Project Report

Anusha Sudini (asudini@memphis.edu)

Chaitanya Pochampally (cpchmpll@memphis.edu)

Sindhuja Datla (sdatla@memphis.edu)

Project Description :

This project is the implementation of tiling puzzle solver.

The entire project can be divided into 4 parts:

1) Parsing the input file to read the tiles

2) Placing the tiles on the board to find all the possible solutions

3) Checking if the current part of the solution is promising to decide whether to proceed further or not

4) Removing all the isomorphic solutions from the solutions available

5) Implementing the steps 2, 3, 4 in distributed fashion.

We shall see each step in detail:

1) Parsing the input file to read the tiles : **This is done recursively**
   - Once the main program is invoked, the file that is passed as input is parsed to extract all the tiles and the board.
   - The file is first read into a two dimensional array and then method looks for a character, if the character is found, it's position in the file and color of it is saved and that character is made to blank in the input array, so that the same tile will not be read again
   - And then checks if there is any character either to the right, left, up, bottom. If found any, the values are stored and the value is made blank in the input array
   - All these adjacent tiles are stored as a single tile object which has coordinates and color for each square of the tile
   - After storing all the solutions, the tile with more number of squares is considered as the target board.

2) Placing the tiles on the board in a distributed manner**: Sockets are used to run the code on the distributed systems**. Two classes have been defined to run the code the distributed systems

   Server Class:

- The server class calls, accepts the connection from the clients and creates a thread for each client connection.
- The client thread sends the tile and it's orientation to be placed in the first position on the board.
- The client thread receives the individual solution list from the client and add to the solution list which contains all the solutions from other clients and the thread is closed
- Once all the tiles and its orientation is sent to all the clients and solutions received. The server then checks for isomorphic solutions, if any removes them from the solution list.

Client class:

- The client receives the tile and its orientation and places it on the first position on the board and further checks all the possibilities of placing the remaining tiles with all orientations in all the positions.
- It stores all the solutions and returns the solution to the server.
- It then closes the connection with server and opens it again to check for other tile and its orientation.

3) Placing the tiles on the board with all possible combinations by the client: **Recursive branch and bound technique is used**

- Each tile will be having a set of maximum of 8 orientations i.e. rotate0, rotate90, rotate180, rotate270, flip0, flip90, flip180, flip270.
- To track if a tile is placed on the board or not, a tilestate array is maintained, which has the values as tile identifier if not placed, otherwise a negative value.
- The steps for placing the tiles is as follows:

1) First a tile is taken starting from the no orientation, if the tile can be fit on the board, the tile status is changed and next free coordinates are found (from left to right and top to down) and executes the 3rd step.

2) If the tile is not able to fit in the current coordinates, then next orientation of the tiles is considered and step 1 is continued.

4) The same function will be called recursively, trying with the other tiles with the new coordinates generated.

5) Each time after placing a tile and before checking for the next coordinates, if the board is not full, we will be checking if the currently filled board is promising or not. If it is promising, we will be proceeding further (step1). If not promising, the currently placed tile is removed from the board and next orientation of the same tile or next tile is tried.

6) After placing each tile on the board, if the board is full, the solution is added to the list of solutions. And process continues for the next tiles.

- To check if a square on the board is filled or not, we will be checking the id. Usually, all the board squares all initially assigned with id as -1. So if a square is filled by a square of a particular tile that tile id is present, instead of -1.

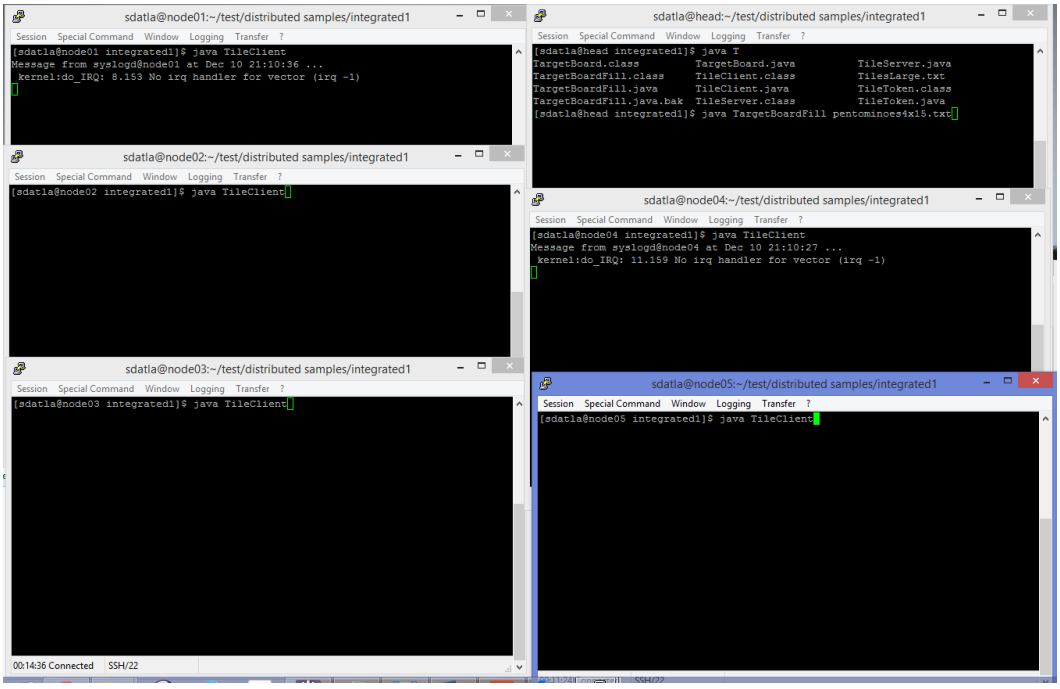
3) Checking for promising solution:

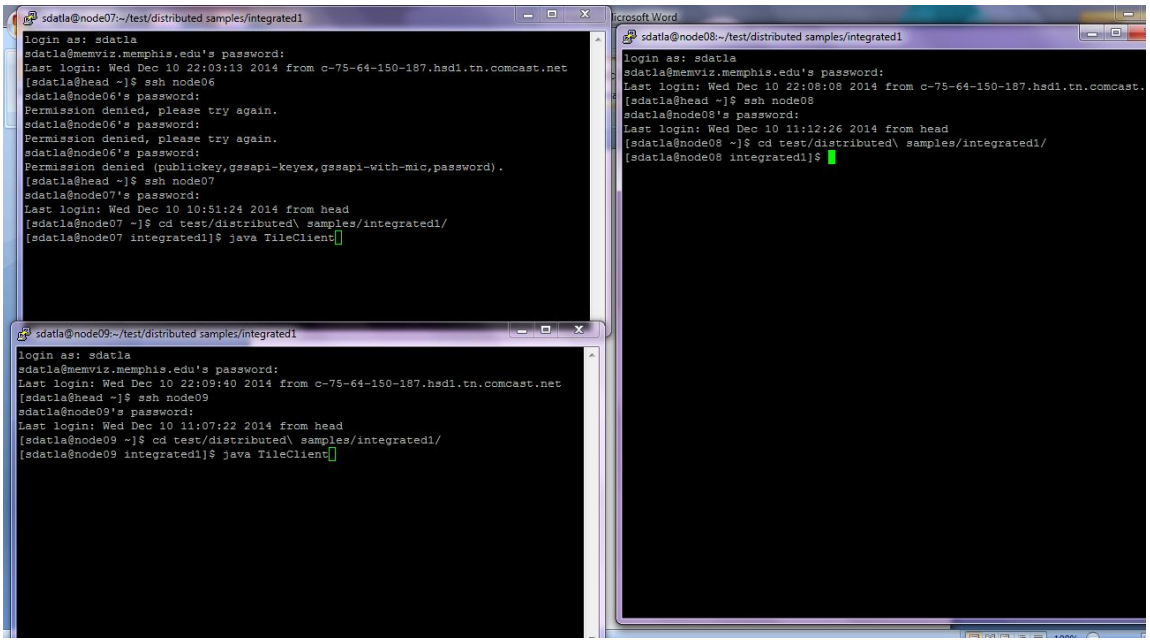
- After checking the number of smallest holes and largest holes. The count of the holes is subtracted by the number of same sized tiles (that are not yet placed on the board) as min or large holes respectively. Even after subtracting, any of the holes count is greater than zero, then there is no possibility of the holes to be filled by other tiles. Hence it is not promising.
- And also if the minimum sized available tile size is more than the smallest hole or the maximum sized available tile size is greater than the largest hole also implies that those holes cannot be filled in any way. Hence solution will not be promising

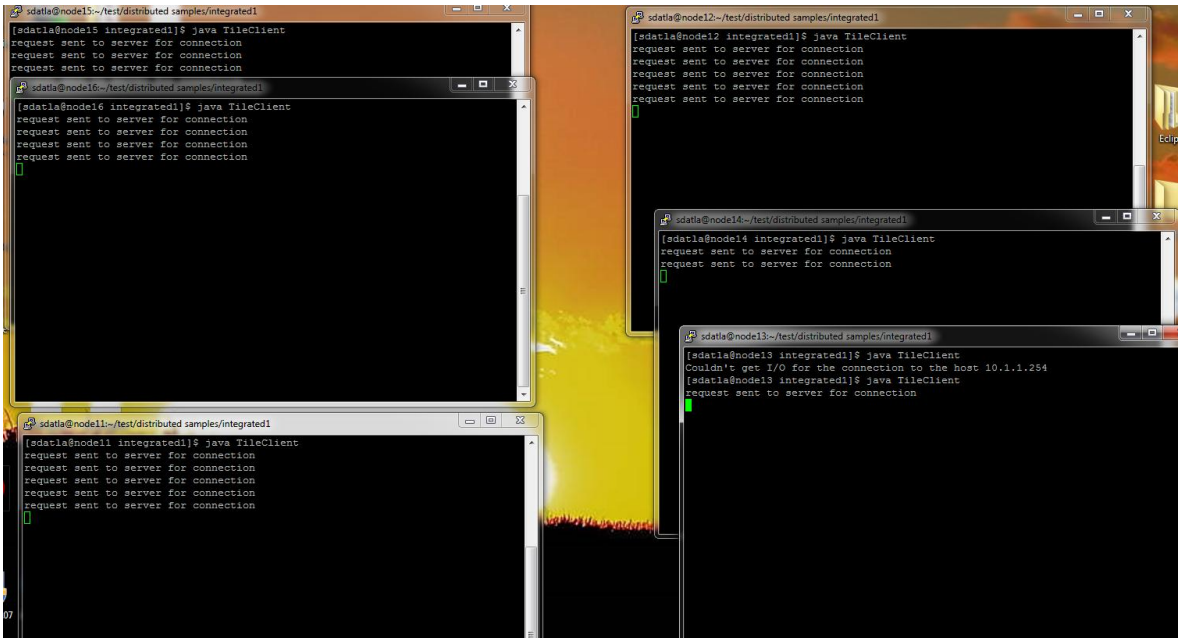
4) Checking isomorphism:
Each solution is oriented in all possible ways and checked with all the solutions, if it matches, then it is isomorphic
Matching is done by comparing id and color of each square with other board solution.

Snapshots:

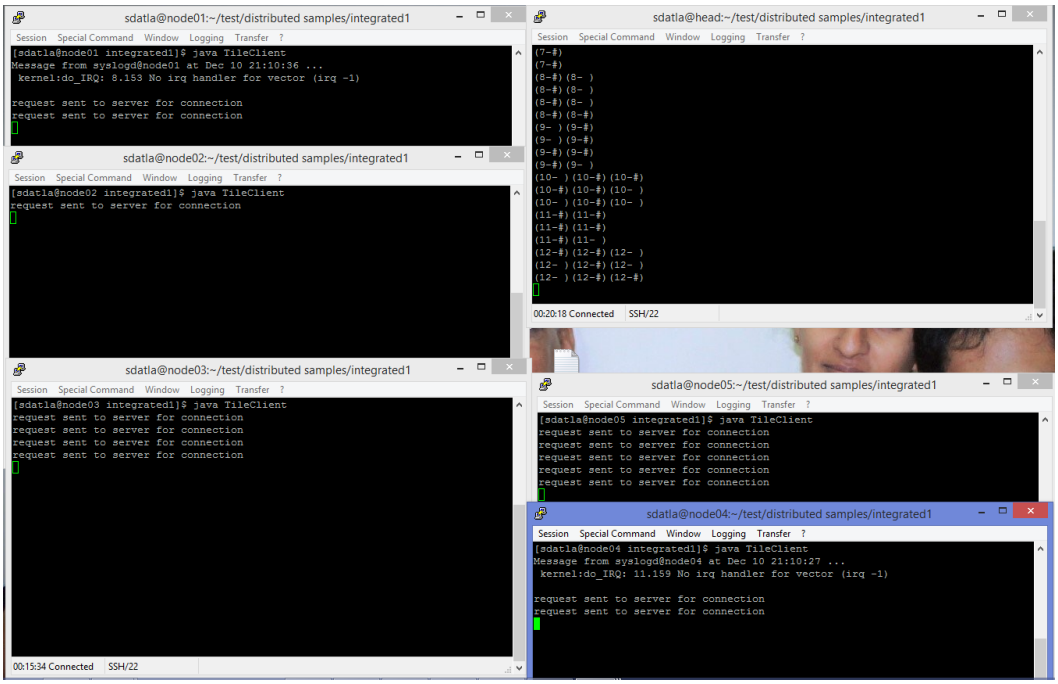
Headnode and client nodes before starting



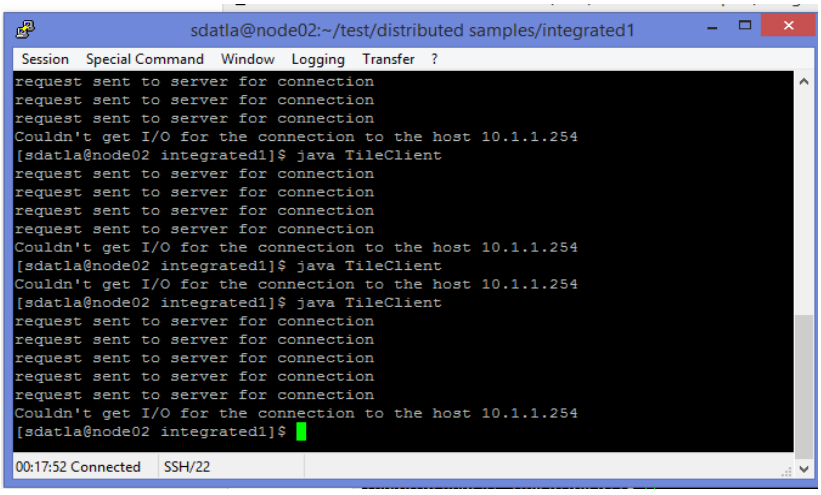

sdatla@node07:~/test/distributed samples/integrated1
login as: sdatla
sdatla@memviz.memphis.edu's password:
Last login: Wed Dec 10 22:03:13 2014 from c-75-64-150-187.hsd1.tn.comcast.net
[sdatla@head ~]$ ssh node06
sdatla@node06's password:
Permission denied, please try again.
sdatla@node06's password:
Permission denied, please try again.
sdatla@node06's password:
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
[sdatla@head ~]$ ssh node07
sdatla@node07's password:
Last login: Wed Dec 10 10:51:24 2014 from head
[sdatla@node07 ~]$ cd test/distributed\ samples/integrated1/
[sdatla@node07 integrated1]$ java TileClient
sdatla@node09:~/test/distributed samples/integrated1
login as: sdatla
sdatla@memviz.memphis.edu's password:
Last login: Wed Dec 10 22:09:40 2014 from c-75-64-150-187.hsd1.tn.comcast.net
[sdatla@head ~]$ ssh node09
sdatla@node09's password:
Last login: Wed Dec 10 11:07:22 2014 from head
[sdatla@node09 ~]$ cd test/distributed\ samples/integrated1/
[sdatla@node09 integrated1]$ java TileClient
sdatla@node08:~/test/distributed samples/integrated1
login as: sdatla
sdatla@memviz.memphis.edu's password:
Last login: Wed Dec 10 22:08:08 2014 from c-75-64-150-187.hsd1.tn.comcast.
[sdatla@head ~]$ ssh node08
sdatla@node08's password:
Last login: Wed Dec 10 11:12:26 2014 from head
[sdatla@node08 ~]$ cd test/distributed\ samples/integrated1/
[sdatla@node08 integrated1]$


sdatla@node15:~/test/distributed samples/integrated1
[sdatla@node15 integrated1]$ java TileClient
request sent to server for connection
request sent to server for connection
request sent to server for connection
sdatla@node16:~/test/distributed samples/integrated1
[sdatla@node16 integrated1]$ java TileClient
request sent to server for connection
request sent to server for connection
request sent to server for connection
request sent to server for connection
sdatla@node11:~/test/distributed samples/integrated1
[sdatla@node11 integrated1]$ java TileClient
request sent to server for connection
request sent to server for connection
request sent to server for connection
request sent to server for connection
request sent to server for connection
sdatla@node12:~/test/distributed samples/integrated1
[sdatla@node12 integrated1]$ java TileClient
request sent to server for connection
request sent to server for connection
request sent to server for connection
request sent to server for connection
request sent to server for connection
sdatla@node14:~/test/distributed samples/integrated1
[sdatla@node14 integrated1]$ java TileClient
request sent to server for connection
request sent to server for connection
sdatla@node13:~/test/distributed samples/integrated1
[sdatla@node13 integrated1]$ java TileClient
Couldn't get I/O for the connection to the host 10.1.1.254
[sdatla@node13 integrated1]$ java TileClient
request sent to server for connection

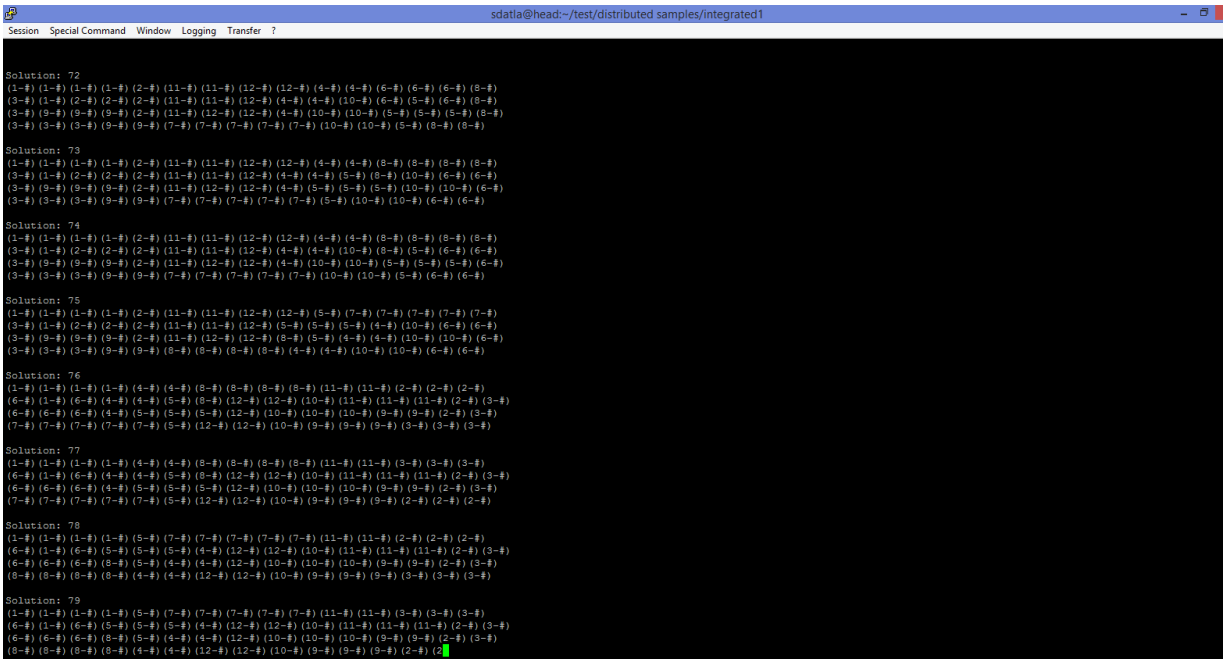
Head and client nodes during execution: head nodes displaying the tiles



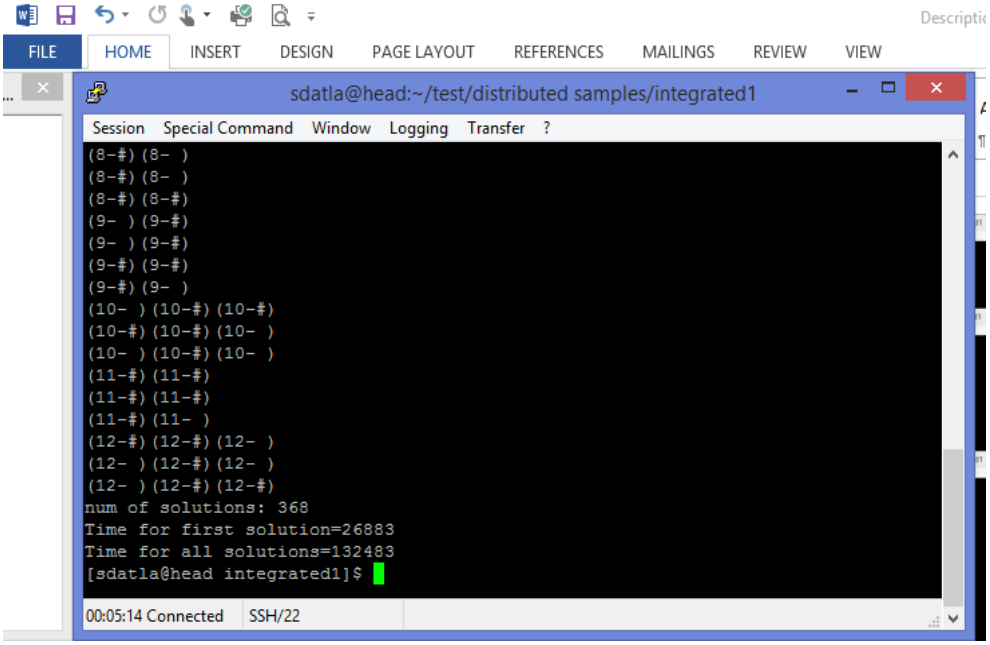
One of the client node, Once the solutions calculation is done



Head node displaying the outputs:



Head node displaying the number of solutions and time taken for solutions:



Puzzle outputs:

Puzzle outputs for the puzzles present in the puzzle project pdf
http://www.cs.virginia.edu/~robins/puzzles

	Puzzle Name	No.of Solutions	Time to find first solution(milliseconds)	Time to find all solutions (milliseconds)
1	pentominoes5x12	1010	12754	139612
2	pentominoes6x10	2339	6439	86535
3	pentominoes3x20	2	23211	45250
4	Pentominoes4x15	368	23029	123957
5	pentominoes8x8_middle_missing	65	1368	7403
6	pentominoes8x8_side_missing	1288	2307	16836
7	pentominoes8x8_corner_missing	5027	3934	26292
8	pentominoes8x8_four_missing_corners	43602	6278	329398
9	pentominoes8x8_four_missing_near_corners	188	3924	12388
10	pentominoes8x8_four_missing_near_middle	21	2478	6456
11	pentominoes8x8_four_missing_offset_near_middle	126	998	4662
13	pentominoes8x8_four_missing_offset_near_corners	54	2220	5791
14	pentominoes8x8_four_missing_diagonal	74	1244	3622
15	trivial	1	2210	2275
16	checkerboard	1558	3243	14876
17	IQ_creator	6	1612	2332
19	Thirteen_holes	8	1530	1914

Instructions to Run:

Execute this from the head node
Java `TargetBoardFill` `abc.txt`

Execute this from all client nodes
Java TileClient

First head node should be executed and then all the client nodes. The solutions will be printed in the head node.