**Architectural Support for Cloud Computing – Project report**

**(Surbhi Shankar)**

**Project description –** This project is a simulation of a data center in a parking lot which has a capacity to hold 2560 cars at any point of time. The total number of cars considered for the simulation is 7680. The structure of the parking lot is designed as four sections called regions and then further divided into four centers and then into four clusters. Each cluster can hold forty cars. This arrangement helps in allotting parking spaces and also assigning jobs and other calculations. The cars are in the parking lot for about 8 hours during which they carry out jobs assigned to them. We also keep copies of the data stored in each of the vehicle so that the data is not lost. If a car leaves before the job assigned to it is completed, then the job has to be migrated to another car which is still in the parking lot. The results of this simulation are stored in a text file which is easy to understand.

**Classes defined –**

There are four major classes in this project which are

1. Class parking\_info

This class is constructed to access information about the parking lot itself. It has member functions and variables related to the parking lot such as Region ID, Center ID (GC as per the instructions document), Access Point ID, Parking Spot ID and so on. This class also contains a member function to print all the parking information.

1. Class cars

This is a class exclusively for data about cars that enter the parking lot. This class supports all the information like Car ID, Jobs associated with the car, Parking information of the car and so on. This also include member function to print the information related to the car.

1. Class jobs\_assigned

This class defines the jobs that are assigned to every car in the parking lot. It defines variables and member functions that gives information about job assigning, car availability information and also has a member function that prints the information about jobs assigned to cars.

1. Class Time\_track

This is simple function that helps to get the timings of the simulation.

**Assumptions –**

1. There are a total of 2560 parking spots available and when the simulation begins, all the spots are filled up. When the simulations begins, all the parking spots are initialized with car numbers and then jobs are assigned to each car.
2. The car stays in the parking lot for 8 hours. (The departure time is calculated in a function – departure\_time())
3. This simulation is programmed to run for 24 hours, which is one whole day.
4. The input we are giving is Arrival time of the cars. This is stored in a text file which is randomly generated in the function timefile().
5. The job durations are uniformly distributed in [3,24] hours. Each job taken as input of 2GB (data produced is between [0.5,2]GB.
6. The parking lot has four regions – R1, R2, R3 and R4. Each region is divided into four centers and each center is divided into four clusters. Every cluster has forty parking spots. This is a very easy method to track the cars and the jobs assigned to them.
7. VM size is 1GB and it is constant throughout the simulation.
8. Data replication process is carried out to store the data so that it is not lost. First copy is within the car where the data is generated, second copy goes into a car that is parked within the Access Point (cluster) where the original car is parked. Third copy is stored in a car that is parked in one of the cars that is parked in a different access point but within the same region.
9. Data migration happens when the car leaves without completing the job. The incomplete job is assigned to another car which has a longer stay time (time it stays in the parking lot).
10. All the required data is stored in text files which can be accessed after the simulation is executed.
11. Most of the assumptions are mentioned in comments in between the code lines.

**Data migrations strategies –**

**Basic idea -** We can create a flagged variable (migrate is the variable used in the implementation), which is a boolean variable. It can be set as “true” or “false” depending on the status of the job, if it needs migration or not.

When migrate = false, the process continues or the job is terminated. This is happens when there are no other available cars or if the data of the job is lost.

When migrate = true, then the car ID is changed to another car ID that is available.

1. The leaving car can migrate the job to the data center controller (DC) before leaving and the DC decides which car should be assigned with the unfinished job that is under consideration.
2. Easiest method of all is to find the car which arrives closest to the departure time of the leaving car will be assigned with the job that is unfinished. This does not compare the job duration left with the stay time of the cars. (This has been implemented in cloudvehicle\_arrival\_time.cpp)

**Evaluation:** This method of migration is creating an empty Failure Job Info text document which is not realistic during a real data center.

1. We can compare the job duration remaining after a part of it is completed by the leaving car with the stay time of the car. This includes the cars where the job is completed and the car is still going to be in the parking lot for some more time.

**Evaluation:** After implementation, this was found to be the most efficient method. This is because, it takes less job transfer time because it is directly transferred from a car to another car which is idle. Also, there is less or no car that is left idle in the parking lot (which might happen when we follow the 2nd method of migration). (This has been implemented in cloudvehicle.cpp)

**Input and OutPut files –**

**arr\_time\_output**– Contains all the arrival timings of the cars. This is considered as an input file.

**Car\_Info** – Gives the information about each car that is in the parking lot and the allotted job information.

**ParkingDetails** – This file contains information about the parking lot which is very important to know the location of the car.

**TotalCars\_simulation** – Gives information about total cars in the parking lot which undergo simulation.

**Car\_Simulation\_Info** – Information about the car with respect to simulation details and car details.

**Completed\_Job\_Info** – This gives information about the cars that successfully completed the jobs.

**DataReplication** – Data is replicated into other cars. This information is printed in this file.

**Failed\_Job\_Info** – This gives information about the cars that could not successfully completed the jobs.

**MigrationSuccessful** – After successful migration, information about job migrations that occur during the simulation is stored in this file.

**MigrationFailure\_Info** – This file contains information about migration failure that occurs during the simulation.

**NewCar** – Contains information about new cars that enter the parking lot.

**NoParkingAllotted** – Information about cars which are idle with no job allotted.

**Note:** The codes are not completely functional. Some of the output files are created but sometimes with no values in it. Some output files are empty and I could not fix the issues. (I tried a lot to fix the bugs but I was not able to).

**References:**

<http://www.cplusplus.com/forum/beginner/98214/>

<http://stackoverflow.com/questions/19641597/what-is-segmentation-fault-core-dumped>

<https://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm>

<http://www.cplusplus.com/doc/tutorial/classes/>

<http://stackoverflow.com/questions/1711990/what-is-this-weird-colon-member-syntax-in-the-constructor>

<http://www.cplusplus.com/doc/tutorial/files/>

<https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm>

<http://www.cplusplus.com/forum/articles/10627/>

<http://en.cppreference.com/w/cpp/header>