CSS533: Program 5

Shreevatsa Ganapathy Hegde

University of Washington, Bothell

sghegde@uw.edu

Prof. Munehiro Fukuda

9th June, 2025

Table of Contents

Documentation:

This program implements a distributed task processing system using ZooKeeper for coordination. The Client submits tasks as znodes under /tasks, while Workers register under /workers and pick up available tasks in a synchronized manner. Each Worker executes its assigned task and deletes the corresponding task znode upon completion. The Client monitors task completion and performs cleanup once all tasks are processed. The program demonstrates distributed coordination, task synchronization, and fault tolerance using ZooKeeper.

To implement the above program, the following modifications were done. Each modification is explained in detail below.

Key.lock(): This method implements a distributed locking mechanism using the /lock znode in ZooKeeper to synchronize task pickup among Workers. The method repeatedly attempts to create an ephemeral /lock znode. If the znode creation succeeds, the Worker has acquired the lock and can safely access and select a task. If the /lock znode already exists (meaning another Worker currently holds the lock), the method sets a watcher on /lock using zk.exists() with a lockWatcher. The Worker then waits by sleeping on a synchronization object until it is notified that the /lock znode has been deleted. Upon notification, the Worker retries acquiring the lock. This ensures that only one Worker at a time can enter the critical section for picking a task, preventing race conditions and ensuring correct task assignment across multiple Workers.

Worker.pickupTask(): This method implements the core logic for safely selecting a task from the pool of available tasks in ZooKeeper. The Worker first acquires the distributed lock by invoking lock() on the /lock znode to ensure exclusive access to the task list. It then retrieves the list of children under /tasks and sorts them to maintain consistent ordering. The Worker iterates over the task znodes and checks the status of each task by reading its data. If a task is found with the status "submitted," the Worker updates the task's data with the current system timestamp, effectively claiming the task for execution. If no "submitted" task is found, the Worker checks whether any task is overdue by comparing the current time with the timestamp stored in the task's data. If a task is overdue by more than 100 seconds, the Worker similarly updates its timestamp to claim it. The method ensures that only one Worker claims any given task at a time by holding the distributed lock during this process. Once a task is successfully claimed, the lock is released, and the task identifier is returned for execution. If no task can be safely claimed, the method returns either "job stalled" or null, depending on whether active tasks are still being processed.

Worker. runTask(): This method launches the GraphBridge program as an external process by constructing and executing a command that specifies the number of vertices to process. It captures and prints the output of GraphBridge to monitor execution progress. After the process completes, the Worker waits for its termination to ensure that the task is fully executed before proceeding to cleanup.

Worker.finishTask(): This method finalizes task processing by deleting the corresponding task znode from /tasks. It first retrieves the latest version of the task znode to ensure consistency, then calls zk.delete() with the correct version number to safely remove the znode. This signals that the task has been successfully completed and allows the Client to detect task completion through its watchers.

These were the necessary changes implemented to achieve the basic functionality of the program. The additional features and their corresponding enhancements will be discussed in the discussion section.

Source Code:

All the source code provided below represents the completed versions with the implemented additional features. I have included comments wherever I felt they would enhance clarity or understanding.

Key.java

```java
/**
 * Key is used among Workers to obtain the /lock znode for non-interruptibly
 * accessing the /tasks znode and its children several times.
 */

import java.io.Closeable;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import org.apache.zookeeper.AsyncCallback.DataCallback;
import org.apache.zookeeper.AsyncCallback.StatCallback;
import org.apache.zookeeper.AsyncCallback.StringCallback;
import org.apache.zookeeper.AsyncCallback.VoidCallback;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.EventType;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.AsyncCallback.ChildrenCallback;
import org.apache.zookeeper.KeeperException.Code;
import org.apache.zookeeper.data.Stat;

public class Key {
    private ZooKeeper zk;                       // ZooKeeper connected to Workers
    private static Object syncObject = null;  // Used to suspend a worker

    /**
     * Is the constructor that accepts a worker's ZooKeeper object and sets up
     * a synchronization object with itself.
     * @param zk_init a calling worker's ZooKeeper object
```

```
     */
    public Key( ZooKeeper zk_init ) {
        this.zk = zk_init;
        syncObject = this;
    }

    ////////////////////// Implement all methods below ////////////////////
    /**
     * This is your homework assignment.
     *
     * Tries to obtain the key on /lock. If not, waits on syncObject (= this).
     */
    public void lock( ) {
        while ( true ) {
            try {
                String lock = zk.create( "/lock",
                        null,
                        Ids.OPEN_ACL_UNSAFE,
                        CreateMode.EPHEMERAL );

                // upon a successful creation of /lock, I got the lock.
                if ( lock != null && lock.equals( "/lock" ) ) {
                    System.out.println( lock + " acquired" );
                    return;
                }
                System.err.println( lock + " error" ); // shouldn't happen

                } catch( KeeperException keeperexception ) {
                    // /lock has been already created by someone
                    System.err.println( "/lock locked already by someone else" );
                try {
                    // YOUR WORK: call zk.exists to set up lockWtacher( )
                    Stat stat = zk.exists("/lock", lockWatcher);
                    // YOUR WORK: if it's not null
                    // YOUR WORK: sleep here on syncObject
                    if (stat != null) {
                        // YOUR WORK: sleep here on syncObject
                        synchronized (this) {
                            this.wait();
                        }
                        System.out.println("/lock notified");
                    }
                    // YOUR WORK: else go back to the top of while( ) - not needed since it
will loop anyway
                } catch ( Exception another ) {
```

```
                    // YOUR WORK: print this exception and go back to the top of while( );
                    another.printStackTrace();
                }
            } catch( Exception others ) { }
        }
    }

    /**
     * This is your homework assignment.
     *
     * Is invoked upon a watcher event: when /lock is deleted.
     */
    Watcher lockWatcher = new Watcher( ) {
        public void process( WatchedEvent event ) {
            System.out.println( event.toString( ) );
            if ( event.getType( ) == EventType.NodeDeleted ) {
                // /lock was deleted
                // YOUR WORK: wake up myself who are sleeping on syncObject
                synchronized (Key.this) {
                        Key.this.notify();
                    }
                System.out.println( "/lock unlocked informed" );
            }
        }
    };

    /**
     * This is your hoemwork assignment.
     *
     * Unlocks the key on /lock. Simply deleted "/lock" znode.
     */
    public void unlock( ) {
        try {
            zk.delete( "/lock", 0 );
        } catch( Exception e ) {
            System.err.println( e.toString( ) );
            return;
        }
        System.out.println( "/lock released" );
    }
}
```

Worker.java

```
/**
 * Worker keeps picking up an available or an incomplete task from a bag of
 * tasks. Once all tasks are exhausted, it gets terminated.
 */

import java.io.Closeable;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import org.apache.zookeeper.AsyncCallback.DataCallback;
import org.apache.zookeeper.AsyncCallback.StatCallback;
import org.apache.zookeeper.AsyncCallback.StringCallback;
import org.apache.zookeeper.AsyncCallback.VoidCallback;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.EventType;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.AsyncCallback.ChildrenCallback;
import org.apache.zookeeper.KeeperException.Code;
import org.apache.zookeeper.data.Stat;

import java.io.*;

public class Worker implements Watcher, Closeable {
    private ZooKeeper zk;                       // ZooKeeper to join
    private String hostPort;                    // ZooKeeper's port
    private volatile boolean connected = false; // true if connected to zk
    private volatile boolean expired = false;   // true if session expired
    private String workerID = null;             // worker-000000000d (d=0-9)
    private Key key = null;                     // the key for /lock

    /**
     * Is the constructor that accepts ZooKeeper's IP addr/port to listen at.
```

```
     *
     * @param hostPort IP port Zookeeing is listening at.
     */
    public Worker( String hostPort ) {
    this.hostPort = hostPort;
    }

    /**
     * Joins ZooKeeper session at the port given through the constructor.
     * The session will be expired at 15 seconds for no communication.
     */
    public void startZK( ) throws IOException {
    zk = new ZooKeeper( hostPort, 15000, this );
    key = new Key( zk ); // creates a key to lock /lock znode.
    }

    /**
     * Implements Watcher.process( )
     */
    public void process( WatchedEvent e ) {
        System.out.println( e.toString( ) + ", " + hostPort );
        if( e.getType( ) == Event.EventType.None ) {
            switch ( e.getState( ) ) {
            case SyncConnected:
                /*
                 * Registered with ZooKeeper
                 */
                connected = true;
                break;
            case Disconnected:
                connected = false;
                break;
            case Expired:
                expired = true;
                connected = false;
                System.err.println( "Session expired" );
            default:
                break;
            }
        }
    }

    /**
     * Implements Closeable.close( )
     */
```

```
@Override
public void close( )
        throws IOException
{
    System.out.println( "Closing" );
    try{
        zk.close();
    } catch (InterruptedException e) {
        System.err.println( "ZooKeeper interrupted while closing" );
    }
}

/**
 * Checks if this worker is connected to ZooKeeper
 *
 * @return true if connected
 */
public boolean isConnected( ) {
    return connected;
}

/**
 * Checks if this worker's session was expired
 *
 * @return true if expired
 */
public boolean isExpired( ) {
    return expired;
}

/**
 * Is Worker's main logic.
 *
 * @param args[] args[0] is Zookeeper's IPaddr:IPport.
 */
public static void main( String args[] ) throws Exception {
    // memorize the ZooKeeper port
    Worker worker = new Worker( args[0] );

    // start ZooKeeper
    worker.startZK( );

    // wait until connected to ZooKeeper
    System.out.println( "wait for connection" );
    while( !worker.isConnected( ) ) {
```

```
                Thread.sleep( 100 );
        }
        System.out.println( "connected" );

        // register my name under /workers
        worker.register( );

        // fall into a task processing cycle.
        for ( String taskID = null;
            ( taskID = worker.pickupTask( ) ) != null; ) {
            // taskID should be "task-000000000d" where d = 0-9 if available.
            // otherwise "job stalled" that indicates a potential worker crash.
            if ( taskID.equals( "job stalled" ) ) {
            Thread.sleep( 10000 );
            continue;
            }
            System.out.println( taskID + " in progress by " + worker.getID( ) );

            // run the task and remove it from /task znode.
            worker.runTask( taskID );
            worker.finishTask( taskID );
        }

    }

    /**
     * Returns this worker's ID: worker-000000000d (where d = 0-9)
     *
     * @return this worker's ID.
     */
    private String getID( ) {
    return workerID;
    }

    ////////////////////// Implement all methods below ////////////////////
    /**
     * This is your homework assignment
     *
     * Registers this worker under /workers znode. The worker should be
     * identified as /workers/worker-000000000d where d=0-9. It's
     * ephemral and stored in workerID.
     */
    private void register( ) throws Exception {
    workerID = zk.create( "/workers/worker-",
                  null,
```

```
                Ids.OPEN_ACL_UNSAFE,
                CreateMode.EPHEMERAL_SEQUENTIAL );
    System.out.println( workerID + " registered" );
    }

    /**
     * This is your homework assignment.
     * Gets a list of tasks from /tasks, checks each task-000000000d where
     * d=0-9, picks up one if its data is "submitted", otherwise examins
     * if its timestamp (i.e., data) gets expired beyond 100 seconds, and
     * if so picks it up as updating its timestamp to the present. This
     * method returns task-000000000d as a task ID to execute or "job
     * stalled" if all remaining tasks are being executed below 100 seconds.
     * If no more tasks are found under /tasks, the method returns null.
     *
     * @returns task-00000000d where d=0-9, as a taskID to execute.
     */
    private String pickupTask( ) {
        boolean jobStalled = false;
        // YOUR WORK: lock
        try {
            key.lock(); // lock znode to prevent other workers from picking up tasks.

            List<String> children= zk.getChildren( "/tasks", taskWatcher, null );
            Collections.sort(children); // sort tasks by their names
            for ( int i = 0; children != null && i < children.size( ); i++ ) {
                System.out.println( children.get( i ) );

                Stat taskStat = new Stat( );
                String taskStatus = new String( zk.getData( "/tasks/" + children.get( i ),
false, taskStat ) );
                System.out.println( taskStatus + "'s version: " + taskStat.getVersion( ) );
                if ( taskStatus.equals( "submitted" ) ) {
                    // get this task
                    Long currTime = System.currentTimeMillis( );

                    // update the task's timestamp to the current time
                    zk.setData("/tasks/" + children.get(i),
String.valueOf(currTime).getBytes(), taskStat.getVersion());
                    // re-fetch the task status
                    taskStatus = new String(zk.getData("/tasks/" + children.get(i), false,
taskStat));
                    System.out.println( taskStatus );
                    // unlock znode after picking up the task
                    key.unlock();
```

```
                    return children.get( i );
                }
                else {
                    // check if this task is overdue.
                    // YOUR WORK: get the current time into currTime.
                    // YOUR WORK: get the submitted time into pastTime.
                    // YOUR WORK: compute diff
                    Long currTime = System.currentTimeMillis();
                    Long pastTime = Long.parseLong(taskStatus);
                    Long diff = currTime - pastTime;

                    System.out.println( "currTime = " + currTime +
                        ", pastTime = " + pastTime +
                        ", diff = " + diff );

                    if ( diff > 100000 ) { // overdue
                        System.out.println( "overdue" );
                        // YOUR WORK: zk.setData( ) to write currTime
                        zk.setData("/tasks/" + children.get(i),
String.valueOf(currTime).getBytes(), taskStat.getVersion());

                        taskStatus = new String(zk.getData("/tasks/" + children.get(i), false,
taskStat));
                        System.out.println( taskStatus );
                        // YOUR WORK: unlock
                        key.unlock();
                        return children.get( i );
                    }
                    else {
                    jobStalled = true;
                    }
                }
            }
        } catch( Exception e ) {
            System.err.println( e.toString( ) );
        }
        // YOUR WORK: unlock
        try {
            key.unlock();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ( jobStalled ) ? "job stalled" : null;
    }
```

```java
    /**
     * Watches any changes of /tasks. Just prints out an incoming watch event
     */
    Watcher taskWatcher = new Watcher( ) {
        public void process( WatchedEvent e ) {
            System.out.println( e.toString( ) );
        }
    };

    /**
     * This is your homework assignment.
     *
     * Receives a taskID, (i.e., task-00000000d where d = 0-9), converts it
     * to 500,000 - 5,0000,000 vertices, and runs:<br>
     * java -Xss512m GraphBridge vertices
     *
     * @param taskID a task obtained from the bag of tasks, from which the
     *        worker runs "java -Xss512m GraphBridge (taskID + 1) * 1000000/2
     */
    private void runTask( String taskID ) throws Exception {
        // compute the number of vertices from taskID
        // int vertices = ( Integer.parseInt( taskID.split( "-" )[1] ) + 1 ) * 1000000 / 2;
        // System.out.println( "vertices = " + vertices );
        Stat stat = new Stat();
        String taskData = new String(zk.getData("/tasks/" + taskID, false, stat));
        int vertices;
        try {
            // vertices = Integer.parseInt(taskData);
            vertices = 150; // test the grapg generation with a small number of vertices
        } catch (NumberFormatException e) {
            // Fallback → if taskData is not a valid number, use default formula
            System.out.println("Warning: taskData not a number → using default formula");
            vertices = ( Integer.parseInt( taskID.split( "-" )[1] ) + 1 ) * 1000000 / 2;
        }

        System.out.println("vertices = " + vertices);

        // creates a task array
        String[] args = { "java", "-Xss512m", "GraphBridge", ( new Integer( vertices )
).toString( ) };

        // YOUR WORK: launch a new process to this task by passing args to exec( )
        Process proc = Runtime.getRuntime().exec(args);
        // YOUR WORK: retrieve this process inputstream into BufferedReader is
        BufferedReader is = new BufferedReader(new InputStreamReader(proc.getInputStream()));
```

```
        for ( String line = null; ( line = is.readLine( ) ) != null; )
            System.out.println( line ); // keep writing the outputs to stdout.

        // YOUR WORK: wait for the termination of this task
        proc.waitFor();
        String taskNumber = taskID.replace("task-", "");

        Process p = Runtime.getRuntime().exec("dot -Tpng graph.dot -o graph-task-" +
taskNumber + ".png");
        // Capture dot stderr → to detect any errors
        BufferedReader err = new BufferedReader(new InputStreamReader(p.getErrorStream()));
        String errLine;
        while ((errLine = err.readLine()) != null) {
            System.err.println("DOT ERROR: " + errLine);
        }

        p.waitFor();
        System.out.println("Graph image saved at: " + new File("graph-task-" + taskNumber +
".png").getAbsolutePath());

    }

    /**
     * This is your homework assignment.
     *
     * Declares a completion of a given task
     * @param taskID the ID of a task to be completed
     */
    private void finishTask( String taskID ) throws Exception {
        try {
            // get the latest version of this taskID
            Stat taskStat = new Stat( );
            // YOUR WORK: call zk.getData( ) to get its taskID's state into taskStat
            zk.getData("/tasks/" + taskID, false, taskStat);
            // YOUR WORK: call zk.delete( ) to delete this task by passing the up-to-date
version.
            zk.delete("/tasks/" + taskID, taskStat.getVersion());
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Client.java:

Client.java was changed only as part of additional features.

```
/**
 * Client creates /workers and /tasks znodes; submits 10 tasks, each named
 * /tasks/task-000000000d where d = 0-9; waits until /workers and /tasks have
 * no more children; and deletes these two znodes.
 */

import java.io.Closeable;
import java.io.IOException;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import org.apache.zookeeper.AsyncCallback.DataCallback;
import org.apache.zookeeper.AsyncCallback.StatCallback;
import org.apache.zookeeper.AsyncCallback.StringCallback;
import org.apache.zookeeper.AsyncCallback.VoidCallback;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.EventType;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.AsyncCallback.ChildrenCallback;
import org.apache.zookeeper.KeeperException.Code;
import org.apache.zookeeper.data.Stat;

public class Client implements Watcher, Closeable {
    private ZooKeeper zk;                       // ZooKeeper to join
    private String hostPort;                    // ZooKeeper's port
    private volatile boolean connected = false; // true if connected to zk
    private volatile boolean expired = false;   // true if session expired

    /**
     * Is the constructor that accepts ZooKeeper's IP addr/port to listen at.
     *
     * @param hostPort IP address:IP port ZooKeeper will listen at
```

```
 */
public Client( String hostPort ) {
this.hostPort = hostPort;
}

/**
 * Joins ZooKeeper session at the port given through the constructor.
 * The session will be expired at 15 seconds for no communication.
 */
public void startZK( ) throws IOException {
zk = new ZooKeeper( hostPort, 15000, this );
}

/**
 * Implements Watcher.process( )
 */
public void process( WatchedEvent e ) {
    System.out.println( e.toString( ) + ", " + hostPort );
    if( e.getType( ) == Event.EventType.None ) {
        switch ( e.getState( ) ) {
        case SyncConnected:
            /*
             * Registered with ZooKeeper
             */
            connected = true;
            break;
        case Disconnected:
            connected = false;
            break;
        case Expired:
            expired = true;
            connected = false;
            System.err.println( "Session expired" );
        default:
            break;
        }
    }
}

/**
 * Implements Closeable.close( )
 */
@Override
public void close( )
        throws IOException
```

```
 {
     System.out.println( "Closing" );
     try{
         zk.close();
     } catch (InterruptedException e) {
         System.err.println( "ZooKeeper interrupted while closing" );
     }
 }

/**
  * Checks if the client is connected to ZooKeeper
  *
  * @return true if connected
  */
 public boolean isConnected() {
     return connected;
 }

 /**
  * Checks if the client's session was expired
  *
  * @return true if expired
  */
 public boolean isExpired() {
     return expired;
 }

  /**
  * Is Client's main logic.
  *
  * @param args[] args[0] is Zookeeper's IPaddr:IPport.
  */
 public static void main( String args[] ) throws Exception {
     // memorize the ZooKeeper port
     for (String arg : args) {

         System.out.println("arg = " + arg);
     }
     Client client = new Client( args[0] );
     client.nTasksSubmitted = (args.length > 1) ? Integer.parseInt(args[1]) : 10;


     // start ZooKeeper
     client.startZK( );
```

```
        // wait until connected to ZooKeeper
        System.out.println( "wait for connection" );
        while( !client.isConnected( ) ) {
                Thread.sleep( 100 );
        }
        System.out.println( "connected" );

        client.createWorkerNode( ); // create /workers
        client.createBagOfTasks( ); // create /tasks/task-0000000000d (d=0-9)
        client.confirmEmptyBag( );  // delete /workers and /tasks
    }

    private String pid = Long.toHexString( ProcessHandle.current( ).pid( ) );
    private int nTasksSubmitted;
    private int nTasksCompleted = 0;

    /**
     * Creates the /workers znode synchronously.
     */
    private void createWorkerNode( ) throws Exception {
        // Check if /workers already exists
        Stat stat = zk.exists("/workers", false);
        if (stat == null) {
            // Create /workers znode
            zk.create("/workers",
                    pid.getBytes(),
                    Ids.OPEN_ACL_UNSAFE,
                    CreateMode.PERSISTENT);
            System.out.println("/workers created by client " + pid);
        } else {
            System.out.println("/workers already exists");
        }
    }

    /**
     * Creates the /tasks znode synchronously and thereafter submits
     * /tasks/task-000000000d where d=0-9. Each task has "submitted" as its
     * data
     */
    private void createBagOfTasks( ) throws Exception {
        System.out.println("nTasksSubmitted = " + nTasksSubmitted);
        // Check if /tasks already exists
        Stat stat = zk.exists("/tasks", false);

        if (stat != null) {
```

```
            // delete old /tasks and all children
            List<String> oldTasks = zk.getChildren("/tasks", false, null);
            for (String task : oldTasks) {
                zk.delete("/tasks/" + task, 0);
            }
            zk.delete("/tasks", 0);
            System.out.println("/tasks deleted before re-creating");
        }
        // Create /tasks znode
        zk.create("/tasks",
                pid.getBytes(),
                Ids.OPEN_ACL_UNSAFE,
                CreateMode.PERSISTENT);


        for (int i = 0; i < nTasksSubmitted; i++) {
            // Random vertices between 500000 and 5000000
            int vertices = 500000 + (int)(Math.random() * 4500000);
            // int vertices = 150; // for testing purposes to generate png files quickly

            String taskID = zk.create("/tasks/task-",
                                      String.valueOf(vertices).getBytes(),
                                      Ids.OPEN_ACL_UNSAFE,
                                      CreateMode.PERSISTENT_SEQUENTIAL);
            System.out.println(taskID + " submitted");
            System.out.println("/tasks created by client " + pid);
            System.out.println(taskID + " submitted with vertices = " + vertices);
        }
    }

    /**
     * Launches a watcher for each task, confirms all tasks have been deleted,
     * deletes /tasks, checks all workers are gone, and finally deletes
     * /workers.
     */
    private void confirmEmptyBag( ) throws Exception {

        List<String> tasks = zk.getChildren("/tasks", false, null);
        Collections.sort(tasks);
        for (String task : tasks) {
            zk.exists("/tasks/" + task, taskWatcher);
            System.out.println("/tasks/" + task + " under watch");
        }

        // nTasksCompleted is incremented by each task watcher, so that the
```

```
        // main thread waits until nTasksCompleted comes up to nTasksSubmitted.
        while ( nTasksCompleted < nTasksSubmitted )
            Thread.sleep( 1000 );

        // All tasks have been processed and deleted. So, it's time to delete
        // /tasks.
        System.out.println( "all tasks deleted" );
        zk.delete( "/tasks", 0 );

        // Waits untill all workers disappeared. Then, we can delete /workers.
        while ( true ) {
            List<String> workers = zk.getChildren( "/workers", false, null );
            if ( workers == null || workers.size( ) == 0 )
            break;
        }
        System.out.println( "all workers signed off" );
        zk.delete( "/workers", 0 );
    }

    /**
     * Watches any changes of /task and increments nTaskCompleted if the
     * change was a task-deleting event. Otherwise, this method reschedules
     * a task watcher for detecting future events.
     */
    Watcher taskWatcher = new Watcher( ) {
        public void process( WatchedEvent event ) {
        System.out.println( event.toString( ) );
            if( event.getType( ) == EventType.NodeDeleted ) {
        nTasksCompleted++;
        System.out.println( "deleted" );
            }
        else {
        try {
            zk.exists( event.getPath( ), taskWatcher );
        } catch( Exception exception ) { }
        }
        }
    };

}
```

GraphBridge.java

This file was modified for the graphical output generation additional feature. The explanation of this file is discussed in the discussion section. The code responsible for the visualization is at the end of main method.

```
/**
 * This is a graph bridge program to be used for testing CSS533's homework assignment. You
need to launch this program
 * from Worker.java through ZooKeeper.
 *  java -Xss512M GraphBridge 100
 *  100's # brdiges = 90, time elapsed = 1 msec
 *  java -Xss512M GraphBridge 1000
 *  1000's # brdiges = 2, time elapsed = 6 msec
 *  java -Xss512M GraphBridge 10000
 *  10000's # brdiges = 14, time elapsed = 29 msec
 *  java -Xss512M GraphBridge 100000
 *  100000's # brdiges = 200, time elapsed = 302 msec
 *  java -Xss512M GraphBridge 1000000
 *  1000000's # brdiges = 1980, time elapsed = 3799 msec
 *  java -Xss512M GraphBridge 2000000
 *  2000000's # brdiges = 3973, time elapsed = 8736 msec
 *  java -Xss512M GraphBridge 3000000
 *  3000000's # brdiges = 6096, time elapsed = 13874 msec
 *  java -Xss512M GraphBridge 4000000
 *  4000000's # brdiges = 8063, time elapsed = 22957 msec
 *  java -Xss512M GraphBridge 5000000
 *  5000000's # brdiges = 10108, time elapsed = 32005 msec
 */

import java.io.*;
import java.util.*;

public class GraphBridge {
    public int nBridges = 0;                // #bridges
    private int nHops = 0;                  // hop count
    private int[] low;                      // low[v] = lowest preorder of any vertex
connected to v
    private int[] disc;                     // disc[v] = order in which dfs examines v

    public static boolean enbPrint = false; // print messages if it's true

    /**
     * Is the constructor that initializes low[0] - low[nVertices] and disc[0] -
disc[nVertices] and thereafter
     * finds out all graph bridges
```

```
     *
     * @param V an array of Integer lists, each maintaining all the neighbors of a different
vertex
     * @param E an array of Integer lists, each maintinaing all edge weights emanating from a
different vertex
     */
    public GraphBridge( ArrayList<Integer>[] V, ArrayList<Integer>[] E ) {
        low = new int[ V.length ];         // low[v]: lowest preorder of any vertex connected
to v
        disc = new int[ V.length ];        // disc[v]: order in which dfs examines

        // initialization
        for ( int i = 0; i < low.length; i++ )
            low[i] = -1;
        for ( int i = 0; i < disc.length; i++ )
            disc[i] = -1;

        // conduct depth-first traverses
        for ( int i = 0; i < V.length; i++ )
            if ( disc[i] == -1 ) // not discovered yet
            dfs( V, E, i, i );

        // print out low[] and disc[] of each verte.
        if ( enbPrint ) {
            for ( int i = 0; i < V.length; i++ )
            System.out.println( "low[" + i + "] = " + low[i] + ", disc[" + i + "] = " +
disc[i] );
        }
    }

     /**
     * Traverse G( V, E ) in a depth-first manner. Upon encountering a circle, the control
tracks back to the origin.
     *
     * @param V       an array of Integer lists, each maintaining all the neighbors of a
different vertex
     * @param E      an array of Integer lists, each maintinaing all edge weights emanating
from a different vertex
     * @param curr   the current vertex to start a dfs
     * @param parent the parent of the current vertex
     */
    private void dfs( ArrayList<Integer>[] V, ArrayList<Integer>[] E, int curr, int parent ) {
        if ( enbPrint ) System.out.println( "dfs( " + curr + ", " + parent + " ) ------------
----------" );
        disc[curr] = low[curr] = ++nHops; // hop increment
```

```
        if ( enbPrint ) System.out.println( "disc[" + curr + "] = " + disc[curr] + ", low[" +
curr + "] = " + low[curr] );

        ArrayList<Integer> vertices = V[curr]; // all neighbors from the current vertex.
        for ( int i = 0; i < vertices.size( ); i++ ) {
            int next = vertices.get( i ); // get a next neighbor.
            if ( next == parent )
            continue; // we don't want to go back to the parent.
            if ( disc[next] == -1 ) {     // next hasn't been discoverred yet
            dfs( V, E, next, curr );  // dig into this neighbor.
            low[curr] = Math.min( low[next], low[curr] ); // upon a back-track, if low[next]
is smaller, I'm on a circle.
            if ( disc[curr] < low[next] ) { // if i'm on a circle, disc[curr] >= low[next]
                if ( enbPrint ) System.out.println( curr + " - " + next + " is a bridge" );
                nBridges++;
            }
            }
            else
            low[curr] = Math.min( low[curr], disc[next] ); // find a circle. in most cases,
disc[next] will be smaller.
        }
    }

    /**
     * Generates a random graph with arg[0] vertices and computes the number of bridges of
this graph.
     *
     * @param args arg[0] is the number of vertices. arg[1] is optional to allow messages to
be printed.
     */
    public static void main( String[] args ) {
        long startTime = System.currentTimeMillis( ); // start a timer

        int nNodes = Integer.parseInt( args[0] );       // # vertices
        enbPrint = ( args.length > 1 ) ? true : false;  // allow debugging messages to be
printed

        if ( enbPrint ) System.out.println( "nNodes = " + nNodes +"\n" );
        // create an array of Integer lists, each manitaining all the neighbors of a different
vertex
        // create an array of Integer lists, each manitaining all the edge weights from a
different vertex
        ArrayList<Integer>[] graph_neighbors = (ArrayList<Integer>[]) new ArrayList[nNodes];
        ArrayList<Integer>[] graph_distances = (ArrayList<Integer>[]) new ArrayList[nNodes];
```

```
for ( int i = 0; i < nNodes; i++ ) {
    graph_neighbors[i] = new ArrayList<Integer>( );
    graph_distances[i] = new ArrayList<Integer>( );
}

// All nodes have the same upper limit
// if nNodes is small below 1K, #neighbors will be 10%: 1-10
// otherwise, #neighbors is always 10
Random rand = new Random( 0 );
int upperLimit = ( nNodes < 1000 ) ? ( int )( nNodes * 0.01 ) : 10;
upperLimit = ( upperLimit == 0 ) ? 1 : upperLimit;

for ( int i = 0; i < nNodes; i++ ) {
    // Generate # neighbors from node i
    int nNeighbors = rand.nextInt( upperLimit );
    if ( nNeighbors == 0 ) nNeighbors = 1;

    // Generate neighboring nodes from node i
    for ( int j = 0; j < nNeighbors; j++ ) {
    int neighbor = rand.nextInt( nNodes );
    int distance = rand.nextInt( nNodes );
    if ( distance == 0 ) distance = 1;

    // check if this is a self-directed link
    if ( neighbor == i )
        continue;
    // check if this is a duplication
    if ( graph_neighbors[i].indexOf( new Integer( neighbor ) ) == -1 ) {
        // store my neighbor and its distance
        graph_neighbors[i].add( new Integer( neighbor ) );
        graph_distances[i].add( new Integer( distance ) );

        if ( graph_neighbors[neighbor].indexOf( new Integer( i ) ) == -1 ) {
        // make sure that a dual edge is created from my neighbor back to me.
        graph_neighbors[neighbor].add( new Integer( i ) );
        graph_distances[neighbor].add( new Integer( distance ) );
        }
    }
    }
}

// print out all the neighbors and edge weights from each vertex
if ( enbPrint ) {
    for ( int i = 0; i < graph_neighbors.length; i++ ) {
```

```
            System.out.println( "Vertex " + i + ":" );
            for ( int j = 0; j < graph_neighbors[i].size( ); j++ ) {
                System.out.println( "\t to " + graph_neighbors[i].get(j) + " with distance " +
graph_distances[i].get(j) );
            }
            }
        }

        // compute # bridges of a given graph.
        GraphBridge bridge = new GraphBridge( graph_neighbors, graph_distances );

        long endTime = System.currentTimeMillis( ); // finish the timer
        System.out.println( args[0] + "'s # brdiges = " + bridge.nBridges + ", time elapsed =
" + ( endTime - startTime ) + " msec" );
        // generate a graph.dot file for visualization
        try {
            PrintWriter out = new PrintWriter("graph.dot");
            out.println("digraph G {");

            for (int i = 0; i < graph_neighbors.length; i++) {
                for (int j = 0; j < graph_neighbors[i].size(); j++) {
                    int neighbor = graph_neighbors[i].get(j);
                    int distance = graph_distances[i].get(j);

                    if (i < neighbor) {
                        out.println("    " + i + " -> " + neighbor + " [label=\"" + distance +
"\"];");
                    }
                }
            }

            out.println("}");
            out.close();

            System.out.println("graph.dot file generated.");
        } catch (Exception e) {
            System.err.println("Error writing graph.dot: " + e);
        }

    }
}
```

Execution Outputs:

Base Implementation:

Three scenarios were used to test the base implementation. Below are the respective outputs for each.

Run at least three workers.

Client Terminal:

```
^C[sghegde@cssmpi23 prog5]$ ./run.sh Client cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers created by client 8e498
/tasks created by client 8e498
/tasks/task-0000000000 submitted
/tasks/task-0000000001 submitted
/tasks/task-0000000002 submitted
/tasks/task-0000000003 submitted
/tasks/task-0000000004 submitted
/tasks/task-0000000005 submitted
/tasks/task-0000000006 submitted
/tasks/task-0000000007 submitted
/tasks/task-0000000008 submitted
/tasks/task-0000000009 submitted
/tasks/task-0000000000 under watch
/tasks/task-0000000001 under watch
/tasks/task-0000000002 under watch
/tasks/task-0000000003 under watch
/tasks/task-0000000004 under watch
/tasks/task-0000000005 under watch
/tasks/task-0000000006 under watch
/tasks/task-0000000007 under watch
/tasks/task-0000000008 under watch
/tasks/task-0000000009 under watch
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000000
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000000
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000001
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000001
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000002
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000002
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000003
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000003
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000004
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000004
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000005
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000005
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000006
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000007
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000006
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000008
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000009
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000008
deleted
```

Zookeeper start screen:

```
[sghegde@cssmpi3 bin]$ ./zkServer.sh start
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /home/NETID/sghegde/storm/apache-zookeeper-3.7.0-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Worker1:

```
[sghegde@cssmpi2 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000001 registered
/lock acquired
task-0000000006
1749438392950's version: 1
currTime = 1749438406591, pastTime = 1749438392950, diff = 13641
task-0000000007
submitted's version: 0
1749438406598
/lock released
task-0000000007 in progress by /workers/worker-0000000001
vertices = 4000000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
4000000's # brdiges = 8063, time elapsed = 55184 msec
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438467405, pastTime = 1749438422365, diff = 45040
/lock released
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438477414, pastTime = 1749438422365, diff = 55049
/lock released
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438487423, pastTime = 1749438422365, diff = 65058
/lock released
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438497434, pastTime = 1749438422365, diff = 75069
/lock released
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438507443, pastTime = 1749438422365, diff = 85078
/lock released
/lock acquired
task-0000000009
1749438422365's version: 1
currTime = 1749438517453, pastTime = 1749438422365, diff = 95088
/lock released
```

Worker2:

```
[sghegde@cssmpi1 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000002 registered
/lock acquired
task-0000000007
1749438406598's version: 1
currTime = 1749438422357, pastTime = 1749438406598, diff = 15759
task-0000000008
1749438417679's version: 1
currTime = 1749438422363, pastTime = 1749438417679, diff = 4684
task-0000000009
submitted's version: 0
1749438422365
/lock released
task-0000000009 in progress by /workers/worker-0000000002
vertices = 5000000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
5000000's # brdiges = 10108, time elapsed = 107053 msec
/lock acquired
/lock released
[sghegde@cssmpi1 prog5]$
```

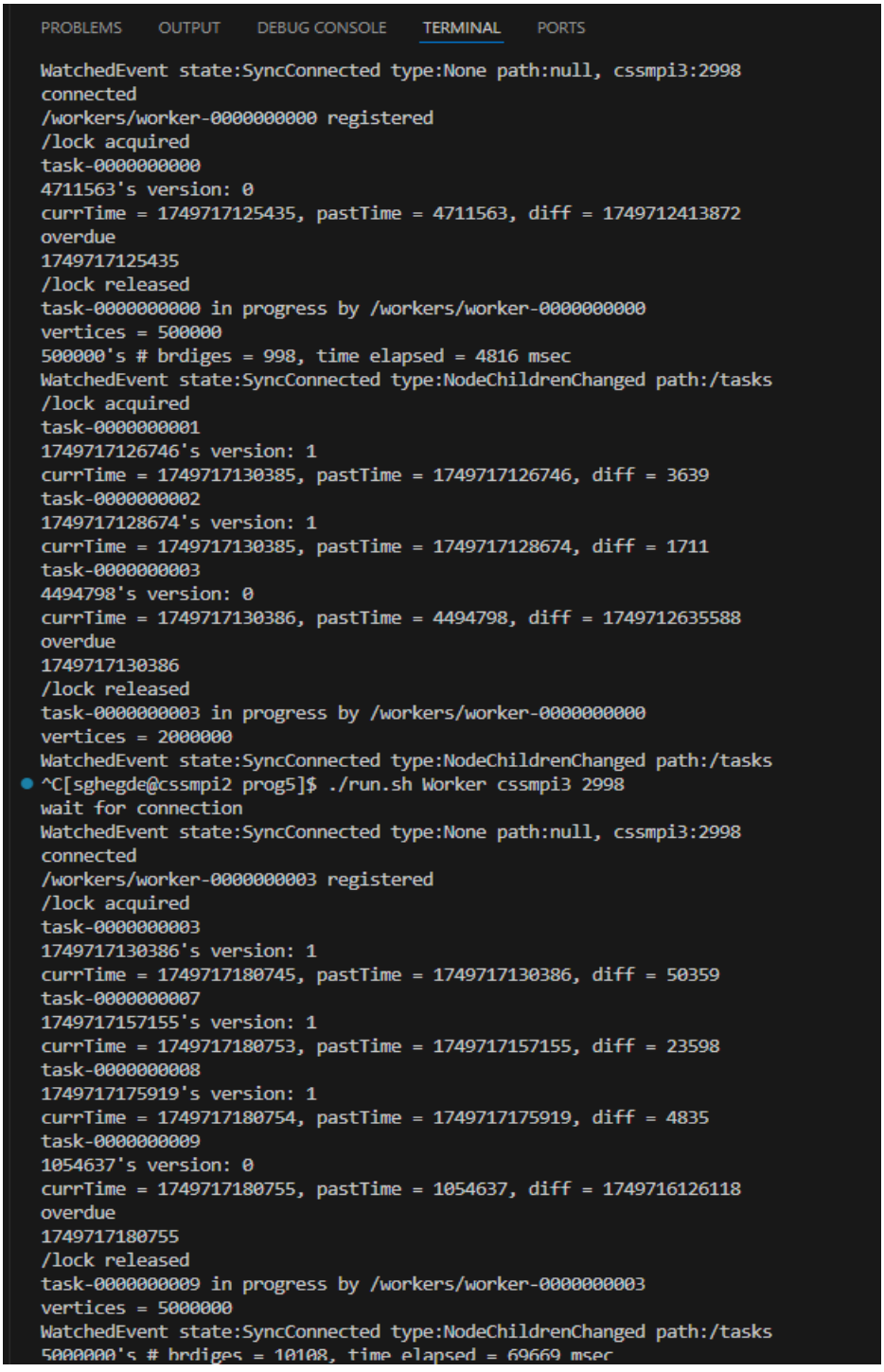
Worker3:

```
[sghegde@cssmpi4 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000000 registered
/lock acquired
task-0000000000
submitted's version: 0
1749438323291
/lock released
task-0000000000 in progress by /workers/worker-0000000000
vertices = 500000
500000's # brdiges = 998, time elapsed = 2804 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000001
submitted's version: 0
1749438326197
/lock released
task-0000000001 in progress by /workers/worker-0000000000
vertices = 1000000
1000000's # brdiges = 1980, time elapsed = 5940 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000002
submitted's version: 0
1749438332235
/lock released
task-0000000002 in progress by /workers/worker-0000000000
vertices = 1500000
1500000's # brdiges = 2941, time elapsed = 8963 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000003
submitted's version: 0
1749438341300
/lock released
task-0000000003 in progress by /workers/worker-0000000000
vertices = 2000000
2000000's # brdiges = 3973, time elapsed = 11915 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
```

Kill at least one of the nodes before all tasks are finished.

&

Rerun at least one new worker.

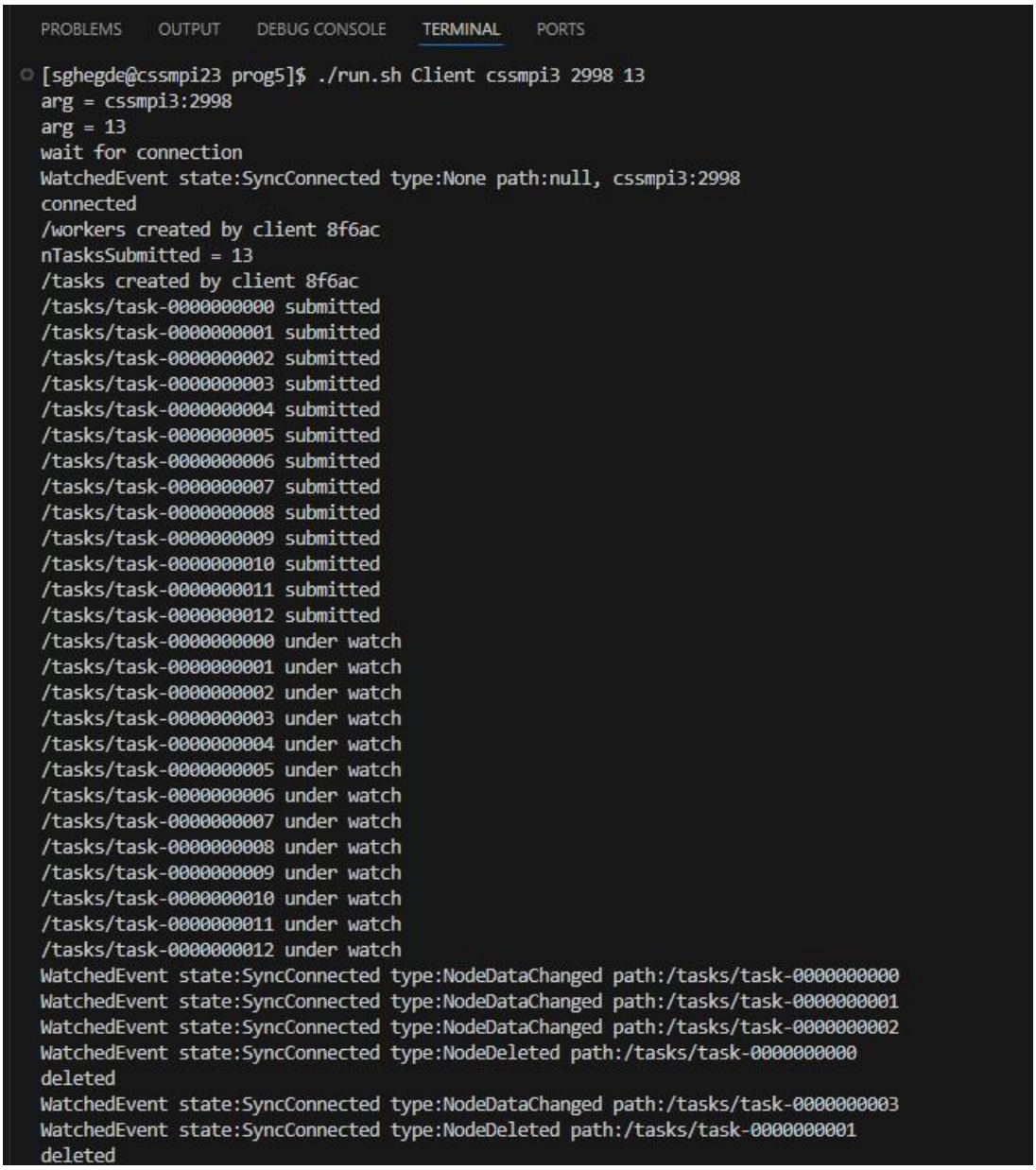


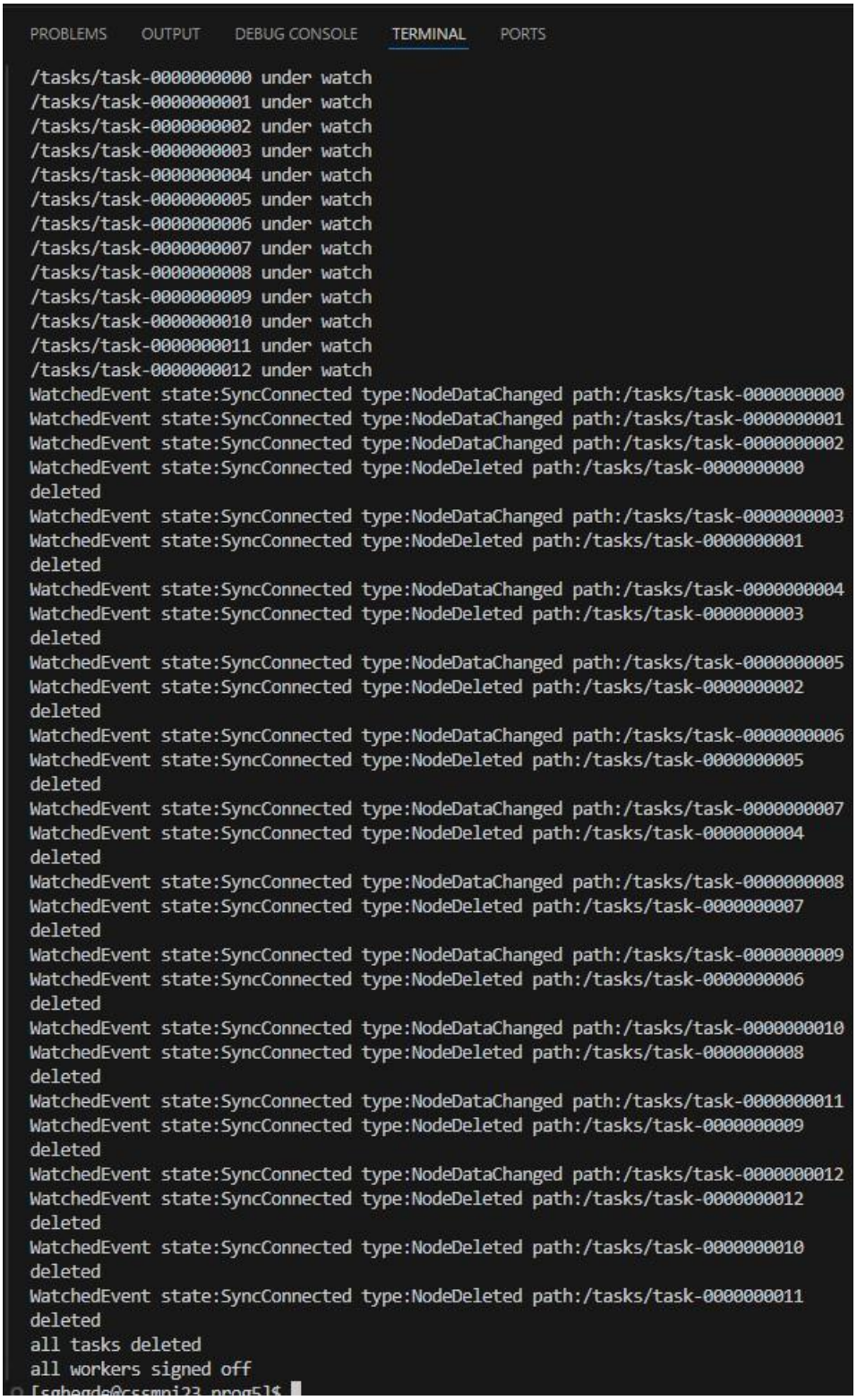
Additional features:

Three additional features were implemented as below.

Run any number of tasks rather than only 10 tasks.

Client screen:




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
/tasks/task-0000000000 under watch
/tasks/task-0000000001 under watch
/tasks/task-0000000002 under watch
/tasks/task-0000000003 under watch
/tasks/task-0000000004 under watch
/tasks/task-0000000005 under watch
/tasks/task-0000000006 under watch
/tasks/task-0000000007 under watch
/tasks/task-0000000008 under watch
/tasks/task-0000000009 under watch
/tasks/task-0000000010 under watch
/tasks/task-0000000011 under watch
/tasks/task-0000000012 under watch
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000000
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000001
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000002
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000000
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000003
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000001
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000004
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000003
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000005
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000002
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000006
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000005
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000007
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000004
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000008
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000007
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000009
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000006
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000010
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000008
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000011
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000009
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000012
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000012
deleted
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000010
deleted
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000011
deleted
all tasks deleted
all workers signed off

Workers:

```
task-0000000006 in progress by /workers/worker-0000000002
vertices = 3500000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
3500000's # brdiges = 7063, time elapsed = 48966 msec
/lock acquired
task-0000000008
1749458974525's version: 1
currTime = 1749459008771, pastTime = 1749458974525, diff = 34246
task-0000000009
1749459001398's version: 1
currTime = 1749459008772, pastTime = 1749459001398, diff = 7374
task-0000000010
submitted's version: 0
1749459008774
/lock released
task-0000000010 in progress by /workers/worker-0000000002
vertices = 5500000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
5500000's # brdiges = 11091, time elapsed = 89900 msec
/lock acquired
task-0000000011
1749459030930's version: 1
currTime = 1749459098916, pastTime = 1749459030930, diff = 67986
/lock released
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
```

```
task-0000000011
1749459030930's version: 1
currTime = 1749459039604, pastTime = 1749459030930, diff = 8674
task-0000000012
submitted's version: 0
1749459039605
/lock released
task-0000000012 in progress by /workers/worker-0000000000
vertices = 6500000
6500000's # brdiges = 13129, time elapsed = 53280 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000010
1749459008774's version: 1
currTime = 1749459096869, pastTime = 1749459008774, diff = 88095
task-0000000011
1749459030930's version: 1
currTime = 1749459096870, pastTime = 1749459030930, diff = 65940
/lock released
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000011
1749459030930's version: 1
currTime = 1749459106877, pastTime = 1749459030930, diff = 75947
/lock released
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
org.apache.zookeeper.KeeperException$NoNodeException: KeeperErrorCode = NoNode for /tasks
/lock released
[sghegde@cssmpi4 prog5]$
```

```
task-0000000008 in progress by /workers/worker-0000000001
vertices = 4500000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
4500000's # brdiges = 9199, time elapsed = 53938 msec
/lock acquired
task-0000000009
1749459001398's version: 1
currTime = 1749459030928, pastTime = 1749459001398, diff = 29530
task-0000000010
1749459008774's version: 1
currTime = 1749459030929, pastTime = 1749459008774, diff = 22155
task-0000000011
submitted's version: 0
1749459030930
/lock released
task-0000000011 in progress by /workers/worker-0000000001
vertices = 6000000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
6000000's # brdiges = 11986, time elapsed = 77430 msec
/lock acquired
/lock released
[sghegde@cssmpi2 prog5]$
```

Run any applications with any problem size.

Client Screen:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[sghegde@cssmpi23 prog5]$ ./run.sh Client cssmpi3 2998 4
arg = cssmpi3:2998
arg = 4
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers already exists
nTasksSubmitted = 4
/tasks/task-0000000000 submitted
/tasks created by client 8f972
/tasks/task-0000000000 submitted with vertices = 1901752
/tasks/task-0000000001 submitted
/tasks created by client 8f972
/tasks/task-0000000001 submitted with vertices = 770639
/tasks/task-0000000002 submitted
/tasks created by client 8f972
/tasks/task-0000000002 submitted with vertices = 1985943
/tasks/task-0000000003 submitted
/tasks created by client 8f972
/tasks/task-0000000003 submitted with vertices = 670774
/tasks/task-0000000000 under watch
/tasks/task-0000000001 under watch
/tasks/task-0000000002 under watch
/tasks/task-0000000003 under watch
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000000
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000001
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000002
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000000
deleted
WatchedEvent state:SyncConnected type:NodeDataChanged path:/tasks/task-0000000003
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000001
deleted
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000002
deleted
WatchedEvent state:SyncConnected type:NodeDeleted path:/tasks/task-0000000003
deleted
all tasks deleted
all workers signed off
[sghegde@cssmpi23 prog5]$
```

Workers:

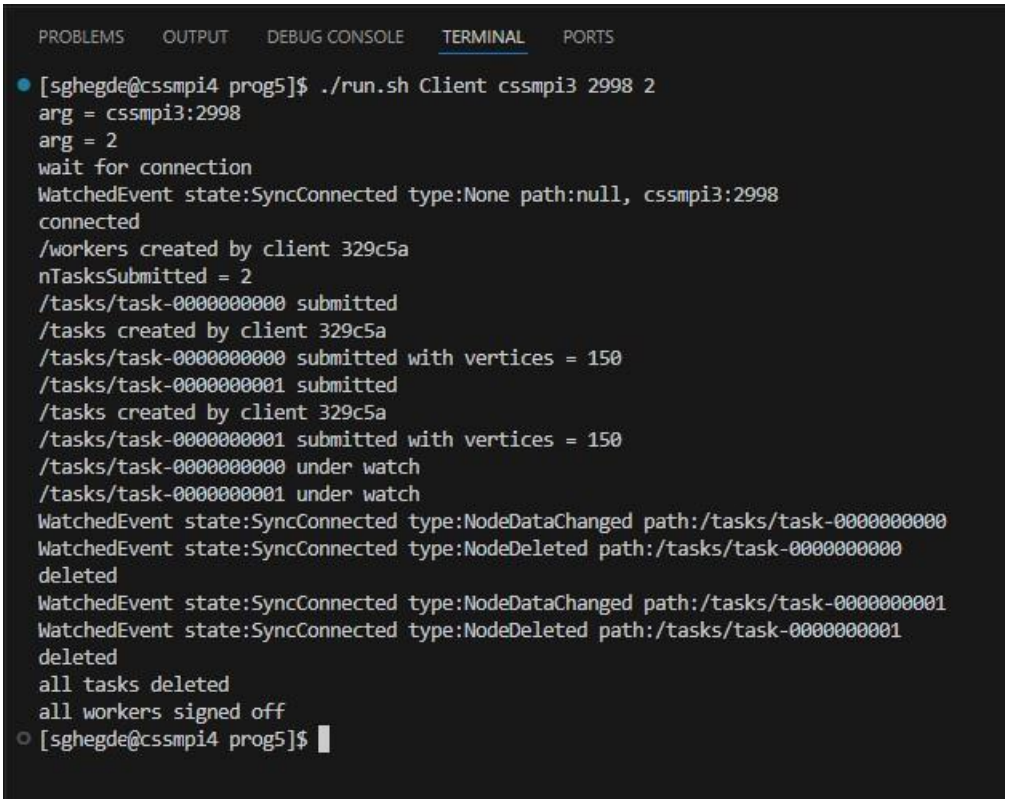

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
[sghegde@cssmpi2 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000003 registered
/lock acquired
task-0000000000
1901752's version: 0
currTime = 1749460119470, pastTime = 1901752, diff = 1749458217718
overdue
1749460119470
/lock released
task-0000000000 in progress by /workers/worker-0000000003
Warning: taskData not a number → using default formula
vertices = 500000
500000's # brdiges = 998, time elapsed = 5349 msec
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
task-0000000001
1749460120490's version: 1
currTime = 1749460124944, pastTime = 1749460120490, diff = 4454
task-0000000002
1749460122239's version: 1
currTime = 1749460124945, pastTime = 1749460122239, diff = 2706
task-0000000003
670774's version: 0
currTime = 1749460124947, pastTime = 670774, diff = 1749459454173
overdue
1749460124947
/lock released
task-0000000003 in progress by /workers/worker-0000000003
Warning: taskData not a number → using default formula
vertices = 2000000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
2000000's # brdiges = 3973, time elapsed = 19923 msec
/lock acquired
/lock released
[sghegde@cssmpi2 prog5]$

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[sghegde@cssmpi4 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000004 registered
/lock acquired
task-0000000000
1749460119470's version: 1
currTime = 1749460120484, pastTime = 1749460119470, diff = 1014
task-0000000001
770639's version: 0
currTime = 1749460120490, pastTime = 770639, diff = 1749459349851
overdue
1749460120490
/lock released
task-0000000001 in progress by /workers/worker-0000000004
Warning: taskData not a number → using default formula
vertices = 1000000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
1000000's # brdiges = 1980, time elapsed = 6115 msec
/lock acquired
task-0000000002
1749460122239's version: 1
currTime = 1749460126699, pastTime = 1749460122239, diff = 4460
task-0000000003
1749460124947's version: 1
currTime = 1749460126700, pastTime = 1749460124947, diff = 1753
/lock released
/lock acquired
task-0000000002
1749460122239's version: 1
currTime = 1749460136708, pastTime = 1749460122239, diff = 14469
task-0000000003
1749460124947's version: 1
currTime = 1749460136709, pastTime = 1749460124947, diff = 11762
/lock released
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
org.apache.zookeeper.KeeperException$NoNodeException: KeeperErrorCode = NoNode for /tasks
/lock released
[sghegde@cssmpi4 prog5]$
```

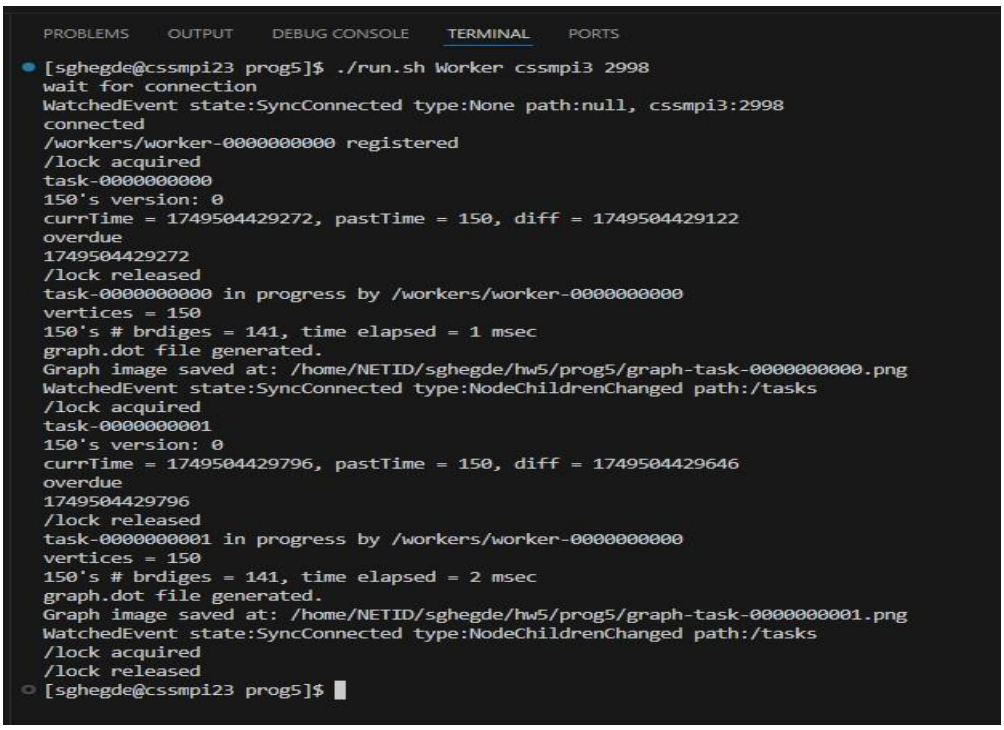
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[sghegde@cssmpi1 prog5]$ ./run.sh Worker cssmpi3 2998
wait for connection
WatchedEvent state:SyncConnected type:None path:null, cssmpi3:2998
connected
/workers/worker-0000000005 registered
/lock acquired
task-0000000000
1749460119470's version: 1
currTime = 1749460122229, pastTime = 1749460119470, diff = 2759
task-0000000001
1749460120490's version: 1
currTime = 1749460122238, pastTime = 1749460120490, diff = 1748
task-0000000002
1985943's version: 0
currTime = 1749460122239, pastTime = 1985943, diff = 1749458136296
overdue
1749460122239
/lock released
task-0000000002 in progress by /workers/worker-0000000005
Warning: taskData not a number → using default formula
vertices = 1500000
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
1500000's # brdiges = 2941, time elapsed = 17826 msec
/lock acquired
task-0000000003
1749460124947's version: 1
currTime = 1749460140206, pastTime = 1749460124947, diff = 15259
/lock released
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks
/lock acquired
org.apache.zookeeper.KeeperException$NoNodeException: KeeperErrorCode = NoNode for /tasks
/lock released
[sghegde@cssmpi1 prog5]$
```

Show graphical outputs.

Client Screen



Worker Screen:

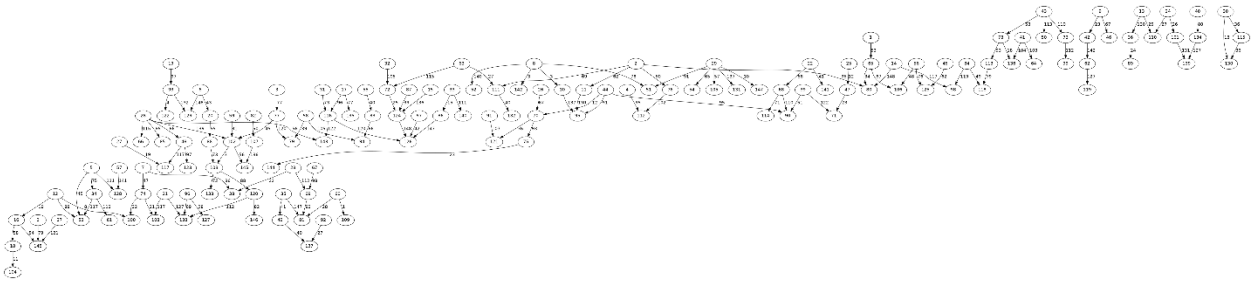


Generated Images:

1000 vertices: This is a Huge low-resolution graph. I couldn't fit this in the document. So, I have uploaded and shared from the UW drive. Please access with UW email. Please zoom in to see the vertices and edges.

https://drive.google.com/file/d/1LrpJzUqJYfbQVqSlD57msSVaUINpA_hP/view?usp=sharing

150 vertices:



If you are not able to see from this Please access the below drive link:

https://drive.google.com/file/d/1ZmyeJw96F6OgfBk-qaearb1z610c62Hb/view?usp=sharing

Discussions:

I have implemented three additional features, which are as listed below. Each of these will be discussed in detail later in the page.

1. Run any number of tasks rather than only 10 tasks.
2. Run any applications with any problem size.
3. Show graphical outputs.

Run any number of tasks: To implement this feature, the Client was modified to accept the desired number of tasks as a command-line argument, allowing it to submit any number of tasks dynamically instead of the fixed 10. The createBagOfTasks() method was updated to delete any existing /tasks znode and its children before creating a fresh set of tasks based on the specified number. This was done to ensure that stale or incomplete tasks from previous runs would not interfere with the current execution and result in inconsistent task processing. While this feature worked correctly, it was observed that as the number of tasks increased significantly, the overall execution time also grew, as Workers required more time to process a larger volume of tasks and the Client had to wait for all tasks to complete before cleanup.

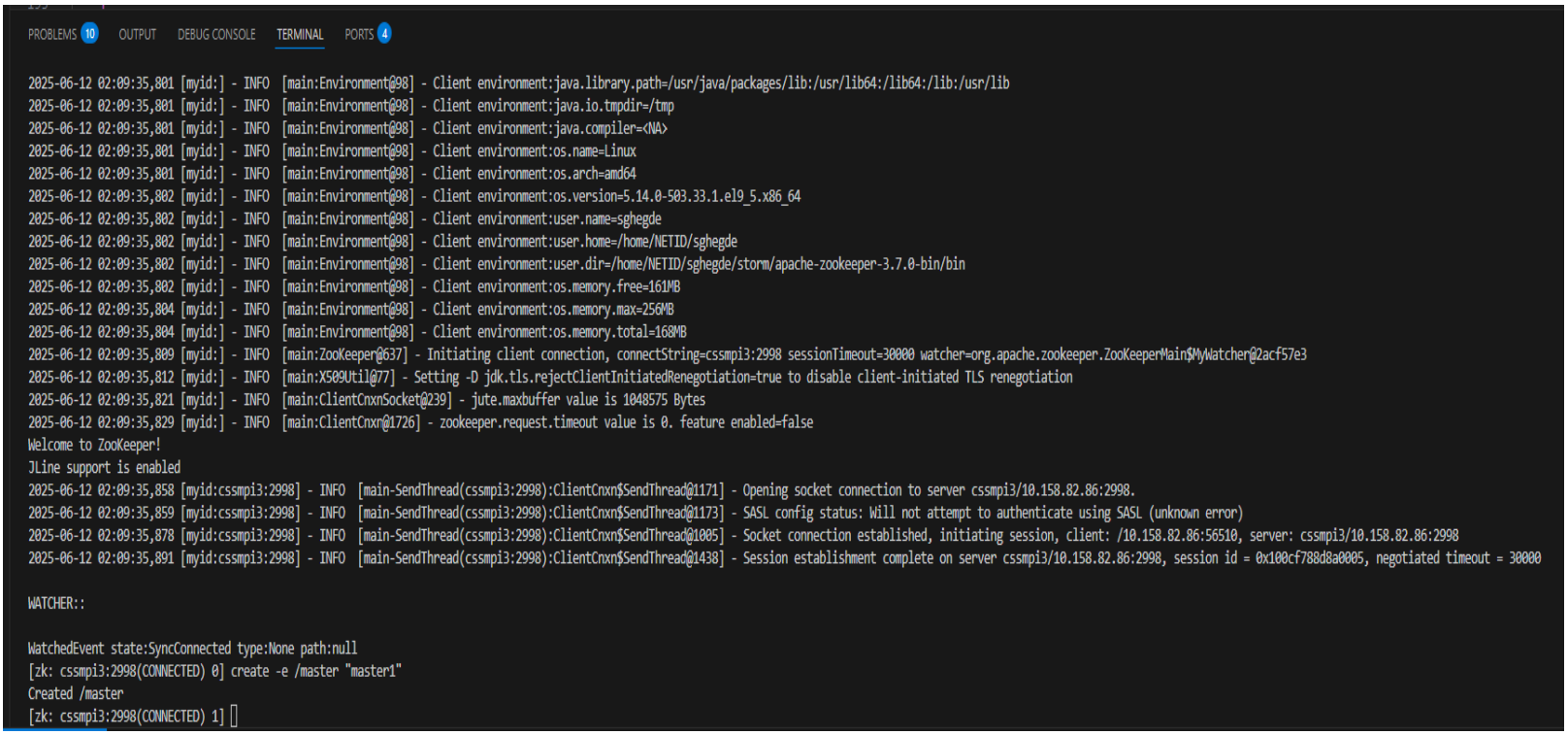
Run any applications with any problem size: This feature involved modifying the Worker to support variable graph sizes based on the task being processed. This was implemented by interpreting the task data or fallback values to compute the appropriate number of vertices to pass to the GraphBridge program. The runTask() method was enhanced to extract this information and calculate the corresponding vertex size dynamically. As a result, Workers were able to execute tasks that generated graphs of varying complexity and size, demonstrating the system's ability to handle heterogeneous workloads. It was observed that tasks with larger vertex sizes naturally took longer to complete, and processing times varied noticeably across tasks. This variation required careful handling of task synchronization and monitoring to ensure consistent system behavior.

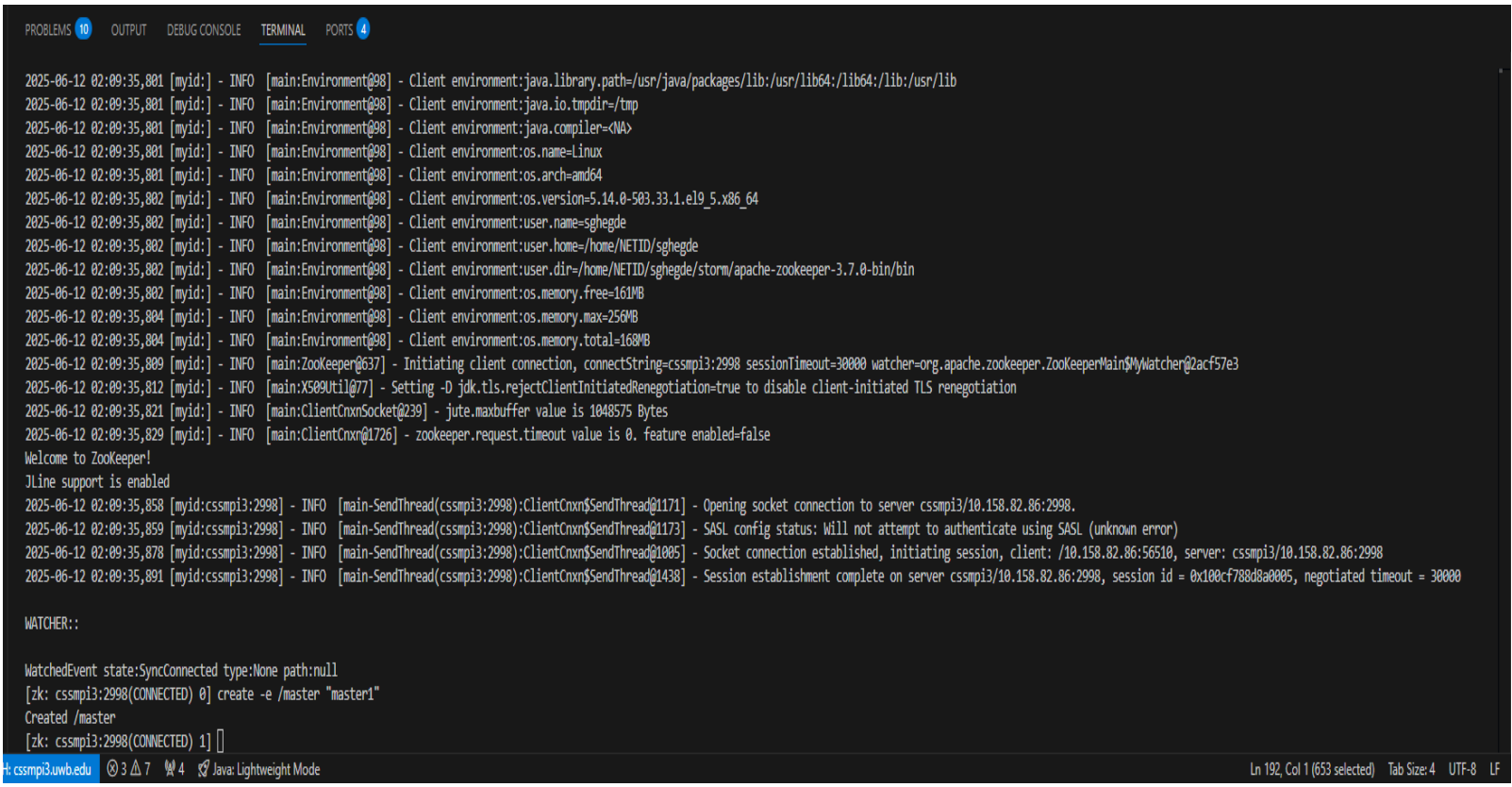
Show graphical outputs: The third additional feature added graphical output generation for each processed graph. This was implemented by extending the runTask() method in the Worker to run the dot tool after executing the GraphBridge program. The GraphBridge program was modified to generate a graph.dot file representing the graph's structure after computing the number of bridges. The Worker then invoked the dot command to convert this .dot file into a .png image and saved it with a name corresponding to the task ID. This allowed each Worker to produce a visual representation of the graph it processed. During testing, it was observed that small and medium graphs generated clear images quickly. However, for larger graphs, the graph.dot file became very large (sometimes exceeding 85 MB), causing the dot tool to take significantly longer to process and sometimes report parsing errors if the dot file exceeded certain internal limits. Despite these challenges, the feature was successfully implemented, and PNG images were generated for the completed tasks where possible. This functionality provided an additional layer of output for visual verification of the distributed processing results.

This completes the additional features explanation. All the source code related to these features is provided in the *Source Code* section, and the corresponding execution outputs are included in the *Execution* section of this report.

Lab 10:

Outputs:





PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS 4

```
2025-06-12 02:11:09,231 [myid:] - INFO  [main:ZooKeeper@637] - Initiating client connection, connectString=cssmpi3:2998 sessionTimeout=30000 watche
2025-06-12 02:11:09,235 [myid:] - INFO  [main:X509Util@77] - Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated
2025-06-12 02:11:09,243 [myid:] - INFO  [main:ClientCnxnSocket@239] - jute.maxbuffer value is 1048575 Bytes
2025-06-12 02:11:09,253 [myid:] - INFO  [main:ClientCnxn@1726] - zookeeper.request.timeout value is 0. feature enabled=false
Welcome to ZooKeeper!
2025-06-12 02:11:09,276 [myid:cssmpi3:2998] - INFO  [main-SendThread(cssmpi3:2998):ClientCnxn$SendThread@1171] - Opening socket connection to serve
2025-06-12 02:11:09,276 [myid:cssmpi3:2998] - INFO  [main-SendThread(cssmpi3:2998):ClientCnxn$SendThread@1173] - SASL config status: Will not attem
2025-06-12 02:11:09,285 [myid:cssmpi3:2998] - INFO  [main-SendThread(cssmpi3:2998):ClientCnxn$SendThread@1005] - Socket connection established, ini
JLine support is enabled
2025-06-12 02:11:09,302 [myid:cssmpi3:2998] - INFO  [main-SendThread(cssmpi3:2998):ClientCnxn$SendThread@1438] - Session establishment complete on

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: cssmpi3:2998(CONNECTED) 0] create -e /workers/worker1
Created /workers/worker1
[zk: cssmpi3:2998(CONNECTED) 1] create /assign/worker1 ""
Created /assign/worker1
[zk: cssmpi3:2998(CONNECTED) 2] ls /workers
[worker1]
[zk: cssmpi3:2998(CONNECTED) 3] ls /assign
[worker1]
[zk: cssmpi3:2998(CONNECTED) 4] ls /assign/worker1
[task-0000000001]
[zk: cssmpi3:2998(CONNECTED) 5] create /task/task-0000000001/status "done"
Node does not exist: /task/task-0000000001/status
[zk: cssmpi3:2998(CONNECTED) 6] create /tasks/task-0000000001/status "done"
Created /tasks/task-0000000001/status
[zk: cssmpi3:2998(CONNECTED) 7]
```

H: cssmpi3.uwb.edu ⊗ 3 ⚠ 7 4 Java: Lightweight Mode

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS 4

```
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/tasks/task-0000000001
get -s /tasks/task-0000000001
cmd
cZxid = 0x993
ctime = Thu Jun 12 02:11:59 PDT 2025
mZxid = 0x993
mtime = Thu Jun 12 02:11:59 PDT 2025
pZxid = 0x998
cversion = 1
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 3
numChildren = 1
[zk: cssmpi3:2998(CONNECTED) 11] get -s /tasks/task-0000000001/status
done
cZxid = 0x998
ctime = Thu Jun 12 02:13:38 PDT 2025
mZxid = 0x998
mtime = Thu Jun 12 02:13:38 PDT 2025
pZxid = 0x998
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 4
numChildren = 0
[zk: cssmpi3:2998(CONNECTED) 12]
```

H: cssmpi3.uwb.edu ⊗ 3 ⚠ 7 4 Java: Lightweight Mode