Chapter 7
A Brief History of Time Warp

David Jefferson and Richard Fujimoto

Abstract This chapter is about the history of the Time Warp algorithm and optimistic approaches to parallel discrete event simulation. It concentrates on the early history from our personal perspective as active developers of the ideas over several decades.

7.1 Introduction

Time Warp is the name of an algorithm for doing discrete event simulation in parallel. It is an *optimistic* simulation mechanism, one that takes risks by performing *speculative* computation which, if subsequently determined to be correct, saves time, but if incorrect, must be rolled back. Time Warp can be used in any computation that uses a global temporal coordinate system for synchronization, but *discrete event simulation* using *simulation time*, is by far the most important example.

Because of its complete embrace of distributed rollback as the fundamental synchronization primitive instead of more conventional primitives such as locks, semaphores, or other process blocking constructs, Time Warp was considered a radical innovation when it first appeared. It is even today, after 35 years, virtually unique in that respect, but it has proved to be an elegant and powerful parallel algorithm, able to achieve excellent parallel performance at a scale of almost 2 million cores with 8 million threads and 250 million LPs so far (Barnes et al. 2013).

This chapter is a brief history of the development of Time Warp, largely concentrating on the 1980s and 1990s. It is necessarily incomplete, and is entirely from our personal perspectives. But this volume on the 50th Anniversary of WSC seems

D. Jefferson (✉)
Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA 94550, USA
e-mail: drjefferson@gmail.com

R. Fujimoto
Georgia Institute of Technology, Atlanta, USA

A. Tolk et al. (eds.), *Advances in Modeling and Simulation*, Simulation Foundations, Methods and Applications, DOI 10.1007/978-3-319-64182-9_7

to be the perfect place in which to recall this early history. The first part of the chapter is by David Jefferson, and the second part by Richard Fujimoto. We hope that other authors will contribute their recollections as well.

7.2 The Early History of Time Warp: David Jefferson's Perspective

This first section provides the perspective of David Jefferson, describing the early years of Time Warp at RAND, JPL, and within the Jade projects.

7.3 Origin of Time Warp at RAND

The development of Time Warp began with a project at the RAND Corp. in Santa Monica in 1981. The Air Force was funding research to improve military simulations in two ways: first, to allow models to be specified in a quasi-natural language so that nonprogrammer generals might build their own models or scenarios, and second, to speed up simulations through parallelism. I was a young assistant professor at the University of Southern California with a background in parallel computation, which at that time was relatively rare, so they recruited me to work as a consultant on the project. I thought the first objective of natural language model building was unlikely to succeed, but I might be able to contribute toward the second of parallelizing discrete event simulations.

I had never read any literature on parallel discrete event simulation (PDES), of which there was not much, and I did not immediately do a literature search. I just started working on the problem from first principles. I was familiar with the sequential discrete event simulation algorithm based on a priority queue, and I thought that creating a parallel version of the algorithm would probably be straightforward. That, of course, turned out to be exceptionally optimistic because now, 35 years later, there are still basic issues, like load balancing or the relationship of PDES to the continuous simulation techniques used for numerical solution of ordinary and partial differential equations, that we are still trying to understand about PDES.

On my first consulting day at RAND, I recognized the core problem as synchronizing the interaction among many concurrently executing processes (called "logical processes", or LPs) sending timestamped event messages to one another. Every LP receives a stream of timestamped event messages sent by other LPs. The incoming messages at an LP do not generally arrive in increasing timestamp order, and in the general case there is no limit on how far out of order they are. Nonetheless, each LP must process event messages strictly in increasing timestamp order. If and only if all LPs do that (and have the same tie-breaking rule as well), the resulting simulation is equivalent to that of the sequential algorithm, as required. This was similar (with some minor differences) to the way Chandy and Misra (1979, 1981), and independently Bryant (1977), had already framed the problem, though I did not know it yet. The most important conceptual difference was that they assumed event messages were transmitted within a static graph of order-preserving message channels between LPs, whereas I assumed any LP could send an event message to any other at any time, and without requiring order preservation.

I quickly realized some key facts about the synchronization problem in parallel discrete event simulations. Except in special cases, one cannot achieve any significant parallelism by trying to keep all the LPs of a simulation tightly synchronized in simulation time, or requiring that events in different LPs be executed in increasing simulation time order. Any such attempt to constrain the simulation would over-synchronize it and effectively sequentialize execution, no matter how many processors were used. Instead, a parallel simulator must allow some LPs to run ahead in simulation time while others lag behind, with no a priori bound on the time difference. Also, which LPs are ahead or behind must be able to change dynamically as the simulation progresses. There cannot be a single, globally shared standard for simulation time as there is in the sequential algorithm and in parallel time-stepped simulation algorithms. Instead, each LP would need its own simulation clock, and in any hypothetical instantaneous snapshot two such clocks would rarely, if ever, agree. These observations applied even in the context of shared memory parallelism, though we were interested in distributed algorithms.

I do not recall the exact moment of realization, but at some point it occurred to me to think in terms of *asynchronous parallel rollback* as a possible synchronization primitive instead of the classical primitives based on process blocking and resumption. In the late 1960s, rollback had been used in limited ways in sequential (but not parallel) debuggers (Balzer 1969). And rollback limited to the scope of a single transaction (transaction abortion) had recently been studied in the context of optimistic parallel database synchronization (Kung and Robinson 1981). Other work I was at least dimly aware of that may have had some influence on me included studies of parallel backtracking algorithms in such applications as parallel tree search, alpha–beta pruning in game trees, and branch-and bound optimization.

But while rollback was conceptually straightforward in the context of sequential computation or transaction abortion, how could it possibly work at all in the general case of an asynchronous parallel computation communicating by timestamped messages, let alone work efficiently? I did not know, but I felt forced in that direction because I had been able to construct simple artificial models with just two or three LPs where it seemed impossible to achieve enough parallelism without rollback. But in implementing rollback, how do you undo the fact that the process being rolled back may have sent out event messages that should be recalled, and the receivers of those messages may by now have sent secondary event messages that also should also be recalled? There would be a potentially large, dynamically- and asynchronously expanding tree of event messages that must all be recalled because of the original rollback, and that tree is growing, potentially exponentially, even while the rollback is in progress! And in fact many parallel and uncoordinated rollbacks could be going on simultaneously in various parts of the simulation that would potentially interact and interfere with one another in unpredictable, nondeterministic ways. How was it possible to accomplish a clean rollback in the context of such a chaotic mess?

These considerations were all something of a disturbing surprise, and it concerned me enough that I checked with the project leader at RAND, Phil Klahr, to be sure that he was OK with me exploring what seemed like radical approaches to the problem of parallelizing discrete event simulation. To his great credit Klahr said that he did not care how I proposed to parallelize a simulation as long as the algorithm achieved speedup and got the correct results, i.e., the same results as would be produced by sequential execution. That freed me to think as far outside the box as I wanted.

7.4 Collaboration with Henry Sowizral

At this point, Klahr also made another crucial decision. He teamed me with a RAND researcher, Henry Sowizral. This turned out to be an extraordinarily fruitful partnership. After filling Henry in on my thinking, he immediately saw where it was going and we worked closely together thereafter. Over the next 9 weeks during my Friday consulting visits to RAND the two of us jointly developed most of the core ideas of what came to be known as Time Warp. Our research "method" was to walk outside along the Santa Monica Pier, the beach, and the palisades above, enjoying the sun and the sights, and brainstorming continuously for hours about PDES.

That collaboration, though lasting less than 2 years, was one of the most productive in my life. Henry and I were complementary and coequal. We adopted speculative execution with general distributed rollback as our computational paradigm, and then systematically rethought virtually every other issue in distributed computation with rollback in mind. This was, and still is, a radical departure from other paradigms of parallel computation. It allows the computation to execute speculatively *down completely incorrect computational paths* that were never intended or envisioned by the programmer. Eventually the incorrectness is detected by a causality violation signaled by the arrival of an event message with a timestamp in the simulation past of the receiving LP. That causes a cascade of actions in which all LPs in the distributed simulation that have been directly or indirectly affected by the original erroneous speculative event may be rolled back to times in their past when their dependencies on the original error began. Execution then proceeds forward in all of them again down a (more nearly) correct path, in a manner that might be described as many parallel backtracking "searches" for the correct forward paths.

The fact that this approach could be made to work at all was astonishing to us. It seemed totally counterintuitive that you could profit by (a) doing some distributed computation speculatively, which might be utterly wrong, while also paying substantial additional overhead to allow for possible rollback, and then (b) also sometimes paying the cost of distributed rollback to undo the speculative computation when required, before (c) finally redoing the correct computation. But we became confident, without real proof yet, that the resulting parallel computation could be faster than an algorithm in which you carefully refrained from doing anything speculative at all, at least sometimes.

But we were still faced with a lot of difficulties. How could a rollback mechanism be implemented so that the speculative execution converged stably and deterministically on the correct execution? How could we guarantee that the simulation made forward progress instead of thrashing forever in rollback activity? If it was theoretically possible at all, how could it not be overwhelmed by state-saving or synchronization overhead? How could we reclaim memory and avoid filling it with data needed to support rollback? And how could this possibly scale well in a distributed-memory platform?

Henry and I kept struggling with these problems. We conceived of a mechanism whereby incorrect event messages could be "unsent" by sending special "cancellation" messages that tell the receiver to throw away the cancelled event message and to undo whatever further computation they may have done based on it. That was promising, because it would induce a tree of such cancellation messages and rollbacks in other LPs as necessary. But would this growing tree of cancellations ever converge? What if, after an LP sent a cancellation message, it had to roll back again to a time before it sent the cancellation? It seemed that we would then have to send out a second-order cancellation message to cancel the previous cancellation message, and third-order cancellations might be required to cancel erroneous second-order cancellations. Would not that mean we would need an infinite hierarchy of higher order cancellation types?

At this point an idea occurred to us that I consider the most beautiful one in the core of Time Warp. Cancelling a cancellation message could be made indistinguishable from resending the original event message, and in that case a third-order cancellation message was indistinguishable from a first-order cancellation. There was an even–odd parity at play, and we formalized it in terms of a message–anti-message duality. A regular event message would be considered "positive" and a cancellation message would be considered "negative", and they were symmetrical. Either one could cause a rollback, and whenever two messages that are identical except for sign were enqueued in the same message queue they would "annihilate" and just disappear. The notion of anti-messages and annihilation made asynchronous distributed rollback work cleanly, and it did so even in all the complex cases. It worked even though the timing was nondeterministic and messages were asynchronous and arbitrarily delayed, and even when messages were not delivered in FIFO order, and even when there were cycles in the communication graph, and when multiple, asynchronously interacting and mutually interfering rollbacks were simultaneously in progress, and when anti-messages were delivered before the messages they were supposed to cancel, and it even worked when *all* anti-messages were delivered systematically more slowly than positive messages! Furthermore, the anti-message mechanism scaled easily to an arbitrarily high degree of parallelism. It seemed miraculous that such a simple, elegant mechanism as timestamped anti-messages could also be so powerful, scalable, and robust. (It still seems that way to me.) Henry and I repeatedly discovered that every issue created by the introduction of distributed rollback seemed to have an elegant, efficient, but often surprising solution, unlike anything we had seen in our computer science experience. That gave us confidence that we were on a very significant research path.

For such a brand-new and different kind of algorithm the correctness and performance properties of asynchronous distributed rollback really should be formally proved. I had substantial background in program verification, so I went through a private exercise of *trying* to prove both weak and strong correctness. I believe I could have done it but it would have required developing of great deal of new formal machinery which would have been a distraction from my main goals at the time. Still, the exercise of trying convinced me privately that there were no flaws for a simulation with a finite number of LPs. I do not think that even today anyone has published a formal proof of correctness of the full Time Warp algorithm (e.g., including the cancelback protocol), but Bagrodia et al. (1991) may have come closest. Of course many analytical and empirical papers have been published on the performance properties of Time Warp.

In those first few weeks, Henry and I considered many variations on Time Warp. The state restoration parts of the rollback mechanisms we considered were all based on saving snapshots of the state of an LP, but we considered several possibilities, including incremental and variable frequency state-saving. We also considered various message cancellation schemes and distinguished two of them, *lazy* and *aggressive cancellation*. We defined *GVT* (global virtual time), and articulated how it resolved commitment issues such as I/O, error handling, and termination detection, and at the same time allowed us to recycle memory using a technique we dubbed *fossil collection*, intentionally echoing the term *garbage collection*. We considered how an LP going down an incorrect execution path might commit a runtime error or get into an infinite loop, and yet even those problems could be cleanly handled by properly implemented rollback. We dealt with the semantics of handling ties in virtual time, i.e., two events at the same LP at the same simulation time, and invented the superposition concept in which the entire set of tying event messages are processed together in a single event execution. We spent an inordinate amount of time worrying about the problem of repeatability with the use of pseudorandom number generators (PRNGs) in the context of asynchronous rollback. For a long time, I had something of a blind spot and failed to realize the obvious, that when you just treat the PRNG seed and state as part of the *model state* that could be rolled back, rather than as part of the *simulator state*, then there is no problem at all.

For the most part, it is not possible to separate Henry's and my contributions to the early ideas in Time Warp. However, I specifically remember that Henry came up with the name "Time Warp". At first I did not like it at all because it sounded like science fiction to me and I thought such a significant algorithm deserved a more dignified and serious name. Eventually I caved, however, because other people liked it and because I had nothing better to offer. It turned out to be a wonderful choice because once people heard the details of the algorithm the catchy name really helped them remember it. I contributed the term "antimessages" myself. People instantly got the allusion to particles and antiparticles in physics, and that also helped sell the unorthodox idea to the research community.

Henry and I did the first proof-of-concept implementation of Time Warp on a network of four Xerox Dolphin workstations used at RAND. We wrote in InterLisp because we could get something working interactively very quickly. Henry was at the keyboard with me looking over his shoulder kibitzing at every line. After we got the Time Warp code barely turning over we needed a benchmark simulation model to demonstrate that it worked. We chose to write a parallel event-driven version of the cellular automaton known as the Game of Life because it was easy, it could scale it to any size, it had plenty of parallelism available, and it could be trivially load balanced. By giving each LP responsibility for larger or smaller "chunks" of the two-dimensional region we could control the event granularity and the communication-to-computation ratio. That model also forced us to deal with event message ties because each cell in the Game of Life is updated only when it gets simultaneous (in simulation time) inputs from its eight neighbors. And finally, The Game of Life was simple and deterministic and easy validate, so we would know instantly if there was a problem.

The first few times we ran it we got no speedup at all compared to sequential execution, and in fact we measured a severe slowdown. To fix this we had to do our measurements after hours when we were not competing for cycles and network bandwidth with other users of the Dolphins. We had to turn off background computations and system services, such as various daemons, demand paging and lisp garbage collection. We also had to turn off our own debugging and trace instrumentation, which otherwise did file I/O for each event. Only then, when all those heavy performance drags and sources of performance noise were eliminated were we able to see the parallelism and measure any speedup. As I recall, we achieved almost a 2.5x speedup on our four-node network under optimal conditions, though no records of those initial runs survive.

Eventually the collaboration between Henry and myself ended as we began moving in different directions. Henry had not yet finished his PhD dissertation and had to get back to it, and thus wanted to hold off for some months before he could resume work on Time Warp, whereas I had a tenure clock running and students to engage, and wanted to take the research to the next level immediately. Also, inexplicably, RAND did not support our desire to apply for additional research funding to allow us to bring more people to the project. Fortunately for me, as a USC professor, I was not subject to RAND management, and could get my own funding. Before the collaboration ended Henry and I wrote the first paper on Time Warp in the form of a RAND Tech Report (Jefferson and Sowizral 1982). The title, "Fast Concurrent Simulation Using the Time Warp Method, Part I: Local Control", hints at a forthcoming Part II, describing global control (GVT, etc.), but we never got around to writing it.

7.5 Conservative Versus Optimistic Synchronization

After Henry and I had developed the core Time Warp methods and while we were still working together I finally began to study the published literature on PDES, mostly that by Mani Chandy and Jay Misra. They approached the synchronization of discrete event simulation by what I viewed as more conventional means, using algorithms based on process block-and-resume primitives rather than rollback. They introduced the now classic Null Message algorithm (Chandy and Misra 1979), a similar version of which had been independently invented earlier by Bryant (1977). Later they introduced another another algorithm based on a repeated cycle of running to global deadlock and breaking the deadlock (Chandy and Misra 1981).

With the Null Message algorithm (also known as CMB after the inventors) the authors were targeting simulations of queueing systems, dataflow architectures, computer networks, and other systems that are characterized by a static set of components and a static graph of interactions among them. When I finally studied their algorithm my reaction at the time was not favorable. The algorithm required not just a static graph topology, but one using strictly FIFO communication channels among the LPs (called “lines” in their paper), with the event messages sent down each channel required to be in increasing timestamp order. It allowed neither dynamic creation of new LPs nor of new channels between LPs. These seemed to me to be strong restrictions on the class of discrete models that the algorithm could be applied to. Henry and I had military combat simulations in mind, and it never occurred to us to assume a static graph of interactions. We needed a method that could simulate models that were much more dynamically malleable, permitting any LP to send an unexpected event message to any other. One could do that within the CMB paradigm only by assuming a complete graph of channels among the set of n LPs, and that would entail $O(n^2)$ channels, and in some cases $O(n^2)$ messages (mostly null) per unit of simulation time, which was clearly unscalable. We also wanted to allow dynamic creation and destruction of LPs at runtime, a capability that was incompatible with a static graph of LPs.

The CMB restriction that in each channel event messages must be transmitted in increasing timestamp order had other, less obvious consequences. An LP A with four successive events at times 10, 20, 30, and 40 might want to send one message per event to another LP B with timestamps 90, 80, 70, and 60 respectively, so that the event messages must be processed at the receiver in the reverse of the order in which they were sent. Chandy and Misra’s version was based on Hoare’s Communicating Sequential Processes and could only handle an inverted sequence of n messages from A to B if B were divided into n separate LPs and there were at n separate channels from A to each of them, with each message sent along a different channel. Since the number of channels had to be static, there was always a static limit on how long a sequence of inverted order messages could be sent from A to B. As a practical matter it is rare for an LP to want to send a long inverted sequence of event messages, but there is no theoretical reason to adopt any static restriction, and no such restriction is needed by either by the sequential discrete event algorithm or by Time Warp.

Finally, I was also concerned that the Null Message algorithm required additional logic in the model code to decide when to send null messages and what timestamp to send on them. Null messages were required to assure good performance and to avoid deadlocks. This extra logic sometimes required fairly deep understanding of the mechanics of parallel simulation on the part of the model programmer, understanding that may not have been within his or her expertise. In some cases even though deadlock was avoided, it was sometimes only barely avoided, possibly requiring a large number of null messages to make a small amount of simulation progress. In current terminology, we would say that such models have very poor lookahead, and it is inherently difficult for them to achieve good performance with any kind of conservative synchronization, including CMB. Such additional logic is not required in either the sequential algorithm or in Time Warp.

These strong limitations led to my low opinion of the Null Message algorithm *at the time*. It seemed to me at best a special-purpose simulation algorithm, not a general discrete event simulator. In fairness, it was always presented as a network simulation algorithm, but many, if not most, readers thought that it applied much more broadly. For all its good qualities, I was then acutely aware at the time of the Null Message algorithm's limitations.

I hasten to add here that despite those limitations the Null Message algorithm has stood the test of time. There are many important "network" models that fit perfectly within its restrictions, and for them, it is often ideal. Even nonnetwork models can often be profitably shoehorned into network form so that it applies. In the intervening years I have implemented it, applied it in real applications, and taught it several times myself, and I have grown to appreciate it much more.

But in the early 1980s, with the Null Message algorithm's limitations paramount in my mind, I developed something of a competitive attitude. I foresaw great difficulty in getting an exotic rollback-based algorithm like Time Warp to be taken seriously when there seemed to be much simpler, more approachable and more understandable algorithms based on conventional synchronization.

Around this time I took a tour of universities, lecturing about Time Warp, and I visited U. T. Austin where I met Chandy and Misra for the first time. The meeting was a little awkward as I recall. In giving my usual lecture on Time Warp I would have compared it to their published algorithms, and probably argued to the audience that Time Warp was a superior approach. That may not have been the most politic thing to do at their own university. For the first time upon meeting them I realized that they were both quite distinguished and senior to me and thus deserved some deference. I think they may have been mildly irritated at my presentation, though they were totally professional about it. As I recall, it was in that first conversation with them that the terms "optimistic" and "conservative" were adopted to apply to speculative, rollback-based methods versus unspeculative, process-blocking methods. I had been using the term "optimistic" to describe Time Warp-like methods, by analogy with optimistic transaction synchronization, and the natural contrasting term I had been using was “pessimistic” to describe nonspeculative methods. Chandy and Misra, understandably, did not like their algorithms characterized as “pessimistic”, and proposed, if I recall correctly, that we perhaps refer to the two approaches as “liberal” and “conservative”. I was not happy with the overtones those two terms evoked. However that conversation went, and it is very fuzzy to me now, by the end we agreed that each of us would choose our own term to describe the methods we had developed. I chose “optimistic”, and they chose “conservative”. That is how that asymmetric pair of terms came to be adopted.

For maybe two decades or longer a rivalry of sorts continued between proponents of optimistic and conservative methods. Each camp had its advocates, who pointed out the strengths of their methods and the weaknesses of the others. These positions were often inherited by the next generation or two of graduate students as well. I recall one of the more dramatic moments in the competition was an occasion when a prominent professor boomed to a conference audience that “Conservative methods are doomed! Doomed!”. Another moment in the opposite direction was publication by Nicol and Liu of a significant paper entitled “The Dark Side of Risk (What your mother never told you about Time Warp)” (Nicol and Liu 1997). We all laugh about the controversy now, but for a long while the positions were passionately held on both sides. I was partly responsible for this rivalry, but I was certainly not alone. In my defense, I felt that I had a very uphill battle to make the case that a radically different, even bizarre, algorithm such as Time Warp, should even be considered for PDES or any other application. Most researchers initially assumed that conservative PDES synchronization, which was invented earlier and was easier to understand and implement, was the natural and reasonable approach. I thus thought it was necessary not only to demonstrate the advantages of optimistic methods, but also to articulate what I thought of as the inherent limitations of conservative methods.

The rivalry slowly faded. Thankfully it never became personal. Echoes of it still crop up indirectly on panels and over drinks at conferences, especially among old timers of my generation. But today there is widespread recognition and understanding of the merits and limitations of both conservative and optimistic synchronization methods. My own position has now matured as well. I now believe that when and where there is good lookahead information that is easily computed, conservative methods will generally dominate optimistic methods. When there is not, optimistic methods will dominate. In complex models where some parts have good lookahead and others do not, a hybrid synchronization system is called for.

7.6 The Virtual Time Paper

In 1983, I wrote a new paper, “Virtual Time” (Jefferson 1985), based on Time Warp. My collaboration with Henry had ended, and for a lot of reasons it was not going to be possible for us to write a jointly authored paper. Thus, he was not a coauthor, though arguably he should have been. Henry, as a RAND employee, was bound by their slow and ponderous rules for submitting anything for publication, whereas with a tenure clock running I could not afford the year-long submission delays he would be subject to. Instead, to justify my sole authorship, I included as much new material of my own as possible beyond what we had already published. The new paper framed the Time Warp algorithm as a *general purpose* synchronization protocol, useful for an array of applications beyond just simulation, such as database concurrency control (Jefferson and Witkowski 1984; Jefferson and Motro 1986). I still believe that it has potentially widespread value in parallel applications with complex synchronization, fault recovery, and load balancing requirements. The paper also presented an extended analogy between rollback-based synchronization and demand paging implementations of virtual memory, in which a rollback (“time fault”) is considered analogous to a page fault (“space fault”). This analogy is what inspired the title, “Virtual Time”. That paper became very well known, winning the Most Original Paper award at an international conference in 1983, and becoming over the years one of the most frequently cited papers in computer science, and the catalyst for much further research on optimistic synchronization.

7.7 Research with My Students at the University of Southern California

After my collaboration with Henry ended, there was still a huge amount of research to do to flesh out the mechanisms and provide useful theoretical underpinnings for Time Warp. Fortunately, I had funding and graduate students at the University of Southern California that enabled us to make further progress.

Critical path lower bound on time performance: One question we addressed was just how much parallelism is available in a parallel discrete event simulation, and how much of that parallelism Time Warp can capture. With my student Orna Berry, now a distinguished scientist and entrepreneur in Israel, we used a *critical path* approach to define the amount of parallelism available in a simulation model or, equivalently, to define a lower bound on the time it takes to execute a simulation in parallel. We were able to prove that no conservative algorithm could execute in less time than the length of its critical path. Time Warp was also bound by the same critical path length if it used *aggressive* cancellation but, in an extremely unexpected result, Berry proved in her dissertation that Time Warp with *lazy* cancellation could sometimes actually execute faster than the critical path lower bound (Berry and Jefferson 1985; Jefferson and Reiher 1991). Even now, after 30 years, this result is not widely appreciated, and its consequences are still relatively unexplored.

Flow control: Another issue we had not considered in the original Time Warp research was message flow control. (I was not even aware of flow control as a generic distributed systems issue until I started teaching operating system courses at

USC.) In the original work at RAND there was no mechanism to prevent fast-sending producer LPs from filling up the memory of slow-processing consumer LPs with queued-up event messages, fatally choking the simulation. Classical "windowing" flow control algorithms do not apply in Time Warp (or any other general PDES simulator) because there are no message "channels" and message streams from multiple senders are merged at the receiver, because unexpected messages from *new* senders can arrive any time without warning, and because the order in which messages must be processed at the receiver (timestamp order) is not generally the same as the order in which they are sent, nor the order in which they arrive. Eventually, Darren West at Jade Simulations in Calgary, and I, working partially independently, came up with a very elegant solution, one based on the idea that a receiving LP whose incoming message queue was excessively long could send high-timestamped messages back to their sending LPs to make room for incoming messages with lower timestamps, in a protocol we called *cancelback*. When the sending LP receives a sentback message, it rolls back to before it sent the message, and executes forward again and resends the message later (Jefferson 1990).

The idea of sending a message backward from receiver to sender is another highly unorthodox feature of Time Warp that cannot work in most parallel computation paradigms because they lack the ability to roll back. Sending messages backward is a communication idea that is nicely symmetric to the computational idea of rollback, and meshes perfectly with it. That observation and others led to greater emphasis on elegance and symmetry in future presentations of the Time Warp algorithm.

Unfortunately, even though Time Warp has been implemented many times in the last 30 years, most implementations leave out the cancelback protocol. This leaves them open to unpredictable and unrepeatable runtime failures due to memory exhaustion in an LP. That kind of nondeterministic failure behavior is essentially a Heisenbug, and when it occurs people can waste a huge amount of time trying to figure out what the problem is or work around it. Fortunately, experience so far shows this does not happen very often, probably because the models we are interested in tend to be approximately well balanced, or because some kind of active throttling of optimism is used. But I expect that as the complexity of models increases, especially with federated and multiscale models, this memory management hazard will also increase in urgency. We should consider the cancelback protocol as a fundamental part of any Time Warp implementation.

Global memory management and memory bounds: The consideration of flow control led to a larger concern of more global memory management, and the question of the minimal memory requirements for a Time Warp simulation. It was obvious that to get good performance a Time Warp simulation would normally require at least several times more memory than an equivalent sequential simulation because it had to store both a sequence of snapshots of each LP's state *and* previously processed event messages *and* anti-messages for every message the LP sent, and it had to retain them at least as far back as GVT. There was a danger that the memory requirements would grow without bound, and the simulation would be unstable for that reason. We needed a theory for how to manage Time Warp memory, and as part of that we wanted to know the *minimal* amount of memory required for a Time Warp simulation to complete (though more memory was always better).

My student, Anat Gafni, proved in her dissertation that Time Warp, using both the cancelback and fossil collection protocols together, could be guaranteed to complete a simulation if given no more than about twice the memory that a sequential execution of the same simulation would require (Gafni 1985). We later improved that result by a factor of two, so that in theory Time Warp could complete in about the *same* amount of memory as a sequential execution when running on a shared memory (or virtual shared memory) platform (Jefferson 1990). Of course, the runtime performance would be terrible when Time Warp is constrained to run with memory near the minimum, but the result made the point that Time Warp could be space optimal. Surprisingly, I was also able to prove that asynchronous conservative methods were far from space optimal in general. I constructed artificial models that, with unfortunate timing, could require many times the memory of the sequential execution, although that was not typical.

Symmetry: The success of the cancelback protocol, in which message sendback was a direct analog to computational rollback, and the success of the analogy between virtual time and virtual memory, led me to adopt *symmetry* as an explicit goal in the Time Warp algorithm. Symmetry helped me present the algorithm more compactly and convincingly, and it led me to search for other places in the algorithm that had near symmetries and to correct them to make them perfect. Over the years I came to recognize a lot of symmetries, including message–anti-message symmetry, state–message symmetry, forward–backward time symmetry, forward–backward message transmission symmetry (cancelback), virtual memory–virtual time symmetry, and others. In the end, the bedrock reason I personally remained inspired by optimistic simulation was fundamentally aesthetic. I did not see similar symmetries in conservative methods, so I did not find them so compelling.

These results from the 1980s gave us confidence that optimistic methods, unorthodox as they were, should be taken seriously, both practically and theoretically, and that conservative methods had some genuine limitations. But the real tests would have to come with a serious parallel implementation of Time Warp and performance studies using both benchmark models and realistic models. We needed to demonstrate that Time Warp, with its heavy overheads and its poorly understood dynamical behavior, could nonetheless achieve real speedup from parallelism on real applications, and could be competitive with conservative algorithms in at least some useful application areas.

7.8 The Time Warp Operating System at Jet Propulsion Laboratory

On a visit to Caltech I had the opportunity to give a series of two talks. Since it is hard to get people to come to a second talk after a week's interregnum I tried a dramatic trick to attract them back. I described the PDES synchronization problem in the first lecture and convinced the audience that the problem, as I framed it, was essentially impossible to solve. I left a cliffhanger, promising to resolve it a week later in the second talk. When that day came, I had a full room and presented Time Warp, cleanly solving all the apparent problems. It was the introduction of rollback, which I had not mentioned in the first talk, that made the problem as I had framed it soluble. Those two lectures turned out to be especially important subsequently because the audience was full of physics-oriented people from the Jet Propulsion Laboratory (JPL) and Caltech who appreciated the analogies between Time Warp and physics (symmetries, anti-messages, etc.). Soon afterward when we started a multiyear Time Warp development effort at the Jet Propulsion Laboratory, some of the core members of that team, especially Brian Beckman, had been present in that audience, and others who were there became supportive of the project.

By 1984 I had moved to UCLA and begun the relationship with JPL that lasted for 7 years. A coincidence of three circumstances made this possible. First the Army, a sponsor at JPL, was interested in speeding up combat models and looked to parallel discrete event simulation as a key enabler. Second, researchers at Caltech and JPL were designing and building a parallel computer with a new architecture, the Caltech hypercube, and were looking for projects that would make constructive use of it and demonstrate its value. And finally, because my collaboration at RAND had ended, I was available and eager to work on the project and was invited to lead it.

The Caltech Hypercube was a 32-node distributed-memory cluster with Intel 80286/87 processors connected by 128 KB/s communication channels in a 5-D hypercube topology. Each node had 256 KB of RAM. It was an ideal machine on which to build the first serious implementation of Time Warp. At that time 32 nodes constituted a very large parallel computer. It was much larger, and more tightly coupled, than the four-node network Henry and I had used—large enough that we would be able to demonstrate significant parallelism and do useful scaling studies. The fact that the Hypercube did not have shared memory was in my view an advantage. I did not want to be tempted to use shared memory as a performance crutch in any way, since any such dependence would leave doubt about the scalability of Time Warp on platforms larger than the practical limit of shared memory.

The Caltech Hypercube had no operating system per se that we could use. This was before the first release of Linux, and long before it became almost a de facto standard OS for cluster machines. The only system software it had was what today we would call a two-sided, synchronous, order-preserving message system, like a primitive MPI that was intended to support what we would today call the Single Program Multiple Data (SPMD) parallel programming model. But we could not use that because Time Warp needed a one-sided, asynchronous, interrupting message system, and we did not need order preservation. Thus, we had to build our own messaging layer, and that layer rested essentially on the "bare metal" of the hypercube. Even this I did not view as a handicap because it fit with my view that Time Warp should be thought of not just as a *simulator*, but as a special-purpose *operating system*. The project was therefore named *TWOS*—the Time Warp Operating System (Jefferson et al. 1985, 1987; Wieland et al. 1989).

I think of Time Warp more as an operating system than as an application because a full implementation requires the same software components as an operating system for a parallel machine, but with alternative, virtual time- and rollback-friendly algorithms in place of the classical ones.

- Time Warp needs a process scheduler with a lowest virtual-time-first discipline rather than any variation on round-robin.
- Its primary synchronization is based on asynchronous interrupting messages and general distributed rollback, not on locks, semaphores or process blocking.
- It needs timestamp-order priority queues with anti-message annihilation, rather than FIFO message queues.
- Its message flow control and global storage management need to be based on fossil collection and cancelback rather than garbage collection and windowing.
- It needs daemons for distribution of GVT (global virtual time), using what today would be called asynchronous all-to-all reduction.
- It needs special normal and abnormal termination detection and error handling, based on GVT.
- It needs special I/O commitment, also keyed to GVT. (It would be nice to also have a file system that supports rollback, but no one has ever built one.)
- It needs custom instrumentation to measure quantities such as events executed, events rolled back, message annihilations, message cancelbacks, and various other performance metrics unique to Time Warp that have no analog in conventional operating systems.

Later in its development, Time Warp also needed special mechanisms for rollback-friendly LP creation and destruction, dynamic LP migration to support load balancing (Reiher and Jefferson 1990), and advanced dynamic message routing to deliver messages to migrating target processes (Ravi and Jefferson 1988).

Despite these considerations, today everyone implements Time Warp as a run-time system *on top of an OS*, rather than as an OS itself. But however practical that decision is, we should be aware that there are performance costs to it. It entails two levels of scheduling, two levels each of synchronization, message queuing, memory management, error handling, and termination detection, as well as reliance on a level of polling for incoming messages.

The TWOS project at JPL was fairly large, with a full-time development staff of anywhere from 8 to 12 at any one time over the 7 years. Two people wrote the core Time Warp algorithms, initially Brian Beckman and later Peter Reiher, both of whom did brilliant work. The project would never have succeeded without them.

Phil Hontalas had the highly technical task of writing the low-level asynchronous messaging system complete with routing and interrupt handling, etc., and then porting it twice to subsequent parallel machines. Mike DiLoreto was a master debugger and performance specialist jack-of-all trades. John Wedell built a high performance sequential simulator that was semantically identical to the TWOS simulator and used for performance comparisons. He kept improving its performance, which challenged the team working on TWOS since its performance was evaluated relative to the performance of Wedell's sequential engine. Fred Wieland built the main parallel benchmark simulations and wargame models for the Army. Steve Bellenot did R&D on GVT algorithms. Van Warren was our graphics specialist. Others, including Leo Blume, Joe Ruffles, Kathy Sturdevant, Larry Hawley, Abe Feinberg, Pierre Laroche, John Spagnuolo, Todd Litwin and two fine interns, Maria Ebling and Matthew Presley, contributed benchmarks, instrumentation, documentation, ran performance studies, etc. Jack Tupman and Herb Younger were our managers interfacing with the Army sponsors, JPL upper management and the proprietors of the Hypercube, and husbanding the finances.

After the first few years, JPL built a new, much more powerful hypercube: the JPL Mark III. It had 64 nodes arranged in a 6-D hypercube, and was based on the Motorola 68020/68881 processor/coprocessor pair with a comparatively whopping 4 MB of RAM per node! Still later our project purchased a BBN Butterfly GP1000, a shared memory machine with 112 nodes, also with 68020/68881 processors and 4 MB per node. We ported TWOS to each of these machines in succession, which allowed us to run still larger models, achieve higher degrees of parallelism, and demonstrate the portability and scalability of Time Warp.

The following graphs, which are re-scans of the original plastic transparencies I used in 1988 in various presentations, illustrate the performance of TWOS about midway through the 7-year project. Figure 7.1 shows a strong scaling study of a military ground combat model called STB88 that the Army commissioned us to build as a benchmark. It had 380 LPs and was run for 326,997 events. The graph shows speedup of the combat model as a function of the number of nodes used on the Mark III Hypercube. The speedup is relative to the performance of our fast sequential simulator, not to Time Warp executing on one node. That is important because if we had chosen the performance of our algorithm on one node as a basis for comparison, which parallel computation researchers often did in those days, it would have yielded much higher but quite artificial speedup values, since a one-node Time Warp execution would pay high and unnecessary overheads for support of rollback and would consequently run much slower than our sequential simulator. The curve in Fig. 7.1 shows approximately linear scaling, from a speedup of about 2.5 on 4 nodes to about 16.5 on 32 nodes. The data in the documentation box shows that the sequential simulation took 1.4 h, while the time on 32 nodes under Time Warp was 5.3 min. The initials of Fred P. Wieland in the last line of the box show that he performed the runs. The sponsor was very happy, and so were we—we put this graph on a T-shirt!

The next two figures show a performance study a few months later of TWOS running essentially the same combat model as in Fig. 7.1, but this time done on the

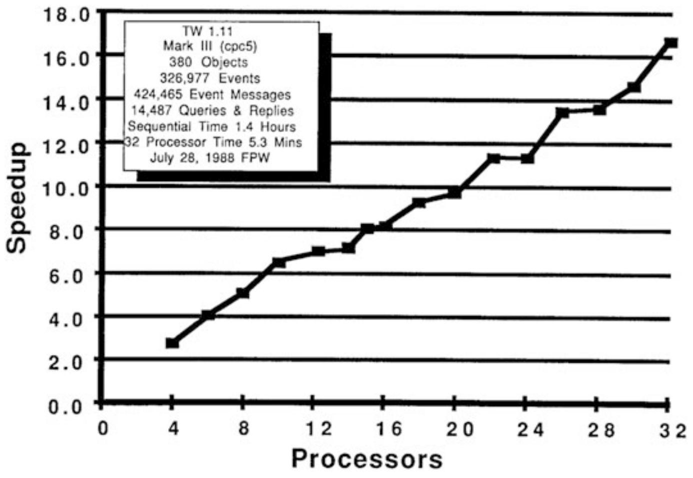

Fig. 7.1 Strong scaling study of TWOS relative to sequential execution of an Army combat model (STB88) running on the JPL Mark III Hypercube in July, 1988

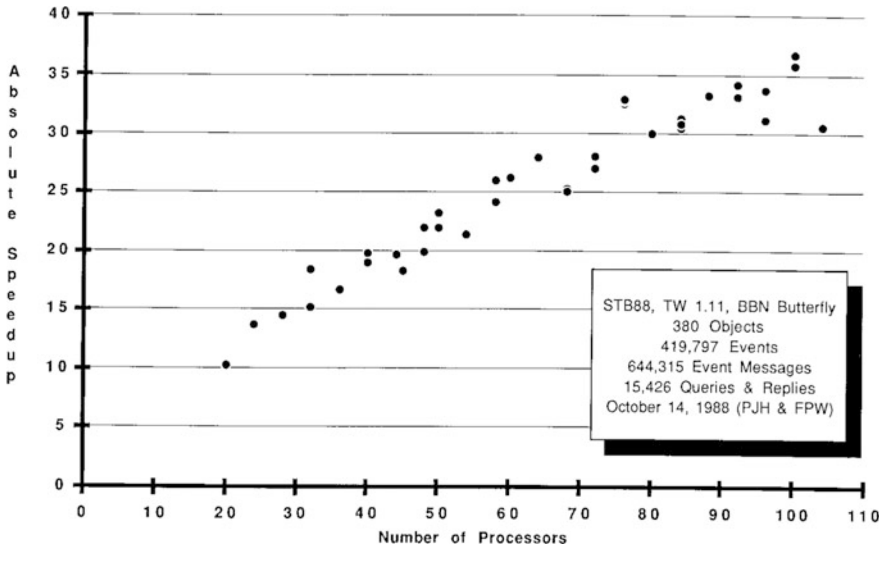

Fig. 7.2 Strong scaling study of TWOS relative to sequential execution of an Army combat model STB88 running on the BBN Butterfly in October 1988

BBN Butterfly by Fred Wieland and Phil Hontalas. Figure 7.2 is another plot of speedup as a function of the number of nodes applied. In this case, we reached a speedup factor of over 35x using 100 nodes and with an approximately linear speedup almost to full scale. We did some runs two or three times, and you can see up to 10% variation in performance even with the exact same configuration

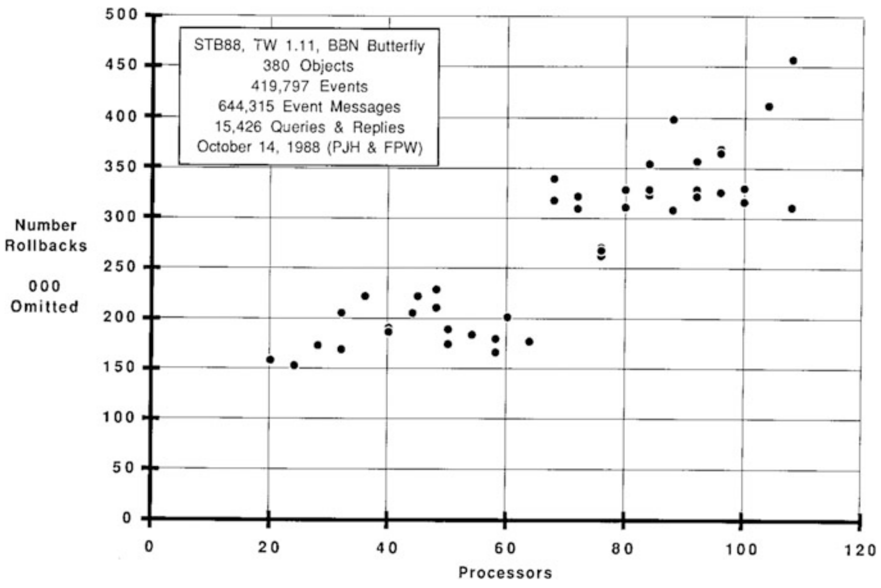

Fig. 7.3 Rollbacks as a function of number of nodes, using the same runs as in Fig. 7.2

measured twice. The variation shown between runs on different numbers of nodes is larger, and is primarily because the combat model was irregular in structure and the load was not always as well balanced at one scale as it was at another.

Figure 7.3 shows a plot of the number of rollbacks measured in the exact same runs plotted in Fig. 7.2. (It may not appear that way because in both graphs there are cases where multiple measurements at the same scale were so close that points coincide.) Figure 7.3 illustrates the fact that in Time Warp the number of rollbacks generally increases as a model is spread over more and more processors. In fact when running over 100 nodes, there were over 400,000 rollbacks when the total number of committed events was 419,797. Although a huge percentage of the events were rolled back and executed more than once, the speedup as shown in Fig. 7.2 continues to increase, at least up to the scale reached in this study.

This result is characteristic of Time Warp and very surprising to people new to it. How can it be that the speedup continues to increase even though the number of rollbacks does also? It is because the great majority of those rollbacks are for events that are way off the critical path of the computation, and thus they do not slow down the global progress of the simulation. While there is such a thing as excessive optimism and too many rollbacks, this graph makes it clear that in tuning a Time Warp simulation one must not just naïvely make it a goal to reduce the number of rollbacks.

7.9 Jade Simulations

During roughly the same years as the JPL project, Brian Unger at the University of Calgary started a company, Jade Simulations, that had the goal of commercializing Time Warp. I was a Board member and advisor. Jade built a completely new distributed implementation of Time Warp that ran well on both networks and clusters. Unfortunately Jade never successfully found a market, partly I think because the relentless acceleration of sequential computation due to Moore's Law allowed people who needed higher performance simulations to just wait a couple of years, and partly because Jade was a Canadian company that was not permitted to compete for U.S. military simulation business. But I also think that Jade was just ahead of its time, when parallel computers were still very expensive. It would take several more years before any market matured sufficiently to make commercialization viable.

7.10 Subsequent Years

The most significant new PDES idea in many years came in 1999 when Richard Fujimoto and his students, Kalyan Perumalla and Chris Carothers, introduced the idea of reversible computation (Carothers et al. 1999). This was a dramatically different and more efficient approach to implementing rollback, and leads some tantalizing programming theory and programming language ideas as well. At the time I had left the field of PDES, and only learned about it several years later when I reconnected. But then I was deeply impressed. Eventually, when I found myself in position to work on it again at Lawrence Livermore National Laboratory, Markus Schordan, at my instigation, built a compiler that can take literally any program written in C++ and create reverse code for it (Schordan et al. 2015; Schordan 2016). We hope this development may make reversible computation the standard way of accomplishing rollback and remove most of the last remaining software engineering barriers to the widespread adoption of optimistic synchronization.

Time Warp has been implemented many times in the years since the RAND, JPL, and Jade projects. Richard Fujimoto, as he describes in the next section of this chapter, created Georgia Tech Time Warp (GTW), originally as a platform through which to study major improvements that could be made to Time Warp in a shared memory environment. He was the first person to do fair comparison studies between conservative and optimistic methods on the same problem and to verify that in many cases (but not all) the optimistic methods could indeed outperform conservative methods. Later his students developed their own implementations of Time Warp, derived to some extent from their experience with GTW. Kalyan Perumalla, now at Oak Ridge National Laboratory, developed the µsik simulator, which he still uses for advanced studies of PDES (Perumalla 2005). And Chris

Carothers, now at RPI, developed the ROSS simulator, which is open source and available to all researchers who wish to work with it (Carothers et al. 2002).

In at least one respect ROSS is now the state of the art in Time Warp implementations. It holds the world record to date (2017) for the largest scale and fastest discrete event simulations ever executed. In 2013 my colleagues Peter Barnes, Chris Carothers and Justin LaPre and I ran a series of benchmark runs on the *Sequoia* supercomputer at Lawrence Livermore National Laboratory, which at that time was the second fastest computer in the world. On the standard PHold benchmark with 251 million LPs we achieved a sustained 5.04 billion events per second using almost 2 million cores and almost 8 million hardware threads (Barnes et al. 2013). Those runs proved that the Time Warp algorithm can scale to gigantic degrees of parallelism.

In the 35 years since Henry Sowizral and I built the first primitive four-node implementation, Time Warp has achieved over a million-fold increase in demonstrated parallelism. While I anticipate a temporary pause in this peak performance increase as the architectures of supercomputers are changing in the current era in ways that do not benefit Time Warp or PDES, the future is nonetheless long and it is very hard to imagine what the next 35 years might bring.

7.11 My Adventures in Time Warp: Perspectives by Richard Fujimoto

The second section of this chapter provides the perspectives of Richard Fujimoto.

7.12 Beginnings

I first became interested in parallel discrete event simulation (PDES) when I was a doctoral student at the University of California in Berkeley (1978–1983). As an undergraduate student at the University of Illinois I became interested in computer architecture, and embarked on a doctoral dissertation at Berkeley looking at high speed switches to interconnect microprocessors, which at that time were just becoming powerful enough to be interesting, to create parallel computers. In order to evaluate our ideas about switches we needed parallel applications to generate realistic message traffic. It immediately became clear to me then that the discrete event simulator I had developed to evaluate our switches would be a perfect benchmark program. This of course led to consideration of the synchronization problem, and ideas akin to those developed by Chandy, Misra, and Bryant (published a few years earlier, but unknown to me) came to mind, but my main interest was in hardware design, so I did not embark on any serious studies of the problem while I was a graduate student.

It was not until years later that I renewed my interest in this area. I then came across Chandy and Misra's and other's papers concerning conservative synchronization, and Jefferson's work on Time Warp. At the time, the initial algorithms were in the published literature, but no one knew which approach was better, and under what circumstances one approach might dominate the other. My attention focused on doing a serious comparison of these algorithms. This work launched a career-long exploration of parallel and distributed simulation techniques.

7.13 Conservative Versus Optimistic Performance

Foremost in my thinking was the need to have a very efficient sequential simulation to determine the speedup obtained using parallel computing. This led to an exploration of the literature in event list implementations for sequential simulations. A paper by Doug Jones comparing different priority queues for discrete event simulations caught my attention (Jones 1986). This comparison, and others that had appeared at the time, were all based on something called the HOLD model, which seemed to be the standard benchmark for evaluating sequential event list performance. I developed a parallel version of the HOLD model, first described in (Fujimoto 1988), which later become known as PHOLD (Parallel HOLD). I used PHOLD in my initial comparisons of the Chandy/Misra/Bryant null message and deadlock detection and recover algorithms, and later Time Warp (Fujimoto 1990). PHOLD continues to be used to this day to evaluate parallel simulation performance.

There were a couple of central conclusions that came from these comparisons that drove much of my work that followed. First, these results showed conclusively that both Time Warp and the conservative algorithms could achieve excellent speedup relative to efficient sequential implementations using state-of-the-art event list data structures. Prior to that time much of the work had reported speedups of conservative methods, but often compared the parallel implementation against a sequential execution of the parallel algorithm, or did not use a particularly efficient sequential implementation. For example, I recall some results reported super-linear speedup. But on closer examination, this work compared performance against a sequential simulator using a linear list implementation of the event list, an approach that becomes very inefficient for large numbers of pending events, leading to inflated parallel performance. There was a dearth of empirical results reporting Time Warp performance at the time, though measurements of the Time Warp Operating System (TWOS) for realistic applications would soon appear.

A second conclusion from this (and other) work was that conservative algorithms relied on exploiting knowledge of the simulation application in order to extract good "lookahead" information, essential to obtaining efficient parallel execution. Briefly, lookahead is a guarantee made by the simulator that any new events it schedules are at least a certain amount of simulation time into the future. The greater the lookahead, the better. Intimate knowledge of the application is required in order to make such a guarantee. This suggested significant drawbacks with conservative synchronization algorithms. It meant that the application itself had to possess good lookahead properties; not all applications possessed such properties. For example, if two entities in the simulation could interact in a small amount of time, e.g., two radios in a simulation of a wireless network could instantaneously communicate, large lookahead may be difficult or impossible to obtain. Further, even if one could build the simulation to have good lookahead, if the simulation model had to be later modified, such modification might destroy these lookahead properties, defeating the parallelization approach. For example, in a queueing network simulation, if one added high priority jobs that preempted service from lower priority jobs, it greatly reduces the lookahead, and could lead to a dramatic reduction in performance, even if there were only a few high priority jobs in the system. As a result, conservative simulation applications were prone to becoming brittle in that if details of the simulation model changed, the original parallelization approach might not yield acceptable performance, or it might not even run at all.

It was quite apparent to me early on that two facts seemed inescapable. The first was that Time Warp stood the best chance of realizing a general purpose parallel discrete event simulation engine over which a variety of applications could be developed. This, of course, seemed to be the most viable path for the technology to see real-world use and have widespread impact because it would enable exploitation of the technology in a wide variety of application domains without intimate knowledge of parallel processing or the synchronization mechanism. In much the same way the developers of sequential simulation models did not need to be concerned with the priority queue data structure that was used. Expecting domain experts, who would be the ones who developed the simulation models, to be experts in parallel computation and PDES seemed a stretch. The reliance of conservative methods on intimate knowledge of the application called for domain experts to have much more expertise in PDES than I thought realistic.

A second observation was that there were significant challenges that needed to be addressed for Time Warp to yield acceptable performance. Two hurdles seemed obvious. The first problem was one that everyone immediately understands as soon as they learn about Time Warp, namely that the computation could spend most of its time rolling back events and recovering from errors rather than completing computations for the simulation model. This problem appeared to be solvable, however, and approaches to addressing this problem began to appear soon after Time Warp had been invented. A second problem that seemed somewhat more difficult was the need for state-saving in order to allow the computation to be rolled back. This could consume much time and memory. Both of these problems were important and interesting, but the state-saving problem was the one that captured my attention first. With my background and interest in computer architecture, I envisioned this was something that could be addressed with hardware.

7.14 The Rollback Chip, Virtual Time Machine, and Reverse Execution

As an undergraduate I had been enamored with clever techniques to manipulate memory addresses in digital circuits to implement cache and virtual memory systems. It seemed natural to use such techniques to implement state-saving in hardware for Time Warp. One only needed to intercept memory writes, and modify the memory address to preserve the original contents of the memory location. Some additional work was needed to implement memory reads, in order to ensure the correct version of memory was accessed. This was straightforward to accomplish in hardware by keeping track of which blocks of memory had been written. These ideas led to a kind of memory management circuit that we called the rollback chip (Fujimoto et al. 1992). Separately, I was approached by two electrical engineers who were keen to develop prototype hardware, and were able to get some funding for the same, resulting in an initial proof-of-concept prototype implementation of the system (Buzzell et al. 1990).

The next logical step beyond the rollback chip work was to extend hardware support beyond state-saving, to other aspects of Time Warp. Here, the conceptual model I found useful was the data dependence, or task graph, where nodes represent event computations, and links represent dependencies between events. Specifically, each event within a logical process (LP) depends on (or more precisely, *could* depend on) the preceding event in the LP with the next smaller time stamp. And if one event schedules a second, then there is obviously a dependence of the scheduled event on the one that scheduled it. Data dependence graphs had been around for some time and were widely used to study parallel computations. One of David Jefferson's students, Orna Berry, used this model to determine properties of the computation such as average parallelism and minimum possible execution time (Berry and Jefferson 1985). Task graphs also greatly influenced my work in developing Time Warp software, described later. I realized that a hardware representation of the task graph, stored in the shared memory of a multiprocessor system, could be used as the basis for a general parallel computer based on Time Warp.

I called the resulting parallel computer the Virtual Time Machine (VTM) (Fujimoto 1989a, b), giving a nod to Jefferson's original Time Warp paper entitled Virtual Time (Jefferson 1985). Admittedly, I also liked the sci-fi-ish nature of the name. A distinction between the VTM design and Time Warp is that it did away with logical processes, and used a shared memory rather than a message-based computation model. At the core of VTM was the space-time memory system, a memory system addressed by a time value in addition to a conventional memory address.

We envisioned the VTM to be a general purpose parallel computer using rollback as its core synchronization primitive that could be used broadly for parallel computation, not just simulations. After all, one could, in principle, take any sequential computation, divide its execution up into blocks of instructions, assign each block a time stamp reflecting its sequential order of execution, and the result would be a discrete event simulation computation where each block represented an event. We envisioned the VTM could be used to automatically parallelize sequential computations, something compiler writers at the time had been struggling with for years.

This vision proved to be a bridge-too-far, however. We focused on the use of the VTM to automatically parallelize sequential discrete event simulations as a first step toward this bigger, broader objective. This was examined by Jya-Jang Tsai for his doctoral research, who had completed some of the early simulation studies of the rollback chip. But even in this friendlier domain, it was difficult to reduce the computation overheads to an acceptable level, and only a modest amount of success was achieved. The computer architecture world at the time was moving toward reduced-instruction-set-computers (RISC) so the trend was toward simpler rather than more complex, and the VTM approach did not fit well into this movement. At a deeper level, sequential programs not written for parallel execution often do not exhibit sufficient parallelism for an approach such as this to succeed; the code needs to be structured for parallel execution earlier on in the software development cycle. So, we did not continue to pursue the VTM hardware architecture, though the VTM ideas influenced much of our later work in distributed simulation.

For example, a software-based distributed simulation approach derived from the Virtual Time Machine and Time Warp was a technique we termed "ad hoc distributed simulations" (Fujimoto et al. 2007). The context for this work was in the use of distributed simulation online to manage operational systems. Here we repurposed the space-time memory used in the VTM to hold a projected future state of the system. Simulations computed forward, ahead of wall clock time to predict future system states, which were stored in the space-time memory. If measurements of the actual system deviated from predictions by more than a specified threshold, a Time Warp style rollback mechanism was used to automatically correct the predictions. Although the approach is applicable to a variety of systems, our work largely focused on evaluating this approach for traffic management applications, and more broadly, systems modeled as queueing networks (Huang et al. 2012).

Getting back to Time Warp, since the VTM hardware approach was never able to get much traction, we focused more on software-based ideas. In particular, the next chapter in our work in addressing Time Warp's state-saving problem was to move away with state-saving entirely, and instead employ a reverse execution method to undo incorrect computations. Chris Carothers and Kalyan Perumalla pursued much of this work, with Kalyan pursuing it for his doctoral research (Carothers et al. 1999). Both continued to work on reverse computation after completing their work at Georgia Tech. Actually, reverse computation did not entirely eliminate state-saving because some computations are inherently irreversible; state-saving was used as a fallback approach in situations where the inverse computation could not be created. Nevertheless, reverse computation can significantly reduce the memory footprint of Time Warp programs and greatly reduce the time required for state-saving. Much of our work in reverse computation focused on proving its viability for various simulation applications. In addition to standard benchmarks such as communication networks, some of this effort focused on applying the technique to scientific computing applications such as modeling the earth's magnetosphere in collaboration with Homa Karimabadi and his group at Scibernet, a start-up company in California, and a graduate student named Yarong Tang (Tang et al. 2006).

Developing the code to invert a computation is not straightforward, and prone to error. To address this problem Kalyan Perumalla developed a reverse execution compiler to automatically instrument the forward execution and generate the reverse code for his PhD dissertation. This work was subsequently continued in collaboration with David Jefferson and others at Lawrence Livermore National Laboratory (LLNL) and Georgia Tech to create the backstroke compiler (Vulov et al. 2011; Schordan et al. 2015, 2016).

7.15 Simulation Ensembles

We never reached the point of creating a hardware realization of the Virtual Time Machine. But as mentioned earlier, the computation model it used formed the basis of other work. One line of research focused on accelerating the completion of multiple simulation runs, called ensemble simulations. Virtually all simulation studies require many runs. Often these runs are similar, and have many computations in common. Our focus was on performing these common computations only once, and sharing their results. As will be seen, some ideas from Time Warp can be used to achieve this goal.

We used task graphs not unlike those used in VTM extensively in this work. Specifically, Steve Ferenci developed something we called updateable simulations (Ferenci et al. 2002). The basic idea is to record the task graph for a computation. Then, to complete subsequent runs of simulations similar to the recorded one, use a VTM/Time Warp style rollback mechanism to "update" the simulation execution in accordance with differences in the new run.

A related idea, developed earlier by Maria Hybinette for her doctoral research, was to use an incremental cloning mechanism to compute a set of similar simulation runs (Hybinette and Fujimoto 2001). This work used the traditional PDES logical process model and Time Warp, although her work did not require the use of Time Warp. Motivated by the use of Time Warp to assess alternate possible futures for air traffic control simulations, the idea was to identify points in the future when decisions would need to be made, and then to replicate or clone the parallel simulation to concurrently explore these alternate futures. The central idea in Maria's work was to use an incremental LP replication method where LPs were replicated as needed as the computations of the different runs diverged. This enabled us to share the results that were common among the different replications.

7.16 Georgia Tech Time Warp

Let me “roll back” now to 1988 and come back to our software implementation of Time Warp. I recognized early on that there were several important challenges in realizing an efficient implementation. My early work focused on developing techniques to create a fast Time Warp system using a variety of methods developed by our research group. These techniques were incorporated into a software system we called Georgia Tech Time Warp (GTW).

My initial work focused on developing an efficient implementation of Time Warp on shared memory multiprocessors. Motivated by the task graph model, my first implementation focused on, in effect, storing the task graph in the shared memory of the multiprocessor system. In particular, when one event scheduled another, we simply stored a pointer from the scheduling event to the event being scheduled. This greatly simplified the Time Warp implementation because it eliminated the need for separate anti-messages, with the pointer effectively implementing an anti-message. It also eliminated the need to create a copy of each scheduled message, and avoided the need to search for the matching positive message when an anti-message was received. I called this technique direct cancellation (Fujimoto 1989b). Direct cancellation allowed one to rapidly track down and correct errors resulting from out of order executions. I believed this was important because while one used anti-messages to correct these computations, the wrong computations themselves were spreading, so it was important to track down the errors as quickly as possible to minimize the damage they caused. The resulting shared memory data structure used to implement Time Warp motivated the space-time memory used subsequently in the Virtual Time Machine work described earlier.

We developed several other innovations to create an efficient shared memory Time Warp system, described in (Das et al. 1994; Fujimoto 2000a, b). For example, working with PhD student Maria Hybinette, we developed an efficient shared memory algorithm for computing Global Virtual Time (GVT) (Fujimoto and Hybinette 1997). GVT is needed in Time Warp to determine a lower bound on the time stamp of future rollbacks, enabling reclamation of memory (fossil collection) and committing irrevocable operations such as I/O. We also developed an efficient, “lazy” approach to implement fossil collection that we called on-the-fly fossil collection. It avoided the need to search through lists to identify chunks of memory that needed to be reclaimed.

Sequential discrete event simulations often use a mechanism to “unschedule” previously scheduled events. This is necessary to model activities such as preemption where some “normal” activity is interrupted by some other event, e.g., in a queueing network, servicing a job might be preempted by the arrival of a higher priority job. We realized that anti-messages, which needed to be implemented in Time Warp for synchronization purposes, could be used to implement this unscheduling of events. The difference now is that the generation of anti-messages was triggered by the application program itself, not the underlying simulation engine. A small wrinkle was that these application-invoked event cancellations could themselves be rolled back, but this was easily accomplished by sending a copy of the original event. The simplicity of the mechanism derived from the simplicity and symmetry of the original Time Warp algorithm. Working with my PhD student Samir Das we developed the algorithms and an implementation, and evaluated its performance. Later, we found that Greg Lomow working with Brian Unger at the University of Calgary independently developed a similar mechanism, but had not published the work because they were working on commercializing the effort in the context of a company called Jade Simulations that Brian was developing. We agreed to jointly publish the work (Lomow et al. 1991).

As we developed new innovations we incorporated them into the GTW Time Warp software (Das et al. 1994). Based on my initial implementation that focused on the direct cancellation mechanism, Samir Das did much of the work in furthering the development of the system. Various developments were incorporated into GTW in the years ahead, including extension to message passing and distributed computing architectures. GTW formed the basis for most of our experimental work in Time Warp such as the reverse execution work described earlier.

GTW was used for a variety of applications, but perhaps the most memorable was its use to create fast air traffic control simulations. In the summer of 1989, I had the pleasure of spending a 3-month period working with David Jefferson and his research group at the Jet Propulsion Laboratory. They were working on the Time Warp Operating System (TWOS) project. During this period, I met Fred Wieland who was one of the TWOS developers, focusing on implementing a combat simulation model. After TWOS, Fred went on to work on simulation applications in the aviation industry with the MITRE Corporation. Fred is a likable guy, and we had good discussions on Time Warp, parallel discrete event simulation, and aviation over many years that followed. Anyway, Fred was interested in creating a fast parallel simulator to model the U.S. aviation system, and learned about GTW. He embarked on developing an air traffic simulation called the Detailed Policy Assessment Tool (DPAT) on our Time Warp system (Mitre Corp. 1997). DPAT was deployed for air traffic analyses, making it at the time one of the few real-world deployments of Time Warp for real-world applications (Wieland 2001). The DPAT/GTW system subsequently formed a basis for Maria Hybinette's work in parallel simulation cloning, discussed earlier.

7.17 Analytic Models, Memory, and Load Management

In 1990, I began work with Ian Akyildiz, a colleague at Georgia Tech, on a project funded by the Ballistic Missile Defense Organization (BMDO), later renamed the Missile Defense Agency (MDA). BMDO was working on developing the technologies for the Strategic Defense Initiative more commonly known as the "Star Wars" program that aimed, among other things, to create the ability to intercept incoming ballistic missiles. Simulation was a key part of the program. The program manager, Lou Lome, was particularly interested in developing PDES technology, and funded some of our work. This project was the first in a string of BMDO projects that funded our work on Time Warp. Our program was part of BMDO's basic research program aimed to develop the underlying technologies to create fast parallel and distributed simulations. Our basic research program ran another 10 years; I was later told we were the only group still funded by this program when BMDO decided to end its basic research program.

Ian had background in developing mathematical performance models. This led to a collaboration focusing on developing analytic models of Time Warp. With PhD student Anurag Gupta we developed a model to predict the performance of homogeneous Time Warp systems where all LPs behave similarly, as would be the case in, for example, many queueing network simulations (Gupta et al. 1991). Development of a model for something as complex as Time Warp required some approximations to be made to make the mathematics tractable, so we validated that the model gave accurate predictions by comparing results produced by the model with measurements of GTW.

We later expanded this collaboration to include Dick Serfozo, another expert in stochastic models and a Georgia Tech professor in the School of Industrial and Systems Engineering. Our interest here was in understanding Time Warp performance when one limited the amount of memory allocated to the parallel simulation. At the time, there was much interest in developing techniques to execute Time Warp in situations with limited memory. David Jefferson had recently published his work on the cancelback algorithm that enabled Time Warp to execute, albeit slowly, within a constant factor of the amount of memory needed for a sequential execution. We wanted to examine how Time Warp performance would change as additional memory was provided. One of Dick's PhD students, Liang Chen, did the heavy lifting in developing an analytic model for Time Warp with limited memory (Akyildiz et al. 1993). We found that Time Warp performance increases rapidly as more memory is added, then hits a knee where diminishing returns set in, and subsequent additional memory provide modest or no benefit. In fact, if too much memory is provided, performance can actually decline as Time Warp becomes overly optimistic, and rolls back more computation than desired. Like our earlier work, we validated the models by comparing performance predictions made by the model with experimental results of the cancelback algorithm which Samir Das added to GTW.

While the analytic modeling work provided a window into Time Warp performance, it really only analyzed Time Warp performance rather than improving it. Our subsequent efforts focused on putting this work to good use to design efficient Time Warp systems. Specifically, this work highlighted the fact that memory could be used as a way to "throttle" the computation to avoid overly optimistic execution (Das and Fujimoto 1997a). We realized that by monitoring the execution of the Time Warp program and limiting the amount of memory that was provided we could adaptively control the execution of Time Warp to maximize its performance. Specifically, the goal was to keep Time Warp operating near the knee of the performance–memory curve so that Time Warp reaped the performance benefits of additional memory, but did not utilize additional memory as that provided only marginal benefit. It also prevented the computation from moving into overly optimistic modes of execution, something that could occur by giving it too much memory. Samir Das designed, implemented, and validated this approach by showing that his algorithm could dynamically control the amount of memory provided to move the execution to the knee of the performance–memory curve (Das and Fujimoto 1997b). While other researchers had proposed other mechanisms to prevent overly optimistic execution, Samir's work was distinguished by having analytic modeling work as a basis for understanding the throttling mechanism. Further, this work provided a way to automatically adapt the execution to continually optimize performance throughout the parallel execution.

Our interest in monitoring the Time Warp execution and development of adaptive mechanisms to optimize its behavior led to an exploration of load balancing techniques. Here, we were particularly interested in the "background" execution of Time Warp on a distributed computing platform that was shared with other computations. We realized that execution of Time Warp on shared platforms presented a challenging test case because LPs that had to compete with many other computations to get CPU cycles would advance in simulation time slowly compared to LPs that had fewer other competitors for CPU cycles. This would cause the latter to race ahead, leading to much rollback and an inefficient execution. PhD student Chris Carothers developed a dynamic load distribution algorithm for GTW that would control the Time Warp execution in such circumstances. He implemented the algorithm and showed that it would yield efficient execution of Time Warp programs in the presence of other computations (Carothers and Fujimoto 2000).

Independent of our research, the concept of executing codes on shared remote servers became increasingly popular in industry, and is now referred to as cloud computing. We continued to have interest in this area. Our initial work was motivated by a somewhat different paradigm, that used in the Search for Extra-Terrestrial Intelligence (SETI) project that farmed out computations to remote servers, e.g., otherwise idle workstations. Alfred Park developed a novel execution approach based on farming out computations and later collecting their results (Park and Fujimoto 2007, 2012). While much of his early work focused on conservatively synchronized codes, he also examined the execution of Time Warp in grid computing environments (Park and Fujimoto 2008). Other subsequent work looked at the implementation of Time Warp in cloud computing environments (Malik et al. 2010).

7.18 The High-Level Architecture

In the U.S., the Department of Defense (DoD) was one of the principle organizations interested in developing modeling and simulation technologies. Beginning in the 1980's with a highly successful Defense Advanced Research Projects Agency

(DARPA) project called SIMNET (Miller and Thorpe 1995), the hot topic in DoD concerned how to reuse existing simulation models and get them to interoperate using distributed computing platforms. By the 1980s simulators to train pilots and equipment operators had become common, and were increasing in realism and sophistication with advances in computer graphics. Around that time local area networks were being invented, raising the possibility of interconnecting these simulators to create a kind of virtual battlefield with many simulated platforms to train military personnel. This was the forerunner to multiplayer video games that are common today. The SIMNET project demonstrated that this was a feasible concept. Throughout the 1990s there was a great emphasis, and investment, in creating interoperable distributed simulations. For example, the Distributed Interactive Simulations (DIS) (IEEE Std 1278.2-1995 1995) standards were developed to facilitate interoperability. While SIMNET and DIS focused on training simulations, the desire for interoperability spread to simulation models used for analysis, the area where Time Warp focused. An effort focusing on integrating wargame simulations called the Aggregate Level Simulation Protocol (ALSP) had begun with this objective (Wilson and Weatherly 1994).

In the mid-1990s an ambitious effort began to, in effect, combine the lessons learned in DIS and ALSP to create a common modeling and simulation architecture spanning the entire DoD. This effort, called the High-Level Architecture (HLA) was being led by Judith Dahmann of the Defense Modeling and Simulation Organization (DMSO). Just as HLA was getting underway, I received a phone call from Richard Weatherly of the MITRE Corporation. Richard had led the ALSP effort, and now was tasked with developing prototype implementations of the new HLA standard that was being developed. I had not met Weatherly before, but he had read some of my papers on parallel discrete event simulation, and asked if I was interested in getting involved in the HLA effort. I immediately accepted and became the technical lead of a working group tasked with defining the time management services of the standard that were largely concerned with synchronization issues in distributed simulations.

How does HLA relate to Time Warp? HLA was about getting separately developed simulations to work together in distributed computing environments. As such, a central objective of the time management services was to get simulations utilizing different mechanisms for time advancement to mesh together and interoperate. This meant getting time stepped and event-driven simulations to be able to synchronize and coordinate their time advances. With parallel discrete event simulation technologies becoming mature and finding some use in the DoD, e.g., through the ballistic missile defense program mentioned earlier, we extended this object to include *parallel* simulators, including both ones synchronized using conservative synchronization algorithms as well as optimistic ones such as Time Warp. Developing an approach to integrate all of these mechanisms together, while still ensuring that a reasonably efficient implementation could be realized was a significant challenge. Our task was to define the application program interface (API) that would be used by these different simulations, both parallel and sequential.

The main observation that enabled us to define an API for integrating these simulations was to realize that there was actually much commonality between conservative and optimistic approaches. This had been realized earlier, as described in work such as Chandy and Sherman's space-time simulation approach (Chandy and Sherman 1989). For example, the key quantity that conservative simulation algorithms needed to compute was a lower bound on the timestamp (LBTS) of messages that might be later received by an LP, because once this had been computed, the LP would know that all events with time stamp less than its LBTS value could be safely processed without fear of later receiving an event with smaller timestamp. LBTS is a close cousin of Global Virtual Time (GVT) used in Time Warp to determine a lower bound on the timestamp of any future rollback; rollbacks are also caused by receiving a message or anti-message in an LP's simulated past. Thus, if the time management services provided a means of computing LBTS and returning this information to each LP, or federate in HLA terminology, such a service would support both conservative and optimistic simulations.

A second observation was that Time Warp could be viewed as a conservative execution, but with the ability to optimistically process messages with time stamp larger than the LBTS value. Our approach in the HLA was to start with a conservatively synchronized distributed simulation, and add the mechanisms needed to allow optimistic processing. At the heart of the HLA's conservative synchronization approach were two mechanisms: one that guaranteed that messages would be delivered to a federate in timestamp order, and a second that enabled a federate to advance its simulation time in a way that guaranteed it would not receive any messages in its simulated past. To support Time Warp, a service was needed to also deliver optimistic messages to each federate, i.e., messages where it was possible a smaller timestamp message might later be received. This led to the *Flush Queue Request* service that when invoked, delivered all incoming messages to the federate invoking the service, regardless of its timestamp relative to LBTS or other events that had been previously delivered. Once these optimistic events were delivered to the federate, it was free to process them optimistically, risking the need for rollback. Should rollback be later required, this (specifically state-saving) was something the federate would have to implement on its own, so the API did not have to deal with this concern. Finally, the other part that was needed was a way to implement Time Warp anti-messages. Here, we leveraged our prior work in application-defined event cancellation (Lomow et al. 1991), discussed earlier. We realized that event retraction, i.e., an application unscheduling a previously scheduled event was a desirable application feature, independent of the underlying synchronization approach, be it optimistic or conservative. This led to the creation of the *Retract Message* service that could be used by conservative federates to unschedule events, as well as optimistic federates to implement Time Warp anti-messages. The services were defined so that the retraction of previously scheduled messages was transparent to conservative federates receiving the retracted message in that the original message would not be delivered to the federate until it could be guaranteed that it would not be later retracted. Of course, this guarantee could not be preserved for optimistic federates that were using the *Flush Queue* service. In this case, if the message being retracted had already been delivered to the federate, the retraction request was simply treated by the receiving federate as an anti-message, and processed according to Time Warp event processing rules. In this way, the HLA time management services could support both conservative and optimistic simulations executing within the same federated distributed simulation. In the end, all this came together relatively smoothly. I worked out many of the technical details on planes between Atlanta and DC, a place free from phone calls, emails (this was well before inflight wifi), and visitors and students coming to my door.

Technical issues aside, a perhaps bigger challenge in the HLA effort was to build consensus among the various stakeholders who had their own ideas of how things should be done. Many folks in the DoD M&S community were not familiar with PDES technologies, and had never heard of CMB or Time Warp. Fortunately, I had some help here. A piece of Jade Simulations, a company founded by Brian Unger at the University of Calgary, was bought by Science Applications International Corporation (SAIC), a major defense contractor with a large stake in defense M&S in general, and HLA in particular. Two of Brian's former students, Darrin West and Larry Mellon, were leading much of the HLA effort in SAIC, and were very well versed in PDES. Together with a small team representing various constituencies we managed to get consensus on the specifications.

The High-Level Architecture was approved as the standard architecture for all M&S in the U.S. Department of Defense in 1996, and was subsequently standardized by IEEE (IEEE Std 1516.2-2000 2000) and later updated (IEEE Std 1516.1-2010 2010). Although the HLA "mandate" that originally required all M&S programs in the U.S. Department of Defense to become compliant with HLA was later rescinded, I was pleased to see HLA come up over the years in many different application areas other than defense. In retrospect, I have the highest regard for many of the individuals involved in the HLA effort. Richard Weatherly and his team at MITRE proved to be very capable developers. I never envied their task of developing an implementation of the standard while the standard itself was being changed! Richard was open to new ideas, and willing to incorporate changes that seemed well motivated. There were substantial differences and plenty of heated arguments during the HLA development. Many of these conflicts came to Judith Dahmann. I also did not envy her task. I found Judy to be both personable as well as being a strong and capable leader. She deserves enormous credit in seeing the HLA effort through to completion. I will also remember Judith as having a subtle sense of humor. The HLA effort had periodic meetings of the Architecture Management Group (AMG) at the DMSO headquarters in Alexandria Virginia, that included perhaps a hundred or so key stakeholders across the DoD M&S community. At one meeting, just before Christmas, Judith delivered a present to each attendee—a screw, with, for good measure, an attached bolt. Her only comment at the meeting was that the interpretation was left entirely up to each of us!

7.19 Pivoting to Federated Simulations

Subsequent to the adoption and standardization of the HLA by IEEE, my research program made a deliberate shift toward the federated simulation approach used in HLA. One of the challenges back then in the PDES community that persists even today concerned creation of PDES codes. Despite substantial efforts, application development still requires a certain amount of sophistication in parallel computing in addition to model expertise. The HLA demonstrated a pathway to take sequential simulation code that had never been developed for parallel or distributed execution, and transform it into a form suitable for parallel execution. Even in the pre-HLA days I saw that developers in the DoD were creating distributed simulations by, in effect, federating a code with itself to create a distributed version. This was in fact easier than federating different simulations because many interoperability issues associated with interconnecting different codes such as use of different model abstractions and representations disappeared. Thus, this seemed to be a practical approach to easily creating parallel discrete event simulations.

In order to get self-federation to work, one needed high performance runtime infrastructure software to implement synchronization (time management) and communication services. While my work in the previous decade focused on the Georgia Tech Time Warp software that served as a platform for our research in PDES, I decided to make a deliberate shift, to focus on HLA-like federated simulations. I spent the 1997–98 academic year on leave from Georgia Tech at the Defense Evaluation Research Agency in Malvern, England, with the intent of developing a new software base to support future research in federated simulation systems. The target platform used in this work was a set of workstations interconnected by a fast, Myrinet switch. Using communications software called fast-messages developed at the University of Illinois, in a few weeks I developed the first version of the RTI-Kit software and demonstrated it using a simplified version of the HLA API to implement communications and time management services (Fujimoto and Hoare 1998). In my initial work, I conducted a number of benchmarking experiments that demonstrated that RTI-Kit could achieve high performance, and was thus suitable for implementing parallel discrete event simulations.

Over the years that followed, RTI-Kit evolved into the Federated Distributed Simulations Kit (FDK) that became a central software base used by our research group over the next decade. We demonstrated that one could take existing sequential simulation codes and create high performance parallel versions suitable for execution on supercomputers. Not all simulations were suitable for parallelization using this approach. For example, we found that some simulations making extensive use of global data structures that were accessed throughout the entire code could not be easily parallelized. However, well-structured codes could be parallelized.

Our most successful illustration of this approach was to create a high performance, parallel version of the NS2 communication network simulator (Fall 1999).

The original NS2 was developed with no consideration of parallel implementation. In an effort by PhD student George Riley whom I co-supervised with networking research Mostafa Ammar, we created a parallel version dubbed PDNS (parallel/distributed NS) (Riley et al. 2004). Through a DARPA-funded project focused on large scale network simulation we demonstrated the largest available discrete event network simulations of the day, executing on more than a thousand processors of a supercomputer at the Pittsburgh Supercomputing Center (Fujimoto et al. 2003).

Much of our work in federated parallel simulations focused on conservative synchronization techniques, because existing sequential codes were not coded to include rollback, and adding rollback mechanisms was not straightforward. However, the federated approach is applicable to Time Warp simulations, or codes that were designed to include rollback. We demonstrated the use of HLA time management services to create Time Warp simulations with a prototype that a student named Steve Ferenci developed (Ferenci et al. 2000).

7.20 In Conclusion: The Future

More than 35 years since Time Warp was invented research continues both in its application and the technology itself. The goal of a general purpose parallel discrete event simulation engine that application developers can readily use without knowledge of parallel processing techniques continues to be elusive and the technology has not fulfilled its potential to see widespread adoption in industry. However, modern research in Time Warp systems is being driven more by the changing world and new technology developments rather than classical PDES problems defined decades ago.

One direction for research in Time Warp is driven from below: new computing platforms on which Time Warp simulations execute. Cloud computing, massively parallel supercomputers, graphical processing units (GPUs), and mobile computing platforms all present new challenges for running Time Warp simulations. For example, cloud computing platforms raise challenges due to the shared nature of the platform, and substantial communication delays, as noted earlier. Massively parallel machines raise questions concerning scalability, especially for real-world applications that are often highly irregular and contain inherent bottlenecks. GPUs, and more broadly heterogeneous computing platforms utilize SIMD, data parallel modes of execution that are very different from the platforms on which Time Warp was originally invented. Mobile computing platforms where data-driven simulations are used to monitor and manage operational systems again present new unexplored challenges for Time Warp. These challenges are discussed in greater detail in Fujimoto (2016).

Another research challenge faced by data centers, supercomputers, and mobile computing platforms concerns the amount of power consumed by the simulation. For mobile computing platforms energy consumption affects battery life, so it is an area of great concern. In supercomputers, power consumption has become a major issue, limiting the performance of supercomputing nodes, and incurring substantial costs in operating data centers. Little is known concerning the power and energy consumption properties of Time Warp, and distributed simulations in general. This represents another important line of inquiry for future research in Time Warp.

More than three decades after its creation, Time Warp continues to be an active area of study and investigation in the parallel and distributed simulation community. While much progress has been made, the world has changed, raising new research challenges and opportunities that did not exist when it was invented. While much has been learned concerning Time Warp and its application, it will likely remain at the center of much parallel and distributed simulation research into the foreseeable future.

References

Akyildiz IF, Chen L, Das SR, Fujimoto RM, Serfozo R (1993) The effect of memory capacity on time warp performance. J Parallel Distrib Comput 18(4):411–422

Bagrodia R, Chandy KM, Liao WT (1991) A unifying framework for distributed simulation. ACM Trans Model Comput Simul 1(4):348–385

Barnes Jr PD, Carothers C, Jefferson D, LaPre J (2013) Warp speed: executing time warp on 1,966,080 cores. In: ACM SIGSIM principles of advanced discrete simulation (PADS), Montreal, Quebec, Canada, May 2013

Balzer RM (1969) EXDAMS: extendable debugging and monitoring system. In: Proceedings AFIPS'69 spring joint computer conference, pp 567–580, May 14–16, 1969

Berry O, Jefferson D (1985) Critical path analysis of distributed simulation. In: Society for Computer Simulation Conference on Distributed Simulation, San Diego, January 24–26, 1985

Bryant R (1977) Simulation of packet communication architecture computer systems. MS thesis, MIT/LCS/TR-188, November 1977

Buzzell CA, Fujimoto RM, Robb MJ (1990) Modular VME rollback hardware for time warp. In: SCS distributed simulation conference, pp 153–156

Carothers CD, Bauer D, Pearce S (2002) ROSS: A high-performance, low-memory, modular time warp system. J Parallel Distrib Comput 62(11):1648–1669

Carothers CD, Fujimoto RM (2000) Efficient execution of time warp programs on heterogeneous, NOW platforms. IEEE Trans Parallel Distrib Syst **11**(3):299–317

Carothers CD, Perumalla K, Fujimoto RM (1999) Efficient optimistic parallel simulation using reverse computation. ACM Trans Model Comput Simul 9(3):224–253

Chandy KM, Misra J (1979) Distributed simulation: a case study in design and verification of distributed programs. IEEE Trans Softw Eng SE-5(5)

Chandy KM, Misra J (1981) Asynchronous distributed simulation via a sequence of parallel computations. In: CACM, vol 24, no 11, April 1981

Chandy KM, Sherman R (1989) Space, time, and simulation. Proceedings of the SCS multiconference on distributed simulation, SCS simulation series 21:53–57

Das S, Fujimoto RM, Panesar K, Allison D, Hybinette M (1994) GTW: a time warp system for shared memory multiprocessors. In: Proceedings of the 1994 winter simulation conference, pp 1332–1339

Das SR, Fujimoto RM (1997a) adaptive memory management and optimism control in time warp. ACM Trans Model Comput Simul 7(2):239–271

Das SR, Fujimoto RM (1997b) An empirical evaluation of performance-memory tradeoffs in time warp. IEEE Trans Parallel Distrib Syst 8(2):210–224

Fall K (1999) Network emulation in the VINT/NS simulator. In: Proceedings IEEE international symposium on computers and communications, pp 244—250

Ferenci S, Fujimoto R, Ammar MH, Perumalla K (2002) Updateable simulation of communication networks. In: 16th workshop on parallel and distributed simulation

Ferenci S, Perumalla K, Fujimoto R (2000) An approach to federating parallel simulators. In: 14th workshop on parallel and distributed simulation

Fujimoto R (2000) Parallel and distributed simulation systems. Wiley

Fujimoto R, Hunter M, Sirichoke J, Palekar M, Kim H-K, Suh W (2007) Ad hoc distributed simulations. Principles of advanced and distributed simulation. IEEE, San Diego, CA, pp 15–24

Fujimoto RM (1988) Performance measurements of distributed simulation strategies. Distrib Simul SCS. 10:14–20

Fujimoto RM (1989a) Time warp on a shared memory multiprocessor. Trans Soc Comput Simul 6 (3):211–239

Fujimoto RM (1989) The virtual time machine. In: International symposium on parallel algorithms and architectures, pp 199–208

Fujimoto RM (1990) Performance of time warp under synthetic workloads. Proc SCS Multiconf Distrib Simul 22:23–28

Fujimoto RM (2016) Research challenges in parallel and distributed simulation. ACM Trans Model Comput Simul 24(4)

Fujimoto RM, Hoare P (1998) HLA RTI performance in high speed LAN environments. In: Proceedings of the fall simulation interoperability workshop, Orlando, FL

Fujimoto RM, Hybinette M (1997) Computing global virtual time in shared memory multiprocessors. ACM Trans Model Comput Simul 7(4):425–446

Fujimoto RM, Perumalla KS, Park A, Wu H, Ammar M, Riley GF (2003) Large-scale network simulation—how big? How fast? Modeling, analysis and simulation of computer and telecommunication systems

Fujimoto RM, Tsai JJ, Gopalakrishnan GC (1992) Design and evaluation of the rollback chip: special purpose hardware for time warp. IEEE Trans Comput 41(1):68–82

Fujimoto RM (2000) Parallel and distributed simulation systems. Wiley (2000)

Gafni A (1985) Space management and cancellation mechanisms for time warp. PhD dissertation, Department of Computer Science, University of Southern California, TR-85–341, December 1985

Gupta A, Akyildiz IF, Fujimoto RM (1991) Performance analysis of time warp with multiple homogeneous processors. IEEE Trans Softw Eng 17(10):1013–1027

Huang Y-L, Alexopoulos C, Hunter M, Fujimoto RM (2012) Ad hoc distributed simulation methodology for open queueing networks. Trans Soc Model Simul Int 88(7)

Hybinette M, Fujimoto RM (2001) Cloning parallel simulations. ACM Trans Model Comput Simul 11(2):378–407

IEEE Std 1278.2-1995 (1995) IEEE standard for distributed interactive simulation—communication services and profiles. Institute of Electrical and Electronics Engineer Inc, New York, NY

IEEE Std 1516.1-2010 (2010) IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—interface specification. Institute of Electrical and Electronic Engineers Inc, New York, NY

IEEE Std 1516.2-2000 (2000) IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—object model template (OMT) specification. Institute of Electrical and Electronic Engineers Inc, New York, NY

Jefferson D, Sowizral H (1982) Fast concurrent simulation using the time warp method, Part I: Local control. RAND note N-1906-AF, The RAND corporation, Santa Monica, California, December 1982

Jefferson D, Witkowski A (1984) An approach to performance analysis of timestamp-oriented synchronization mechanisms. In: acm symposium on the principles of distributed computing, Vancouver, BC, August 1984

Jefferson D, Beckman B, Hughes S, Levy E, Litwin T, Spagnuolo J, Vavrus J, Wieland F, Zimmerman B (1985) Implementation of time warp on the caltech hypercube. Society for Computer Simulation Conference on Distributed Simulation, San Diego, January 24–26, 1985

Jefferson D (1985) Virtual time. ACM Trans Program Lang Syst (TOPLAS) 7(3):404–425

Jefferson D, Motro A (1986) The time warp mechanism for database concurrency control. In: Proceedings of the IEEE 2nd international conference on data engineering, Los Angeles, CA, February 4–6 1986

Jefferson D, Beckman B, Wieland F, Blume L, DiLoreto M, Hontalas P, Laroche P, Sturdevant K, Tupman J, Warren V, Wedel J, Younger H, Bellenot S (1987) Distributed simulation and the time warp operating system. In: 11th symposium on operating systems principles (SOSP), Austin, TX, November 1987

Jefferson D (1990) Virtual time II: the cancelback protocol for storage management in time warp. In: Proceedings of the ACM symposium on the principles of distributed computing, Quebec City, Quebec, August 1990

Jefferson D, Reiher P (1991) Supercritical speedup. In: ANSS'91 proceedings of the 24th annual symposium on simulation, 1991

Jones DW (1986) An empirical comparison of priority-queue and event-set implementations. Commun ACM 29(4):300–311

Kung HT, Robinson JT (1981) On optimistic methods for concurrency control. ACM Trans Database Syst 6(2):213–226 (1981)

Lomow G, Das SR, Fujimoto RM (1991) Mechanisms for user invoked retraction of events in time warp. ACM Trans Model Comput Simul 1(3):219–243

Malik AW, Park AJ, Fujimoto RM (2010) An optimistic parallel simulation protocol for cloud computing environments. SCS Model Simul Mag 1(4). Society for Modeling and Simulation International

Miller DC, Thorpe JA (1995) SIMNET: the advent of simulator networking. Proc IEEE 83 (8):1114–1123

Mitre Corp (1997) DPAT: detailed policy assessment tool, brochure, Center for Advanced Aviation System Development (CAASD)

Nicol D, Liu X (1997) The dark side of risk (what your mother never told you about time warp). In: Proceedings of 11th workshop on parallel and distributed simulation (PADS), Lockenhaus, Austria, June 1997

Park A, Fujimoto RM (2007) A scalable framework for parallel discrete event simulations on desktop grids. In: 8th IEEE/ACM International Conference On Grid Computing

Park A, Fujimoto RM (2008) Optimistic parallel simulation over public resource-computing infrastructures and desktop grids. In: Workshop on distributed simulations and real-time applications

Park A, Fujimoto RM (2012) Efficient master/worker parallel discrete event simulation on metacomputing systems. IEEE Trans Parallel Distrib Syst 23(5)

Perumalla KS (2005) μsik-a micro-kernel for parallel/distributed simulation systems. In: Workshop on principles of advanced and distributed simulation. IEEE, pp 59–68

Ravi TM, Jefferson D (1988) Message routing to migrating processes. In: Proceedings of the 1988 international conference on parallel processing (ICPP), August 15–18, 1988

Reiher P, Jefferson D (1990) Virtual time-based dynamic load management in the time warp operating system. In: Distributed simulation, Nicol D, Fujimoto R (eds), Simulation series, vol 22, no 2. Society for Computer Simulation, San Diego, January 1990

Riley G, Ammar M, Fujimoto RM, Park A, Perumalla K, Xu D (2004) A federated approach to distributed network simulation. ACM Trans Model Comput Simul 14(1):116–148

Schordan M, Jefferson D, Barnes Jr P, Oppelstrup T, Quinlan D (2015) Reverse code generation for parallel discrete event simulation. In: 7th conference on reversible computation, Grenoble, France, July 16–17, 2015

Schordan M, Oppelstrup T, Jefferson D, Barnes P, Quinlan D (2016) Automatic generation of reversible C++ code and its performance in a scalable kinetic Monte-Carlo application. In: SIGSIM PADS (Principles of advanced discrete simulation), Banff, Alberta, Canada, May 16–18, 2016

Tang Y, Perumalla KS, Fujimoto RM, Karimabadi H, Driscoll J, Omelchenko Y (2006) Optimistic simulations of physical systems using reverse computation. Simul: Trans Soc Model Simul Int 82(1):61–73

Vulov G, Hou C, Vuduc R, Quinlan D, Fujimoto RM, Jefferson D (2011) The backstroke framework for source level reverse computation applied to parallel discrete event simulation. In: Winter simulation conference

Wieland F (2001) Practical parallel simulation applied to aviation modeling. In: 15th workshop on parallel and distributed simulation, Lake Arrowhead, CA

Wieland F, Hawley L, Feinberg A, DiLoreto M, Blume L, Reiher P, Beckman B, Hontalas P, Bellenot S, Jefferson D (1989) Distributed combat simulation and time warp: the model and its performance. In: Distributed simulation, Unger B, Fujimoto R (eds) Simulation series, vol 21, no 2. Society for Computer Simulation, San Diego, 1989

Wilson AL, Weatherly RM (1994) The aggregate level simulation protocol: an evolving system. In: Proceedings of the 1994 winter simulation conference, pp 781–787