Final Project Design

Team 15 – Ethan Grantz, Joshua Jeng, Garrett Mills, Abhigyan Saxena, QiTao Weng

Project Name

IntEarthNet using Swarm

Project Synopsis

Swarm is a small domain-specific language for distributed computing, used to build IntEarthNet, an "internet health check" portal that shows global network speeds.

Project Description

Nearly all frameworks for distributed computing rely on external libraries for existing language to retrofit intra-machine communication using things like MPI. While these work, the integration can be clunky and requires the programmer to have some understanding of the inner working of the library they are using. Furthermore, most of these libraries have no logic for scheduling which nodes the code runs on by feature, instead relying on external schedulers to allocate the nodes beforehand. By contrast, Swarm is distributed by default. Code blocks and iteration are natively parallelized by default, and the logic for synchronizing results and sharing data structures is built into the language natively. Swarm also schedules its resources internally based on declarations of what resources are provided by which nodes. This not only removes the need for an external scheduler, but also requires the programmer to have less of an understanding of the underlying distribution logic and sync mechanisms. We used this DSL to build IntEarthNet, a web portal displaying internet access health around the world by using Swarm to run a suite of network tests on nodes around the world. As the final prototype for this class, Swarm can lex, parse, and interpret basic code to execute an external script, aggregate the results, then store them in a shared database. IntEarthNet reads from that database and displays the up-to-date metrics on a virtual globe.

Project Milestones

Sem 1:

1. Setup environment (tools, language, schema) (9/30)
2. DSL Feature List, Syntax, and Grammar (10/16)
3. Website UML Diagrams (10/16)
4. Complete Lexer (11/27)
5. Complete front-end UI (11/27)
6. Complete website UI (12/29)
7. Complete location pins (12/29)

Sem 2:

1. Complete Parser (1/29)
2. Complete AST (de-)serialization (2/10)

3. Initial pass of interpreter working w/ tests (2/10)
4. Remote Executor completed (3/12)
5. Complete Interpreter (4/2)
6. Complete ping report program (4/29)
7. Final Integration with global servers (5/6)
8. ML Proof of concept (5/6)

(See last page for Gantt Chart)

Project Budget

To demo the final project, we have set up remote executors on international servers according to most popular local hosting services ($12/mo. per server approx.) These are used in the final IntEarthNet report which uses VPS local to the region to do network performance tests against top sites in the region to measure load times, page latency, and packet loss at any given moment.

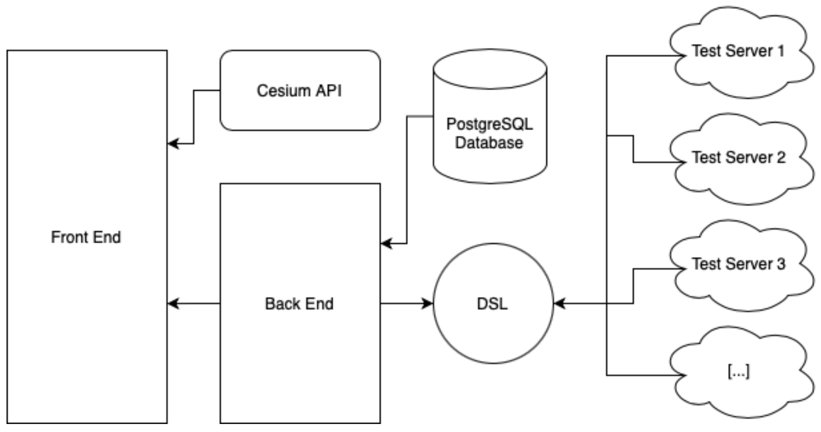
- At least one for each continent - except Antarctica
- Additionally subdivided by industrialization of different regions

Service	**Approximate price**	**Date needed**
US AWS	$5 *3 = 15	2022-03-01 -> 2022-05-31
Brazil HostGator	$14 * 3 = 42	2022-03-01 -> 2022-05-31
Germany Ionos	$2 * 3 = 6	2022-03-01 -> 2022-05-31
Australia OVHCloud	$5 * 3 = 15	2022-03-01 -> 2022-05-31
Israel premiumrdp	$8 * 3 = 24	2022-03-01 -> 2022-05-31
Total	$102 (tax excluded)	

*Likely more as we add additional VPS for countries with vastly different economic development.

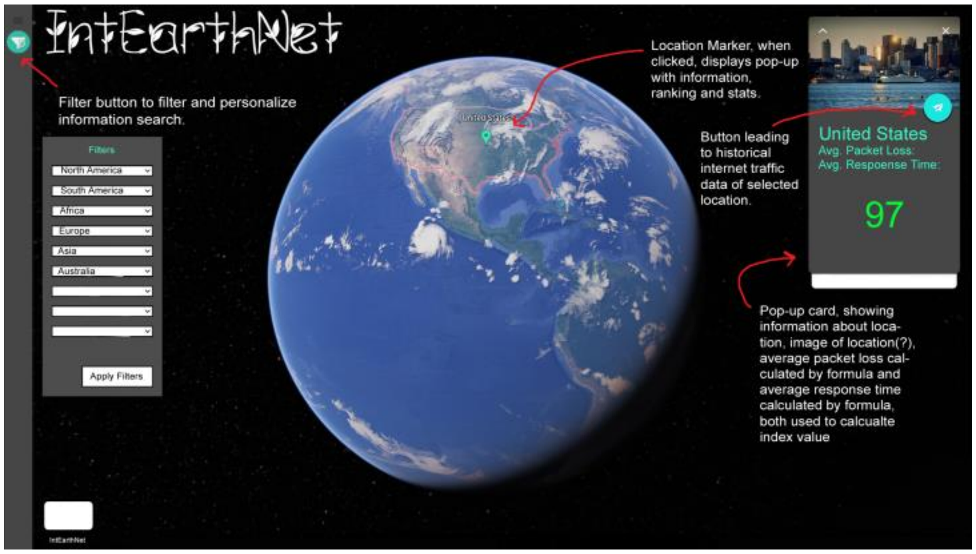
Total: $130

Preliminary Project Design



Website Structural UML Diagram

The IntEarthNet web application leverages an open-source API known as CesiumJS for many major aspects of its front-end. CesiumJS is effectively a JavaScript API that can display various 3D models and various data and labels related to that model. Each web server that is used to generate the internet weather report is represented on the globe by a pin. Thus, to construct this, IntEarthNet's user interface uses Cesium's built-in model of Earth, as well as its pin builder and billboard. The remaining aspects of the UI were designed from scratch in HTML/CSS/JS. To use the data gathered by the DSL, the web application has a PHP backend that queries a PostgreSQL database. The PostgreSQL database stores all information needed to display the data on the globe, including latitude and longitude coordinates, location names, and the actual data gathered by the DSL. Finally, the website itself is hosted on an Ubuntu-based virtual private server, on an Apache webserver.


IntEarthNet
Filter button to filter and personalize information search.
Location Marker, when clicked, displays pop-up with information, ranking and stats.
Button leading to historical internet traffic data of selected location.
United States
Avg. Packet Loss:
Avg. Respoense Time:
97
Pop-up card, showing information about location, image of location(?), average packet loss calculated by formula and average response time calculated by formula, both used to calcualte index value
Filters
North America
South America
Africa
Europe
Asia
Australia
Apply Filters
Website mockup illustrating the pin filter

Users can filter report data per diagnostic server by location. The pin filter feature divides the pins into various sets, divided by continents. The user can select which continents show their data. In addition, basic statistical analyses of the selected data, such as averages or standard deviations, are shown on the screen.

Besides the contemporary internet weather report, the web application also displays historical information, allowing users to compare internet diagnostics over various points in time. As a result, the website can display reports side-by-side across various points in time for easy visual comparison. This data is stored in the PostgreSQL database, to which the DSL adds new data every 5 minutes.

When performing its internet weather report, IntEarthNet needs to execute tasks across the various web servers we have across the globe. To perform these actions in parallel, the website utilizes a domain-specific language called Swarm, which we are writing ourselves from the ground up. Swarm's lexer and parser are built using Flex and Bison, respectively. These two tools have traditionally been the industry standard, so we can trust in their reliability.

The syntax is largely C-like but incorporates intentionally distinct keywords to denote the semantic differences between a synchronous language like C and an asynchronous, distributed language like Swarm. This syntax was chosen due to its high popularity – allowing for a low learning curve and easy adoption for use – and ease of parsing of its syntactic structure.

Most of the language constructs are as you would expect from a C-derived language – things like variable declarations and assignments, function calls, and block designations all track – but we have added our own constructs, like enumerations and resource blocks, which fit our particular use case. A cornerstone of the functionality of Swarm is the enumeration block. An example usage might look like:

```
enumerate mySet as myItem {
    doSomething();
}
```

The key difference between this enumeration block over a set and something like a for-loop iterator over a list in C++ is twofold:

1. The "list" type in Swarm is a set, which has no ordering.
2. Likewise, enumerations do not execute in a guaranteed order, and may execute in parallel.

So, the Swarm interpreter takes the items in the set and breaks up the processing within the block, so it all happens in parallel among different workers – one "control stream" for each item in the set. This is analogous to all the iterations of a for-loop executing at once.

As a result, Swarm introduces concepts to control two things: safe distributed access to shared data structures, and a way to request the execution of a block with access to specific computing resources.

We use Redis to implement a distributed lock to ensure synchronization issues such as race conditions can be prevented natively through the Swarm itself. Swarm allows users to create shared variables and data structures that behaves consistently when operations across multiple machines are executed concurrently. Swarm is able to pass along variables and data structures during inter-process communications. Allowing the execution of the language to solve synchronization issues makes distributed computing much more easily accessible. Instead of booting up an instance of the same processed manually and iteratively on different computers and data centers and requiring the user to create a separate locking mechanism to prevent issues from arising when operating on the same data.

To solve the other case, Swarm introduces a concept of resource-blocks. These blocks request a computing resource that needs to be available to the executor for the code to execute correctly. For example, say we want a block of code to execute only if a particular file is available on the filesystem:

```
with file('/var/myfile.csv') as fileBuffer {
    doSomething();
}
```

In this case, the with-statement tells the interpreter that the code block within the braces can only be run on an executor that has the `/var/myfile.csv` file available. The interpreter finds an executor satisfying that criterion and the code within the with-block is run on that executor, which makes the file resource available in the `fileBuffer` variable.

This was accomplished with a two-stage compiler-then-interpreter. The compiler takes the Swarm source code, lexes it, then parses it into an AST. The AST was annotated with type information and has its symbols resolved, so they can be linked throughout definitions & references. Operations on the AST are implemented as a series of walks over the AST, including the interpretation pass. While interpreting the AST, if we encounter a block in the source that should be distributed/parallelized, that subtree of the AST is serialized and sent to the remote host to be deserialized and executed.

There are some design constraints considered. The user interface for the IntEarthNet application is built for a web-based platform, utilizing the CesiumJS framework. As part of the open-source Apache license, we include an open-source licensing page on the website. Besides the normal web languages (HTML, CSS, JavaScript, PostgreSQL, PHP, etc), the parallelized tasks are done using our domain-specific language Swarm. While we do have expenses, namely the access to various worldwide servers required to check global internet speeds, the cost is covered by the University of Kansas without any predefined budget constraint. Finally, as with all other Senior Design projects, the project deadline is in May 2022.

Ethical Issues

IntEarthNet is an application that pings various servers across the globe and then records data into an SQL table, displaying it onto a three-dimensional globe in the form of pins. Overall, we do not believe that IntEarthNet has any ethical issues per se. However, the data centers we are pinging are a major risk to the environment. Data centers are what can be equated to, factories of the information age, data centers make communication across the internet possible, but it comes at the cost of operating the sheer (and growing) amount of data centers using electricity. The energy consumption of digital technology is increasing rapidly over the year which is a major concern for the environment.

Intellectual Property Issues

Much of the work done in our project is original, so there are not many intellectual property issues concerning the originality of the code. However, we need to take care to credit and/or make original snippets of code used from examples or documentation for the libraries we are integrating. This is a broader issue in the software industry in general, but in our case that means crediting snippets of code used in the Swarm parser/interpreter that were taken from Flex, Bison, and class examples. Likewise, with IntEarthNet, we are building our UI based on the open-source Cesium project which means referencing their documentation quite heavily. As such, we need to take care to credit the original authors of the examples we reference, if any, in our final product. Finally, there's the question of who owns the code we write. This will be something we will hash out amongst ourselves as the project concludes. Different sections of Swarm and IntEarthNet were written by different authors so integrating the authors' IP is a concern.

Changelog

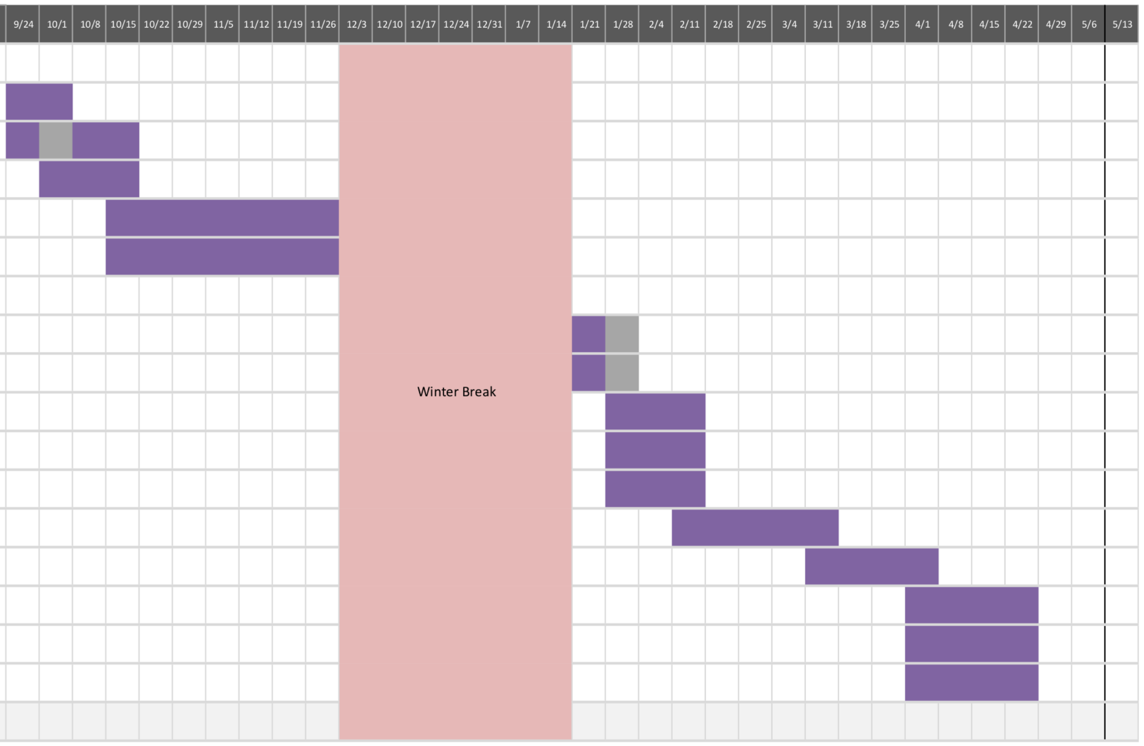
- Google Earth turned out to be more difficult to use than anticipated, so we found an alternative in the form of CesiumJS. In addition, while CesiumJS does support using KML files, we were able to remove them from the design, as it is simpler to query data directly from the database and store it with JavaScript.
- A feature to be able to filter the pins that are displayed was added.
- Recognized the need to include an open-source license page to IntEarthNet.
- Interpretation will no longer be done using an IR, instead it will be done right off the AST.
- Based on feedback from our project proposal, we added dates for which we expect to pay for web services.
- We have updated our spring semester milestones in accordance with what we accomplished in the fall semester.
- Language and tensing have been updated according to progress from last semester.

IntEarthNet

University of Kansas
Team 15

	Tue, 1/18/2022
Project Start:	Fri, 9/24/2021
Display Week:	1

TASK	ASSIGNED TO	PROGRESS	START	END
Semester 1				
Setup environment (tools, language, schema)	All	50%	9/24	9/30
DSL Feature List, Syntax, and Grammar	Garrett, QiTao, Joshua	20%	9/26	10/16
Website UML Diagrams	Ethan, Abhi	0%	9/30	10/16
Complete Lexer	Garrett, QiTao, Joshua	0%	10/16	11/27
Complete front-end UI	Ethan, Abhi	0%	10/16	11/27
Semester 2				
Complete Parser	Garrett, QiTao, Ethan	100%	1/18	1/29
Complete location pins	Abhi, Joshua	80%	1/18	1/29
Debug name/type checking & write tests	QiTao	20%	1/29	2/10
Complete AST (de-)serialization	Garrett, Ethan	0%	1/29	2/10
Initial pass of interpreter working w/ tests	Garrett, QiTao, Ethan	0%	1/29	2/10
Remote Executor completed	Garrett, QiTao, Ethan	0%	2/10	3/12
Complete Interpreter	Garrett, QiTao, Ethan	0%	3/12	4/2
Complete ping report program	Ethan	0%	4/2	4/20
Final Integration with global servers	Abhi, Joshua	0%	4/2	4/20
ML Proof of concept	QiTao	0%	4/2	4/20


9/24
10/1
10/8
10/15
10/22
10/29
11/5
11/12
11/19
11/26
12/3
12/10
12/17
12/24
12/31
1/7
1/14
1/21
1/28
2/4
2/11
2/18
2/25
3/4
3/11
3/18
3/25
4/1
4/8
4/15
4/22
4/29
5/6
5/13
Winter Break