PRANK: Population Record Aggregation Network

6.1800 Design Project Final Report

Liam Kronman, Sarah Zhang, Sophie Zhang
lkronman@mit.edu, sazhang@mit.edu, sophiez@mit.edu

Recitation Instructor: *Manya Ghobadi*
TA: *Jay Lang*
WRAP Instructor: *Laura McKee*

May 9, 2023

TABLE OF CONTENTS

1 Introduction

The fictional country of Fictlandia needs to modernize its census system fast! Currently, Fictlandians must fill out almost identical information for three separate census systems at the municipal, state, and national levels, and the result has been lower and lower turnouts for each. Consequently, past and current censuses do not accurately reflect the distribution and demographics of citizens across Fictlandia, impeding governments and public entities from most effectively serving their people.

Census information is critical to the decision-making process for Fictlandian governments: the national and state levels use census records to allocate representation in the national legislature and define legislative districts, while the municipal government uses voter identification information to run elections at all levels of government. At a local level, Election Boards use census data to maintain accurate voting rolls, while School Boards rely on the census to track, assign, and support children who should be enrolled in schools each year.

Luckily, Fictlandia will find the perfect solution to its growing census problem in ***PRANK***, **P**opulation **R**ecord **A**ggregation **N**etwor**k**, a unified system for aggregating, managing, and distributing multi-layered census data.

The design of PRANK prioritizes *privacy* and *robustness*. Privacy is important in ensuring protection of personal identifying data against malicious actors who might engage in identity theft, stalking, or discrimination. The confidentiality of census records prevents misuse and protects the well-being of Fictlandians. Data robustness ensures census data is correct and comprehensive, thereby useful to government authorities and researchers. Our system emphasizes privacy by filtering and encrypting sensitive data to only be accessible by permitted end users, as well as by applying differential privacy. PRANK guarantees data robustness through data duplication and preventing delivery of inconsistent datasets.

In Section 2, we provide an overview of the system design, followed by a more detailed discussion of implementation in Section 3. In Section 4, we evaluate the impact of our system in various use cases and also consider its limitations.

2 System Overview

In this section, we provide a brief overview of the functionality of PRANK, which is modularized into several components and diagrammed in Figure 1. PRANK's features are designed with two categories of users in mind: (1) Fictlandian census participants and (2) organizations and individuals accessing past census information.

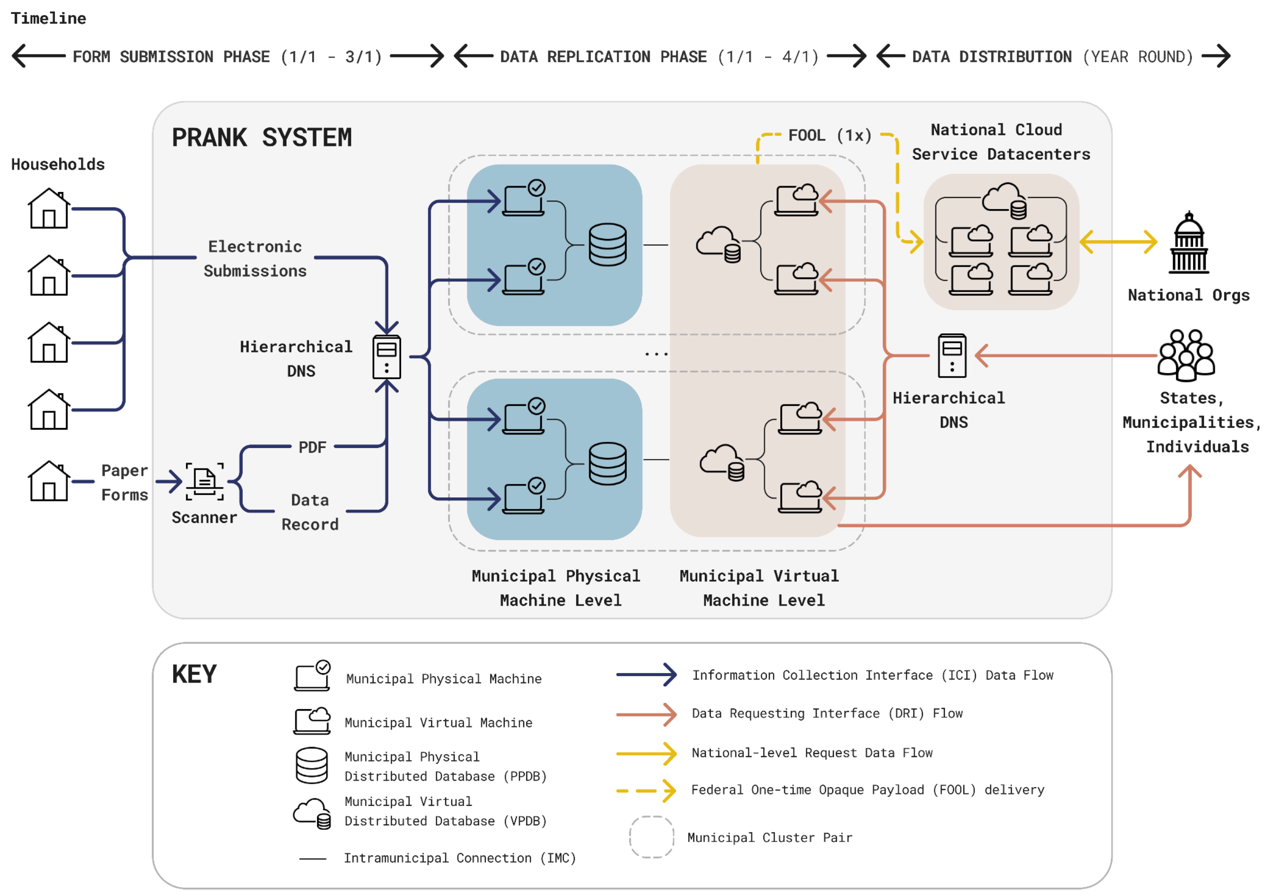
Timeline
FORM SUBMISSION PHASE (1/1 - 3/1)
DATA REPLICATION PHASE (1/1 - 4/1)
DATA DISTRIBUTION (YEAR ROUND)
PRANK SYSTEM
Households
Electronic Submissions
Hierarchical DNS
Paper Forms
Scanner
PDF
Data Record
FOOL (1x)
National Cloud Service Datacenters
National Orgs
Hierarchical DNS
States, Municipalities, Individuals
Municipal Physical Machine Level
Municipal Virtual Machine Level
KEY
Municipal Physical Machine
Municipal Virtual Machine
Municipal Physical Distributed Database (PPDB)
Municipal Virtual Distributed Database (VPDB)
Intramunicipal Connection (IMC)
Information Collection Interface (ICI) Data Flow
Data Requesting Interface (DRI) Flow
National-level Request Data Flow
Federal One-time Opaque Payload (FOOL) delivery
Municipal Cluster Pair

Fig. 1: Diagram of PRANK system illustrating the user and system interactions as well as the connections between machines within the system. We highlight the distinction between standalone physical clusters (in blue) and the connected municipal virtual machines on the national cloud service (both in beige).

Access and distribution APIs. There are two APIs available to end users: the Information Collection Interface (ICI) for Fictlandian form submitters, hosted on each municipal physical machine, and Data Requesting Interface (DRI) for organizations requesting state, municipal, or school information, hosted on each municipal virtual machine. Both have an associated DNS nameserver that directs requests to an available municipal machine. We manage interactions with the system through APIs to introduce *encapsulation* of the system and standardize the type of requests that must be handled. ICI is an endpoint hosted on municipal physical machines that accepts online form submissions (available from January 1st to March 1st), while DRI is an endpoint hosted on municipal virtual machines that handles requests for information (updated every April 1st).

Data storage of census information lives on duplicated Postgres databases in the virtual and physical machines of a municipality, known as the Physical PDB (PPDB) and Virtual PDB (VPDB), respectively. This coupled database structure uses replicated state machines, further explained in Section 3.1. Since the information needed at national and state levels are strict subsets of the municipal records, we store just the municipal records on both databases.

Another novel feature of PRANK is a process called the Federal One-Time Opaque Payload (FOOL). FOOL is a synchronous delivery of all national records from every municipality to the national cloud filesystem after the national census occurs, once every 10 years on April 1st. FOOL gives access to the full set of national records to the national government so they can perform lookups and calculate statistics on their own computational resources. This not only improves user convenience and system performance but also protects data consistency and privacy. Sections 3.4.1 and 4.1.2 further describe and justify the robust methodology of FOOL.

We employ the Transmission Control Protocol (TCP) at the transport layer for all communications. TCP provides reliable, ordered delivery of data in addition to error-checking mechanisms that prevent information corruption during transmission, thus providing data robustness. We also apply chunking during data transmissions while building our replicated databases during the data preparation phase and for FOOL in order to provide more efficient recovery from failure, avoid resending large quantities of data, and prevent the end user from potentially receiving multiple copies of data.

3 System Description

3.1 Database Structure

Each municipality has a pair of associated Postgres databases that maintain the same information: one hosted on the Municipal Physical Machine Cluster (MPMC) called PRANK Physical Database (PPDB) and the other hosted on the Municipal Virtual Machine Cluster (MVMC) called PRANK Virtual Database (VPDB). Consistency between the PPDB and VPDB for each municipality is maintained by following a replicated state machine model, outlined in Section 3.2.3.

Both PPDB and VPDB are made up of three tables: `official_census_records`, `ongoing_census_records`, and `permissioned_keys`.

The `official_census_records` table contains the latest complete census for public use and the `ongoing_census_records` table is used internally for the storage of census form submissions before they have been fully collected and aggregated into the `official_census_records` table.

ongoing_census_records is merged with official_census_records (and subsequently cleared) only after PRANK has completed processing the new census (at latest April 1st). We do this to guarantee data robustness and minimize exposure of personally identifiable information at the cost of potentially making data available to users earlier.

Each row of both ongoing_census_records and official_census_records contains the following fields: fictlandian_id, submission_year, submission_id, first_name, last_name, birthdate, gender, race, address, household_relations, car_ownership, income_supp_eligibility, healthcare_eligibility, previous_address, voter_registration, party_affiliation, dog_ownership, transportation_usage, primary_language_spoken, primary_language_written, secondary_language_spoken, secondary_language_written, pdf, chunk_id, and is_replicated.

We designate fictlandian_id to be the primary key for both tables as it is the only guaranteed unique identifier out of all the columns[1].

permissioned_keys is a table for checking the permissions of requesting users to access sensitive records. It contains two fields: public_key and permission_level. permissioned_keys is queried upon any request to DRI to validate that the requesting entity's public key is registered with the level of access corresponding to the information being requested. The maintenance of this table is the responsibility of an appointed superuser for each municipality (further explained in Section 3.4.3).

3.2 Data Collection

PRANK processes all census form submissions on MPMCs. The online form interface and scanner-computer-pair communicate with our system through the Information Collection Interface (ICI) API. ICI accepts requests between January 1st and March 1st and returns an exception message outside of that time. ICI supports the following requests:

[1] Multiple people in a household share the same submission_id and name is possibly overloaded.

Function	Behavior
`SUBMIT_ELECTRONIC_FORM(form_data)`	Queries DNS server, which identifies an available MPMC machine to direct the electronic form submission data to. Creates a municipal data record out of the given form data and saves it in the physical machine's distributed database.
`SAVE_RECORD_AND_PDF(data_record, pdf)`	Queries DNS server, which identifies an available MPMC machine to direct the data record and PDF to. Saves the PDF on its own filesystem and inserts the provided data record in the physical machine's distributed database.

Table 1: ICI Functions

For either of these requests, ICI will return a "Success" message once the form submission has reached a computer, or a "Failure" message if no computer is able to be reached (i.e. network failure).

3.2.1 Processing Electronic Submissions and Paper Forms

We define the act of a form submission to start when a user hits submit on the form and end when their corresponding data record has been saved to the database. We assume that the web interface can cache progress in users' browsers and also blocks users from hitting "submit" until their form is complete, so PRANK neither supports progress-saving nor deals with receiving incomplete forms. When a user does hit "submit", the web client makes a `SUBMIT_ELECTRONIC_FORM` request to the ICI and an available municipal machine to process the form data is identified by a load-balancing DNS (details in section 3.2.2). Lastly, processing the submission consists of converting the form contents into a Postgres entry and saving it in the machine's PPDB.

Paper forms follow a similar routine. Once the paper form has been scanned, the computer attached to the scanner makes a `SAVE_RECORD_AND_PDF` request, which converts the raw data record to a Postgres entry and saves it to the PPDB of an available machine and saves the PDF into its filesystem using the `fictlandian_id` as the PDF name. If a family with size greater than 11 needs to submit a paper form, they should fill out another set of paper forms and contact their officials to merge their paper submissions. To avoid this clunky workaround that lowers accessibility for larger families, we suggest that paper forms be able to take an arbitrary amount of household members.

3.2.2 Load-Balancing via DNS

To access API endpoints for a municipality, one must navigate PRANK's hierarchical DNS system to find the IP of an available machine in that municipality. PRANK's DNS nameservers follow the following naming scheme: `mun0.state0.prank.gov`; where the top-level nameserver for PRANK would control `prank.gov`, state0's nameserver would control `state0.prank.gov`, and mun0's (a municipality in state0) nameserver would control `mun0.state0.prank.gov`. This naming scheme is adopted as a layer of abstraction for users to more easily access the correct endpoint for a municipality or state.

PRANK uses DNS to load-balance form submissions and data requests within a municipality by round-robin scheduling[1]. If a municipality machine is full (i.e. already processing 125 form submissions), then the DNS checks if the next machine in its records is available. If this fails, DNS returns the address of a random municipal physical machine to the Fictlandian form submitter, and will add the IP address of the submitter's machine into an in-memory queue on the assigned municipal machine. We chose to randomize the machine selection as opposed to spending time searching for the machine with the shortest queue to prioritize the average efficiency and immediate-term performance of the system, at the cost of potentially making some households wait slightly longer to access a form. Once the municipal physical machine becomes available to process another user, it will send an HTTP notification to the form-submitting client to enter an active form-submitting session.

3.2.3 Making Use of Downtime: Populating the Database

Since we only expect users to be filling out and submitting census forms between 8am and 10pm, there are 10 hours of guaranteed downtime (10pm-8am) each night where our municipal computers are free from processing form submissions. Additionally, even within the 8am-10pm timeframe, there may be downtime where few people are submitting forms. Whenever this downtime occurs, PRANK can work on other tasks, namely, supporting other MPMC machines and populating the VPDB by sending new data to the municipal virtual machine cluster (MVMC) via Intramunicipal Communication (IMC).

Intramunicipal Communication (IMC) bridges the MPMC and MVMC for municipalities. It is established to share information between their separate Postgres instances and consists of daemonized processes on physical and virtual machines listening for HTTP communications. While municipalities are still accepting submissions, every data record will be replicated from MPMC to MVMC using IMC. This is a choice to introduce *modularity* and *failure resistance*: as future requests for state- and municipal-level data will be made to virtual machines (ideally) exclusively, as they have higher bandwidth and are unaffected by municipal-level machine failure. All transmissions of information between virtual machines and physical machines will be encrypted using public keys of the receiving cluster (each machine cluster has its own unique public/private key pair), maintaining *privacy* across all communications.

IMC relies on the replicated state machine model. During the data collection phase, the physical municipal machine acts as the primary coordinator and handles all incoming requests to the municipal machine cluster. The virtual municipal machine serves as a backup coordinator and retains a copy of the state machine synchronized with the primary. If the primary coordinator fails, the backup takes over as the new primary coordinator and begins handling form submission requests.

There are three different states a physical municipal computer can be in during the data collection period of January 1st to March 1st. Within each of these states, the computer has an associated task to perform. We assign priorities to each of these tasks, and each machine maintains a priority queue to track what task to perform next. These states and tasks are enumerated (in order of execution priority) and described below:

Data Collection State Machine (DCSM):
State 1. Processing form submissions as described in 3.2.1.
State 2. Populating PDBs: whenever there are available resources, we send the saved municipal records to the VM paired with the physical machine via IMC. The VM then saves this municipal data record copy in the VPDB. (This will be the primary action on the computers between 10pm-8am.)
State 3. Serving data access requests: see 3.4 for details. Note that this action only returns data from a past existing and complete census in the PDB.

This methodology guarantees the most efficient use of available resources while prioritizing service towards households, who are the most critical users during this period.

3.3 Data Preparation

After March 1st, the ICI will close and the municipality machines focus on finishing preparing the data records and saving them in the VPDB (DCSM State 2). Upon completion, PPDB and VPDB are direct duplicates of each other, so that the system is robust to machine failures. In events of failure, network or machine-level, the retention of copies between virtual and physical machines ensures that virtual machines can request missing records from physical servers when needed.

To send records between MVMC and MPMC correctly and completely (in the spirit of data robustness), we combine the guarantees of TCP with an application-layer chunking protocol. As mentioned in section 3.1, the `ongoing_census_records` table in both PPDB and VPDB includes the fields `chunk_id` and `is_replicated`, which are of type number and boolean, respectively. Whenever a record is added to `ongoing_census_records` via ICI, `is_replicated` is initialized to false and `chunk_id` is set based on a selected chunking algorithm.

When a municipal machine is sending records over IMC to another cluster (DCSM State 2), each chunk will be sent separately, sequentially (a chunk being the aggregation of all records that share a `chunk_id`), with their `chunk_id` and the chunk's size added, unencrypted via the header. Each chunk's contents are encrypted via the methodology outlined in section 3.4.3.

Once the receiving cluster has acknowledged all the packets that make up the chunk (by validating the total amount of data received matches the expected chunk size) and written all the new information to its database, it will send an ACK back to the sender, specifying that the chunk was delivered correctly and completely. When the sender receives this ACK, it updates the `is_replicated` field to true for each row in `ongoing_census_records` that has that `chunk_id`. In future replication procedures, these records will be skipped when selecting chunks to replicate to other clusters.

Adding this chunking protocol greatly improves performance and fault tolerance, enabling better resumption rather than having to resend all the records from the beginning. It also simplifies consistency: in the case of failure when sharing over IMC, there is less inconsistent data between the two clusters since at most one chunk's worth of data will need to be garbage-collected by the receiver after recovery.

3.4 Access and Distribution

The two key features that make up our access and distribution system are an API called DRI (Distribution Request Interface) and a synchronized data delivery called FOOL (Federal One-time Opaque Payload). DRI is meant for accessing municipal, state, and school-level census information while FOOL distributes national information. DRI supports the following request types:

Function	**Behavior**
`GET_ALL_RECORDS(public_key, organization_level, year)`	If `organization_level` is *'municipal'* or *'school'*, we call `SELECT * FROM official_census_records;` on VPDB and filter out the fields not accessible to their organization, either forwarding the request to another machine in VPDB or to PPDB in the case of network failure or directly responding with records encrypted with that organization's public key (if they have the correct level of access) or a failure message. *'state'* requests are processed similarly, but are forwarded to other machines in the same state and aggregated as described in 3.4.2.
`GET_RECORDS(public_key, organization_level, year, target_records)`	Shares the same behavior as `GET_ALL_RECORDS` except passes through specific rows that they would like information for. Before response, a layer of differential privacy operations is performed.
`GET_FIELDS(public_key, organization_level, year, target_fields)`	Shares the same behavior as `GET_ALL_RECORDS` except passes through `target_fields` instead of `*` during the SQL query to VPDB. Additionally, before response, if this organization's level does not have permission to access particular fields, those fields are filtered out before the SQL query. Applies differential privacy.
`GET_STATISTIC(public_key, organization_level, year, target_fields, statistic)`	Shares same general behavior as `GET_FIELDS`, but instead of directly responding with records, the MVMC for a municipality performs a desired calculation on the records, which is encrypted using the key corresponding to organization level. Applies differential privacy before calculating statistics. The `years` field can either be a single year or a list of years. If a list of at least 2 years are provided, then time-series statistical analyses can be performed. Supports the following statistical operations: `count`, `average`, `median`, `mode`, `minimum`, `maximum`. Time-series analysis options include: `percent_change`, `lin_regression`, `correlation`.

Table 2: DRI Functions

By providing data on a request basis, potentially irrelevant or untimely data do not reach end users, evading the risk of jeopardizing data quality downstream. Servicing database requests strictly as needed prevents unnecessary bandwidth usage and improves efficiency. All DRI requests apply differential privacy to protect sensitive information about individual Fictlandians while preserving the accuracy of general statistics and is implemented through an established system such as FLEX[3] which is built to operate on top of existing SQL databases with negligible performance overhead.

3.4.1 One-Time National Records Delivery via FOOL

FOOL occurs on April 1st and consists, first, of the machines in all MVMCs reading all the national records from the VPDB into memory, and then sending the records to the national cloud filesystem over TCP. Since PRANK does not have control over the operations going on in the national cloud filesystem, FOOL is designed to send encrypted records all at once to maintain data robustness and prevent organizations from potentially accessing partial amounts of new census information. FOOL also improves efficiency and user convenience: by sending data to the national cloud, the federal government can access records and perform future computations immediately on its own compute resources.

The nature of FOOL creates a many-to-few data transmission relationship (every municipal VM, of which there are at least 3000[2], is delivering records to likely fewer national cloud filesystem machines). Since we deliver data over TCP, we acknowledge having every MVMC machine send data at once may lead to an incast problem, which is where the flood of incoming packets overwhelms the ethernet switch of the receiver's ability to buffer packets, creating a massive amount of congestion and a large drop in throughput. To mitigate this, we spread out the times that MVMC machines send their data. Specifically, each MVMC is assigned a specific export time which is randomly determined by sampling from a uniform distribution between 2:00 and 08:00[3] on April 1st. Then at that assigned time, every VM within the MVMC will send up to 100,000 records in parallel to the national cloud machines. This helps balance the load that the national cloud filesystem computers will experience on April 1st and should eliminate the risk of incast from FOOL delivery.

[2] The population of Fictlandia is 300 million and there is one municipal machine per 100,000 people, thus there are 300,000,000/100,000=3000 municipal VMs.

[3] We begin exports at 2:00 because as shown in Section 4.1.2, it takes just under 2 hours for a machine to read up to 100,000 records from the database. We choose to end exports six hours later since this averages out to one municipal exporting every 7 seconds and each export takes less than a tenth of a second, which leaves plenty of buffer time for the client to intake and process the delivered data. Ending at 8:00 also leaves enough to recover from various machine failures and still complete FOOL by the end of April 1st–further discussed in section 4.1.2.

To ensure data robustness when delivering FOOL, we employ a strategy nearly identical to the one used for IMC (see section 3.3), utilizing an application-layer chunking protocol. The main difference here is that instead of referencing `ongoing_census_records` tables, we read and update from the `official_census_records` tables of MVMCs. This strategy shares the same fault tolerance benefits as outlined for IMC, perhaps even more significantly, as file resumption is especially important in securing the timely delivery of such large data streams as FOOL's.

The load-balancing delivery scheduling scheme described prior means that data will be arriving at different times over the course of the day of April 1st. Thus, we recommend that if the National Cloud Filesystem already has an existing national records database from a prior census, then the national government should not update or overwrite the existing database until April 1st is over in order to guarantee data consistency. Instead, they should build up a new temporary database (analogous to PRANK's `ongoing_census_records` table) that will become populated while FOOL runs and if users need to retrieve records or analyze national data on April 1st, they should continue using data from their pre-existing database. Then, at 00:00 on April 2nd, after FOOL is guaranteed to be complete (in fact, we will later show in section 4.1.2 that FOOL will almost certainly be complete before 12:00), the government can shift to using the newest data (analogous to merging `ongoing_census_records` with `official_census_records`).

We recognize that FOOL hands over control of national records to the government and thus we place trust in the government that they will handle it responsibly and securely. We recommend that the national database also use differential privacy to protect individual Fictlandian data.

3.4.2 Handling Requests

Like ICI, requests to the DRI initially pass through DNS to determine which machine they should be directed to, and once a machine is found, there is a daemonized listener process that can handle that request.

We do not use chunking for DRI requests because on average, DRI requests handle a smaller amount of data and have lower stakes compared to the data duplication phase and FOOL procedure, and performance does not suffer significantly by retransmitting entire requests upon failure.

Handling municipal-level requests. When a user makes a request for municipal-level data, the load-balancing DNS assigns an available machine in the MVMC to gather records within the municipality and return the data to the user.

Handling state-level requests. When a state-level request is made by an end user, the request goes to an available machine in the user's MVMC. This MVMC machine will serve as the central controller for the operation and references the domains of every other municipality DRI endpoint within the state (stored in the `other_municipal_endpoints.txt` file on every municipal physical/virtual machine). The controller then makes RPCs to one machine in every MVMC to request the municipality's data. During this process, the central controller keeps track of which municipalities' data it has received and handles re-requesting data in case of timeouts and machine failures. Once the controller has received data from every single municipality in the state, it aggregates and sends the full dataset back to the end user, completing the initial request. This process is analogous to MapReduce[2], where workers are MVMC machines, the map function is preparing a table of state-level data, and the reduce function is aggregating all municipality's data into a single dataset. Figure 2 below illustrates this process.

The decision to have the machine which originally receives the request act as a controller and keep track of what data it has received or not is in accordance with our *data robustness* design priority, as we guarantee that the end user never receives incomplete data.

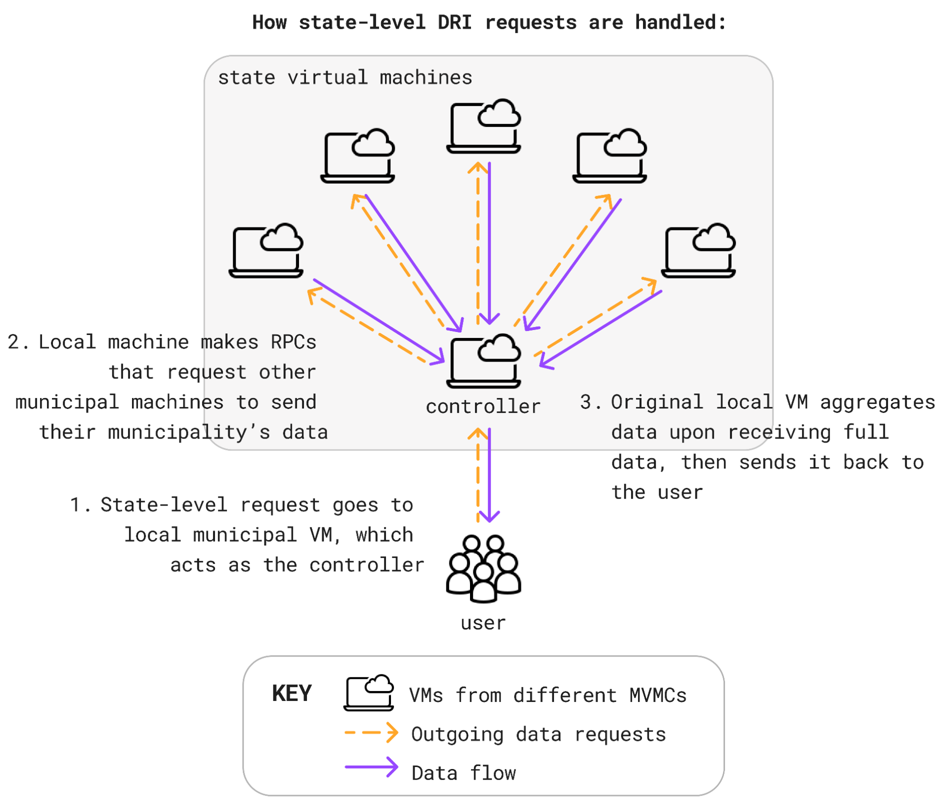
How state-level DRI requests are handled:
state virtual machines
2. Local machine makes RPCs that request other municipal machines to send their municipality's data
controller
3. Original local VM aggregates data upon receiving full data, then sends it back to the user
1. State-level request goes to local municipal VM, which acts as the controller
user
KEY
VMs from different MVMCs
Outgoing data requests
Data flow

Fig. 2: Diagram showing the process PRANK uses to handle state-level DRI requests.

3.4.3 Encryption

We choose to not store any data encrypted because we operate under the assumption that our database itself is secure and will never be directly exploited. Further, even if we wanted to encrypt data it's unclear which key to encrypt records with initially. This is because for every level of government, we encrypt records based on the public key of the requesting individual, if they are properly permissioned for that level. That individual will then be able to decrypt the response using their private key. DRI is a public API that can be requested for information at any time, but if someone requests information for a level for which they don't have permission, they will not have the key that enables them to decrypt it. This encryption layer is conducted before a response is done to all requested information except for data that is over 70 years old. This is checked by doing a query on the `submission_year` field of the to-be-returned records.

When users make requests, they will transmit their public key alongside their data request through the `public_key` field of each DRI request. This public key will be checked against the `permissioned_keys` table, returning a failure response to the client if that key is not found with the alleged level of access. This encryption methodology is secure because even if an actor seeking to exploit the system passes through the public key of an entity with access, they will not possess the private key necessary to decrypt the information.

To maintain each municipality's `permissioned_keys` tables, a superuser is appointed to each municipality who has permission to add, remove, and update records. Those that seek access to a certain level of census information submit a permission request (along with their public key) to government officials of that level, who contact the corresponding superusers to update their `permissioned_keys` tables.

A unique private key is distributed to each superuser (renewed once a year). When a superuser tries to log onto their `permissioned_keys` table instance, they will first receive an authentication secret encrypted using their public key which they must send back decrypted (using their private key) to proceed with their log-in.

4 Evaluation

4.1 Quantitative Evaluation

We define the following set of metrics–which include the database read rate, virtual machine network connection speed, and the size and communication overhead of TCP/IP/application-level metadata–to be the *standard configuration* that we reference and use for all evaluations in this section and the following (section 4.2).

Standard configuration: Database reads in both the cloud and physical machine database service take 70 ms per census record. Each packet will contain information about one encrypted record plus 43 bytes of metadata. Specifically, the metadata includes 20 bytes of TCP header information, 20 bytes of IP header information, and 3 bytes[4] to store application-level information (e.g. the chunk ID). The MVMC machines have a 10Gbps network connection. We assume a generous 5 percent increase in traffic due to retransmissions from TCP[4] and an additional 5 percent increase due to application-level protocol overhead.

4.1.1 ICI Requests

We first evaluate our system's capacity to service ICI requests in the average case by considering one of the 3,000 concurrently operating physical municipal machines, without loss of generality. Each computer is used by up to 100,000 people, with 80% of users making requests to the ICI. Assuming an average of 2.6 people in a household, approximately 30,770 households are submitting online

[4] We upper bound the number of chunks at the population of the largest municipality, 1.5 million, which takes 20 bits to express in binary and thus can be covered by 3 bytes.

forms to each onsite machine. In the average case, households will submit forms at a uniform rate over the two-month data collection period. The largest proportion of users (one third) plan to fill out forms between 7pm and 8pm each day. Thus we have on average 170 households intending to submit electronic forms during the hour interval of most activity.

Each household, on average, takes 31 minutes to fill out a form, and each machine can support up to 125 parallel users simultaneously submitting online forms. Our in-memory queue supports web clients attempting to request to a fully occupied machine by notifying the waiting client once the machine is available, ensuring all households may access and begin form submission in their intended hour every day.

We assume the per capita form submission data is approximately the same size as each municipal census record. Our system is constrained by 80,000 electronic form submission users to each machine, 2,400 bytes per municipal census record, and a physical machine network connection of 1000Mbps. The time required for census records sourced from electronic submissions to be sent to the municipal physical distributed database is approximately 1.54 second, inconsequential relative to the time needed for each household to fill out a form.

We additionally evaluate our system's capacity to service ICI requests in the worst case: all households attempt to submit online forms at the same time. Maximizing the compute bottleneck, this scenario results in at most 246 serialized form submission intervals, each on average taking 31 minutes. Assuming users will only fill out forms for 14 hours each day between 8am and 10pm, our system will take 9 days to process all census records. To ensure all census records can be submitted to the database even in the worst case, households must begin filling out forms at the latest 9 days before March 1. We acknowledge that in the worst case, we cannot service all form submissions should they occur on the final day of data collection. We therefore propose to establish a deadline for user form submission in advance of the true census data collection deadline.

4.1.2 FOOL Delivery

Our decision to deliver all national records directly to the national government via FOOL is a distinct and unique element of our system design. We decided that this method provides significant benefits over the alternative of servicing requests for national records on an as-needed basis through the DRI API for the following performance and security reasons.

FOOL improves *user convenience*: by delivering all records to the national cloud filesystem, the national government has full and immediate access to national records and can perform computations with its own resources. Further, FOOL is completely resistant to all machine or network failure cases which means the national government is guaranteed to receive the full dataset. Additionally, using FOOL improves our *system performance*: PRANK system resources are freed up from servicing national requests and can execute requests for municipal- and state-level requests faster on average.

FOOL also provides *data consistency*: while states and municipalities retake censuses every 5 years and 1 year, respectively, the national census is only done every 10 years, and the national government should not use data mixed across years. FOOL ensures that the national government uses data that is consistent by year and must separately request to the DRI, should they need more recent data. Lastly, FOOL improves *data privacy and security*: the data privacy of Fictlandians is protected by limiting the national government's access to machines that hold more personally-sensitive data in state and municipality records. This means any potential bad actors in the national government who want to examine more fine-grained data than they are allowed to do not have any way to do so with just the data we send the national government through FOOL.

We now analyze the amount of time that FOOL takes to run and show its resilience to different failure cases. Since municipality machines deliver data independently from each other, we can calculate the amount of time each machine takes to read their municipality’s records from the VPDB and then deliver the data.

Each machine in an MVMC will be responsible for at most 100,000 records, and each record takes 70 ms to read from the database, so reading records takes up to 70 ms * 100000 ≈ 1.94 hours per machine.

Each packet will have a size of 643 bytes, with 1,200 bytes for a single encrypted national record and 43 bytes of communication protocol metadata. Using the standard configuration metrics, we find that each MVMC machine takes up to 0.057 seconds to deliver its data:

FOOL delivery time per machine

= (size of 1 record in a packet) * (number of records per municipal VM) * (network speed) * (TCP and application level overhead)

= (1243 bytes / packet) * (1 packet / national record) * (100,000 records) * (1/10^10 bits/second) * (8 bits / byte) * (1.05^2 overhead)

≈ 0.11 seconds

Since MVMC machines send data in parallel, each MVMC will take slightly under 2 hours to complete its segment of FOOL:

FOOL total runtime per municipality

= [time to read data from database] + [transport time per machine]

= 7000 + 0.11 seconds

≈ 1.94 hours

Given that the data exports occur by latest 8:00, this means that barring mechanical failure, FOOL will be complete by 8:01. PRANK is unaffected by machine and network failure: machine crashes are resolved within 5 minutes, catastrophic machine failure takes on average 20 minutes to fix, and network failures take on average 3 hours to recover from. So, if a machine fails while aggregating data, there is always enough time to re-aggregate data from scratch, and if the network fails, it takes less than a second for a machine to resend data before the end of April 1st. Since catastrophic machine failure relies on sysadmin intervention, we recommend that sysadmin be on call for the day of April 1st to ensure this time-critical event goes smoothly.

4.1.3 DRI Requests

We use a system such as FLEX[3] to implement differential privacy, which can be overlaid on existing databases and supports SQL queries and further is found to have "negligible (0.03%) performance overhead"[3]. Thus, we discount the effect of performing differential privacy in our evaluations.

A wide range of specific use cases for DRI are outlined and evaluated in the next section. Thus, we limit evaluation within this section to the amount of time it takes to complete a request for (1) a single municipality record (which can be used as a basic unit for larger requests) and (2) all state records within a state (which is an upper bound for state-level requests).

First, a single encrypted municipal record is 2,400 bytes, and each record is contained in a packet with 43 additional bytes of transport and application level metadata per the standard configuration. We find that a single record takes just over .07 s to read and distribute:

Total time to read and distribute 1 municipal record

= (0.07 s / record) + [(1 packet / record) * (2443 bytes / packet) * (8 bits / byte) * (1 s / 10^10 bits) * (1.05^2 overhead scalar)]

= .07 + 1.1*10^-6 s / record

≈ .07 s / record

Next, we analyze the amount of time it takes to read and distribute all state records within a state. Since state-level DRI requests are parallelized across municipalities as described in section 3.4.2, this amounts to the summing the time it takes to delivery a single municipality's records to the controller (i.e. step 2 in Figure 2) and the time it takes to deliver all state records to the user (i.e. step 3 in Figure 2). We assume the amount of time it takes to make request calls to be negligible. Each municipality averages a population of 170,455 people, each state has an average of 7,500,000 people, and an encrypted state record is 900 bytes.

Average time to send state records from a municipality to controller
= (170,455 records) * [(0.07 s / record) + [(1 packet / record) * (1843 bytes / packet) * (8 bits / byte) * (1 s / 10^10 bits) * (1.05^2 overhead scalar)]]
≈ 11932 s ≈ 3.3 hr

Time to deliver all state records from entire state
= (7.5 million records) * (1 packet / record) * (1843 bytes / packet) * (8 bits / byte) * (1 s / 10^10 bits) * (1.05^2 overhead scalar)
≈ 12.2 s

Total time to complete request
≈ 11932 s + 12.2 s
≈ 3.3 hrs

Thus, a `get * from state` request takes, on average, 3.3 hours to complete. We note that the biggest bottleneck for all data transmissions is the time to read records from the database.

4.1.4 Storage Constraints

The average and largest PDF size is 2 MB and 6 MB, respectively, so on average each machine will store (2 MB per pdf) * (20,000 pdfs / machine) = 40 GB of pdfs per year, and at most 120 GB in the worst case. Further, the database must store at most an additional (800 bytes / municipal record) * (100,000 records) = 80 MB of data per year per municipal machine (for every municipal census).

Thus in the worst-case scenario, an SSD for a physical municipal machine (1 TB) would have enough storage for roughly 8 years before running out of storage, while a national cloud machine (four 2 TB SSDs) can last up to 66 years. So, our recommendation would be to add an extra 1 TB SSD to every municipal physical machine once every 8 years and another to each national cloud machine (conservatively estimating that we don't necessarily have full access to all 4 SSDs).

4.2 Special Use Cases

We now analyze how PRANK's design supports various special use cases.

4.2.1 Redistricting

A primary use case is redistricting of the national legislature. State governments access national census data via DRI requests to municipal virtual machines to facilitate redistricting, which occurs every five years. We assume state governments will engage with provided census data to ensure redistricting is fair, equitable, and representative for all constituents.

In the event of network failure, state governments are notified and must wait for repair. We prioritize data robustness at the potential cost of performance because having updated and complete data enables state legislatures to most effectively perform redistricting. Under normal operation, requests for data will be made exclusively to virtual machines to enforce modularity. In the event of machine-level failure, the retention of copies between virtual and physical machines permits requests to be made to backup physical servers. This data redundancy guarantees correctness and immediate access to backup records, obviating the need for frequent sysadmin intervention. Machine-level failures are handled similarly in all use cases that access data via DRI requests.

The number of legislative districts can be calculated only after all national-level census data by state have been read from the database and sent to each state government. Because state-level requests are made via parallel RPCs across the state (see section 3.4.2), the time of data transmission for statewide queries is bottlenecked only by the read rate from the database.

The maximum population per municipality is 1,500,000. Each encrypted national census record is 1200 bytes, and each record is contained in a packet with 43 additional bytes of transport and application-level metadata, detailed in section 4.1.2. Using the standard configuration metrics, we compute the total time required to read and transmit statewide census records.

Total time to deliver data for redistricting
> = (number of records) * (time to read record from database + time to transmit record over network)
> = (1,500,000 records) * [(0.07 s / record) + [(1 packet / record) * (1243 bytes / packet) * (8 bits / byte) * (1 s / 10^10 bits) * (1.05^2 overhead scalar)]]
> ≈ 105,000 s ≈ 29 hr

In the event virtual machines fail, states need to request data from their physical counterparts. Though the network connection speed in municipal physical machines is an order of magnitude slower than that of the MVMC, the database read time, identical across all machines, is several orders of magnitude greater than time required to transmit records across the network. The slowdown in network speed is therefore negligible.

In both average and failure cases, data retrieval and distribution to state governments can easily occur within the two-month interval from April 1st to June 1st between when census records are fully processed and when state governments must complete redistricting.

4.2.2 Municipal Elections

Our second primary use case is supporting elections at the municipal level by the local Election Board. Voter registrations that occurred after census data was collected take priority over census data and are assumed to be handled independently by the municipal Election Board.

This use case requires time-sensitive data access and consequently places a stronger emphasis on system performance. Because any voter can register up to the last-in person voting date, our system must ensure all data is up to date as of that time. In Fictlandia, anyone who is a citizen or legal permanent resident and is 16 or older is eligible to vote. We assume on average, Fictlandians in the 10th grade and younger are therefore ineligible to vote. Given that each grade, Pre-K through 12th, contains 1.2% of the population, the eligible voter population is upper bounded by 100% - (1.2% * (12 grades from Pre-K through 10th)) = 85.6% of the population, or a voting population per municipality of approximately 145,900.

The local Election Board requires census data at the municipal level. As calculated in section 4.1.3, each encrypted municipal record takes 0.07 s to read from the database and transmit over the network. Thus, the total time required to read and transmit voters' municipal census records is as follows.

Total time to deliver voter data for municipal elections
= (number of records) * (time to read and distribute one municipal record)
= (145,900 records) * (0.07 s / record)
≈ 10,200 s ≈ 2.83 hours

Our system guarantees the local Election Board can receive registered voter data within a couple hours, permitting voters to register up to the last in-person voting date. As in section 4.2.1, data redundancy between virtual and physical machines enforced by replicated state machines guarantees efficient access to backup records in the event of machine-level failure.

4.2.3 School Enrollment

Similar to how the Election Board interfaces with PRANK, the School Board collects student identification records via DRI requests to municipal virtual machines. We assume the School Board independently utilizes provided census data to assign students to school districts. This use case places moderate demand on efficient data access; the School Board requires all relevant census information early enough in the year such that assignments can be completed, including language accommodations for special case students, prior to the fall term.

Fictlandia's school system consists of 14 grades, each making up on average 1.2% of any population. Each municipality therefore has on average (14 grades * 1.2% population per grade * 170,455 people per municipality) ≈ 28,600 children who are enrolled each year. Each encrypted school record is 1320 bytes, and each record is contained in a packet described in the standard configuration.

Total time to deliver student data for school enrollment

= (number of records) * (time to read record from database + time to transmit record over network)

= (28,600 records) * [(0.07 s / record) + [(1 packet / record) * (1363 bytes / packet) * (8 bits / byte) * (1 s / 10^10 bits) * (1.05^2 overhead scalar)]]

≈ 2000 s ≈ 0.56 hr

All relevant information for student enrollment can be obtained in spring soon after data collection, granting the School Board several months to prepare student assignments for each school year.

4.2.4 External Researchers

In addition to government and public sectors, our system supports analyses performed by external researchers. To carry out studies across the whole country, government-authorized researchers may access census data from the national cloud file system. Requests for time-trend data not on the national scope are handled by making DRI requests for data across a specified time interval.

Researchers who wish to perform time-series analysis on census data are able to access data from any desired year since the PDB stores all census data over time and all DRI API functions include a `year` parameter. Further, our system supports some basic statistical analysis which researchers can use to get summary statistics or run regressions via the `DRI.GET_STATISTIC()` request.

5 Conclusion

Our proposed system PRANK successfully manages the collection, storage, maintenance, and communication of census data throughout Fictlandia while prioritizing data robustness and user privacy. PRANK achieves data robustness through data redundancy and by requiring information to be wholly submitted and delivered. PRANK additionally uses multi-tiered encryption to ensure only authorized entities may access requested census records. Rigorous qualitative evaluation shows that PRANK is able to complete operations within established timelines and maximizes its available resources to be as efficient as possible.

There are several improvements and points of uncertainty which should be resolved ahead of implementation. A systemic issue with the current census design is that the presence of unhoused peoples is not considered, and we urge Fictlandia to expand its census procedure to account for these populations. We also recommend conducting a survey to researchers in Fictlandia to ask what additional statistical operations they would like to see supported in the DRI API so that we can support researchers as best as possible.

In conclusion, after examining the features and benefits of PRANK, we hope that Fictlandia will find PRANK to be an ideal model for its new census system, ensuring efficient data management, robustness, and utmost privacy.

6 Author Contributions

The team collectively designed all major system components, the system diagram, and editing and revisions across the entire report. Liam focused on writing sections 1, 2, 3.1, 3.3, 3.4.3, and 5. Sarah focused on writing sections 3.2, 3.4.1, 3.4.2 of the system description, sections 4.1.2 and 4.1.3 of the evaluation, the acknowledgements, and references. Sophie focused on drafting the initial design diagram and evaluating specific use cases (4.1.1, 4.1.4, 4.2).

7 Acknowledgments

We would like to especially thank our Recitation Instructor Prof. Manya Ghobadi and TA Jay T. Lang for their detailed technical feedback and help throughout the design project, as well as our communication instructor Laura McKee for her guidance throughout the semester and feedback on organizational and visual aspects of our report. Additionally, we acknowledge lecturer Prof. Katrina LaCurts for her enthusiastic and engaging lectures that presented the key foundational knowledge on systems design and engineering that allowed us to tackle this project. Lastly, we thank all the Spring 2023 6.1800 course instructors for coordinating project materials and for their support on Piazza.

References

[1] Cloudflare. (n.d.). What is round-robin DNS? | Cloudflare. https://www.cloudflare.com/learning/dns/glossary/round-robin-dns/

[2] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6 (OSDI'04). USENIX Association, USA, 10.

[3] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. Proc. VLDB Endow. 11, 5 (January 2018), 526–539. https://doi.org/10.1145/3177732.3177733.

[4] Pentikousis, Kostas & Badr, Hussein & Andrade, Asha. (2011). A comparative study of aggregate TCP retransmission rates. International Journal of Computers and Applications. 32.10.2316/Journal.202.2010.4.202-2660