TDT4280 - Answers to Exercise 3

Alexandre Balon-Perin and Stian Kristoffersen

March 4, 2012

Abstract

In this report we describe our design choices and implementation of a simple distributed computing network using JADE, which proved to be a useful framework for FIPA ACL communication protocol. Parallelizing the task decomposition and assignment would be an interesting place to start for future work. Information on how to use the system is presented in Section 2.1.

1 Design Choices

1.1 Tasks Decomposition - LR(0) Parser

We chose to use Reverse Polish Notation (postfix) for the arithmetic expression, because you always have the operands available at the stack when you encounter an operator, making the reduction simple. To parse the expression and build an Abstract Syntax Tree (AST) we use an LR(0) parser. LR(0) was chosen because of it is simple to implement. By using a Depth First Traversal of the AST, we can compute each sub-expression in the right order.

1.2 Auction type - First Price Sealed Bid

We chose the FPSB auction type because of its simplicity and it makes sense in the setting of distributed computing; each computer has a fixed cost related to the speed of the hardware. By taking into account the cost of the current tasks, the computer that will finish the new task first is chosen, to minimize the wall clock time. It would not make sense to refine the cost, because the cost is fixed (due to hardware.)

1.3 Class-level design

The principal classes are: TaskAdministrator, ComputeAgent and Parser.

1.3.1 TaskAdministrator

The TaskAdministrator class has the duty to coordinate the operations between agents. It receives the expression from the user by the TaskAdministratorGui, create an object of Parser in order to parse it and delegates each sub-task among the solver agents. The final part is accomplished by a behaviour called RequestPerformer. The action() function is composed of a switch statement where each case is a step towards the completion of a sub-task.

- **Case 0:** focuses on the retrieval of the sub-task to be sent to the solver and on the discovery of the appropriate solver through the Directory Facilitator.
- **Case 1:** is in charge of sending the Call For Proposal message to the agents selected in Case 0.
- **Case 2:** In this step, the TaskAdministrator processes the proposals received by the solvers and determines which one is the best.

- **Case 3:** The accept-proposal message is sent to the winner of the award with the sub-task to be computed as the content.
- **Case 4:** Finally, the TaskAdministrator receives the result of the sub-task or a failure message indicating that an error has occurred during the computation (i.e. a division by zero). If the computation went well, there are two possibilities. Either the result is the final result of the expression entered by the user. In this case, the TaskAdministrator waits for a new job. Or the current_node is not null, meaning that there are more sub-task to be performed. The step variable is set to zero again and the switch starts from the beginning.

1.3.2 Parser & Node

The Parser class is in charge of parsing the expression entered by the user. It disposes of a stack which is an ArrayList of Nodes. Each Node has a left and a right leaf as well as an operator and a value. This makes it easy to transform an operator by the result of the operation computed. When a Node is a leaf its left and right leaves are set to null. As said before, the function parseExpression(String expression) of the Parser checks that the expression is correctly formed and builds the AST at the same time. At the end of the parsing, only the root node is left on the stack.

The function getNextNode() of the Parser is used to get the next atomic operation to be sent to the appropriate solver. Since the stack still contains the root node, it is possible to go through the AST to find the next node where children are values.

1.3.3 ComputeAgent

This abstract class is the superclass of all type of solvers. In the setup() function, the agent register itself to the DirectoryFacilitator with the type defined in the solver subclass.

After that, the behaviour OfferRequestServer is triggered in order to receive possible Call For Proposal messages.

Finally, if the agent is awarded with the task the ComputeTaskServer behaviour is triggered to compute the task required. The compute(String operation) function called in this behaviour is an abstract function that must be implemented in the solver subclasses. Polymorphism allows us to write the minimum code possible for each type of solvers.

1.3.4 Solvers

Four solvers have been implemented: an AdditionSolver, a SubstractionSolver, a MultiplicationSolver and a DivisionSolver. Each of these are very simple subclasses of ComputeAgent. The type of the solver ("compute+", "compute-", "compute/" or "compute*") is set in the constructor and sent to the superclass in order to register to the DirectoryFacilitator. The compute(String operation) function is then implemented. Firstly, it gets the operands from the message content which is in the form "<int><operator><int>". The function getOperands(String operation) is in charge of this and can be found in the ComputeAgent class. Secondly, the arithmetic expression is calculated.

2 Implementation

2.1 How to Run

Our JADE program can be run with the run.sh script or with the following command:

```
java -cp lib/jade.jar:classes jade.Boot -gui -host 127.0.0.1 -agents
"TaskAdministrator:BalonPerinKristoffersen.
DistributedSolver.TaskAdministrator;
compute01:BalonPerinKristoffersen.DistributedSolver.AdditionSolver;
compute02:BalonPerinKristoffersen.DistributedSolver.AdditionSolver;
compute03:BalonPerinKristoffersen.DistributedSolver.SubstractionSolver;
compute04:BalonPerinKristoffersen.DistributedSolver.SubstractionSolver;
compute05:BalonPerinKristoffersen.DistributedSolver.DivisionSolver;
compute06:BalonPerinKristoffersen.DistributedSolver.DivisionSolver;
compute07:BalonPerinKristoffersen.DistributedSolver.MultiplicationSolver;
compute08:BalonPerinKristoffersen.DistributedSolver.MultiplicationSolver;"
```

This will launch one Task Administrator, one Directory Facilitator and eight Compute Agents, two of each each type. Our test problem, shown in Figure 1, can be run by entering

```
23 2+5*7 4/-
```

in the GUI of the Task Administrator, and hitting "Compute!".

2.2 System Decomposition

A screenshot of the Sequence diagram composed by the JADE sniffer agent is shown in Figure 1. The same eight steps are repeated 4 times, one for each of the operators. The eight steps are described below.

Initialization

The expression received from the GUI is parsed by our LR(0) parser and an AST is built. A behaviour is then triggered to perform the following steps.

Step 1: Yellow-pages Lookup

The next sub-expression is found and the Task Administrator looks for suitable Solvers by contacting the Directory Facilitator.

Step 2: Yellow-pages Reply

Directory Facilitator replies with the list of appropriate solvers that are available.

Step 3,4: Call for Proposals

The Task Managers sends out a Call For Proposals to the appropriate agents (there are two agents for each operator).

Step 5,6: Bids

Each of the agents receiving the CFP computes a price based on the queue of existing tasks plus a random price for the new task. The total cost is placed in a bid and sent to the Task Manager.

Step 7: Selection and Award

After selecting the cheapest solver the Task Manager sends the Accept Proposal Message to the winner.

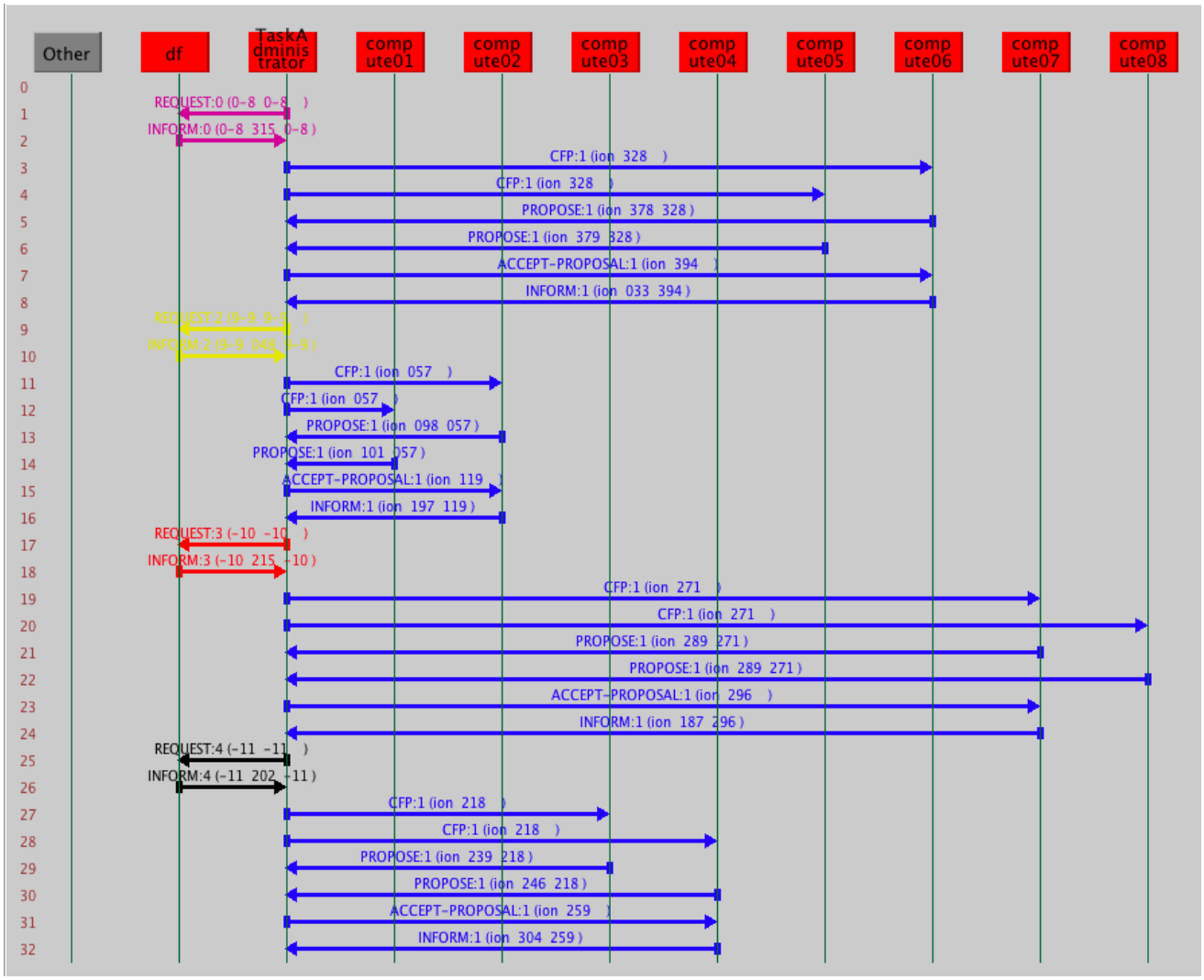

Figure 1: Sequence Diagram

Step 8: Sub-task completion

The solver that was awarded the task adds a behaviour that will trigger after a number of milliseconds defined by the cost. Then the result will be computed and communicated to the Task Administrator.

Finalization

The node in the AST containing the current operator is set to be the result of the current computation. Its children are set to null in order to indicate that this is now a leaf node. The parent operator-node can then use it in future computations. If the current node was the root node of the AST, we are done, otherwise go to step 1.

3 Possible Improvements

3.1 Optimal Task Administrator Strategy

Assumptions

We assume that there is only one Task Administrator with complete knowledge of the system, or that it can behave in this way. This can be achieved by querying all agents of their current states, find the optimal solution and submit jobs (the operations are known but the data needs to be sent later) to all agents before anyone else can submit any more jobs (race conditions not considered.)

Outline of the Optimal Strategy

The problem reduces to finding the shortest critical path in the AST, by selecting appropriate agents for all computations a priori. Each agent can of course only work on one sub-problem at one point in time. Sub-branches of the AST should also be divided to avoid excessive serial work.