Remy Robotics - Pizza cooking robots

Abstract

The purpose of this document is to describe the components and behaviour of an hypothetical system for pizza cooking robots.
The initial installation of the facility includes:

- 2 pizza ovens stacked one above the other
- 2 robots before the ovens, each is able to:
    - Spread tomato sauce on a pizza crust
    - Scatter cheese over tomato sauce
    - Place pizza in one of the ovens
- 2 robots after the ovens, each is able to:
    - Pick pizza from one of the ovens
    - Slice it into pieces
    - Pack pizza into the box

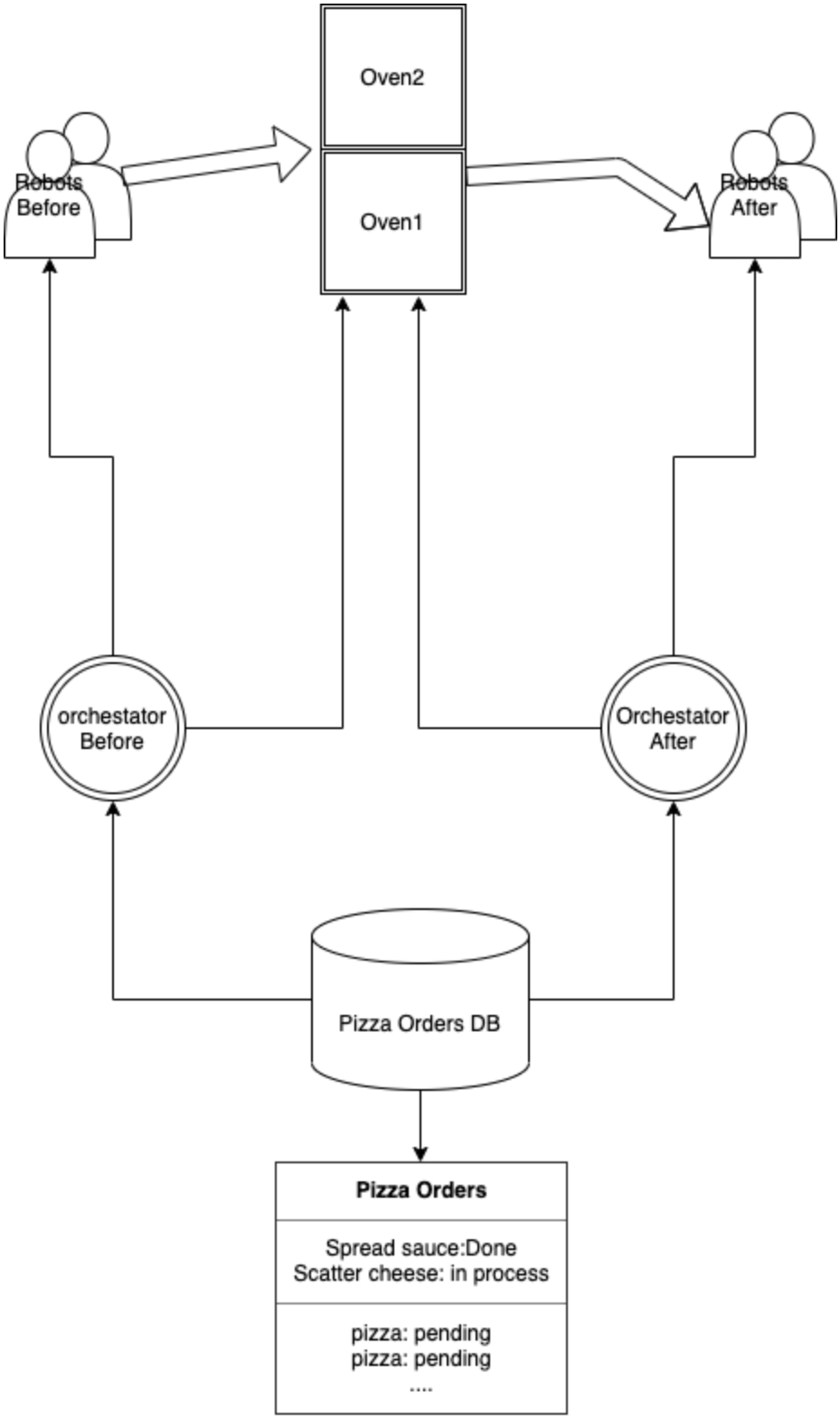
Preconditions

For the sake of a simpler solution I will assume the following preconditions:

1. All robots are equal only difference is tools available and positions
2. When a robot starts an operation a pizza/pizza crust in previous state will be available in a designated place
3. When a robot ends an operation will place the resulting pizza/pizza crust in the designated place for the next stage to pick it
4. Ovens will start when you put a pizza in them and will stop when pizza is done
5. Ovens can only cook 1 pizza at a time
6. Robots can fail when a task is assigned (i.e scatter cheese) this may be related to no pizza crust available in the designated place or no more cheese in the expected place. Retrying later will probably solve the issue so order is intact.
7. Robots can also fail when doing a task, when this happens that pizza is lost
8. Robots after a failure are immediately ready for the next operation without penalty.
9. Robots can do different task one after the other with no penalty
10. Any alert generated by any part of the system (no more cheese, robot offline, oven don't start) will be attended by someone/something outside the scope of this document
11. All communications with robots and ovens will be done using a REST API.

10.000 ft. view of the system

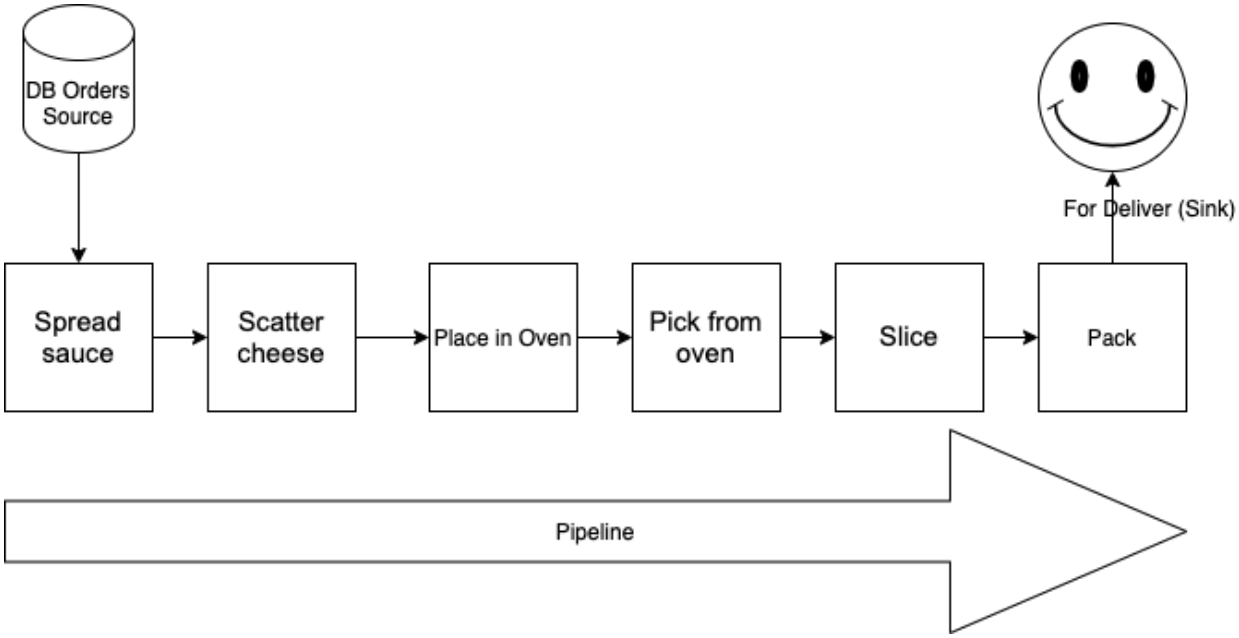
The system will be built around a Relational Database like PostgreSQL and 2 orchestrators (processes) in charge of dispatching orders to robots and cross monitor each other.



Taking advantage of the separation of robot's roles in "Before" and "After" oven I'm also adding 2 orchestrators that will check pizza orders and send the corresponding task to the robots under

its commandment. Doing this while looking at pizza order from db can help them monitor the other orchestrator in an indirect way (More on this later on)

The main idea behind this system is seen the process of building a pizza as pipeline or a stream where pizza orders are the source of events, each task performed on a pizza is a transformation and the sink is the boxed pizza ready for delivery



Components

Orchestrator “Before”:

Orchestrator before is in charge of getting the pizza crust through every step of the pipeline upto any of the available[1] ovens. For doing this will use “Before” Robots keeping record of what each robot is doing to know the current step a pizza is on.

Orchestrator “After”:

Orchestrator after is in charge of getting the pizza from the oven through every step of the pipeline upto packed for delivery. For doing this will use “After” Robots keeping record of what each robot is doing to know the current step a pizza is on.

[1] Available oven is an idle and empty oven. As we are assuming ovens are almost autonomous we put a pizza on it and they start cooking until pizza is done. At that time the oven becomes idle but not empty.

Robot “Before” or “After”:

Robots can perform a bunch of predefined tasks on a pizza. Depending on their initial configuration the group of tasks they can do.
They provide a simple API so the Orchestrator can use them to prepare pizzas.
Api provides 1 method to start doing a task and 1 method to see if the latest task is in progress, succeeded or failed.

Oven:

Ovens can cook a pizza and are almost autonomous. After a pizza it’s placed in the oven it will cook it until done and will stop automatically after that.
Ovens provide a single api to know it’s status. Oven status can be: Cooking, Pizza done/idle, Empty

Pizza order DB:

Pizza order DB keeps track of current orders and is used from the outside to place orders and from the orchestrators to keep track of pizza order progress.

How cross validation between orchestrators works

The main idea of this cross validation is that each orchestrator with little or no extra information can know if the opposite half of the pipeline is stalled or slow. This by no means remove the need of an external monitoring system that can query each component and can trigger alerts in case something is unreachable or broken (like Prometheus + AlertManager) but in case the other half of the pipeline is misbehaving the Orchestrator can be aware and trigger an alarm or do some something else.

Orchestrator Before:

Can detect that Orchestrator “After” is stalled if all ovens are idle but not empty. Which means nobody is removing pizzas from the other side. So if all ovens are in idle state after X amount of time Orchestrator before can trigger the Orchestrator “After” stalled. On a similar approach as Both orchestrators have access to DB, Orchestrator “Before” can check how many orders have reached upto the oven in the last Y minutes and see how many have been completely fulfilled. Y completely fulfilled orders are consistently lower; it may be an indicator that the “After” part of the pipeline is slow or is ruining many orders that need to be restarted.

Orchestrator After:

Can detect that Orchestrator “Before” is stalled if all ovens are empty and there are many old pending orders on the Database. If this happens it means that the “Before” Orchestrator is not operating or can’t get any order up to the ovens. If that’s the case, trigger the Orchestrator “Before” stalled alarm.
If the number of pending orders over a period of time is constantly higher than new orders in the oven in the same period of time it may be an indicator of slowness in the “Before” part of the pipeline or that “Before” part is ruining many orders and needs to be restarted.

How the Orchestrator gets its work done

Orchestrator works as a simple reactor, handling events as it needs to. For this Orchestrator needs to be configured with the addresses of the ovens, all robots under its control and the Database that keeps track of the orders.
A reactor loop will look like similar to this:

1. For each Robot:
    a. If robot has finished a task update order status accordingly
    b. If robot is busy for more than a Y time trigger alert Robot not working, configure order to start again and remove it from pool
2. For each oven:
    a. Read oven status. Used for Top function below.
3. Get Top N Orders that need to be processed by this Orchestrator (Before or After oven) (where N is the number of free robots, Top is some function that prioritize which order need to be tackled next probably with the idea of finishing more pizzas instead of having more in progress)
4. For each free robot:
    a. Assign the next task and update status accordingly
5. Verify conditions that may indicate the other orchestrator is dead and act accordingly

Difference between Before and After Orchestrator flows

Both Orchestrators are very similar, they only have 2 differences that probably don’t alter the flow but just some code changes

1. What to check in step 5 to indicate other half of the orchestrator is not working properly
2. How to calculate Top Funcion. Difference here is related to how to operate the oven; the main purpose of the “Before” Orchestrator is to put pizzas in the oven (to give the second half of the pipeline something to do). So top priority there will be if an oven is empty, put a pizza there. On the other hand the top priority for the second half of the pipeline is to remove pizzas from the oven to avoid blocking the first half.

Q&A:

Q:What elements does your system have?

A: It has 4 or 6 items (depending if before and after are counted as 1 or 2 items) as shown in the 1st picture:

1. Robots (Before/After)
2. Ovens
3. Orchestrators (Before/After)
4. Orders DB

Q:What are the interactions between those elements?

A: it's mostly described in “How the Orchestrator gets its work done” but the idea is that all components interact with one or both of the Orchestrators through a REST API that supports executing an action and getting it’s status.

Q:What are the differences between cooking 1 pizza, 2 pizzas, and N pizzas?

A: There is not too much difference as the system works like a stream of pizzas having 1, 2 or N pizzas in the stream won’t make the system behave differently.
Having said that, the particular case of 1 pizza will have half of the components in an idle state. And more than 2 pizzas probably the bottleneck will be the ovens but as preconditions says that after a robot finishes a step it will have a place to put it’s resulting pizza without locking so worst case scenario this places work like unbounded buffers that will hold pizzas in pre-oven state and when an oven becomes free next task in the loop will be put one pizza on it.

Q:Why is it easy to add new robots into your system?

A: Orchestrators see robots as an array of processors that can perform any task available to them. To add more robots to the system it’s a matter of just adding more robots to that array as the Orchestrator tries to maximize Pizza output by keeping everyone busy.