CrowdOS Algorithm Library

Existing Algorithms in CrowdOS

There are four task allocation algorithms currently implemented in the CrowdOS kernel: T_Most, PT_Most, T_Random, and GGA_I. The specific code can be found in the algorithms section of the kernel code:

CrowdOS\kernel\src\main\java\cn\crowdos\kernel\algorithms

The default algorithm used in the kernel is "DefaultAlgo", which is a simple default implementation. To use the four classic task allocation algorithms mentioned above, the "algoSelect" function needs to be used for algorithm selection.

Interfaces in the Algorithm Library

The algorithms package in the kernel defines the crowd-sourcing related algorithms used in the system. Currently, interfaces for task allocation, task recommendation, and participant selection algorithms are provided, which are TaskAssignmentAlgo, TaskRecommendationAlgo, and ParticipantSelectionAlgo respectively. The algorithms package uses the factory pattern, with each algorithm factory producing a specific type of algorithm implementation. Each algorithm factory provides the above three algorithms.

The AlgoFactory interface defines all the algorithms used in the kernel. Currently, it defines three functions:

Return Value	Prototype	Meaning
TaskAssignmentAlgo	getTaskAssignmentAlgo();	Return the task assignment algorithm
TaskRecommendationAlgo	getTaskRecommendationAlgo();	Return the task recommendation list algorithm
ParticipantSelectionAlgo	getParticipantSelectionAlgo();	Return the participant selection algorithm

The definitions of TaskAssignmentAlgo, TaskRecommendationAlgo, and ParticipantSelectionAlgo are similar. Taking TaskAssignmentAlgo as an example, the TaskAssignmentAlgo interface defines the functional interface of the task allocation algorithm, which includes single task allocation and multiple task allocation:

Return Value	Prototype	Meaning
List<Participant>	getTaskAssignmentScheme(Task task)	Get the list of participants assigned to a specific task.
List<List<Participant>>	getTaskAssignmentScheme(List<Task> taskList)	Return a list of lists of participants assigned to each task in the input list. Each sublist corresponds to one task.

Integration of Algorithms

The algorithms package in the kernel uses the factory pattern, with each factory producing a specific type of algorithm implementation. Therefore, when integrating new algorithms into the kernel, a new algorithm factory needs to be created. The AlgoFactoryAdapter is provided in the kernel, which provides basic implementations of task allocation, task recommendation, and participant selection. Therefore, the newly created algorithm factory can be implemented by inheriting from AlgoFactoryAdapter. Taking the PT_Most task allocation algorithm as an example, the system resources at this time are obtained first:

Then, the task allocation algorithm is overridden. There are single task allocation algorithms and multiple task allocation algorithms. When adding a new algorithm, the system may provide an implementation of the multiple task allocation algorithm, but not the single task allocation algorithm. Therefore, when participants integrate a new algorithm, they must provide at least the implementation of single task allocation. Taking the single task allocation as an example, the resources required from the resource pool are obtained:

```
1 usage  LHY
public class PTMostFactory extends AlgoFactoryAdapter {

    1 usage  LHY
    public PTMostFactory(SystemResourceCollection resourceCollection) {
        super(resourceCollection);
    }
```

Then, the parameters required by the new algorithm are calculated in advance:

```
ParticipantPool participants = resourceCollection.getResourceHandler(ParticipantPool.class).getResource

int taskNum = 1;

List<Constraint> taskLocation = task.constraints().stream()
        .filter(constraint -> constraint instanceof POIConstraint)
        .collect(Collectors.toList());
if (taskLocation.size() != 1) {
    return null;
}
List<Participant> candidate = participants.stream()
        .filter(task::canAssignTo)
        .collect(Collectors.toList());
int workerNum = candidate.size();
```

Then, the prepared algorithm instance is created, and the task allocation results are obtained with the input parameters:

```
        List<Participant> candidate = participants.stream()
                .filter(task::canAssignTo)
                .collect(Collectors.toList());
        int workerNum = candidate.size();


        Coordinate tLocation = (Coordinate) taskLocation.get(0);
        double[][] distanceMatrix = new double[workerNum][taskNum];
        for (int i = 0; i < candidate.size(); i++) {
            Participant worker = candidate.get(i);
            Coordinate wLocation = (Coordinate) worker.getAbility(Coordinate.class);
            distanceMatrix[i][0] = wLocation.euclideanDistance(tLocation);
        }

        double[][] taskDistanceMatrix = new double[][]{{1}};
```

Finally, register the newly integrated algorithm in the kernel to complete the algorithm integration.

```
public void initial(Object...args){
    systemResourceCollection = new SystemResourceCollection();
    try {
        systemResourceCollection.register(new TaskPool());
        systemResourceCollection.register(new ParticipantPool());
        systemResourceCollection.register(new AlgoContainer(new AlgoFactoryAdapter(systemResourceCollection)), resourceName: "DefaultAlgo");
        systemResourceCollection.register(new Scheduler(systemResourceCollection));
        systemResourceCollection.register(new MissionHistory());
        systemResourceCollection.register(new TrustBasedIncentiveImpl());
        systemResourceCollection.register(new AlgoContainer(new PTMostFactory(systemResourceCollection)), resourceName: "PTMost");
        systemResourceCollection.register(new AlgoContainer(new T_MostFactory(systemResourceCollection)), resourceName: "T_Most");
        systemResourceCollection.register(new AlgoContainer(new T_RandomFactory(systemResourceCollection)), resourceName: "T_Random");
        systemResourceCollection.register(new AlgoContainer(new GGA_IFactory(systemResourceCollection)), resourceName: "GGA_I");
    } catch (DuplicateResourceNameException e) {
        throw new RuntimeException(e);
    }
    initialed = true;
}
```

Using Algorithms in the Kernel

To use the newly added algorithm in the kernel, the newly added algorithm factory needs to be registered during the initialization of the kernel and given a unique and representative name. In this way, the algorithm factory is added to the system resources:

```
systemResourceCollection.register(new AlgoContainer(new PTMostFactory(systemResourceCollection)), resourceName: "PTMost");
```

The algorithm selection function "algoSelect(String name)" is provided in the kernel, which can be used to select the desired algorithm. It selects the algorithm factory in the system scheduler and then selects the desired algorithm