First, let us start with the scripts required. If we develop two different scripts, one for calling the second script to process each payment alone and keeping track of the time required to finish the whole process, and if it exceeds the 5 minutes limit we throw a new exception or respond with HTTP 408 code to notify the tech support.

As for the other script, it should handle the steps required to handle each payment. And to handle payment was declined exception, we wrap it in a (try-catch-finally). The try block to check if the payment went through, if it didn't go through we add it on the database with the success attribute of 0 and we throw a new exception to catch it. In the catch block, we should first check if this payment was already checked twice (didCheckAgain) if not we change the variable to true and recheck, if yes we skip checking again. In the finally block, we check if the payment was accepted on the second try and update the success attribute to 1, or add it with the success attribute of 1 if no exceptions were thrown(we check using didCheckAgain).

Since the system is based on LAMP architecture with multiple servers, then for sure there is a load balancer that balances the requests to the different apache nodes available, and a lot of memory that we can work with.

That being said, and knowing that PHP is multi-threaded the payments won't go in sequential order. Each payment would have its thread.

Starting with the load balancer, it should be configured with a lbmethod that is Round Robin which is typically used for scheduling. Using this algorithm, we make sure that the process is balanced evenly between the servers. More configuration is needed to handle exceptions returned by the servers.

With the use of the load balancer, we can monitor how much the whole process took time to finalize and proxying different payments to different nodes.

The advantages of this solution are. The process is distributed between the servers, hence we are utilizing multiple servers. Handling exceptions and processes are being monitored in the load balancer-manager.

The disadvantages of this solution are. We are using the apache nodes to process the payments, hence we are consuming the server memory and increasing the load in each node which will lead to slower serving for clients. Having different requests to the online payment platform which I suppose is a 3rd party platform might be rejected due to the extensive HTTP requests (the requests might be flagged as DoS attack).

A Second Approach - not advisable:

With the use of the same script following the same logic, and since pThreads was introduced in PHP 7. The time needed for each payment to process is 2 seconds, the main reason would be contacting the online payment platform and waiting for the response.

This step can be done asynchronously, so we can create a new operation class that extends Thread and do this job alone and add it to the stack when it finishes.

The advantage of this approach is utilizing the whole server memory and virtual CPUs to finish up this process.

But since we are dealing with multi-threads and creating our own new jobs we might have some major bottlenecks in the CPU, RAM, or I/O (network, storage ...).