NodeJS Coding Problem

Introduction

This coding task is designed to evaluate how you approach creative problem-solving in writing a simple algorithm using NodeJS. The task is based on a real-world scenario in which we are scraping data from an API hosted on premises with the need to throttle requests, for example to ensure availability is maintained for other users during normal business hours.

Time

A senior developer typically solves this problem in 30-60 minutes, and it should not take more than 2 hours in total.

Requirements

Write a NodeJS function that solves the problem to the best of your ability. Along with your code, please provide 1-2 paragraphs of text explaining why you chose this approach, and what other methods you considered and rejected. Note that **no** external libraries/dependencies are allowed at all.

You can use StackOverflow etc. but be aware we are aware of all similar solutions and many do not fulfill all of the requirements.

Task

You are given a list of async tasks to complete in an array:

```
const taskList =
['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T
','U','V','W','X','Y','Z'];
```

Where each task is mocked to take up to a few hundred milliseconds with the following function:

```
var doTask = (taskName) => {
 var begin=Date.now();
 return new Promise(function(resolve,reject){
    setTimeout(function(){
     var end= Date.now();
     var timeSpent=(end-begin)+ "ms";
        console.log('\x1b[36m', "[TASK] FINISHED: " + taskName + " in " +
timeSpent ,'\x1b[0m');
        resolve(true);
    },(Math.random()*200));
  });
}
```

Your task is to write a NodeJS function that will process N number of tasks concurrently, until the taskList is exhausted. N defines the number of concurrent promises that your script should attempt to resolve, and should be **changeable on the fly.** For example, if the current local time is between 9am and 5pm, N = 10, otherwise N = 150.

It is very important to note that promises must be processed in parallel, such that if N = 10, then there are always 10 promises being resolved (e.g. API requests). When one finishes, another immediately starts. It is **not** acceptable to resolve 10 promises, wait until all 10 have resolved, and then begin another 10.
For example, if N = 3, then you will execute 'A', 'B', and 'C' in the `taskList` concurrently. Once B finishes, you then execute 'D', and so on.

Restrictions: Use pure NodeJS (JS or TS) with **no** external libraries/dependencies at all allowed.

Key Requirements

1) Promises must be resolved concurrently maintaining a maximum number in progress at any given time.
2) The concurrency level must be changeable on the fly for example based on time of day or some other external event.
3) Write clear and commented code adhering to best practices.

Hints:

You can generate a taskList of arbitrary length for testing purposes with the following code:

```
numberOfTasks = 500;
  const taskList = [...Array(numberOfTasks)].map(() =>
  [...Array(~~(Math.random() * 10  + 3))].map(() =>
     String.fromCharCode(Math.random() * (123 - 97) + 97)
  ).join('') )
```

Your setup might look something like this:

```
async function init() {
 numberOfTasks = 20;
 const concurrencyMax = 4 ;
  const taskList = [...Array(numberOfTasks)].map(() =>
  [...Array(~~(Math.random() * 10  + 3))].map(() =>
     String.fromCharCode(Math.random() * (123 - 97) + 97)
  ).join('') )
 const counter = 0;
 const concurrencyCurrent = 0
 console.log("[init] Concurrency Algo Testing...")
 console.log("[init] Tasks to process: ", taskList.length)
 console.log("[init] Task list: " + taskList)
 console.log("[init] Maximum Concurrency: ", concurrencyMax,"\n")
```

```
 await manageConcurrency(taskList,counter,concurrencyMax,concurrencyCurrent);

}
```

Your output might look something like this: