Enhancing fault tolerance of distributed R-tree

Mathieu Valero, Luciana Arantes, Maria Potop-Butucaru, Pierre Sens

LIP6 - University of Paris 6 - CNRS - INRIA
4 place Jussieu, 75006 Paris, France
[FirstName.LastName]@lip6.fr

Abstract

Distributed R-trees (DR-trees) are appealing infrastructures for implementing range queries, content based filtering or k-NN structures since they inherit the features of R-trees such as logarithmic height, bounded number of neighbors and balanced shape. However, they are crash-sensitivite since each single crash can potentially break the tree structure connectivity. In this article, we present a fault tolerant approach which exploits replication of non leaf nodes ensuring the tree connectivity in presence of crashes. Our contribution is twofold. First, we enhance the connectivity without modifying the R-tree structure. Second, via extensive simulations we prove that our approach drastically reduces the cost of both message traffic and stabilization time when compared to the original approach proposed in [3] where all nodes of a disconnected subtree are reinserted. Finally, our approach can be easily extended to other crash-sensitive structures.

I. Introduction

Tree-based structures have been extensively used in computer science since its early days: in graph theory, to represent hierarchical structures, to store/cache/index data or to connect physically or virtually entities of a network. The nice property of trees is that their recursive definition of trees can often be used to prove properties much easily than with other structures. Moreover, many common and useful algorithms dealing with graphs are related to their diameters and the use of height-balanced trees ensures logarithmic properties.

R-trees [5] are a class of trees with nice features: they are height-balanced and support multi-dimensional spatial filters. That is, they can handle objects with a polyspace rectangle representation (e.g. two dimensional queries, transactions that span on several objects). DR-trees are the P2P distributed extension of classical R-tree structure. They have been introduced in [3] in order to construct peer-to-peer overlays optimized for selective and efficient dissemination of information. They have been exploited recently in publish/subscribe systems [3], [2], [8]. The system subscribers, distributed over the P2P network, organize themselves in a virtual DR-tree based on the semantic relation between their subscriptions.

However, DR-trees are highly sensitive to crash faults. Indeed, each single crash potentially breaks the connectivity of the whole structure. The crash of one or more nodes may lead to loss of data and references and thus the disconnection of its subtrees in the DR-tree. In [3], in case of node failures, the DR-tree is restored by explicitly increasing the upward connectivity of nodes. When a peer detects a crash, it broadcasts a message to its descendants that will exploit upward links pointing to a still connected part of the structure to reinsert themselves in the DR-tree. This approach is very costly in terms of messages and stabilization time. Furthermore, it may modify the R-tree structure. Our approach exploits the semantic organization of the R-trees. More precisely, the R-tree structure supports leaf deletion without any extra cost however non leaf nodes deletion may completely damage the structure. This is also true for

the extended DR-tree. If a peer holding only a leaf crashes, this leaf can be deleted without altering the R-tree structure. On the other hand, if a peer holding one or more internal nodes of the R-tree crashes those nodes have to be restored. Therefore, in our approach, each non leaf node is just replicated on each other peer that holds one of its children. Such a replication ensures that the restoration of a node n concerns only the peers that keep a replica of n. We prove via simulations that this strategy significantly outperforms the original DR-trees [3] in terms of message traffic and stabilization time.

Note that the replication mechanism we propose in this paper can be applied to other crash-sensitive logical structures such as other tree structures or rings.

The rest of the paper is organized as follows. Section II comments on the related work while section III presents the R-tree and DR-tree concepts. In section IV, we introduce the replication technique further used to restore the DR-tree connectivity. Sections V and VI respectively evaluate the replication cost and the impact of multiple crashes on the modified DR-trees. Finally, the last section concludes the work and discusses some future work.

II. Related work

Distributed R-trees were introduced in [3]. The original paper discusses some strategies to make the structure fault tolerant. Each node stores "some knowledge" materialized as virtual links to its ancestors (starting with its grand-father up to the root). Such upward links can be obtained by sporadically walking toward the root or by randomly piggy backing nodes information during routing. When a peer detects a crash, it broadcasts a message to its descendants that will reinsert themselves in the valid part of the structure. To this end, each peer contacts the root to reinsert itself. For instance, in Figure 1, if $p7$ fails, $p8$ will contact $p1$ (the peer root) to reinsert $n11$ and $p9$ will contact $p1$ to reinsert $n12$. The closer from the root the crash occurs, the more expensive is the structure restoration. With N peers, in the worst case, this algorithm requires $log_m(N)$ $N/2$ to restore the system; up to $N/2$ peers have to be reinserted and each reinsertion generates in average $log_m(N)$ messages. Moreover, in such case there is a risk that the traffic generated by reinsertions overloads other peers.

BATON [6] and VBI [7] are two frameworks based on balanced binary trees. Virtual nodes are divided in two categories: leaf nodes and non leaf nodes. Basically, the former are used for storage while the latter are used for routing. A structural property of AVL ensures that there are roughly as much leaves as internal nodes: every physical node holds one leaf and one internal node. Furthermore, each node in the tree maintains links to its parent, children, adjacent nodes and selected nodes which are nodes at the same level. The latter are chosen during node insertion and are mainly used to balance routing load. At the same time, the left-right links also provide fault tolerance since they can be used as shortcuts to reconstruct missing parent-child links.

In P-GRID [1] each peer stores a part of the overall tree. Every peer's position is determined by its path, i.e., the binary bit string representing the subset of the tree's overall information for which the peer is responsible. For fault-tolerance, multiple peers can be responsible for the same path. A sampling-based method is used to detect imbalance and dynamically adapt replication.

Caron et al. [4] propose a fault tolerance protocol that reconnects subtrees after crashes in order to have again a connected graph and then reorder the nodes to build a consistant tree. In other words, the protocol consists of a recovery phase followed by a reorganization phase. Our approach is different since we do not need a reorganization phase which results in gain of performance.

III. Background

In this section we recall some generic definitions and the main characteristics of the DR-trees [3] overlay.

A. Distributed R-trees

R-trees [5] are height-balanced tree handling objects whose representation can be circumscribed in a poly-space rectangle. A R-tree is characterized by the following structural properties:

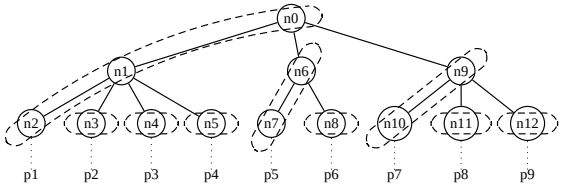
- The root has from 2 to M children
- Every internal node has from m to M children ($m \leq M/2$)
- All leaves are at the same level

Distributed R-trees [3] (DR-trees) extend the R-tree index structures where peers are self-organized in

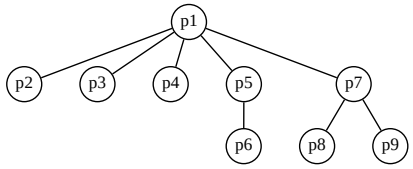
a balanced virtual tree overlay based on semantic relations. The structure preserves the R-trees index structure features: bounded degree per node and search time logarithmic in the size of the network. Moreover, the proposed overlay copes with the dynamism of the system.

Physical machines connected to the system will be further referred as *p-nodes* (shortcut for physical nodes). A DR-tree is a virtual structure distributed over a set of p-nodes. In the following, terms related to DR-tree will be prefixed with "v-". Thus, DR-trees nodes will be called *v-nodes* (shortcut for virtual nodes). The root of the DR-tree is called the *v-root* while the leaves of the DR-tree are called *v-leaves*. Except the v-root, each v-node n has a v-father (*v-father(n)*), and, if it is not a v-leaf, some v-children (*v-children(n)*). These nodes are denoted *v-neighbors* of n.

The physical interaction graph defined by the mapping of a DR-tree to $p - nodes$ of the system is a communication graph where there is a $p-edge\ (p, q)$, $p \neq q$, in the physical interaction graph if: there is a v-edge (s, t) in the DR-tree, p is the p-node holding v-node s, and q is the p-node holding v-node t.



(a) A distribution of a given R-tree



(b) Corresponding physical interaction graph

Fig. 1: A distribution of a given DR-tree and its corresponding physical interaction graph

Figure 1 shows a representation of a DR-tree composed of v-nodes $\{n0, \ldots, n12\}$ mapped on p-nodes $\{p1, \ldots, p9\}$. Dashed boxes represent nodes distribution. There is a p-edge $(p1, p5)$ in the interaction graph because there is a v-edge $(n0, n6)$ in the DR-tree, $p1$ is the p-node holding v-node $n0$, and $p5$ is the p-node holding v-node $n6$.

The key points in the construction of a DR-Tree are the join/leave procedures. When a p-node joins the system, it creates a v-leaf. Then the p-node contacts another p-node to insert its v-leaf in the existing DR-tree. During this insertion, some v-nodes may split and then Algorithm 1 is executed.

Algorithm 1 $void$ onSplit(n:VNode)

1: **if** $n.isVRoot()$ **then**
2: $\quad newVRoot = n.createVNode()$
3: $\quad n.v - father = newVRoot$
4: **end if**
5: $m = selectChildIn(n.v - children)$
6: $newVNode = m.createVNode()$
7: $n.v - children, newVNode.v - children = divide(n.v - children)$
8: $newVNode.v - father = n.v - father$

Distribution invariants: The following two properties are invariant in the implementation of DR-tree proposed in [3]:

- $Inv1$: each p-node holds exactly one v-leaf
- $Inv2$: if p-node p holds v-node n, either n is a v-leaf or p holds exactly one v-children of n

We denote the top and $bottom$ v-node of a p-node the v-node which is at the top and bottom of the chain of v-nodes kept by the p-node respectively. The above invariants ensure that the communication graph is a tree:

- The p-root is the p-node holding the v-root
- A p-node p is the p-father of the p-node q (*p-father(q)*) if p holds the v-father of the v-node at the top of the chain of v-nodes held by q

For instance in Figure 1a, *p-father(p5) = p-father(p7) = p1*. The above distribution invariants also guarantee that p-nodes have a bounded number of p-neighbors. In a system with N p-nodes and a DR-tree with degree $m : M$, the DR-tree height is $log_m(N)$; the p-root holds $log_m(N)$ v-nodes. Since each v-node has up to M v-neighbors, the p-root may have up to $M\ log_m(N)$ p-neighbors.

IV. Fault tolerance

As explained in the Related Work section, in the classical DR-tree implementation [3], when a p-node fails all its subtrees are reinserted in the non-faulty structure (p-node by p-node) in order to guarantee

the DR-tree invariants. We propose a novel strategy which both ensure fault-tolerance and preserves the DR-tree structure and its invariants by exploiting the non leaf v-nodes replication.

The pattern for v-nodes replication is:

- the p-root holds no replica;
- each p-node holds a replica of the v-father of its top v-node.

Therefore, each non leaf v-node is replicated on each p-node holding one of its v-children. For a DR-tree of degree $m : M$ ($m \geq 2$), the v-root is replicated on 1 to M-1 p-nodes while each internal v-node is replicated on m-1 to M-1 p-nodes.

In Figure 1a, the DR-tree has four non leaf v-nodes; $n0$, $n1$, $n6$ and $n9$. The following table shows on which p-nodes they are replicated:

internal v-node	replicated on p-nodes
$n0$	$p5$, $p7$
$n1$	$p2$, $p3$, $p4$
$n6$	$p6$
$n9$	$p8$, $p9$

Therefore, the restoration of a given v-node n concerns only those p-nodes holding a replica of n, i.e, the $m-1$ to $M-1$ p-nodes which hold v-children of n. Moreover, the distribution invariants ensure that with N p-nodes, no p-nodes holds more than $\lfloor log_m(N) \rfloor$ v-nodes. For instance, in Figure 1a, if $p1$ fails, the internal v-nodes $n0$ and $n1$ have to be restored. The former concerns $p5$ and $p7$ while the latter concerns $p2$, $p3$, and $p4$.

Upon the detection of the failure of a p-node p, every p-node holding the leftmost replica of each v-node, previously held by p, restores the v-node. For instance, in Figure 1a, if $p1$ crashes, $n0$ will be restored by $p5$ while $n1$ will be restored by $p2$, the p-nodes that keep the leftmost replica of $n0$ and $n1$ respectively. Note that the physical interaction graph changes with restoration but the DR-Tree structure is kept unchanged. Moreover, it is not necessary to reorganize the nodes after the restoration phase which makes the strategy effective in terms of message and duration.

V. DR-tree restoration cost

This section investigates the cost of replication. Then it compares the cost of system reparation facing a single crash using replication and the classical reinsertion strategy, proposed in [3].

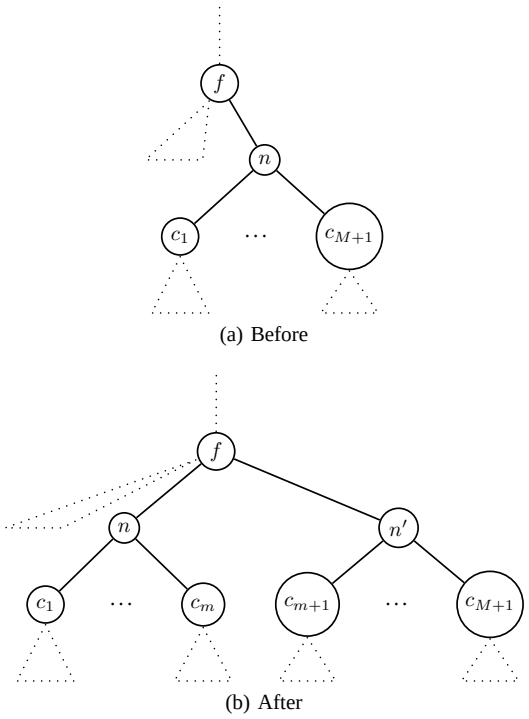


Fig. 2: R-tree split of v-node n

A. Replication cost

Replicas are created at join operations and modified both at join and split operations. When a v-node is modified its holder should notify the p-nodes which hold its replicas.

In the sequel, we consider a DR-tree of N v-nodes with degree $m : M$. We also assume that the cost of updating one replica is one message.

Figure 2 illustrates what happens in the DR-tree when a v-node n splits. Basically n creates a new v-node n' and delegates half of its children to n'. n' is added to the children of n's v-father, i.e. f's children. In its turn, if f gets more than M children, it splits according to the same algorithm.

In order to evaluate the cost of replica updates, it is also necessary to calculate how many v-nodes are modified during a split operation. Figure 2 shows that when a v-node n splits, its father, half of its children and n itself are modified. Therefore, since a node splits when it has exactly M v-children, each

split modifies $m+2$ v-nodes. Let $cost_s$ be the cost of updating replicas during a split. Since, in this case, each v-node has $M-1$ replicas and that each update costs one message, we have:

$$cost_s = (m+2)(M-1)$$

A p-node joining the system may trigger between 0 and $\lfloor log_m(N) \rfloor$ splits. Its v-leaf is added to the v-children of another v-node that we denote in the sequel the *joined v-node*. The latter has:

- between m and M v-children;
- $\lceil log_m(N) - 1 \rceil$ v-ancestors;
- between $m-1$ and $M-1$ replicas.

In the following calculation we will define an upper bound on the number of updates considering that each v-node has $M-1$ replicas.

A v-node may have m to M v-children and thus has $M-m+1$ possible numbers of v-children. It splits only when it has exactly M v-children. The probability p for a v-node to split is:

$$p = \frac{1}{M-m+1}$$

The probability for a p-node to trigger k splits is the probability p_k that the joined v-node and its $k-1$ first v-ancestors have exactly M v-children while its k-th ancestor does not split. Hence:

$$p_k = p^k(1-p)$$

Let $cost_r$ be the average cost of replicas updates when a p-node joins a DR-tree. We have:

$$cost_r = \underbrace{p_0(M-1)}_{no\ split} + \underbrace{\sum_{k=1}^{\lfloor log_m(N) \rfloor} (p_k\ k\ cost_s)}_{some\ splits}$$

The first term corresponds to the case where no splits are triggered, i.e., $M-1$ replicas of the joined v-node are to be updated. The second term corresponds to the other cases.

We could have distinguished the case where the v-root splits. Its splitting probability is different since it has $M-1$ possible v-children. However, for $m > 2$, this probability is smaller than p so we just give an upper bound in order to simplify the calculation.

B. Performance evaluation

In the following, we respectively denote *reinsertion policy* and *replication policy* the approach which uses DR-Tree insertion operations and the one that we propose, which uses internal v-nodes. The first one was introduced in [3] and is described in section II.

In order to compare the two policies, we used an ad hoc discret simulator. We ran 1,000,000 simulations. Each one built a DR-tree of degree $2:4$ that was distributed over 1000 p-nodes. For each DR-tree, the crash of one non leaf p-node was generated. We then measured the cost of system restoration in terms of number of messages and stabilization time in both the reinsertion and replication policies.

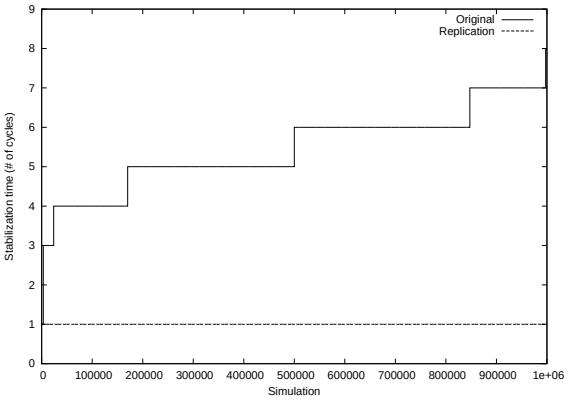


Fig. 3: Stabilization time

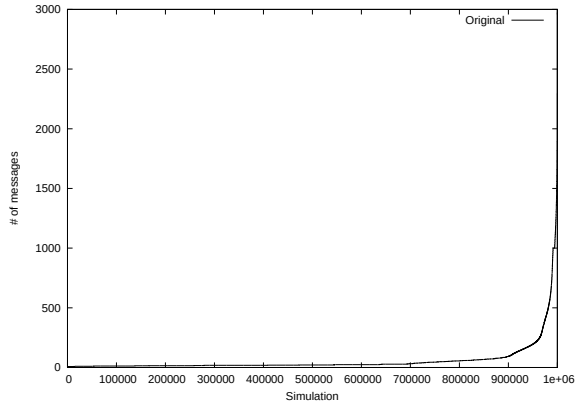
1) Stabilization time: In Figure 3, simulations are ordered by stabilization time. Y-axis indicates the time required to stabilize the system from a non leaf p-node crash.

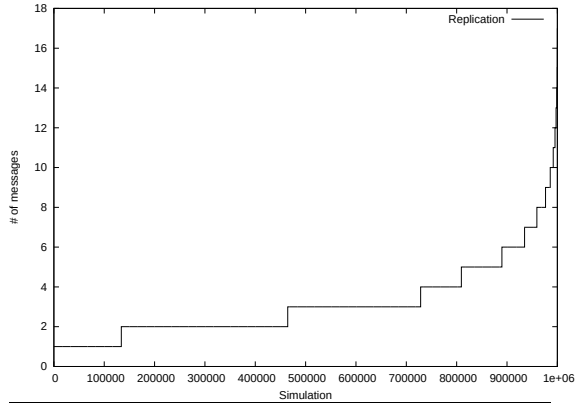
The reinsertion mechanism stabilizes the system in a number of cycles which is proportional to the height of the interaction graph. It is also related to the level of the crashed p-node in the interaction graph. Indeed, the stabilization time is the time of the longest reinsertion, that is, it is proportional to $log_m(N)$.

On the other hand, with the replication policy, the stabilization time is constant and independent of the faulty p-node: v-nodes restoration are done concurrently and, once the crash is detected, only one cycle is required to restore them.

2) The message cost of the restoration phase: In figure 4, simulations are ordered by the number of messages. Y-axis indicates the number of messages required to stabilize the system from a non leaf p-node crash. We observe that the costs are of different magnitude.

With the reinsertion policy, the number of message





	min	max	avg	stddev
Original	6	2964	57.93	141.181
Replication	1	17	3.1347	1.94967

Fig. 4: Number of messages to restore DR-tree

distribution is very skewed which results in a high standard deviation. The closer to the p-root the faulty p-node is, the more expansive DR-tree restoration is. The best case is when the crashed p-node has only p-leaves as p-children. Then the cost is proportional to the height of the communication graph ($log_m(N)$). The worst case is when the faulty p-node is close to the p-root. Up to half of p-nodes have to be reinserted, leading to a cost in number of messages proportional to $log_m(N)\ N/m$.

However, with the replication strategy, the best case cost is $m-1$ messages because only replicas of the *joined v-node* have to be updated. The worst case cost is when a p-node holding $log_m(N)$ v-nodes crashes. Then M replicas of each v-node has to be updated, leading to a cost in the number of messages proportionnal to $log_m(N)\ M$.

VI. Impact of multiple crashes on the DR-tree connectivity [6]

The previous section focuses on the system tolerance of a single crash. We now investigate by extensive calculations what happens when multiple crashes occur simultaneously. In the same way that we distinguish p-nodes from v-nodes, we distinguish also crashes from losses. The former are related to p-nodes while losses are related to v-nodes.

In the sequel, $\tilde{p}$ is true if the p-node p is alive, false if p has crashed. Let V be the binary vector storing the states of p-nodes. For exemple on figure 1:

$$V = (\tilde{p1}, \tilde{p2}, \tilde{p3}, \tilde{p4}, \tilde{p5}, \tilde{p6}, \tilde{p7}, \tilde{p8}, \tilde{p9})$$

We consider a system with N peers (and thus N v-leaves). Let N' be the number of internal v-nodes. Let Ω be the set of possible V and Ω_k be the set of V related to k losses. k may vary from 0 when no v-node is lost to $N'+1$ when the v-root and all internal v-nodes are lost. Thus:

$$\Omega = \bigsqcup_{i=0}^{N'+1} \Omega_i$$

A. Configuring the System

We virtually order the bits of V. The p-root does not hold any replica. Its state is represented in the first bit of the vector. As each other p-node holds exactly one replica, we can virtually order remaining V bits according to held replicas: v-nodes are firstly ordered by a prefix walk of the DR-Tree and then, for a given v-node, its replicas are ordered from left to right. For exemple on Figure 1:

$$V = (\tilde{p1}, \underbrace{\tilde{p5}, \tilde{p7}}_{n0}, \underbrace{\tilde{p2}, \tilde{p3}, \tilde{p4}}_{n1}, \underbrace{\tilde{p6}}_{n6}, \underbrace{\tilde{p8}, \tilde{p9}}_{n9})$$

We argue that for a complete DR-tree, V has a structure that simplifies the calculations. Indeed, for a complete DR-Tree of degree d we have:

$$V = (\underbrace{v_1, \underbrace{v_2, \ldots, v_d}_{root\ replicas}}_{root}, \underbrace{\underbrace{v_{d+1}, \ldots, v_{2d}}_{}, \ldots, \underbrace{v_{N-d-1}, \ldots, v_N}_{}}_{d-1\ bits\ subvectors})$$

For a given degree $m : M$ and a given number of peers N, numerous different DR-tree can be built. It is worth pointing that they are not all complete. However, some relevant configurations can be modeled

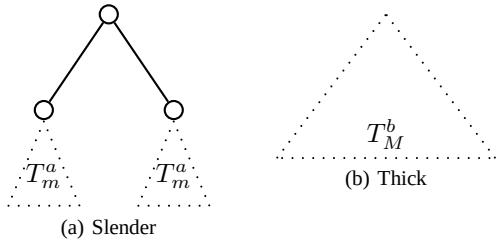


Fig. 5: Revealant DR-tree configurations

based on complete DR-trees. Let T_d^h be the complete DR-tree of degree d and height h.

Both DR-trees of Figure 5.(a) and Figure 5.(b) have the same number of v-leaves. They illustrate extreme case configurations obtained with the highest and lowest bounds on v-nodes degrees (see the Rtree definition). The "slender" tree (Figure 5.(a)) has the maximum number of internal v-nodes where each one has the minimum number of children. The v-root has two children and all other internal v-nodes have m children. Remark that the v-root's children are complete subtrees of degree m. Therefore:

$$V_s = (\underbrace{v_1, v_2}_{root}, \underbrace{v_3, \ldots, v_{m-2}}, \ldots, \underbrace{v_{N-m+1}, \ldots, v_N}) \quad \underbrace{\qquad\qquad}_{m-1\ bits\ subvectors}$$

The "thick" tree (Figure 5.(b)) has the minimum number of internal v-nodes where each one has the maximum number of children. The DR-tree is a complete tree of degree M. Therefore:

$$V_t = (\underbrace{v_1, \ldots, v_M}_{root}, \underbrace{v_{M+1}, \ldots, v_{2M}}, \ldots, \underbrace{v_{N-M+1}, \ldots, v_N}) \quad \underbrace{\qquad\qquad}_{M-1\ bits\ subvectors}$$

In both cases V is composed of two distinct parts: its first bits are related to the v-root while the rest is composed of constant size subvectors related to internal v-nodes.

We should point out that the greater the number of internal v-nodes a DR-tree contains, the fewer the number of replicas it has.

B. Distribution of losses

Our goal is to calculate cardinals of Ω_i to determine the probability of a given number of losses. In terms of fault tolerance, the slender configuration is the most stressing one: the tree has the greatest number of internal v-nodes but the smallest number of replicas of each internal v-node. Hence, internal v-nodes are more likely to be lost. On the other hand, the thick configuration is the least stressing one: the minimum number of internal v-nodes which keep the greatest number of replicas. Thus, by studying these two configurations, we aim at determining both the lower and the upper bounds on $card(\Omega_i)$.

In a configuration with l losses, two cases must be considered:

- $Case1$: either v-root is lost and exactly $l-1$ internal v-nodes are lost;
- $Case2$: or v-root is safe and exactly l internal v-nodes are lost.

Let $m : M$ be the degree of the DR-Tree. We give an upper bound to the number of losses by considering that an internal v-node is lost if all p-nodes holding its replicas are crashed.

1) Worst case: Slender configuration (V_s)

The v-root is lost if the two first bits of V are false, i.e., both the v-root holder and the p-node holding v-root replica are crashed. An internal v-node loss corresponds to a subvector where all bits are false. Let $f(n,d,l)$ be the number of configurations of n internal v-nodes with a replication degree d and exactly l losses:

$$f(n,d,l) = \binom{n}{l}(2^d-1)^{n-l}$$

Let N'_s be the number of internal v-nodes of the slender configuration for N peers:

$$N'_s = 2\sum_{i=0}^{a} m^i$$

Since in a slender configuration, the replication degree is $m-1$, we have:

$$\begin{aligned} card(\Omega_l) &\geq \underbrace{card(\Omega'_l)}_{Case1} + \underbrace{card(\Omega''_l)}_{Case2} \\ card(\Omega'_l) &= f(N'_s, m-1, l-1) \\ card(\Omega''_l) &= 3f(N'_s, m-1, l) \end{aligned}$$

2) Best case: Thick configuration (V_t)

The v-root is lost if the M first bits of V are false, i.e., both the v-root holder and p-nodes holding v-root replicas are crashed. Similarly to the previous case, an

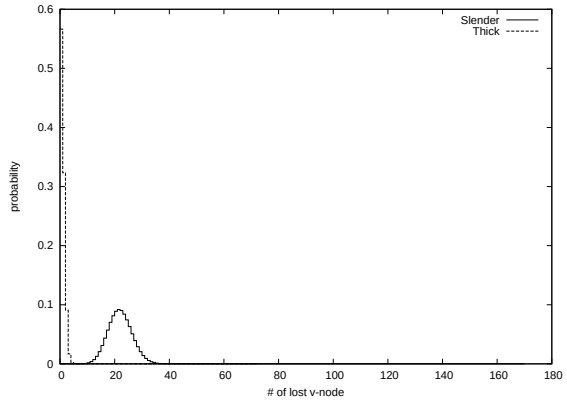


Fig. 6: Loss quantity versus probability for a given 512 p-nodes system with degree $4:8$

internal v-node loss corresponds to a subvector whose bits are all false. Let N'_t be the number of internal v-nodes of the thick configuration for N peers. Thus:

$$N'_t = \sum_{i=1}^{b} M^i$$

Since in a thick configuration, the replication degree is $M-1$, we have:

$$\begin{aligned} card(\Omega_l) &\leq \underbrace{card(\Omega'_l)}_{Case1} + \underbrace{card(\Omega''_l)}_{Case2} \\ card(\Omega'_l) &= f(N'_t, M-1, l-1) \\ card(\Omega''_l) &= \sum_{i=0}^{M-1} \binom{M}{i} f(N'_t, M-1, l) \end{aligned}$$

Figure 6 shows the probability of a given number of losses for a 512 p-nodes system with degree $4:8$. X-axis is the number of lost v-nodes. Y-axis is the probability of a given number of losses. Loss distributions are gaussian. With the increase of the degree of replication the Gaussian becomes sharper and the loss number becomes lower. This figure gives statistics assuming that every crash distribution is equiprobable. In a slender configuration, one has 10% of chances to loose 20 v-nodes. In a thick configuration, one has more than 55% of chances to loose 1 or fewer v-nodes.

C. From crashes to losses

A given number of crashes may cause different numbers of losses. In fact, the latter depends on the distribution of crashes. Basically, in a slender configuration, if both the holder of the v-root and the holder of its replica crash then the v-root is lost. But for any degree with $m > 2$ it is clear that two crashes cannot cause the loss of an internal v-node.

We will determine now the distribution of the number of losses for a given number of crashes. Let $\Omega_l|_k$ be the set of configurations with l losses and k crashes. Our aim is to propose a formula that gives the cardinal of such sets. This will be further used to investigate the DR-tree structure "reaction" in relation to a given number of crashes.

Let $g(n,d,k)$ be the number of configurations of n internal v-nodes with a replication degree d with k crashes and no loss.

$$g(n,d,k) = \begin{cases} n = 1 \wedge k \geq d \ : 0 \\ n = 1 \wedge k < d \ : \binom{d}{k} \\ n > 1 : \sum_{i=0}^{k} g(\frac{n}{2}, d, i) g(n - \frac{n}{2}, d, k-i) \end{cases}$$

The latter can be used in the definition of $h(n,d,k,l)$, i.e., the number of configurations with n internal v-nodes, a replication degree d, k crashes and l losses.

$$h(n,d,k,l) = \begin{cases} l < 0 : 0 \\ l \geq 0 : \binom{n}{l} g(n-l, d, k-ld) \end{cases}$$

The first case is distinguished by anticipating the use of this function to calculate $card(\Omega_l|_k)$.

1) Worst case: Slender configuration

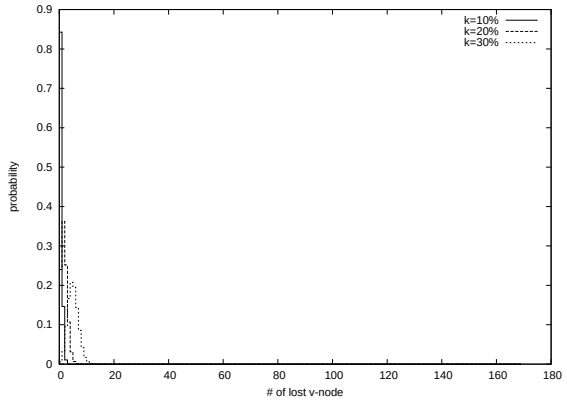
$$\begin{aligned} card(\Omega_l|_k) &\geq \underbrace{card(\Omega'_l|_k)}_{Case1} + \underbrace{card(\Omega''_l|_k)}_{Case2} \\ card(\Omega'_l|_k) &= h(n, m-1, k-2, l-1) \\ card(\Omega''_l|_k) &= 2h(n, m-1, k-1, l) + h(n, m-1, k, l) \end{aligned}$$

In $Case1$ the v-root is lost, i.e., both its holder and replica's holder are crashed. In $Case2$ the first term corresponds to situations where the v-root holder nor its replica's holder are crashed while the second one corresponds to the situation where neither its holder nor replica's holder are crashed.

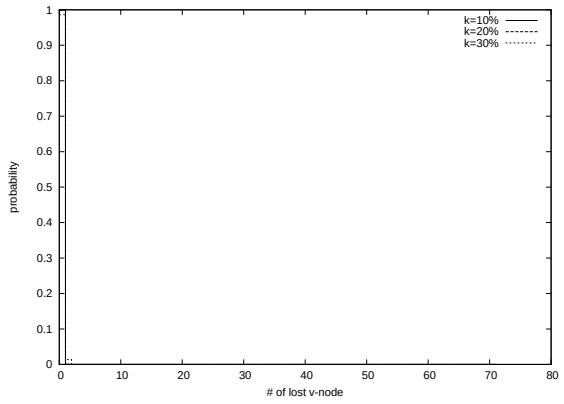
2) Best case: Thick configuration

$$
\begin{aligned}
card(\Omega_l|_k) &\leq \underbrace{card(\Omega'_l|_k)}_{Case1} + \underbrace{card(\Omega''_l|_k)}_{Case2} \\
card(\Omega'_l|_k) &= h(n, M-1, k-M, l-1) \\
card(\Omega''_l|_k) &= \sum_{i=0}^{M-1} \binom{M}{i} h(n, M-1, k-i, l)
\end{aligned}
$$

In $Case1$, the v-root is lost if both its holder and replicas holders are crashed. In $Case2$, the v-root is not lost and we can distribute up to $M-1$ crashes in the subvector of V related to the v-root. The subvectors concerning internal v-nodes contains thus the remaining crashes and all losses.



(a) Slender configuration



(b) Thick configuration

Fig. 7: Number of losses versus probability for different crash rates

Figure 7 shows the probability of a given number of losses for a system with 512 p-nodes, a degree of $4:8$, and different crash rates. The X-axis and Y-axis represent the number of lost v-nodes and the probability of a given number of losses respectively.

The slender configuration shows that the loss distribution is gaussian. With 30% of simultaneous crashes, there is 20% of chance of loosing 6 v-nodes. With 10% of simultaneous crashes, there is 85% of chance of loosing 1 or fewer v-node.

The thick configuration shows that increasing the replication degree drastically reduces the quantity of losses. The replication degree is almost $log(N)$; even with 30% simultaneous crashes, there exist more than 95% of chances of loosing 1 or fewer v-nodes.

D. Probability of no loss

A particular case of the previous calculations concerns the number of configurations with no loss.

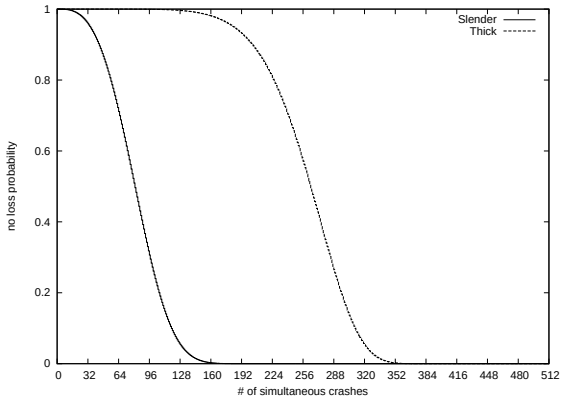


Fig. 8: No loss probability for a 512 p-nodes system with degree $4:8$

Figure 8 shows the probability of no loss versus the number of crashes for both the slender and thick configurations of a 512 p-nodes system with a degree $4:8$. X-axis and Y-axis respectively indicates the number of simultaneous crashes and the probability of no loss for a given number of simultaneous crashes. It illustrates how this probability decreases when the number of crashes increases. The curves are shifted basically because the replication degree of the thick configuration is twice greater than in the slender configuration.

On one hand, it is theoretically interesting to have an idea of DR-tree structure's behavior facing an arbitrary number of crashes. On the other hand, in practice, we are often interested in configurations

where the probability of no loss is ”close” to one. Moreover, we usually want to specify bounds on the number of simultaneous crashes. Basically, it is another way of parametrising faults. Figure 9 illustrates such a case. The X-axis shows the number of simultaneous crashes while the Y-axis is the probability of no loss for a given number of simultaneous crashes. We have considered that up to 10% of p-nodes can crash simultaneously. With the slender configuration, there are more than 80% of chances to have no loss. With the thick configuration, this same probability is 99.999%. Notice that since this probability is close to 100% the curve does not appear clearly on the figure.

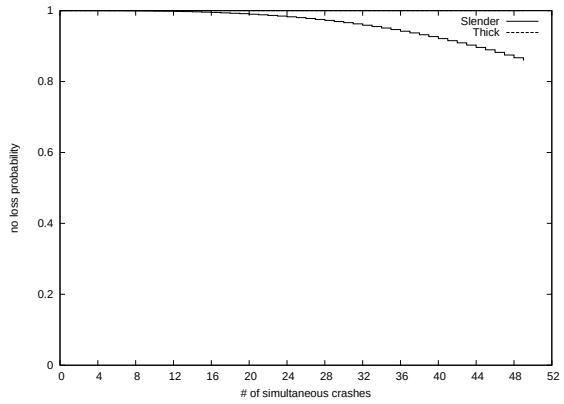


Fig. 9: No loss probability versus bounded simultaneous number of crashes

VII. Conclusion and Future Work

This article has presented another advantage of the distinction between the DR-tree logical structure and the interaction graph [2], [8]. In other words, introducing replication at interaction graph does not impact the DR-tree structure and thus enforces separation of concerns: routing is addressed at DR-tree level while fault tolerance is addressed at interaction graph level. Evaluation results show that, in presence of crashes, our approach preserves DR-tree connectivity and structure much more efficiently than the fault tolerance mechanism proposed in [3] and that the former outperforms the latter in terms of message traffic ($log_m(N)$ instead of $N\ log_m(N)$) and stabilizing time (proportional to M instead of $log_m(N)$).

It is worth pointing out that we have studied the most stressing configuration for replication. In fact, we have determined lower bounds. However, they are underestimated since we have considered the maximum number of losses (an internal v-node is lost if all p-nodes holding its replicas are crashed). It would be interesting to avoid such an approximation theoretically or at least evaluate it.

Our work is based on calculations over V, the binary vector of bits representing the global system state. The replication scheme exploits logical proximity. Furthermore, as mentioned in [8], the concept of distribution is easily extendable to different classes of graphs. Therefore, we intend to apply our approach to other distributed tree structures, graphs, or even rings, whose connectivity is also crash-sensitive.

References

[1] K. Aberer, P. Cudre-Mauroux, A. Datta, Z. Despotovic, M. Hauswith, M. Punceva, and R. Schmidt. P-Grid: a self-organizing access structure for p2p information. In *CoopIS*, 2001.

[2] L. Arantes, M. G. Potop-Butucaru, P. Sens, and M. Valero. Enhanced dr-tree for low latency filtering in publish/subscribe systems. In *AINA*, 2010.

[3] S. Bianchi, A. K. Datta, P. Felber, and M. Gradinariu. Stabilizing peer-to-peer spatial filters. In *ICDCS*, 2007.

[4] E. Caron, F. Desprez, C. Fourdrignier, F. Petit, and C. Tedeschi. A repair mechanism for fault-tolerance for tree-structured peer-to-peer systems. In *HiPC*, 2006.

[5] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD*, 1984.

[6] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *VLDB*, 2005.

[7] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *ICDE*, 2006.

[8] M. Valero, L. Arantes, M. Gradinariu, and P. Sens. Dynamically reconfigurable filtering architectures. In *SSS*, 2010.