Scalable, Privacy-Preserving Contextual Communication

ABSTRACT

In this paper, we present a new communication abstraction referred to as "contextual communication" that generalizes name- or address-based communication to communication based on arbitrary attributes. Compared to prior approaches, our primary contribution is the design, implementation, and evaluation of CNS, a system that achieves provably scalability of $\Theta(\sqrt{n})$, and our hardness results help shed light on why prior approaches to this seemingly well studied problem achieved suboptimal scalability. Furthermore, the accompanying CNS federation architecture and our proposed privacy protocol allow users to rest assured that the CNS service provider itself can not infer sensitive user information. We have designed and implemented the proposed approach in a system, and our experimental evaluation with microbenchmarks as well as a case study based on a real hazardous weather warning application shows that the CNS can dramatically enhance the scalability of contextual communication compared to the state-of-the-art.

1. INTRODUCTION

Many modern applications, especially mobile apps, can benefit from a *contextual communication*, an abstraction that generalizes name- or address-based communication to more general attributes. For example, peer-to-peer taxi or ride sharing apps (e.g., Uber) issue a search for taxis matching passenger criteria; local-business apps (e.g., Yelp) issue a search for establishments matching user preferences; hyperlocal notification apps for public safety [9, 17] or hazardous weather [7, 15] issue alerts to devices matching a geofence or other attribute descriptors.

Our position is that contextual communication as a primitive as universal as TCP/IP sockets can significantly simplify the development and maintenance costs of mobile applications. Today, every app developer has to redundantly set up and maintain a cloud database to collect attribute information reported by mobile devices, use them to resolve contextual queries, and employ narrow smartphone notification service APIs to dispatch notifications. Our vision of contextual communication is a universal primitive like `bind(lat, long, radius)` to bind a socket-like structure to a *context*, i.e., an attribute constraints descriptor, so that any messages written to it are automatically sent to end-client principals matching the context. Contextual communication as a universal infrastructure service can simplify application development as developers don't have to worry about managing group membership or even be aware of the destination principals actually contacted; they only specify their intent and the rest is as easy as what students learn in a first course on networking today.

The above vision is attractive but challenging to realize, and perhaps the most compelling evidence of both is the fact that we are by far not the first to pursue this vision (refer to §2.1 for a long lineage of related work), yet Internet applications today still work with the same host-to-host communication primitive from over four decades back [16] and widely criticized by networking researchers as an abstraction misaligned with application needs (e.g., location-identity conflation for mobility [34, 31, 22]; lack of information-centrism [25, 21, 6]; etc.).

Our work draws inspiration from more recent work in this lineage on MobilityFirst [36], a so-called "future Internet architecture" [20], which advocates a global name service (GNS) as an indispensable part of a mobility-friendly, secure network architecture. Sharma et al. [35] hinted that a scalable solution for high mobility in a name- or address-space can form the foundation for handling mobility in any general attribute space, and presented a proof-of-concept demonstration (§4.3.3 [35]), but left out the details of making it work at scale as a general-purpose communication abstraction. Our work is a step towards addressing the hard scalability and privacy challenges head-on.

Our primary contribution is the design and implementation of the *contextual notification service*

(CNS), which at its core is simply a database service but with two novel traits. First, it enables multidimensional attribute-range queries in a *provably* and massively scalable manner. Our analytical contributions therein, to our knowledge, are the first to formally describe the scalability problem, show that linear scaling (the ideal case) is impossible (§3.2), shed light on why prior approaches in this seemingly old problem domain are either unscalable or suboptimal, and present a *region mapping* (§3.3) algorithm that is asymptotically provably optimal. Our formal scalability results (§3) and proofs (§6) are rigorous, yet simple to understand.

Second, our proposed GNS-CNS separation enables an important property, namely, privacy from the service providers themselves. Without such an assurance of privacy, our position is that the vision of contextual communication would be a non-starter. Our protocols for updates, searches, and lookups anonymize attribute information in a manner that, despite collusion, the CNS or the GNS can only know of the existence of attribute-value pairs but not be able to correlate them to user identities in the absence of external information or side-channels.

We have implemented a prototype of the CNS that is interoperable with the open-source MobilityFirst GNS system. Our prototype-driven experiments on Emulab clusters show that the CNS significantly outperforms state-of-the-art search systems in the search and update capacity achieved as a function of the cluster size, and that privacy-preservation imposes only a modest performance overhead on the overall capacity and request latencies. Finally, we conduct a semi-realistic application case study motivated by a hazardous weather alerting app, ANON[15], currently being beta-tested with the help of over 40 emergency personnel in and around the ANON metroplex using high-fidelity weather data from next-generation X-band radars [14] spread over a ≈12,500 sq. mi area (compared to ≈150 previous-generation "NEXRAD" NWS radars that provide blanket coverage for the U.S. and are used by practically all hazardous weather applications today). Our results suggest that the CNS can dramatically increase the search query volume compared to existing systems such as the GNS or HyperDex [19].

2. BACKGROUND AND RELATED WORK

The vision of contextual communication is not new. To our knowledge, our two main intellectual contributions—provable scalability (or hardness) and privacy from the service provider—haven't been addressed before. However, we do build upon a large body of prior work with closely related goals.

2.1 Contextual communication lineage

The broad vision of contextual communication (variously referred to as attribute-based addressing or communication, intentional naming, etc.) has long been around. Lampson alludes to it as "descriptive names" [29] but admits it is unclear how to specify or implement such a name service. Intentional Naming System (INS) is similar to contextual communication in its ability to specify *intent* as opposed to endpoint addresses but differs in many respects in that INS (1) routes multicast data through integrated resolver nodes (intentional multicast), but our proposed design only provides the control plane offloading data transfer to the underlying network; (2) does not support range queries, which makes the problem much harder; (3) is designed for intradomain deployment; and (4) to quote, "*most importantly, needs to incorporate security mechanisms in the naming architecture before a more wide-scale deployment*". In comparison, our vision of contextual communication, specifically the control/data factoring and the security mechanisms, are inspired by more recent work on the MobilityFirst [36] Internet architecture.

Pub-sub systems broadly overlap with our goal of matching attributes to events, as do other pub-sub-like "information-centric" network architectures like PURSUIT [21], TRIAD [25], or NDN [37] that match interests to data over a hierarchical name space. These designs inherently prize a data-pull abstraction (via stateful interests or subscriptions) and focus on scalable data flow in contrast to our focus on a more traditional IP- or SMS-like push abstraction and a scalable control plane. Our position is that a push abstraction is intrinsically better suited for a number of agile notification scenarios such as emergency alerts, an application space of particular interest to us; furthermore, it is unclear how to take off-the-shelf pub-sub-like systems and adapt them to a scalable and privacy-preserving instantiation of "Uber", or public safety [9, 17] or hazardous weather apps [7, 15], etc. without being stymied by the very challenges we address in this paper.

2.2 Scalable multidimensional search

An ideal system would be *linearly scalable*, i.e., adding more resources results in a proportional increase in the overall load it can accommodate without significantly degrading request latencies.

To appreciate why scaling contextual communication is challenging, consider a traditional database storing geolocation attributes of end-users. A notifier seeking to notify users in a geofence would perform a `select` for all keys whose `[lat, long]` attributes lie in some range, which is efficiently accomplished using B-tree indexes in a centralized (single-machine) setup. In a massively geo-distributed setup such as that envisioned for a GNS, careful placement of name records identified by their primary key or *globally unique identifier* (GUID) can ensure low-latency lookups and linearly scaling lookups and updates. However, achieving scalable key-based lookup/update performance as well as scalable value-based search is fundamentally hard.

At the heart of this challenge is that textbook techniques for scalable load balancing to map records to machines (such as consistent hashing based on randomization or planned placement strategies [35]) are fundamentally at odds with search efficiency, a goal that requires mapping records predictably to machines based on the values of their attributes, and whose computational and storage complexity exacerbates with the number of searchable attributes. A further challenge is to adapt this mapping to the possibly non-uniform nature of the distribution of the values of the attributes over their ranges and the distribution of the attribute ranges in search queries. The MobilityFirst GNS [24] currently adopts a rather simplistic "QueryAll" [1] approach that dispatches every contextual search query to all machines and aggregates their results, and periodically refreshes each active query until its expiration. This simplistic approach preserves lookup/update scalability but quickly hits a ceiling with respect to search scalability as increasing the number of machines linearly increases the work per search!

Systems such as Mercury [11] seek to combine the load balance benefits of randomization with multi-dimensional range queries using a DHT-based approach that provably bounds the routing hop-count to be logarithmic in the number of machines. Distributed B-tree approaches [10] seek to spread out B-tree nodes over the set of available machines in a balanced manner while maintaining the logarithmic complexity of updates and searches with respect to the total number of records and machines. Nevertheless, our strong hardness result (§3.2) rules out the existence of any distributed B-tree, DHT, or other approach that can achieve better than $\sqrt(n)$ scalability with respect to the number of machines n. It is easy to reconcile the two observations if one views a distributed B-tree (or a comparable approach) as an efficient *routing* mechanism; our hardness result rules out the existence of scalable *placement* strategies even assuming zero-cost routing.

HyperDex [19], a system that does focus on placement with single-hop routing, is closely related to and forms a baseline for our scalable mapping problem (§3.1.1). Indeed, our early efforts sought to use HyperDex or a comparable system as an off-the-shelf solution for the scalable search part of the problem. However, our attempts to aggressively stress-test its subspace partitioning algorithm made us realize that it is technically *unscalable* (as formally (§3.2) and experimentally §5) shown. Furthermore, we found that subspace partitioning as well as some implicit design assumptions in HyperDex are strictly unnecessary, which led us to a simpler, provably scalable region mapping algorithm.

2.3 Privacy-preserving searches

Database privacy systems like CryptDB [33] attempt to push the envelope of what is practical for a privacy-preserving near-general-purpose database system. Homomorphic encryption techniques [23] can work directly on encrypted data but incur a high overhead; others like order-preserving encryption [32, 12, 13] are more tractable but fundamentally leak more information than public- or symmetric-key encryption. In comparison, contextual search addresses a narrower problem but promises a theoretically tight privacy guarantee at a modest cost. Furthermore, order-preserving encryption transformations can still be overlaid on top of our design.

3. CONTEXTUAL COMMUNICATION: DESIGN AND IMPLEMENATION

The CNS has the following high-level design goals. Although each of these design goals is simple to state, it is nontrivial to achieve the combination, or even just the first and third goals.

1. **Scalability**: Increasing resource provisioning should gracefully increase the system capacity.

2. **Availability**: The CNS must be available despite the failure of a small number of machines.

3. **Privacy**: The CNS provider should provide lookup, update, and search over user attributes but itself remain oblivious to the data.

4. **Federation**: The design must permit federated deployment in a competitive market.

Our proposed high-level architecture assumes a pre-existing geo-distributed, federated global name service (GNS) (e.g., [35]). In order to achieve the above goals, we propose a design that (1) logically separates the contextual notification system (CNS)

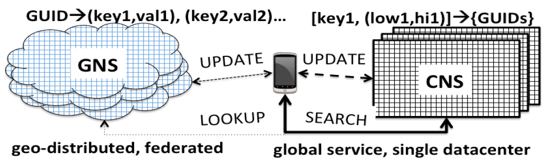

GUID→(key1,val1), (key2,val2)...
[key1, (low1,hi1)]→{GUIDs}
GNS
UPDATE
UPDATE
CNS
LOOKUP
SEARCH
geo-distributed, federated
global service, single datacenter

Figure 1: High-level GNS-CNS separation: The GNS enables rapid lookups and updates while the CNS enables scalable indexing for efficient search.

from the geo-distributed GNS, a separation also critical for privacy-preserving contextual queries (§4), and (2) relies on a novel demand-adaptive region mapping strategy. Although our design presupposes a GNS, it is largely agnostic to its implementation or the feasibility and merits of a GNS-driven internetwork architecture (as argued by [36]), and is equally applicable for more specific application scenarios such as a "privacy-preserving Uber" service. We begin with a formal introduction to the region mapping problem, which is central to scalability.

3.1 Model and background

Let $\mathcal{A}$ denote a universal set of m attributes, and $\{A_i\}_{1\leq i\leq m}$ be an arbitrary but fixed ordering thereof, wherein each attribute A_i can take a value from a corresponding ordered range set $R(A_i)$, and let $H(\mathcal{A})$ denote the corresponding global *value space*, i.e., the set of all vectors $[v_1, \ldots, v_m]$ such that $v_i \in R(A_i)$. Let $\mathcal{D}$ denote a key-value datastore or a collection of key-value records wherein each key is a globally unique identifier (or GUID) and its value is an m-dimensional vector of attribute-value pairs $[(A_1, v_1), \ldots, (A_m, v_m)]$ where $v_i \in R(A_i), 1 \leq i \leq m$. We use the notation $X.A$ to mean the value of attribute A of GUID X.

Our goal is to efficiently handle searches and updates over $\mathcal{D}$. A *search* query is represented as a vector of attribute-range pairs $[(a_1, r_1), \ldots, (a_k, r_k)]$ for some subset of attributes $\{a_i\}_{i=1}^{k} \in \mathcal{A}$ and $r_i = [low_i, high_i]$ is an interval in $R(a_i)$, $1 \leq i \leq k$, which must return all GUIDs X in the data store $\mathcal{D}$ such that $(low_1 \leq X.a_1 \leq high_1) \wedge \cdots (low_k \leq X.a_k \leq high_k)$. (Disjunctive queries are discussed in §4.4.) An *update* is a vector of attribute-value pairs $[(a_1, v_1), \ldots, (a_k, v_k)]$ for $\{a_i\}_{i=1}^{k} \in \mathcal{A}$ that assigns $X.a_i \leftarrow v_i$ for every $1 \leq i \leq k$.

3.1.1 *Value-space and subspace partitioning*

Value-space partitioning assigns subsets of the global value space $H(\mathcal{A})$ to the global set of n machines in a deterministic manner so as to improve search and update performance. A simplistic approach is to split each attribute's value range into n mutually exclusive and exhaustive (MEE) partitions that are bijectively assigned to the n machines in any deterministic manner. A strawman approach is to store each record in $\mathcal{D}$ on up to m different machines that are each responsible for the partitions in which the values of each of the m attributes lie; a search query need only contact the machines responsible for the range specified for any *one* attribute in the query, however an update may have to contact up to m machines. It is possible to do much better than this strawman, as we explain next.

The scalable region mapping problem is as follows. Let $\mathbf{P}_i$ denote a set that partitions the value range $R(A_i)$ of each attribute A_i into MEE intervals. A *region* is an element of the set $regions(\mathcal{A}) = \{\mathbf{P}_1 \times \ldots \times \mathbf{P}_m\}$ that partitions the global value space $H(\mathcal{A})$ into MEE subspaces. Let $map(r)$: $regions(\mathcal{A}) \rightarrow 2^{\mathbf{N}}$ be an arbitrary function that maps each region to a subset of all available machines.

3.1.2 *Subspace partitioning (prior work)*

A recent system, HyperDex, addresses the problem via *subspace partitioning* that (1) first partitions the universal set of attributes $\mathcal{A} = \cup_{j=0}^{k}\mathcal{S}_j$ into $k < n$ MEE subsets; (2) partitions the set of n machines also into the same number k of MEE subsets; (3) surjectively maps the subspaces on to the subsets of machines; and (4) partitions value space (as above) within each subspace $H(\mathcal{S}_j), 1 \leq j \leq k$ and its corresponding set of machines. This two-level mapping strategy can be viewed as a special case of a slightly relaxed region mapping problem that does not require regions to be mutually exclusive and exhaustive as in the formulation above.

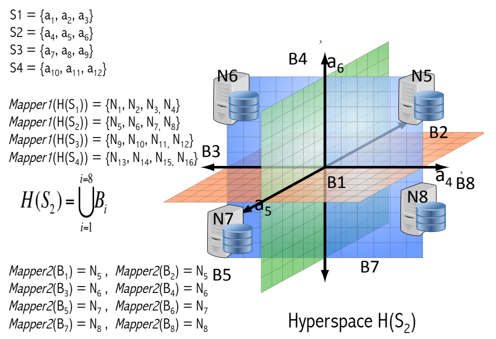

S1 = {a1, a2, a3}
S2 = {a4, a5, a6}
S3 = {a7, a8, a9}
S4 = {a10, a11, a12}
Mapper1(H(S1)) = {N1, N2, N3, N4}
Mapper1(H(S2)) = {N5, N6, N7, N8}
Mapper1(H(S3)) = {N9, N10, N11, N12}
Mapper1(H(S4)) = {N13, N14, N15, N16}
H(S2) = ⋃_{i=1}^{i=8} Bi
Mapper2(B1) = N5 , Mapper2(B2) = N5
Mapper2(B3) = N6 , Mapper2(B4) = N6
Mapper2(B5) = N7 , Mapper2(B6) = N7
Mapper2(B7) = N8 , Mapper2(B8) = N8
B4
a6
N6
N5
B2
B3
a4
B8
B1
N8
N7
a5
B5
B7
Hyperspace H(S2)

Figure 2: Subspace hashing to one-one map multidimensional regions on to server partitions.

Figure 2 exemplifies subspace partitioning with a value space of 12 attributes partitioned over a datacenter consisting of 16 machines by subdividing the value space into 4 subspaces each consisting of 3 attributes and mapping each subspace to 4 machines. Each attribute is divided into $p = 2$ partitions for a total of $p^3 = 8$ regions per subspace that are spread across the 4 machines assigned to that subspace (with 2 regions assigned to each machine). The figure shows a single subspace S_2 for simplicity with 8 regions B_1–B_8 on machines N_1–N_4.

A natural question is if subspace partitioning as above is *sufficient* to be linearly scalable or even scalable at all. Is subspace partitioning *necessary* for scalability? Is it necessary that "in general, a m dimensional hyperspace will need $O(2^m)$ servers" (§3 [19]) or for that matter a number of servers exponential in m or m/K for some constant K? Our answers to all of these questions are in the negative.

3.2 Hardness of scalable region mapping

In order to explain the necessary and sufficient conditions for scalable search, for ease of exposition, we first assume that a single update or search operation (e.g., an indexed `select`) on a single machine takes a constant amount of computation; relaxing the assumption so that search complexity is logarithmic (as with B-trees) in the total number of records stored on the machine does not affect either of the formal claims below (proved in the Appendix). We further assume that all value ranges are a finite subset of reals between an arbitrary (but finite) minimum and maximum value.

LEMMA 3.1. *For any fixed partitioning (but no replication) of records across machines, there exists an a priori known workload for which the system is unscalable, i.e., the aggregate capacity (requests/s) does not increase as the number of machines.*

We encourage the reader to glance at the proof in the Appendix as it is easy and captures the essential insight that the strong hardness result below generalizes to subsume strategies involving arbitrary combinations of partitioning and replication.

THEOREM 3.2. *For any given fraction ρ of search queries, there exist workloads containing searches and updates for which no subspace partitioning and replication scheme–even with complete knowledge of the workload–can achieve better than $\Omega(\sqrt{(n)})$ scalability unless ρ is 0 or 1.*

The hardness result is especially damning because the proof does not rely on pathological workloads; indeed, even complete a priori knowledge of a simple, uniformly distributed workload (as well as a broad class of non-uniform workloads) can not achieve linear scaling–the best case–unless the workload contains no searches or contains no updates (ρ is 0 or 1, in which case even linear scaling is trivial).

3.3 Practical region mapping strategy

The nonexistence of a linear scalable strategy is disappointing, but can we achieve the asymptotically optimal $\sqrt{n}$-scaling with a practical algorithm? The proof of the hardness result above naturally suggests the crux of a subspace mapping strategy that does achieve $\Theta(\sqrt{n})$ scaling. However achieving this scaling provably requires some workload assumptions and performing well in practice requires further heuristics. We first present a sufficient condition to ensure $\Theta(\sqrt{n})$-scaling (or simply *maximal scaling*) for a region mapping algorithm.

THEOREM 3.3. *A mapping $map(r) : regions(\mathcal{A}) \rightarrow 2^{\mathbf{N}}$ is maximally scalable if $|regions(\mathcal{A})| = \sqrt{n}$ and these regions are mapped bijectively to a set of $\sqrt{n}$ MEE subsets of the n machines, and updates and searches are uniformly distributed across all regions.*

The condition in the above theorem is simple to state and sufficient but not necessary. Furthermore, it is not necessary for the partitions of an attribute to be uniform-sized. Implementing it well in practice requires a method to partition the global value space into well balanced regions, i.e., regions that roughly incur similar update and query loads, by learning some information about workload patterns.

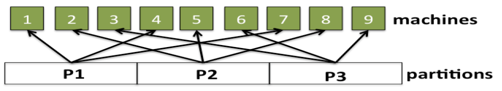


Figure 3: Maximal scalability for a single attribute.

It is easy to prove in the single-attribute case that the above condition is sufficient to ensure maximal scalability. Consider a partitioning of the attribute's value range into $\sqrt{n}$ MEE partitions each of which is assigned to $\sqrt{n}$ machines and replicated $\sqrt{n}$-fold (refer Fig. 3). Each update touches at most $2\sqrt{n}$ machines, the two (if different) machines to which the old and new partition are mapped. A search query need never touch more than $\sqrt{n}$ machines as there exists a subset of $\sqrt{n}$ machines that covers the entire value space. The proof is conceptually similar for the multidimensional case.

It is important to note that the basic heuristic in subspace partitioning [19] of partitioning the hyperspace along attributes into subspaces is not inherently flawed and is even intuitive compared to defining arbitrary regions in a multidimensional setting. However, a key design element missing in subspace partitioning is *replication* as exemplified above. Subspace partitioning maps regions bijectively (one-one) to machines and subspaces bijectively to MEE subsets of machines. However, in general, either subspaces must be replicated on more than one subset of machines or regions within a subspace must be replicated on more than one machine assigned to that subspace for provably maximal scaling. Our mapper implementation defaults to the former choice.

3.3.1 Replicated record storage

With or without subspace partitioning, search is more efficient if each machine to which a region is mapped stores the entire record, i.e., the GUID primary key and all attributes and values including those not belonging to the region (or subspace with subspace partitioning). A traditional centralized index, say based on B-trees, strictly needs to only store compact primary keys at tree nodes so as to return them as part of the result set. However, in a distributed scenario, storing only primary keys necessitates multiple rounds of searches if the search query's attributes span multiple regions; on the other hand, storing entire records allows machines mapped to a single region, say the region with the greatest overlap with the value ranges specified in the search query, to return local result sets to the "entry server" that aggregates and sends them to the querying end-client. Our findings and design in this respect are consistent with those of [19].

3.3.2 Update triggers

Update *triggers* efficiently notify stateful queries of changes to the context membership as GUIDs wander in or out of context. To this end, the CNS maintains a logical map of *active* queries, i.e., stateful queries that have been *opened* but not yet *closed* by their queriers. Any update inducing a change to the result set of an active query auto-triggers a notification to the querier containing the list of GUID *joins* and *leaves*. This event-driven design obviates more wasteful rate-limited polling approaches that appear to be currently used by GNS clients [3].

The CNS does not maintain *trigger* state—the set of active search queries and corresponding queriers—as a centralized map but instead seamlessly integrates it with the distributed region-mapping approach. The trigger for each active query is stored on the same region-mapped machines where the search query is processed to compute the result set. On each such machine, the trigger is stored in a separate database table than the table that stores the GUID primary key and its attribute values. Thus, the expected number of machines touched upon an update are at most the number of machines touched for a search or an update without triggers.

4. PRIVACY FROM PROVIDER

Our goal is to enable contextual communication while protecting the privacy of sensitive user information even from the service providers. To this end, we present a design that efficiently implements searches and updates while ensuring that no entity other than those explicitly identified in the access control list (ACL) of an attribute of a GUID can infer the value of the attribute.

4.1 GNS background

The GNS enables an ACL-protected key-value store or a collection of records $X \rightarrow [(A_1, v_1, \mathbf{acl}_1), \ldots, (A_m, v_m, \mathbf{acl}_m)]$ each of which maps a GUID X to its m-dimensional vector of attribute-value-ACL 3-tuples, wherein each $\mathbf{acl}_i, 1 \leq i \leq m$ is an ACL or a set of GUIDs authorized by X to read the value v_i of $X.A_i$. The GNS [35] allows X—the owner of the information contained in its record—to associate an arbitrary read-ACL and a write-ACL for each attribute, each of which may be specified as either a whitelist or a blacklist. For ease of exposition, we assume here that every ACL is a read-whitelist and the write-ACL for any attribute $X.A$ is a whitelist that is the singleton set $\{X\}$.

The GNS resolves simple attribute-value lookup queries in an ACL-bound manner by requiring a querier GUID Z to authenticate itself by signing the (nonced) query with Z^- and verifying the signature using Z^+ after checking that Z belongs to the ACL of the queried attribute.

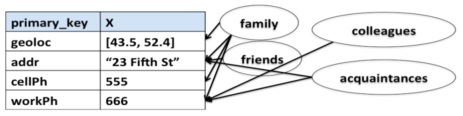
4.2 Privacy-preserving GNS–CNS design

Our threat model is that of an honest-but-curious adversary wherein the service provider follows the protocol correctly but may peek at all protocol data and metadata that it handles. We require honesty only for availability—a service provider not following the protocol can simply render the service unavailable (and probably cease to be an attractive service provider in practice). This threat model is similar in spirit to that used by oblivious cloud storage systems [2, 5, 4] or ISPs carrying sensitive end-user information over SSL without being able to infer the content of the payload.

Our approach enables privacy from the provider by physically separating the indexing subsystem, CNS, from the GNS. The GNS stores records with the GUIDs in clear but values of ACL-protected attributes in encrypted form, while the CNS stores the values in clear but the GUIDs in pseudonymized form. An end-client issues updates in two separate steps, one involving the GNS and the other the CNS, but issues searches only to the CNS.

4.2.1 *Identical read privilege pseudonymization*

An *identically read privileged* (IRP) set of a GUID X is a set of GUIDs that all have the same read privileges with respect to X's attributes, i.e., for any two GUIDs Y and Z in the IRP set, $Y \in \mathbf{acl}(X, A_i)$ iff $Z \in \mathbf{acl}(X, A_i)$ for all $1 \leq i \leq m$. A *maximal* IRP set I of X is an IRP set such that for any $Y \in \bigcup_{i=1}^{m} \mathbf{acl}(X, A_i)$, either $Y \in I$ or $I \cup \{Y\}$ is not an IRP set. The maximal IRP partitioning of X is the set of maximal IRP sets of X that form a MEE partition of $\bigcup_{i=1}^{m} \mathbf{acl}(X, A_i)$.



primary_key	X
geoloc	[43.5, 52.4]
addr	"23 Fifth St"
cellPh	555
workPh	666

Figure 4: Basis ACLs for maximal IRP partitioning.

Figure 4 illustrates a maximal IRP partitioning of a GUID X's ACLs. Pedagogically, it helps to think of maximal IRP sets as the smallest set of (the largest) "basis" ACLs from which all ACLs of a GUID can be generated. For example, if X restricts each of its attributes to be readable by the union of one or more mutually exclusive "circles" [28] such as "family", "friends", "colleagues", "acquaintances", etc., then these circles form a maximal IRP partitioning. The singleton set partitioning of $\bigcup_{i=1}^{m} \mathbf{acl}(X, A_i)$ of course trivially forms an IRP partitioning, but it is important to have a small number of basis ACLs in order to reduce the overhead of maintaining pseudonyms, as explained next.

The CNS associates a unique *pseudonym* with each maximal IRP subset of a GUID X's ACLs. The CNS stores the value of each of X's attributes in a replicated manner, once for each pseudonym, thereby commensurately increasing its storage overhead, which is why it is important to create the smallest number of basis ACLs via maximal IRP partitioning. Pseudonyms associated with basis ACLs allow the CNS to answer conjunctive queries while maintaining attribute privacy.

4.3 GNS-CNS update, search, and lookup

CNS updates. We focus on single-attribute updates for simplicity of exposition. In order to update $X.a$, an end-client (X or any entity possessing the corresponding private key X^-) issues the following message

$[\text{UPDATE}, (a, v), \mathbf{psnyms}(X, a), \mathbf{crnyms}(X, a)]$ where $\mathbf{psnyms}(X, a) = \{psnym(X, Y)\}_{Y \in \mathbf{acl}(X,a)}$ is the set of pseudonyms corresponding respectively to the basis ACLs of $\mathbf{acl}(X, a)$, and $\mathbf{crnyms}(X, a) = \{[X]_{Y+}\}_{Y \in \mathbf{acl}(X,a)}$ is the set of *cryptonyms* generated by encrypting X with the public keys of each of the members of $\mathbf{acl}(X, a)$.

CNS search. An end-client with GUID Z issues a search as a set of attribute-range pairs (§3.1) simply in clear as $[\text{SEARCH}, [(a_1, r_1), \ldots, (a_k, r_k)]]$ expecting to get back a set of cryptonyms generated by encrypting each member of the result set with Z^+, which Z can decrypt with its private key Z^-.

The CNS implements the search by storing records using a maximally scalable region mapping (§3.3) but with the difference that it uses pseudonyms received in the UPDATE messages instead of the GUID as the primary key.

GNS update and lookup. In order to update $X.a$, an end-client issues the message $[\text{UPDATE}, X, (a, \sigma(v)), \{[\sigma]_{Y+}\}_{Y \in \mathbf{acl}(X,a)}]$ signed with X^-, where the value v is encrypted with symmetric key σ, and the last argument is a certificate that only needs to be updated if the ACL changes. Upon a subsequent lookup for $X.a$ by Y, the GNS returns $[\sigma(v), [\sigma]_{Y+}]$ that Y can decrypt using Y^-. Note that the GNS stores and serves the value v only as ciphertext.

4.3.1 *Computing the maximal IRP partitioning*

Computing a small IRP partitioning is important to keep the number of pseudonyms low. Fortunately, a simple greedy algorithm can compute the MIRPP given a set of attribute-ACL pairs $(A_1, \mathbf{acl}_1)$, $\ldots, (A_m, \mathbf{acl}_m)$. The algorithm starts with a singleton set partitioning of the union of ACLs $\bigcup_{i=1}^{m} \mathbf{acl}_i$; then greedily merges as many GUIDs as possible with the first GUID to create the first maximal IRP set; and recursively does the same with remaining singleton sets. The worst-case complexity of this algorithm is $O(|\bigcup_{i=1}^{m} \mathbf{acl}_i|^2)$; a tighter bound is $O(\kappa|\bigcup_{i=1}^{m} \mathbf{acl}_i|)$ where κ is the size of the MIRPP.

4.3.2 *Privacy property*

Formalizing the privacy property offered by any system claiming to be privacy-preserving is generally not easy. What we have is an easy-to-understand (albeit weak) statement of the privacy property maintained by our CNS design with singleton IRP partitioning: *In the absence of timing or side-channel attacks, a* CNS *can not infer the value v of an attribute $X.a$ with better* accuracy*—defined as the probability of inferring the value with any given error ϵ for real-valued attributes—than an adversary with access to all clear-text attribute-value two-tuples but no information at all about primary key* GUID*s.* Note that this property holds despite collusion between the CNS and GNS and, as stated, is resistant to publicly available out-of-band information as such information is presumed available also to the adversary. Furthermore, the adversary has access only to two-tuples but no direct means to infer that two two-tuples belong to the same GUID. With maximal IRP partitioning, the privacy property is weakened accordingly to an adversary that also knows that attribute-value two-tuples corresponding to the same pseudonym are correlated.

4.4 Design implications and limitations

The CNS design comes with several caveats, some specific to our goals and others (like timing or side-channel attacks) that are well known to be notoriously hard to eliminate in general.

The ACL-protected nature of attributes means that, in general, different queriers will get different results for the same query. For example, a geo-query might return a GUID X matching the query to querier Y present in X's geolocation's ACL but not to Z not present in that ACL. Although this behavior is implied by our problem definition, it presents a usability challenge especially in emergency notification scenarios—an important motivating focus of our work. A privacy-paranoid user may choose to not make their geolocation readable by any entity (including emergency management personnel), but that means that they will not get critical notifications in potentially life-threatening scenarios.

Incomplete views like above above can also be induced by our envisioned federated design. Unlike the federated GNS design, each CNS provider is a global service provider responsible for contracting with all or most GNS providers. Incomplete coverage on part of a CNS provider may result in incomplete views.

Our threat model allows for collusion between the GNS and CNS, yet the design relies on a physcial (not just logical) separation between the two, which might seem counterintuitive at first glance. The separation is needed to make timing attacks harder, otherwise a combined GNS-CNS system can infer the correspondence between a pseudonym and its GUID as the GNS-UPDATE and CNS-UPDATE messages are likely to be temporally correlated. Any approach (e.g., based on delay noising) to thwart timing attacks would be equally effective irrespective of whether the GNS and CNS are physically or just logically separate.

The CNS design can not protect against information leakage because of external channels. For example, if a geo-query for a 1 sq. km grid returns three cryptonymized GUIDs, and the CNS provider knows out-of-band that there is just one household with three members living in that grid, it can connect the pseudonym or cryptonym to the corresponding GUID with 1/3 probability (as is allowed by Property). Data noising techniques [27, 26] can alleviate this inference risk but commensurately degrade result accuracy. (Note that differential privacy techniques [18], which offer formally tight privacy properties irrespective of external information sources, are not directly applicable here as their threat model trusts the database provider.)

Implementation.

We have implemented a prototype of the CNS with all of the CNS features described above in 29.7K newlines and 9.9K semi-colons of Java in addition to simple Python scripts for the mapper to compute the region mapping and bootstrap configuration in a workload-aware manner. The region mapper defaults to subspace replication with no replication within each subspace (refer §3.3), the setting used in all of our experiments. We additionally made small changes as needed to a fork of the open-source GNS [24] in order for a CNS client to inject updates to the GNS-CNS system.

5. EVALUATION

The goals of our evaluation are three-fold: (1) to compare CNS' scalability to state-of-the-art systems; (2) to quantify the latency and capacity impact of privacy mechanisms; and (3) to use realistic mobility and weather data to evaluate CNS' scalability with respect to the number of apps.

5.1 Experimental setup

Testbeds: We use an Emulab d710 cluster for all experiments. Each node in the cluster has 8 64-bit Intel Quad Core Xeon E5530 CPUs, 12 GB of memory and 16 GB of disk space. We configured all nodes in a LAN of 1000 Mbps and no delay emulation, and use `mysql` as the database on each node.

Workload: Our microbenchmark typically use a synthetic workload of 10K GUIDs, $m = 20$ attributes per GUID, and the value of each attribute is uniformly distributed between 1.0 and 1500.0; our qualitative findings are insensitive to the absolute range size. The search and update workloads are as follows. Search queries are a conjunction of predicates of 4 (out of 20) randomly chosen attributes. Each predicate has an interval whose lower limit is chosen randomly and the interval length is half the range size and upper limit is wrapped around the range if necessary. Update queries pick a GUID, an attribute, and a value, all uniformly randomly.

Unless otherwise specified, the experiments use a single client. As our goal is to measure server capacity, we do not send the full result set of GUIDs to the client so as to ensure that it does not become the bottleneck. We measure CNS capacity by running each experiment for 100 s, waiting up to an additional 100 s to get replies, and measuring the response rate. Dedicated Emulab machines show little variation in capacity measurements, so we perform just 5 runs for each and plot averages.

5.1.1 Replication of subspaces

This experiment seeks to compare the scalability of three systems, CNS, HyperDex, and GNS, and confirm that *replication* of subspaces does significantly increase capacity as predicted by our model.

The workload for this experiment differs from §5.1 only in that we use $m = 3$ attributes in this experiment, and search queries have 2 predicate attributes. The search query fraction ρ is 0.8. In CNS and HyperDex there is a single subspace of 3 attributes but CNS replicates that one subspace $\sqrt{n}$ times and assigns $\sqrt{n}$ nodes to each while HyperDex does not perform replication and assigns all nodes to the single subspace.

The results show that all the schemes give similar capacity for 1 and 4 nodes, but for 16 and 25 nodes CNS performs better than HyperDex and GNS. The reason is that a search in CNS goes to respectively to 1, 2, 3, 4 or 5 machines when the total number of machines is 1, 4, 9, 16 or 25. In HyperDex, a search goes to 1, 4, 7, 13 or 17 machine with 1, 4, 9, 16 and 25 total machines respectively. In GNS, a search goes to all nodes. The size of the result set for all schemes is an average of 2500.

5.1.2 No subspace replication

Next, we evaluate the scalability of CNS without replication for the default 20-attribute workload. The CNS uses its optimal region mapper that automatically determines the subspace replication factor and the number of subspaces K. In this experiment, the CNS happens to choose to not replicate subspaces, determines an optimal $K = 2$ for 1 and 2 nodes, and optimal $K = 4$ for 4, 8, 16, 24 and 32 nodes. For $\rho = 0.0$, the optimal value of K is 2 and for $\rho = 1.0$ the optimal value of K is 10, but as the CNS doesn't know the workload ρ a priori in this experiment, it picks K assuming the default value of $\rho = 0.5$. As the CNS indexing metadata is exponential in m/K, or the number of attributes per subspace, we limit it to 10.

Fig. 6 shows the context service capacity against the number of nodes for different values of the search query ratio ρ. Figure 6 shows that the capacity of the context service in different cases scale with the number of nodes. Knowing the demand workload, specifically the search query ratio ρ, can make a nontrivial difference in system capacity.

5.2 Trigger capacity scalability with nodes

In this experiment, we evaluate the performance benefit of triggers (§3.3.2) that notify queriers of stateful queries of changes to the result set against varying values of the query lifetime.

The workload is as in 5.1.2 with $\rho = 0.1$. Whenever a search query is issued, the CNS creates a trigger for it that last 30, 60, or 120 seconds (*query-Expiry* time in the figures) and notifies the querier of any updates performed to the queried attribute range during this period. For trigger durations of 30s and 60s, we run the experiments for 200s, and for a trigger duration of 120s, we run the experiments for 400s so that the CNS warms up with the triggers and reaches a steady state.

Fig. 7 shows that as the duration of the triggers increases the capacity of the context service decreases as more triggers are active simultaneously, so notifications get triggered. For 32 nodes, trigger durations of 30s, 60s, and 120s respectively give 5.5x, 8.0x, and 9.8x lower capacity compared to if we had the same workload but didn't have any of the overhead of maintaining the triggers.

Fig. 8 shows the average number of search queries that are notified by a trigger because of an update. "Added groups" displays the average number of triggered search queries that a guid was added to during an update, and likewise "Removed groups" shows the average number of triggered search queries that a guid was removed from during an update. These are consistent with our expectation (as per Little's Law [30]) that the number of active queries increases linearly with the refresh interval, which implies the same linear increase (all else being equal) for the number of affected queries.

5.3 Privacy capacity scalability with nodes

The goal of this experiment is to measure the overhead of privacy mechanisms. The workload for this experiment matches § 5.1.2 except we also cre-

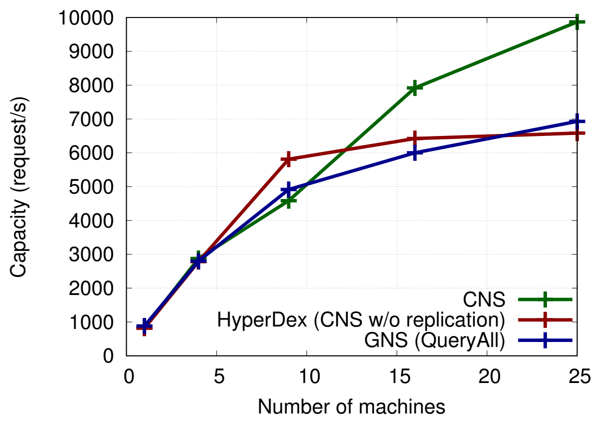


Figure 5: Capacity with 3 attributes, $\rho = 0.8$

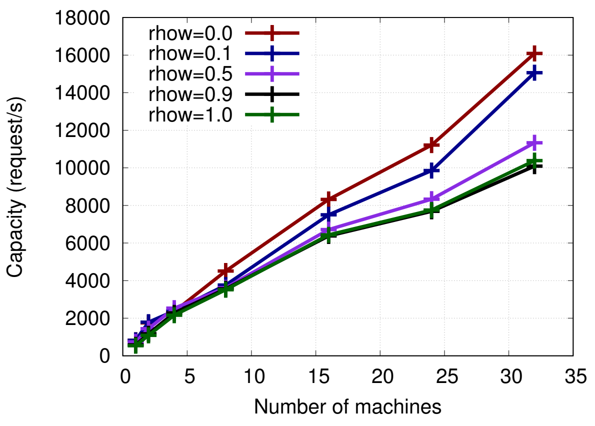


Figure 6: Capacity vs. #nodes for varying ρ

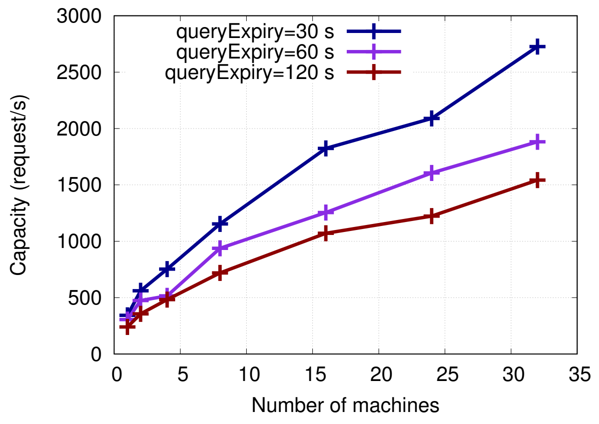


Figure 7: Capacity vs. #nodes with triggers

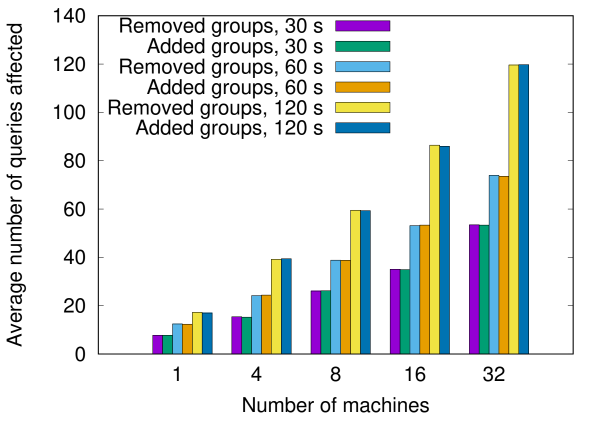


Figure 8: Affected search queries per update

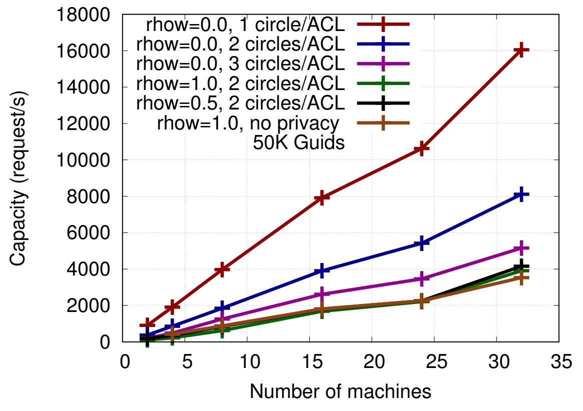


Figure 9: Privacy capacity variation with number of nodes.

ate ACLs for each attribute of each GUID. To create these ACLs for one GUID we randomly select 10 GUIDs and partition them into 5 pairs of GUIDs called circles. We then uniformly choose circles to assign to the ACL for each attribute in the one selected GUID. This is repeated until each attribute in each GUID in the system has its own ACL.

We then create pseudonyms for each GUID as described in § 4. In the experiment, CNS creates 5 pseudonyms per GUID. We evaluate the CNS capacity with privacy so we don't bottleneck our client by performing decryptions to map cryptonyms to their GUIDs in a search reply. We first run experiments where we issue only update operations ($\rho = 0$) with 1, 2, or 3 circles of GUIDs assigned to each ACL. We then assigned 2 circles of GUIDs to each ACL and issued only search operations ($\rho = 1$), or an equal mix of operations ($\rho = 0.5$).

With one circle of GUIDs assigned to each ACL we find that the update capacity is close and slightly smaller than the update capacity of CNS without privacy features as shown in Figure 6. This is what we expect as an update of a single attribute in a GUID only causes the update of 1 pseudonym. But a single pseudonym can be deleted from a node and inserted on a new node and inserting a pseudonym requires inserting the cryptonym in addition to attribute value pairs, which causes slightly lower capacity in this case. Similarly, with 2 circles of GUIDs per ACL or 3 circles per ACL the capacity is roughly $\frac{1}{2}$ or $\frac{1}{3}$ the 1 pair case since each update affects the respective number of pseudonyms for that GUID.

The search capacity with privacy is lower than the default case without privacy shown in Figure 6 since for each GUID it must also store all of the pseudonyms associated with that GUID. For example, if each GUID has 5 pseudonyms then the system with privacy would need to store 50k pseudonyms rather than just the 10k GUIDs stored in a system without privacy. For each search the context service using privacy must read from the mapping of pseudonyms-to-GUIDs stored in the database along with the pseudonyms themselves. In this experiment there are two circles of GUIDs per ACL so each pseudonym has two cryptonyms. Since in this experiment there are 5 pseudonyms per GUID, the capacity for 10,000 GUIDs with privacy is similar to the capacity for 50,000 GUIDs without privacy.

5.4 Application case studies

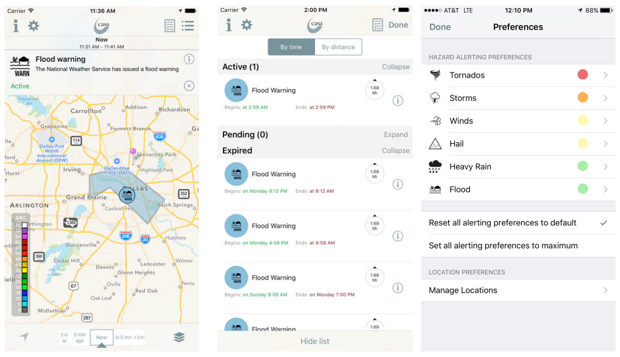
5.4.1 Weather case study setup

In this experiment, our goal is to show using a semi-realistic workload of weather and mobility traces that the context service is more scalable and thereby can support more context-based applications compared to the naive QueryAll scheme.

We simulate a weather notification application, which is a type of location context-based app, to perform the case study. A screenshot of a hazardous weather notification app is shown in Fig. 10. The weather notification application issues queries based on the weather data to notify users. In the case study, we compute the maximum number of such applications that the different schemes can support. Next we describe the real world weather and mobility data. Then we describe the experiment setup to evaluate the context service and the basic query all scheme of GNS.

We use National Weather Service (NWS) weather data[8] generated by Buffalo, NY radar from 28 Jan 2014 00:00:00 GMT to 30 Jan 2014 00:00:00 GMT. The rectangular area that we consider is given by two diagonally opposite points [41.8, -80.0] and [45.1, -74.5], where the elements of the tuple are latitude and longitude respectively. The total area is around 59342 Km^2. We pick the following area because the weather events generated by the radar were located in this area. We get 56 such weather events in our specified duration. Each weather event consists of the following; the issue and expiry time in GMT, the weather phenomenon type, the event significance and affected areas in the form of geoJSON polygons. For our case study we use the issue time, expiry time and geoJSON polygons to issue search queries. In our weather data, on average a weather event is active for 18.28 hours. We use the mobility data of 43 unique users from the same duration as of the weather data in the Buffalo area. From the mobility data we have trajectories of 43 users over the specified duration. A trajectory of a user is a time series of the user's location, in terms of latitude and longitude, over the duration. Next we describe how we use these traces to generate search and update workload for the case study.

For the search workload we use geoJSON polygons in the weather data to create search queries. We find the bounding rectangle for the arbitrary polygon described by a weather event's geoJSON and issue search queries using the corresponding bounding rectangle. We have 56 weather events and we assume that a single weather notification application will issue search queries for these events. Applications other than weather notification will also issue similar geoJSON based search queries, so we measure the maximum number of applications that different schemes support in our case study. Each application issues search queries based on all the 56 events. Our mobility log data has only 43 distinct trajectories, but for the udpate workload we require more than 43 users. We use the user trajectories from the mobility data, which we call log trajectories, to generate new trajectories, which we call synthetic trajectories, for additional users. For creating a synthetic trajectory for a user, we randomly pick a log trajectory and transform all its points using the following transformation. We transform the first point of a log trajectory to a random point in the case study area and then we calculate the distance and the azimuth angle from the log trajectory point to the transformed point. We use geodesy library to transform all other points in the log trajectory by the same distance and azimuth angle. The earth surface is curved so the geodesy library uses distance and azimuth angle to accurately transform points. We pick only those synthetic trajectories who have all the points within the case study area. By using the above procedure we create trajectories for all the users in our experiments. Generating synthetic trajectories using the described procedure preserves the mobility pattern and update rates of users in the mobility data. Next we describe how we run the workload in our case study experiment.


Carrier 11:36 AM
Now
11:31 AM - 11:41 AM
Flood warning
The National Weather Service has issued a flood warning
WARN
Active

Carrier 2:00 PM
Done
By time By distance
Active (1) Collapse
Flood Warning 1.68 Mi
Begins: at 2:59 AM Ends: at 2:59 PM
Pending (0) Expand
Expired Collapse
Flood Warning 1.68 Mi
Begins: on Monday 9:12 PM Ends: at 9:12 AM
Flood Warning 1.68 Mi
Begins: on Monday 4:58 PM Ends: at 4:58 AM
Flood Warning 1.68 Mi
Begins: on Sunday 8:09 AM Ends: on Monday 7:00 PM
Flood Warning 1.68
Hide list

●●●●○ AT&T LTE 12:10 PM 88%
Done Preferences
HAZARD ALERTING PREFERENCES
Tornados 🔴
Storms 🟠
Winds 🟡
Hail 🟡
Heavy Rain 🟢
Flood 🟢
Reset all alerting preferences to default ✓
Set all alerting preferences to maximum
LOCATION PREFERENCES
Manage Locations

Figure 10: ANON hazardous weather app screenshots

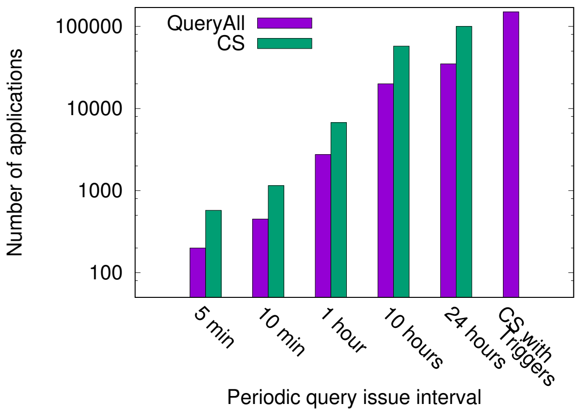


Figure 11: Weather case study with 100K GUIDs

In our experiment, we contract time from 2 days, which we call the real time, in the traces to 15 minutes, which we call the experiment time. Precisely, 1 s of the experiment time is 192 s of the real time. We create synthetic trajectories for 100K users. We create different number of applications, which issue search queries based on the weather data, based on the experiment setting. The experiment runs a clock and issues those updates and searches whose timestamps are equal or less than the corresponding real world time for the current experiment time. A search query is active from its issue time to its expiry time in the log. When a search query is active it is also periodically refreshed based on the periodic query issue interval setting of an experiment. We vary the number of applications, where each application issues and periodically refreshes search queries, and measure the maximum number of applications supported by different schemes for different periodic query issue intervals. In the experiment, there are N=25 nodes and the context service is initialized with 2 attributes, latitude and longitude, and there is just 1 distinct subspace and $\sqrt{N}$ replicas of that subspace. We use 5 clients to issue update and search requests. We run each experiment for 15 minutes and wait for another 200 s after 15 minutes for replies before timing out.

Fig. 11 shows the maximum number of applications for QueryAll and CNS without triggers and CNS with triggers. In CNS without triggers a search query from an application goes to 1 node and an update goes to 5 nodes. While in the QueryAll scheme a search query goes to all the 25 nodes and an update goes to 1 node. CNS without triggers scheme gives 2.5-3 times more applications than the QueryAll scheme across different periodic query issue intervals. CNS with triggers gives more applications that CNS without triggers and QueryAll because the search queries do not need to be repeated at periodic query issue intervals when the CNS notifies through triggers.

6. CONCLUSIONS

We began with an ambitious goal, namely, enabling contextual communication as a general-purpose communication primitive similar in spirit to TCP/IP sockets. What we have actually accomplished is but a first step towards that goal. Our primary contribution is the design, implementation, and evaluation of the CNS that achieves provably scalability of $\Theta(\sqrt{n})$, and our hardness results help shed light on why prior approaches to this seemingly well studied problem achieved suboptimal scalability. Furthermore, the accompanying GNS-CNS federation architecture and our proposed privacy protocol allow users to rest assured that the CNS or GNS service provider can not infer sensitive user information, similar in spirit to how ISPs today can not infer sensitive user bits carried over SSL. Our case study based on a semi-realistic hazardous weather warning workload and smartphone app suggests that the CNS can scale to a large number of apps, potentially eliminating redundant development effort and leveraging economies of scale.

7. REFERENCES

[1]

[2] Amazon web services, https://aws.amazon.com/.

[3] Gns github repository, https://github.com/mobilityfirst/gns.

[4] Google cloud platform, https://cloud.google.com/.

[5] Microsoft azure, https://azure.microsoft.com/.

[6] Named Data Networking. http://www.named-data.net/.

[7] NOAA Weather Alerts: Severe Push Notifications and Warnings.

[8] Nws weather data source, https://mesonet.agron.iastate.edu/vtec/search.php.

[9] ping4alerts app, http://www.ping4.com/.

[10] Aguilera, M. K., Golab, W., and Shah, M. A. A practical scalable distributed b-tree. *Proc. VLDB Endow. 1*, 1 (Aug. 2008), 598–609.

[11] Bharambe, A. R., Agrawal, M., and Seshan, S. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2004), SIGCOMM '04, ACM, pp. 353–366.

[12] Boldyreva, A., Chenette, N., Lee, Y., and Oneill, A. Order-preserving symmetric encryption. In *Advances in Cryptology-EUROCRYPT 2009*. Springer, 2009, pp. 224–241.

[13] Boldyreva, A., Chenette, N., and ONeill, A. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Advances in Cryptology–CRYPTO 2011*. Springer, 2011, pp. 578–595.

[14] CASA. `http://www.casa.umass.edu/main/research/urbantestbed/`.

[15] CASA Alerts. `https://itunes.apple.com/us/app/casa-alerts/id1094271600`.

[16] Cerf, V. G., and Icahn, R. E. A protocol for packet network intercommunication. *SIGCOMM Comput. Commun. Rev. 35*, 2 (Apr. 2005), 71–82.

[17] Codered mobile alert app: onnecting the world with real-time, location specific alerts to save lives. `http://ecnetwork.com/codered-mobile-alert-app/`.

[18] Dwork, C. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation* (2008), Springer, pp. 1–19.

[19] Escriva, R., Wong, B., and Sirer, E. G. Hyperdex: A distributed, searchable key-value store. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (New York, NY, USA, 2012), SIGCOMM '12, ACM, pp. 25–36.

[20] NSF Future Internet Architecture Project. `http://www.nets-fia.net/`.

[21] Fotiou, N., Nikander, P., Trossen, D., and Polyzos, G. C. Developing Information Networking Further: From PSIRP to PURSUIT. In *Broadband Communications, Networks, and Systems* (2012), Springer, pp. 1–13.

[22] Gao, Z., Venkataramani, A., Kurose, J. F., and Heimlicher, S. Towards a quantitative comparison of location-independent network architectures. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 259–270.

[23] Gentry, C. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[24] MobilityFirst/GNS project on github. `https://github.com/MobilityFirst/GNS`.

[25] Gritter, M., and Cheriton, D. R. An Architecture for Content Routing Support in the Internet. In *USENIX USITS* (2001).

[26] Hore, B., Mehrotra, S., Canim, M., and Kantarcioglu, M. Secure multidimensional range queries over outsourced data. *The VLDB JournalThe International Journal on Very Large Data Bases 21*, 3 (2012), 333–358.

[27] Hore, B., Mehrotra, S., and Tsudik, G. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30* (2004), VLDB Endowment, pp. 720–731.

[28] Kairam, S., Brzozowski, M. J., Huffaker, D., and Chi, E. H. Talking in circles: Selective sharing in google+. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 12)* (New York, NY, 2012), pp. 1065–1074.

[29] Lampson, B. W. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 1986), PODC '86, ACM, pp. 1–10.

[30] Leon-Garcia, A. *Probability, Statistics, and Random Processes for Electrical Engineering*, 4th edition ed. Prentice Hall, 1 2015.

[31] Nikander, P., Gurtov, A., and Henderson, T. Host identity protocol (hip): Connectivity, mobility, multi-homing, security, and privacy over ipv4 and ipv6 networks. *Communications Surveys Tutorials, IEEE 12*, 2 (2010), 186–204.

[32] Popa, R. A., Li, F. H., and Zeldovich, N. An ideal-security protocol for order-preserving encoding. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 463–477.

[33] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2011), SOSP '11, ACM, pp. 85–100.

[34] Saltzer, J. On the Naming and Binding of Network Destinations, 1993.

[35] Sharma, A., Tie, X., Uppal, H., Venkataramani, A., Westbrook, D., and Yadav, A. A Global Name Service for a Highly Mobile Internet. In *ACM SIGCOMM* (2014).

[36] Venkataramani, A., Kurose, J., Raychaudhuri, D., Nagaraja, K., Mao, M., and Banerjee, S. MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture. *ACM SIGCOMM Computer Comm. Review (CCR)* (2014).

[37] Zhang, L., Estrin, D., Burke, J., Jacobson, V., Thornton, J. D., Smetters, D. K., Zhang, B., Tsudik, G., Massey, D., Papadopoulos, C., et al. Named Data Networking (NDN) Project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).

Appendix

7.1 Proof of Lemma 3.1

Proof. The proof of this claim only needs a workload with a single attribute. Consider a fixed value space partitioning (defined in §3.1.1) that splits the value range of real length R into smaller, contiguous partitions of uniform size R/n assigned respectively to the n machines. Let ρ denote the (fixed) fraction of search queries. Partitioning without replication implies that each update of the attribute's value touches at most two machines, one corresponding to the partition of the old value and another to the new partition (if different). Consider a search workload querying for a random sub-range of a constant length C. It is straightforward to verify that the expected number of machines as well as the complexity of a search query is $1 + C/(R/n) = 1 + Cn/R$. The expected amount of work per query is at least $\rho(1 + Cn/R) + (1 - \rho)$, where the two outer additive terms correspond respectively to search and update complexity). This work per query increases as $O(n)$, so even if the load across queries were perfectly balanced across all n machines, the system capacity does not scale.

Any non-uniform partitioning strategy can do no better asymptotically. Consider a non-uniform partitioning of the value space into ranges of size $C_1, \ldots, C_n$. It is straightforward to show that the expected number of machines involved in a search query is at least $1+Cn/R$, where C is the length of the longest range, completing the proof as above. $\square$

7.2 Proof of Theorem 3.2

Proof. We consider two cases: one where the partitioned and replicated placement is fixed, and the other where it can be changed dynamically.

Case 1 (fixed placement): Consider a workload consisting of just a single attribute with updates uniformly distributed over the global value range of length R and searches for a uniformly random interval of fixed length C. Consider any fixed value space partitioning and replication strategy that maps mutually exclusive and exhaustive intervals $r_1, \ldots, r_p$ on to the set of n machines in a possibly one-to-many manner (because of replication). Note that we can make the mutual exclusion assumption without loss of generality to encompass any partitioning and replication strategy including even strategies that map every point in the global value range (containing uncountably many real points) to one of the 2^n subsets of the set of all n machines. However, for ease of formal exposition, we assume that p is finite.

Let $map(r_i)$ denote the subset of n machines to which the range r_i is mapped. The expected update cost per query is at least $(1-\rho)\sum_{i=1}^{p} \frac{|r_i|}{R}|map(r_i)|$ as the probability of the query being an update is $1-\rho$ and the probability that the updated value lies in r_i is r_i/R; note that the lower bound here is not tight as it is ignoring the work required to delete the record being updated from machines in $map(r_j)$ where r_j is the range wherein the old value lies, but the lower bound suffices for this proof.

Definition Let *rmax* denote the longest interval $[a, b)$ in R such that there exists a machine s such that each point in $[a, b)$ is contained in some $r_i, 1 \leq i \leq p$ that is mapped to s, i.e., $s \in map(r_i)$.

The expected search cost per query is at least $\rho(\lceil \frac{C}{rmax} \rceil)$ as each search query over an interval of length C must, by the definition above, induce a search operation on at least one additional machine for every MEE sub-interval of length *rmax*. Thus, the expected work per query is

$$\geq (1-\rho)\sum_{i=1}^{p} \frac{|r_i|}{R}|map(r_i)| + \rho(\lceil \frac{C}{|rmax|} \rceil) \qquad (1)$$

$$\geq (1-\rho)\frac{|rmax|}{R} + \frac{\rho C}{|rmax|} \qquad (2)$$

Let q denote the overall query rate, i.e., the search and update rates are respectively ρq and $(1-\rho)q$. We claim that there exists at least one machine that incurs an update load of at least $\frac{q(1-\rho)}{p}$. This is because $|rmax| \geq \max\{|r_1|, \ldots, |r_p|\} \geq R/p$, and by definition of *rmax*, there exists a machine s such that an update to any value in *rmax* induces an update operation on s.

We can place an asymptotic upper bound on $|rmax|$ as follows. There exists at least one machine s that incurs an update load of $q(1-\rho)\frac{|rmax|}{R}$. In order to achieve a scalability of $c(n)$, i.e., for the sustainable query load q to increase as $c(n)$, the load $c(n)(1-\rho)\frac{|rmax|}{R}$ on s must be $O(1)$ as it is bounded by its (fixed) capacity, so

$$rmax = O(\frac{1}{c(n)}) \qquad (3)$$

We can also place an asymptotic lower bound on *rmax* as follows. The expected amount of work done by a search query is at least $\frac{\rho C}{rmax}$ (by the reasoning for and just above equation 1). In order to achieve a scalability of $c(n)$, this work must increase slower than the total amount $\Theta(n)$ of resources available, i.e.,

$$\frac{c(n)\rho C}{rmax} = O(n) \qquad (4)$$

implying that

$$rmax = \Omega(\frac{c(n)}{n}) \qquad (5)$$

The only growth function satisfying both equations 3 and 5 and asymptotically minimizing equation 2 is

$$rmax = \Theta(\frac{1}{\sqrt(n)}) \qquad (6)$$

and for $rmax = \Theta(\sqrt(n)$, the expected work per (search or update) query grows as $\Omega(\sqrt{n})$, so $c(n) = O(\sqrt{n})$.

Case 2 (dynamically changing placement): We prove that any dynamic strategy for subspace partitioning and replication can not outperform the best static strategy with respect to scalability. Let $t_0, t_1, \ldots$ demarcate epoch boundaries such that in any epoch $[t_i, t_{i+1})$, the subspace placement—i.e., the mapping of subspaces to subsets of machines, the partitioning of each attribute's value range, and the $map(.)$ function to map regions to machines within a subspace—remains fixed. Within each epoch, one can use the reasoning as in the static case above to argue that the expected work per query grows as $\Omega(\sqrt(n))$. Changing the placement at epoch boundaries only increases the expected work per query when amortized across epochs. $\square$