BỘ THÔNG TIN & TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN 2

ĐỒ ÁN CUỐI KỲ
CÁC HỆ THỐNG PHÂN TÁN

ĐỀ TÀI
ĐỒNG BỘ ĐỒNG HỒ VẬT LÝ CÁC MÁY TÍNH TRONG HỆ THỐNG THEO THUẬT TOÁN BERKELEY

THẠC SĨ, GIẢNG VIÊN HƯỚNG DẪN: LÊ NGỌC BẢO

MÃ MÔN HỌC: INT1405

NHÓM 7

NGUYỄN DƯƠNG PHI – N20DCCN125
NGUYỄN THỊ HIỀN – N20DCCN099
NGUYỄN THỊ THU HIỀN – N20DCCN100
NGUYỄN VŨ QUANG – N20DCCN130
NGUYỄN TRỌNG NHÂN – N20DCCN122

TP. HỒ CHÍ MINH, THÁNG 01 – 2024

– LỜI CẢM ƠN –

– Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Học viện Công nghệ Bưu chính Viễn thông cơ sở tại Thành phố Hồ Chí Minh đã đưa bộ môn Các hệ thống phân tán vào chương trình giảng dạy.

– Tiếp theo chúng em xin chân thành cảm ơn thầy Lê Ngọc Bảo trong thời gian qua đã dạy, hướng dẫn, giúp đỡ chúng em trong quá trình học tập. Thầy không chỉ là người hướng dẫn mà còn là nguồn động viên quan trọng, giúp em nhận thức rõ hơn về tầm quan trọng của các hệ thống phân tán trong lĩnh vực công nghệ thông tin. Thầy đã chia sẻ những kiến thức bổ ích và cung cấp ví dụ minh họa, giúp em tiếp cận môn học một cách linh hoạt và thú vị.

– Chúc thầy luôn luôn mạnh khỏe, luôn vui tươi, dồi dào sức sống và có nhiều thành công trong công việc giảng dạy!

– Cuối cùng chúng em xin cảm ơn gia đình, người thân và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên chúng em trong suốt quá trình học tập và thực hiện đồ án cuối kỳ này.

Nhóm sinh viên thực hiện.
NHÓM 7

– MỤC LỤC –

CHƯƠNG I. TỔNG QUAN VỀ ĐỀ TÀI

1. LÝ DO CHỌN ĐỀ TÀI:

Trong thời đại ngày nay, khi cuộc cách mạng công nghệ thông tin liên tục chuyển động, hệ thống phân tán nổi lên như một trụ cột quan trọng trong việc xử lý và quản lý thông tin. Sự phát triển của máy tính phân tán đặt ra một thách thức lớn: làm thế nào để duy trì đồng bộ về thời gian, đảm bảo tính nhất quán và độ tin cậy của dữ liệu. Trong ngữ cảnh này, việc áp dụng các thuật toán đồng bộ đồng hồ vật lý trở nên cực kỳ quan trọng, và thuật toán Berkeley nổi bật như một lựa chọn không thể thiếu.

Sự lựa chọn của đề tài này xuất phát từ sự cần thiết ngày càng tăng về việc đảm bảo sự đồng bộ thời gian giữa các nút máy tính phân tán trong một hệ thống. Một hệ thống phân tán hiệu quả đòi hỏi không chỉ sự tương tác chính xác giữa các thành phần mà còn đặt ra yêu cầu cao về tính nhất quán trong đánh dấu thời gian. Thuật toán Berkeley, được thiết kế đặc biệt để giải quyết vấn đề này, trở thành sự lựa chọn lý tưởng để tiến hành nghiên cứu và triển khai.

Qua việc áp dụng thuật toán Berkeley, chúng ta có thể tối ưu hóa quá trình đồng bộ hóa thời gian, giảm thiểu độ trễ và tăng cường khả năng chịu lỗi của hệ thống. Hơn nữa, việc này mang lại những lợi ích to lớn cho việc phát triển các ứng dụng và dịch vụ phức tạp, nơi tính nhất quán và đáng tin cậy là chìa khóa của sự thành công.

Trong bối cảnh hiện nay, với sự tập trung ngày càng cao của tổ chức và doanh nghiệp vào việc kết nối liền mạch giữa các thành phần hệ thống, nghiên cứu về áp dụng thuật toán Berkeley trở nên càng trở nên thiết thực và cần thiết. Bài báo cáo này đặt ra nhiệm vụ tập trung vào việc trình bày cơ sở lý thuyết và ứng dụng thực tế của thuật toán Berkeley trong các hệ thống phân tán, với hy vọng góp phần vào sự hiểu biết và phát triển của lĩnh vực này.

2. GIỚI THIỆU THUẬT TOÁN:

2.1. KHÁI NIỆM:

Thuật toán Berkeley là một phương pháp đồng bộ hóa đồng hồ trong tính toán phân tán mà không chịu bất cứ máy có nguồn thời gian chính xác nào. Nó được phát triển bởi Gusella và Zatti tại Đại học California, Berkeley vào năm 1989.

Thuật toán Berkeley được sử dụng rộng rãi trong các hệ thống phân tán, chẳng hạn như mạng máy tính, hệ thống mạng lưới, hệ thống điều khiển công nghiệp, v.v…

2.2. CÔNG THỨC:

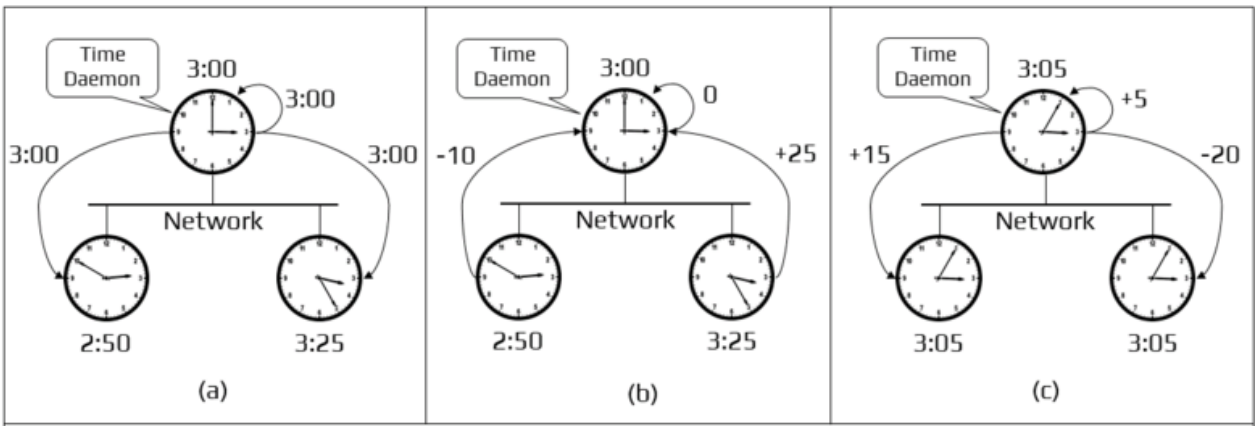
Nếu T_c là thời gian hiện tại của máy tính con, T_s là thời gian hiện tại của máy chủ, và δ là sự chênh lệch giữa chúng, thì sự điều chỉnh của đồng hồ (δt) có thể được tính bằng công thức:

$\delta t = T_s - T_c + (\text{round−trip time}) / 2$

Trong đó, "round–trip time" là thời gian mà thông điệp mất để đi từ máy con đến máy chủ và quay lại.

Công thức này tính toán sự chênh lệch giữa thời gian hiện tại của máy tính con và thời gian hiện tại của máy chủ và cố gắng đưa đồng hồ của máy tính con về phía đồng bộ với máy chủ.

2.3. Ý TƯỞNG:



CHÚ THÍCH:

Hình (a). Time Daemon quảng bá giá trị đồng hồ vật lý của mình.

Hình (b). Các máy khác trả lời.

Hình (c). Time Daemon tính toán giá trị trung bình và gửi cho các máy giá trị cần hiệu chỉnh.

Như hình trên, giả sử có 3 đồng hồ T1, T2, T3. Trong đó T1 là đồng hồ Server đang là 3:00, các đồng hồ còn lại là của Client với T2 = 2:50' và T3 = 3:25'.

Server sẽ gửi đồng hồ của mình cho các Client.

T2 và T3 tính độ lệch thời gian và gửi về cho Server lần lượt là: T1' = –10' và T2' = +25'.

Server sẽ tính trung bình: T = (0 + T1' + T2') / 3 = (0 + (–10) + 25) / 3 = 5'.

Sau đó Server sẽ đặt thời gian của mình là: T_{new} = 3h + 5' và báo cho các máy Client T2 (+15') và T3 (–20')

Ý TƯỞNG:

– Server chủ động cho các máy khác biết thời gian chuẩn của mình là C_{utc}.

– Sau đó yêu cầu thông tin về thời gian của các client.

– Client trả lời khoảng thời gian chênh lệch giữa nó và server.

– Server tính khoảng thời gian mà các client so với thời gian chuẩn của server lúc đó và gửi cho các máy khách cách điều chỉnh thời gian cho phù hợp.

2.4. ƯU ĐIỂM:

– Độ chính xác cao: Thuật toán Berkeley có độ chính xác cao trong các mạng nội bộ, với độ sai lệch thời gian trung bình chỉ khoảng 1 micro giây.

– Dễ triển khai: Nó không đòi hỏi nhiều tài nguyên tính toán hoặc mạng, điều này làm cho việc triển khai và duy trì thuật toán trở nên đơn giản.

– Đồng bộ hóa đơn giản: Thuật toán giúp các máy tính trong mạng đồng bộ hóa thời gian của chúng với một máy chủ thời gian (time server) mà không gặp nhiều khó khăn. Điều này hữu ích trong việc đảm bảo các sự kiện và ghi chú thời gian trên các máy tính là nhất quán.

– Khả năng mở rộng: Thuật toán Berkeley có thể mở rộng dễ dàng cho hệ thống phân tán lớn.

2.5. NHƯỢC ĐIỂM:

– Yêu cầu độ trễ mạng thấp: Thuật toán Berkeley yêu cầu độ trễ mạng thấp để đảm bảo độ chính xác. Trong các mạng diện rộng, độ trễ mạng có thể cao hoặc không đồng đều, dẫn đến độ chính xác của thuật toán Berkeley bị giảm sút.

– Yêu cầu đồng nhất thời gian: Thuật toán Berkeley yêu cầu các đồng hồ trên các máy tính trong hệ thống phải đồng nhất thời gian trong khoảng ±1 micro giây. Nếu các đồng hồ không đồng nhất, độ chính xác của thuật toán Berkeley sẽ bị giảm sút.

– Yêu cầu một máy chủ thời gian đáng tin cậy: Hiệu suất của thuật toán phụ thuộc lớn vào sự đáng tin cậy của máy chủ thời gian. Nếu máy chủ gặp sự cố hoặc bị tấn công, toàn bộ hệ thống có thể bị ảnh hưởng.

CHƯƠNG II. XÂY DỰNG CHƯƠNG TRÌNH

1. FILE client.js:

```
let socket = io();
const delayTime = 1000;
let socketDate = new Date(Date.now());
let viewTime = document.getElementById("server_time");
let table = document.getElementById("clock-table");

socket.on("request_time", () => {
  socket.emit("datetime", customClock());
});

socket.on("synchronize", (time) => {
  console.log("Hey! There is a new time:", new Date(time).toLocaleTimeString());
  clearInterval(interval);
  displayDiff(time);
  socketDate = new Date(time);
  console.log("socketDate", socketDate);
  interval = setInterval(intervalFun, delayTime);
});

socket.on("disconnect", () => {
  socket.close();
  console.log("Socket connection closed!");
});

function customClock() {
  let timeDiff = new Date(Date.now()).getSeconds() - socketDate.getSeconds();
  console.log("timeDiff", timeDiff);
  let customTime = socketDate;
  customTime.setSeconds(socketDate.getSeconds() + timeDiff);
  return customTime;
}

function displayDiff(time) {
  let syncTime = new Date(time);
  console.log("syncTime", syncTime);
  let secondsDiff = syncTime.getSeconds() - socketDate.getSeconds();
  let minutesDiff = syncTime.getMinutes() - socketDate.getMinutes();
```

```
38    let hoursDiff = syncTime.getHours() - socketDate.getHours();
39    let difference = `Hours: ${Math.abs(hoursDiff)}, minutes: ${Math.abs(
40      minutesDiff
41    )}, seconds: ${Math.abs(secondsDiff)}`;
42    addTableRow(
43      socketDate.toLocaleTimeString(),
44      difference,
45      syncTime.toLocaleTimeString()
46    );
47  }
48
49  function addTableRow(initialTime, timeDiff, newTime) {
50    let row = table.insertRow(1);
51    let startTime = row.insertCell(0);
52    let diff = row.insertCell(1);
53    let syncTime = row.insertCell(2);
54    startTime.innerHTML = initialTime;
55    diff.innerHTML = timeDiff;
56    syncTime.innerHTML = newTime;
57  }
58
59  function intervalFun() {
60    viewTime.innerText = customClock().toLocaleTimeString();
61  }
62
63  function editTime() {
64    let hoursEdit = document.getElementById("hoursControl");
65    let minutesEdit = document.getElementById("minutesControl");
66    let secondsEdit = document.getElementById("secondsControl");
67    socketDate.setHours(hoursEdit.value);
68    socketDate.setMinutes(minutesEdit.value);
69    socketDate.setSeconds(secondsEdit.value);
70  }
71
72  let interval = setInterval(intervalFun, delayTime);
```

MÔ TẢ SƠ LƯỢC:

THIẾT LẬP KẾT NỐI SOCKET & CÁC BIẾN:

`let socket = io();` : Khởi tạo kết nối WebSocket, cho phép giao tiếp thời gian thực giữa client và server.

`const delayTime = 1000;` : Đặt thời gian trễ 1000ms (1 giây) cho các thao tác định kỳ.

`let socketDate = new Date(Date.now());` : Tạo biến lưu trữ thời gian đồng bộ, ban đầu được đặt bằng thời gian hiện tại của client.

`let viewTime = document.getElementById("server_time");` : Lấy tham chiếu đến phần tử HTML dùng để hiển thị thời gian đồng bộ từ server.

`let table = document.getElementById("clock-table");` : Lấy tham chiếu đến bảng HTML dùng để hiển thị lịch sử đồng bộ và sai lệch thời gian.

`let interval = setInterval(intervalFun, delayTime);` : Tạo một khoảng thời gian định kỳ, gọi hàm intervalFun mỗi giây.

XỬ LÝ CÁC SỰ KIỆN WEB SOCKET:

```
socket.on("request_time", () => {
  socket.emit("datetime", customClock());
});
```

Được gọi khi server yêu cầu thời gian từ client. Gửi thời gian đồng bộ của client (được điều chỉnh dựa trên thời gian server trước đó) về server.

```
socket.on("synchronize", (time) => {
```

Được gọi khi server gửi thời gian đồng bộ mới.

`console.log("Hey! There is a new time:", new Date(time).toLocaleTimeString());`	Hiển thị thông báo thời gian mới trong console.
`clearInterval(interval);`	Dừng khoảng thời gian hiện tại `clearInterval(interval)`.
`displayDiff(time);`	Gọi hàm **`displayDiff`** để hiển thị sự sai lệch giữa thời gian client và server.
`socketDate = new Date(time);`	Cập nhật thời gian **`socketDate`** với thời gian mới nhận được từ server.
`interval = setInterval(intervalFun, delayTime);`	Tạo một khoảng thời gian mới.
`socket.on("disconnect", () => {`	Được gọi khi kết nối socket bị ngắt kết nối.
`socket.close();`	Đóng kết nối socket.
`console.log("Socket connection closed!");`	Hiển thị thông báo trong console.

CÁC HÀM CHỨC NĂNG:

Hàm đồng hồ tùy chỉnh:

```
function customClock() {
  let timeDiff = new Date(Date.now()).getSeconds() - socketDate.getSeconds();
  console.log("timeDiff", timeDiff);
  let customTime = socketDate;
  customTime.setSeconds(socketDate.getSeconds() + timeDiff);
  return customTime;
}
```

Tính toán chênh lệch thời gian giữa thời gian hiện tại của client và thời gian **`socketDate`**. Sau đó, điều chỉnh **`socketDate`** và trả về thời gian đã điều chỉnh.

Hàm hiển thị chênh lệch:

```
function displayDiff(time) {
  let syncTime = new Date(time);
  console.log("syncTime", syncTime);
  let secondsDiff = syncTime.getSeconds() - socketDate.getSeconds();
  let minutesDiff = syncTime.getMinutes() - socketDate.getMinutes();
  let hoursDiff = syncTime.getHours() - socketDate.getHours();
  let difference = `Hours: ${Math.abs(hoursDiff)}, minutes: ${Math.abs(minutesDiff
    )}, seconds: ${Math.abs(secondsDiff)}`;
  addTableRow(
    socketDate.toLocaleTimeString(),
    difference,
    syncTime.toLocaleTimeString()
  );
}
```

Tính toán chênh lệch thời gian giữa thời gian mới nhận được từ server và thời gian **`socketDate`**. Log chênh lệch thời gian và cập nhật bảng HTML với một hàng mới chứa thời gian ban đầu, chênh lệch thời gian và thời gian server mới.

Hàm thêm hàng bảng:

```
function addTableRow(initialTime, timeDiff, newTime) {
  let row = table.insertRow(1);
  let startTime = row.insertCell(0);
  let diff = row.insertCell(1);
  let syncTime = row.insertCell(2);
  startTime.innerHTML = initialTime;
  diff.innerHTML = timeDiff;
  syncTime.innerHTML = newTime;
}
```

Thêm một hàng mới vào bảng HTML với các ô cho thời gian ban đầu, chênh lệch thời gian và thời gian server mới.

Hàm cập nhật định kỳ:

```
function intervalFun() {
  viewTime.innerText = customClock().toLocaleTimeString();
}
```

Cập nhật thời gian server hiển thị mỗi 1000 miligiây (theo `delayTime`) bằng cách gọi hàm `customClock()` và cập nhật văn bản trong phần tử `viewTime`.

Hàm chỉnh sửa thời gian:

```
function editTime() {
  let hoursEdit = document.getElementById("hoursControl");
  let minutesEdit = document.getElementById("minutesControl");
  let secondsEdit = document.getElementById("secondsControl");
  socketDate.setHours(hoursEdit.value);
  socketDate.setMinutes(minutesEdit.value);
  socketDate.setSeconds(secondsEdit.value);
}
```

Lấy giá trị từ các trường nhập liệu giờ, phút, giây trên giao diện.
Cập nhật thời gian `socketDate` với các giá trị mới nhập vào.

2. FILE server.js:

```
"use strict";
const express = require("express");
const io = require("socket.io");
const port = process.env[2] || 5500;
const app = express();
const http = require("http").Server(app);
let socket = io(http);
app.use(express.static(__dirname + "/public"));
let nodes = [];
let clients, master_time, synchronized_time;
let allDifferencesSum,
  average_time = 0;

socket.on("connection", (node) => {
  console.log("Node connected:", node.id);
  node.join("time room");
  clients = socket.sockets.adapter.rooms["time room"];
  node.on("datetime", (time) => {
    nodes.push({ id: node.id, time: new Date(time) });
    if (nodes.length === clients.length) {
      berkeleyAlgorithm();
      node.emit("synchronize", synchronized_time);
      node.broadcast.emit("synchronize", synchronized_time);
    }
  });
  node.on("disconnect", () => {
    node.leave("time room", "");
    deleteDisconnectedNode(node);
  });
});

async function getServerDate() {
  const today = new Date();
  const datetime = today.toISOString();
  return datetime;
}

function berkeleyAlgorithm() {
  nodes.forEach((slave) => {
    let time_difference = slave.time.getTime() - master_time.getTime();
    allDifferencesSum += time_difference;
    console.log("time server current: ", master_time.toLocaleTimeString());
    console.log("slave.time: ", slave.time.toLocaleTimeString());
  });
  average_time = allDifferencesSum / (nodes.length + 1);
  synchronized_time = master_time.getTime() + average_time;
  console.log(
    `New server time: ${new Date(synchronized_time).toLocaleTimeString()}`
  );
}

function deleteDisconnectedNode(node) {
  console.log("Node disconnected:", node.id);
  let index = nodes
    .map((object) => {
      return object.id;
    })
    .indexOf(node.id);
  nodes.splice(index, 1);
}
```

```
61
62  function resetAlgorithmTime() {
63    nodes = [];
64    allDifferencesSum = 0;
65    average_time = 0;
66  }
67
68  setInterval(async () => {
69    await resetAlgorithmTime();
70    await getServerDate()
71      .then((datetime) => {
72        if (synchronized_time === undefined) {
73          master_time = new Date(datetime);
74        } else {
75          master_time = new Date(synchronized_time);
76        }
77      })
78      .catch((err) => {
79        console.log(err.Error);
80      });
81    await socket.to("time room").emit("request_time");
82  }, 3000);
83
84  http.listen(port, () => {
85    console.log("Running on Port: " + port);
86  });
```

MÔ TẢ SƠ LƯỢC:

KHAI BÁO VÀ IMPORT THƯ VIỆN:

```
"use strict";
```

Áp dụng các quy tắc mã hóa nghiêm ngặt để bắt lỗi tốt hơn, tăng tính an toàn và kiểm soát trong JavaScript.

```
const express = require("express");
```

Import thư viện Express để tạo WebServer.

```
const io = require("socket.io");
```

Import Socket.IO cho giao tiếp thời gian thực.

```
const port = process.env[2] || 5500;
```

Đặt cổng server, tùy chọn sử dụng biến môi trường hoặc mặc định là 5500.

```
const app = express();
```

Tạo một thể hiện ứng dụng Express.

```
const http = require("http").Server(app);
```

Tạo server HTTP, truyền ứng dụng Express vào.

```
let socket = io(http);
```

Khởi tạo Socket.IO trên máy chủ HTTP.

```
app.use(express.static(__dirname +
"/public"));
```

Phục vụ các tài nguyên tĩnh từ thư mục `/public`.

KHỞI TẠO CÁC BIẾN:

```
let nodes = [];
```

Mảng để lưu trữ thông tin các node trong mạng.

```
let clients, master_time,
synchronized_time;
```

Khai báo 3 biến được sử dụng để lưu trữ thông tin về số lượng clients (node), thời gian chủ, và thời gian được đồng bộ hóa.

```
let allDifferencesSum,
  average_time = 0;
```

Các biến để tính toán sự khác biệt về thời gian và thời gian trung bình.

XỬ LÝ KẾT NỐI:

```
socket.on("connection", (node) => {

  console.log("Node connected:", node.id);

  node.join("time room");

  clients =
socket.sockets.adapter.rooms["time room"];
  node.on("datetime", (time) => {

    nodes.push({ id: node.id, time: new
Date(time) });

    if (nodes.length === clients.length) {

      berkeleyAlgorithm();

      node.emit("synchronize",
synchronized_time);
      node.broadcast.emit("synchronize",
synchronized_time);
    }
  });
  node.on("disconnect", () => {
    node.leave("time room", "");
    deleteDisconnectedNode(node);
  });
});
```

Lắng nghe kết nối nút mới tới máy chủ. Khi một nút kết nối:
Ghi lại một thông báo về việc một node đã kết nối thành công, kèm theo ID của node đó.
Thêm nút đã kết nối vào phòng ảo có tên `"time room"` giúp trong việc quản lý các node để thực hiện đồng bộ hóa thời gian.
Lấy danh sách các clients (nodes) trong phòng `"time room"` và lưu vào biến **`clients`**.
Lắng nghe các sự kiện có tên `"datetime"` được phát ra bởi node đã kết nối:
Lưu dấu thời gian đã nhận và ID node vào mảng **`nodes`**.
Kiểm tra số nodes đã đủ để thực hiện đồng bộ hóa chưa. Nếu số lượng nodes bằng số lượng clients trong `"time room"`, tiến hành đồng bộ hóa.

Gọi hàm thực hiện thuật toán Berkeley để tính toán thời gian đồng bộ hóa.
Gửi thời gian đồng bộ trở lại nút đã kết nối.
Phát thời gian đồng bộ cho tất cả các nút khác trong phòng `"time room"`.

Gắn một sự kiện lắng nghe cho sự kiện `”disconnect”` khi node ngắt kết nối. Node rời khỏi phòng `"time room"`. Điều này được thực hiện khi node ngắt kết nối. Gọi hàm để xóa thông tin của node đã ngắt kết nối khỏi mảng **`nodes`**.

LẤY NGÀY, GIỜ MÁY CHỦ:

```
async function getServerDate() {
  const today = new Date();
  const datetime = today.toISOString();
  return datetime;
}
```

Lấy ngày và giờ máy chủ hiện tại:
Tạo một đối tượng Date mới.
Trả về ngày và giờ ở định dạng ISO.

ĐẶT LẠI CÁC BIẾN THUẬT TOÁN:

```
function resetAlgorithmTime() {
  nodes = [];
  allDifferencesSum = 0;
  average_time = 0;
}
```

Gán giá trị mảng rỗng cho biến **`nodes`**.
Đặt biến allDifferencesSum về giá trị 0.
Đặt biến average_time về giá trị 0.

THUẬT TOÁN BERKELEY:

```
function berkeleyAlgorithm() {
  nodes.forEach((slave) => {
    let time_difference = slave.time.getTime()
- master_time.getTime();

    allDifferencesSum += time_difference;

    console.log("time server current: ",
master_time.toLocaleTimeString());
    console.log("slave.time: ",
slave.time.toLocaleTimeString());
  });
  average_time = allDifferencesSum /
(nodes.length + 1);

  synchronized_time = master_time.getTime() +
average_time;
  console.log(
    `New server time: ${new
Date(synchronized_time).toLocaleTimeString()}`
  ); }
```

Duyệt qua các node.
Tính sự khác biệt thời gian giữa mỗi node **(slave.time)** và của máy chủ **(master_time)**.
Tổng các sự khác biệt về thời gian được tích lũy vào biến **allDifferencesSum.**
In thông tin về thời gian hiện tại của máy chủ và thời gian của mỗi node ra console. Đây là bước kiểm tra và ghi log để theo dõi quá trình.

Thời gian trung bình được tính bằng cách chia tổng các sự khác biệt về thời gian cho số lượng node cộng thêm 1 (để tính cả thời gian chủ).
Tính thời gian đồng bộ hóa bằng cách cộng thời gian trung bình vào thời gian của máy chủ.
In ra console thời gian mới sau khi đồng bộ hóa. Điều này làm thời gian của máy chủ được cập nhật theo thời gian trung bình của các node.

XỬ LÝ NGẮT KẾT NỐI NÚT:

```
function deleteDisconnectedNode(node) {
  console.log("Node disconnected:",
node.id);
  let index = nodes.map((object) => {
      return object.id;
    })
    .indexOf(node.id);

  nodes.splice(index, 1);
}
```

In ra console một thông báo cho biết rằng một node đã ngắt kết nối và in ra ID của node đó.
Tìm chỉ số của node trong mảng **nodes** sử dụng **map** để chuyển đổi mảng các node thành mảng các ID của chúng, sau đó sử dụng indexOf để tìm chỉ số của **node.id** trong mảng ID.
Xóa đi node đã ngắt kết nối khỏi mảng **nodes**.

ĐỒNG BỘ HÓA ĐỊNH KỲ:

```
setInterval(async () => {

 await resetAlgorithmTime();
 await getServerDate().then((datetime) => {
  if (synchronized_time === undefined) {
    master_time = new Date(datetime);
  } else {
    master_time = new Date(synchronized_time);
  }
  });
 await socket.to("time
room").emit("request_time");
}, 3000);
```

Sử dụng **setInterval** để thực hiện một chuỗi các hành động mỗi 3 giây.
Gọi hàm **resetAlgorithmTime()** để đặt lại các biến sử dụng trong thuật toán Berkeley.
Gọi hàm **getServerDate()** để lấy thời gian từ server. Nếu chưa có thời gian đồng bộ, đặt thời gian chủ bằng thời gian server. Nếu đã có thời gian đồng bộ, đặt thời gian chủ bằng thời gian đồng bộ để duy trì sự đồng bộ.
Gửi sự kiện **"request_time"** đến tất cả các node trong phòng **"time room"**, yêu cầu chúng gửi dấu thời gian của chúng.

KHỞI ĐỘNG MÁY CHỦ:

```
http.listen(port, () => {
  console.log("Running on Port: " + port);
});
```

Tạo một máy chủ HTTP lắng nghe trên cổng port. Xuất ra console cho biết máy chủ đã được khởi động và đang chạy trên cổng port.

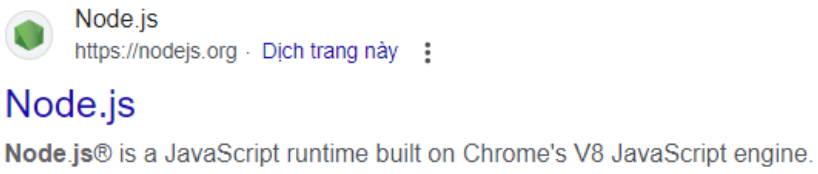
CHƯƠNG III. CÀI ĐẶT CHƯƠNG TRÌNH VÀ THỰC NGHIỆM

1. CÀI ĐẶT NODE JS:

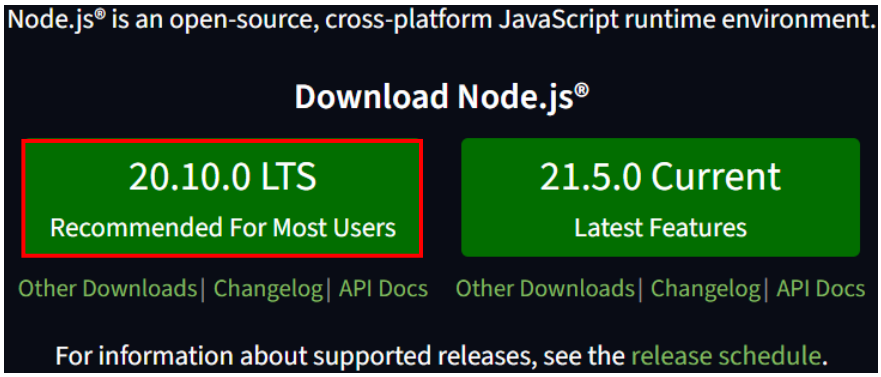
Node.js là một nền tảng chạy JavaScript mã nguồn mở, đa nền tảng được sử dụng để xây dựng các ứng dụng mạng. Nó được xây dựng trên V8, một máy ảo JavaScript được phát triển bởi Google. Node.js sử dụng kiến trúc hướng sự kiện, cho phép nó xử lý nhiều yêu cầu đồng thời một cách hiệu quả. Nó cũng hỗ trợ I/O không đồng bộ, có nghĩa là các tác vụ I/O không làm tắc nghẽn luồng chính của chương trình, thường được sử dụng để phát triển ứng dụng web và API có hiệu suất cao.

BƯỚC 1: Tải và cài đặt Node.js.

– Truy cập trang chính thức của Node.js tại https://nodejs.org/en



– Chọn phiên bản muốn tải xuống. Thường thì phiên bản LTS (Long Term Support) là lựa chọn ổn định và được khuyến nghị cho hầu hết người dùng.



– Tải xuống tệp cài đặt và chạy trình cài đặt theo hướng dẫn.

BƯỚC 2: Kiểm tra cài đặt Node.js và npm.

– Sau khi cài đặt xong, mở terminal (hoặc command prompt trên Windows) và kiểm tra việc cài đặt Node.js và npm (Node Package Manager) bằng cách gõ các lệnh sau:

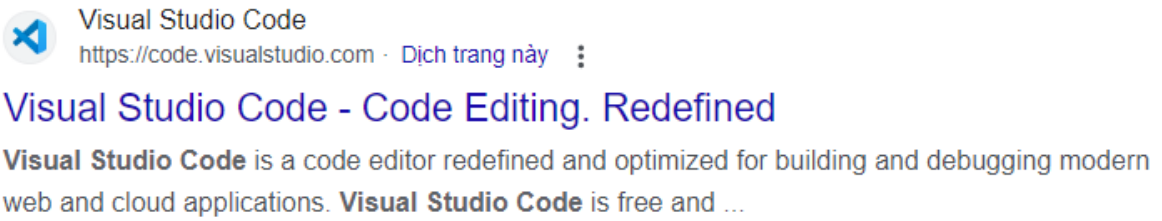
```
C:\Users\N20DCCN125>npm -v
10.2.4
```

– Nếu nhìn thấy các số phiên bản, điều này có nghĩa là Node.js và npm đã cài đặt thành công.

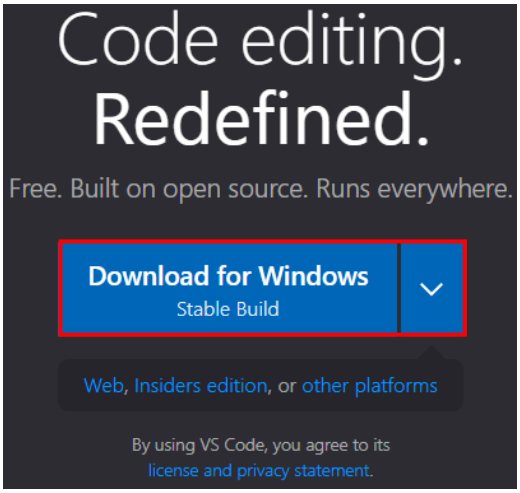
2. CÀI ĐẶT VISUAL STUDIO CODE:

Visual Studio Code là một trình soạn thảo mã nguồn mở và miễn phí được phát triển bởi Microsoft. Nó là một ứng dụng dùng để viết mã, chỉnh sửa và debug mã nguồn, và hỗ trợ nhiều ngôn ngữ lập trình khác nhau. Visual Studio Code là một công cụ mạnh mẽ có thể được sử dụng để phát triển các ứng dụng trên nhiều nền tảng.

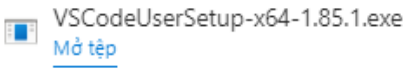
– Truy cập trang chính thức của Visual Studio Code tại https://code.visualstudio.com/



– Tải phiên bản phù hợp với hệ điều hành (Windows, macOS, hoặc Linux).

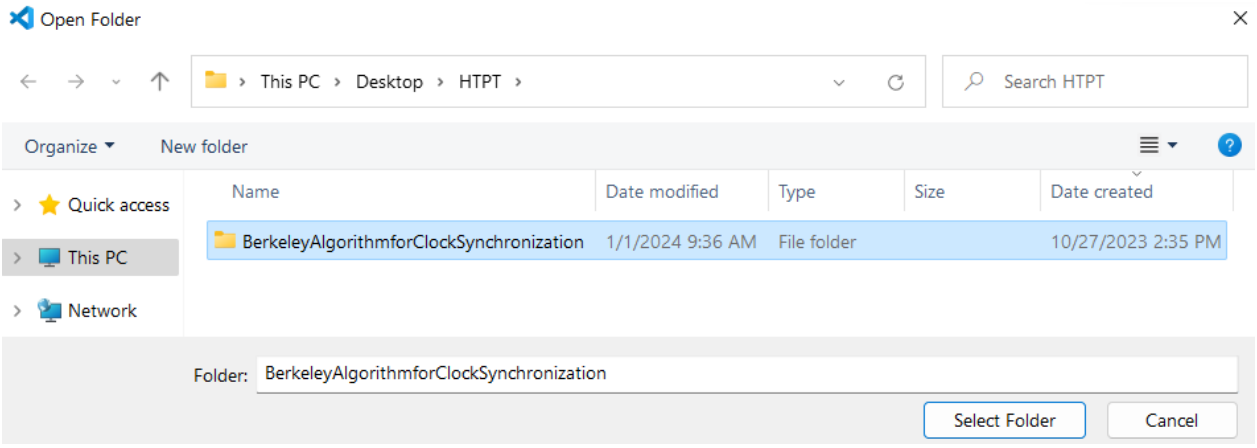


– Tải xuống tệp cài đặt và chạy trình cài đặt theo hướng dẫn.

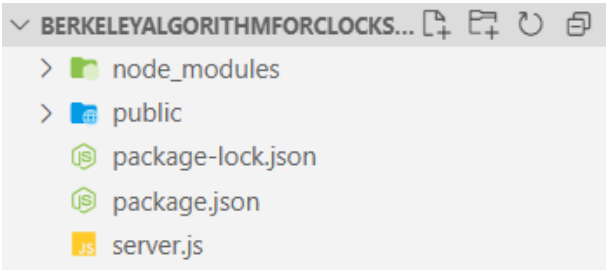


3. CHẠY DỰ ÁN:

– Trong Visual Studio Code, chọn **File → Open Folder**, sau đó chọn đến thư mục dự án và chọn **Select Folder**.



– Đây là cây thư mục của dự án.



– Chọn **Terminal → New Terminal** để mở cửa sổ dòng lệnh tích hợp.

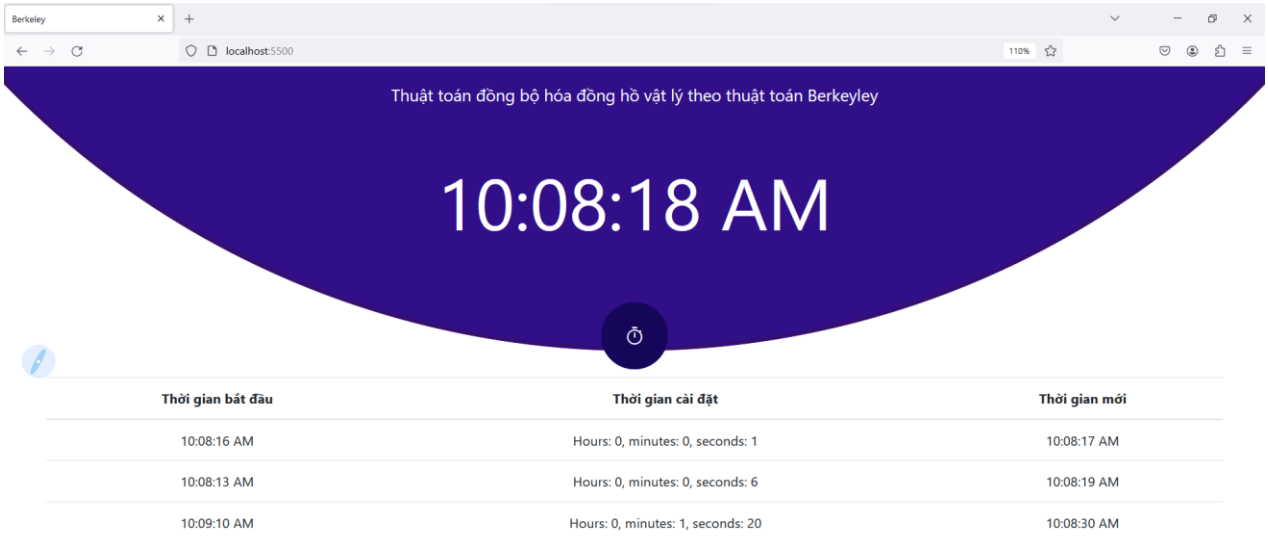
– Sau khi đã mở Terminal và đang ở trong thư mục dự án, gõ lệnh **`npm start`** và nhấn Enter.

```
PS C:\Users\N20DCCN125\Desktop\HTPT\BerkeleyAlgorithmforClockSynchronization> npm start

> berkeley-algorithm@1.0.0 start
> node server.js

Running on Port: 5500
```

– Mở trình duyệt web, truy cập vào địa chỉ http://localhost:5500/



Thời gian bắt đầu	Thời gian cài đặt	Thời gian mới
10:08:16 AM	Hours: 0, minutes: 0, seconds: 1	10:08:17 AM
10:08:13 AM	Hours: 0, minutes: 0, seconds: 6	10:08:19 AM
10:09:10 AM	Hours: 0, minutes: 1, seconds: 20	10:08:30 AM

CHƯƠNG IV. KẾT LUẬN

Đề tài đã đạt được những kết quả quan trọng trong việc nghiên cứu và triển khai thuật toán Berkeley. Trong quá trình thực hiện, chúng em đã có cái nhìn tổng quan về khái niệm, công thức, ý tưởng, ưu nhược điểm của thuật toán Berkeley. Bên cạnh đó, chúng em cũng đã xây dựng chương trình và thực hiện thử nghiệm để kiểm tra hiệu suất của chương trình trong môi trường thực tế.

Tuy nhiên, như mọi nghiên cứu, đề tài vẫn còn nhiều khía cạnh có thể phát triển và cải tiến cũng như không khỏi tránh được những thiếu sót. Chúng em hy vọng rằng kết quả của đề tài này sẽ đóng góp một phần nhỏ vào lĩnh vực nghiên cứu các hệ thống phân tán, cũng như đồng bộ đồng hồ vật lý và thuật toán Berkeley, đồng thời mở ra các hướng nghiên cứu mới cho cộng đồng nghiên cứu trong tương lai.

– HẾT –