MIDDLEWARE 1 & 2

Group Members

Iec2016027(Bhanu Bhandari)

Iec2016065(Harshit Srivastava)

Iec2016072(Ruchin Agrawal)

Iec2016012(Nikhil Mundra)

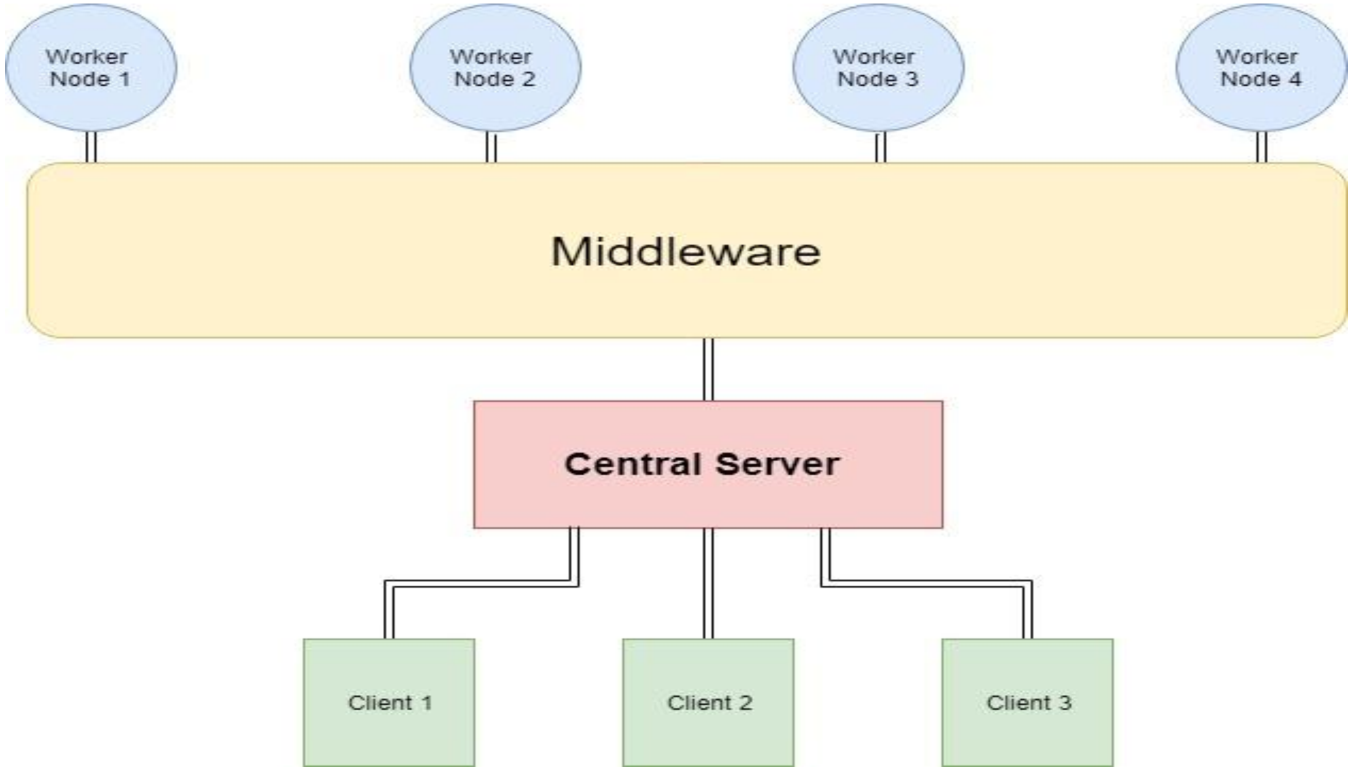
Iec2016026(Agam Dwivedi)

Iec2016038(Kuldeep Gautam)

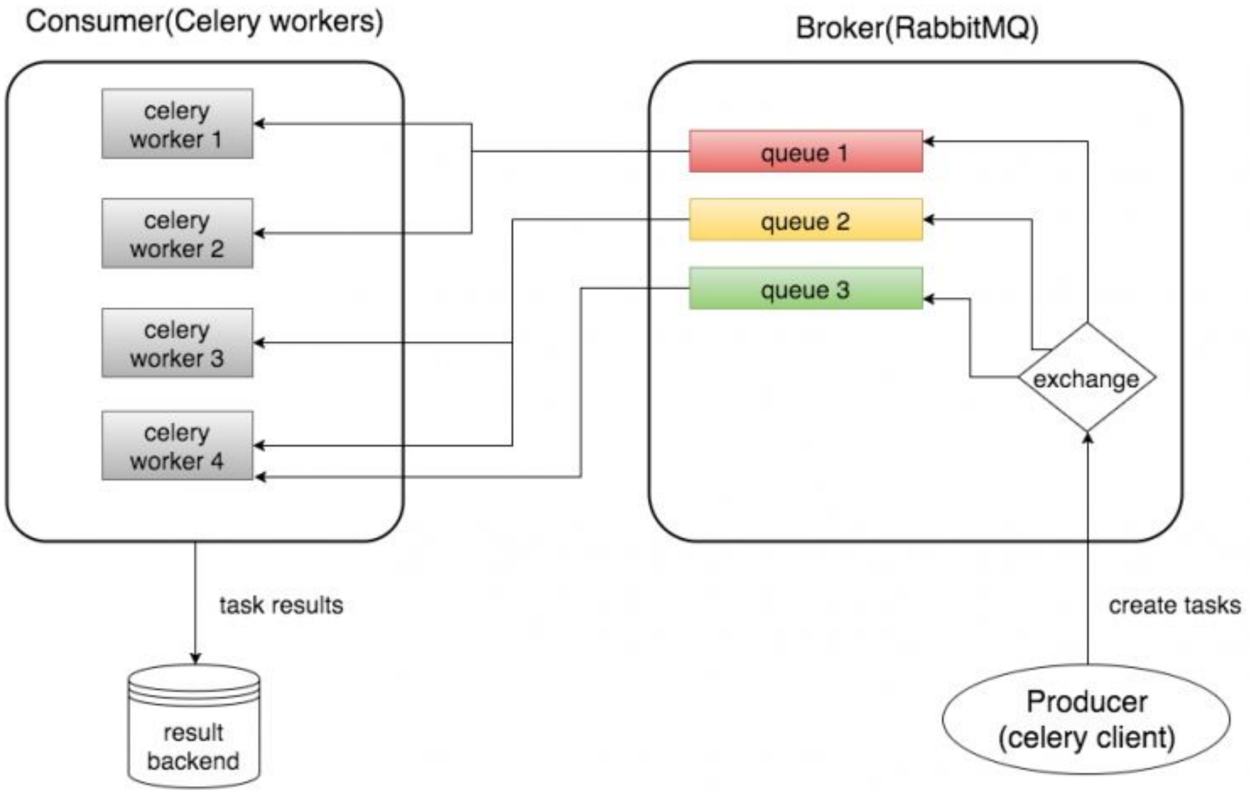
Tasks to be Performed

- Web based interface to submit workload to be run, input file and output file name.
- Results to go back in original machine
- Multiple machines, jobs can be submitted at any machine
- Load Balancing: Machines do distributed load monitoring/balancing
- Jobs are migrated before start to appropriate machine .
- Fault Tolerance: Node may fail, needs strategy to handle, monitor, re-run (if needed) running jobs.
- Node may recover and rejoin system.

Architecture



Technology


Consumer(Celery workers)
Broker(RabbitMQ)
celery worker 1
celery worker 2
celery worker 3
celery worker 4
queue 1
queue 2
queue 3
exchange
task results
result backend
create tasks
Producer (celery client)

Technology used

- Flask
- Python3
- Celery
- RabbitMQ
- OpenCV
- NumPy

Workload Implemented

- Arithmetic Operations (Addition & Multiplication)
- Image Processing

How do we ensure fault-tolerance in the system?

By checking liveness of the worker node

- RabbitMQ piggybacks **heartbeat message** along with the tasks message which is sent to the celery worker nodes.
- After receiving the message, task worker node sends the acknowledgment message back to the server.
- If the acknowledgment is successfully received then the task is deleted from the queue.
- If the worker node crashes before acknowledging the task, then the can be redelivered to another worker.

Fault Recovery

Whenever any soft-error occurs at any of the worker node then it is rebooted. Once it is back online it sends connection request to the Server and after the connection is established successfully worker node synchronises its queue with the task queue.

How do we ensure load balancing in the system?

Celery has a feature of Gossip in which it allows the worker nodes to exchange the individual performance related information (CPU Utilization, Throughput, Number CPU cores etc) among themselves. Now based on these information celery finds out the best worker node and reroutes the task to it. There might exist multiple task queues and for each queue the best worker in found out by celery.

Results

Two Machines:

Addition task: 8.423 seconds

Multiplication: 23.623 seconds

Three Machines.

Addition Task: 5.341 seconds

Multiplication: 16.42 seconds