Distributed Discrete Event and Psuedo Real-time Combined Simulation for Multi-agent Controlled Power Plants

Joel H. Van Sickel, *Student Member, IEEE*, and Kwang Y. Lee, *Fellow, IEEE*

***Abstract*—Development, testing, simulation, and validation of complex real-time distributed systems is a challenge on many levels. Designing multi-agent systems for power plant control can be approached numerous ways, but it is important to enhance the flow of development to implementation. Using new Distributed Real-time Agent Framework with Time-warp (DRAFT) agents can enhance the development of multi-agent systems. Discrete event simulation allows a fast as possible simulation and forces synchronization while guaranteeing all tasks are accomplished on time to assist with initial stages of development when it is more important to focus on larger design concepts. The DRAFT agents can switch over to standard operation and operate on psuedo real-time or real-time environments to further develop all aspects of a multi-agent system with real-time requirements. It also gives the agents self simulation capabilities that could be used to enhance overall functionality of the multi-agent system.**

***Index Terms*—Discrete event simulation, multi-agent system, optimistic simulation, power plant, real-time, time warp.**

I. Introduction

DEVELOPING a real-time multi-agent systems is a challenging prospect in a nascent domain with few design standards. Numerous approaches have been taken by many different groups in research and industry, all valid with different tradeoffs. This paper seeks to develop a Distributed Real-time Agent Framework with Time-warp (DRAFT) agent that enhances the overall design process for multi-agent systems. The work is based on common design approaches taken in developing complicated distributed real-time systems.

A very common design practice in developing these systems is to have an initial stage where the goal is to create a simplified simulation for the purposes of testing general concepts, hierarchies, and big picture strategies while ignoring many of the smaller details involved in the development of a distributed real-time system. These simulations are typically not high fidelity and numerous bugs and inconsistencies will become evident once implemented in a real-time system. This is further exacerbated by the common practice of having this first stage implemented by teams consisting mainly of software engineers familiar with sequential programming practices and lacking experience in real-time and distributed applications. Also, this first stage does not provide an opportunity to learn more about real-time and distributed programming.

The second stage is often a group with the experience required to take the original work and implement it in real-time hardware, possibly the final product, or a test bed to further examine the system before its real world implementation. Some errors detected at this stage must then be sent back to the first stage, and the process is repeated.

Finally, after a quality design is fully fleshed out, it can be implemented in the real world environment it was intended for. This process has the benefit that it catches a lot of problems before implementation, but reducing the time between switching between the first 2 stages will further speed the development process.

A totally separate design process that is sometimes undertaken is that the entire system is designed from the ground up in the real-time environment from the very beginning. A benefit of this is that there is no jumping between design stages. However, this approach is challenging because it is harder to break into tractable pieces and it is hard to isolate problems.

The DRAFT agents seek to address the design issue by following a route similar to the multi-stage approach, but combine the first two stages onto the same development bed. Using distributed discrete event simulation and forcing synchronization, the multi-agent system can simulate itself, but forces all messages to be received on time, and forces all tasks to be accomplished on time while performing fast as possible simulation. This is ideal for the initial stages of development where the larger issues can be focused on without worrying about timing issues and synchronization.

Once this has been accomplished, the agents can toggle to standard operation where they act as they would in a real-time environment, but are attached to a pseudo real-time simulation environment instead. This allows a higher fidelity simulation where other issues can be addressed without having to rewrite any code, significantly decreasing development time and enhancing the testing process. This is shown in Fig. 1.

This work was supported in part by the National Science Foundation under grant ECCS 0801440.

J. H. Van Sickel is with the Department of Electrical Engineering, the Pennsylvania State University, University Park, PA 16802 (email: jhv107@psu.edu).

K. Y. Lee is with the Department of Electrical and Computer Engineering, Baylor University, Waco, TX 76798 (e-mail: Kwang_Y_Lee@baylor.edu).

978-1-4244-5098-5/09/$26.00 ©2009 IEEE

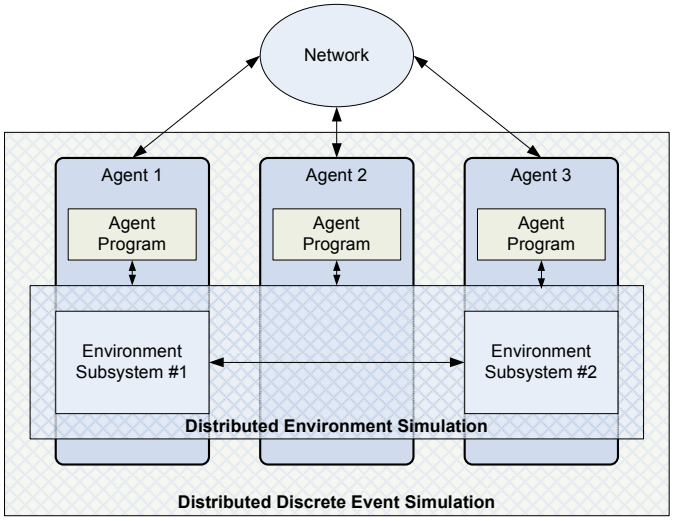

Fig. 1. DRAFT Simulation Set up.

II. Discrete Event Simulation

A new fundamental change to the agents in this multi-agent system is that they all are embedded with discrete-event simulation capabilities. These agents employ a scheme based on time warp and optimistic simulation [1].

Distributed simulations offer the advantage of faster simulations. Each processor involved in a distributed simulation is referred to as a logical process (LP). Each LP is responsible for simulating one piece of the full simulation and giving its data to the other LPs which require it for their own simulation purposes. The challenge is to keep the LPs synchronized. If the simulations process events are out of order, this is referred to as breaking the local causality constraint. Overcoming this issue is traditionally accomplished in one of two ways, conservative and optimistic simulations.

There are two types of simulation that were considered for this application, conservative and optimistic. Conservative will be discussed briefly but since optimistic techniques were chosen in the end, that is where the focus of this paper will lie.

A. Conservative Simulation

The most obvious way to keep each LP synchronized is to force them to constantly communicate with each other to ensure that no LP goes ahead in simulation time until every other LP is ready to move on as well. There are few issues involving detecting a deadlock that have been well addressed in literature [2] and should not hinder using such a scheme. Traditionally, unless more advanced features are added to a conservative simulation system, a lot of time can be spend waiting for other LPs before moving onto the next calculation. Regardless of how this type of simulation is implemented, it will also have a large amount of extra messages being sent to maintain synchronization.

Shown below is an example of how conservative simulation works and can achieve faster than real-time performance when simulating a real-time system. It also shows how the LPs wait for the other LPs before moving on to the next time step.

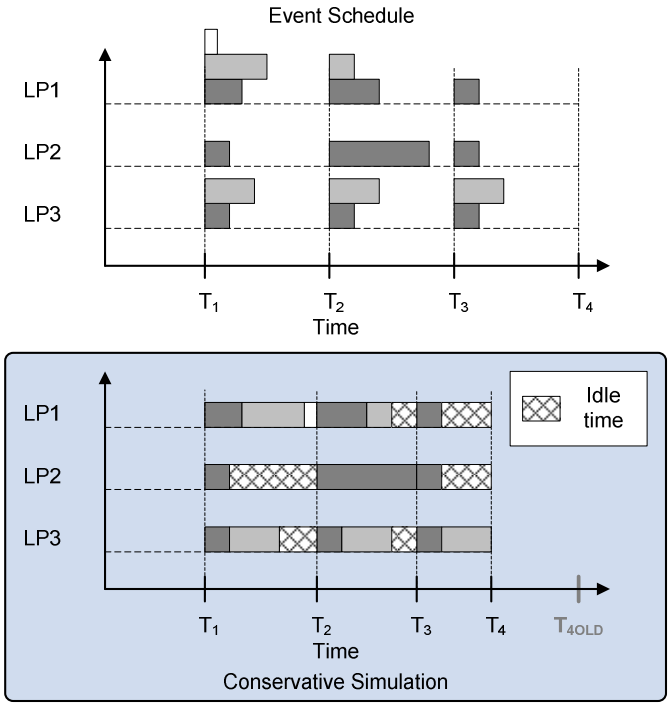

Fig. 2. Conservative Simulation.

One of the best ways to get good performance from a conservative system is to use look ahead. Look ahead allows LPs to continue with their own calculations further on in simulation time if they know that none of the current LPs are going to violate the local causality constraint. It can be compared to seeing into the future and knowing that there is a certain period of time for which messages will not be received from other LPs. This is far from a trivial and is also considered one of the most important aspects of a conservative simulation.

This is the main detraction from using a conservative simulation for a multi-agent system, as there is no way of determining each agent's look ahead as they are typically fairly complex programs that cannot be simplified into rules, or they would not be needed in the first place. This is even more true where heterogeneous agents are working together.

B. Optimistic Simulation

Optimistic simulation does not enforce the local causality constraint. Each LP is allowed to process at its own speed and ignore where other LPs are located in simulation time. It is called optimistic because each processor moves forward in time, optimistically assuming it has not violated the causality constraint. This will inevitably lead to the local causality constraint being broken at some point in simulation time [3].

To deal with this, a capability known as time warp is built into each LP, allowing them to step back, or time warp, to right before local causality was broken and proceed from there. This works because for local causality to be broken, it requires a time stamped message from another LP to be received. Once this happens, the LP that received the message too late now has the message it originally ignored, and can time warp back and proceed on, now having the correct

information in hand. This LP must alert the other LPs as well that they must go back as well.

This is an over simplified explanation and can be compared to a group of people running together, and whenever they realize that someone has fallen behind, they run back to that person, and then continue on, however, they are not checking all of the time, which means the runner who fell behind may be able to catch up before they run back. This can be contrasted to a conservative simulation where the runners would never let themselves outpace each other.

This optimistic scheme works because it is not unusual for an LP that falls a little behind to catch up later without ever having sent a message that would violate the local causality constraint. The challenge with an optimistic simulation is that time is wasted when recovering from violating the local causality constraint. If the recovery mechanisms are fast, it is not much of an issue, but even with fast recovery, if the LPs are so mismatched that they are continually having to time warp and resynchronize, the simulation could feasibly take longer than a single processor simulation. To prevent this, a mechanism can be used to slow down the faster LPs.

For this particular application, there is no lack of computing resources, but there is still a bandwidth requirement. This tips the scale in the direction of an optimistic simulation. Additionally, the look-ahead issue is either impossible or extremely challenging in heterogeneous agent applications. On the other hand, optimistic schemes do not require much additional communication overhead to perform correctly. As the full system was fleshed out, other additional benefits of using an optimistic simulation system described in Fig. 3 became apparent as well:

- Time warp requires all previous events be remembered, this makes debugging drastically easier as all data on the system is already available (unless memory management requires some to be deleted).
- Time warp messages are easily embedded into already necessary agent messages.
- Time warp capabilities are easy to deactivate, allowing the same code to be used for a 'fast as possible' optimistic simulation as well as a soft real-time simulation.

Returning to an earlier time for an LP is like an advanced *undo* feature. All that is required is that the LP rolls back its states to the time of the received message. This can be achieved through two main ways. The first method is full state saving. The other method means that for each event processed by the LP, only the states modified are saved. This process requires that to roll back, each previously processed event must be undone individually. The first method requires more memory, but it can quickly step back to previous times. The second method uses less memory but it takes longer to roll the states back. Which one takes less overhead while the LP is running normally depends heavily on implementation and number of state variables in the LP. Both of these methods are demonstrated in Fig. 4.

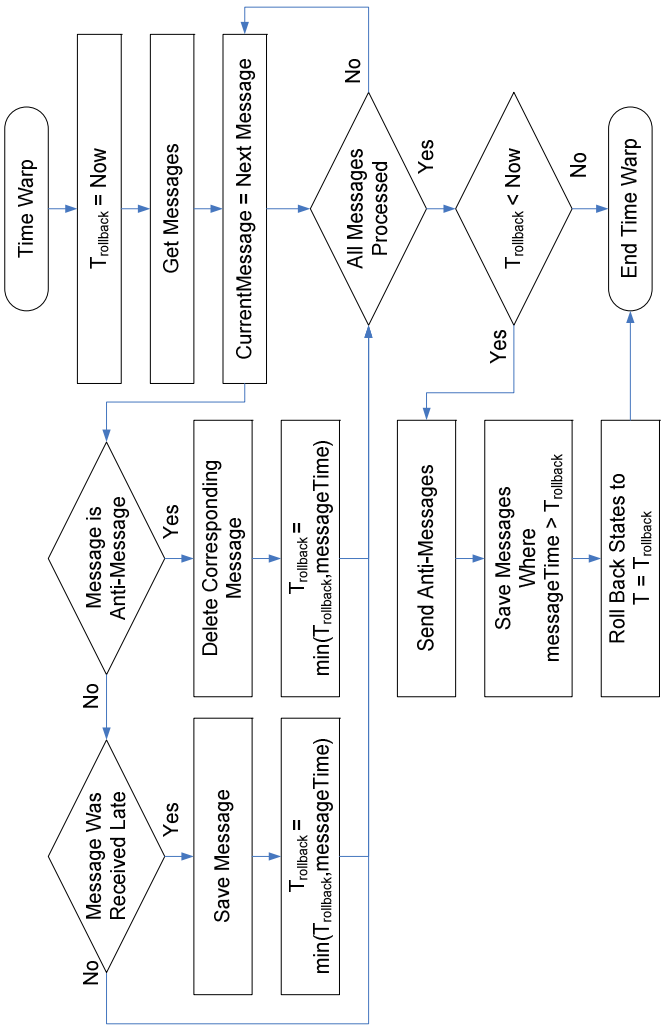

Fig. 3. Time Warp Mechanism.

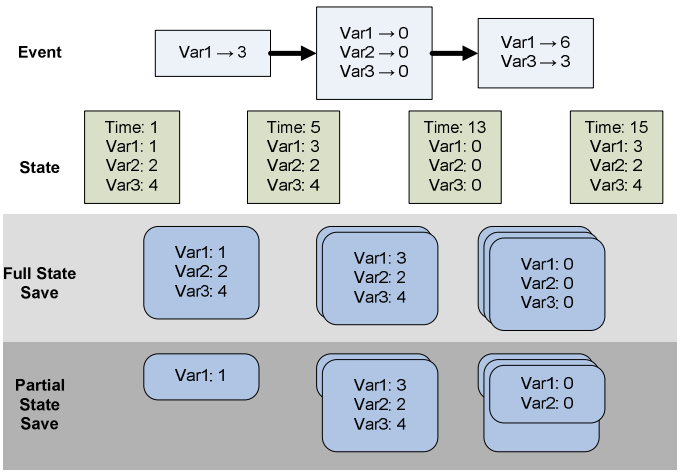

Fig. 4. Time Warp Mechanism with different state savings.

Besides rolling back the state variables, the second big issue is that of *unsending* messages. This is because a single LP may have sent messages to other LPs after the time which it needs to step back to. That means any actions taken by those LPs based on the message need to be undone as well.

This is achieved with a construct called an *antimessage*. An antimessage tells other agents which messages they have received that were sent too early. When an agent receives and antimessage, they will undo any actions caused by the message represented by the antimessage. All messages must have unique identifiers so the agents can determine which messages need to be undone. Therfore, each LP needs to keep track of all of the messages it receives and sends. When an antimessage is received, the LP responds similarly to as if it had received a late message, except that when it rolls back, it also deletes the message which corresponds to the antimessage. It, in turn, will send any required antimessages, and this causes a cascade effect so that all affected LPs eventually roll back to the correct time and receive all the message they should and delete the ones they should not have. An example of a time warp simulation is shown in Fig. 5.

Another issue is the global control mechanism [4]. Stave saving uses up more memory as the simulation progresses. The only way to be able to free some of this memory is to know a certain time that will never be 'time warped' back to. This time is called Global Virtual Time (GVT). GVT is more explicitly defined as the smallest time stamp of all the messages (including anti-messages) and events that are being processed or are unprocessed. Calculating GVT does not have to be precise, as a conservative answer will still allow some memory to be freed. Currently, memory is not a restricting factor, and keeping all data is desirable for testing and debugging purposes, so a GVT algorithm is not used. If at some point this work must be extended to include a GVT mechanism, it is recommended that Samadi's GVT algorithm [5] be used as it only requires acknowledgements to be added.

III. Simulation Performance

The DRAFT agents can be used for numerous purposes and can be organized in different manners. Three different simulation cases are demonstrated to show the performance of the system and the effect of using time warp. The first two simulations represent simplified versions of the multi-agent system and the final case represents the multi-agent system running at full capability for all agents. Some agents not shown are the interface and fault diagnosis agents [6] which do not directly connect back into the multi-agent system and are primarily used for assisting humans. Also not included is a server agent which is of no interest in standard operation. Important to note is that the simulation cluster is made of multiple agents that are all synchronized by data streams. Major agents in this simulation are a reference governor agent [7], gain tuning agent [8], and identification agent [9].

A. Simulation Case 1

The first simulation case is over simplified, but is a nice example of the system's operation and is shown in Fig. 6. Fig. 7 shows the times taken for the different agents to perform the simulation. The statistics of the performance are shown in Table I. The specific performance on a single run is shown in Fig. 8.

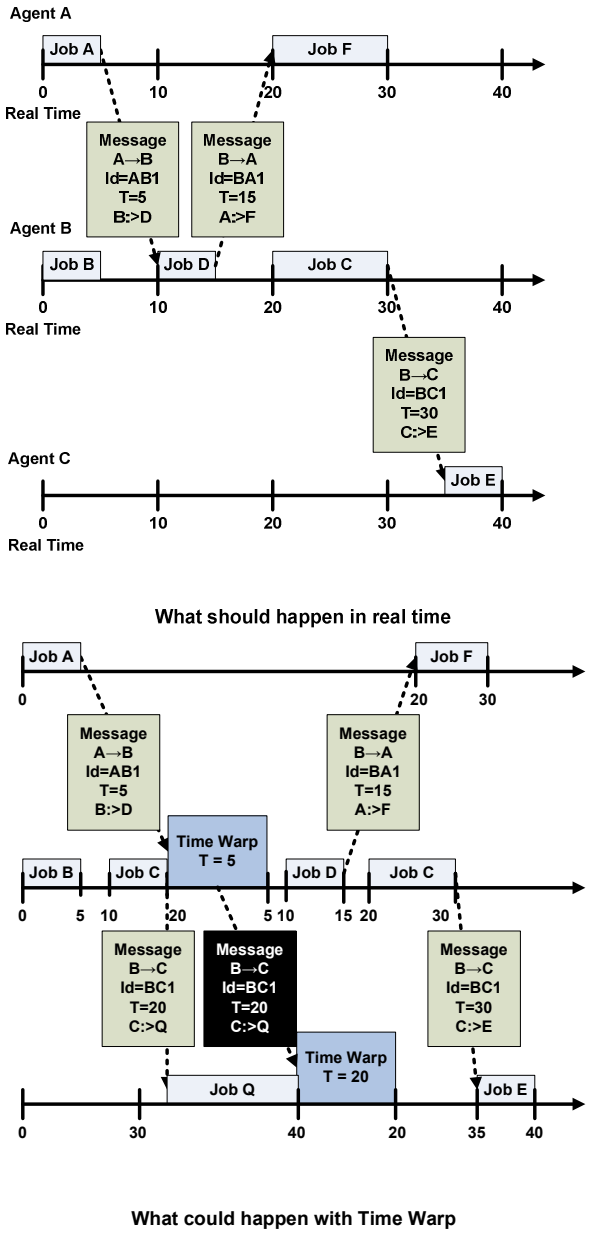

Fig. 5. An example of Time Warp Mechanism.

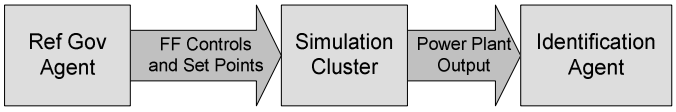

Fig. 6. Simulation Case 1.

TABLE I. Results for Simulation Case #1 – single machine.

	Ref Gov Agent	Sim Cluster	Id Agent
Avg Sim Time [s]	58.95	75.83	73.08
Max Sim Time [s]	61.98	84.07	85.52
Min Sim Time [s]	57.29	73.95	70.32
Avg Anti Mesg Sent	0	1.00	0
Avg Anti Mesg Rec	0	0	1.00
Late Messages Rec	0	2.00	9.24
Avg. Time Warps	0	2.00	9.20

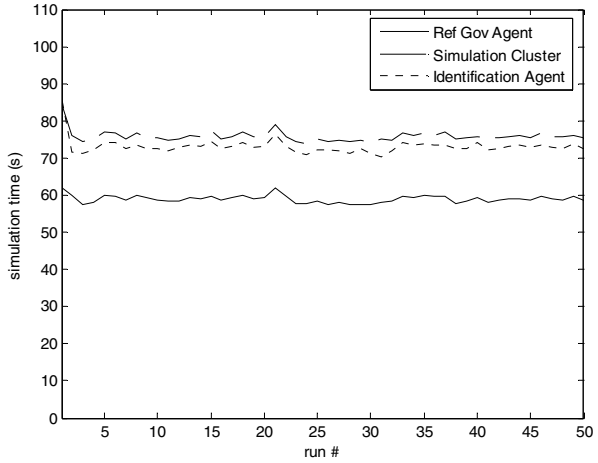

Fig. 7. Simulation Times for Simulation Case #1.

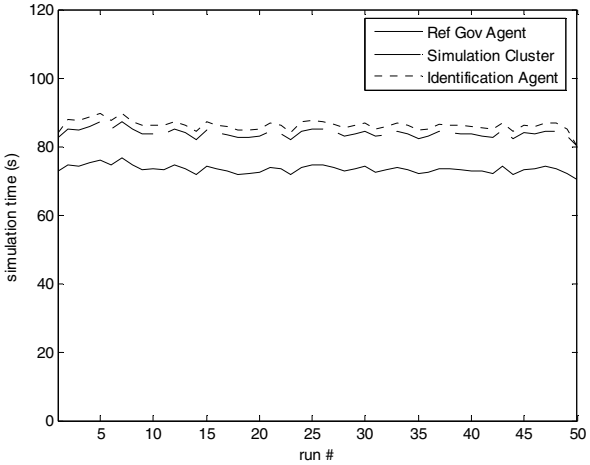

Fig. 10. Simulation Times for Simulation Case 2.

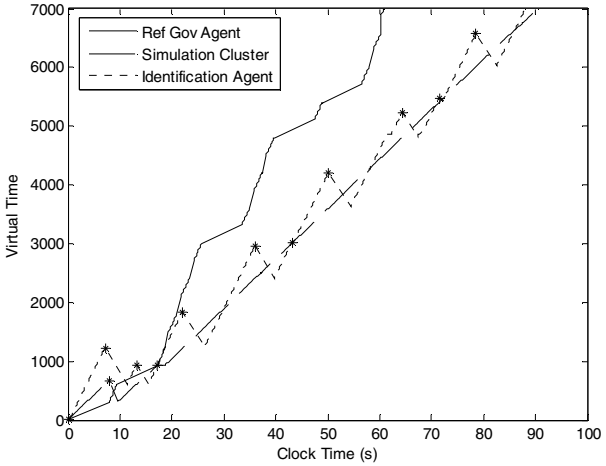

Fig. 8. Simulation Performance for Simulation Case #1.

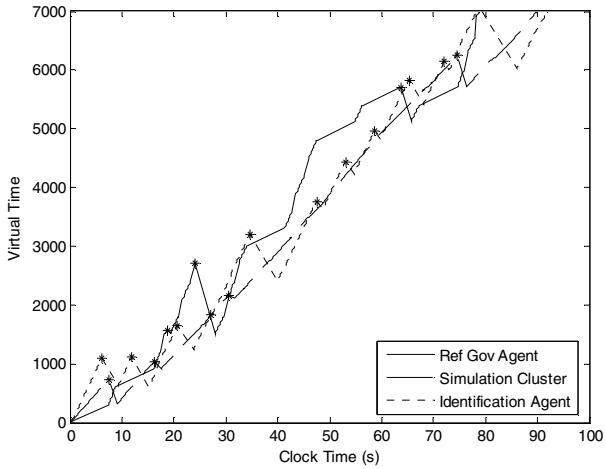

Fig. 11. Simulation Performance for Simulation Case 2

B. Simulation Case 2

The second case is similar to the first case with the configuration in Fig. 9. Figs. 10 and 11 along with Table II show the performance of simulation case 2.

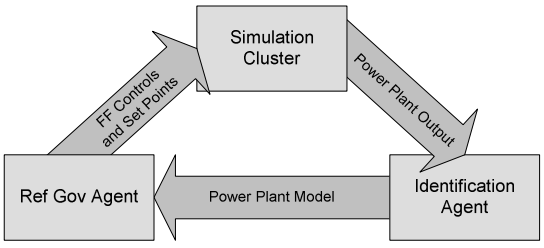

Fig. 9. Simulation Case 2.

TABLE II. RESULTS FOR SIMULATION CASE #2 – SINGLE MACHINE.

	Ref Gov Agent	Sim Cluster	Id Agent
Avg Sim Time [s]	73.39	83.85	86.11
Max Sim Time [s]	76.68	87.20	89.71
Min Sim Time [s]	70.27	80.16	80.02
Avg Anti Mesg Sent	3.32	2.12	2.62
Avg Anti Mesg Rec	2.62	3.32	2.12
Late Messages Rec	3.92	3.02	11.72
Avg. Time Warps	2.20	3.02	11.72

C. Simulation Case 3

The final case is using all of the standard multi-agent functionality that involves communication loops. Fig. 12 shows the agent configuration and the results of performance are shown in Figs. 13-14 with the data in Table III.

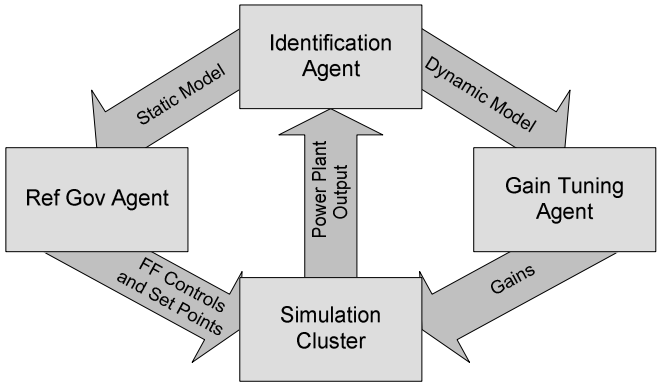

Fig. 12. Simulation Case 3.

TABLE III. RESULTS FOR SIMULATION CASE #3 – FULLY DISTRIBUTED.

	RG Agent	Sim Cluster	Id Agent	GT Agent
Avg Sim Time [s]	132.25	145.46	159.29	145.04
Max Sim Time [s]	145.30	157.92	171.45	157.67
Min Sim Time [s]	123.80	136.49	150.02	136.19
Avg Anti Mesg Sent	40.13	7.60	12.26	1.40
Avg Anti Mesg Rec	6.60	21.40	7.53	25.73
Late Messages Rec	8.53	23.20	18.60	6.46
Avg. Time Warps	6.26	23.13	18.60	6.46

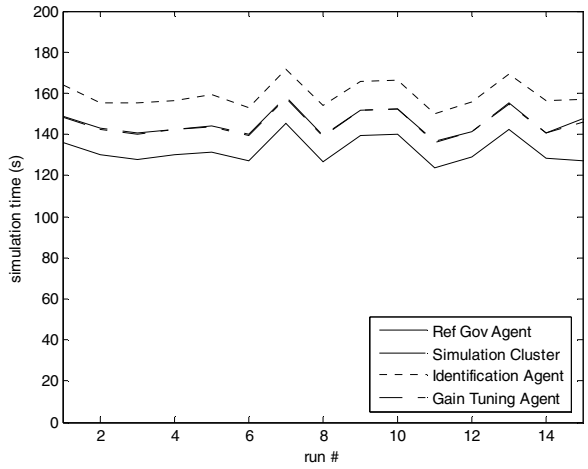

Fig. 13. Simulation Times for Simulation Case 3.

D. Discussion of Results

The new distributed simulation works well. The times between Cases 1 and 2 cannot be compared to Case 3 which was fully distributed on a different set of computers. Case 3 was also run for an extended time to show how, near the end, the simulation slow down. This is due to the fact that as time passes each agent continues to use more and more memory to store past actions. As mentioned before, this can be solved using GVT, or reducing the number of times an agent cycles through its logic process when idle. Overall, this is a promising approach for developing multi-agent systems.

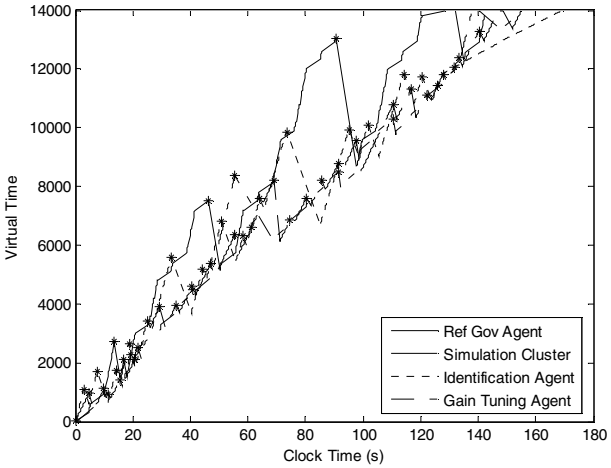

Fig. 14. Simulation Performance for Simulation Case 3.

IV. CONCLUSIONS

Using DRAFT agents has proven an effective way to combine the continuous simulation of a power plant with the discrete nature of a multi-agent system and achieve fast results which take advantage of the distributed nature of agents. Optimistic techniques proved to work well and have numerous implementation benefits. Using this approach has proven invaluable in the development of the multi-agent systems used for power plant control.

V. REFERENCES

[1] R. M. Fujimoto, "Optimistic Approaches to Parallel Discrete Event Simulation," Transactions of the Society for Computer Simulation, v 7, pp. 153-191, 1990.

[2] D. W. Jones, "An Empirical Comparison of Priority-Queue and Event-Set Implementations", *Comm. ACM*, 29: 300-311, 1986.

[3] D. R. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, pp. 404-425, 1985.

[4] L. Yi-Bing, "Design Issues for Optimistic Distributed Discrete Event Simulation," *Journal of Information Science and Engineering*, v. 16, pp. 243-269, 2000.

[5] B. Samadi, *Distributed Simulation, Algorithms and Performance Analysis*. Ph. D. Thesis, University of California, Los Angeles (UCLA), 1985.

[6] J. S. Heo, and K. Y. Lee, "A Multi-Agent system-based intelligent identification system for power plant control and fault-diagnosis," *2006 IEEE Power Engineering Society Meeting*, 2006.

[7] J. H. Van Sickel, and K. Y. Lee, "Reverse Normal-boundary Intersection for Multi-objective Optimization for Power Plant Operation," *IFAC Symposium on Power Plants and Power Systems Control*, 2009.

[8] J. H. Van Sickel, K. Y. Lee, and J. S. Heo, "Differential Evolution and its Applications to Power Plant Control," International Conference on Intelligent Systems Applications to Power Systems, 2007

[9] J. S. Heo, and K. Y. Lee, "A Multi-agent System-based Intelligent Steady-state model for a Power Plant," *13th International Conference on Intelligent Systems Application to Power Systems*, 2005, pp. 419-424.

VI. BIOGRAPHIES

Joel H. Van Sickel received his B.S. degree in Electrical Engineering from Grove City College in 2005. He is currently pursuing his Ph.D. in Electrical engineering at the Pennsylvania State University. His interests include distributed artificial intelligence, distributed simulation, intelligent control, power generation, multi-agent systems and optimization techniques.

Kwang Y. Lee received his B.S. degree in Electrical Engineering from Seoul National University, Korea, in 1964, M.S. degree in Electrical Engineering from North Dakota State University, Fargo, in 1968, and Ph.D. degree in System Science from Michigan State University, East Lansing, in 1971. He has been with Michigan State, Oregon State, Univ. of Houston, the Pennsylvania State University, and Baylor University where he is currently a Professor and Chair of Electrical and Computer Engineering. His interests include power system control, operation, planning, and intelligent system applications to power systems. Dr. Lee is a Fellow of IEEE, Associate Editor of IEEE Transactions on Neural Networks, and Editor of IEEE Transactions on Energy Conversion. He is also a registered Professional Engineer.