Midterm Project Proposal

Noah Fryer

October 15, 2020

1 Problem

I am choosing to use the Distributed Sudoku starting project example described in the course website. The project takes the classic $9 * 9$ Sudoku board and considers each $3 * 3$ square as a separate node in a distributed system. Each node may only communicate with its adjacent neighbors, which forces the nodes to pass messages between each other to compute an acceptable configuration of the board. The problem of solving the board this way can be broken down into major steps:

1. Creation of 9 independent nodes
2. Establishment of communications between adjacent nodes
3. Generation of the Sudoku Board
4. Division of $3 * 3$ squares among the 9 nodes
5. Solving the board
6. Establishing consensus that a solution has been reached

Some of these steps may be broken down significantly further. The board must either be constructed by a single node or each node must generate its own square. The general procedure of creating a valid Sudoku board has its own steps. The division of the board among the nodes will involve a number generation step and a communication step where the nodes either divide and disperse a board amongst themselves, or they agree that they have each constructed a $3 * 3$ square that creates a valid whole. The general procedure for solving the board has its own steps, and in this context it is given additional complexity as each node checks with its neighbors and searches for potential solutions.

The creation of the nine nodes could be handled in a variety of ways; I will choose to use Docker for this project in the same way that we used it in the first assignment. Docker will create a container for each node such that they are isolated and must communicate using a network interface. The requirement that nodes only communicate with their adjacent neighbors will be enforced by the container that each node “lives” in. The nodes will communicate via gRPC calls.

The Sudoku board may be generated using a generalized algorithm. Squares in the diagonal are independent of each other, and they may be filled with $1-9$ while the other squares are initially left blank. Then, the other squares are recursively filled until they represent a complete board. Finally, a number of numbers are removed, and the resulting incomplete board will be the puzzle for the system to solve.

Solving the board will involve a great deal of communication between the nodes. The first node will start in its first available column in its first available row (i.e., $(0,0)$) and choose ‘1’. It will check if ‘1’ is available within itself and if it is available in the rows and columns of its adjacent neighbors. If the answer is no, it will increment by ‘1’ and check again. It will proceed down the row until it reaches the square governed by the next node. If it finds a cell where no number is available then it will move back to the previous cell, increment by one, and check again. Once it reaches the next node, it will “pass off” to its neighbor, which will repeat the same process down its row. If the neighbor must backtrack to the previous square, it will “pass off” to the last node. The third node in the row will complete its row, and then “pass off” to its previous neighbor and indicate that the neighbor should pass off to the first node in their node row so that they next row of cells can start at the left side. In other words once a row of nodes has completed their work, they will pass off backwards and then downwards. The next rows will continue moving forward and then backtracking as necessary until the puzzle is solved.

Protocol

The procedure described above has many checks and “pass offs” underlying it. A node will call a service in its neighbors to check if a number is available for a particular cell. It must communicate what number it wants to use and which row or column it is checking. Furthermore, its neighbor must also ask the next neighbor down the line the same question before it gives the caller the affirmative. If the candidate number does not violate the rules, the neighbors will both answer “yes.” Otherwise, they will say “no” and the node that is currently working will try a different number.

When a node has finished a row of cells, it must make a call to its neighbor to take over. When a row of nodes decides that they are complete (all three of their cell rows have been filled), they will need to coordinate and hand off to the next row of nodes such that the next row starts with the leftmost node. Finally, the last node must be aware that it is the last, and indicate that it has found a valid configuration so that all the other nodes know that they are finished. They will need to transfer their squares to each other and finally to a coordinator that will be responsible for compiling the board and printing the output.

2 Distributed Systems Discussion

2.1 Architecture

In the problem and solution I have laid out above, the individual nodes are acting as both clients and servers in a peer-to-peer network whose structure is defined by the constraints of the problem specification. Section 2.3 in Tanenbaum and Steen's text discusses three different architectural styles; peer-to-peer networks are considered a decentralized style. "In this type of distribution, a client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load" (Steen 81). This particular system is structured–the nodes are in a grid overlay in which they cannot all directly communicate, but they are sufficiently connected such that they can relay messages.

The nodes must be spatially aware of where they are located in relation to the game board so that they know which neighbors to ask and which neighbors to pass off to. There is probably an intricate automated solution for distributing the matrices of the board and making adjacent neighbors aware of each other, but for this project I may predetermine where nodes lie on the board using a docker-compose configuration.

2.2 Communication

Subtle problems arise when using Remote Procedure Calls, as I will be with gRPC. The main issue concerns pointers and references being passed between machines operating in different contexts. gRPC takes care of proper parameter marshaling behind the scenes. An issue could arise if I tried passing "...very large, nested, or otherwise intricate data structures..." but the protocol that I outlined in the problem section avoids this by passing simple "yes" or "no" answers (Steen 179-182). The most complex communication step will involve passing matrices at the beginning and at the end to construct the board and then print the output.

2.3 Coordination

The nodes will coordinate to solve the puzzle by asking each other questions about their squares and taking turns modifying the board. At any given time, only one of the nodes should be actively trying to fill in cells. This scheme simplifies leadership and mutual exclusion problems while the board is being solved, but ultimately they all have to agree if the puzzle is solved and decide which one will output the completed board. If the node at $(0, 0)$ starts first, then $(3, 3)$ will finish the game. At this point it can announce that it is finished so that everyone knows to check their configurations one more time. They will pass a yes or no answer back to $(3, 3)$, who will then ask them to pass their squares to it so that it can print the final output. If one of the nodes indicates that its configuration is not correct at this point, then the solving algorithm has not been implemented correctly.

2.3.1 Consistency and Availability

The peer-to-peer structure of the network avoids some problems with availability (the nodes should not be trying to modify each other's squares), but creates new problems with consistency. If one of the nodes goes down, how will the others resurrect it so that the structure of the board is not lost? One node could hold a "master" copy of the board and distribute this copy when there is an indication that a node has been lost. It is also possible that adjacent nodes will hold copies of each other's squares. A solution in which the master copy is modified as solutions are explored would be intricate, and for the purpose of this project I will only attempt to maintain a static copy of the original board from before the solving part of the game began.

Since the nodes are solving the puzzle row-wise, it would make sense for all the nodes in a row to hold a copy of that row's squares as they were when the game first started. No attempt will be made to modify a row as the game progresses; if a node crashes, all of them will have to start over. Progress will be lost, but a new board will not have to be generated. Since all of the nodes will be running inside Docker containers, Docker will ultimately be responsible for restarting a node. Once a node has been restarted, one of its neighbors can pass its square back to it and restart the game.

3 Technologies

- Docker can set up nine different containers on the same machine (or instance) and run each node in a container such that they must use network facilities to communicate. These techniques were used in Assignment 3.
- I will use Python as the implementation language
- The nodes will communicate using gRPC similarly to how we used it in Assignment 3.
- The implementation will run on a virtual Ubuntu instance on the Google Cloud Platform account that I set up for this class.
- Version control through a private github repository.

4 Testing

- Test Driven Development in Python
- Python Testing Modules
- pytest: helps you write better programs

This is my first time using Python for a large task. I will subscribe to the test-driven development model that the references above describe. In particular, I plan to use the `py.test` module to create unit test suites that I can run off of the command line. Tests should be implemented before functions are written, and test suites should be run before commits are moved into the main git branch.

5 Benchmarks

The primary benchmark for this project will be its correctness. The solving algorithm outlined above is a brute-force approach. It is not expected to be an optimal solution. It may be interesting to keep track of the number of cycles that are required to reach a solution for a particular board. Other interesting data would include the number of times a node asks its neighbors about a cell entry and the number of times that the nodes pass off control between each other.

6 Work Plan

The first steps in this project will be writing the `.proto` file to model the communication protocol off of and getting `docker-compose` running the way that I need it to run. I can have those done next week. After that I'll be building tests for small for functions, functions, and implementing the services specified in the `.proto` file.

1. Implement row and column checking functions/gRPC calls
2. Implement node "pass off" gRPC calls
3. Implement solution announcement/square-passing gRPC calls (October 23rd)
4. Generate a sudoku board (November 8th)
5. Implement backtracking algorithm (November 22nd)
6. Implement square compilation and output (November 30)

All of the services outlined in the `.proto` file should be implemented first because they are the foundation for all communication, and writing them out should be fairly mechanical. The hard parts will be making the board and then solving the board. They will be broken down into small functions (according to test-driven development, the tests should be written before the functions are) that should be almost identical across all nodes. Finally, the solution compilation will be implemented and output presented on the command line. All steps will involve testing to ensure correctness before introducing additional complexity.