Immunizing Systems from Distant Failures by Limiting Lamport Exposure

Cristina Băsescu
Swiss Federal Institute of Technology (EPFL)
cristina.basescu@epfl.ch

Bryan Ford
Swiss Federal Institute of Technology (EPFL)
bryan.ford@epfl.ch

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.
>
> Leslie Lamport

ABSTRACT

Failures far away from a user should intuitively be less likely to affect that user. Today's ecosystem miserably fails this test, however, despite high-availability best practices. Correlated and cascading failures – triggered by misconfigurations, bugs, and network partitions – often invalidate assumptions of failure independence. We propose that distributed services need not and should not expose local activities to distant failures or partitions, no matter how severe. Limix is an *exposure-limiting* architecture, guaranteeing that neither the availability nor the performance of strongly-consistent accesses within a local area may be impacted by distant failures. Preliminary results suggest that infrastructures today could use Limix to limit exposure at a manageable cost.

ACM Reference Format:
Cristina Băsescu and Bryan Ford. 2021. Immunizing Systems from Distant Failures by Limiting Lamport Exposure. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3484266.3487387

1 INTRODUCTION

Distributed services – especially cloud services – generally aspire to be fast and always available from anywhere. The CAP theorem [5, 6], however, presents us in theory a simple choice of two among the three properties of consistency, availability, and partition-tolerance. Cloud platforms attempt to simplify this choice by reducing network partitions to "rare events": assuming partitions are sufficiently unlikely, we can achieve strong consistency with high availability.

In practice, however, individually-rare but correlated or cascading failures frequently puncture the cloud's illusion of having "solved" the CAP tradeoff. Leslie Lamport's quote above, while dating from the 80s, pointedly describes today's state of affairs. Reality abounds with unavailability and slowdowns, caused by misconfiguration [18, 23], cascading failures due to partially functional equipment [15], and partitions due to software bugs [3, 9], despite best-practice geo-replication. A single Fastly customer recently triggered a bug causing a global outage [11]. Failures such as these result in widespread downtime and loss of revenue [2].

We ask a simple question: is it truly acceptable that "the failure of a computer you didn't even know existed" – located anywhere in the world and run by one of many companies you also probably didn't even know existed – "can render your own computer unusable"? Is it fair or responsible that we *expose* users continually to a vast multitude of failure risks that are both *non-transparent* (infrastructure and operators "you didn't know existed") and *unbounded* (potentially located anywhere) – no matter how simple or localized a user's actual computing needs might be? And what would the implications be for distributed service designs if we were to decide that the answer to this question is *no*?

Suppose Alice is conferencing with Bob, another user located in the same city or corporate campus, while collaboratively editing a document that is similarly accessed primarily by users in the same local area. We conceptually define the *Lamport exposure* of Alice's activity as the set of infrastructure elements anywhere that could contribute to halting her work. That is, Alice's Lamport exposure is the complete set of infrastructure failure risks – including cloud services, network, power, etc. – that her particular activities depends on, directly or indirectly, and whose individual or joint failure could ultimately grind her work to a halt.

We believe distributed systems should be designed to make Lamport exposure *transparent* to users, under the *control* of users, and ideally *limited* to the smallest set of failure risks feasible for a given application. As a purely-illustrative example, simple packet forwarding, in an idealized network run

HotNets '21, November 10–12, 2021, Virtual Event, United Kingdom

ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487387

by a shortest-path routing algorithm, has a natural *Lamport exposure-limiting* property. If Alice is communicating with Bob at a network distance of Δ, then *no* combination of node or link failures beyond a distance of Δ from Alice can halt Alice's activity – because nodes and links beyond this radius cannot be on the shortest path. Alice's chat with Bob is thus "immune" to failures farther than Δ from her, hence her chat's Lamport exposure is *limited* to a radius of Δ.

The real Internet fails to guarantee such exposure limiting even for basic TCP/IP communication, due to congestion or complex off-path failures such as BGP hijacking, for example. And almost no complex applications or services higher up the stack – especially those requiring strong consistency – currently guarantee any readily-definable exposure limits. If Alice's activity A depends on strongly-consistent state replicated globally, then A's Lamport exposure is generally a large, global set of devices that Alice probably "didn't even know existed" – so A is vulnerable to correlated failures and partitions arising anywhere even if they are uncommon.

But the most critical *needs* for access are often localized: Alice cares most that her data and services are accessible from where she usually access them. Trust is also localized: people tend to trust local businesses [16] and governments [4] more and may similarly be more willing to trust digital service providers if they can show that critical services depend only on local resources. Thus, limiting a service's Lamport exposure, if achievable, could translate into availability guarantees more meaningful to users.

Failures far away from a user should intuitively be less likely to affect that user, but the current ecosystem completely fails to fulfill this intuition. Can we build services that are *usually* available from anywhere, but which can offer *transparency* about their dependencies, and *guarantee* accessibility from where they are typically used or most needed – even in the face of more distant failures or partitions? This paper argues that that distributed systems designers and practitioners can and should build reliable, fast systems by making Lamport exposure a core consideration in their design.

How exactly do we even *define* or *measure* Lamport exposure precisely, let alone *control* or *limit* it systematically? This paper only scratches the surface of this hard problem by examining a few key principles, challenges, and potential approaches to limiting exposure. The next section outlines the current infrastructure context and opportunities. Section 3 describes key research questions for exposure-limiting systems, and Section 4 sketches a preliminary system design, Limix, that addresses practical concerns such as cost and scalability.

2 CONTEXT AND OPPORTUNITIES

We review the current context in terms of architectures for distributed systems deployments. With example applications, we explain that typical deployments may create implicit but unnecessary dependencies, leading to a suboptimal Lamport exposure. Finally, for each example, we give some insights of how to design and deploy systems with a low exposure.

2.1 Geo-Replicated Cloud Services

It is customary to deploy cloud services on a region level and manage cross-region deployments manually. This trend matched the restricted offering of cloud providers that started with a few disjoint, coarse-grained regions, roughly matching (sub)continents. Several providers now lower their latency to users by offering more than 25 geographic regions and 80 availability zones (AZs), with further expansion plans [1, 7, 17]. Despite this surge, the deployment of cross-region and cross-AZ services remains largely the burden of the programmer, and perhaps for a good reason. Providers merely ensure infrastructure-level independence between AZs, leaving customers to handle application-specific dependencies.

Geo-replication is not enough. Best-practice geo-replication across regions cannot alone limit Lamport exposure. Consider a collaborative document-editing application where users in different regions edit the document. Storing the document in one fixed region penalizes far-away users with higher exposure to latency and availability risks. Geo-replicating the document across regions requires coordination for consistency among replicas. Such an application typically masks independent replica failures or partitions via consensus [13, 19], which increases exposure in two ways: (1) Data-plane: Even if the requested document has a nearby replica, the user may be unable to access the item without synchronizing with a quorum of replicas or with the leader, which may be slow or unreachable. (2) Control-plane: Even if enough document replicas are in the user's zone, merely locating the document often depends on global state that may be located anywhere. Location metadata cannot usually migrate along with the data, because the system requires a fixed, known entry point for lookup. Akkio, a recent geo-replicated KV store [2], migrates data but not the location database, for example.

Opportunity. We propose Limix, an *exposure-limiting architecture* addresses this challenge by removing false or implicit but unnecessary dependencies. In Limix's control plane, a set of potentially-overlapping protection areas or *zones* each runs an independent distributed discovery service. Each zone's discovery service ensures that all users within the zone can locate and access any data-plane object or *item* in the same zone without availability or performance dependencies outside that zone. If the (most recent version of the) document is not in that zone, Limix's lookup automatically proceeds to the next-larger overlapping exposure zone. A data-plane item may still be geo-replicated across multiple state-storage

sites (e.g., data centers); but we consider an item to be within a zone only if all of its replica sites are in that zone. For example, if a data item is replicated across sites in Germany, France, and Italy, with one replica each, then the item is "in" the EU-West but not "in" Germany. EU-West users are guaranteed to contact a consensus quorum and make progress by only relying on infrastructure and logic in the EU-West region. Lamport exposure metrics could be geographic distance or latency, leveraging the clouds' rare partitions and stable inter-region latencies (Section 3.2).

2.2 Peer-to-peer Internet applications

Unlike clouds, which have private interconnects, decentralized peer-to-peer applications must handle unpredictable wide-area network delays. These delays might be due to attacks, e.g., denial of service (DoS) or BGP prefix hijacking, but may also have benign causes, such as autonomous system (AS) misconfiguration or network sharing effects. Such events arise because the Internet is a collaboration between ASes with mutual trust. The BGP routing algorithm continues to operate largely on a trust basis, though more secure proposals such as BGPSec are advancing. Likewise, unrelated traffic shares the Internet links on a best-effort basis, which is why these links are susceptible to DoS attacks [25, 26].

Absent attacks and misconfigurations, Internet routing algorithms have, *in principle*, the Lamport exposure property we aim for. Specifically, these algorithms limit exposure defined through AS routing policies. However, this property gets lost in the upper software layers, as we explain later.

Blockchains over public networks. Consider consensus-based public blockchains handling token transfers over the Internet. Strong consistency is a must in order to avoid double spending. Consider a transaction between a sender Alice in France and a receiver Bob in Germany. Bob needs to wait for a Byzantine quorum of validators to agree on the transaction. In a global blockchain, this quorum likely involves participants outside the borders of France and Germany, increasing the Lamport exposure with all the components and networks necessary for remote communication.

Opportunity. Blockchains might reduce Lamport exposure with a "trust-but-verify" architecture. Such architectures have been proposed in the past, but rely on selecting an unbiased sample of validators [12]. If the selected sample is far away from Alice and Bob, e.g., containing nodes in China, then Lamport exposure remains large. One approach is to define Lamport exposure along legal jurisdictions. The transaction between Alice and Bob might require only enough validators in their two countries to agree. Bob is likely to trust local validators because they are in a common legal framework, and consequences are more readily enforceable. The transaction still propagates globally along increasing exposure boundaries, which independently check the transaction and enable Bob to enforce retroactive verification.

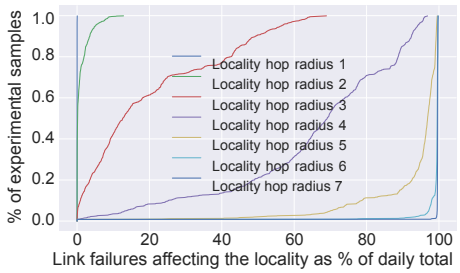


Figure 1: CDF of the propagating effect of link failures in January '20 over various exposure localities.

To get a rough, numerical estimate of exposure-limiting opportunity, we make a simplistic computation of the failure exposure of an imaginary exposure-limiting system deployed over the Internet. This illustrative example has many limitations: for example, we would like the experiment to use all types of failures across the software stack, not just presumed link-level failures. We used CAIDA data for the AS graph from January '20, which contains several daily snapshots. We denote as "stable graph" the graph with edges appearing in at least half of these snapshots, and we consider as failures those edges that appear in the stable graph but are missing from a daily snapshot. We run 1000 trials, where each trial builds localities with a random AS center and hop-count radiuses 1 to 7. For each trial, we select a random day's snapshot and count the number of failed links in the locality compared to the total number of link failures in the snapshot. Figure 1 depicts the CDF of the results. Smaller localities are exposed to fewer failures, e.g., users connecting to ISP services within a locality with hop radius 2, in 95% of the cases, are immune to 90% of the failures. Our goal is to have the same level of guarantee for the upper levels of the software stack.

3 CHALLENGES

The prospect of building systems that limit Lamport exposure presents many questions and challenges, of which we briefly explore three: layering, metrics, and zoning.

Setup. We define locality in terms of administratively-defined *zones*: e.g., each country might be a zone. We define data-plane access targets as *items*. Items may simply be, e.g., key/value pairs in a distributed data store. But Limix could also be used as a control-plane system to manage the placement, discovery, and migration of heavyweight data-plane objects such as bulk storage volumes or VMs. For this paper, we care only that items have a location and a set of exposure-limiting constraints they must satisfy. We similarly define

sites, which could be data centers, but also finer-grained edge-network sites, storage nodes in a sensor network, etc. For simplicity, a user accesses the system though some site.

3.1 Dependencies across layered systems

One challenge is complex cross-layer dependencies. Today's web/cloud paradigm is addicted to considering the location of data, metadata, and code to be unimportant. While convenient for development, this paradigm increases Lamport exposure. While Alice and Bob collaboratively edit a document, for example, layering creates many dependencies beyond the document data itself: (1) Higher in the stack, web-based user interfaces may pull icons and JavaScript from all over the world. (2) Lower in the stack, Alice's requests crossing the public internet depend on BGP routing and many network elements out of the application provider's control. (3) Network congestion or other loads on Alice's zone from requests coming from outside the zone might degrade Alice's access.

We might try to apply Limix across all layers in a system. This approach could in principle account for all dependencies, but would require many stakeholders to coordinate. A more tractable approach is to focus primarily on one layer, such as the application, accepting that uncontrolled residual Lamport exposure may persist in other layers. While likely impossible to eliminate, applications might reduce residual Lamport exposure from lower layers by building on reservation-based infrastructure proving performance isolation, such as MPLS tunnels or leased lines at the network layer.

3.2 Metrics for Lamport exposure

To protect users from distant failures, we must precisely define what "distant" means. Several different matrics may be appropriate, depending on the situation.

One approach leverages administrative or legal boundaries, such as countries or economic areas. For data or services in Germany, for example, regulatory considerations might demand a metric that treats all sites outside the EU as more distant than other sites in the EU, independent of geography.

A second obvious metric is geographic straight-line distance. Because users within geographical proximity tend to interact more [22], this metric improves the resilience of common-case interactions. These first two metrics could use public geo-location databases to define zones, yielding static zones incurring low administrative overhead.

A third distance metric is round-trip time (RTT), particularly relevant for time-sensitive use cases such as gaming and voice calls. A fourth metric might be network hop count, on the grounds that network paths with fewer hops are likely to have fewer hidden dependencies and thus be more resilient.

3.3 Defining exposure-limiting zones

Given a distance metric, how precisely do we define exposure-limiting zones, and with what structure and granularity?

We may wish to locate sites not in just one zone each, but in several overlapping zones to satisfy multiple exposure-limiting constraints. Addressing concerns of data protection and sovereignty [20, 21], Limix can simultaneously guarantee for example that all users within Germany can access an item in Germany without any availability dependencies outside of Germany, *and* that all users within the broader European Union (EU) can access the *same* item without dependencies outside the EU. Limix parallelizes both reads and updates of location hints across all relevant zones, ensuring that an item may migrate from Germany to Austria for example, for either automated or administrative reasons, while continuously limiting exposure to the enclosing EU zone.

Even absent explicit contractual or regulatory constraints, ordinary users dislike it when their local activities are brought to a halt by distant outages across the globe. Addressing this common-case challenge, Limix's *autozoning* scheme builds on compact graph summarization theory [27, 28] to construct exposure-limiting zones automatically based on any distance metric, such as those defined above in Section 3.2. For any user accessing an item of interest from a distance Δ, autozoning limits the access's exposure to a perimeter of at most $O(\log N) \times \Delta$ around the item. Limix thus guarantees that failures or slowdowns far away cannot impact the availability or performance of this user activity. Although autozoning might create several overlapping zones of varying sizes, it ensures that any item need be in at most $O(\log N)$ zones, and that each zone's discovery service need only bear the aggregate load of users within the same zone. A detailed description, however, is out of the scope of this paper.

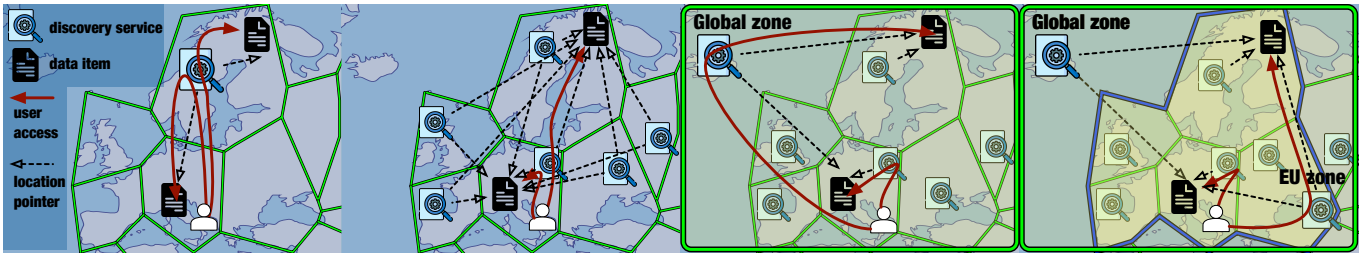
4 DESIGN

We now outline the design of Limix, motivated by four main goals: (1) providing strong exposure-limiting guarantees, (2) satisfying simultaneous constraints, (3) spreading workload across zones, and (4) enforcing strong data-plane consistency.

4.1 Item discovery

Limix needs to enable clients to find data located anywhere, regardless of where a lookup is initiated. Exposure-limiting item discovery is challenging, however. Figure 2a illustrates a straightforward but inadequate approach, relying on a central service to store the item discovery metadata. This service increases the user's exposure beyond the perimeter of the user's and data's common zone. The single zone may also become overloaded with requests from all zones.

We can make this strawman scalable by distributing the discovery service across many or all zones, using standard



(a) Global discovery service (DS), holds pointers to all data items. **(b) Full DS per zone, each maintaining pointers to all data items.** **(c) Local DS per zone, each holding pointers *only* to zone local data items.** **(d) Limix DS with overlapping zones, points to zone local data items only.**

Figure 2: Strawmen for the discovery service (DS).

techniques such as consistent hashing of keys. Physalia [3] takes this approach in its discovery cache, for example. This strawman still increases a node's Lamport exposure beyond its zone boundaries, however. To locate a particular item, a client may need to query discovery service nodes outside the zone one holding the requested data. A partition might thus prevent the client from reading the item's location, although the partition did not isolate the client from the data itself.

Limix thus needs to ensure that a user in a given zone can always find an item within the same zone using *only* resources within that zone. Efficiently collocating data and metadata so that they have the same Lamport exposure represents the first major technical challenge for Limix, which we address by having a distributed discovery service per zone.

4.2 Discovery load on (local) zones

To enable clients to find any item starting from any zone, our next strawman would be to replicate *all* discovery metadata within *all* zones, as depicted in Figure 2b. This approach invites the question: What is the maintenance workload imposed on each zone? Consider the case of updating the discovery service after an item insertion or migration. Either the destination zone could push the item's new location to all zones, or the user's zone could pull the item's new location on demand. Both the push and pull approaches may incur $O(n)$ load and communication overhead *per client request* for n zones.

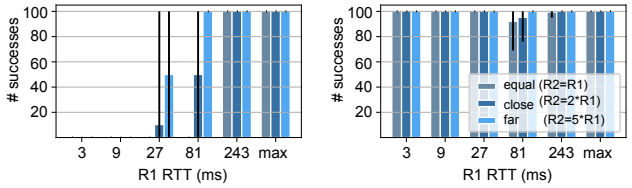
Limix instead spreads discovery loads, and limits the burden on small zones, by organizing a default global zone that overlaps with all local zones. While the global zone must handle the storage and lookup costs of all items, it can also spread this load globally across all sites. In our next strawman illustrated in Figure 2c, local zones store the location only for items they contain. In contrast, the global zone always serves as the master reference point, whose globally-distributed discovery service knows every item's location. Every zone propagates location updates to the global zone. Instead of $O(n)$, per-item update overhead becomes $O(1)$. A user queries only one local discovery service and the global one, without incurring load on other small zones. Location updates propagate only eventually, off the critical path, to limit the source zone's exposure to failures beyond its borders.

With this approach, each zone has its own discovery service that stores "location hints" for where an item was last known to be located. Because the location hints propagate only eventually, however, metadata might get out of date. For example, an item's location could time out and get evicted from a zone's discovery service, e.g., due to a long-term network partition. Then, clients on the "wrong side" of the partition cannot distinguish whether the target item no longer exists or is merely unreachable momentarily. Despite each zone's discovery service being an eventually-consistent cache, Limix can ensure strong consistency for the data plane (Section 4.4).

4.3 Item placement and zone overlap

Aside from the global zone, we assumed so far that local zones are disjoint. This assumption has a significant limitation, however: it cannot support *simultaneous* exposure-limiting policies, which may apply by law or contractual obligation. An item located in Germany may need to be accessible by users in Germany with Lamport exposure limited to sites within Germany, *and also* ensure that any user in the EU can access the same item with exposure limited to the EU. In the above strawman, EU users outside Germany must query the global discovery service, yielding global Lamport exposure.

Fortunately, we can address this problem by allowing local zones to overlap. All zones have a discovery service, and every zone propagates location updates to larger overlapping zones, up to the global zone. Update overhead increases slightly compared to the single global zone, from $O(1)$ to $O(v)$, where v is the maximum overlap depth. However, this approach limits Lamport exposure to the smallest zone containing both the item and the user accessing it. Location updates propagate only eventually, outside the critical path, to limit the source zone's exposure to failures outside its borders. Figure 2d illustrates the final discovery service architecture.



(a) Vanilla deployment. **(b) Limix autozoning.**

Figure 3: Availability during network partitions on AWS.

4.4 Data-plane consistency and migration

Limix may be used only as a control-plane service agnostic to the data plane design, but it might also be applied to the data plane. We envision this data-plane design similarly leveraging the zoned architecture to spread data-access overheads and limit load on local zones. Each zone is still independent, e.g., a data store using its internal mechanisms to replicate the data within the zone. For strong consistency across zones, we can apply standard token-passing techniques [24] to identify uniquely the most recent version of an item.

An item's placement may need to change, e.g., due to client-perceived performance and load-balancing algorithms. A location change could also result from a policy change, e.g., a new constraint on which zone(s) an item is allowed to be placed in. One key technical challenge is ensuring strong consistency during an item's migration. Our prototype Limix design takes the following approach: (1) Commit a record at the old site indicating the item is being migrated to the new site and should no longer be updated at the old site; (2) Migrate the item's data-plane state to the new site; (3) Commit a record at the new site indicating migration is complete and the item is now usable at its new site; (4) Update discovery service information independently in parallel across all zones containing either the item's old or new location; and (5) Finally, delete the item's state at the old site. This migration process might be driven either by a zone hosting the item or by a client with administrative control over the item.

4.5 Preliminary results

We implemented a preliminary Limix prototype including the above discovery service and an autozoning scheme (Section 3.3). We applied Limix to CockroachDB, a strongly-consistent key-value store, without changing its code base.

Figures 3a, 3b depict experimental results on an AWS deployment spanning 20 regions. The two graphs summarize availability for CockroachDB accesses, where all users of a data item of interest are located within a locality of a certain radius R_1. In each experiment, a network partition breaks all connectivity beyond a certain radius $R2 \geq R_1$ from the same center. Because vanilla CockroachDB uses consistent hashing to spread replicas around the world disregarding access locality, it exhibits catastrophic availability failures when a localized set of users is sharing a data item while partitioned from the rest of the Internet (the left half of the left figure). Limix CockroachDB, in contrast, preserves availability with almost perfect resilience in such "hard" cases for traditional geo-replicated systems, by ensuring that the replicas of an item are reachable within the same zone as its principal users, even when the rest of the network is unreachable.

5 RELATED WORK

CAP tradeoffs. Confronted with the CAP theorem, much research focuses on flavors of consistency. Many systems relax consistency in favor of availability during partitions, for example. Segmentation [6] identifies parts of the application or cases that require different flavors or consistency: e.g., flight seat reservation requires strong consistency only when a few seats are left. Gemini [14] distinguishes access types that require a strongly- or eventually-consistent reply. Seredinschi et al. [8] provide the user with several replies, increasing in consistency guarantees, enabling the client to perform speculative work. In Limix, all accesses are strongly consistent, and the focus is to provide the smallest possible exposure to availability failures and slowdowns.

Availability metrics. Unlike the availability metric that Hauer et al. [10] recently proposed, which *reactively* analyzes failures after they occur, Limix *proactively* limits exposure in the first place. Limix's Lamport exposure concept ensures worst-case guarantees for a user, including rare events that might not significantly affect the median availability, but still account for many hours of downtime. In contrast with the *blast radius* notion proposed by Brooker et al. [3], which attempts to reduce the damage caused by a partition, Limix focuses on users, aiming to insulate their accesses from *any* partitions or slowdowns outside a relevant local zone.

6 CONCLUSION

We believe that distributed systems can, and should, limit the Lamport exposure of local user activities to distant failures. An exposure-limiting system design like Limix offers users higher immunity to remote failures and slowdowns when accessing the local data and services they need most. Preliminary results suggest that exposure-limiting systems are feasible despite many open questions and challenges.

Acknowledgments

We thank Carmela Troncoso, David Lazar, Vero Estrada-Galiñanes, Kirill Nikitin, and the anonymous reviewers for their valuable feedback. This research was supported in part by EU Horizon 2020 grant 825377, Handshake, Oracle, the AXA Research Fund, and US ONR grant N000141912361.

REFERENCES

[1] Amazon. 2021. Amazon Web Services cloud regions. https://aws.amazon.com/about-aws/global-infrastructure/?p=ngi&loc=1. (2021).

[2] Muthukaruppan Annamalai, Kaushik Ravichandran, Harish Srinivas, Igor Zinkovsky, Luning Pan, Tony Savor, David Nagle, and Michael Stumm. 2018. Sharding the Shards: Managing Datastore Locality at Scale with Akkio. In *Conference on Operating Systems Design and Implementation (OSDI)*.

[3] Marc Brooker, Tao Chen, and Fan Ping. 2020. Millions of Tiny Databases. In *Conference on Networked Systems Design and Implementation (NSDI)*.

[4] Kathy Frankovic. 2020. Americans trust local governments over the federal government on COVID-19. https://today.yougov.com/topics/politics/articles-reports/2020/04/27/americans-trust-local-governments. (2020).

[5] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *ACM SIGACT News* 33, 2 (2002).

[6] S. Gilbert and N. Lynch. 2012. Perspectives on the CAP Theorem. *Computer* 45, 2 (Feb. 2012).

[7] Google. 2021. Google cloud regions. https://cloud.google.com/about/locations. (2021).

[8] Rachid Guerraoui, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2016. Incremental Consistency Guarantees for Replicated Objects. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 169–184.

[9] Andreas Haeberlen, Alan Mislove, and Peter Druschel. 2005. Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures. In *Symposium on Networked Systems Design and Implementation (NSDI)*.

[10] Tamás Hauer, Philipp Hoffmann, John Lunney, Dan Ardelean, and Amer Diwan. 2020. Meaningful Availability. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[11] Alex Hern. 2021. Fastly says single customer triggered bug behind mass internet outage. https://www.theguardian.com/technology/2021/jun/09/fastly-says-single-customer-triggered-bug-that-caused-mass-outage. (May 2021).

[12] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *SPIEEE Symposium on Security and Privacy (SP)*. IEEE, 19–34.

[13] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News* 32, 4 (Dec. 2001), 51–58.

[14] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. 2012. Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary. In *Conference on Operating Systems Design and Implementation (OSDI)*.

[15] Tom Lianza and Chris Snook. 2020. Cloudflare outage. https://blog.cloudflare.com/a-byzantine-failure-in-the-real-world/. (Nov. 2020).

[16] Ben Lobel. 2020. Local businesses are more trusted than large enterprises, finds survey by Yell. https://smallbusiness.co.uk/local-businesses-trusted-large-enterprises-2538416/. (2020).

[17] Microsoft. 2021. Microsoft Azure cloud regions. https://azure.microsoft.com/en-us/global-infrastructure/geographies/. (2021).

[18] Netscout. 2021. Netscout security report. (2021). https://www.arbornetworks.com/resources/infrastructure-security-report.

[19] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *USENIX Annual Technical Conference (ATC)*.

[20] Zachary N. J. Peterson, Mark Gondree, and Robert Beverly. 2011. A Position Paper on Data Sovereignty: The Importance of Geolocating Data in the Cloud. In *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.

[21] Renata Ávila Pinto. 2018. Digital Sovereignty or Digital Colonialism? *SUR* 15, 27 (July 2018), 15–27.

[22] Salvatore Scellato, Cecilia Mascolo, Mirco Musolesi, and Jon Crowcroft. 2011. Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades. In *Proceedings of the International Conference on World Wide Web (WWW)*.

[23] Laura Stevens. 2017. Amazon Finds the Cause of Its AWS Outage: A Typo. (2017). https://www.wsj.com/articles/amazon-finds-the-cause-of-its-aws-outage-a-typo-1488490506/.

[24] D. Stevenson. 1989. Token-based consistency of replicated servers. In *Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage*.

[25] Ahren Studer and Adrian Perrig. 2009. The Coremelt Attack. In *ESORICS*.

[26] Min Suk Kang, Soo Bum Lee, and V.D. Gligor. 2013. The Crossfire Attack. *IEEE Symposium on Security and Privacy (S&P)*.

[27] Mikkel Thorup and Uri Zwick. 2001. Approximate distance oracles. In *ACM Symposium on Theory of Computing*. 183–192. https://doi.org/10.1145/1044731.1044732

[28] Mikkel Thorup and Uri Zwick. 2001. Compact routing schemes. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM Press, New York, NY, USA, 1–10. https://doi.org/10.1145/378580.378581