Parallel simulation Researches survey

Introduction

The Following survey aims at identifying some of the prominent researches in the filed of modeling and simulation, by means of their published papers. The selected researches are Richard Fujimoto, David M.Nicol, and Philip A.Wilsey. The following document will attempt to summarize some of their latest works in the field of modelling and simulation, as for the selection of the topics it was based according to their relativity to our interest in parallel simulation of complex networks.

1. Richard Fujimoto

Richard Fujimoto is a prominent researcher in the field of Parallel and Distributed and Simulation. Currently he is a professor in the College of Computing at Georgia Institute of Technology. He got his PhD and MS degrees from the University of California at Berkeley in 1980 and 1983 respectively. He has been active in research in the area of Parallel and Distributed Simulation since 1985. He has given tutorials and delivered lectures on parallel simulation in leading conferences around the globe. He is also very active with the D.O.D (Department of Defense) research activities; especially recently he is the technical lead for time management issues for D.O.D's HLA (High Level Architecture).

Also, he is a contributing member of various IEEE societies including the one on Parallel and Distributed Simulation. Besides IEEE, he is an area editor for ACM Transactions on Modeling and Computer Simulation, has chaired the steering committee for the Workshop on Parallel and Distributed Simulation (PADS) from 1990 to 1998. He was also a member of the Conference Committee for the Simulation Interoperability Workshop. In addition to numerous conference and journal contributions, he has co-authored books on Parallel and Distributed Simulation too. In the past decade he has published research activities in Parallel and Distributed Simulation of Communication Networks.

<u>Position Statement of Richard Fujimoto:</u> [10]

Interoperable distributed simulations have been widely used for D.O.D activities but this technology is yet to find widespread application for the non-military purposes. Most importantly, the feasibility must be sufficiently attractive for a business to invest in the initial expenditures in technology. Embedded computing industry provides a good scope for modeling and simulation. Embedded computers are used to make “smart” devices. Parallel networks of these smart devices will add another dimension i.e. devices will be capable to anticipate and adapt to future events. The distributed systems of embedded devices must be power efficient and their modeling and simulation process must be automated. Interoperability issues amongst components from different or even the same manufacturer must be resolved. Permulla et al, 2002 describes a simulation of a military network using ns2 and GloMoSim. In this case the network models an offshore landing. The network provides communication between troops on the land and naval ships. The simulation models actual networks. Such a simulation can be modeled for non-military purposes too.

(1) Experiences Parallelizing a Commercial Network Simulator

Following is an overview of a paper by Dr. Fujimoto, relating to “Experiences parallelizing a commercial network simulator”

This paper approaches a methodology which extends sequential simulators to run on parallel machines. This methodology will be applied to OPNET simulator. The results show that considerable speedup can be obtained for some OPNET models provided proper partitioning strategies are implemented and simulation attributes are adjusted appropriately.

It is very expensive, time consuming and in some cases impossible to construct real models of huge networks. It is also impractical to deploy new protocols throughout the internet. Modeling and simulation of networks over a single processor is often time consuming too. Parallel and distributed simulation provides one solution to this problem. There have been a number of parallel simulators built over the past decade. In spite of these endeavors, sequential simulators are still widely used today. This is due to the overheads in transition to new software running on different languages.

The approach in this paper is to parallelize sequential simulators. The methodology is to decompose the system being modeled into subsystems, and running the subsystems on different processors. The methodology implemented in this research particularly assumes that source code of the simulation programs is not available. Hence, there will be minimal changes to the original sequential simulator.

Parallel Network Simulation Architecture:

Each federate runs a sub-network. A sequential simulator runs this sub-network. RTI provides the communication interface between the sequential simulators running on different machines. A proxy model is added to each federate running on a single processor, providing an interface between the sequential simulator and the RTI.

Methodology for Parallel Simulation:

1. The whole network is partitioned into sub-networks.
2. Each sub-network runs on a different processor
3. Proxy model is responsible for communication of one sub-model with the others
4. Optimizations must be applied to improve performance

Data flow across federates:

A network model consists of node objects and link objects. When a big network is broken down into sub-nets, some links are broken. So end nodes of some links are not available (they are in a different federate). Proxy objects are used to communicate with these nodes. Proxy objects make use of the RTI functions too. Another important feature of the proxy objects is too translate native simulator message format to a well-known one used by the proxies and vice versa.

Proxy is divided into two parts. 1. gen_proxy, independent of the protocol, takes care of the time and events. 2. pro_proxy, protocol dependent portion to process specific protocol packets. Data channels between federates may be uni-directional or bi-directional. These channels are implemented based on the HLA publishing/receiving class mechanism.

Simulation Time and Event Management:

As this is a discrete event simulation, unprocessed events are stored in a queue and processed in a time stamp order. Local time of each simulator must be synchronized with the others. Synchronization is a challenging problem. It is to be made sure that no federate receives an event in its past. Therefore synchronization among federates is an important task. For that purpose an LBTS value is maintained and no federate can advance its simulation beyond that LBTS value.

Performance Related Issues

Lookahead is used to improve parallelism and hence performance in the system. The larger the value of lookahead, the more the parallelism in the system. When a federate needs information beyond its sub-model, a ghost object is created that models that specific part of the network. This results in reduced memory burden as compared to defining the overall network in every federate.

Another research in the same filed is related to "**a parallel OPNET simulation**"

Kowing that OPNET consists of an event based simulation engine, libraries to write models in C, drag and drop style graphical interface and a library of network components.

Implementation:

FDK (Federated Simulations Development Kit) developed by Fujimoto et al. at Georgia Tech was used for this project. An important task is to calculate the propogation delays, at the link objects. The proxy model computes real delays. OPNET models heavily rely on global state information. To resolve this issue, ghost objects implemented on each federate, store information of the whole network. This process is static and not modifiable at run-time. OPNET also uses interrupts, that make interaction through RTI very tough, so a detailed analysis of the whole network is required to increase the lookahead.

Performance:

1. Performance is increased has lookahead is increased. For this, either the network model is partitioned at links with low bandwidth, or distance is increased between federates mapped to low bandwidth.
2. An increase in event density improves performance
3. An improvement in traffic locality reduces cost and increases performance.

<u>CONCLUSIONS:</u>

This method is easy if the sequential simulator doesn't extensively use global state information. Problems like zero lookahead and global state make parallelization difficult. Recently OPNET has introduced support for HLA. But this technique is superior because it allows use of existing network models.

(2) Generic Framework for Parallelization of Network Simulations

Another research by Richard Fugimoto is a study of a Generic Framework for Parallelization of Network Simulations.

The goal of the research was to develop and demonstrate a practical, scalable approach to parallel and distributed simulation that will enable widespread reuse of sequential discrete event simulation models and software. The focus was on an approach to parallelization where an existing network simulator was used to build models of subnet works that were composed to create simulations of larger networks.

Simulation tools have not been able to keep up with the rapid increase in the size, complexity and speed of modern networks. Which is why an approach that exploits parallel and distributed simulations is needed to improve the performance of the simulation of networks. The approach used in the paper, was to extend the features of ns, and allow it to be interconnected to create parallel simulations. Each simulator will be given the network topology and data flow characteristics, which describe only a portion of the network being simulated. Interactions between the different simulations were done using a runtime infrastructure. A methodology for parallelization was described for simulations run on shared-memory, symmetric multiprocessors and via distributed computing on several workstations. The basic steps required were:

1. Determine how many processes (threads) will be assigned to run the parallel simulation. Ideally, on a system with n-CPUs, the work would be divided into n-processes.
2. Divide the state set into n partitions and create a one-to-one mapping between partitions and processes.
3. Maintain a separate event list for each physical process, so each process will be concerned with only the events that affect the states in it's state set.
4. Distribute events during the execution among the physical processes.
5. Add a synchronization/communication mechanism to ensure consistent state management between the processes.
6. Perform optimizations

With the above steps a parallel simulation can be constructed on an SMP. However, there are several issues concerning distributed simulations on separate workstations. The issues concern defining physical and logical connectivity between sub models of a divided simulation model. To define connectivity between sub models, such as a source and a sink, which reside on different workstations, the IP Address and port number is used. The steps needed to create a distributed simulation are to determine routing paths, event time management and event communication.

Routing paths can be determined by the simulator run some existing and well known routing protocols while the simulation is running in order to exchange dynamic routing information between the sub models. Event time management needs to be implemented. This means, that each simulator must determine that no other simulator can create events at an earlier time before it can be allowed to process it’s most recent event. This can be done using a lower bound time-stamp (LBTS). Both event communication and event time management is provided with a runtime library such as RTIKIT, which provides these services using a multicast group management strategy known as MCAST.

Optimizations were made to the event communication/management schemes by decreasing LBTS overhead and using polling on the listener sockets used for communication only when it was sure that it would not block forever. After conducting experiments using an eight-node model in a distributed system using the TCP protocol for communication, an increase in performance was observed that stated a successful parallel simulation.

2. David M.Nicol

Next is yet another researches in the field of network simulation .Mr.David M.Nicol, who is curretnly a Professor in the Electrical and Computer Engineering, department in the univeristy of Illionis. Professor Nicol's area of research is parallel simulation, of large scale networks, either building tools dor analysis or investigation of causes for the precessince of certain applications(such as Worm inestation).

(1) A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations

This paper was a proceeding of the 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems written by David Nicol along with other authors. The purpose of this paper is to model large-scale worm infestations in order to assess their threat levels, evaluate countermeasures and investigate their possible influence on the Internet infrastructure. The paper describes the approach of the simulation, the collection of data and modeling of certain essential model elements, such as topology, population distributions, and scanning traffic.

The method used for modeling Internet-worm infestations is based on a mixed abstraction simulation by using selective abstraction through Epidemiological models combined with detailed protocol models. The epidemiological model originally developed for the study of biological diseases, greatly simplifies modeling the worm propagating in the network because it reduces the complexity of the model, and it is a better match for the limited available data on the events. The epidemiological model also helps in gathering information about worm propagation dynamics and it effect on the routing infrastructure.

To improve the reliability of the simulation, the authors made an assumption that the worm scanning traffic induces an increase in BGP (Border Gateway Protocol) routing message traffic. Based on this assumption, three models are required for simulation; a model of how the worm propagates and infects hosts in the Internet, a traffic model for the scans emitted by the worm and a model of how the worm scans induce stress on routers.

Furthermore, in order to study the system at the level of inter-domain routing, the system is decomposed spatially into autonomous systems (AS's). This would help in developing a stratified epidemic model for worm propagation such that the host population is stratified into AS's.

The underlying data of the simulation includes both stochastic (chaotic) and deterministic versions. Since the population is sufficiently large, the stochastic models are approximated by a system of equations based on a continuous state-continuous time deterministic model. These equations rest upon AS's the law of mass action AS's which incorporates the principle of Homogeneous mixing.

Unfortunately, due to limited time and memory size, there was a constriction in the number of BGP routers used in the mixed abstraction model. As a result the model was down scaled to simulating only a few hundred autonomous systems. However, in the future, the use of parallel execution techniques and judicious abstraction could make the simulation of a few thousand AS's possible. Thus a better interpretation of the model output would be achieved.

(2) Utility Analysis of Parallel Simulation

Another document also published by David M. Nicol carrying the title "Utility Analysis of Parallel Simulation". We shall attmept to summarize the model and partitioning analysis part of the original document. The summary section of this document is divided into two parts, Model (a summary of the model), and partitioning (a summary of the partitioning and analysis section)

1.0 Utility Analysis of Parallel Simulation summary:

1.1 Model:

Recognizing that large problems are user dependent, the approach uses the notation of user defined utility. The problem size described by variable m is supposed to be able to be characterized into problem units, and $\mu(m)$ is used to denote the user utility of simulating a problem with size m. Although the size is discreet, using it as a continues quantity wont effect the obtained results. The purpose of the model is to capture the notion that the users utility grows as the problem size simulated grows. A simple model that expresses a wide rang of growth is $\mu(m) = c_m m^{\alpha}$, for some positive constant c_m. Exponent α expresses how rabidly the utility grows, and turns out to be a key determinant to the optimal system configuration. With large problem sizes, and to push the system to equilibrium, the problem must be advanced further into simulation time. This implies a trade-off problem, is the added utility large enough to offset the added computational cost?

With a parallel machine with N processors, the system might be used in a variety of ways to execute the simulation. One extreme is using all processors concurrently to run problem not larger than size m_x, another extreme is to use the entire machine in parallel to simulate one problem of size no grater than Nm_x.

A utility rate can be associated with each partition of the system, and can be calculated by dividing the utility gained by one experiment of the chosen size by the time needed to complete the experiment. The aggregate utility rate can be found by adding all the systems partitions utility rates, and can be used to compare different configuration of the system. The approach can be extended by adding a cost that varies with the number of used processors of the parallel machine. When approaching optimization problems with a model that is dependent on the problem size, and the number of processors used, it shows that the maximized aggregate rate is a result of using one of the extremes; fully parallel or not at all. "Determination of which extreme is best depends on the rate of utility increase (α) in problem size, the rate (ϵ) at which length of the simulation must grow to reach equilibrium as the problem size grows, and the rate of performance increase (β) as additional processors are used in the simulation. Of these only α is subjective, and the user's perception of how utility increases in problem size effectively determines which of the extreme configurations optimizes the aggregate utility rate."[1.3]

The cost of using a machine with *N* processors is supposed to be proportional to the execution time multiplied by N^{ρ}, for ρ > 0. for any value for ρ > 0 it is shown that the configuration that optimizes utility rate b ρ > 0 per unit cost is an extreme.

The native application behaviors is described be the problem size *m*, and the length of simulation time *T(m)* needed for interesting run of the problem of size *m*. The simulation length can be either dependent or independent of the problem size.

Two characteristics are used to describe the capabilities of simulation. The first is (γ) the average execution time needed to evaluate a unit of problem simulation second on one CPU. The dependence of both the simulation length, and the execution time per unit problem can be denoted by modeling the execution on processor as

$$x(m,1) = c_t\left(\gamma m^{\varepsilon_1}\right) * m * m^{\varepsilon_2}$$
$$x(m,1) = c_t \gamma * m^{1+\varepsilon}, where\, \varepsilon = \varepsilon_1 + \varepsilon_2$$

the second characteristic describes the ability of the simulation to be parallel. Letting a(*N*) be the speedup of execution on a parallel system sing *N* processors, the speedup is let to be a(*N*) = N^{β}, for β Є(0, 1), and *N* Є[1, N_x]. this model accounts for behavior where adding processors improves performance.

Using these concepts the execution time of an application is expressed as

$$x(m,n) = \frac{x(m,1)}{a(n)}$$
$$x(m,n) = \frac{c_t * \gamma * m^{1+c}}{n^{\beta}}$$

Using the utility model $\mu(m)$ the utility rate at which utility is accrued simulating a problem of size *m*, using *N* processors is

$$\lambda_{\mu}(m,n) = \frac{\mu(n)}{x(m,n)}$$
$$\lambda_{\mu}(m,n) = K_{\mu} m^{\alpha-(1+\varepsilon)} N^{\beta}$$
$$Where\, K_{\mu} = \frac{C_m}{\gamma * C_t}$$

1.2 Partitioning:

For a parallel system, and to employ the systems resources, many partitioning possibilities can be used, for example ½ the system can work on one problem, a ¼ on a smaller problem, and the remaining ¼ on individual problems. An analysis was conducted using the utility function and the utility rat equation on different partitioning scenarios. In the analysis the equations were treated as continues although they are discrete to simplify the analysis. The analysis was conducted through assuming a number of theories and lemmas and proving them, and in all of the lemmas and theories the analysis showed that the optimal partitioning is fully parallel or fully serial.

2.0 Conclusion:

When using parallel systems, partitioning is used to decide whether to run simulations using the entire machine in parallel, in serial, or a mix of both. Using a utility function, and a utility rate function that was derived using the equations for simulation length, and execution time, it was shown that the most optimized solution to maximize the aggregate rate at which user’s utility is accrued is an extreme. The two extremes that maximize the aggregate rate were either using the machine fully parallel, or using it fully serial.

3. Philip A. Wilsey

Another promenant researhcer in the filed of network simulation and analysis is Mr. Philip A. Wilsey whose work in the area of parallel simulation of complex network has greatly benefeted other researhces in the files. We will attempt to look on his analysis of the Active Networking Architecture, in the following overview of his work

Active networking architecture enabled the integration of embedded computational abilities, within conventional networks. Therefore increasing efficiency and capacity of current networks through incrementing their customization ability for each specific computation. However this happened with the added side effect of increased complexity and the massive increase in size, thus making conventional analytical methods of modelling, simulation and analysis techniques obsolete. The Answer was a discrete event simulation technique so as to simplify and parallelize the simulation process so as to maintain maximum efficiency.

The paper at hand by Dhananjai M. Rao and Philip A. Wilsey, describes an integrated environment for the modelling and simulation (including parallel mode simulation) of Active networks. The Environment “Active Network Simulation Environment” (A.N.S.E.) incorporates a synchronized Time warp simulation kernel of WARPED. Thus enabling parallel simulation, it also provides support for Packet Language for Active Network (P.L.A.N.). In this comprehensive survey we shall attempt to shed the light on the theory behind the architecture, and the construction of A.N.S.E.

<u>Theory behind the Architecture:</u>

A.N.S.E. was created with the intention of parallel simulation in mind from day one, this lead to the development of a framework around a general purpose discrete simulation kernel, with the use of object oriented rather than a structured infrastructure; this provided us with a mush required “separation of concerns”. And the ability to use various simulation kernels without need to change the modules already created.

As mentioned earlier ANSE incorporates a time warp WARPED synchronized kernel. WARPED is an API(Application program Interface) with various implementation, one being a Time warped optimistic synchronization strategy. This implementation has been used since an active network is based on the idea that the nodes constituting the network have a customizable computational ability on the datagrams flowing through it.
Thus enabling Kernel to organize the simulation into asynchronous communicating logical processes (LP’s). Communication between various LP’s is done through exchange of virtual-time stamps, while each process maintains its own Local virtual time (LVT).

However this mechanism is error prone with errors referred to as straggler events may occur. Nevertheless a rollback feature is made available to recover from the causality error. Recovery is only for LP’s prior to the error, while those where the error was created in or resulted in creating the error are destroyed, and then continue execution of LP’s in their previous order. Each LP also maintains a list of input/outputs and another for transitions between states to perform efficient rollbacks, discarding events that are no longer needed. Finally the Warp kernel provides an Interface to build LP's according to the Jefferson definition of time Warp. Also the ability to create different LP's with unique state definitions, with the clustering nomenclature adding more simplicity to the API, without the hassle of having to synchronize clusters, since control is exchanged between the application and the simulation Kernel through cooperative use of function calls.

Overview of the blocks constructing ANSE:

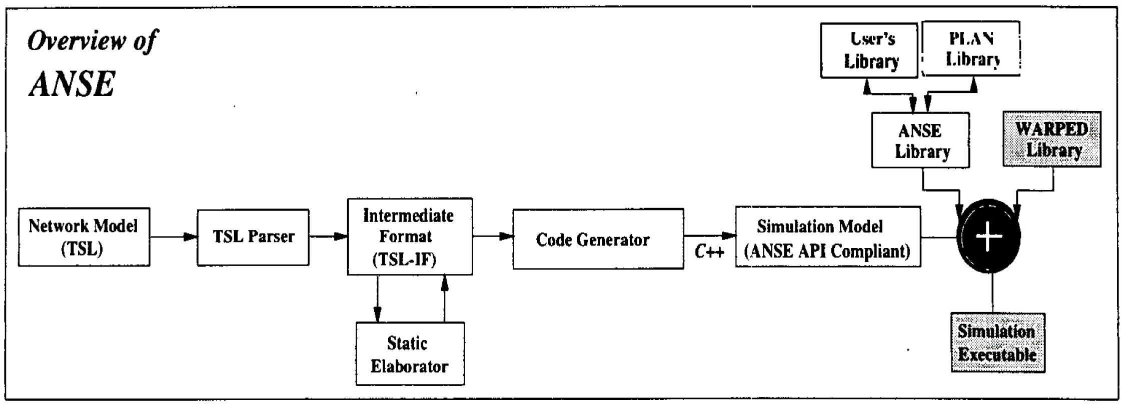

Figure 1. Overview of ANSE

We shall attempt to look at each module presented in figure 1, and describe its operation, in order to understand the construction of ANSE.

Topology Specification Language (TSL):

The main input of the environment which is to be simulated (the network at hand) is given in TSL. The Backus Normal Form (BNF) grammar of TSL specifies a set of interconnected topology specifications each consisting of 3 main categories

1. Object definition section
   Contain module details, which will be used in the simulation
2. Object instantiation section
   Specifies the various nodes constituting the Topology
3. The Netlist section
   Defines interconnectivity, between variously instantiated nodes.

The topology also makes use of labels to define related segments of the code.

TSL Parser:

The parser is used to convert (Parse) the input topology into an object oriented TSL intermediate format (TSL-IF). TSL-IF is implemented in C++ and is a set of cross referenced classes. It is available through the Purdue Compiler Construction Tool Set (PCCTS). The intermediate Format is accomplished by filling in the references in the various C++ classes with appropriate values.

Static elaborator:

The Part of the Environment used to reformat specification of large networks for the use of smaller sub networks, as "Hierarchical constructs provide convenient techniques to specific large networks by reusing the specification for smaller sub networks". While elaboration is defined to be braking down of large hierarchical constructs into their constituting components. The Elaborated Topology is in TSL-If. The elaborator traverses the user-specified sub-topologies in the model creating and instantiating objects and sub-topologies, as sub-topologies are instantiated they are then imploded into a major (enclosing) topology. Static elaboration is done since we are operating before the code generation step(opposite to the choice of dynamic elaboration).

Code Generator:

Generates a C++ code (simulatable model) from the Elaborated TSL-If description, supplied from the Static elaborator. The generated code is compliant with the ANSE API. It is also worth mentioning that the Code Generator may be replaced to provide compatibility with other frameworks.

ANSE API and Library:

As mentioned earlier ANSE provides an interface to define logical processes (LP's). The processes are defined as entities with the ability to send, receive and act upon events by applying a set of internal states (internal to the LP). The Lp's are created using an object oriented infrastructure with a class performing the role of a master (Object) class which is "NetworkNode", from which all classes are inherited, and created. The API also provides State support through classes such as "NetworkNodeState", and "ActiveNodeState" (baring in mind the role of nodes in creating active network architecture, shows the importance of such classes).

The state classes are used to hold state information for each node/component. This enables the simulation kernel (WARPED) to perform **rollbacks,** thus a recovery mechanism from casual violations that might occur due to the optimistic nature of the time warp simulation. The discrete event in the system is the Packet represented by the Packet Class. Finally it is worth mentioning that the API is created using C++ making use of its robust operation.

PLAN Library:

"PLAN is a simple, functional programming language based on a subset of ML with some added primitives to express remote evaluation" [1]. In active network architecture packets can contain PLAN programs, to help customize operation for various network operations. Same as the API libraries, PLAN also makes use of an object oriented infrastructure enabling the use of Master classes such as "PacketInjectors" to inject PLAN programs or packets into the simulation environment, to give an example.
As for runtime operation the support of PLAN from The ANSE enables simulation of large, complex networks within limited hardware requirements of course to a certain point of complexity.

Conclusion:

In conclusion It is the Testimony of the respected researches who have wrote this paper that "it is better to have a simple, yet flexible language such as TSL, for modelling network Topologies. It is useful to have a clear delineation between the languages for developing the software modules for networking components and network modelling language."[1]

The inter-operability between different types of models, and simulators from my point of view is certainly the greatest achievement of the ANSE Project.

Glossary

Federation: In HLA, a parallel/distributed simulation.

Federate: individual simulator.

Lookahead: "In parallel simulation, it is the minimum of the packet delivery delays in all the links of a sub-model that cross boundaries of partition"

LBTS: Lower bound on time stamp.

References:

[1] Modeling and simulation of Active Networks, by: Dhananj M. Rao and Philip A. Wilsey, Experimental Computer Laboratory.

[10] Distributed Simulation and Industry: Potentials and Pitfalls
Proceedings of the 2002 Winter Simulation Conference