Graphical Representation of OMNeT++ Simulation Traces

Nicky van Foreest

July 7, 2002

1 The Problem

It is often difficult to track the behavior of a simulated distributed system. The problem is that a number of instances communicate and change state continuously. To understand 'what is going on' it is helpful to have an overview of the instance states at the same time. As simulation traces are usually just prints of lists of events, i.e., sequential in time, it is hard to obtain this desired information, i.e., a cross section in time of the instance states.

2 Some Sort of a Solution

A graphical representation of the evolution of the system helps towards getting an overview of its behavior. One method to represent this is a message sequence chart (msc). Here I demonstrate the use of msc's by means of a multiserver fifo queue. In brief, the behavior of a multiserver queue is as follows. A generator generates jobs and sends these jobs to a queue. When a job arrives at the queue and sees a free server, it starts its service immediately. When all servers are busy, the job is queued until a server becomes free. Once a job's service is finished, it moves from the server to the sink.

The msc in Figure 1 contains six instances: a job generator, a queue, three servers, and a job sink. Jobs are represented as arrows between instances. The arrival times, as well as the service starting times, are shown at the far left; the departure times are at the right. Time runs downward. When a job is generated it moves to the queue. This is shown by an arrow from the `Gen` to the `Queue`. At the `Queue` a small arrow from right to left gives the number of jobs in queue that the arriving job sees. For instance, `job-4` sees 1 job in front of it in queue. When a job moves from the `Queue` to one of the `Servers`, a small arrow gives the number of jobs in queue that the job leaves behind. Consider `job-0`, clearly the system (and the queue) is empty when it arrives, and since the job can start its service immediately, it leaves an empty queue behind.

3 Producing the MSC

I wrote a class `MSC` that writes LaTeX commands to a file during a simulation. When the simulation finishes, LaTeX produces the msc using `msc.sty`, a style file written by V. Bos and S. Mauw, [1].

Admittedly, this procedure is a bit of a hassle. I tried to find a GPL tool that could parse an automatically generated list of messages. However, I could not find such a tool. The work-around via LaTeX is quite acceptible for my goals. A nice offspin of using the msc style file is that the result looks really pretty.

As an aside, the code can handle multipage msc's without problems (LaTeX may run out of memory when processing the file, though). However, I find reading very long msc's not that enjoyable, although it is still far better than reading the event lists. Hence, I use the msc's for very short simulations to check (and debug) my code. Once I decide that the code is bug free, I switch off producing msc's.

References

[1] V. Bos and S. Mauw. *The MSC Macro Package*. 2002. Style file and manual available at: `http://www.win.tue.nl/~sjouke/mscpackage.html`.

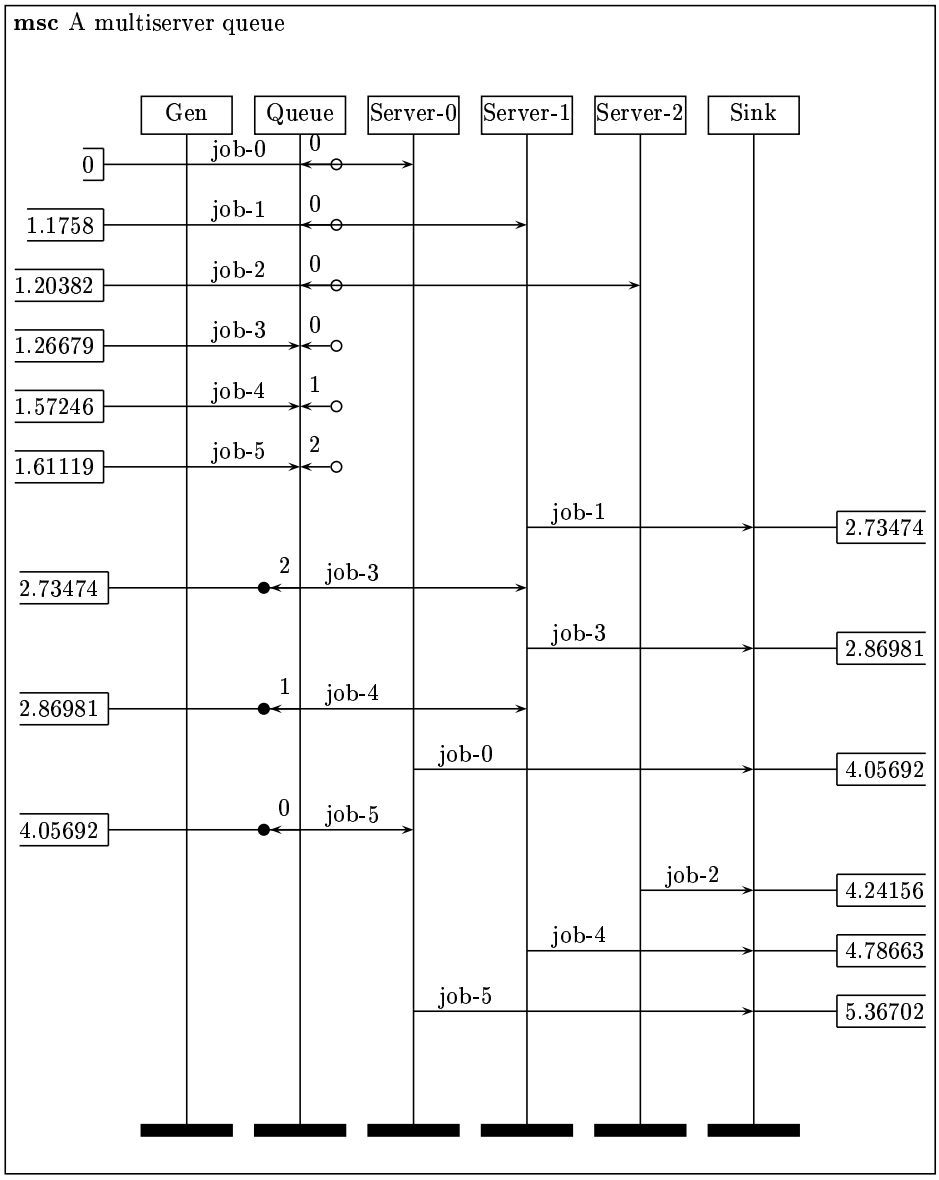

Figure 1: A demonstration of using msc's to represent the evolution of a multiserver queue.