PARALLEL AND DISTRIBUTED IMAGE DECRYPTION USING DES

A PROJECT REPORT

Submitted by

C Bharath Sai Reddy (17BCE0403)

P Harsha (17BCE0553)

M.V.Rakesh Kumar (17BCE0350)

S Kowshik (17BCE0421)

P.V.Prabhat (17BCE0337)

Course Code: Parallel and Distributed Computing

Course Title: CSE4001

Under the guidance of

Gopichand G

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

INDEX

ABSTRACT

The project proposes and implements a method to implement a task in its entirety and to parallel it across distributed systems. It makes sense to experiment with a computationally intensive task to illustrate the usability. The project therefore uses image encryption for the same purpose as a testing and benchmarking algorithm.

A client must upload a photo for encryption or decryption. The encryption and decryption process itself will be carried out using the JavaScript-implemented DES (Data Encryption Standard) algorithm. By spreading the work between different nodes of the server network, the image encryption will be parallelized. A main server will be hosted and Clustering Module will be used to build other server instances. The work will be split between different servers. The load will be shared between the server instances. The picture data will be separated and sent to various instances of the server. The encryption / decryption process is performed on different child servers in parallel, and then the computed data is returned to the main server which combines the data. Then the user can download the processed image.

In this project, compared to running in a single core machine, we measure the performance to accomplish the above task with distributed systems.

1. INTRODUCTION

Parallelism is used to accelerate any program code execution. The main part of parallelism is the ability to decide which part of a system needs to be parallelized. Frames can be read into memory as data matrices in the case of Image Processing, and operations on these pixels or array of pixels are typically parallel. The task must be analyzed in order to judge whether this parallelism is sufficiently important to be put into practice. It is better to make it parallel if the task is computationally intensive. These can be parallelized if the functions are separated and need less overhead interaction.

Today, parallelism of single machine systems is observed and applied most of the time. Given the limited transmission rate in network transmission devices, it takes considerable effort to get an algorithm running parallel between a network of computers. There must also be smooth interaction between these nodes in the network, and some form of ordering between the nodes must be based on a common consensus.

Nevertheless, we have several heterogeneous machines in a distributed system (different number of processors for each machine). In order to achieve maximum parallelism, we can use these cores and further divide our tasks into individual cores after distribution among the nodes.

This project aims to explore and introduce a use-case that achieves parallelism on distributed systems, observing and interpreting outcomes for the same purpose. The use case is the encryption of the picture.

2. LITERATURE SURVEY

Authors and Year (Reference)	*Title (Study)*	*Concept / Theoretical model/ Framework*	*Methodology used/ Implementation*	*Dataset details/ Analysis*	*Relevant Finding*	*Limitations/ Future Research/ Gaps identified*
Manjula K G,M N Ravikumar (2016,July)	Colour Image Encryption and Decryption Using DES Algorithm.	proposed plan has strength to resist the developing attacks on security of image file transmissions	Color image encryption and decryption is done by using DES algorithm, by providing required security for image between two authorized users or clients. In our project DES guarantee the unbreakable security for color image	represented the color image encryption and decryption with MATLAB. Here we select the Lena image with size 225x225.The proposed DES algorithm is applies on the Lena image. The input image is split into 8x8 block based on the 64 bit constrain.	The results of image encryption and image decryption using ECC based DES algorithm is done successfully.	enhancing T-DES security level by implementing DES technique, by repeating three times of DES key to generate three sets of key for TDES algorithm for make T-DES algorithm more secure.
Sanjay Kumar,Sandeep Srivastava (2014,October).	Image Encryption using Simplified Data Encryption Standard (S-DES)	In this paper a number of image encryption algorithms based on chaotic maps has been proposed.	The transformation modeled by the Chaotic map and then found a chaotic image of the original image, that chaotic image is then used for encryption of the original image.	we are using only first iteration of the chaotic image of original image of lena, and this chaotic image treated as the key for encryption of the	The use of S-DES algorithm increases the confusion of the encrypted image. Indeed, all performance analysis proves the security robustness	1. The key size is low in this algorithm. 2. Due to low key size, the security of S-DES algorithm is reduced. 3. If we use lots of data such as an image, then that algorithm

				original image of lena.	of the proposed algorithm.	can’t satisfy the encryption requirement.

3. OVERVIEW OF THE WORK

3.1 PROBLEM DESCRIPTION

Image encryption is an algorithm-based (usually two-way) method of changing pixel data inside an image. This algorithm uses individual pixel information and transfers it to a different value for later decryption.

In sequential order, a serial algorithm would have to pass through each pixel and encrypt the pixel data. As with any other processing on an image, this takes a considerable amount of time.

It would increase the speed of encryption to parallel this cycle between multiple cores. This makes this method highly effective when used in combination with multiple systems and their individual heart, limited only by the overhead interaction.

3.2 SOFTWARE REQUIREMENTS

- Image Encryption and Decryption using DES algorithm implemented on JavaScript
- NodeJS and NPM for Server-Side Scripting.
- Clustering Module: Cluster - to create child server instances
- Jimp package for image processing/
- Docker for containerizing the servers
- Docker Swarm for Orchestrating the Containers

3.3 HARDWARE REQUIREMENTS

- Linux Machines with multiple cores (Minimum two machines with multiple cores)
- WLAN/Ethernet Connection between two machines

4. SYSTEM DESIGN

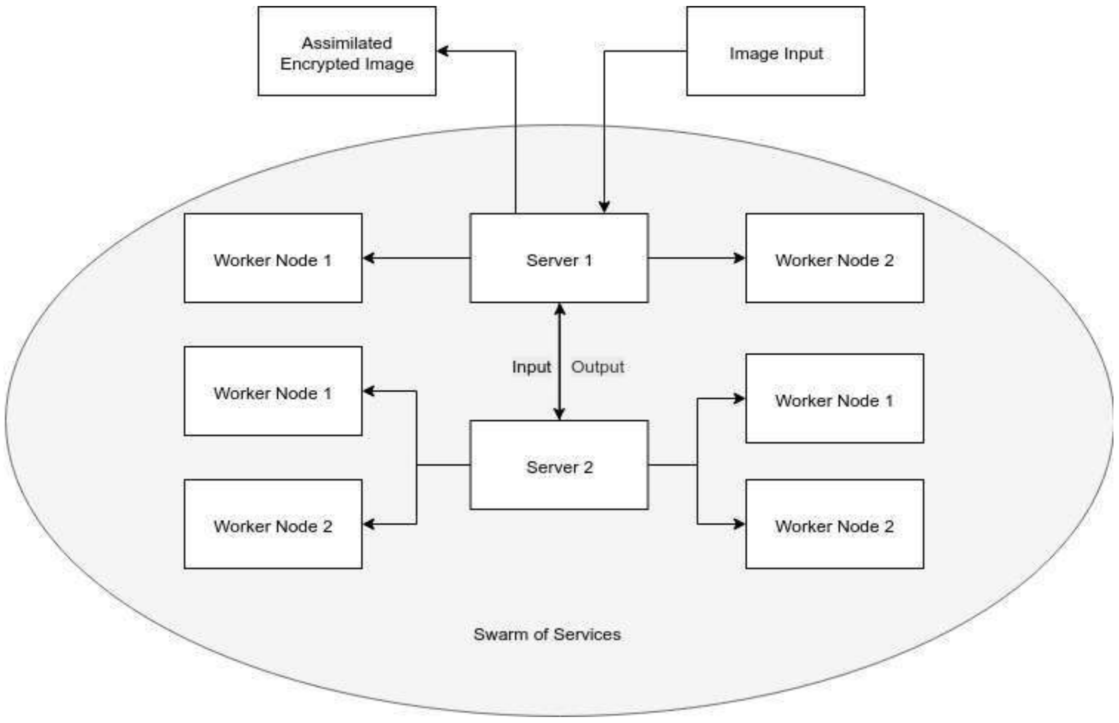


Fig.No.1 System Design of Image Encrypting Process

5. IMPLEMENTATION

5.1 DESCRIPTION OF MODULES/PROGRAMS

Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment running outside of a browser. Usually, JavaScript is mainly used for client-side scripting, where JavaScript-written scripts are inserted in the HTML of a web page, and a JavaScript engine operates client-side in the user's web browser. Node.js allows developers to use JavaScript to write command line tools and server-side scripting— running server-side scripts to create dynamic web page content prior to sending the page to the user's web browser. Node.js is therefore a "JavaScript anywhere" framework that unifies the development of web applications in a single programming language, instead of separate server side and client side scripts languages.

Express

Express is a minimal and flexible web application framework for Node.js that provides a robust set of web and mobile applications development features. It enables the rapid development of Web applications based on nodes. Below are some of Express Framework's core features

- Allows you to set up middlewares to respond to HTTP requests.
- Defines a routing table used to perform various HTTP and URL-based behavior.
- Allows HTML pages to be dynamically rendered by passing arguments to templates.

Cluster

The cluster module is a NodeJS module that contains a set of functions and properties that help us forge multi-core processes. It is probably the first level of scalability that you need to take care of in your node application, especially if you are working in an HTTP server application, before you go to a higher level of scalability (meaning to scaling vertically and horizontally in different machines).

In any number of child / parent processes, a parent / master process can be forked with the cluster module and communicate with them via IPC communication. Node.js runs a single instance in a single thread. Sometimes the client will want to start a cluster of Node.js processes to handle the load in order to take advantage of multi-core systems. The cluster module enables child processes that all server ports share to be easily created.

5.2 SOURCE CODE

```
const computeImage = require("./services/image");

const task = require("./services/task");

const cluster = require("cluster");

const numCPUs = require("os").cpus().length;

const server = require("./services/create-server");

let numberOfActiveChildProcesses = 0;

let workers = [];

// let pos = 0;

let encryptedData = [];

let temp = [];

if (cluster.isMaster) {

  console.log(`Number of processors are ${numCPUs}`);

  masterProcess();

} else {

  childProcess();

}
```

```
function mergeData(tempArray, callback) {
  // console.log(tempArray.length);
  tempArray.sort(function(a, b) {
    return a.id - b.id;
  });
  for (let obj of tempArray) {
    for (let pixel of obj.data) {
      encryptedData.push(pixel);
    }
  }
  return callback(encryptedData);
  // return computeImage.write(encryptedData, callback);
}

function childProcess() {
  process.on("message", obj => {
    console.log(
      `Message recieved by child process Id ${process.pid} is ${
        obj.data.pixels.length
      }`
    );
    let encryptedPixels = task.encrypt(obj.data.pixels);
    console.log(
      `Process ${process.pid} recieves pixel data of length ${
        encryptedPixels.length
      }`
    );
    process.send({ id: obj.data.id, data: encryptedPixels }, () => {
      process.exit();
    });
  });
}

module.exports.encryptImage = function(pixelArray, callback) {
  numberOfActiveChildProcesses = 0;
  workers = [];
  let pos = 0;
  encryptedData = [];
  temp = [];

  // cluster.disconnect(async () => {
  console.log(`Master ${process.pid} is running`);
  for (let i = 0; i < numCPUs; i++) {
    console.log(`Forking process number ${i}`);
    const worker = cluster.fork();

    numberOfActiveChildProcesses++;

    workers.push(worker);


    worker.on("message", msg => {

      console.log(

        `Master recieves msg ${msg.data.length} from worker ${msg.id}`

      );

      temp.push(msg);

    });


    worker.on("disconnect", () => {

      numberOfActiveChildProcesses--;

      if (numberOfActiveChildProcesses === 0) {

        console.log("encryption is done");

        return mergeData(temp, callback);

      }

    });

  }


  let numOfPixels = pixelArray.length;

  // console.log("Width of the image:", image.bitmap.width);
```

```
  // console.log("Height of the image:", image.bitmap.height);
  console.log("Encrypting the image...");

  workers.forEach((worker, index) => {
    worker.send({
      data: {
        id: index + 1,
        pixels: pixelArray.slice(
          pos,
          pos + parseInt(numOfPixels / workers.length)
        )
      }
    });
    pos = pos + parseInt(numOfPixels / workers.length);
  });
  // });
};

function masterProcess() {
  server.initiate(() => console.log("Listening on http://localhost:3000"));
  // encryptImage();
}
```

5.3 EXECUTION

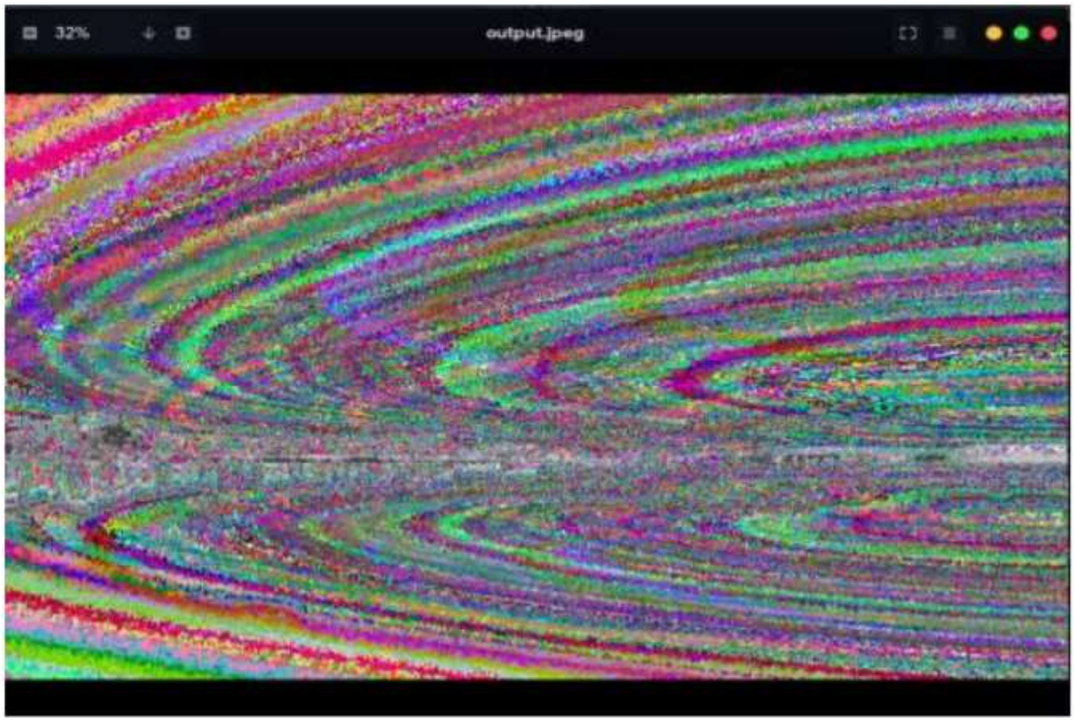
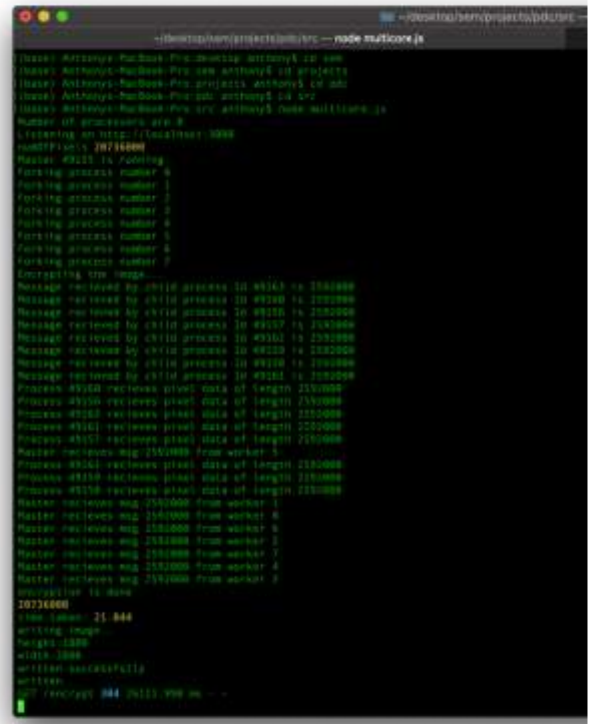
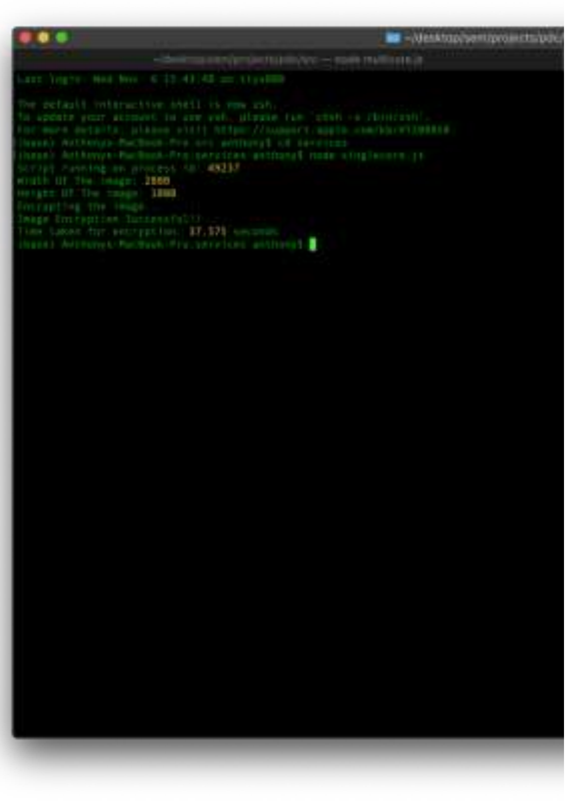


Fig.No. 2 Encrypted Image



```
~/desktop/sem/projects/pdc/src — node multicore.js
(base) Anthonys-MacBook-Pro:desktop anthony$ cd sem
(base) Anthonys-MacBook-Pro:sem anthony$ cd projects
(base) Anthonys-MacBook-Pro:projects anthony$ cd pdc
(base) Anthonys-MacBook-Pro:pdc anthony$ cd src
(base) Anthonys-MacBook-Pro:src anthony$ node multicore.js
Number of processors are 8
Listening on http://localhost:3000
numOfPixels 20736000
Master #49155 is running
Forking process number 0
Forking process number 1
Forking process number 2
Forking process number 3
Forking process number 4
Forking process number 5
Forking process number 6
Forking process number 7
Encrypting the image....
Message recieved by child process ID #49163 is 2592000
Message recieved by child process ID #49160 is 2592000
Message recieved by child process ID #49156 is 2592000
Message recieved by child process ID #49157 is 2592000
Message recieved by child process ID #49162 is 2592000
Message recieved by child process ID #49159 is 2592000
Message recieved by child process ID #49158 is 2592000
Message recieved by child process ID #49161 is 2592000
Process #49160 recieves pixel data of length 2592000
Process #49156 recieves pixel data of length 2592000
Process #49163 recieves pixel data of length 2592000
Process #49161 recieves pixel data of length 2592000
Process #49157 recieves pixel data of length 2592000
Master recieves msg 2592000 from worker 5
Process #49162 recieves pixel data of length 2592000
Process #49159 recieves pixel data of length 2592000
Process #49158 recieves pixel data of length 2592000
Master recieves msg 2592000 from worker 1
Master recieves msg 2592000 from worker 8
Master recieves msg 2592000 from worker 6
Master recieves msg 2592000 from worker 2
Master recieves msg 2592000 from worker 7
Master recieves msg 2592000 from worker 4
Master recieves msg 2592000 from worker 3
Encryption is done
20736000
time taken: 21.044
writing image...
height 1800
width 3800
written successfully
written
GET /encrypt 304 21111.990 ms - -
```

Fig.No.3 Execution in multicores



```
Last login: Wed Nov  6 15:43:48 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Anthonys-MacBook-Pro src anthony$ cd services
(base) Anthonys-MacBook-Pro:services anthony$ node singlecore.js
Script running on process id: 49237
Width Of The image: 2880
Height Of The image: 1800
Encrypting the image
Image Encryption Successful!!
Time taken for encryption: 37.575 seconds
(base) Anthonys-MacBook-Pro:services anthony$
```

Fig.No. 4 Execution in singlecore

6. CONCLUSION AND FUTURE WORKS

When used with two devices, the total time taken for encryption amounted to 20.15 seconds-one with 4 cores and one with 8 cores. When running the same in a single4-core machine, the time taken is 50.47 seconds, and it takes 93.69 seconds when running on a single core, making implementation considerably faster.

Our implementation's only limitation is the time it takes to pass servers responses (communication overhead) and assimilate them at one end.

This system can be expanded to multiple users, and a solution based on blockchain can help to regulate the flow. A cryptocurrency can be used when connecting to the network to control the computing power that can be pledged. The entire network will operate as a global virtual machine if network assets are needed to be used, with the number of threads being the total number of threads added by the people on the network. Furthermore, if there are 50000 connected nodes on the network, the virtual machine will be equivalent to at least 50000 threads or process managers of a heterogeneous framework. The threads will be selected based on a random selection, depending on the program requirements. So if a code is sent to the network to calculate, the nodes offering resources to calculate will be credited with a quantity of cryptocurrency, empirically determined from the node requesting. There will therefore be no need to manage full distributed networks at one end, as the network will have a well-defined consensus.

7. REFERENCES

[1] Manjula K G,M N Ravikumar(2016,July) .Colour Image Encryption and Decryption Using DES Algorithm.

https://www.irjet.net/archives/V3/i7/IRJET-V3I7322.pdf

[2] Sanjay Kumar,Sandeep Srivastava(2014,October). Image Encryption using Simplified Data Encryption Standard (S-DES)

https://pdfs.semanticscholar.org/fee0/e8b594b5ab56e62ae96bbda8f7c84d59c120.pdf