

# Fast Simulations of Large-Scale Highly Congested Systems

Lee W. Schruben<sup>†</sup>

Theresa M. Roeder

Department of Industrial Engineering and Operations Research

University of California, Berkeley

4135 Etcheverry Hall

Berkeley, CA 94720

*schruben@ieor.berkeley.edu*

Focusing on resource cycles, the authors developed a semiconductor wafer factory (fab) simulation that executed more than an order of magnitude faster than a job-tracing simulation previously in use. The authors summarize the methodologies used and conclude that the differences in execution speeds are due to the fundamental differences in using an event graph paradigm to model the discrete event system dynamics instead of the more popular process flow paradigm that is used by almost all commercial simulation packages. However, the execution speed of a resource-driven model is insensitive to system congestion, whereas a job-driven model slows dramatically (or halts) as the system becomes heavily loaded. The authors conclude that a resource-driven approach using event scheduling logic offers the best approach to modeling very large-scale highly congested systems such as those found in communication, transportation, and unit-manufacturing operations.

**Keywords:** Resource driven, job driven, modeling taxonomy, process interaction, event scheduling, discrete event simulation

## 1. Background

When developing simulation models of discrete event systems, it is useful to classify entities as being either resident entities or transient entities. Resident entities remain part of the system for long intervals of time, whereas transient entities enter into and depart from the system with relative frequency. In a factory, a resident entity might be a machine; a transient entity might be a part. Depending on the level of detail desired and objectives of the simulation study, a factory worker might be regarded as a transient entity in one model and a resident entity in another.

In describing a particular aspect of the dynamic behavior of a system, it is often useful to focus on the cycles of the resident entities. For example, we might describe the busy-idle cycles of machines or workers. Alternatively, we might focus on the paths along which transient entities flow as they pass through the system (e.g., parts moving through a factory). Transient entity system descriptions tend to be more detailed (and more informative) than resident entity descriptions. Each type of modeling has advantages: modeling resident entity cycles tends to be easy and efficient, while modeling the details of transient entity flow gives

more information. Typically, a mixture of both viewpoints is used, but one or the other predominates.

In highly congested systems, in which there are relatively few resident entities and a great many transient entities, it is usually more efficient to study the cycles of the resident entities. Examples include semiconductor factories with thousands of wafers, communication systems with millions of messages, and transportation systems with tens of thousands of vehicles. In simulating such systems, the cycles of resident entities might be described by the values of only a few variables, while the flow of transient entities might require many variables to describe. This is a great disadvantage of job-driven simulations because, whenever the simulation becomes highly congested, the simulation's memory footprint becomes larger, and its execution slows, or it may stop running altogether.

On the other hand, systems in which there are only a few transient entities and many resident entities (a construction project or an airline maintenance facility) may be efficiently studied by examining the flow of transient entities. The advantage of using a job-driven event graph model over a resource-driven model is that the detailed experiences of individual jobs can be easily tracked. This is important, say, in modeling low-volume, high-mix manufacturing systems.

While different terminology might be used, the classification of entities in a system as resident and transient is

<sup>†</sup>To whom all correspondence should be addressed.

common to many discrete event system-modeling methodologies. Some examples are summarized in Table 1.

Conventional literature describes three simulation worldviews. (see, e.g., Derrick et al. [1]). These worldviews can be seen as different ways of implementing resource-driven and job-driven (and mixtures of the two) simulations. The dichotomy between resident entity cycle modeling and transient entity flow modeling is a simpler yet more meaningful highest level for a modeling taxonomy for discrete event simulation than the three classical worldviews.

The most prevalent traditional worldview is that of process interaction. The name *process interaction* derives from the focus on modeling how different processes in the system being simulated interact with each other. For example, loading and unloading parts into/from a machine and repairing a machine when it fails are two separate processes. In this case, the production processes need the machine and an operator—either of which may be busy with a repair process. The processes are interacting through shared resources. Process interaction is by far the most popular way of implementing a job-driven simulation. The jobs are the active system entities that “seize” available system resources when they need them. This approach typically requires that every step in the processing flow path of every job in the system be explicitly represented. Records of all jobs in the system are created and maintained. The jobs move through their processing steps (often represented by a block flow diagram), seizing available system resources whenever they are needed. These job-driven models are convenient when predicting system performance and fast simulation execution speed are not as important as detailed system animation. The biggest advantage to using this approach is that there often is a direct mapping from simulation logic to animation—the code focuses on describing the things in the system that move.

Figure 1 shows the GPSS blocks for a single-server queuing system. Jobs are GENERATED with a certain interarrival time  $T_A$ . They then join the QUEUE for resource 1. If resource 1 is available, they SEIZE the resource, and DEPART the queue (computing waiting time statistics). After a service time  $T_S$ , the job releases the resource and is TERMINATED.

Another classical worldview is the event scheduling worldview. The three elements of a discrete event system model are the state variables, the events that change the values of these state variables, and the relationships between the events (one event causing another to occur). The event scheduling worldview focuses on these events. The simulation proceeds by executing the next event, which, in turn, may schedule further events or cancel events that had been already scheduled.

Event graphs are an easy way of implementing an event scheduling-based simulation [2]. In the graph, events are represented as vertices (nodes), and the relationships between events are represented as edges (arrows) connecting pairs of event vertices. Time sometimes elapses between

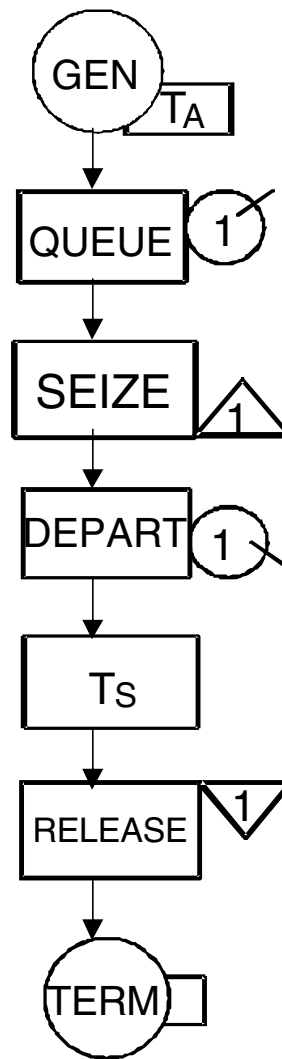


Figure 1. Traditional GPSS block diagram for a G/G/1 queue

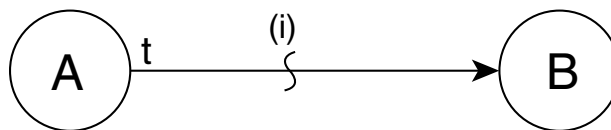


Figure 2. Basic component of an event graph

the occurrences of events. Figure 2 shows the basic structure of the graph. It states that if condition (i) is true at the instant event A occurs, then event B will be scheduled to occur  $t$  time units later.

The third classical worldview is the activity scanning worldview. Much as the process interaction worldview is closely related to job-driven simulations, activity scanning

**Table 1.** Summary of terminology used by different applications

Technology	Resources	Jobs
Queuing theory	Servers	Customers
Petri nets	Tokens	Tokens
Event graphs	Resident entities	Transient entities
Arena	Resources	Entities
Simscript	Permanent entities	Temporary entities
GPSS	Facilities and storages	Transactions

is often used to develop a pure resource-driven model. Timed stochastic Petri nets are a popular graphical implementation of activity scanning that can be useful in simulating certain types of resident entity cycles [3]. Timed stochastic Petri nets are a special case of the event graphs discussed in the previous paragraph, but their usefulness is limited to a subset of the models easily simulated by event graphs; see Schruben [4] for an algorithm to automate converting any Petri net into a more compact event graph. (There is an interesting confusion of terminology here due to the term *event graph* having been also used to denote a very special class of a Petri net—a construct completely different from the one referred to above.)

In activity scanning/event scheduling-based models, the individual transient entities (jobs) in the system are passive, and they are “moved” or “processed” by the system’s resources. Event scheduling can be used to implement both job-driven and resource-driven simulations. For resource-driven simulations, rather than maintaining a record of every job in the system, only integer counts of the numbers of jobs of particular types and different stages of processing or in different states are necessary. The system’s state is described by the availability of resources (also integers) and these job counts. Thus, all the state variables in a resource-driven simulation are nonnegative integers. The state changes for each event are difference equations that increase or decrease one or more state variables by integer amounts. Very large and highly congested queuing networks (with any number of jobs of any number of types at any countable discrete stage of processing) can be modeled this way with a relatively small, finite set of integers. The memory footprint of the simulation is proportional to the number of resources in the system, not the number of jobs. As a consequence, the simulation is almost insensitive to system congestion.

When using event scheduling to implement a job-driven simulation, the integer counts of the jobs at a given step are replaced by lists of these jobs. As in the process interaction approach, the memory footprint of the simulation will increase with system congestion.

## 2. Previous Results

In Roeder et al. [5], we presented the results of a study done of simulations of a semiconductor wafer fab. The

purpose of the research was to investigate the differences—advantages and disadvantages—of resource-driven and job-driven simulations with a real large-scale simulation model that is currently being used.

### 2.1 System Description

The fab modeled produced more than 5 part types using more than 80 different tool types. Each tool type had a varying number of (functionally identical) tools available for processing. The tool included serial processing, batching, and stepper tools. Each tool type was subject to a set of preventive maintenances (PMs) and failures that occurred according to certain probability distributions. Test and rework wafers visited the tools periodically. All tools required load and unload times, and stepper tools also had setup-dependent setup times. Wafer lots were queued based on critical ratio ranking, and the first lot in queue was selected to be processed next. Proprietary processing rules were used in processing parts on the stepper tools.

Lots revisited tools repeatedly and visited subroutes with certain frequencies (e.g., every 10th job to get to step 4 executed step 4; all others moved directly to step 5). The basic route followed by the parts consisted of more than 500 processing steps.

Operators and generic tools were not modeled to facilitate the process. To be able to compare the output of the two simulations accurately, the resource-driven simulation actually simulated the job-driven simulation, not the fab itself. This was done to prevent differences in output caused by different approaches to modeling the same aspects of the fab.

### 2.2 Comparison

The job-driven simulation is coded in AutoSched AP (ASAP), a commercial software package by AutoSimulations, Inc. [6]. It is heavily used in the semiconductor industry. Data and options are specified in spreadsheet files and are processed by the software. Users also need to code customized functionality in C++. ASAP is, in our context, a process interaction-based software.

The resource-driven simulation developed at University of California, Berkeley was implemented using the software package SIGMA for the underlying simulation engine and to generate C source code. Additional coding

was done in C to mimic the functionality of the job-driven simulation. It is easy to implement both resource- and job-driven simulations using SIGMA (see, e.g., Schruben and Schruben [7]). The resulting simulation code itself is a simple executable file. The ASAP input files were imported into a Microsoft Access database, which, along with Microsoft Excel, was also used to create plain text input files. In our context, SIGMA uses an event scheduling algorithm to model system dynamics. A high-level version of the event graph used can be found in Figure 3.

There were two main modeling differences highlighted in the previous paper [7]. They showed practical advantages and disadvantages to pure resource-driven and job-driven models and argue for a combination of the two. The first difference was the inability of the pure resource-driven model to accurately model job-based queuing disciplines. For example, to queue lots based on their critical ratios, the system needs to know how long the job has been in the system. Since this information is not available, however, it is not possible to calculate the critical ratio or to sort lots by it. Even more fundamentally, however, the pure resource-driven model cannot represent first-in, first-out (FIFO) queues with multiple job types. No information is kept on individual jobs, and so it is not known what job type arrived to the queue first. Since we cannot, in this case, implement the queuing disciplines used in the job-driven simulation, we (arbitrarily) decided to process the job with the largest number waiting in queue next. We shall see later that the model can be easily enriched to include job information; however, the expected penalty of significantly increased runtimes was not observed.

A second big difference is that it is very difficult to model PMs/failures accurately in the job-driven approach. Because system resources are passive, they must be seized by jobs—or by the next PM/failure. As a consequence, PMs/failures are treated as high-priority jobs. This increases the system congestion (one tool type experiences up to 90 different PMs/failures). It also makes it difficult to treat failures properly, so this simulation treats failures as PMs. (The failure will “wait” to occur until the current lot has completed processing.) In addition, it is possible to queue PMs and failures: another PM of type  $i$  might join the tool’s queue while the tool is still being maintained by PM  $i$ .

A third difference between job-driven and resource-driven modeling is that it is very easy to accurately model simultaneous resource usage in a resource-driven model, whereas this is often difficult using a job-driven approach. In fact, the common problem in job-driven models of resource deadlock (an artifact of the simulation model) simply does not occur in an event graph model whether modeling job flow, resource cycles, or a combination of both. See Venkatesh [8] for a discussion and partial solution of this problem with process interaction modeling.

The major result of the study was the large speedup in execution time. Table 2 shows execution speeds for 2

years of simulated data on a Dell Dimension Pentium 4 1.7-GHz desktop with 512 MB of RAM. (The processing times include reading large data files into the simulation and generating output reports, so the speedup in actual simulation execution was even greater.) The speedup for the resource-driven model is significant, while the weekly output statistics (e.g., utilization percentage, throughput, time spent in PM/repair) were almost completely comparable.

### 3. Recent Results

The previous research project had demonstrated advantages and disadvantages of resource-driven and job-driven models and had shown that the resource-driven model was significantly faster, but at a loss of modeling ability and available output. The next step was to create hybrid resource-driven/job-driven models. We created a purely job-driven model using the event scheduling software to see what the worst-case scenario runtimes were. This was very easy using specialized functions in SIGMA to put and get job information from the queues.

Surprisingly, in the worst case, the runtimes for the job-driven model were only slightly longer—roughly 7.5 minutes, compared to the resource-driven runtime of a little under 6 minutes. Some minor refinements are necessary in the implementation (e.g., implementing dedication constraints on the stepper tools), but they should not account for more than a few seconds’ difference in execution time.

The result has led us to examine more closely whether the differences in runtime were truly due to differences between resource-driven and job-driven models, as originally thought, or whether they were due more to the actual implementation of these paradigms.

#### 3.1 Process Interaction versus Event Scheduling

Traditionally, the terms *job-driven simulation* and *process interaction worldview* have been used almost interchangeably. The reason is that process interaction-based software is the most popular software available and is inherently job driven. However, as we will show later, this paradigm can also be used to develop logic for resource-driven simulations. The “opposite” of job-driven simulations have been resource-driven simulations. The worldview that most closely reflects this approach to modeling is the activity-scanning worldview, exemplified by Petri nets. Event scheduling has been described as a worldview that can be used to implement both job-driven and resource-driven simulations.

It is possible to focus on modeling resource cycles using a process interaction paradigm. An early example of this can be found on page 142 of the classic GPSS textbook by Tom Schriber [9]. Here the workers processing a large number of parts are introduced as “transactions” that cycle through states of being busy and idle.

A more general way of developing resident entity models using a process interaction paradigm can be seen by

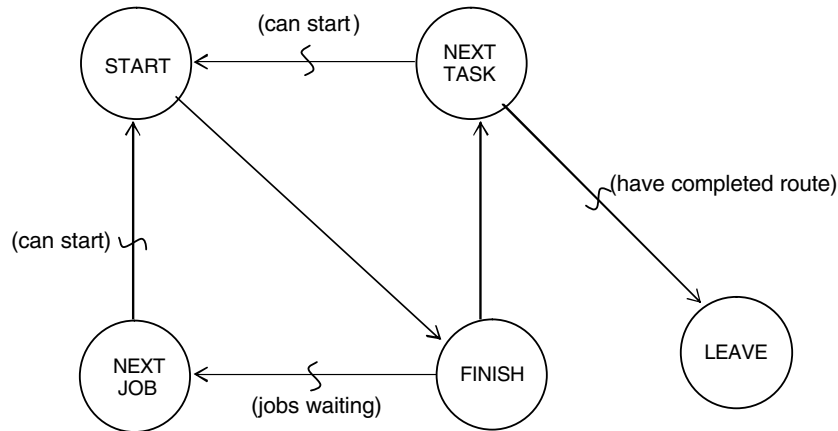


Figure 3. High-level event graph for the wafer fab simulation

Table 2. Simulation run lengths for the fab model

	Job Driven	Resource Driven
With preventive maintenances (PMs)/failures	≈ 3 hours	< 10 minutes (≈ 352 seconds)
Without PMs/failures	≈ 1.5 hours	< 10 minutes (≈ 300 seconds)

considering the following two models of a multiple server queue. We are using conventional GPSS block notation, but other process-oriented languages (Arena, SLAM, etc.) have similar block-diagram constructs. In Figure 1, the traditional process model is presented where transient entities are created and progress through the various blocks that describe their processing paths, seizing and releasing the servers as they become available.

In Figure 4, we are modeling the same system in an entirely different manner. Here, the jobs that find an idle server *become* that server, serving subsequent job arrivals until they become idle, at which time they are terminated.

The same simple multiple-server queuing model with  $R$  servers is shown in Figure 5 using event graphs.  $Q$  denotes the number of jobs in the system, the only dynamic state variable in the model. The (possibly random) variables,  $t_a$  and  $t_s$ , represent the interarrival and service times, respectively. To change the model from the current resource-driven model to a job-driven one, the state changes  $Q = Q + 1$  and  $Q = Q - 1$  would be changed to  $Q = Q + \text{PUT}()$  and  $Q = Q - \text{GET}()$ , where  $\text{PUT}()$  and  $\text{GET}()$  represent functions to put/get the attributes of queued jobs in/from a priority queue data structure, like the functions available in SIGMA. Job attributes can be passed between events as they are scheduled. Note that the simulation in Figure 5 will simulate processing jobs at the same speed no matter how many jobs are in the queue, 10 or 10 million—the difference being only the value of the integer  $Q$ .

Since it is possible to use a process interaction-based simulation language to implement resource-driven models, we simulated a  $G/G/s$  queue using Arena, a widely available commercial process interaction software, as well as using an event scheduling software. In both cases, resource-driven and job-driven simulations were run.

From experience, “job-driven” simulations become very slow and even stop running when the system becomes congested. To introduce system congestion, we ran multiple runs with increasing arrival rates while also increasing the number of servers to keep the system stable. (The traffic intensity  $\rho$  is defined as the arrival rate  $\lambda$  divided by the total service rate  $s\mu$ :  $\rho = \lambda/s\mu$  was held constant at  $\rho = 0.85$  to simulate moderately heavy traffic.)

Figures 6 and 7 show the runtimes in seconds for the experiments; 20,000 time units of simulated time were run on a Sony Vaio laptop with a Pentium 3 645-MHz processor and 256 MB of RAM.<sup>1</sup> In these two figures, we can see that there is very little difference between the four implementations when the system is small and has very few jobs. The runtimes start diverging as the system size (and number of jobs) increases. Indeed, the process interaction models were unable to complete their runs because the number of entities in the system exceeded the limit imposed by the

1. It should be noted that the runtimes for the process interaction-based runs are not as exact as those for the event scheduling-based runs. The process interaction software reports runtimes in fractions of minutes, while the event scheduling software returns the runtime in exact seconds.

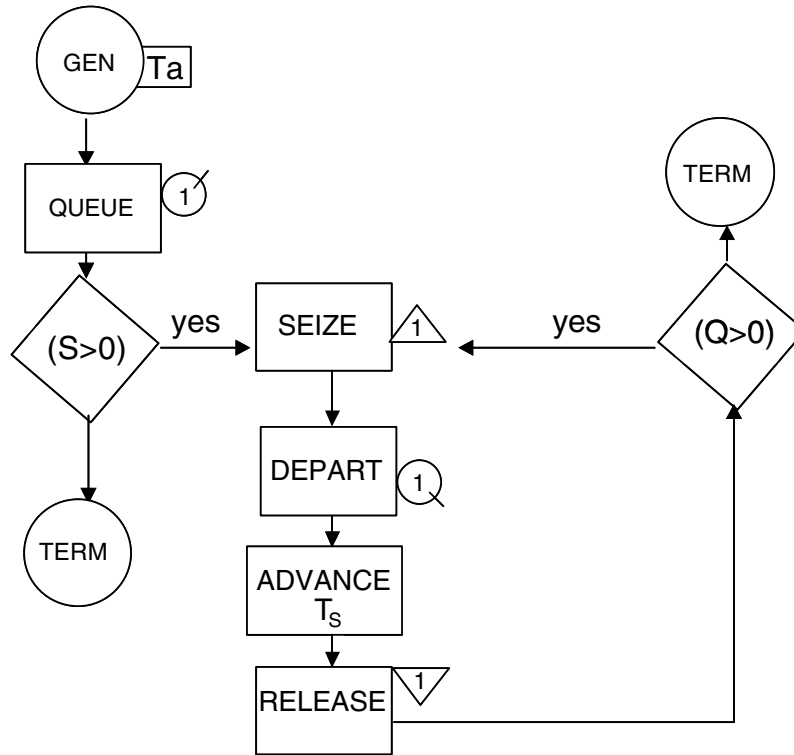


Figure 4. A resource-driven simulation using GPSS blocks

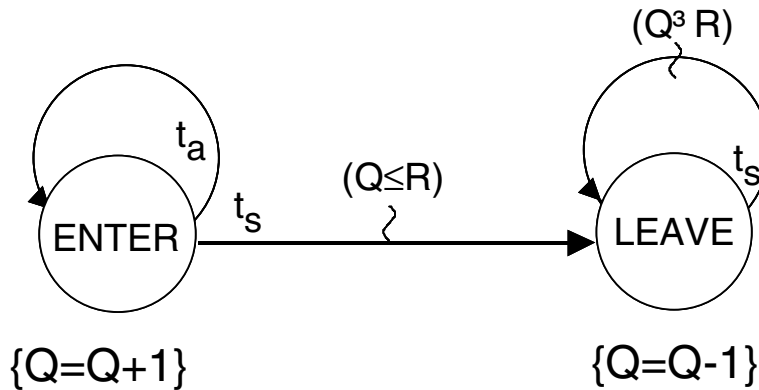


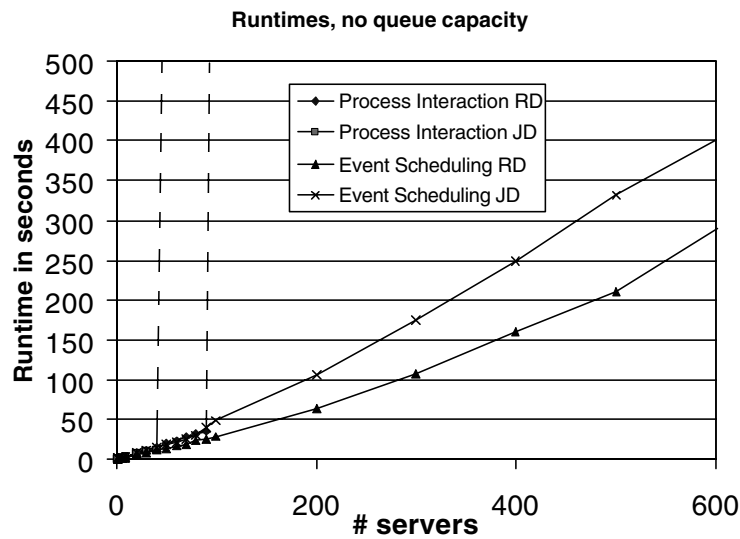
Figure 5. An event graph-based simulation of a  $R$  server queue

academic version of Arena.<sup>2</sup> For 1000 servers, the event scheduling job-driven runtimes also eventually became extremely long and are no longer shown on the chart (it took nearly 700 seconds to run the simulation). The runtimes for the resource-driven models also increased but are still

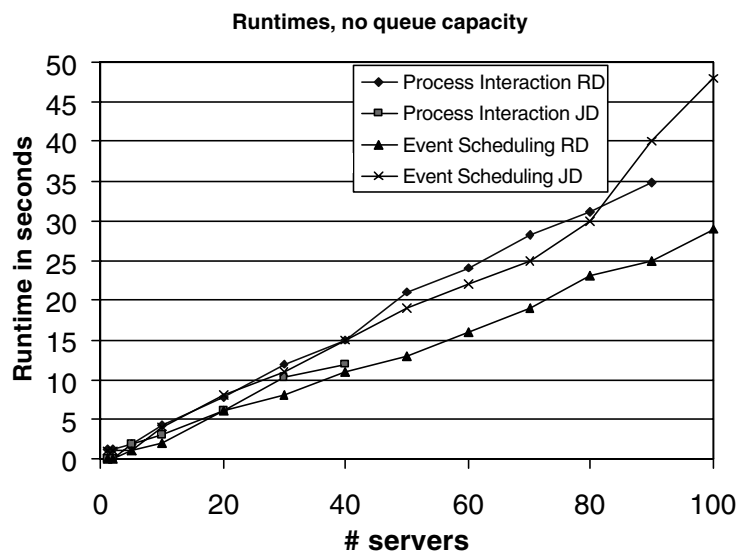
2. Since the software was unable to simulate these systems, we assigned the runs “infinite” run lengths. This is indicated by the dashed lines in the graph.

less than those for the job-driven simulations. This confirms the idea that job-driven simulations take significantly longer than resource-driven simulations when the system is large or highly congested.

Figure 8 shows the detail of Figure 7 for even smaller systems. Here, the process interaction resource-driven and event scheduling job-driven runtimes are nearly identical for several data points. At smaller system sizes, the same



**Figure 6.** Runtimes in seconds for the different paradigms and implementations for a G/G/s queuing system with varying numbers of servers. The queue has infinite capacity. RD = resource driven; JD = job driven.



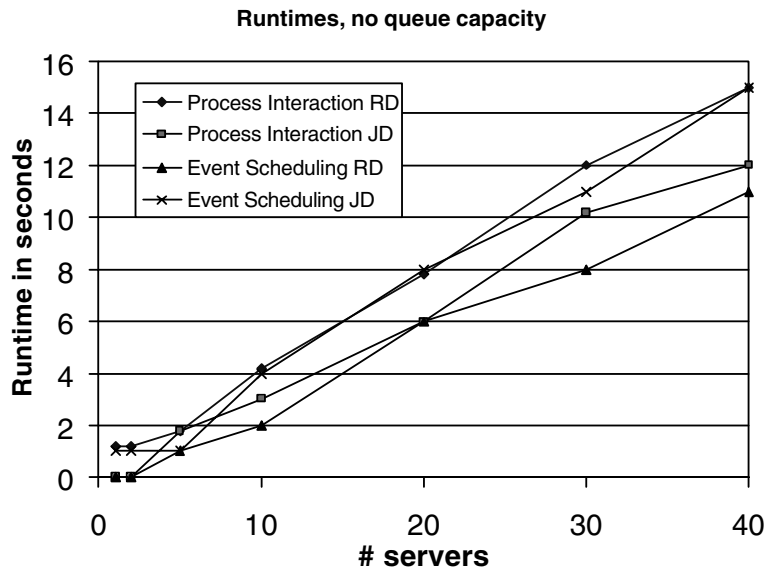
**Figure 7.** Detail for Figure 6 when the system is lightly loaded. RD = resource driven; JD = job driven.

holds for the event scheduling resource-driven and process interaction job-driven runs, although for the larger systems, the job-driven model is clearly above the resource-driven model. The lack of differences in the runtimes at lower intensities is likely in the experimental noise since the runtimes are all extremely short.

The graphs lead us to hypothesis that the differences in runtimes we experienced earlier in our large fab simulations are likely due to the actual implementation of the job-driven and resource-driven models using the process interaction and the event scheduling paradigms. (The fab

model was a large but very stable system with very low congestion. The engineers using this simulation confirmed that when loading is increased, the job-driven simulation runtimes can become arbitrarily long; increasing the loading has little effect on the runtimes of the pure resource-driven fab simulation.)

The conclusions drawn from Figures 6 through 8 may appear tenuous as there are many missing data points for the process interaction job-driven model (the version of Arena we had was limited to 100 entities). However, the message is only one of degree: eventually, under high enough



**Figure 8.** Detail of Figure 7 for very lightly loaded systems. RD = resource driven; JD = job driven.

congestion, the process interaction job-driven approach will break. To try to combat the simulation's terminating because of running out of space for entities, we ran the same set of experiments with a queue capacity (set to less than the number of entities allowed in the system).

We also used another measure of congestion by initializing the queue to capacity at the start of the runs. Figure 9 shows the runtimes for this system.

The first thing to note is that we did not manage to avoid the process interaction entity starving from occurring. This is because, for  $s \geq \text{max entities}$ , it does not matter that the queue is capacitated. If the system is busy, each server will be serving one entity, and so there will be more entities than allowed.

The second thing to note is that, again, the runtimes for the event scheduling job-driven and resource-driven simulations grow apart as the system grows larger (i.e., the number of entities in the system increases). In addition, the event scheduling runtimes are always less than the process interaction runtimes (where the software was able to execute the simulation).

Figure 10 shows the runtimes when the traffic is light. The event scheduling resource-driven model never takes more time than the other setups. An added twist shows that the process interaction resource-driven runs can take relatively longer than the process interaction job-driven model; this is because the Arena software is specifically designed for running the slower job-driven paradigm.

Finally, we present the runtimes from a simple  $G/G/1/c$  queue in which the traffic intensity  $\rho$  is steadily increased to simulate system congestion. (We are aware that most practitioners do not simulate unstable systems, in which jobs

arrive faster than they can be served. We believe, however, that this example underscores the above points by showing a complete set of runtimes, as the process interaction software was able to simulate all of the experiments run.)

Figure 11 shows the runtimes for the four different setups in which the queue is initialized at capacity. The first thing to note is that the runtimes do not exceed 50 seconds. This result is not surprising since the number of entities in the system is bounded, and so the simulation will not use more and more memory as the traffic intensity increases.

The second thing to note is that the runtimes for the event scheduling job-driven and resource-driven simulations are actually identical at the highest traffic intensity. Both are less than the runtimes for the process interaction software. In fact, again, at low congestion, the job-driven process interaction run takes less time than the resource-driven process interaction software.

Figure 12 shows the runtimes when the traffic intensity is light. The event scheduling resource-driven model almost always takes less time than the other setups. And, again, for higher intensities, the process interaction runtimes are longer than the event scheduling times.

#### 4. Discussion

Previous research has focused on the differences between resource-driven and job-driven simulations. One of the main conclusions drawn was that resource-driven simulations (especially of complex systems) are significantly faster than their job-driven counterparts. With this paper, we have shown that the event scheduling simulation logic is usually orders of magnitude more efficient than

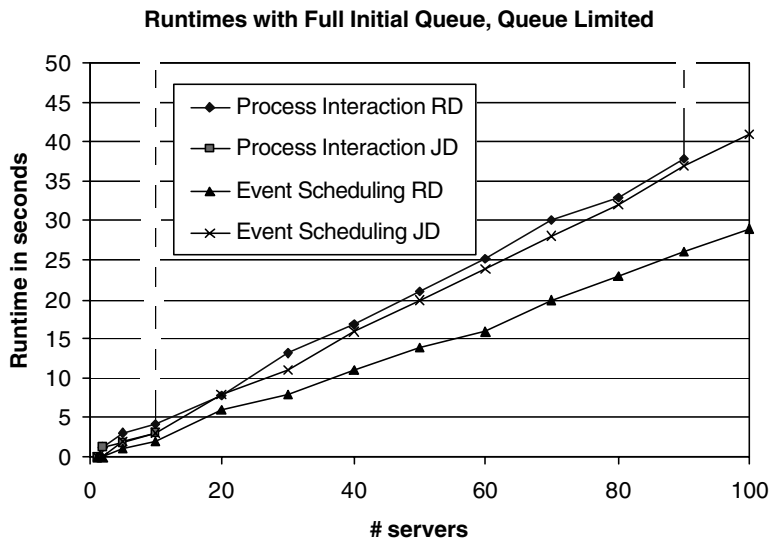


Figure 9. Runtimes for the G/G/s/c queuing systems with queues initialized to full. RD = resource driven; JD = job driven.

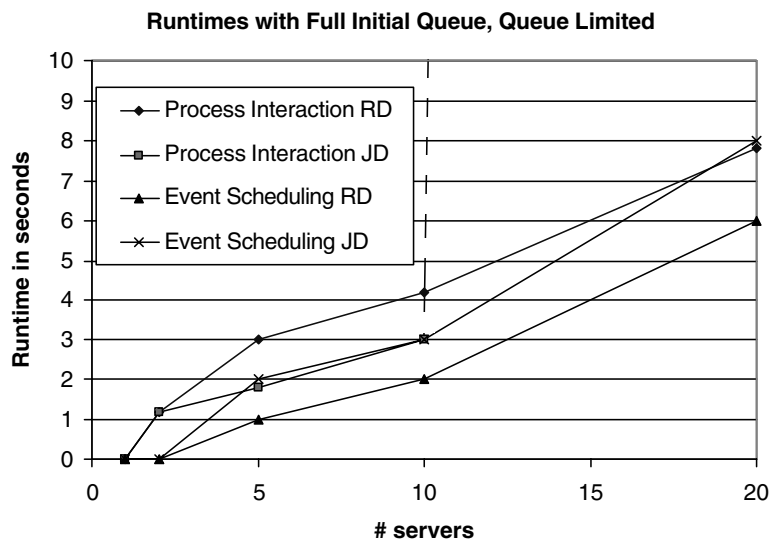


Figure 10. Runtimes for the G/G/s/c queuing system for small system sizes. RD = resource driven; JD = job driven.

process interaction logic for modeling large-scale highly congested systems, regardless of the software implementation. However, using the process-oriented Arena software, there were instances of very low congestion systems in which the job-driven simulations ran faster than the resource-driven models.

For very lightly congested systems, the differences in runtimes reported here are insignificant. Even for the very detailed fab simulation, the difference in runtimes between the job-driven SIGMA model and the resource-

driven SIGMA model was less than 2 minutes when simulating 2 years of data. Table 3 summarizes the runtimes. The difference between event scheduling resource-driven and job-driven simulations is insignificant compared to the difference between event scheduling job-driven and process interaction job-driven simulations.

Some of the disadvantages discussed for job-driven simulations (modeling failures as PMs, queuing of PMs/failures, deadlock, etc.) are actually phenomena of the process interaction paradigm, not of the job-driven

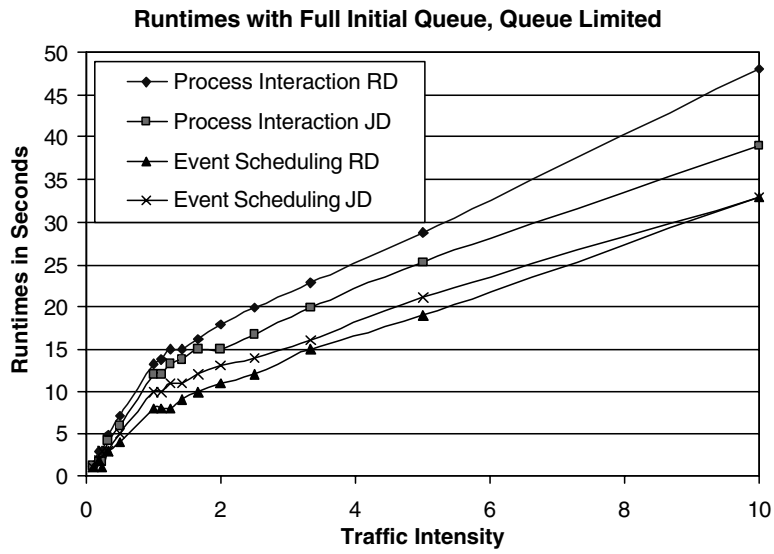


Figure 11. G/G/1/c queuing system with increasing traffic intensity. RD = resource driven; JD = job driven.

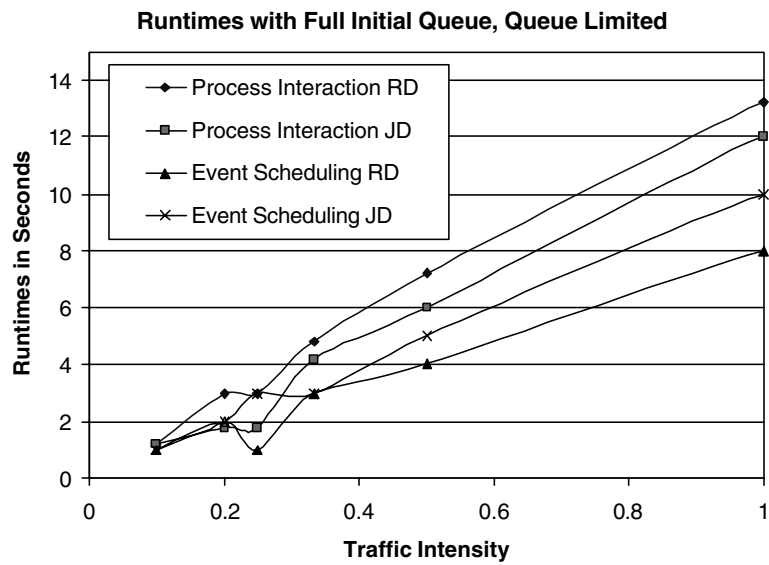


Figure 12. G/G/1/c queuing system for light traffic intensities. RD = resource driven; JD = job driven.

Table 3. Summary of runtimes for the fab model

	Runtime
Event scheduling, resource driven	~350 seconds
Event scheduling, job driven	~440 seconds
Process interaction, job driven	~3 hours

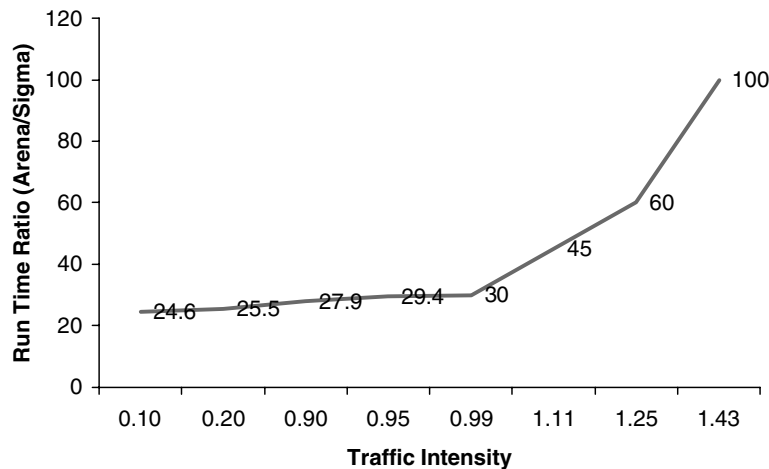


Figure 13. Comparison of runtimes for simple queue using two educational software packages

approach itself. Implementing a job-driven model using an event scheduling paradigm does not have any of these difficulties.

We hope that these results will cause more attention to be paid to the advantages of different simulation modeling approaches. Too often, simulation practitioners learn one paradigm or software package and assume that it is appropriate for all applications. Limiting oneself to only one simulation approach or software package is like a carpenter who has only one tool and finds himself or herself trying to drive nails with a saw. Our findings have perhaps more important implications in simulation education. Courses are too often designed around a single software package, thus shortchanging students by not introducing them to different modeling worldviews. Figure 13 illustrates this by comparing runtimes for educational versions of SIGMA and Arena, both used with the worldview they were primarily designed around.

## 5. Acknowledgments

The authors wish to thank Judy Rathmell of Rockwell/Arena for her generous help with our Arena modeling questions. Discussions with Debbie Pederson and Wai Kin Chan were useful. Thanks also to Seth Fischbein, Edward Yellig, and Mani Janakiram of the Intel Corporation. This research was supported in part by a graduate fellowship from the Graduate Academic Diversity Program at the University of California, Berkeley, administered by Carla Trujillo, and NSF grant DMI-9713549 and SRC contract 2001-NJ-960.

## 6. References

- [1] Derrick, E., O. Balci, R. Nance, and H. Shen. 2000. Conceptual frameworks for discrete event simulation modeling. Computer Science working paper, Virginia Polytechnical Institute.
- [2] Schruben, L. 1983. Simulation modeling with event graphs. *Communications of the ACM* 26 (11): 957-63.
- [3] Törn, A. 1981. Simulation graphs: A general tool for modelling simulation designs. *SIMULATION* 37 (6): 187-94.
- [4] Schruben, L. 2001. Efficient simulation of stochastic timed Petri nets. IEOR technical report BSL.01.1, University of California, Berkeley.
- [5] Roeder, T., S. Fischbein, M. Janakiram, and L. Schruben. 2002. Resource-driven and job-driven simulations. In *Proceedings of the 2002 International Conference on Modeling and Analysis of Semiconductor Manufacturing*, pp. 78-83.
- [6] AutoSimulations, Inc. 1999. *AutoSched AP user's guide v 6.2*. Bountiful, UT: AutoSimulations, Inc.
- [7] Schruben, D., and L. Schruben. 2001. *Graphical simulation modeling using SIGMA*. CITY: Custom Simulations.
- [8] Venkatesh, S., J. Smith, B. L. Deuermeyer, and G. L. Curry. 1998. Deadlock detection and resolution for discrete event simulation: Multiple-unit seizures. *IIE Transactions* 30 (3): 202-16.
- [9] Schriber, T. 1991. *An introduction to simulation using GPSS/H*. New York: John Wiley.

**Lee W. Schruben** is (POSITION?) in the Department of Industrial Engineering and Operations Research, University of California, Berkeley.

**Theresa M. Roeder** is (POSITION?) in the Department of Industrial Engineering and Operations Research, University of California, Berkeley.