Two Main Practical Challenges of Performance Modeling

Reading about performance modeling / simulation and system sizing, you often see two completely opposite views of the subject. Either authors describe in detail how you can model performance using some math and you may feel that as soon as you comprehend that math, you won't have any problem with modeling. Or authors say that it is a black magic and you'd better stay away from it or do it in a minimal way with simple trending (while you probably won't see that view in serious books, it is often can be seen in Internet discussions).

The truth, as usual, is in the middle. Modeling is a very helpful and works well if you use it properly and understand its limitations. And there are two main challenges here that rarely get highlighted – while everybody who wants to approach the topic should understand them clearly.

The first challenge is that modeling works well for known resource limitations. You should know these limitations in advance (and how your system uses that limited resource – which is also a challenge, but more technical one). For example, if your system is processor-bound and you know how much cpu it takes per transaction, you may build a rather simple and pretty accurate model (using queuing theory or even something pretty simple – for example, if you stay away from heavy cpu utilization, linear model may work well with multi-processor systems).

But that model would never tell you when you run out of another resource and run into another kind of bottleneck until you build it into the model. And beyond a few common resources (processor, memory, disk, network) and explicitly introduced throttling, you usually don't know about bottlenecks until you run into them. This is the primary reason that results of your model (which may be perfect from the math point of view) are not reliable if you model significantly higher load than you tested / validated – as far as there is a high probability that you run into another bottleneck you are not taking in consideration now. However, the model would provide the best possible case (which turns correct when you fix all other bottlenecks you didn't take in consideration at the moment of modeling), which is important information by itself. A model would also be very useful to see if the system behaves up to expectations – or there are internal issues degrading performance and preventing scalability (that may be not so trivial to catch in complex systems).

Another challenge is a lack of performance-related metrics of hardware to use in modeling. You can find detailed hardware specifications, but they won't tell you how fast your systems would work on this hardware. As far as I understand, the only relatively objective approach (without testing the real system on the real hardware) is to use existing benchmark results to compare performance (keeping in mind that they represent results of this specific benchmark, not your systems). Most serious commercial modeling tools come with a library of hardware configurations and their performance metrics, allowing what-if performance analysis. It looks like keeping such libraries is a pretty time-consuming task and their quality may differ. Such a library is usually a major advantage of commercial modeling tools in comparison with free or inexpensive modeling tools (which may be quite good from the mathematical point of view, but you need to provide all numbers yourself).

IDC made an interesting move here introducing QPI (Qualified Performance Indicator) http://www.idc.com/QPI/index.jsp as a part of IDC's Server Decision Suite Metrics (free 30-days trial available http://www.idc.com/QPI/server.jsp). A kind of independent performance library that may be used for proper performance modeling / sizing (and, as far as I understand, going well beyond performance, integrating this information with other IT-related metrics such as price, power, and size – it should be a very interesting optimization task to find the best hardware configuration based on all these metrics).