Project 4
Nathan Bunch, Teresa Soley, Ryan Nickelsen, Ryan Ozzello

How it works:

Our program attempts to solve consistency problems involved in our database website and its distributed system across four of our computers. Our goal was to develop a program that implements eventual consistency so that users accessing our database from any machine can receive timely and up to date requests. Our program uses vector clocks to understand what version of the database each machine has so that older versions are not replicated making the data out of date and inaccurate.

Each machine has a vector clock noting its version of the database. Whenever an update is received or sent the local machine increments its version by one and takes the maximum versions received of the other machines. This tells each machine how up to date it is at each request received. Our put function, open for users to insert into our database, works in two ways. First, when a user makes a direct put request, the local machines knows to update its database and vector clock and then replicate this information to other computers. Secondly, if a request from an outside machine is being received, noted by the presence of a vector clock in the received package, then the local machine checks if the update is from a newer or older version than its own and accordingly updates its own data.

To handle a direct put request from a user and replicate the data, our program simply calls a replicate function that tells other computers to run their put function with the arguments of the local machine that sent the request. To accomplish the second purpose of receiving requests from other computers, we implemented two functions that detect a conflict between vector clocks and check if they are newer or older. A conflict is when two vector clocks are the same. This is handled by arbitrarily choosing the longer data in length to trump the shorter. To determine a newer or older version, causality rules apply.

We have also made a way to add new machines (nodes) to our system. If a new computer is added to the system, then this new machine would know the existence of the other machines but not vice versa. Therefore, if a computer receives a request with a vector clock that has additional machines, it will update its vector clock and machines.txt file to include the new computer.

Difficulties:

A difficulty in designing the vector clocks was debugging. When a put request is made and replicated, we noticed that the local machines vector clock is incremented by two, rather than one like on other machines, preventing vector clocks between machines to be the same after a replication. Despite this faulty replication of the vector clocks, our system still makes a put request and updates all machines without infinite looping or failing to replicate.

Missing features:

Due to time constraints, we were unable to add an eventual consistency system for our get function. In theory, if a key was missing in the local machine it would ask other machines for the data and update its vector clock and data if it were found. We also wanted to include a function that runs in the background, updating the local machine even when users are not making requests. This would ensure that our machines would be eventually consistent even when all requests have stopped. Since we are lacking in these features, we currently are unable to get up-to-date a computer that goes offline.

Program output:

Shown in person.