TrGen: a Traffic Generation System for Interconnection Network Simulators

F.J. Ridruejo, A. Gonzalez, J. Miguel-Alonso
Department of Computer Architecture and Technology, The University of the Basque Country
P.O. Box 649, 20080 San Sebastian – Spain

Abstract

In this paper we introduce TrGen, a traffic generation environment specifically designed to interact with simulators of interconnection networks for parallel and distributed systems. This environment is able to generate synthetic traffic, and actual traffic taken from traces (of previous program runs). It can also cooperate with complete-system simulators to assemble a complete execution-driven simulation arrangement.

1. Introduction

Design of interconnection networks is a topic of great interest in the field of parallel computer architecture. Evaluations of architectural proposals need to be carried out during all design stages, including the most preliminary ones. Simple, functional simulators help us assessing routing algorithms, deadlock avoidance mechanisms, fault-tolerance, and so on. These simulators do not incorporate all the details required in an actual, hardware-implemented system; however, the most relevant aspects of the design are there, allowing us to check the viability of a proposal—or the lack of it. In subsequent stages, more detailed simulators (such as SICOSYS [12]) or even hardware prototypes can be used to refine the design.

As important as a good model of the interconnection subsystem is a good characterization of the traffic it will deal with. In the same way processors are designed taking into account the programs that will run on them, interconnection subsystems must be evaluated using workloads (interchanges of packets) as realistic as possible. Ideally, this traffic should reflect the actual way parallel and distributed applications communicate [6]. How should this traffic be? That is a non-trivial question. Interconnection networks (IN) may work with very different traffic patterns, depending on the kind of system it forms part of. Some examples:

- An IN for a small-size SMP (symmetric multiprocessor)
- An IN for a large-scale CC-NUMA (cache coherent non uniform memory access) multiprocessor
- An IN for an even larger MPP (massively parallel processor)
- An IN for distributed applications based on web services

In fact, market response has been different for each of these needs. We can even go further: what we demand from the network is different when running master-slave applications with infrequent interchange of long messages and when running a fine-grained scientific application where messages are short but interchanged very often. In the first case we need a high throughput, while in the second latency is the main constraint.

For all these reasons, any environment for simulation and evaluation of interconnection network designs must have a good method to generate traffic. Many techniques could be used for that purpose:

1. Synthetic traffic patterns, such as uniform traffic, matrix transpose, hot-spot, etc. They are very simple to implement in a simulator, and in many cases they emulate the behavior of typical CS&E (computational science & engineering) applications. Very often they are criticized because they are not "representative enough" of actual workloads; however, in our experience, they are extremely useful during initial design stages.
2. Traces from actual applications. The target application is run on an existing computer, and those traces are used to "feed" the simulation of a different network. This technique is restricted by the size of the computer used to obtain the traces.
3. Execution-driven simulation. The whole system, compute nodes and interconnection subsystem, is

This work has been done with the support of the Ministerio de Educación y Ciencia, Spain (TIN2004-07440-C02-02) and of the Diputación Foral de Gipuzkoa (OF-846/2004).

Contact author: miguel@si.ehu.es.

simulated. Nodes actually run the target application, so they inject and consume traffic interchanged via the network. This is, obviously, the most realistic way of doing the simulation. However, it can be prohibitively slow, and the size of the system to simulate has to be necessarily small.

An additional question may arise: which application is *representative*? Some parallel computers are built with a single application in mind (maybe a limited set of related applications). However, from a vendor's position, it is more interesting to build general-purpose computers, able to deal not only with CS&E applications, but also with commercial applications such as OLTP (on line transaction processing). We know that, from the point of view of the "pressure" exerted over the network, commercial applications are very different from CS&E applications.

In this paper we introduce TrGen, an environment to generate traffic for interconnection networks. Using TrGen, researchers can focus on the modeling and simulation of networks, using this tool to test their designs under a variety of traffic sources. Section 2 describes the general structure of TrGen, and the way it communicates with simulators. The next two sections describe some implementation details of TrGen for synthetic and trace-based traffic (Section 3) and for execution-driven simulation (Section 4). Section 5 reviews some related work. Finally, Section 6 contains the conclusions of this paper, and indicates some lines of future work.

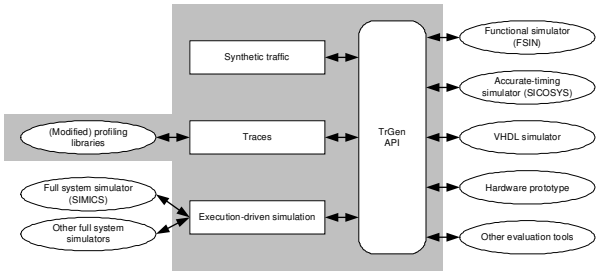

Fig. 1. Model of TrGen. A common API allows for the connection of different simulators with a variety of traffic sources.

2. TrGen Design and API

2.1. Design

The design of TrGen follows the scheme of Fig. 1. At the right side of the picture we have represented different simulators for interconnection networks; the first two are concrete, existing systems (FSIN and SICOSYS), while the remaining three are generic. At the left we represent the sources of traffic: totally synthetic, traces (obtained in advance) and execution-driven traffic.

Independently of the characteristics of the traffic source, a common API allows the simulators to interchange packets with it. This API allows for the completion of three basic operations:

1. Initialization. Selection of traffic pattern, and of its parameters. These parameters can vary substantially from case to case: a trace file to open, a mean of a random number generation function, an IP address of a traffic server connected to a full system simulator, and so on.
2. Requests of traffic to be injected into the network nodes.
3. Notifications of arrivals of packets from the network. This part is crucial when using execution-driven simulation, an also when traffic is reactive: the reception of a packet triggers the generation of new ones.

2.2. API

Current version of TrGen's API is very simple, so in the future it will be expanded to add flexibility. Right now, these are the available functions:

```
void source_new(source_t * s);
void source_init(source_t s, long clock,
source_e type, ...);
```

Functions source_new() and source_init() create and initialize a traffic source. Relevant parameters are the traffic source s and the type of traffic to be generated, type. Currently available traffic types are STREAMED for synthetic traffic, and FILED for trace-based traffic. Function source_init() requires a collection of additional parameters, whose number and nature depend on the traffic type.

```
t_packet * source_next(source_t s, long *
npackets);
```

This function returns a vector of packets to be injected in the current simulation cycle, as well as the length of that vector. Parameter s selects the traffic source.

```
void source_notify(source_t s, t_packet
packet);
```

This function allows the network (simulator) to inform the source s about the delivery of a packet after

it has traversed the network. This function is important when there are causal relationships between packets.

```
bool_t source_finished(source_t s);
```

This function tells us whether or not the source s has more traffic to inject. A generation finishes its work when a certain condition that depends on the traffic type is reached. Examples are: end of trace file, maximum number of generated packets reached, maximum number of simulation cycles reached, etc.

The current API is biased towards time-driven simulation. In the future we will expand it to ease interoperability with event-driven simulators.

3. Synthetic and Trace-Based Traffic

3.1 Synthetic Traffic

This class of traffic is the easiest to generate. It comes characterized by three distributions: temporal, spatial and packet-size.

1. The temporal distribution determines the intervals between packet generations, and their correlations.
2. The spatial distribution determines the packet destination nodes.
3. The packet-size distribution determines the size of the generated packets.

As an example, in the initial stages of the evaluation of a network design, it is common to use synthetic traffic with a Bernoulli temporal distribution, a uniform spatial distribution (or another one with a very regular pattern), and a constant packet size. TrGen allows, though, the specification of very diverse traffic sources, to simulate a variety of applications. The right selection of distributions (or parameters for a given distribution) allows the evaluation of different aspects of the network.

TrGen uses the "stream" concept to define a source of synthetic traffic. Each stream has a source, a spatial distribution, a temporal distribution and a packet-size distribution. Each source node may have several streams associated to it, to fine-tune the traffic behavior. Each distribution can be parameterized, and the collection of available distributions is extensible. In the current version of TrGen, these are the available ones:

Temporal:
- Bernoulli
- Constant bursts
- Markov

Spatial:
- Uniform
- Distributed
- Zipf
- Hotspot
- Constant: transpose, butterfly, perfect-shuffle, inverse, etc.

Packet-size:
- Constant
- Uniform
- Polynomial

These distributions have been frequently described and discussed in the literature. For more information, readers can check [2,3].

3.2. Traces

Synthetic sources provide very useful insights into a network's potential. However, obtained performance metrics can be unrealistic: actual applications do not produce random traffic patterns, so it is difficult to model them with the sort of synthetic sources described above. Application processes often communicate following a precise order, at precise moments. Additionally, it is common to find applications in which the reception of a message triggers the delivery of a new one. Therefore, the traffic pattern includes causal relationships between the reception and the sending of messages between processes. With TrGen it is possible to use traces obtained from the execution of actual applications to perform simulations that closely follows the traffic patterns explicitly defined in the traces, in terms of spatial distributions, packet sizes and causal relationships.

We use a modified version of MPICH (one of the most popular implementations of MPI [5]) to obtain trace files usable with TrGen. MPICH includes an easy-to-use mechanism to obtain trace files from running applications. However, these traces are not useful for our purposes because collective operations (such as barriers, broadcasts, reductions, etc.) appear as such in the trace files, without reflecting the actual interchange of packets necessary to implement those operations in the network—for networks without native support for collective operations. Internally, MPICH implements collective operations with point-to-point operations (if no better alternative is available). Our changes in MPICH consist of making those point-to-point operations visible, registering them in the trace files instead of the corresponding collective operation.

Trace files are slightly pre-processed before using them with TrGen. Only a few, relevant fields are selected (event type, source node, destination node, message size) and organized in a format more suitable for TrGen. It would be possible to use this format to feed simulations with traces obtained from sources different to MPICH, a network sniffer being a good example, just building the right pre-processor.

Part of the pre-processing is the elimination of event timestamps, however keeping temporal order and causal relationships. This is because timestamps strongly depend on node and network characteristics. Without timing information, when we simulate a network using a trace file, we submit the network to the maximum pressure, as if infinitely fast processors were using it, and we force the network to be the running application's bottleneck—and measure how well it performs. We plan to enhance TrGen to take into account node (CPU) speed, scaling the temporal distance between events to reflect this speed.

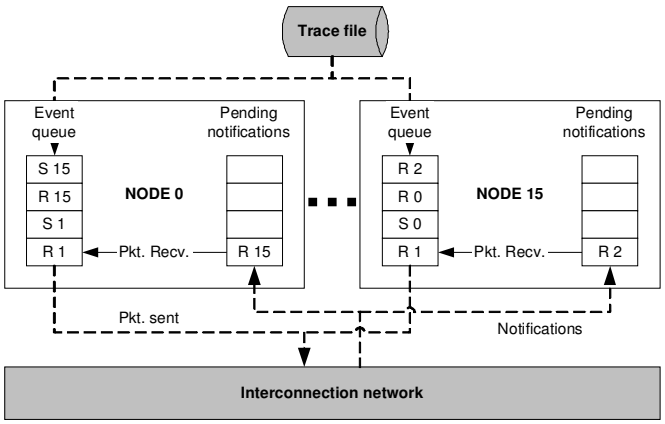

Fig. 2. Data structures used to keep event causality in trace-based traffic sources. Node 0 is waiting for the reception of a packet from node 1, that will trigger sending a new, response packet. After that, node 0 will wait for the reception of a packet from node 15; however, this packet has already been received, because it is in the list of pending notifications, so TrGen will deliver immediately the message sent to node 15. Situation is similar at node 15: events "R 1" and "R 0" will stop the generation of messages at that node, until the corresponding notifications arrive, while event "R 2" will be consumed immediately.

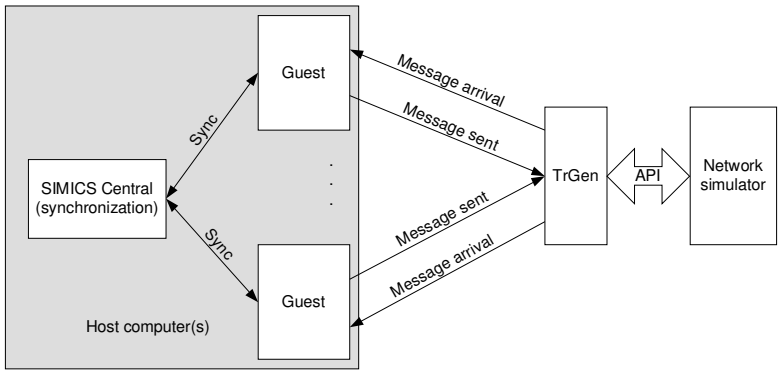

Fig. 3. Interactions of the different elements involved in an execution-driven simulation: compute nodes, SIMICS central, TrGen and an interconnection network simulator.

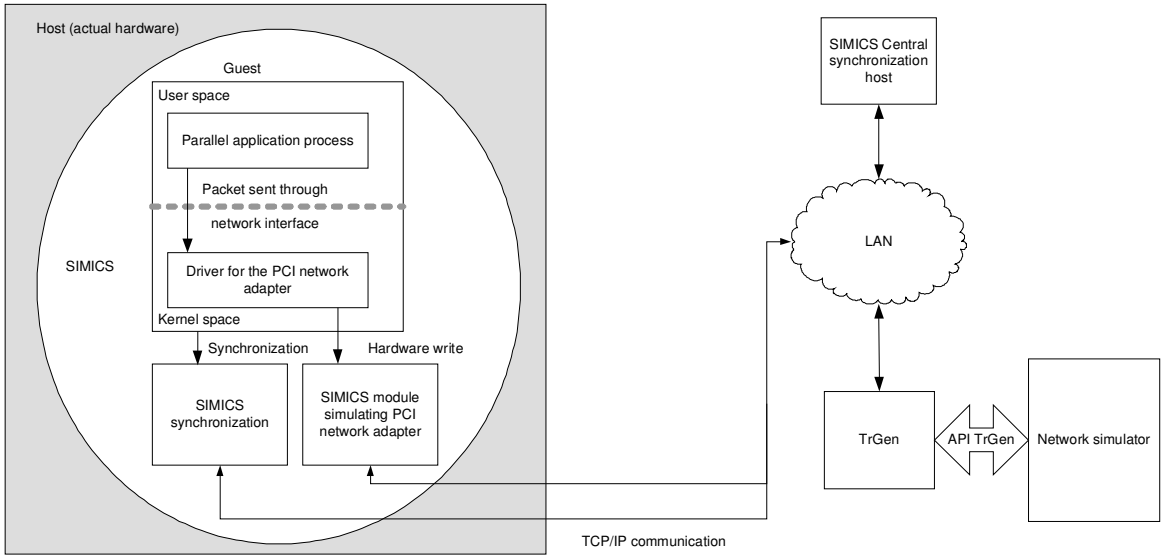

Fig. 4. Structure of message interchange, from their creation at a parallel application (running in a simulated compute node) until their arrival to the computer running the interconnection network simulator.

Figure 2 represents how trace-based traffic works. Each node of the simulated applications has an event queue. This queue is fed from the trace file. A packet is sent to the network when an "S" (send) event is in the queue's head. If a "R" (receive) event is in the head, it is necessary to access the pending notifications queue, to check if the expected event has happened already; otherwise, processing of events is blocked until the network notifies the awaited reception. The pending notifications queue at each node, thus, stores reception events that arrive before the application requests them, and it is crucial to keep event causality.

Traces obtained from one system are often used to evaluate via simulation the performance potential of other system. However, this approach has some drawbacks. In the context of interconnection network design and evaluation, traces obtained with the same collection of nodes running a parallel application with IN "A" and IN "B" may be different, because properties of "A" and "B" are different and those properties have an influence on the way nodes interchange messages. For this reason, performance results obtained with traces may not be accurate [4].

4. Execution-Driven Traffic Generation

4.1 General Schema

An execution-driven simulation requires the detailed simulation of the behavior of compute nodes as well as the interconnection network. The desired application will run on top of this simulated system. Simulation will be as realistic as detailed is the description of each component. The price to pay is an enormous slowdown: execution time will be several orders of magnitude slower than that of a real system.

Currently, the simulation of the compute nodes that run parallel applications is carried out using Virtutech's SIMICS [14]. It allows the detailed simulation of a range of complete hardware systems on top of a consumer PC. For example, we could simulate a 4-way Sparc multiprocessor running the Solaris operating system (guest system) on top of an Intel-based PC with the Windows operating system (host system). For the simulation of the interconnection network we can select the simulator of our choice, using the TrGen interface.

We are using SIMICS on a collection of PCs running the Linux operating system to simulate compute nodes based on x86 processors with 256 MB of RAM. Each of these nodes runs its part of a parallel application that generates messages for the remaining nodes. We can have several host PCs, each one simulating several guest PCs to reach a network as large as our resources allow.

SIMICS includes a mechanism called SIMICS Central to synchronize all the guest nodes. In fact, it behaves as a simulated LAN, so it allows guests to communicate. We keep the synchronization function of SIMICS Central, but use a separate system (an interconnection network simulator connected via TrGen) for communication purposes: messages generated by a guest are intercepted and sent to TrGen which, in turn, handles them to the network simulator. Also, messages generated in the simulator are injected (via TrGen) to the destination guest. This interchange of messages is depicted in Figure 3.

As we said before, guests (compute nodes) are SIMICS virtual machines that run as normal processes on a host computer. SIMICS Central acts as an Ethernet switch that

also provides services such as ARP, BOOTP and DHCP. Moreover, SIMICS Central keeps all the guest nodes synchronized, thus allowing the simulation to run in a deterministic way.

4.2. Implementation

A fairly complex mechanism has been developed to redirect messages created by a running process (part of a parallel application running in simulated compute nodes) to the simulator of the interconnection network, and then back to the destination process. The two most important pieces are:

- A module for SIMICS[1] that implements (simulates) a PCI network adapter. This module manages all the messages exchanged by the parallel application running on the compute nodes. In fact, it is the interface with the (simulated) interconnection network. It has been written in C.
- A Linux kernel driver that allows applications to access the new network adapter. We have written this driver following the guidelines provided in [1, 13]. It is the final responsible of writing/reading to/from the new network adapter, when applications request these operations via system calls.

As we stated before, SIMICS Central is another important piece. Applications do not use the other PCI network adapter (the standard one, connected to Central) for traffic exchange, because the provided functionality (that of an Ethernet network) is not valid for our purposes. However, the synchronization of all the compute nodes is a requirement, and Central provides it.

In our current setup, each host computer runs one or several instances of SIMICS, each one simulating a guest computer. Our choice of guest is a PC with 256 MB of RAM, an Intel x86 processor, a Voodoo 3 graphic adapter, and an IDE hard disk. For connectivity, a guest hast two Ethernet adapters connected to the PCI bus: one connected to SIMICS Central, the other one to TrGen. Obviously, all this hardware is simulated using SIMICS modules[2]. Each simulated computer runs under the RedHat 7.3 Linux operating system.

When all guests are installed, configured and ready (with the O.S. running), we can install the parallel application of our choice, for example, any of the NAS Parallel Benchmarks [7]. When a process participating in a parallel application sends a MPI message to other process (running in a different guest) it uses the new network adapter. The driver that manages this device (running in kernel space) gets that message, and sends it to the network via writing in the registers and memory of the network adapter. The adapter then redirects the message to TrGen.

In a similar way, when the module that simulates the new network adapter gets a message from TrGen, injects it in the appropriate guest, again writing in the device's memory and registers. It also triggers interrupts that are managed by de driver, which then delivers the message to the appropriate receiver process.

Communication between the network adapter and TrGen can be performed in many different ways. We use a collection of TCP connections. TrGen acts as a bridge with the network simulator of our choice. This simulator must be able to incorporate all the mechanisms required from an interconnection network for a cluster or a multicomputer.

Figure 4 represents all the interchange of messages (simulated and real) involved in the communication between compute nodes.

5. Related Work

Most network simulators include mechanisms for traffic generation. Interconnection network simulators such as SICOSYS [12] include internal mechanism for the generation of synthetic traffic. It is also possible to run this simulator jointly with RSIM [11] for execution-driven simulations, although this is done using some ad-hoc glue software. More general network simulators like OPNET [9] include sophisticated mechanisms for synthetic traffic generation, including the generation of traffic that accurately emulates typical Internet applications. Trace files have been extensively used for simulation; for example, in [10] a mechanism to integrate trace-driven simulations into NS [8] is described.

As far as we know, in all cases traffic generation is performed internally, or attached to the simulator using an ad-hoc, non-standard API. The main advantage of TrGen is the definition of a unified API to access a set of very different mechanisms for feeding simulators. This feature allows researchers to focus in modeling its network, and also sets a fair mechanism to compare results obtained from different simulators.

6. Conclusions and Future Work

TrGen is a live project, still being actively developed. In this article we have stated that we are considering the integration of new functionality, and the improvement of those already implemented. Currently, the generation of synthetic traffic (with the above-mentioned characteristics) is fully functional. Capture, adaptation and generation of traffic based on traces is also functional. The generation of actual traffic using SIMICS is under test.

We have tested the interoperability of TrGen with FSIN, a functional simulator developed in-house. Results are very satisfactory: TrGen greatly improves the spectrum of experiments that can be performed with FSIN,

[1] SIMICS allows the implementation of new devices via modules that extends the simulator's capabilities.

[2] All of them included in the SIMICS package, except for the network adapter that interacts with TrGen.

without a relevant increase in used resources (memory, CPU time)—compared with the reduced-functionality traffic generation routines built in FSIN.

Currently, efforts of the development team focus in completing the parts of the design that are not ready yet, and in thoroughly testing the system. These are not, however, the only lines for future work.

Traffic generation offers many possibilities, and TrGen is still in its infancy. We plan to increase the choice of available sources of synthetic traffic: spatial distributions with memory, reactive traffic patterns, traffic types for particular applications (such as FTP, WWW, POP, etc.), and spatial distributions based on node distance.

Finally, in order to test the usefulness of TrGen, we need to integrate it with additional simulators. A short-term plan is to integrate it with SICOSYS. Also, we plan to build TrGen-Lite, a lightweight version of TrGen (with reduced functionality) for particular simulation environments were resource consumption has to be kept as low as possible.

References

[1] Donald Becker, Linux network drivers, in Scyld's corporate web: http://www.scyld.com

[2] Jose Duato, Sudhakar Yalamanchili, Lionel Ni. Interconnection Networks: An Engineering Approach. Morgan Kaufmann, 2002.

[3] Itamar Elhanany (Ed.) Fabric Benchmarking Traffic Models Rev. 1.0. Network Processing Forum, 2003. Available at http://www.npforum.org/techinfo/BM_Fabric_TrafficIA.pdf

[4] S. Goldschmidt and J. Hennessy. "The accuracy of trace-driven simulation of multiprocessors". In ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, pages 146-- 157, May 1993.

[5] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Available at http://www-unix.mcs.anl.gov/mpi/standard.html

[6] J. Miguel, A. Arruabarrena, R. Beivide y J.A. Gregorio. "Assessing the performance of the new IBM SP2 communication subsystem". IEEE Parallel and Distributed Technology, Vol. 4, nº 4 (1996), 12—22.

[7] NASA. The NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB.

[8] The Network Simulator ns-2. Available at http://www.isi.edu/nsnam/ns/

[9] OPNET Technologies, Inc. corporate web page, available at http://www.opnet.com

[10] Philippe Owezarski And Nicolas Larrieu. "A trace based method for realistic simulation". IEEE International Conference on Communications (ICC'2004), QoS and performance modeling symposium, Paris, France, 20-24 June 2004

[11] V.S. Pai, P. Ranganathan, and S.V.Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. IEEE TCCA New., Oct. 1997.

[12] V. Puente, J.A. Gregorio, R.Beivide. SICOSYS: An Integrated Framework for studying Interconnection Network in Multiprocessor Systems, Proceedings of the IEEE 10th Euromicro Workshop on Parallel and Distributed Processing. Gran Canaria, Spain. January 2002.

[13] Alessandro Rubini, Jonathan Corbet. Linux Device Drivers 2nd Edition. O'Reilly.

[14] Virtutech, Inc. Simics web page (http://www.virtutech.se/simics/simics.html), inside Virtutech, Inc corporate web. (http://www.virtutech.se/)