Distributed Web Scraping

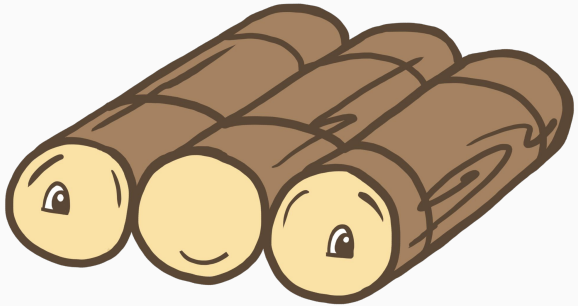
By Michail Roesli, Quinn Gieseke, and Blake Smith

Problem

Web scraping as a service

- Utilize idle phones
- Citizens can support organizations
- Improve performance of systems

Using RAFT and Phones together!



Related work

Folding@home

- distributed computing project for simulating protein dynamics
- helping scientists to better understand biology
- providing new opportunities for developing therapeutics

Distributed Web Scraping

- Distributed computing project for crawling websites and doing data gathering
- Helping organizations with performance for more efficient data acquisition

Functions

- Word occurrences (first attempt) - A/B testing
- Link frequency
- Images
- Word-relation maps
- Unique sentences
- Tweet maps
- Geo-tagged data for clustering

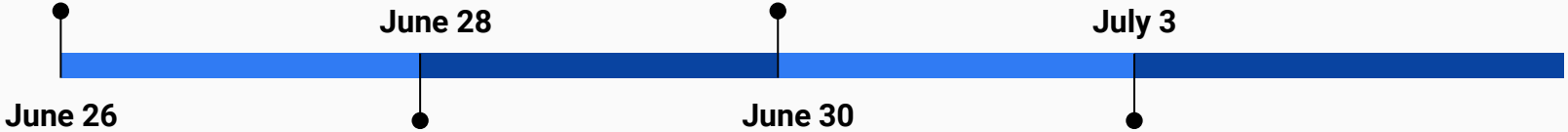
And all sorts of cool machine learning applications

RAFT Infrastructure

Run RAFT cluster and create GRPC calls structure. Brainstorm how we will communicate with phones and linux machines.

Fault Tolerance

Run RAFT cluster and withstand failures with android devices coming online and offline at will.



RAFT cluster communication

Run RAFT cluster that can communicate with phones and linux machines.

Plugins

Load and run provided plugins and send results to distributed database

Setting Up

- Created an Intellij project for the Linux client and an Android Studio project for the Android App
- Symlinked shared Kotlin (.kt) files and Proto (.proto) files into both projects
- Set up a basic gRPC server in Golang, and set up communication between the Android app and the Golang server

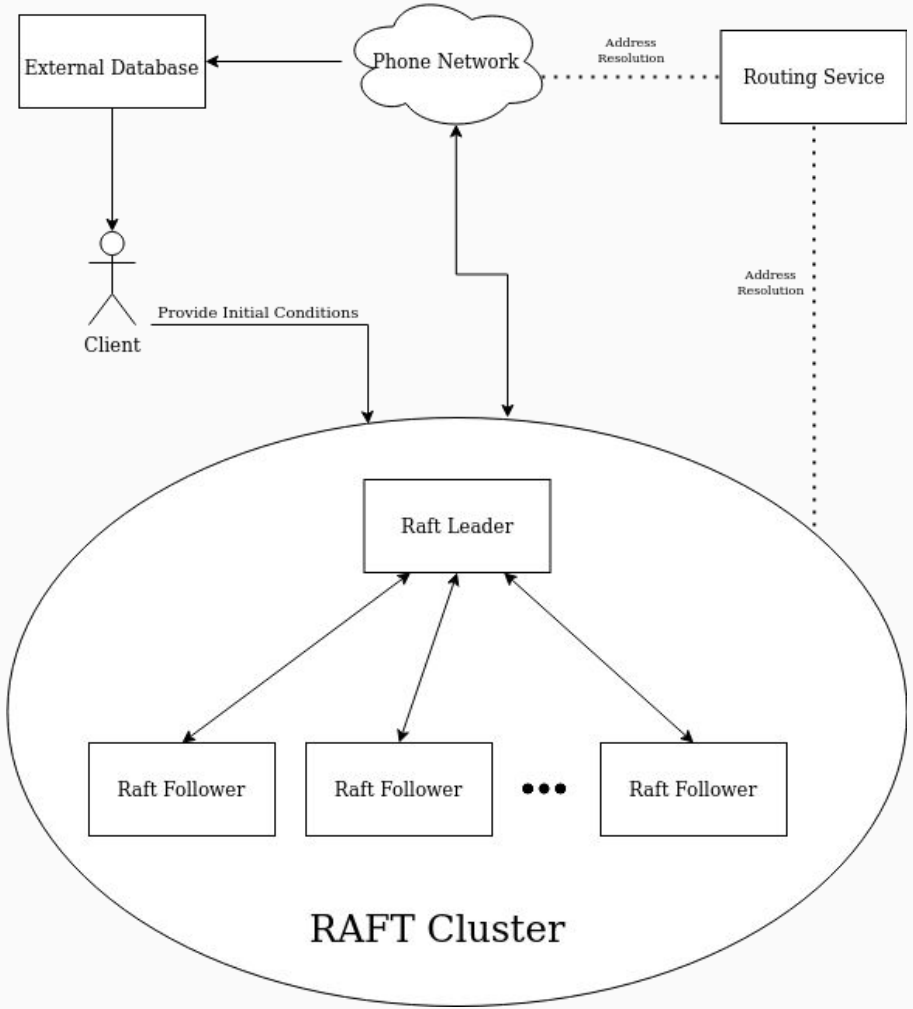
Design

Use RAFT to connect computers and mobile devices into a reliable distributed network

Leader - communicates between phones

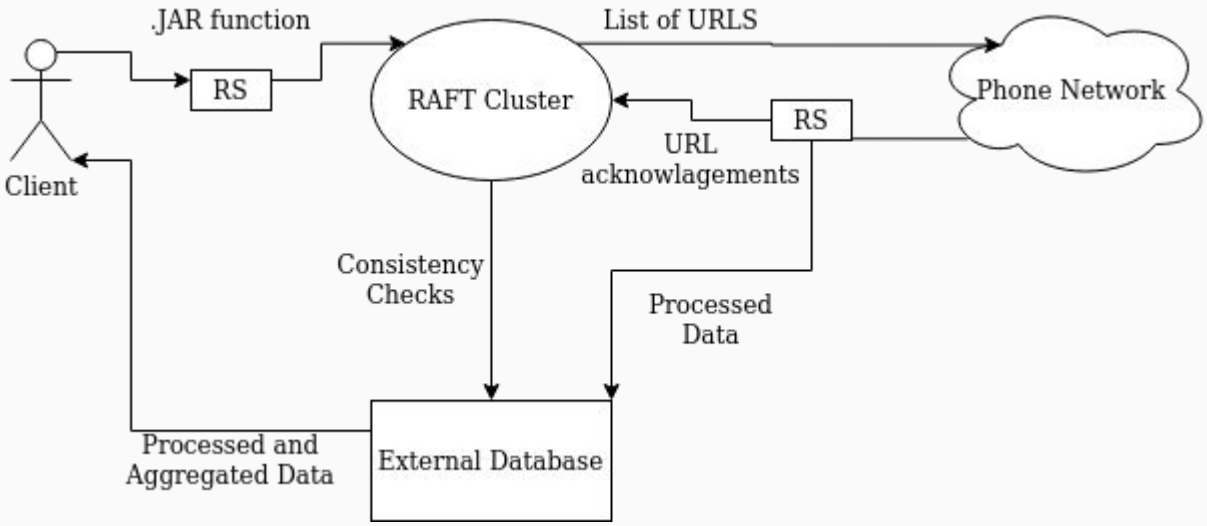
Followers - crawl the provided urls

Phones - scrape urls collected



Data Flow

1. Produce a stream of urls to be processed
2. Distribute the urls
3. Map the HTML of the webpage at each given url to JSON data (via a user provided function)
4. Store the JSON data in an External Database



gRPC & Routing

- gRPC client stubs in Golang, Kotlin, and Android (android uses different libraries)
  - A Service for Raft related RPC calls
  - A Service for web scraping related PRC calls
    - RequestJob()
    - CompleteJob()
- A Routing Service for address resolution (Python & Flask)
  - New clients request addresses of the Raft cluster and Database server
  - Raft leader updates addresses on each election

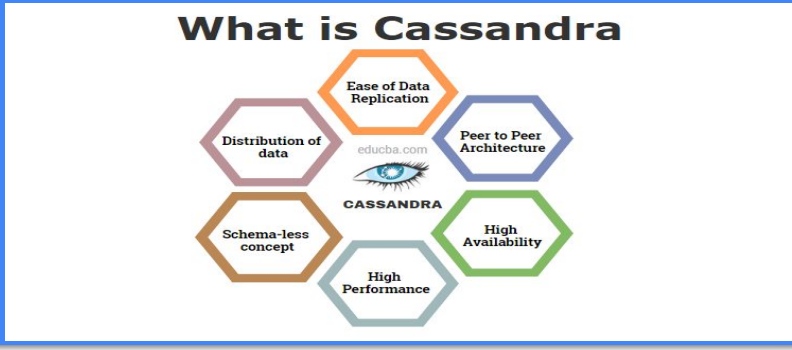
Android / Linux Client

- Requests a Job from the Raft Leader via `RequestJob() -> Job`
  - Receives a list of URLs
- For each URL
  - Make an HTTP request to retrieve raw HTML
  - Parse HTML into a Document (DOM) usingx1 Jsoup
  - Make a call to the user provided scraping function `(Document) -> String`
- Push results to Database Server
- Inform Raft Leader of completion via `CompleteJob() -> Confirmation`

Golang Master Service

- Receive Job requests via `RequestJob() -> Job`
  - Allocate a set of URLS and sent to the client OR
  - Allocate a base domain to crawl
- Receive completed jobs via `CompleteJob(JobResult) -> Confirmation`
  - `JobResult will include`
- Re-allocate jobs which do not get completed within a certain period of time

Database Server



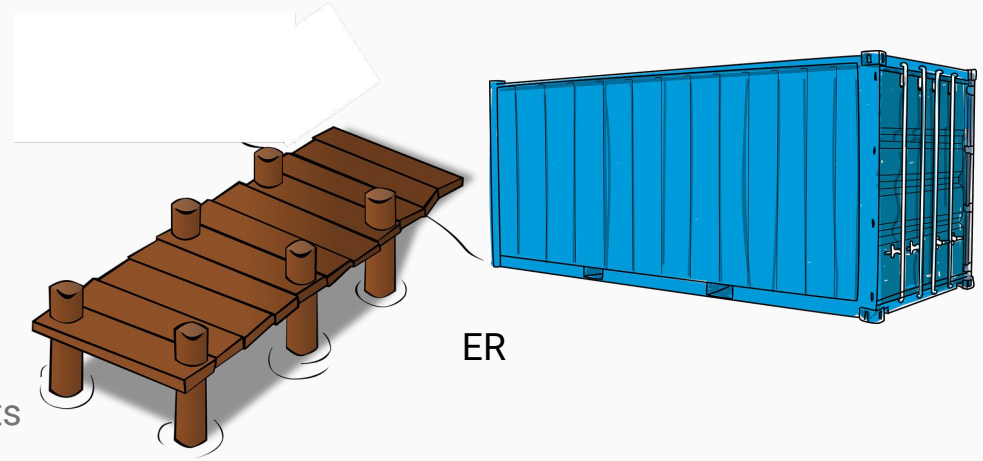
- For the database service we chose to integrate with Cassandra
  - Linear read & write scalability
  - Cheap writes compared to reads
  - Familiar SQL-like query language, without relational data
  - No set schema => easily add table rows on the fly
  - Excellent object mapping in many languages
  - Support for collections as values (including map)
- Clients get connected to the Cassandra cluster via the RoutingService
- Clients write directly to the Database => Less work to do at the Leader
- Urls are used as the PRIMARY KEY => always only one entry per URL

What’s finished so far?

- Golang Master Service
  - Dynamic crawling to discover new URLs
  - Discovered URLs grouped into Jobs & supplied via a channel
  - Dynamic allocation of Jobs to clients
- Kotlin Client / Android Client
  - Successfully receives and processes jobs from the master
  - Stores results into a database (currently produces a word-count)
- Tested over UsedVictoria.com running 9 concurrent clients (over localhost)
  - Counted occurrences of over 55,000 unique strings over 1452 pages
  - For example; the word “victoria” appears `143946 times`
  - See the result at https://blakeasmith.github.io/Distributed-Computing/wc.html

Deployment

- Setup
- Docker
- Continuous Integration
  - CircleCI
  - Go Tests
- Deploy and host on server
  - Google Compute Cloud - $300 credits
  - Heroku - free hobby server



Testing

- Run 100, 200, 500 instances of the client application in Docker Containers
  - Simulate mobile clients dropping frequently
- Simulate Leader failure
- Compare data output to the output of a synchronous web scraping implementation (over a set list of URLs)
- Graph our results
  - Bytes of data being processed with varying number of containers
  - Performance vs Timeout

RAFT Infrastructure

Run RAFT cluster and create GRPC calls structure. Brainstorm how we will communicate with phones and linux machines.

Fault Tolerance

Run RAFT cluster and withstand failures with android devices coming online and offline at will.

RAFT cluster communication

Run RAFT cluster that can communicate with phones and linux machines.

Plugins

Load and run provided plugins and send results to distributed database

Phase II

- Dynamic crawling to acquire URLs
  - instead of using a fixed list of URLs
- Collection of data into a scalable distributed database such as Cassandra
- Multiple client jobs running over the cluster simultaneously
  - Plugin system to define new workflows