An Object-Oriented Framework for Parallel Simulation of Ultra-large Communication Networks*

Dhananjai Madhava Rao and *Philip A. Wilsey*
Computer Architecture Design Laboratory
Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221–0030
(513) 556-4779, phil.wilsey@uc.edu

Abstract. Communication networks have steadily increased in size and complexity to meet the growing demands of applications. Simulations have been used to model and analyze modern communication networks. Modeling and simulation of networks involving thousands of nodes is a complex task due to the sheer size and complexity of the hardware/software components. Complete simulation models of the ultra-large networks need to be developed in order to conduct in-depth studies of scalability and performance. Parallel simulation techniques need to be carefully exploited to provide optimal time versus resource tradeoffs. Due to the complexity of the system, it becomes critical that the design of such complex frameworks follow well established design principles such as object-oriented design, so as to meet the diverse requirements of portability, maintainability, extensibility, and ease of use. This paper presents the issues involved in the design and implementation of an object-oriented framework to enable parallel simulation of ultra-large communication networks. The framework is developed on an object-oriented parallel simulation kernel based on the Time Warp synchronization paradigm. The paper illustrates the various object-oriented (OO) design techniques employed to configure the framework in order to optimize the tradeoffs between performance and resource utilization. The need for OO techniques to enable simulation of ultra-large network models are illustrated. The application program interfaces needed for model development are also presented along with some experimental results.

1 Introduction

Computer and communication networks have constantly increased in size and complexity to meet the ever growing demands of modern software applications. The Internet, a global data network now interconnects more than 16 million nodes [7]. The underlying networking technologies, both hardware and software, have witnessed corresponding growth in complexity. Modeling and studying today's networks that involve complex interactions is a challenging task [5, 7].

* Support for this work was provided in part by the Advanced Research Projects Agency under contract DABT63–96–C–0055.

Simulations have been employed to aid modeling and analysis of communication networks [5]. Parallel simulation techniques have been used for large simulations in order to meet the resource requirements and time constraints. One of the critical components of simulation analyses are valid models [10]. The network models should reflect the actual network sizes in order to ensure that crucial scalability issues do not dominate during validation of simulation results [7]. Many networking techniques that work fine for small networks of tens or hundreds of computers may become impractical when the network sizes grow [7]. Events that are rare or that do not even occur in toy models may be common in actual networks under study [7]. Since today's networks involve a large number of computers ranging from a few thousands to a few million nodes, modeling and simulating such ultra-large networks is necessary.

Modeling and parallel simulation of ultra-large communication networks involves complex interactions between the various components of the system. Simulation of large networks involving millions of nodes places high demands on system resources; system memory in particular. Hence, the data structures used in the system need to be optimally designed with robust interfaces to enable efficient exchange of control and data in the distributed system [8]. To ease modeling, verification, simulation, and validation of ultra-large networks, well defined design principles such as object-oriented design have to be employed. The complex interactions between the different components of the simulation kernel necessitate well defined and robust interfaces [8]. In order to meet these diverse needs, an object-oriented framework for simulation of ultra-large communication networks was developed. The framework is built around WARPED [8], an optimistic parallel simulator. In conjunction with WARPED, the framework is implemented in C++ and utilizes the object-oriented capabilities of the language. Object-oriented techniques such as inheritance, virtual functions, and overloading have been exploited to develop efficient and robust interfaces.

The issues involved in the design and implementation of the framework along with a compendium of object-oriented techniques used are presented in this paper. The paper attempts to highlight the need for object-oriented design to enable simulation of ultra-large networks. In Section 2, a brief description of WARPED along with the Time Warp paradigm is presented. A detailed description of the framework is presented in Section 3. The issues involved in its design and the various object-oriented techniques employed are also presented in this section. Some of the experiments conducted using the framework are presented in Section 4. Section 5 contains some concluding remarks with pointers to future work.

2 Background

The framework for ultra-large networks simulation is built around WARPED, a parallel optimistic simulator. WARPED uses the Time Warp [3] paradigm to achieve distributed synchronization. A Time Warp synchronized simulation is organized as a set of asynchronous logical processes (LPs) that represent the different physical processes being modeled. The LPs communicate between each other by exchanging discrete *virtual time* stamped events [3]. Virtual Time [4] is used to model the passage of time and define a total order on the events in the system. The WARPED kernel presents an interface to build logical processes with unique definitions of state [8]. The basic functionality for sending and receiving events between LPs using a message passing system is supported by the kernel. In order to reduce communication overheads and enable optimal process allocations, LPs are placed into groups called "clusters". Control is exchanged between the application and the simulation kernel through cooperative use of function calls. Each LP asynchronously processes its events incrementing its local virtual time (LVT), changing its state, and generating new events. The LPs must be synchronized in order to maintain the causality of the simulation; although each LP processes local events in their correct time-stamp order, events are not globally ordered. Causality violations are detected when a event with time-stamps lower than the current LVT (a *straggler*) is received. On receiving a straggler event a rollback mechanism [3] is invoked to recover from the causality error. Each LP maintains a queue of state transitions along with lists of input and output events corresponding to each state to enable the recovery process. A periodic garbage collection technique based on Global Virtual Time (GVT) [3] is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order. Further details on the API and working of WARPED is available in the literature [8].

3 The Ultra-large Scale Simulation Framework (USSF)

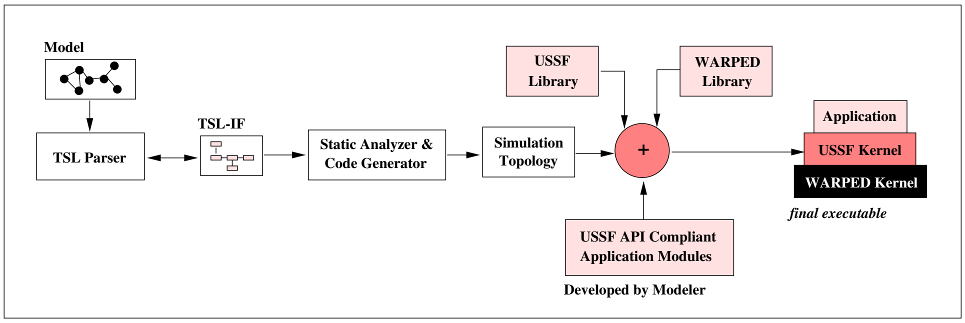


Fig. 1. Overview of USSF

The ultra-large scale simulation framework was developed to ease modeling and simulation of large communication networks. As shown in Figure 1, the primary input to the framework is the topology to be simulated. The syntax and semantics of the input topology is defined by the Topology Specification

Language (TSL) which provides simple and effective techniques to specify hierarchical topologies [9]. The topology is parsed into an object-oriented Intermediate Format (TSL-IF). Static analysis is performed on the intermediate format to extract and collate common object definitions. The analyzed TSL-IF is then used to generate an optimal simulatable network topology. This technique helps to isolate the static optimization techniques from the implementation language. The current implementation of USSF, in conjunction with WARPED and the generated code, is in C++. The generated topology includes code to instantiate the necessary user defined modules that provide descriptions for the components in the topology. The generated code is compiled along with the USSF library, WARPED library, and the application program modules to obtain the final executable. The following subsections describe the various components in detail.

3.1 Topology Specification Language (TSL)

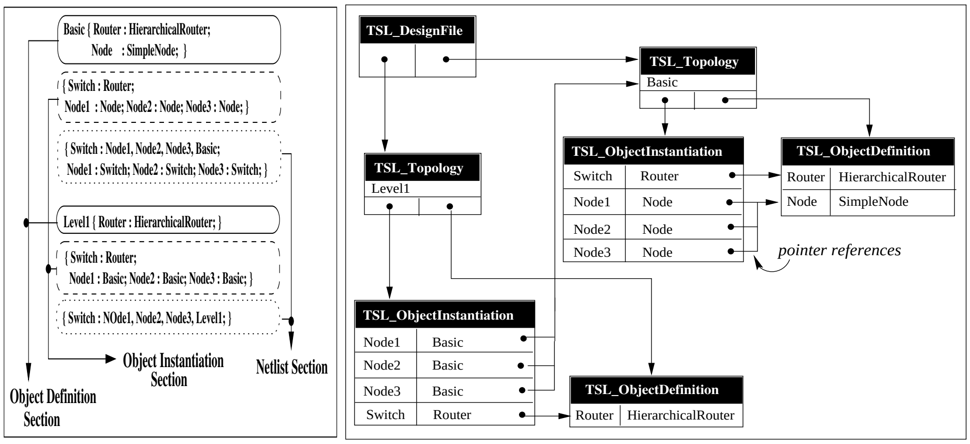
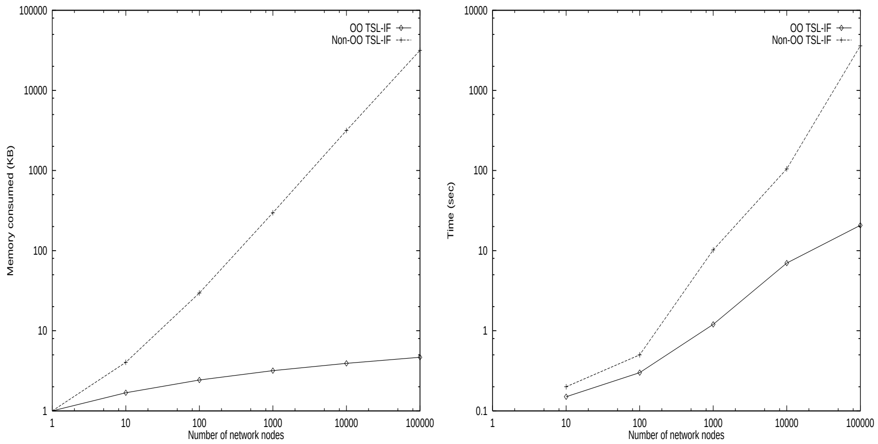


Fig. 2. Example of topology specification in TSL with its IF

The topology of the network to be simulated is provided to the framework in Topology Specification Language (TSL) syntax. The input topology configuration is parsed by a TSL-Parser into an object-oriented Intermediate Format (TSL-IF). The current implementation of the TSL-Parser is built using the Purdue Compiler Construction Tool Set (PCCTS) [6]. An example of a network topology specification in TSL along with its TSL-IF is shown in Figure 2. As illustrated in Figure 2, a TSL description consists of three main sections, namely; (i) the *object definition section* that contains the details of the modules to be used to simulate the topology; (ii) the *object instantiation section* that specifies the various nodes constituting the topology; (iii) the *netlist section* that deifines the interconnectivity between the various instantiated components. An optional *label* may be associated with each topology. As illustrated in Figure 2, an optional *label* may be associated with each topology that can be used to specify hierarchical topologies. Using this techique, a simple sub-net consisting of merely ten nodes can be recursively used to construct a network with five levels of hierarchy to specify a million (10^5) node network. The input topology configuration is parsed by a TSL-Parser into an object-oriented TSL Intermediate Format (TSL-IF). TSL-IF is designed to provide efficient access to related data from the various TSL sections. Each component of the grammar is represented by a corresponding TSL-IF node (or object). Every node in TSL-IF is built from the same basic building block; in other words every node is derived from the same base class. The intermediate format is composed by filling in the appropriate references to the different nodes generated by the parser. Since composition is achieved via base class references, each node can refer to another node or even a sub network. This mechanism provides an efficient data structure for representing and analyzing hierarchical networks.



Memory consumption **Time for parsing**

Fig. 3. Quantitative comparison between object-oriented (OO) and non-OO techniques used to construct TSL-IF

Prior to such an object-oriented implementation, references to sub networks were replaced by their definitions; in other words, “elaboration” was performed by cloning the sub networks. This technique worked fine for small networks but as the size increased, the time and memory requirements increased exponentially. A quantitative comparison of the the two techniques is shown in Figure 3. The data was collected on a workstation with dual Pentium II processors with 128 MB of main memory running Linux. The memory consumption of the parsing routines was monitored by overloading the `new` and `delete` calls of C++. As illustrated in the figure, the object-oriented representation dramatically out performed the traditional technique employed earlier. Analysis of topologies consisting of a million nodes was not feasible with the earlier technique as the system would run out of memory. Failure of the front end to meet the fundamental requirements proved a hurdle for further system development. Using the object-oriented (OO) representation, efficient analysis of topologies with tens of million nodes was possible. The OO representation also enabled "just-in-time" elaboration of the sub networks thereby reducing the time and resources consumed by the analysis and code-generation modules. This is an excellent example to highlight the importance of the object-oriented paradigm. The techniques not only improved efficiency of the system but were necessary to enable analysis of ultra-large topology specifications. The generation of configuration information in TSL format can be automated to generate different inter-network topologies [1]. Further details on TSL is available in the literature [9].

3.2 Static Analysis and Code-Generation Modules

The static analysis and code-generation modules of USSF play an important role in reducing the memory consumed during simulation. TSL-IF generated by the TSL parser forms the primary input. The static analyzer collates information on the modules repeated in the topology. Using this information duplicate object descriptions are identified and subsumed to a single entry in the various internal tables. Having a single entry rather than a few hundred thousands dramatically reduces their memory consumption. It also reduces the processing overheads of the data structures during simulation. The code-generator module uses the analyzed intermediate format to generate the final simulation topology. The generated code is compiled and linked with the USSF library, the WARPED library, and USSF API compliant user developed modules to obtain the final executable. Although the generated code is currently in C++, in compliance with the implementation of WARPED and USSF, the techniques employed are independent of the implementation languages.

3.3 USSF Kernel

The core functionality to enable simulation of ultra-large models is provided by the USSF kernel modules. The kernel is built on top of the WARPED simulator and presents a similar application program interface (API) to the model developer. The core of the USSF kernel is the USSF cluster. The USSF cluster module represents the basic building block of USSF simulations. A USSF cluster performs two important functions; namely (i) it acts as an LP to WARPED; and (ii) it acts as a cluster to the application programmer. As shown in Figure 4, the USSF cluster is used to group a number of LPs that use the same description together. A single copy of an user process is associated with different data and states to emulate its various instances. In order to enable this feature, the data associated with the different model descriptions need to be decoupled. The USSF API employs standard object-oriented techniques to insulate the model developer from such implementation intricacies. The USSF cluster uses file caches to maintain the data and states of the various processes. The caching helps in regulating the

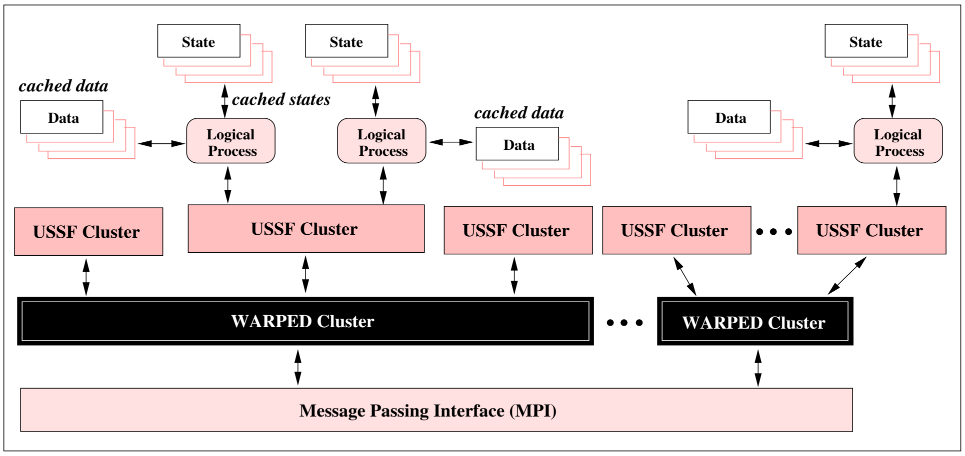


Fig. 4. Layout of an USSF Simulation

demands on main memory. Separate data and state caches are maintained to satisfy concurrent accesses to data and state spaces and to reduce cache misses. A simple Least-Recently-Used (LRU) cache replacement policy is used. Object oriented techniques have been used to decouple the various memory management routines from the core. This design not only provides a simple mechanism to substitute various memory management algorithms but also insulates the USSF cluster from their implementation details.

The USSF cluster is responsible for scheduling the various application processes associated with it. The cluster appropriately translates the calls made by the WARPED kernel into corresponding application process calls. It is also responsible for routing the various events generated by the application to the WARPED kernel. The WARPED kernel permits exchange of events between the USSF clusters. To enable exchange of events between the various user LPs, the framework cluster translates the source and destination of the various events to and from USSF cluster ids. In order for the USSF kernel to perform these activities, a table containing the necessary information is maintained by the kernel modules. The table is indexed using the unique process ids that need to be associated with each user LP. To reduce the number of entries in this table, a single entry is maintained for a group of LPs sharing a process description. The static analysis phase assigns contiguous ids to processes constructed using the same simulation objects. This fact is exploited to efficiently construct and maintain the table. The framework cluster also maintains a file based state queue in order to recover from rollbacks [3] that could occur in a Time Warp simulation. An simple incremental state saving mechanism with a fixed check-pointing interval is used for this purpose [2]. Since the state of the USSF cluster consists of merely the offsets into this file, the memory overheads of state saving are considerably reduced. A simple garbage collection mechanism triggered by the garbage collection routines in WARPED is used to prune the state queues. Access to the various methods in the framework kernel is provided via a set of simple application program interface (API). The API is illustrated in the following subsection.

3.4 USSF Application Program Interface

The USSF API closely mirrors the WARPED API [8]. This enables existing WARPED applications to exploit the features of the framework with very few modifications. The USSF Kernel presents an API to the application developer for building local processes based on Jefferson's original definition [3] of Time Warp. The API has been developed in C++ and the object oriented features of the language have been exploited to ensure it is simple and yet robust. The API plays a critical role in insulating the model developer from the intricacies involved with enabling ultra-large parallel simulations. The interface has been carefully designed to provide sufficient flexibility to the application developer and enable optimal system performance.

The basic functionality the kernel provides for modeling LPs are methods for sending and receiving events between the LPs and the ability to specify different types of LPs with unique definitions of state. On the other hand, the user is expected to override some of the kernel methods. The USSF kernel calls these methods at various times through out the simulation. Each method in this set has a specific function. The `initialize` method gets called on each LP before the simulation begins. This gives each LP a chance to perform any actions required for initialization. The method `finalize` is called after the simulation has ended. The method `executeProcess` of an LP is called by the USSF kernel whenever the LP has at least one event to process. The kernel calls `allocateState` and `allocateData` when it needs the LP to allocate a state or data on its behalf. Although it is the responsibility of the modeler to assign unique ids to each LP, the static analysis modules in the USSF perform this functionality. Interfaces for constructing application data, states, and events are also specified.

4 Experiments

This section attempts to illustrate the various experiments conducted using the ultra-large simulation framework. The network model used to conduct the experiments was a hierarchical network topology similar to the toplogy illustrated in Figure 2. Validation of the simulations were done by embedding sanity checks at various points in the models of the nodes constituting the network. The routers inter-connecting the nodes routed traffic by exploiting the fact that the topologies were hierarchical. The `basic` topology in Figure 2 was modified to consist of ten nodes and was recursively used in a hierarchical fashion in order to scale the network models to the required sizes. The models of the network components used in the specification were developed using the framework's API. Since the models were packet based, a module to convert packets into time stamped events and vice versa was employed to ease the interplay between the models and the framework. The TSL parser was used to analyze the hierarchical topologies and generate appropriate network models. The generated models were compiled and linked with USSF and WARPED libraries to obtain the final executable.

The models for the nodes used in the experiments generated events to be routed to a random destination using a normal distribution. The rate of traffic generation was based on random Poisson distributions. All of the experiments were performed using eight workstations inter-connected by fast Ethernet. Each workstation consisted of dual Pentium II processors with 128 MB of RAM (Random Access Memory) running Linux. Figure 5 presents the time taken for simulating the generated topologies. The simulation time for the smaller configurations is higher than expected due to the initial setup costs. In order to study the scalability issues of the framework the hierarchical topology consisting of hundred thousand nodes was used. Figure 6 presents the timing information obtained from this experiment. The times were measured using the standard Unix time command. As illustrated by the graph, the framework scales effectively utilizing the available resources. As illustrated by the experiments, the ultra-large scale simulation framework enables simulation of very large communication networks. The current implementation of USSF does not include a number of other proposed optimizations. The initial goal of the research was to enable such large-scale simulations. Studies are being conducted in order improve the performance of the framework.

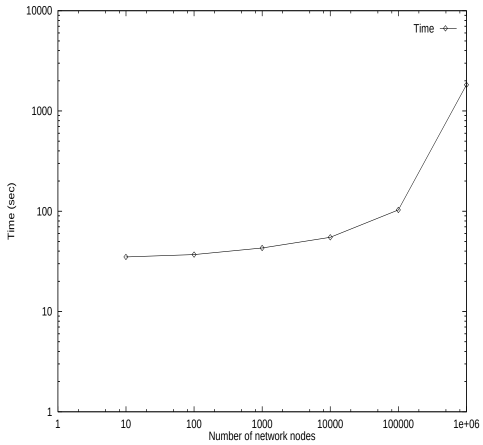


Fig. 5. Simulation Time

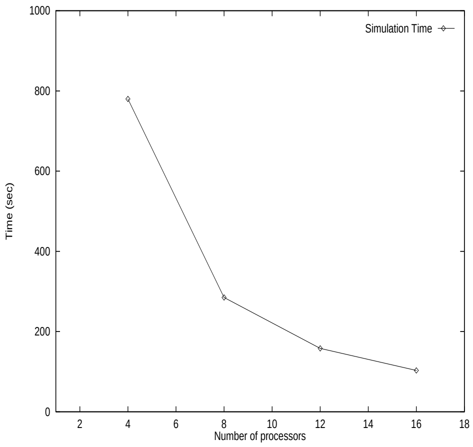


Fig. 6. Scalability Data

5 Conclusion

Computer and communication networks and their hardware/software components have steadily increased in size and complexity to meet the demands of modern applications. To ease modeling and analysis of such complex networks, parallel discrete event simulations have been employed. Simulation analysis with models built to reflect the ultra-large sizes of the networks is important inorder to study scalability and performance issues. The size and complexity of such parallel simulations requires the system to be carefully developed using standard design techniques such as object oriented design. Efficient and robust interfaces are necessary to ease application and simulation kernel developement. In order to insulate the application developer from such intricacies and to ease modeling and simulation of large networks, an object oriented frame work for simulation of ultra-large networks was developed. The issues involved in the design and development of the framework were presented. The experiments conducted using the framework were illustrated. The need for efficient object oriented design to enable the system was quantitatively highlighted. From the experimental results the capacity to perform such large simulations in resource restricted platforms was demonstrated. Currently, standardized techniques to improve performance of object oriented designs are being explored to further optimize the framework. The object oriented ultra-large scale simulation framework provides a convenient and effective means to model and study ultra-large communication networks of today and tomorrow.

References

1. E. Zegura, K. C., and Bhattacharjee, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (Apr. 1996).
2. Fujimoto, R. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (Oct. 1990), 30–53.
3. Jefferson, D. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July 1985), 405–425.
4. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM 21*, 7 (July 1978), 558–565.
5. Law, A. M., and McComas, M. G. Simulation software for communications networks: The state of the art. In *IEEE Communications Magazine* (Mar. 1994), pp. 44–50.
6. Parr, T. J. *Language Translation Using PCCTS and C++*. Automata Publishing Company, January 1997.
7. Paxson, V., and Floyd, S. Why we don't know how to simulate the internet. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997), pp. 44–50.
8. Radhakrishnan, R., Martin, D. E., Chetlur, M., Rao, D. M., and Wilsey, P. A. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, vol. LNCS 1505. Springer-Verlag, Dec. 1998, pp. 13–23.
9. Rao, D. M., Radhakrishnan, R., and Wilsey, P. A. FWNS: A Framework for Web-based Network Simulation. In *1999 International Conference On Web-Based Modelling & Simulation (WebSim'99)* (Jan. 1999), Society for Computer Simulation.
10. Robinson, S. Simulation model verficiation and validation: Increasing the users' confidence. In *Proceedings of the 1997 Winter Simulation Conference* (Sept. 1997).