	333		No	No	100	100
SG8-2000P	SG	No	8	666		No	No	100	100
SG8-ML	SG	No	8	333		Yes	No	100	100
SG8	SG	No	16	333		No	No	100	100
SG16	SG	No	16	333		No	No	100	100
SG32	SG	No	32	333		No	No	100	100
SG8-Opt	SG	Yes	8	333	3	No	No	100	100
SG16-Opt	SG	Yes	16	333		No	No	100	100
SG32-Opt	SG	Yes	32	333		No	No	100	100
SGDBS-T1	SGDBS[7]	Yes	8	46		Yes	Yes	[4, 32], $\mu = 18$, $\sigma = 8$	500
SGDBS-T2	SGDBS	Yes	8	46		Yes	Yes	[32, 64], $\mu = 48$, $\sigma = 8$	500
SGDBS-T3	SGDBS	Yes	8	46		Yes	Yes	[64, 100], $\mu = 82$, $\sigma = 8$	500
HDFS-T1	HDFS[8]	-	8	46		Yes	Yes	[4, 32], $\mu = 18$, $\sigma = 8$	500
HDFS-T2	HDFS	-	8	46		Yes	Yes	[32, 64], $\mu = 48$, $\sigma = 8$	500
HDFS-T2	HDFS	-	8	46		Yes	Yes	[64, 100], $\mu = 82$, $\sigma = 8$	500

[6]Swarm Guidance Algorithm (perfect environment)
[7]Swarm Guided Distributed Backup System
[8]Hadoop Distributed File System