Improving Avg Exec Time in ds-sim

1. Author

Name	OneID
Fei Huang	44129866

2. Abstract

The idea of this algorithm is to calculate the fitness value between each server and the current work, and then convert the fitness value into the weight corresponding to each resource, and finally add the weight of the 3 resources of each server. After that, perform an ascending sort again. The server with the lowest total weight is the one with the best resource utilization because it wastes the least resources, so it is more likely to run one more job on this server.

3. Introduction

Due to there are a lot of servers in the ds-sim system, and the configuration of each server is not the same. Moreover, there are many jobs that need these servers to complete, and the performance requirements of each job are also different. Therefore, for this complex situation, the optimization direction I chose is to improve the performance of each server to complete the work that reduces the Avg Exec Time.

The resources of each server are fixed, so in order to shorten the Avg Exec Time, I hope to be able to perform multiple jobs on each server through the algorithm as much as possible to avoid waste of extra resources.

4. Definition

- Fitness Value, it means that the current server has many available resources in addition to the resources required by the current job. There are three fitness values in ds-sim, namely fitnessCores, fitnessMemory, fitnessDiskSpace. If a specific fitness value is negative, it means that this resource of the current server does not meet the conditions for running the current job. Its calculation formula is:
  Fitness Value = Specific Resource on Current Server - Specific Resource for Current Job required.

- SID, used to confirm the unique ID of each server, all subsequent operations will use SID as an identifier. Its calculation formula is:
  SID = Server Type + Server ID.

- Weight, after calculating the fitness value of each resource of each server, it is put into a List in ascending order, and then the weight ratio of the specific resource of each server in this List is calculated. The closer the weight is to 0, the closer the fitness value is to 0, which means the closer the specific resources of this server are to the work requirements. Choosing a server with a

lower weight to perform the current job can increase the resource utilization of this server. Its calculation formula is:
Weight = Index of Specific Resource List (asc sorted) / Specific Resource List Size.

- Total Weight, the total weight is to add up the weights of all resources of each server (in this case, the number of resources is three), and the total weight obtained is to find the server whose comprehensive resources are closest to the work requirements. Its calculation formula is:
  Total Weight = Server Cores Weight + Server Memory Weight + Server Disk Weight.

- Parameter, a series of parameters used to evaluate the efficiency of the algorithm, including Actual Simulation End Time, Total Servers Used, Avg Utilisation, Total Cost, Avg Waiting Time, Avg Exec Time, Avg Turnaround Time.

- Performance Comparison, this parameter is for comparison with other algorithms (in this case, others are FF, BF and WF). Through the performance comparison, we can see which specific improvements or decreases in each algorithm of my algorithm. Its calculation formula is:
  Performance Comparison = -((My Score / MIN(the Score of FF, BF, WF)) -1).

5. Algorithm design

I will first obtain all the servers capable of the current job through the RESC Capable command and put them into a list. Then through for loop these servers, exclude servers whose server's state is unavailable or fitness value <0. I will put the rest of the server in the fitness map of various resources, map's key is SID, map's value is specific fitness value.

When all the servers have completed the for loop, my algorithm will check each fitness map. If there is any map is empty, it means that no current server configuration can meet the requirements of the current job, so I directly return the first server in the server list.

Next, I will put these fitness maps into the other 3 fitness lists, and sort them in ascending order according to their corresponding fitness value.

After the fitness list is sorted, my algorithm will start to calculate the fitness weight of the resources corresponding to each server.

Then, I will add the three fitness weights just calculated according to the SID of each server to get their total weight.

Finally, I also need to sort the total weight of each server in ascending order, and then return to the server with index 0. This is what I need to find the best server for the current job.

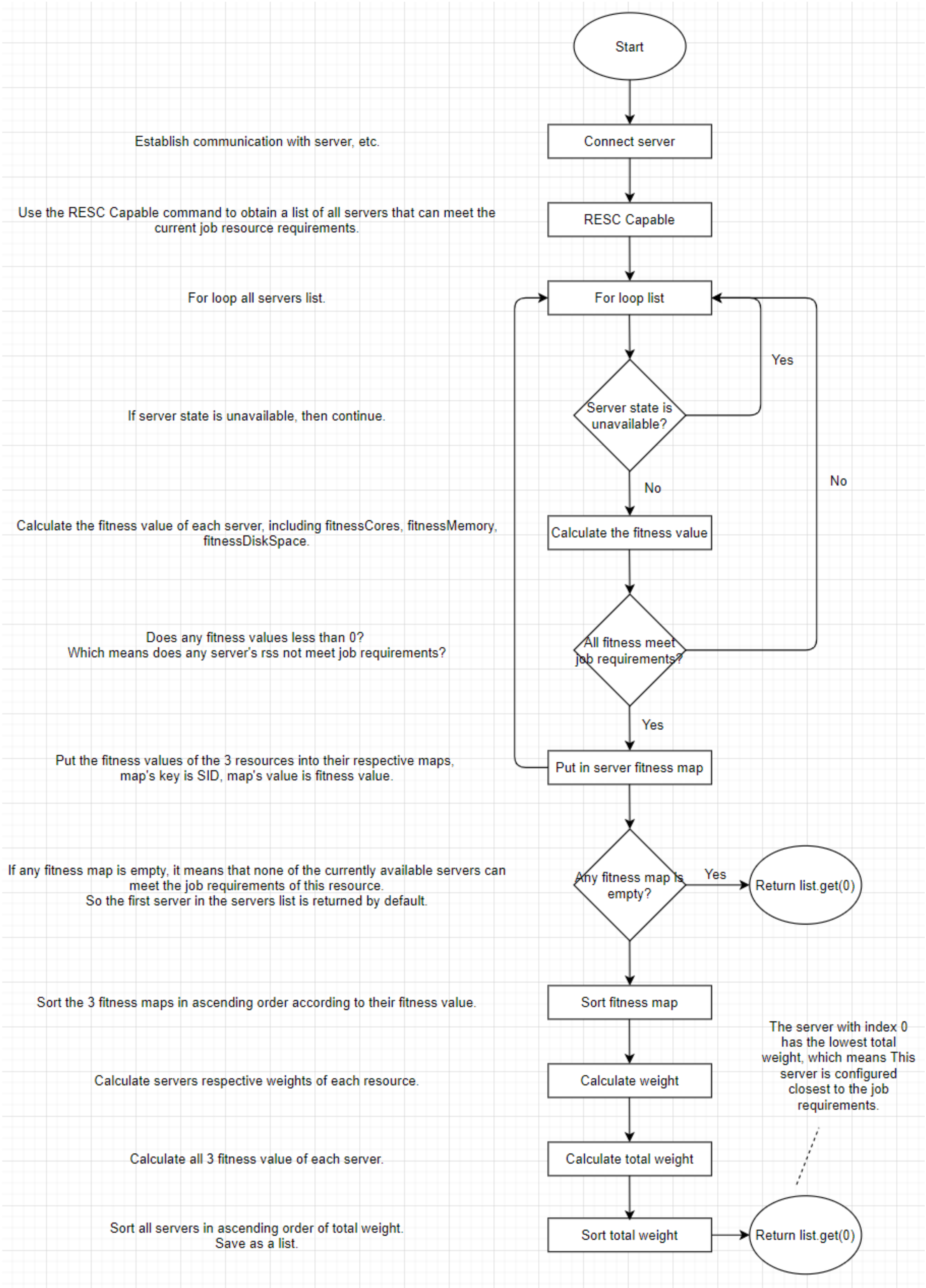

Figure 1 Algorithm Flowchart

6. Pseudo code

```
DECLARE an ArrayList<HashMap<String, String>> variable serverList
DECLARE an HashMap<String, String> variable schdServer

DECLARE a HashMap<String, HashMap<String, String>> variable championServerMap

DECLARE a HashMap<String, Integer> variable serverFitnessCores
DECLARE a HashMap<String, Integer> variable serverFitnessMemory
DECLARE a HashMap<String, Integer> variable serverFitnessDisk

DECLARE an ArrayList<HashMap<String, Integer>> variable serverCoresRank
DECLARE an ArrayList<HashMap<String, Integer>> variable serverMemoryRank
DECLARE an ArrayList<HashMap<String, Integer>> variable serverDiskRank

DECLARE an ArrayList<HashMap<String, Double>> variable serverCoresWeight
DECLARE an ArrayList<HashMap<String, Double>> variable serverMemoryWeight
DECLARE an ArrayList<HashMap<String, Double>> variable serverDiskWeight

DECLARE an ArrayList<HashMap<String, Double>> variable serverTotalWeight

SET serverList to RESC Capable's result
```

```
champion:
  // Algorithms of Champion
  // This algorithm focus on finding the best-fitness server for each job,
  // by sequentially looking at different server states.
  // This algorithm can adjust the Avg Turnaround Time and Avg Exec Time to improve
work efficiency.

  CALL restoreChampionParameter

  FOR each server of serverList
    DECLARE a String variable currentServerType
    DECLARE a String variable currentServerID
    DECLARE a String variable currentServerState
    DECLARE an int variable currentServerCPUCores
    DECLARE an int variable currentServerMemory
    DECLARE an int variable currentServerDiskSpace

    SET server's type to currentServerType
    SET server's id to currentServerID
    SET server's state to currentServerState
    SET server's CPU cores to currentServerCPUCores
    SET server's memory to currentServerMemory
    SET server's disk space to currentServerDiskSpace

    IF currentServerState is unavailable THEN
      CONTINUE
    ELSE
      DECLARE an int variable fitnessCores
      DECLARE an int variable fitnessMemory
      DECLARE an int variable fitnessDiskSpace

      SET fitnessCores is currentServerCPUCores - job's cores requirements
      SET fitnessMemory is currentServerMemory - job's memory requirements
      SET fitnessDiskSpace is currentServerDiskSpace - job's disk requirements

      IF fitnessCores<0 or fitnessMemory<0 or fitnessDiskSpace<0 THEN
        CONTINUE
      ENDIF

      DECLARE a String variable SID

      SET SID is currentServerType + currentServerID
      SET map's key is SID, map's value is server in championServerMap

      SET map's key is SID, map's value is fitnessCores in serverFitnessCores
      SET map's key is SID, map's value is fitnessMemory in serverFitnessMemory
      SET map's key is SID, map's value is fitnessDiskSpace in serverFitnessDisk
    ENDIF
  ENDFOR
```

```
   IF serverFitnessCores is empty or serverFitnessMemory is empty or serverFitnessDisk
is empty      THEN
      SET first item in serverList to schdServer

      RETURN
   ENDIF

   CALL sortServerMap(serverFitnessCores), and SET its result to serverCoresRank
   CALL sortServerMap(serverFitnessMemory), and SET its result to serverMemoryRank
   CALL sortServerMap(serverFitnessDisk), and SET its result to serverDiskRank

   CALL weightCalculation(serverCoresRank), and SET its result to serverCoresWeight
   CALL weightCalculation(serverMemoryRank), and SET its result to
serverMemoryWeight
   CALL weightCalculation(serverDiskRank), and SET its result to serverDiskWeight

   CALL totalWeightCalculation(serverCoresWeight, serverMemoryWeight,
serverDiskWeight), and SET its result to serverTotalWeight

   CALL sortServerTotalWeight(serverTotalWeight), and SET its result to
serverTotalWeight

   CALL setChampionServer(serverTotalWeight)
```

```
restoreChampionParameter:
   // Reset all the variables that will be used in order to find the champion server.

   INITIALIZE championServerMap

   INITIALIZE serverFitnessCores
   INITIALIZE serverFitnessMemory
   INITIALIZE serverFitnessDisk

   INITIALIZE serverCoresRank
   INITIALIZE serverMemoryRank
   INITIALIZE serverDiskRank

   INITIALIZE serverCoresWeight
   INITIALIZE serverMemoryWeight
   INITIALIZE serverDiskWeight

   INITIALIZE serverTotalWeight

   INITIALIZE schdServer

sortServerMap:
   // Sort the servers according to the fitness value of each resource from small to large.
   // Map's key is SID, which is ServerType + ServerID
   // Map's value is fitness value of specific resource.

weightCalculation:
   // Calculate servers respective weights of each resource
   // The smaller the weight, the higher the priority,
   // which means that the current resources of this server are closer to the job
requirements
   // Weight = index / list.size ()

totalWeightCalculation:
   // Calculate all 3 fitness value of each server.
   // Add up the weights of the 3 fitness value of each server,
   // to get the total weight of each server.

sortServerTotalWeight:
   // Sort all servers in ascending order of total weight.
   // Map's key is SID,
   // Map's value is total weight.

setChampionServer:
   // According to the sorted list, select the server with index = 0,
   // This server is configured closest to the job requirements.
```

7. Performance Comparison

Through the following 4 comparison tables and charts, we can find that the performance of Avg Exec Time parameters has been improved in 7 configuration files, and the performance of Actual Simulation End Time parameters in 3 configuration files has also been improved, especially ds-config-s2-4.xml. The performance of the Actual Simulation End Time parameter is improved by 21.95% compared to the other three algorithms.

For more performance details, please find link from references.

Config File	Parameter	First-Fit	Best-Fit	Worst-Fit	My Client	Performance Comparison
config_simple2	Avg Exec Time(seconds)	19286	19578	20333	19190	+0.50%
config_simple4	Avg Exec Time(seconds)	3005	3054	3100	2990	+0.50%
config_simple5	Avg Exec Time(seconds)	1325	1353	1390	1321	+0.30%
ds-config-s2-3	Avg Exec Time(seconds)	18869	19183	19957	18800	+0.37%
ds-config-s2-4	Avg Exec Time(seconds)	1433	1435	1461	1428	+0.35%
ds-config-s3-6	Avg Exec Time(seconds)	763	765	774	762	+0.13%
ds-config-s3-7	Avg Exec Time(seconds)	1299	1301	1323	1295	+0.31%

Figure 2 Avg Exec Time Table

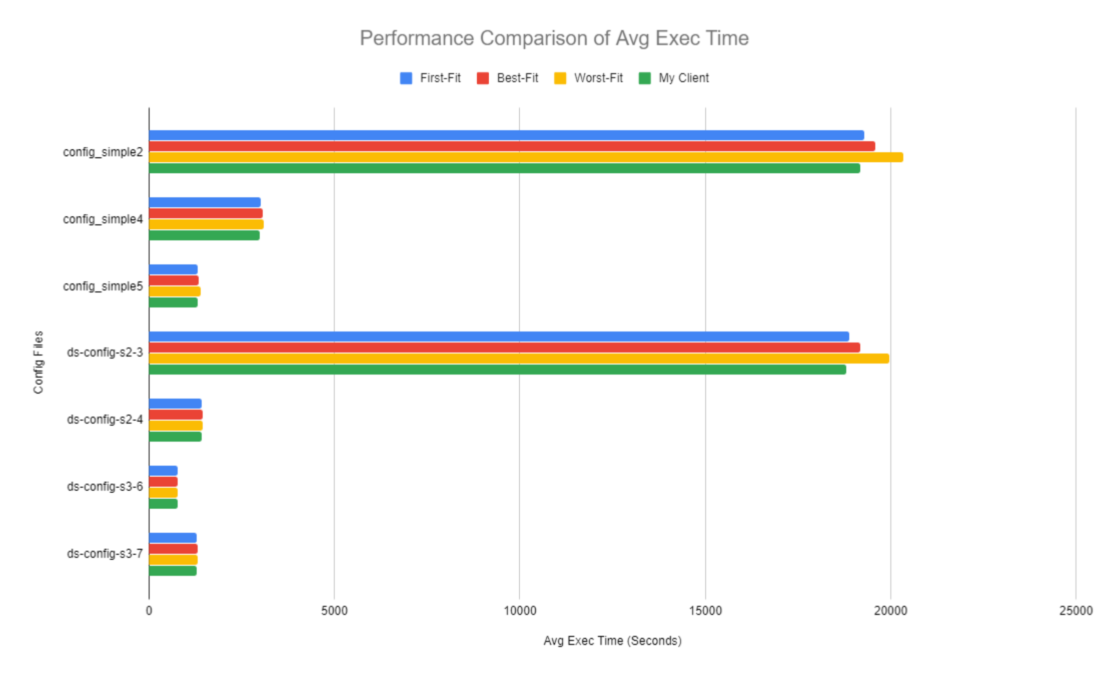

Figure 3 Avg Exec Time Chart

Config File	Parameter	First-Fit	Best-Fit	Worst-Fit	My Client	Performance Comparison
config_simple2	Actual Simulation End Time(seconds)	1007344	1007344	1007344	1007344	
config_simple4	Actual Simulation End Time(seconds)	159413	159413	159413	159413	
config_simple5	Actual Simulation End Time(seconds)	70295	90197	214854	70295	
ds-config-s2-3	Actual Simulation End Time(seconds)	1007344	1007344	1007344	1007344	
ds-config-s2-4	Actual Simulation End Time(seconds)	413014	439314	308012	240400	+21.95%
ds-config-s3-6	Actual Simulation End Time(seconds)	189104	193600	224878	188946	+0.08%
ds-config-s3-7	Actual Simulation End Time(seconds)	299315	299651	595065	295981	+1.11%

Figure 4 Actual Simulation End Time Table

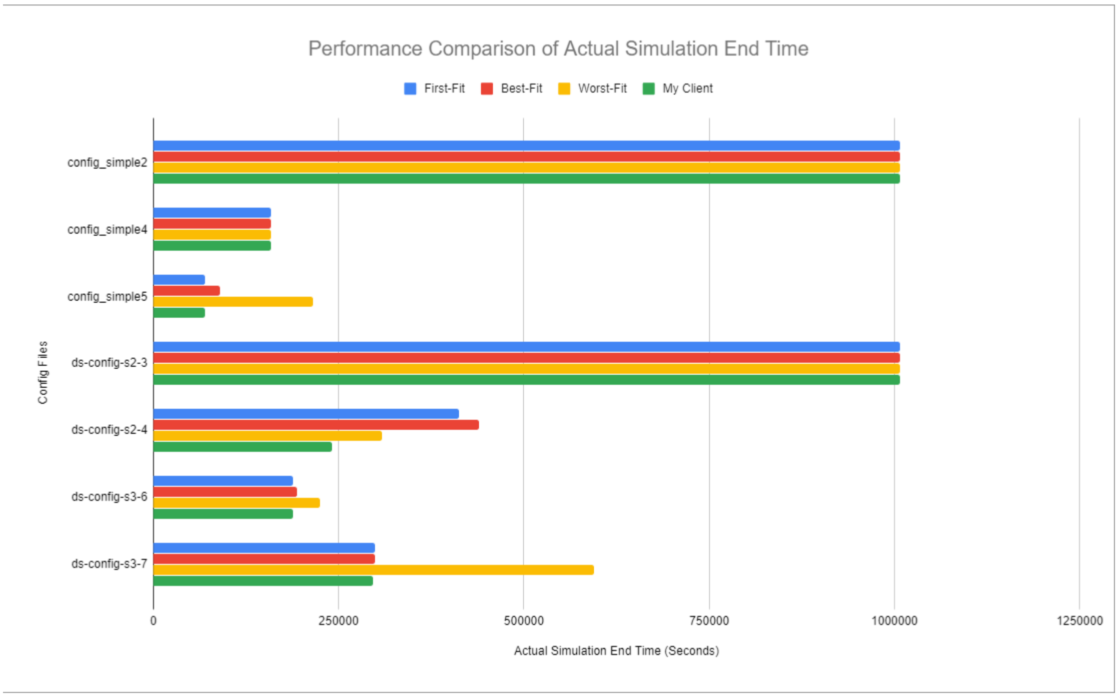

Figure 5 Actual Simulation End Time Chart

8. Conclusion

Through the theoretical analysis and actual performance test of the algorithm I designed above, I can confirm that this algorithm can indeed improve the performance of Avg Exec Time in the ds-sim model, and it will also improve the performance of Actual Simulation End Time on some test files. Because Actual Simulation End Time is not only affected by Avg Exec Time factors but also by Avg Turnaround Time and Avg Waiting Time. Not only this, while this algorithm could improve the performance of Avg Exec Time, it may also bring some negative effects to other performance parameters, such as Total Servers Used and Total Cost, because due to the need to find the most resource-saving server for each job, so this means that more servers may be used than other algorithms, and this often means that it will cost more to activate these more servers.

In short, I will also find a better algorithm to balance the advantages and disadvantages of various performance parameters in the subsequent version updates.

9. References

- GitHub: https://github.com/SnakeCN21/COMP3100-Group-Project/tree/S3_Snake
- Performance Testing: https://docs.google.com/spreadsheets/d/1Q_eK1kGGLV-dHFWf1JwVCwnY5B4QWS2_gAbOARYvSBs/edit?usp=sharing