<https://github.com/harshitgupta1337/fogsim/commit/02d3275c1f109e2f080ad31da9c0ed397b4c4b52?diff=unified>

or

<https://github.com/Cloudslab/iFogSim/blob/a61b684c86af4210cb06aa75597872094115f065/src/org/fog/placement/DistributedMicroservicePlacementLogic.java>

=========

package org.fog.placement;

import org.apache.commons.math3.util.Pair;

import org.fog.application.AppEdge;

import org.fog.application.AppModule;

import org.fog.application.Application;

import org.fog.entities.FogDevice;

import org.fog.entities.Tuple;

import org.fog.entities.ControllerComponent;

import org.fog.entities.MicroserviceFogDevice;

import org.fog.entities.PlacementRequest;

import org.fog.utils.Logger;

import org.fog.utils.ModuleLaunchConfig;

import java.util.\*;

/\*\*

\* Created by Samodha Pallewatta on 6/1/2021.

\* Per Placement Request Placement

\*/

public class DistributedMicroservicePlacementLogic implements MicroservicePlacementLogic {

/\*\*

\* Fog network related details

\*/

FogDevice fogDevice; //fog devices considered by FON for placements of requests

List<PlacementRequest> placementRequests; // requests to be processed

protected Map<Integer, Map<String, Double>> resourceAvailability;

private Map<String, Application> applicationInfo = new HashMap<>();

private Map<String, String> moduleToApp = new HashMap<>();

int fonID;

protected Double currentCpuLoad = 0.0;

protected List<String> currentModuleMap = new ArrayList<>();

protected Map<String, Double> currentModuleLoadMap = new HashMap<>();

protected Map<String, Integer> currentModuleInstanceNum = new HashMap<>();

protected Map<PlacementRequest, Integer> prStatus = new HashMap<>();

public DistributedMicroservicePlacementLogic(int fonID) {

setFONId(fonID);

}

public void setFONId(int id) {

fonID = id;

}

public int getFonID() {

return fonID;

}

@Override

public PlacementLogicOutput run(List<FogDevice> fogDevices, Map<String, Application> applicationInfo, Map<Integer, Map<String, Double>> resourceAvailability, List<PlacementRequest> prs) {

this.fogDevice = fogDevices.get(0); // only consists of current device

this.placementRequests = prs;

this.resourceAvailability = resourceAvailability;

this.applicationInfo = applicationInfo;

mapModules();

PlacementLogicOutput placement = generatePlacementMap();

updateResources(resourceAvailability);

postProcessing();

return placement;

}

@Override

public void updateResources(Map<Integer, Map<String, Double>> resourceAvailability) {

int deviceId = fogDevice.getId();

for (String moduleName : currentModuleInstanceNum.keySet()) {

Application app = applicationInfo.get(moduleToApp.get(moduleName));

AppModule module = app.getModuleByName(moduleName);

double mips = resourceAvailability.get(deviceId).get(ControllerComponent.CPU) - (module.getMips() \* currentModuleInstanceNum.get(moduleName));

resourceAvailability.get(deviceId).put(ControllerComponent.CPU, mips);

}

}

@Override

public void postProcessing() {

}

private PlacementLogicOutput generatePlacementMap() {

Map<Integer, Map<String, Integer>> placement = new HashMap<>();

for (PlacementRequest placementRequest : placementRequests) {

placement.put(placementRequest.getPlacementRequestId(), placementRequest.getPlacedMicroservices());

}

Map<Integer, Map<Application, List<ModuleLaunchConfig>>> perDevice = new HashMap<>();

Map<Integer, List<Pair<String, Integer>>> serviceDiscoveryInfo = new HashMap<>();

if (placement != null) {

for (int prID : placement.keySet()) {

//retrieve application

PlacementRequest placementRequest = null;

for (PlacementRequest pr : placementRequests) {

if (pr.getPlacementRequestId() == prID)

placementRequest = pr;

}

Application application = applicationInfo.get(placementRequest.getApplicationId());

for (String microserviceName : placement.get(prID).keySet()) {

int deviceID = placement.get(prID).get(microserviceName);

//service discovery info propagation

List<Integer> clientDevices = getClientServiceNodeIds(application, microserviceName, placementRequest.getPlacedMicroservices(), placement.get(prID));

for (int clientDevice : clientDevices) {

if (serviceDiscoveryInfo.containsKey(clientDevice))

serviceDiscoveryInfo.get(clientDevice).add(new Pair<>(microserviceName, deviceID));

else {

List<Pair<String, Integer>> s = new ArrayList<>();

s.add(new Pair<>(microserviceName, deviceID));

serviceDiscoveryInfo.put(clientDevice, s);

}

}

}

}

int deviceId = fogDevice.getId();

for (String microservice : currentModuleInstanceNum.keySet()) {

Application application = applicationInfo.get(moduleToApp.get(microservice));

AppModule appModule = new AppModule(application.getModuleByName(microservice));

ModuleLaunchConfig moduleLaunchConfig = new ModuleLaunchConfig(appModule, currentModuleInstanceNum.get(microservice));

if (perDevice.keySet().contains(deviceId)) {

if (perDevice.get(deviceId).containsKey(application)) {

perDevice.get(deviceId).get(application).add(moduleLaunchConfig);

} else {

List<ModuleLaunchConfig> l = new ArrayList<>();

l.add(moduleLaunchConfig);

perDevice.get(deviceId).put(application, l);

}

} else {

List<ModuleLaunchConfig> l = new ArrayList<>();

l.add(moduleLaunchConfig);

HashMap<Application, List<ModuleLaunchConfig>> m = new HashMap<>();

m.put(application, l);

perDevice.put(deviceId, m);

}

}

}

return new PlacementLogicOutput(perDevice, serviceDiscoveryInfo, prStatus);

}

public void mapModules() {

for (PlacementRequest placementRequest : placementRequests) {

Application app = applicationInfo.get(placementRequest.getApplicationId());

List<String> failedMicroservices = new ArrayList<>();

List<String> modulesToPlace = getMicroservicesToPlace(app, placementRequest.getPlacedMicroservices(), failedMicroservices, fogDevice.getName());

while (!modulesToPlace.isEmpty()) {

for (String microservice : modulesToPlace) {

//try to place or add to failed list, add to mapped modules

if (app.getSpecialPlacementInfo().containsKey(microservice) &&

!app.getSpecialPlacementInfo().get(microservice).contains(fogDevice.getName())) {

failedMicroservices.add(microservice);

} else if (getModule(microservice, app).getMips() + currentCpuLoad <= resourceAvailability.get(fogDevice.getId()).get(ControllerComponent.CPU)) {

Logger.debug("ModulePlacementEdgeward", "Placement of operator " + microservice + " on device " + fogDevice.getName() + " successful.");

currentCpuLoad = getModule(microservice, app).getMips() + currentCpuLoad;

System.out.println("Placement of operator " + microservice + " on device " + fogDevice.getName() + " successful.");

moduleToApp.put(microservice, app.getAppId());

if (!currentModuleMap.contains(microservice))

currentModuleMap.add(microservice);

placementRequest.getPlacedMicroservices().put(microservice, fogDevice.getId());

//currentModuleLoad

if (!currentModuleLoadMap.containsKey(microservice))

currentModuleLoadMap.put(microservice, getModule(microservice, app).getMips());

else

currentModuleLoadMap.put(microservice, getModule(microservice, app).getMips() + currentModuleLoadMap.get(microservice));

//currentModuleInstance

if (!currentModuleInstanceNum.containsKey(microservice))

currentModuleInstanceNum.put(microservice, 1);

else

currentModuleInstanceNum.put(microservice, currentModuleInstanceNum.get(microservice) + 1);

} else {

failedMicroservices.add(microservice);

}

}

modulesToPlace = getMicroservicesToPlace(app, placementRequest.getPlacedMicroservices(), failedMicroservices, fogDevice.getName());

}

if (!failedMicroservices.isEmpty()) {

//check for cluster placement or send to parent

if (((MicroserviceFogDevice) fogDevice).getIsInCluster()) {

int deviceId = placeWithinCluster(failedMicroservices, app);

if (deviceId != -1)

prStatus.put(placementRequest, deviceId);

else

prStatus.put(placementRequest, fogDevice.getParentId());

} else {

prStatus.put(placementRequest, fogDevice.getParentId());

}

} else if (allModulesPlaced(app, placementRequest)) {

//all modules placed

prStatus.put(placementRequest, -1);

} else {

//specially mapped modules left

prStatus.put(placementRequest, fogDevice.getParentId());

}

}

}

private boolean allModulesPlaced(Application app, PlacementRequest placementRequest) {

List<String> microservicesToPlace = new LinkedList<>();

for (AppModule module : app.getModules()) {

if (!placementRequest.getPlacedMicroservices().keySet().contains(module.getName())) {

microservicesToPlace.add(module.getName());

return false;

}

}

return true;

}

private int placeWithinCluster(List<String> failedMicroservices, Application app) {

List<Integer> clusterDeviceIds = ((MicroserviceFogDevice) fogDevice).getClusterMembers();

if (clusterDeviceIds.isEmpty())

return -1;

List<Integer> sortedClusterDevices = new ArrayList<>();

for (Integer id : clusterDeviceIds) {

//sort list from min to max

if (sortedClusterDevices.isEmpty())

sortedClusterDevices.add(id);

else {

boolean isPlaced = false;

for (int i = 0; i < sortedClusterDevices.size(); i++) {

double sorted = resourceAvailability.get(sortedClusterDevices.get(i)).get("cpu");

double current = resourceAvailability.get(id).get("cpu");

if (sorted < current) {

continue;

} else {

sortedClusterDevices.add(i, id);

isPlaced = true;

break;

}

}

if (!isPlaced)

sortedClusterDevices.add(id);

}

}

List<AppModule> sortedMicroservices = new ArrayList<>();

for (String microservice : failedMicroservices) {

//sort list from min to max

AppModule appModule = getModule(microservice, app);

if (sortedMicroservices.isEmpty())

sortedMicroservices.add(appModule);

else {

boolean isPlaced = false;

for (int i = 0; i < sortedMicroservices.size(); i++) {

double sorted = sortedMicroservices.get(i).getMips();

double current = appModule.getMips();

if (sorted < current) {

continue;

} else {

sortedMicroservices.add(i, appModule);

isPlaced = true;

break;

}

}

if (!isPlaced)

sortedMicroservices.add(appModule);

}

}

double cpuMax = resourceAvailability.get(sortedClusterDevices.get(sortedClusterDevices.size() - 1)).get("cpu");

if (cpuMax >= sortedMicroservices.get(0).getMips()) {

return sortedClusterDevices.get(sortedClusterDevices.size() - 1);

} else {

return -1;

}

}

private List<String> getMicroservicesToPlace(Application app, Map<String, Integer> placedMicroservices, List<String> m\_failed, String deviceName) {

List<String> failed = new ArrayList<>();

failed.addAll(m\_failed);

for (AppModule module : app.getModules()) {

if (app.getSpecialPlacementInfo().containsKey(module.getName()) && !app.getSpecialPlacementInfo().get(module.getName()).contains(deviceName)

&& !m\_failed.contains(module.getName())) {

failed.add(module.getName());

}

}

return app.getDAG().getSources(new ArrayList<>(placedMicroservices.keySet()), failed);

}

private AppModule getModule(String moduleName, Application app) {

for (AppModule appModule : app.getModules()) {

if (appModule.getName().equals(moduleName))

return appModule;

}

return null;

}

public List<Integer> getClientServiceNodeIds(Application application, String

microservice, Map<String, Integer> placed, Map<String, Integer> placementPerPr) {

List<String> clientServices = getClientServices(application, microservice);

List<Integer> nodeIDs = new LinkedList<>();

for (String clientService : clientServices) {

if (placed.get(clientService) != null)

nodeIDs.add(placed.get(clientService));

else if (placementPerPr.get(clientService) != null)

nodeIDs.add(placementPerPr.get(clientService));

}

return nodeIDs;

}

public List<String> getClientServices(Application application, String microservice) {

List<String> clientServices = new LinkedList<>();

for (AppEdge edge : application.getEdges()) {

if (edge.getDestination().equals(microservice) && edge.getDirection() == Tuple.UP)

clientServices.add(edge.getSource());

}

return clientServices;

}

}