Embracing the Lag: Real-Time Challenges in Multi-Agent Systems

Sarah Dumnich[1], William Birmingham[1], Britton Wolfe[2]
[1]Saint Vincent College
[2]Grove City College
sarah.dumnich@stvincent.edu, william.birmingham@stvincent.edu, wolfebd@gcc.edu

Abstract

In this paper, we discuss the interplay between distributed multi-agent systems and the concept of "lag." We introduce a new representation of these systems to more clearly include time and the effects of latency. We also present real-world situations where communication delays can mean the difference between success and failure, between order and chaotic disarray.

Introduction

In the dynamic world of distributed multi-agent systems, where agents collaborate, coordinate, and make decisions in real time, the concept of "lag" poses a formidable challenge. As these autonomous entities interact across vast networks, communication delays and latency inevitably arise, impacting the system's correctness. It is this inherent problem against the ticking clock that drives our research.

Consider the arenas of distributed video games, self-driving vehicles, and online trading and auction systems. Each of these is impacted by latency. While there are several methods employed to combat the effects of latency in these arenas, it is notable that latency itself is not explicitly accounted for, particularly in POMDP models. We argue that latency needs to be modeled.

The System

Agents act under strict real-time deadlines in a distributed system. As the system is distributed, there are no shared clocks and no common memory. All state exchange among agents is performed over a network via message passing.

Agents are located on machines in the network. A machine can host several agents, although each machine hosting just one agent is a common scenario. Those agents sharing a machine must behave as agents on different machines: each has a unique (logical) clock (although they can share a physical clock), its own state, and state exchange between agents must occur over the network (although localhost may be used for performance reasons). In other words, agents maintain local state and exchange state via a network, just like networked agents.

As the system is real-time and the network is a physical entity, it suffers from latency and information loss. For simplicity, but without restricting the model, we consider information loss as a form of latency (i.e., TCP retransmission protocol). The network is, for example, simplex between agents following the design pattern of Unix sockets.

State

In models of dynamical systems like POMDPs, there is one (potentially hidden) world state vector. That state vector, along with the dynamics of the system, determines the evolution of the entire system. In multi-agent variations of POMDPs (Spaan, Oliehoek, and Vlassis 2008; Messias, Spaan, and Lima 2013) there is a single state vector, but different agents receive different observations, leading to different "views" of the world. A key hypothesis of this work is that agents' different views of the world must be modeled as distinct, possibly conflicting, state vectors, rather than noisy observations of a *single* state vector.

For example, in distributed, multi-agent video games, different players will have different values for a given player's position, health, etc. Due to latency in network communication, these values cannot be perfectly synchronized. Our model explicitly acknowledges this fact, with the goal of enabling agents to reason how those differing views of the world will impact other agents' behavior and could change their own behavior. Human players in video games do this to some degree, adjusting their strategies when observing different amounts of lag in the game. For example, a player

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

might avoid precision targeting of an opponent whose position appears to change erratically due to lag, or, more commonly, rage quit.

As another example, consider autonomous vehicles approaching an intersection and trying to determine whom has right-of-way. It is important to recognize state—e.g., which vehicle arrived first—will differ among vehicles' views. While one might argue that the ground truth is defined based upon the physical positions of the vehicles in the real world, none of the vehicles actually have this information. Rather, they have separate models of the intersection *possibly communicated via a network* to the other cars. So, from a decision-making standpoint, the physical position of the vehicles is irrelevant as the cars make their decisions based on their models. The physical world, however, does have an important role: it determines the rewards of actions: did the cars collide or not? Furthermore, all the agents' views of state will impact the future evolution of the system. Put another way, modeling each agent's state vector is important for reasoning about behavior in the system.

Rather than modeling the system as a single state vector of attribute-value (AV) pairs, each agent maintains its own state vector of AV pairs.

In general, the state of a system is $s_{i,j}^{k}$, where:

- k is the controlling process—more on this later,
- i is the state (AV pair) index, and
- j is the agent.

Typically, agents are across different machines, but they do not have to be. Since a machine can have more than one agent, we must distinguish between agents and machines.

Further, an agent can be (will be, in our case) *replicated* over the network. In this case, one copy of the agent *controls* the other copies of the agent. These copies of the agent that are being controlled are referred to as puppets. Here is an example: in a multiplayer video game each player controls a character on his machine. That character is replicated over the network to appear on all the other players' machines. However, those replicated characters—puppets—are controlled by the player; these puppet agents are not able to make decisions on their local machines. In a well-designed game, it *appears* to the player that all the characters are acting under their own direction.

We define the controlling agent for a given AV pair to be the unique agent that can write to that state variable. So the controlling agent, k in our above notation, is the agent that has the ability to update $\{s_{i,j}^{k}\ \forall\ j\}$. Changes to these AV pairs can only come through messages sent by the controlling agent. For example, if $\{s_{3,j}^{k}\ \forall\ j\}$ represented Agent 1's location and Agent 1's puppets' locations, then $k = 1$. Further, we can think of $s_{3,2}^{1}$ as the location of Agent 1's puppet on Agent 2's machine. The puppet itself would not be able to change its location. Instead, any changes to $s_{3,2}^{1}$ would come as a result of a message from Agent 1.

We note that it is possible for $s_{i,j}^{k} \neq s_{i,m}^{k} if m \neq j$ (Equation 1).

For example, in the system $s_{i,j}^{k} = a, s_{i,m}^{k} = \neg a$ (Equation 2) is not only possible, but likely if network delays are greater than zero. We note that this property is *not transitory nor inconsistent.* While it could be possible to use some synchronizing event to resolve the state of an agent with its puppets, this would require the loss of real-time decision making as the messages required to synchronize are necessarily not part of the normal agent communication. As there is no underlying world that the agents observe (to borrow POMDP language), the resolution of the agents to the same value is not required (nor necessarily preferred). The seemingly strange implication of Equation 2 is that, from the view of other agents in the system, neither agent is correct even though both refer to the same attribute.

For example, suppose that i is position. With reasonable network delays, it is possible that an agent is at $(1,1)$ and its puppet is at $(0,0)$. Both locations are correct. Furthermore, other agents on the agent's machine will make decisions based on location $(1,1)$, while those on the puppet's machine will make decisions based on location $(0,0)$. All those decisions are valid, if mutually inconsistent. This is because from the viewpoint of these other agents, those locations are the only ones they know!

We further note that the locations of the agent and its puppet may *never* be the same. Even if the agent sends regular and timely position updates, those updates will necessarily be delayed and occasionally lost. The matter is made worse as typically puppet locations are interpolated based on received locations to make the motion look smooth to a player. Thus, the puppet may be in a position that the agent was never in.

The network's purpose in our model is to assign state between agents. Thus, generally the network operation is: $s_{i,j}^{k} \leftarrow s_{i,m}^{k}$ (Equation 3), where ← means assignment as used in programming languages. We impose the stronger requirement, that Equation 3 holds if and only if $m = k$. In other words, agents can write only to their puppets, but not vice versa.

Fully distributed problems can exhibit strange behavior to observers, such as *teleporting* in multiplayer, networked games (we're looking at you, Halo). In the case of teleporting, the agent on machine *j* must have a position out-of-date from the agent on machine *m*. At some point, the agent on *j* will accept the received position and display it to the player. Typically, this results in the character on the screen changing position suddenly and often jarringly. Methods for handling this are well known, but they cannot remove the problem.

Agents in POMDPs

The agents in POMDPs are considered to view the same underlying system. Thus, agents either do not have replicated state or do not describe them. In other words, state is defined in such a way so that Equation 2 cannot hold. In those systems where there is a shared state, the entire decision-making process removes any inconsistency among state variable values. In Equation 2, our system relaxes an assumption made by MDP and POMDP systems.

Time

It is time to discuss time, which is a required element of any real-time system. We make no claim on how time is measured—milliseconds, ticks, *etc.*—only that the clock provides a strictly increasing value. In some algorithms, time can be measured using *vector clocks*, which are integer vectors that count the order in which network messages are sent (Dumnich and Birmingham 2020). These are able to give a partial ordering of events when a total ordering is not possible due to the asynchronous nature of independent machines with network latency.

While time is a state variable like any state variable, it is so important that we distinguish it: $t_{i,j}^{\emptyset}$ (Equation 5). We use $\emptyset$ to show that no agent controls the clock, it is controlled by solely by the machine; an agent may only read it. Further, we say that $t_{i,j}^{\emptyset} \leftarrow t_{i,k}^{\emptyset}$ is meaningless since neither agent controls its clock.

Messages

A message has the form: $m\left(j, s_{i,l}^{l}, t_{m,l}^{\emptyset}\right)$ (Equation 6), where agent l sends a message to agent j to update an AV pair indexed by i. The sending agent, l, includes its time, $t_{m,l}^{\emptyset}$. Time is not necessary for the message; however, because the agents operate in real time and will usually execute algorithms to compensate for lag, time is practically necessary. Further, by including time in messages, agents have at least a partial ordering to their messages and have the option to disregard outdated information.

For example, consider the system described in Figure 1. The arrows drawn between the agents represent the controlling agent for each part of state. In this example, $s_{1,j}^{1}$ is controlled by Agent 1 and $s_{2,j}^{2}$is controlled by Agent 2. We have $t_{3,1}^{\emptyset}$ and $t_{3,2}^{\emptyset}$ as time for Agent 1 and Agent 2, respectively. Further, we know that the messages sent between agents are subject to random delay. As the machines execute the processes, state values could change as depicted in Figure 2.

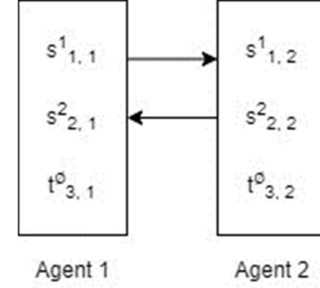

Figure 1. Example of state with two agents. Here there are two AV pairs in addition to time.

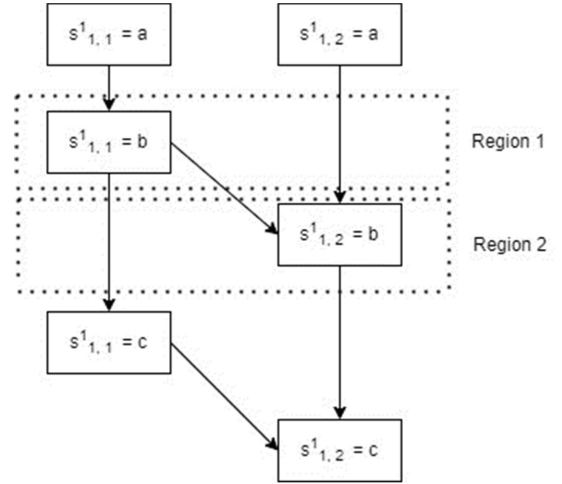

Figure 2. Two messages are sent from Agent 1 to Agent 2 to update $s_{1,2}^{1}$. In Region 1 the states are inconsistent while in Region 2 they are consistent.

In Region 1 (regions are for explanation purposes only), the shared attribute s_1 has different values for each agent. Thus, we have two equally valid, yet inconsistent states between agents. The inconsistency may never be resolved in general (although it is in this example). As to the apparent inconsistency – there is no single correct state. Region 2 is no more correct than Region 1. It seems natural to say Region 2 *is* correct and Region 1 is transitory, and therefore, "less correct." However, Machine 2 operates completely independently of Machine 1 *until* it receives a message, which it cannot possibly know is coming. The actions taken by the agents on both machines are valid because that is all they know.

Summary

Classic models of multiagent systems assume one common state vector from which agents receive different observations. The fact that communication between agents takes time is not explicitly modeled, nor is the fact that those communication delays cause agents to have inconsistent state values. If these delays are not taken into account, a learning agent may unknowingly learn a suboptimal policy. We propose a model that makes these realities explicit. By including agents' inconsistent—but semantically related—state vectors in the model, the goal is to enable agents to reason about behaviors under such conditions, which match those of physical real-time systems.

References

Dumnich, S.; and Birmingham, W. 2020. It's About Time: Vector Clocks and Distributed Systems. In Proceedings of the Second Joint SIAM/CAIMS Annual Meeting.

Messias, J.; Spaan, M. T. J.; and Lima, P. U. 2013. Asynchronous Execution in Multiagent POMDPs: Reasoning over Partially-Observable Events. In Proceedings of the Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM-2013).

Spaan, M. T. J.; Oliehoek, F. A.; and Vlassis N. 2008. Multiagent Planning Under Uncertainty with Stochastic Communication Delays. In Proceedings of 18th Int'l Conference on Automated Planning and Scheduling.