Construction of Agent-Navigable Multi-Dimensional Spaces from File Inputs

Yuna Guo

Term report of work done as CSS600

Master of Science in Computer Science & Software Engineering

University of Washington, Bothell

Spring 2020

Project Committee:

Munehiro Fukuda, Ph.D., Committee Chair

Min Chen, Ph.D., Committee Member

Robert Dimpsey, Ph.D., Committee Member

1. Introduction

Parallel data analysis has been widely used in Big Data Analytics as increasing amount of data volumes and demand in processing speeds. A variety of software tools such as MapReduce, Spark and Storm have been used by scientist for easy parallelization of user applications, with key concepts of data streaming through analyzing units (e.g., map/reduce function in MapReduce, lambda expressions in Spark, and spouts/blots objects in Storm). However, these big data tools flatten and shuffle dataset and the structure of data couldn't be maintained. In contrast, scientific datasets usually have complex structures, such as array or graph. For example, climate analysis uses NetCDF datasets that maintain data in a multi-dimensional array and should be scanned several times to detect a climate change. Thus, developing tools of data streaming which preserve a data structure in distributed memory is necessary [1].

To apply parallel big data analysis in specific scientific dataset, Prof. Fukuda's group developed parallel agent-based models (ABMs) simulator named multi-agent spatial simulation (MASS) to apply agent-based data discovery in various scientific fields, such as bioinformatics, climate science, space cognition and environmental data science. The current specification of the MASS library includes two classes: Places and Agents. The Places are simulation spaces with a multi-dimensional array that is distributed over a cluster system. The Agents are simulation entities with a set of mobile objects that can migrate over Places. MASS library performs parallel execution of Places and Agents using multi-thread communicating processes forked over cluster nodes through JSCH and connected via TCP sockets. Prof. Fukuda's group has successfully applied MASS library to parallel data analysis – a global-warming analysis based on NetCDF climate data [2] and biological network motif search [3]. These previous work facilitates our development of agent-based algorithms and infrastructure technologies to identify more features in agent-based data discovery.

However, the only data structure that is supported by current MASS is multi-dimensional distributed array. In order to handle different data structures such as contiguous space, trees or graphs, we should derive Space, Tree and Graph class from Places. In this individual study, I focused on designing Space class from MASS Places. Compared to current Places, the Space class is a continuous space which creates Place dynamically based on input file. The contiguous

multi-dimensional Space class is able to distribute data points in an optimal way and be used in a variety of applications, such as Closest Pair Of Points and Voronoi Diagrams.

2. Progress

The Space has been implemented and tested in the application of Closest Pair of Points. The modified MASS with Space class was successfully executed in single node, and the program was under debugging in multi-nodes. I expect to finish the implementation and testing of Space MASS in multi-nodes in next quarter.

3. Methods

By deriving contiguous Spaces from Places, the MASS reads input file directly and distributes a set of given data points over a geometric space.

1) SpacePlaces.java : The SpacePlaces class is derived from Places, which reads data points from input file. In this class, SpacePlace object is instantiated and the number of SpacePlace is determined by the input data point. In addition, the fineness of SpacePlace object could be set by user defined granularity.

public SpacePlaces extends Places (int handle, String classname, int dimension, int granularity, String filename)
- classname: user's class name *(e.g. SpacePlace)* - dimension: int - granularity: int - filename: String - Places : SpacePlace[] - numOfPlace : int *(equals number of points)*

public SpacePlace extends Place (Object args)
-index: int[] -coordinates: double[] -min: double[] -max: double[] -agentsMap : Hashtable<Integer, Set<Integer>> *<key, value> = <(linear sub-index of sub-place), (a set of agents which reside in the sub-place)>*

+ setIndex(int[] index) + addAgent(int agentId, int[] subIndex) *(add agent to the HashMap)* + removeAgent(int agentId, int[] subIndex) *(remove agent from HashMap)*

Internal Design: The internal structure of Space is the same as Places, but the init_algorithm identifies the original data space in a continuous manner. Each Spaces constructs SpacePlace which is partitioned into sub-SpacePlace. The finesse of sub-SpacePlace is defined by the granularity. Since the SpacePlace has a range in each dimension, agent migration must be recorded to detect collision. A HashMap will be used to trace agent and its corresponding sub-index in Place. If another agent moves to the same sub-SpacePlace, two agents collide. The current SpacePlaces is able to read data points from text file.

2) SpaceAgent.java: The SpaceAgent class is derived from Agent, which includes its coordinates, corresponding index and sub- index of Place. The function of SpaceAgent includes spawn, kill, migrate and propagate.

public SpaceAgent extends Agent(Object args)
-originalCoordinates: double[] -nextCoordinates: double[] -index: int[] -subIndex: int[]
+ spawnAgent() + killAgent() + migrateAgent() + propogateAgent()

Internal design: The internal structure of SpaceAgent is the same as Agent. The attributes include originalCoordinates which is the input data point, nextCoordinates which is the location agent migrating and subIndex which is the sub-index of Place it resides. There are two patterns of agent migration – Moore and Von Neumann. For example, in 2D space, the agent migrates in eight directions (N, S, W, E, NE, NW, SE and SW) in Moore-Neighbor or in four directions (N, S, W, E) in Von Neumann-Neighbor. In migrateAgent(), agent migrates only in Moore-Neighbor; in propagateAgent(), agent migrates in Moore-Neighbor and Von Neumann-Neighbor alternately.

4. Results

Comparison of Space MASS and original MASS

The continuous Space MASS library was tested using Closest Pair of Point and the execution performance was compared to the original MASS library. Table 1 shows the results of comparison between Space MASS and original MASS.

Table 1. Performance of MASS in Closest Pair of Points (single core)

	original MASS			**Space MASS**		
Input points	**# of Places**	**# of Agents**	**Execution Time (ms)**	**# of Places**	**# of Agents**	**Execution Time (ms)**
256 points	999*798	2304	9214	17*17	2304	491
128 points	999*797	72918	19036	13*13	1152	365
64 points	999*797	>131672000 *	N/A	9*9	576	234
32 points	N/A	N/A	N/A	7*7	288	173
16 points	N/A	N/A	N/A	5*5	144	135
8 points	N/A	N/A	N/A	4*4	72	109
4 points	N/A	N/A	N/A	3*3	36	84

*java.lang.ArrayIndexOutOfBoundsException due to the # of agent in node exceeds capacity (n = 1000)

This result shows that the original MASS only executed successfully in 256 points and 128 points. When the input points = 64, as the points are more dispersed, more agents need to be spawned until they collide, which exceeds the capacity of agents in each node. Therefore, the original MASS is not suitable for dispersed input points. In addition, the execution time of 128 points is greater than 256 points because more agents need to be spawned for collision.

In Space MASS, this issue is avoided because the number of Places is dynamically determined by the number of input points. Because the range of SpacePlace is continuous, fewer Places and Agents are needed when there are fewer input points. And the execution time is reduced as less Places and Agents are created.

Compared to original MASS, the execution time using Space MASS is about 20 times faster in 256 points and >50 times faster in 128 points. And the number of Place and Agent created in program is dramatically reduced. Overall, the Space MASS has a better performance (less execution time and less memory usage) than the original MASS.

Comparison of migrate() and propagate() in Closest Pair of Points

Table 2. Performance of migrate() and propagate() in Closest Pair of Points (single core)

	propagate()		**migrate()**	
Input points	**# of Agents**	**Execution Time (ms)**	**# of Agents**	**Execution Time (ms)**
256 points	1280	384	2304	491
128 points	640	265	1152	365
64 points	320	188	576	234
32 points	160	150	288	173
16 points	80	118	144	135
8 points	40	87	72	109
4 points	20	76	36	84

In Table 2, the execution time using propagate() method is shorter than migrate(). In propagate(), agents are spawned in Moore Neighbor (n = 8) and Von Neumann Neighbor (n = 4) alternately while agents are spawned in Moore Neighbor only in migrate() method. In migrate(),

more agents are created which increases execution time. To further improve the performance, a boolean attribute could be added to the SpacePlace class which tracks whether the Place has been visited by an agent which is spawned from the same original data point. In this way, agent will only ‘spread out’ from the original point without ‘move back’ and less agents will be created.

5. Project Package

Right now all Source code can be found under the Project directory. Link to the repository: https://drive.google.com/drive/u/2/folders/1xj3PTnUWqm5kFqTkzw2oCVmjjxqDROIv

6. Conclusion and Discussion

Conclusions

From the result for the Closest Pair of Points application, the execution performance of Space MASS implementation outperforms original MASS. With this continuous multi-dimensional Space class, users are able to distribute data points in an optimal way that minimizes the number of Place and Agent. This functionality could be used in a variety of geometric application, such as Closest Pair of Points and Voronoi Diagrams application.

Limitations and Future Work

Unfortunately, the current program was not able to execute in multi-nodes. When the program was tested in two nodes, the connection was established successfully but the master node was not able to receive message from remote node. I am trouble shooting this problem and expect to include the multi-nodes execution with other applications in next quarter.

Space MASS gives the best performance when the input points are evenly distributed and may not be suitable for clustered points. To solve this problem, a Tree class (e.g. quad/octo tree) needs to be implemented which dynamically adds or deletes node. For example, a quad-tree construction reads points and divides a space quarterly if a new point resides on a space that includes another point and the division can be done in parallel.

In current Space MASS library, when the program runs in multiple nodes, each computing node reads the entire input file. In order to read file more efficiently, each SpacePlace should read/write file in parallel. For divisible file, such as text or CVS file, the text/CSV file can be partitioned and stored at a different computing node's /tmp which can be accessed by local MASS process. For reading non-divisible file, such as NetCDF, each MASS process should be set with a different offset and each MASS process jumps to its partitioned data.

The current Space MASS has only been tested by Closest Pair of Points. In order to test the programmability and execution performance, it will also need to be verified using other applications, such as computational geometry application (Voronoi Diagrams) and data science application (e.g. KNN). The LOC and number of instances will be used to test the programmability and the run time will be used to test the execution performance. In addition, the programmability and execution performance of Space MASS needs to be evaluated in comparison to Spark.

6. References

1. Munehiro Fukuda, Collin Gordon, Utku Mert, and Matthew Sell. Computational Framework for Distributed Data Analysis. Computer. 53(3):16-25.
2. Jason Woodring MS, Munehiro Fukuda, Hazeline Asuncion, Eric Salathe. A Multi-Agent Parallel Approach to Analyzing Large Climate Data Sets. In 37th IEEE International Conference on Distributed Computing Systems;**Atlanta, GA, June 2017**:1693-48
3. Matthew Kipps WK, Munehiro Fukuda. Agent and Spatial Based Parallelization of Biological Network Motif Search. In Proc. 17th IEEE International Conference on High Performance Computing and Communication - HPCC 2015;**New York, NY, August 2015**:786-91