Cliff-Edge Consensus: Agreeing on the Precipice

François Taïani, Barry Porter, Geoff Coulson, Michel Raynal

Cliff-Edge Consensus: Agreeing on the Precipice

François Taïani[1], Barry Porter[3], Geoff Coulson[4], Michel Raynal[1,2]

[1] Université de Rennes 1, IRISA, France
{francois.taiani,michel.raynal}@irisa.fr,
[2] Institut universitaire de France, France
[3] School of Computer Science, University of St Andrews, UK
bfp@st-andrews.ac.uk,
[4] School of Computing and Communications, Lancaster University, UK
g.coulson@lancaster.ac.uk

Abstract. This paper presents a new form of consensus that allows nodes to agree locally on the extent of crashed regions in networks of arbitrary size. One key property of our algorithm is that it shows local complexity, i.e. its cost is independent of the size of the complete system, and only depends on the shape and extent of the crashed region to be agreed upon. In this paper, we motivate the need for such an algorithm, formally define this new consensus problem, propose a fault-tolerant solution, and prove its correctness.

Keywords: distributed computing, fault-tolerance, failure detection, consensus, scalability

1 Introduction

Modern distributed computer systems are increasingly large and complex, often involving tens of thousands of machines distributed across continents to deliver key global services such as search, content delivery, or messaging to millions of users. Constructing such systems requires distributed services that are *scalable* to account for the global size of these systems, *efficient* to meet the increasing expectations of users, and *robust* to overcome the unavoidable failures of individual elements in such large-scale systems.

One strategy to provide these properties is to eschew any form of global knowledge or global coordination, and instead rely on decentralized topologies in which each node only perceives one limited part of the system. Coordinating the work of individual nodes in such decentralized topologies is however difficult, leading to a number of works that aim to provide fundamental coordination services such as consensus in systems whose size might be unknown, and in which participants only have a partial knowledge of each other [12, 11, 7, 2, 4, 17].

In this article we look at one such fundamental service for the consistent detection of crashed regions in networks of arbitrary size. Our premise is that large-scale distributed systems can be benefit from a collective response to the

emergence of crashed regions, so that there is a need for the nodes around a crashed region to come to an agreement on the shape and extent of this region, and possibly decide on some unified recovery action to be undertaken. This problem of collective agreement can be cast as a new type of specialized consensus, where nodes that border a crashed region (i.e. nodes on the "cliff-edge") want to agree on the extent of this crashed region (the "precipice" of our title).

This form of agreement in large-scale systems presents two interesting and related challenges, which clearly set it apart from existing works in the area: (i) The solution should be scalable, and work in networks of arbitrary size, i.e. it should only involve nodes in the vicinity of a crashed region, and never the complete system. (ii) Because of ongoing failures, nodes might disagree on the extent of a crashed region, but as they do so they'll also disagree on *who* should even take part in the agreement, since both *what* is to be agreed (the crashed region), and *who* should agree on it (the nodes bordering the region) are irredeemably interdependent. We've termed this second facet of this emergent agreement the *self-constituency problem*.

Contributions: In this article, we formally define this new consensus problem, present a fault-tolerant solution that uses perfect failure detectors, and prove its correctness. Our solution works in systems of arbitrary size, in a scalable manner (the algorithm only involves nodes bordering a crashed region), for any number of faults.

Paper organization: We first present the cliff-edge consensus problem in Section 2, then move on to describe our solution (Section 3). We present our proof of correctness (Section 3.3), and finish with some related work (section 4) and a conclusion.

2 The problem

2.1 Overview

We consider systems in which individual nodes only have a partial knowledge of the rest of the system. This partial knowledge (nodes x knows node y) defines a form of *spatial proximity* between nodes, captured in our model by an undirected graph. (We revisit these points more formally in Section 2.2 below.) In case of correlated failures (for instance because the network's topology mirrors physical proximity as in some distributed hash table protocols, or because neighbouring nodes rely on the same relay to communicate), whole regions of the network might disappear, requiring surviving nodes to (i) *identify and agree* on the extent of the crashed region, and (ii) *decide* on a common action to mitigate the failure.

For instance, in the network of Fig. 1-a, the nodes in region F_1 and F_2 have crashed. These crashes are being detected by the *border nodes* (i.e. the neighbouring nodes) of each crashed region: *paris*, *london*, *madrid* and *roma* for F_1 and *tokyo*, *vancouver*, *portland*, *sydney*, and *beijing* for F_2. (This detection occurs with the help of an appropriate failure detector, which we discuss in more detail in Section 2.2.)

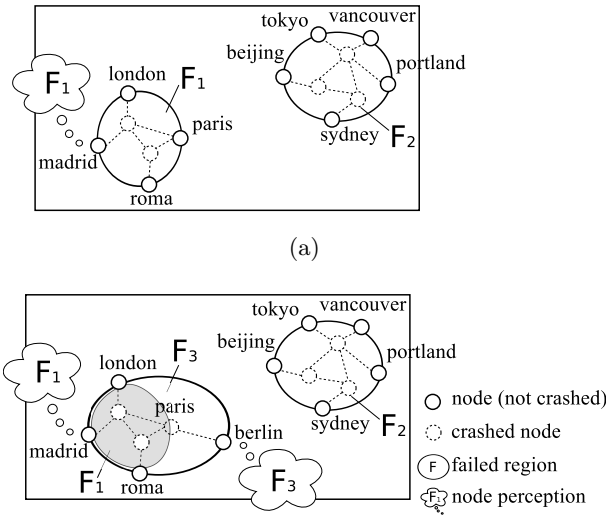

Fig. 1: Protocol instances and conflicting views

Our scalability requirements impose that communications related to F_1 (resp. F_2) should be limited to nodes bordering F_1 (resp. F_2). For instance *vancouver* should not have to communicate with *madrid* to decide on a repair strategy for F_2. This excludes traditional consensus approaches that would involve the entire network in a protocol run.

Because of ongoing crashes, nodes bordering the same crashed region might however possess divergent views regarding the extent of their region, and hence have diverging perceptions of whom should get involved in a protocol run. In Fig. 1-b, for instance, *paris* fails after *madrid* has detected F_1 as crashed, but before an agreement on F_1 has been reached. The crashed region F_1 thus grows into F_3, and a new node *berlin* (*paris*'s still non-crashed neighbour) becomes involved. *berlin* detects the entirety of F_3 as crashed.

madrid and *berlin* now have different, albeit overlapping views. If *madrid* is slow to detect *paris*' crash, it might try to agree on F_1 with *london* and *roma* alone, while *berlin* will try to involve all nodes bordering F_3 to decide on F_3. Each node's effort could stall each other, or could lead to duplicated or inconsistent decisions. Our protocol prevents this and insures that any decisions pertaining to the same part of the network *converge* to a unified view, a problem that we have termed the *convergent detection of crashed regions*.

2.2 System model and assumptions

We model our system as a finite undirected graph $\mathcal{G} = (\Pi, \mathrm{E})$ of asynchronous message-passing nodes $\Pi = \{p_1, .., p_n\}$, where $\mathcal{G}$ represents the knowledge that nodes have of each other in the system.

A node is *faulty* if it crashes at some point, *correct* if it does not crash during the execution of the algorithm. Any two nodes might exchange messages through asynchronous, reliable, and ordered (fifo) channels. We also assume that each node can query $\mathcal{G}$ on demand, either by directly contacting live nodes, or using some underlying topology service for crashed nodes.

The *border* of a node p is the set of p's neighbours. By extension, the border of a set $S \subseteq \Pi$ of nodes are the nodes that have a neighbour in S but do not belong to S: $\mathsf{border}(S) = \{q \in \Pi \backslash S \mid \exists p \in S : (p, q) \in \mathrm{E}\}$. A *region* is a connected subgraph of $\mathcal{G}$. A *crashed region* at a time t is a region in which all nodes have crashed.

To specify the liveliness of our protocol, we need to define the three additional notions of *adjacency*, *faulty domain* and *faulty cluster*, which capture the maximum extent of crashed regions during a run. More precisely, a *faulty domain* is a region in which all nodes are faulty, but whose border nodes are correct. By construction, two faulty domains can only be either equal or disjoint.

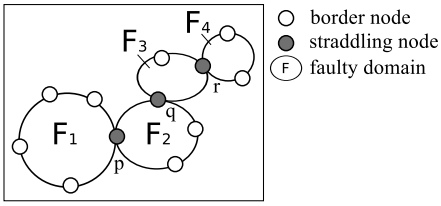

Fig. 2: A cluster of adjacent faulty domains

Two faulty domains F and H are *adjacent* (noted $F \parallel H$) if their borders intersect (e.g. $F_1 \parallel F_2$ in Fig. 2). We say that two faulty domains F_0 and F_n are in the same *faulty cluster*, noted $\mathsf{clustered}(F_0, F_n)$, if they are transitively adjacent[5], i.e. if there is a sequence of faulty domains F_i so that $F_1 \parallel F_2 \ldots F_{n-1} \parallel F_n$. For instance, we have $\mathsf{clustered}(F_1, F_4)$ in Fig. 2.

2.3 Convergent detection of crashed regions: specification

Operations We use a mono-threaded event-based programming model to specify the convergent detection of crashed regions, and present our solution. Our service starts when a node detects one of its neighbours q as crashed ($\langle \mathbf{crash} \,|\, q \rangle$ event). It stops by raising a $\langle \mathbf{decide} \,|\, S, d \rangle$ event, where S is the crashed region decided by the local node, and d is the decision taken by this node with respect to S (e.g. a repair plan, or some other form of coordinated action). We call S the *view* of the deciding node.

[5] More formally, $\mathsf{clustered}(.,.)$ is the transitive closure of the adjacency relation, and *faulty clusters* are the equivalence classes of this closure.

Properties The convergent detection of crashed regions is characterised by the following properties:

CD1 (Integrity) No node decides twice on the same region.
CD2 (View Accuracy) If a node p decides (V, d), then $p \in \mathsf{border}(V)$, and V is a crashed region.
CD3 (Locality) Communication is limited to faulty-domains and their borders, i.e. a node p only exchanges messages with a node q if there is a faulty domain S such that $\{p, q\} \subseteq S \cup \mathsf{border}(S)$.
CD4 (Border Termination) If p decides (V, d), then all correct nodes in $\mathsf{border}(V)$ eventually decide.
CD5 (Uniform Border Agreement) If two nodes p and q decide, and p decides (V, d), and $q \in \mathsf{border}(V)$, then q decides (V, d).
CD6 (View Convergence) If two correct nodes decide V and W, $(V \cap W \neq \emptyset) \Rightarrow (V = W)$.
CD7 (Progress) In each faulty cluster, at least one correct node bordering a faulty domain in the cluster eventually decides: if $\mathcal{D}$ is the set of all faulty domains, $\forall V \in \mathcal{D} : \exists W \in \mathsf{clustered}(V, \cdot) : \exists p \in \mathsf{border}(W) : p$ decides.

These properties capture the requirement that the nodes bordering a crashed region should agree on the extent of this crashed region, and decide on a common course of action. CD1 (*Integrity*), CD5 (*Uniform Border Agreement*), and CD4 (*Border Termination*) are directly adapted from (uniform) consensus; CD2 (*View Accuracy*) is taken over from the strong accuracy of fault detectors; and CD7 (*Progress*) is a weak form of termination.

The problem's originality resides in the two remaining properties: CD6 (*View Convergence*) and CD3 (*Locality*). CD6 (*View Convergence*) forbids conflicting agreements on overlapping crashed regions (F_1 and F_3 in Fig. 1). CD3 (*Locality*) provides scalability by limiting the system's reaction to the vicinity of crashed regions. In particular, CD3 (*Locality*) implies that nodes with no faulty neighbours do not take part in the protocol. As a result, the protocol only depends on the amount of failures in the system, but not on the system's actual size. *Locality* also excludes the use of a system-wide consensus to fulfil the other properties.

This *scoping strategy* creates however a pernicious inter-dependency between the protocol's participants (the 'constituency') and what they are agreeing to: To start our protocol, a node needs to know with whom it should be agreeing (its fellow border nodes), but this set of nodes depends on the final outcome of the protocol (the crashed region agreed upon).

In the following, we first illustrate this problem, which we have termed *self-defining constituencies*. We then move on to define formally the properties of this problem; we present our solution to this problem; and propose a proof of its correctness.

3 A Cliff-Edge Consensus Protocol

3.1 Preliminaries: Failure detector, multicast, region ranking

Our algorithm uses a perfect failure detector provided in the form of a *subscription-based* service: a node p subscribes to the crashes of a subset of nodes S by issuing the event $\langle \textbf{monitorCrash} \,|\, S \rangle$ to its local failure detector. Our failure detector is perfect and ensures: *(i) Strong Accuracy*: if a node p receives a $\langle \textbf{crash} \,|\, q \rangle$ event, then q has crashed, and p did subscribe to be notified of q's crash; and *(ii) Strong Completeness*: if a node q has crashed, and p has subscribed to be notified of q's crash, then p will eventually receive a $\langle \textbf{crash} \,|\, q \rangle$ event.

For the sake of conciseness, we use a basic multicast service, represented by the events $\langle \textbf{multicast} \,|\, R, [\text{m}] \rangle$ and $\langle \textbf{mDeliver} \,|\, p, [\text{m}] \rangle$. This service simply sends to each recipient a multicast message [m] over underlying point-to-point channels, in a plain loop. This service provides no guarantees beyond those of the underlying channels, and is essentially a shorthand to keep our code brief.

We also use a *ranking relation* between regions, noted $\succ$: $R \succ S$ iff either (i) R contains more nodes than S, or (ii) they contain the same number of nodes but R's border contains more nodes than S's border, or (iii) R and S have the same size, and so do their respective borders, but R is greater than S according to some strict total order relation $\rhd$ on sets of nodes. The actual ordering relation $\rhd$ on node sets does not matter. One possibility is to use a lexicographic order on node IDs. By construction, $\succ$ is a strict total order on regions. For a set $\mathcal{C}$ of regions, $\mathsf{maxRankedRegion}(\mathcal{C})$ is the highest ranked region in $\mathcal{C}$.

Finally, for a subset S of nodes, $\mathsf{connectedComponents}(S)$ returns the set of the maximal regions of S, i.e., formally, the vertex sets of the connected components of the subgraph $\mathcal{G}[S]$ induced by S in $\mathcal{G}$.

3.2 Algorithm

The pseudo code of our algorithm is given in Figure 1. $\langle \textbf{init} \rangle$ is executed by all nodes when the protocol starts. Each node then remains idle until one of its neighbours fails, as notified by a $\langle \textbf{crash} \,|\, q \rangle$ event.

The bulk of the protocol is primarily a superposition of flooding uniform consensus instances [8, 13] between the border nodes of proposed views. This superposition is complemented by an arbitrating mechanism to deal with overlapping but conflicting views (line 26). Because of this arbitration, all consensus instances must be tracked concurrently by our protocol, in the variables opinions[·][·][·] and waiting[·][·], which are indexed by proposed views (in addition to rounds, and, for opinions, participants).

A node starts a consensus instance when it detects that one of its neighbours has crashed (line 17). The view it proposes has been incrementally built when receiving $\langle \textbf{crash} \,|\, . \rangle$ events (line 5), and is the highest ranked crashed region known to the node at this point. The view construction continues in the background as the consensus unfolds (lines 5-10), to be used if the attempt to reach an agreement fails.

The opinion vectors received from other nodes in a round are gathered at line 18. Because a node might be involved simultaneously in multiple conflicting consensus instances, messages related to conflicting views are also gathered and processed. The resulting opinion vectors, indexed by round and proposed view (line 24) are stored in opinions[·][·][·].

If a node becomes aware of a conflicting view with a lower rank (line 26), it sends a special $\text{op}_{\mathsf{reject}}$ vector to this view's border nodes, and subsequently ignores any message related to this view (lines 28-31).

Rounds are completed at line 32 when all non-crashed border nodes of view have replied: if no more rounds are needed (line 34), and the node's final vector only contains accept values, a decision value is deterministically selected for the proposed view (line 35), and the node decides[6]. Otherwise the whole process is reset, and restarts at line 12 as soon as a new crashed node is detected.

3.3 Proof of correctness

In the following, we use a subscript notation to distinguish between the same protocol variable at different nodes: e.g. opinions_p for variable opinions of p.

Theorem 1. *our protocol fulfils properties CD1 (Integrity), CD2 (View Accuracy), and CD3 (Locality).*

Proof. CD1 is fulfilled by construction. For CD2, connectedComponents() at line 8 and the strong accuracy of the failure detector insure that proposed views are crashed regions. Using recursion on $\langle \mathbf{crash} \,|\, . \rangle$ events, a node p can be shown to respect the two invariants *(i)* $p \in \mathsf{border}(\text{locallyCrashed}_p)$ and *(ii)* $\{p\} \cup \text{locallyCrashed}_p$ is connected, thus yielding that p is on the border of any view it proposes. CD3 follows from CD2, and the fact that two nodes only exchange messages when both border a region detected as failed by one of them.

Our proof of the remaining four properties reuses elements of the proof of the consensus algorithm presented in [8] for strong failure detectors (S), of which the flooding uniform consensus is derived. The difficulty lies in that our protocol uses multiple overlapping consensus instances, each indexed by the view it proposes, with no prior agreement on either the set the consensus instances, their participants, or their sequence. In addition, our arbitrating mechanism means a node can first propose and then reject the same view, thus complicating the *uniform border agreement*, as we shall see.

Lemma 1. *At any execution point the vectors* $\text{opinions}_p[V][r][\cdot]$ *of* p *are such that* $\forall q \in \mathsf{border}(V)$:
1) $\text{opinions}_p[V][r][q] = \mathsf{reject} \Rightarrow q$ *rejected* V *earlier* $\wedge$
2) $\text{opinions}_p[V][r][q] = (\mathsf{accept}, \cdot) \Rightarrow q$ *accepted* V *earlier*

[6] For clarity's sake, the presented version is not optimized. A classical optimization consists in terminating a consensus instance once a node sees that all nodes in its border set know everything (i.e. no $\perp$), i.e. after two rounds, in the best case.

Algorithm 1 Convergent detection of crashed regions executed by node p

1: **upon event** $\langle$ **init** $\rangle$
2: decided $\leftarrow \perp$; proposed $\leftarrow \perp$
3: locallyCrashed, maxView, candidateView, V_p, received, rejected $\leftarrow \emptyset$
4: **trigger** $\langle$ **monitorCrash** $|$ $\mathsf{border}(p)\rangle$

5: **upon event** $\langle$ **crash** $|$ $q\rangle$ ▷ View construction
6: locallyCrashed $\leftarrow$ locallyCrashed $\cup \{q\}$
7: **trigger** $\langle$ **monitorCrash** $|$ $\mathsf{border}(q)\backslash$locallyCrashed$\rangle$
8: $\mathcal{C} \leftarrow \mathsf{connectedComponents}$(locallyCrashed)
9: **if** maxView $\prec \mathsf{maxRankedRegion}(\mathcal{C})$ **then**
10: maxView $\leftarrow \mathsf{maxRankedRegion}(\mathcal{C})$
11: candidateView $\leftarrow$ maxView

12: **upon event** proposed $= \perp \wedge$ candidateView $\neq \emptyset$ ▷ New consensus instance
13: $V_p \leftarrow$ candidateView ; candidateView $\leftarrow \emptyset$
14: proposed $\leftarrow \mathsf{selectValueForView}(V_p)$
15: $\mathrm{op}_{\mathsf{accept}}[p_k] \leftarrow \perp$ **for all** $p_k \in \mathsf{border}(V_p)\backslash\{p\}$
16: $\mathrm{op}_{\mathsf{accept}}[p] \leftarrow (\mathsf{accept}, \text{proposed})$; $r \leftarrow 1$
17: **trigger** $\langle$ **multicast** $|$ $\mathsf{border}(V_p)$, $[1, V_p, \mathsf{border}(V_p), \mathrm{op}_{\mathsf{accept}}]\rangle$ ▷ proposing V_p

18: **upon event** $\langle$ **mDeliver** $|$ p_i,$[r, V, B, \mathrm{op}]\rangle \wedge V \notin$ rejected ▷ Updating opinions
19: **if** $V \notin$ received **then**
20: received $\leftarrow$ received $\cup \{V\}$ ▷ Initialise data structures for V
21: opinions$[V][r][p_k] \leftarrow \perp$ **for all** $p_k \in B \wedge 1 \leq r < |B|$
22: waiting$[V][r] \leftarrow B$ **for all** $1 \leq r < |B|$
23: **for all** p_k **such that** (opinions$[V][r][p_k] = \perp \wedge \mathrm{op}[p_k] \neq \perp$) **do**
24: opinions$[V][r][p_k] \leftarrow \mathrm{op}[p_k]$
25: waiting$[V][r] \leftarrow$ waiting$[V][r]\backslash(\{p_i\} \cup \{p_k | \mathrm{op}[p_k] = \mathsf{reject}\})$

26: **upon event** $\exists L \in$ received $: L \prec V_p$ ▷ Rejecting a lower ranked view
27: **trigger** $\langle$ **reject** $|$ $L\rangle$

28: **upon event** $\langle$ **reject** $|$ $L\rangle$
29: $\mathrm{op}_{\mathsf{reject}}[p_k] \leftarrow \perp$ **for all** $p_k \in \mathsf{border}(L)\backslash\{p\}$
30: $\mathrm{op}_{\mathsf{reject}}[p] \leftarrow \mathsf{reject}$; received $\leftarrow$ received$\backslash\{L\}$; rejected $\leftarrow$ rejected $\cup \{L\}$
31: **trigger** $\langle$ **multicast** $|$ $\mathsf{border}(L)$, $[1, L, \mathsf{border}(L), \mathrm{op}_{\mathsf{reject}}]\rangle$

32: **upon event** $V_p \in$ received $\wedge$ waiting$[V_p][r]\backslash$locallyCrashed $= \emptyset \wedge$ decided $= \perp$
33: **if** $r = |\mathsf{border}(V_p)| - 1$ **then** ▷ Consensus instance completed
34: **if** $\forall p_i \in \mathsf{border}(V_p)$: opinions$[V_p][r][p_i] = (\mathsf{accept}, v_{p_i})$ **then**
35: decided $\leftarrow \mathsf{deterministicPick}(\{v_{p_i}\}_{p_i \in \mathsf{border}(V_p)})$ ▷ Decision
36: **trigger** $\langle$ **decide** $|$ V_p, decided$\rangle$
37: **else** proposed $\leftarrow \perp$ ▷ Consensus attempt failed, reset
38: **else** ▷ New round
39: $r \leftarrow r + 1$
40: **trigger** $\langle$ **multicast** $|$ $\mathsf{border}(V_p), [r, V_p, \mathsf{border}(V_p), \text{opinions}[V_p][r-1]]\rangle$

Proof. First let us note that the only location where opinions[V][q][q] is explicitly assigned an accept (resp. reject) value is when q accepts (resp. rejects) view V at line 16 (resp. line 30). This accept (resp. reject) value then propagates to the opinion vectors of other border nodes through the network (lines 17, 31 and 40), and the assignment of line 24. A recursive data-flow argument on the values of opinions[V][r][·] taken at these lines yields the lemma.

Lemma 2. *A node proposes (resp. rejects) a given view V at most once. A node never proposes a view it has previously rejected.*

Proof. The uniqueness of rejection follows from the use of the rejected and received variables. The use of the strict ranking relation $\prec$ (line 9) means the series of values taken by view is strictly monotonic according to $\prec$, and by construction that this is also true of view, thus completing the lemma.

Lemma 3. *If two nodes p and q complete a consensus instance on the same* $\text{view}_{p|q} = V$ *(line 34), they obtain the same opinion vector:*

$$\text{opinions}_p[V][N][\cdot] = \text{opinions}_q[V][N][\cdot]$$
$$\textit{where } N = |\mathsf{border}(V)|$$

Proof. We prove this lemma by contradiction. Let's assume $\exists k \in \mathsf{border}(V)$: $\text{opinions}_p[V][N][k] \neq \text{opinions}_q[V][N][k]$. If one of the two values is $\perp$, we can use the well-known argument on cascading crashes, identifying $N-1$ distinct nodes in $\mathsf{border}(V)$ that did not complete the consensus instance, contradicting the fact that p and q completed it.

Let's now assume both values are non-$\perp$. The first sub-case is when both values are accept for k, with different decision values on p and q, i.e. $\text{opinions}_p[V][N][k] = (\mathsf{accept}, v_k^p)$ and $\text{opinions}_q[V][N][k] = (\mathsf{accept}, v_k^q)$ with $v_k^p \neq v_k^q$. Using lemma 2, we conclude that line 16 is executed only once by k for V, and that $v_k^p = v_k^q$, yielding the contradiction.

Finally, let's assume one value is accept, while another is reject, e.g. without loss of generality, $\text{opinions}_p[V][N][k] = (\mathsf{accept}, \cdot)$ and $\text{opinions}_q[V][N][k] = \mathsf{reject}$. From lemma 1 we conclude that k has both proposed and rejected V. Let's call e^k_{accept} and e^k_{reject} the corresponding execution points. Because of lemma 2, e^k_{accept} and e^k_{reject} are unique, and e^k_{accept} happened before e^k_{reject}. Because the best-effort multicast is fifo, this means q received the message for e^k_{accept} before that of e^k_{reject}, and because line 24 only updates $\perp$ values, that $\text{opinions}_q[V][N][k] = (\mathsf{accept}, \cdot)$, yielding the contradiction.

Theorem 2. *our protocol fulfils properties CD5 (Uniform border agreement) and CD4 (Border termination).*

Proof. Let's assume p and q decide, p decides $(V, \text{decided}_p)$, and $q \in \mathsf{border}(V)$. If p decides on V, then p completed the corresponding consensus instance with only accept values, and since $q \in \mathsf{border}(V)$ we have $\text{opinions}_p[V][N][q] = (\mathsf{accept}, \cdot)$. By lemma 1, q proposed V. Since by construction a node *(i)* cannot propose

any new view once it has decided on one, and *(ii)* cannot start a new consensus instance before completing the current one, q proposed V and completed the corresponding consensus instance before deciding. By lemma 3, q obtained the same vector opinions$_q$ as p on V, and hence decided $(V, \text{decided}_p)$ by determinism of deterministicPick (line 35), thus proving CD5.

CD4 follows the same line, with the observation that if a node p completes a consensus instance on a view V, then all other nodes in $\mathsf{border}(V)$ either took part in each round or crashed, implying that all correct nodes eventually complete the instance with the same opinion vector as p (by way of lemma 3).

Theorem 3. *our protocol fulfils CD6 (View convergence).*

Proof. Let's consider two correct nodes p and q that decide on overlapping crashed regions V_p and V_q: $V_p \cap V_q \neq \emptyset$. If one node is in the border of the other's region, e.g. $p \in \mathsf{border}(V_q)$, then *Uniform Border Agreement* (CD5) and *Integrity* (CD1) give us $V_p = V_q$.

Let's now assume $p \notin \mathsf{border}(V_q) \wedge q \notin \mathsf{border}(V_p)$, and use a proof by contradiction. Since $V_p \cap V_q \neq \emptyset$, there is a node $a \in V_p \cap V_q$ (Fig. 3). V_p being a region bordered by p (CD2), there exists a path $(n_0 = p, n_1, ..., n_k = a)$ that links a to p through V_p: $\{n_1, ..., n_k, a\} \subseteq V_p$. Since $a \in V_q$, we can consider the point when this path "penetrates" for the first time into V_q, i.e. we can consider $n_{i_0} \in V_q$ and $\forall i < i_0 : n_i \notin V_q$. Since p is correct, $n_{i_0} \neq p$, i.e. $i_0 \geq 1$, and we can look at n_{i_0-1}, the node in the path just before n_{i_0}. Let's call this node r (Fig. 3). Because n_{i_0} is the first node in the path to belong to V_q, we have $r \in \mathsf{border}(V_q)$, and since $p \notin \mathsf{border}(V_q)$, $r = n_{i_0-1}$ cannot be p ($i_0 > 1$). Because, with the exception of p, the path connecting p to a is embedded in V_p, this means that r is in fact located in p's crashed region. This reasoning thus yields us a node (r) that is both on $\mathsf{border}(V_q)$ *and* in p's crashed region: $r \in V_p \cap \mathsf{border}(V_q)$. Using an identical argument, we can find a node s such that $s \in V_q \cap \mathsf{border}(V_p)$ (Fig. 3).

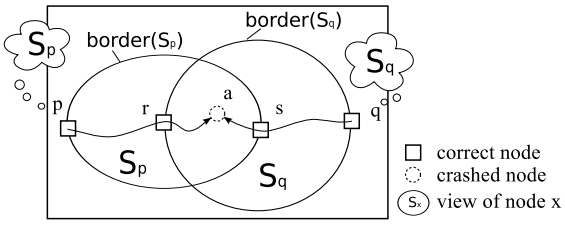

Fig. 3: Convergence between overlapping views

To complete our proof, we now look at the happen-before relationships between events related to r and s. Let's first consider s. Since $s \in \mathsf{border}(V_p)$ and p decided on V_p, s itself did propose V_p (lemma 1). Since $r \in V_p$, s did detect r as crashed as some point. By a similar reasoning, we conclude that r proposed V_q, and hence detected s as crashed as some point.

We thus end up with a set of 6 events that form a circular chain of happen-before events: *s_detects_r* $\rightarrow$ *s_proposes* $\rightarrow$ *s_crashes* $\rightarrow$ *r_detects_s* $\rightarrow$ *r_proposes* $\rightarrow$ *r_crashes* $\rightarrow$ *s_detects_r* ... This provides our contradiction.

Theorem 4. *our protocol fulfils properties CD7 (Progress).*

Proof. Again we use a contradiction: consider a cluster of adjacent faulty domains (Fig. 2), and assume none of its correct border nodes ever decide. Since this situation lasts indefinitely, we can consider the case where all crashed regions are maximal and all remaining nodes are correct.

Because the views proposed by a node are strictly monotonic according to $\prec$, and because $\mathcal{G}$ is finite, a node cannot propose an infinite sequence of views. A correct border node p that does not decide falls therefore into two cases: either (**C1**) p is blocked waiting for the reply of another node q (line 38); or (**C2**) the last view proposed by p failed (line 37), and p does not detect any new crashed node (line 5).

Case C1: If p is waiting for the reply of some other node q, q must be correct (if it were not, q would eventually crash, thus unblocking p). Since there's a path of crashed nodes from p to q (since p is waiting for q), q is on the border of the same faulty domain as p, so q never decides (by assumption).

As for p, q falls in either case **C1** or **C2**. Let's first assume that the last view V_q^{max} proposed by q failed, and q does not detect any new crashed node (**C2**). Since we've assumed that all faulty nodes have crashed, by strong completeness of the failure detector, V_q^{max} is a faulty domain, and because of the use of $\mathsf{maxRankedRegion}$ (line 10) and the fact that $\prec$ subsumes set inclusion, V_q^{max} is higher ranked than any crashed region bordered by q.

Since p is waiting for q, $V_p \neq V_q^{\mathsf{max}}$, and since q is on the border of both V_p and V_q^{max}, V_p is lower-ranked than V_q^{max}: $V_p \prec V_q^{\mathsf{max}}$. q has received a round-1 message proposing V_p (line 18), and should have rejected it (line 31), thus ending p's wait on q, which contradicts our assumption.

We therefore conclude that q cannot fall in case **C2**, and instead is blocked in a consensus round proposing a crashed region V_q (case **C1**). q received p's proposal message, and did consider it for rejection (line 26). Because p is waiting for q, we know it did not receive any rejection message from q, and therefore, $V_p \succeq V_q$. Since p is waiting for q, q is not proposing the same view as p, yielding a strict ordering between the two views $V_p \succ V_q$.

This construction can be repeated recursively, first for q, and then for the node q is waiting on, etc, each time yielding an infinite number of pairwise distinct crashed regions (via CD2) that are strictly ordered by the ranking relationship: $V_{p_1} \succ V_{p_2} \succ ... \succ V_{p_i} \succ ...$ This contradicts our assumption that each faulty cluster contains a finite number of faulty domains, each containing a finite number of nodes.

Case C2: Let's now assume the last view V_p^{max} proposed by p failed, and p does not detect any new crashed node. As with V_q^{max} above, V_p^{max} is a faulty domain, and all its border nodes are correct. Because the failure detector is strongly accurate, for p's proposal to fail, one node $q \in \mathsf{border}(V_p^{\mathsf{max}})$ must have rejected

V_p^{max} because it was proposing a higher-ranked view V_q^{higher}. By assumption, q never decides, it must either fall in case **C1** or **C2**. If q is in case **C1**, we can repeat the same argument as for p in Case **C1**, above. If **C2**, q's last view V_q^{max} is higher or equal than any view q ever proposed, implying $V_p^{\mathsf{max}} \prec V_q^{\mathsf{higher}} \preceq V_q^{\mathsf{max}}$.

By recursively applying this argument, we either come back to case **C1** at some point, or obtain an infinite sequence of strictly ordered faulty domains $V_{p_1}^{\mathsf{max}} \prec V_{p_2}^{\mathsf{max}} \prec V_{p_3}^{\mathsf{max}} \prec ...$, which yields our contradiction.

4 Related work

The algorithm we presented builds on our earlier work on the generic repair of overlay networks [16], in which we first sketched some of the ideas presented in this paper, albeit without any formal definition or proof.

Our algorithm can be viewed as a combination of an ad-hoc group formation and 'preference-based' leader election [18], with the important difference that the algorithm attempts to find a *stable* region of a network (crashed region) to operate on.

Consensus [5, 8] and leader election [14, 18] are both well-studied fields, although most approaches do not address the ad-hoc group formation problem; i.e. the inter-dependency that arises between those who are *agreeing* (the border set) and that which they are *agreeing to* (the crashed region, and thus constituency of the border set itself). Our work has however some similarities with consensus with unknown participants, where the set of participants is fixed, but unknown to the nodes involved [12, 6, 7, 3]. These works introduce the notion of a participant detector (PD) and study the properties this detector should fulfil to permit consensus under different assumptions.

These works are however quite different from what we are proposing, in that in our case participants are not only unknown, but evolve as failures occur. Our work also puts a strong focus on scalability with the *locality* property.

The service we propose is also related to group membership [9]. Deciding on a view in our protocol can be seen as the equivalent of installing a view. The link is particularly true with partitionable group membership (PGM) services [15, 10, 1], which look at how successively installed views should evolve to ensure that both reachability and unreachability between nodes are reflected in their installed views.

As in partitionable group membership, our service requires views held by nodes to converge when these nodes enter a particular relationship. This relationship depends on reachability in PGM, while ours arise when two nodes propose views that overlap (CD6).

The key difference however is that, whereas PGM services are defined in terms of eventual convergence of installed views, our specification is stricter in that nodes can only decide once on a given region (CD1), and must therefore detect when they have reached a convergent state, while insuring liveliness in the system (CD7).

5 Conclusion

In this paper we have formally specified a service for the convergent detection of crashed regions, where the nodes of an arbitrary large distributed system attempt to reconcile their views of neigbouring crashed regions. We have described a fault-tolerant solution to this problem, and proved its correctness. One key aspect of our specification is that it only involves nodes bordering a crashed region (*locality*), and requires nodes to explicit decide when they've converged on a unified view.

Beyond the detection of correlated crashed regions, we think this form of agreement can be seen as a particular case of a wider class of algorithms that attempt to create local collective knowledge about some distributed condition in a manner that is both deterministic and scalable. Scalability here means costs only depend on the 'extent' of the knowledge to be constructed, independently of the actual size of the system, a powerful property in very large systems.

Being crashed can also be seen as a particular case of stable property, and it could be interesting to see how this work could be extended to the detection of connected regions of nodes that share a given stable predicate (say a particular stable state). A further challenge could be to investigate how the notion of predicate-based regions and the properties of the corresponding agreement protocols could be evolved to tackle unstable properties.

References

1. Babaoglu, O., Davoli, R., Montresor, A.: Group communication in partitionable systems: Specification and algorithms. Tech. Rep. UBLCS-98-01, University of Bologna (1998)
2. Baldoni, R., Bertier, M., Raynal, M., Tucci Piergiovanni, S.: Looking for a definition of dynamic distributed systems. In: Procs of the 9th Int. Conf. on Parallel Comp. Techn., PaCT 2007, Pereslavl-Zalessky, Russia, September 3-7, 2007. LNCS, vol. 4671, pp. 1–14. Springer (2007)
3. Bar-Joseph, Z., Keidar, I., Lynch, N.: Early-delivery dynamic atomic broadcast. In: Proc. of the 16th Int. Symp. on DIStributed Comp. (DISC). DISC '02, vol. 2508, pp. 1–16. Springer, Toulouse, France (2002)
4. Bonnet, F., Raynal, M.: The price of anonymity: Optimal consensus despite asynchrony, crash, and anonymity. ACM Trans. Auton. Adapt. Syst. 6(4), 23:1–23:28 (Oct 2011), http://doi.acm.org/10.1145/2019591.2019592
5. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM 32(4), 824–840 (1985)
6. Cavin, D., Sasson, Y., Schiper, A.: Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Research Report IC/2005/026, EPFL (2005)
7. Cavin, D., Sasson, Y., Schiper, A.: Consensus with unknown participants or fundamental self-organization. In: Procs. of the 3rd Int. Conf. on Ad-Hoc, Mobile, and Wireless Networks, (ADHOC-NOW 2004). LNCS, vol. 3158, pp. 135–148. Springer, Vancouver, Canada (Jul 22-24 2004)
8. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. of the ACM 43(2), 225–267 (1996)

9. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comp. Surveys 33(4), 427–469 (2001)
10. Fekete, A., Lynch, N., Shvartsman, A.: Specifying and using a partitionable group communication service. ACM Trans. Comput. Syst. 19(2), 171–216 (2001)
11. Friedman, R., Raynal, M., Travers, C.: Two abstractions for implementing atomic objects in dynamic systems. In: Proc. of the 9th Int Conf on Principles of Dis. Sys. pp. 73–87. OPODIS'05, Springer, Pisa, Italy (2006)
12. Greve, F., Tixeuil, S.: Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In: Proc. of the 37th IEEE/IFIP Int. Conf. on Dependable Sys. and Networks (DSN'07). pp. 82–91. Edinburgh, UK (2007)
13. Guerraoui, R., Rodrigues, L.: Introduction to Reliable Distributed Programming. Springer (2006), 300 pages, ISBN 978-3540288459
14. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. and Comp. 88(1), 60–87 (1990)
15. Keidar, I., Sussman, J., Marzullo, K., Dolev, D.: Moshe: A group membership service for WANs. ACM Trans. Comput. Syst. 20(3), 191–238 (2002)
16. Porter, B., Taïani, F., Coulson, G.: Generalised repair for overlay networks. In: 25th Proc. of the IEEE Symp. on Reliable Dist. Syst. (SRDS'06). pp. 132–142 (2006)
17. Raynal, M.: Communication and Agreement Abstractions for Fault-Tolerant Asynchronous Distributed Systems. Synthesis Lectures on Distributed Computing, Morgan & Claypool (2010), 273 pages, ISBN 978-1-60845-293-4
18. Singh, S., Kurose, J.F.: Electing "good" leaders. J. of Parallel and Distributed Comp. 21, 184–201 (1994)