Synchronization Methods for Distributed Agent Based Models

Christine Harvey and James E. Gentile, Ph.D.
The MITRE Corporation

Is there a faster way to synchronize agent states in distributed models?

Some methods communicate agent states upon request, at each time step. This can reduce processing efficiency.

Introduction

Distributed computing addresses the traditional scaling limitations of Agent Based Models (ABMs) and allows for the development of massive-scale models. Synchronization of agent states between multiple processors is complex and needs to be reliable and efficient. The protocol used to synchronize agent states across processors has a significant impact on the efficiency of the tool. Repast HPC is one of the current tools available for distributed ABMs [1]. It approaches the problem by performing a complete synchronization of all entities of interest at every time step.

This poster reviews an alternative approach to entity synchronization, a design which manages persistent, pertinent information. This protocol performs an initial synchronization among all entities with relationships to other agents and then only performs updates and synchronization following changes to relevant information. The pertinent data synchronization technique is an event-driven method to manage the communication and synchronization between the processors. The conservative and the event-driven approaches are both described and analyzed in the following sections.

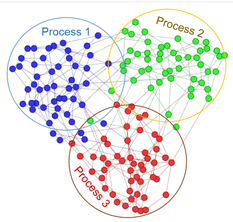


Figure 1: General overview of distributed agents on three processors

Our event-driven approach watches for agent state changes and synchronizes only when those changes occur.

Literature Cited

[1] Collier, N., and M. North. 2013, Oct.. "Parallel agent-based simulation with Repast for High Performance Computing". SIMULATION 89 (10): 1215–1235.

Further Information

For more information on the mabm.py, the Massive Agent Based Modeling Toolkit, visit the MABM Github repository:
https://github.com/ceharvs/mabm

Methods

With other Agent Synchronization techniques, local ghost copies are created of agents on foreign processors. These local copies are basic shadow copies of the actual agents, only containing the relevant state information. This process of creating local agent copies can be seen in **Figure 2.**

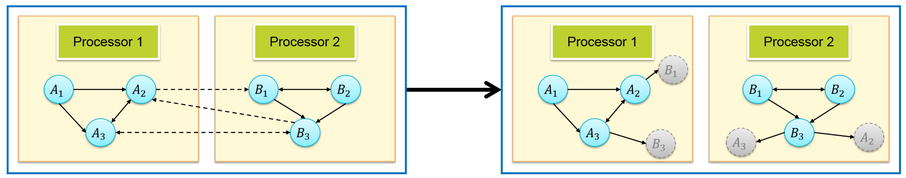


Figure 2: Overview of the local agent copying process. Agents with a relation to agents on foreign processors are recreated as basic local copies.

Agent Requests

The conservative approach to the agent synchronization problem performs a complete synchronization of pertinent information at every time step of the simulation. Each processor cycles through their agents and compiles a list of non-local agents from which information is needed. The root processor aggregates this information and broadcasts the requests to all processors which return the requested state information to the root processor. Once the root processor distributes the information gathered, each processor creates local copies of their non-local agents of interest. These temporary copies only contain necessary state information on the requested entity. Therefore, each processor has information regarding the current state of its own agents as well as basic state information on all agents of interest. This entire process is repeated at the beginning of every time step.

Agent Watches

The alternate approach recognizes that not all pertinent information changes at every time step in the model. Complete synchronization can be achieved by only tracking and reporting the changes to relevant information. Agent watching only synchronizes information when an entity has experienced a change in state.

After the creation of a relationship between two entities, if the agent of interest is not local, it is added to a global list of watched entities. During the synchronization process, the states of newly watched agents are communicated to any processor which has an interest in the agent. Basic, persistent local copies of these watched agents are made on the processors that require the state information of the non-local agent. The processor uses these local copies as a source of information for the updates on their local agents. When watched agents experience a change in state the processor sends the updated state information to the root node to be broadcast to all processors. Then, at the start of the next time step, remote copies of agents are updated to the correct, current state. With this method, fewer and smaller messages are sent at each time step than with the previous techniques.

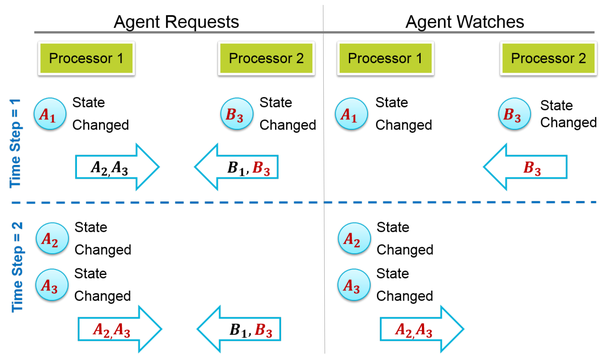
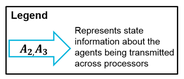


Figure 3: Basic outline of the synchronization process for the Agent Requests and the Agent Watches techniques. This graphic displays the excessive message passing that is part of the Agent Requests technique. Messages are sent for every agent that has a cross-process connection at every time step. The same outcome is accomplished with the Agent Watches technique with many fewer messages passed between the processors.



Event-driven updates are nearly 2x faster!

Results

A complete timing analysis was performed on both techniques using Python's cProfile tool. The sample model used in this experiment was a basic rumor model, where each agent has exactly two assigned neighbors. The connections between agents were distributed according to the probability of having a neighbor on a foreign processor. Trials were run with 20 million and 80 million agents, split across 4, 8, 16 and 32 processors. Three trials were completed at each design point, the average of these three trials can be seen in the charts below. The displayed timing is the actual simulation running time, which does not include the overhead to initialize the model.

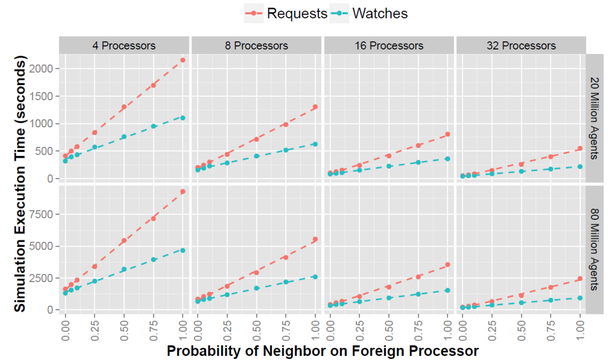


Figure 4: Simulation running time for the test models. The simulation time does not include the time required to initialize the simulation. The information given in these graphs shows that as the number of cross-process connections increases, the impact of the Agent Watches technique becomes more significant. In the most extreme cases, the Agent Watches technique is nearly twice as fast as the Agent Requests technique.

This approach is especially useful for *more static* agents.

Conclusions

The completed timing analysis shows that the watches technique is significantly faster than the requests method while maintaining the same standard of reliability. The run times for the method are quicker overall than with the requests technique. In addition to the overall speed increase, the watches technique also scales up in both number of processors and number of agents per processor more efficiently than the requests method. The use of the agent watches synchronization technique enhances the usability, scalability, and speed for anyone executing massive-scale agent based simulations.

Acknowledgements

Christine Harvey and Dr. James Gentile would like to thank the MITRE Innovation Program for their continued support of this research.