Stochastic Models: Model Selection and Optimisation

Lecture 6 – From Evaluation to Decisions

Sebastian Müller

Lecture 6

Where We Are

- **Lecture 1**: Motivation, Poisson refresher, simulation mindset.
- **Lecture 2**: Queueing basics, M/M/1 and M/M/c, Little's Law.
- **Lecture 3**: Error control, CLT and concentration, variance reduction, sensitivity.
- **Lecture 4**: Parameter estimation for Poisson, M/M/1, M/G/1; plug-in performance.
- **Lecture 5**: M/G/2 evaluation exercise from event log to fitted model.

Today: move from evaluation of a given system to optimisation of design choices.

Goals for Lecture 6

- Decide which queueing model ($G/G/c/K/\ldots$) is appropriate for a given dataset and question.
- Use fitted models to run simulation experiments with confidence intervals.
- Perform **scenario analysis**: how changes in load, variability, or capacity affect performance.
- Introduce **costs** and formulate a simple optimisation problem over design variables (e.g. number of servers).
- Connect the full pipeline: data $\to$ model $\to$ estimation $\to$ simulation $\to$ decision.

From Data to Candidate Models

- Starting point: cleaned event log (Lecture 5) with arrivals, service times, and queue lengths.
- We want a **parsimonious** model that captures key features relevant for decisions.
- Two equally important dimensions:
  - **Structural choice**: M/M/1, M/M/c, M/G/1, M/G/c, GI/G/c, finite buffers, abandonment, priorities, networks.
  - **Parameter estimation**: $\hat{\lambda}$, service distribution parameters, variability measures.
- Model choice should be driven by *data*, *domain knowledge*, and *decision needs*.

Dimensions of Choice

- **Arrival process**
  - Poisson (homogeneous) vs. time-varying rate vs. overdispersed arrivals.
  - Diagnostics: inter-arrival histogram, ECDF, overdispersion of counts, time-of-day effects.
- **Service-time distribution**
  - Exponential vs. deterministic offset + exponential vs. heavy-tailed.
  - Diagnostics: empirical CV, skewness, log-survival plot, QQ-plot against exponential.
- **System structure**
  - Number of servers c, buffer capacity K, priority classes, reneging/abandonment.
  - Single-node approximation vs. network / multi-stage system.

Simple vs Complex Models

Idea

Start with a simple baseline model and add complexity only when necessary for decisions.

- Baselines:
  - M/M/c for systems with no strong evidence against exponential assumptions.
  - M/G/1 or M/G/c when service-time variability clearly deviates from exponential.
- More flexible options:
  - GI/G/c approximations when inter-arrivals also deviate from Poisson.
  - Empirical service distributions (resampling) when no simple parametric family fits well.
- Trade-off:
  - Simpler models are easier to explain and calibrate.
  - Richer models may better capture tails or bursts.

Model Selection Guidelines

- **Structural consistency**: data must not contradict assumptions (e.g. observed concurrency $\leq c$ if you model c servers; no abandonment if model excludes it).
- **Descriptive fit**: do simulated histograms / quantiles for arrivals and services match the data?
- **Performance fit**: does the model reproduce key metrics (mean W_q, L_q, utilisation) within uncertainty bands?
- **Decision relevance**: only add complexity if it changes the recommendation (e.g. number of servers to deploy).
- **Interpretability**: stakeholders should understand the assumptions and their limits.

Pick the simplest model that fits the data and supports the decision without violating observed behaviour.

Case Study Pitfall: Fitting M/G/3 Data with M/G/2

- **What went wrong in Lecture 5?**
  - Data inspection showed moments with 3 jobs in service $\Rightarrow$ evidence for $c = 3$.
  - Fitting M/G/2 underestimates capacity, inflates utilisation, and biases wait predictions.
- **How to avoid it**
  - Plot max number in service over time; compare to assumed c.
  - Refit with $c = 2$ vs $c = 3$ and compare simulated W_q, L_q, tails *with CIs*.
  - Prefer the model that (i) matches observed service concurrency and (ii) does not degrade fit on waits/queues.
- **Report explicitly** when structural choices are ambiguous; give a decision recommendation for each plausible c.

What-If Analysis

Idea

Use the fitted model as a baseline and explore how performance changes when we perturb inputs or design choices.

- Vary arrival rate: $\lambda = \alpha\,\hat{\lambda}$ for $\alpha \in \{0.8, 1.0, 1.2\}$ (demand uncertainty).
- Vary number of servers: $c \in \{1, 2, 3, 4\}$ (staffing / capacity decisions).
- Optionally vary service variability: more/less variable G with the same mean.
- For each scenario, simulate and compute CIs for W_q, L_q, utilisation, and service-level metrics.

Goal: identify regimes where the system is robust and regimes where it is fragile (near saturation).

Interpreting Scenario Tables

- Present results as tables or heatmaps:
  - Rows: number of servers c.
  - Columns: demand multiplier α or other scenario parameter.
- For each cell, report:
  - Point estimates and 95% CIs for W_q, L_q, utilisation.
  - Possibly probability of violating a service-level target (e.g. $P(W_q > w^*)$).
- Use these tables to communicate trade-offs:
  - Where does adding a server significantly improve waiting times?
  - Where do CIs for different configurations overlap (no clear winner)?

Adding a Cost Model

Definition

We translate performance metrics into monetary (or utility) costs to support design decisions.

- Basic ingredients:
  - Server cost c_{server} per server per unit time.
  - Waiting cost c_{wait} per unit waiting time per job.
  - Optional penalty for SLA violations (e.g. if $W_q > w^*$).
- Given arrival rate λ and mean waiting time $W_q(c)$ for configuration c:

$$C(c) = c_{\text{server}} \cdot c \; + \; c_{\text{wait}} \cdot \lambda \, W_q(c).$$

- $W_q(c)$ is estimated via simulation; hence $C(c)$ is also stochastic with a CI.

Simulation-Based Optimisation

- For small discrete design spaces (e.g. $c = 1, \ldots, 6$), we can simply enumerate options:
  - For each c, estimate $W_q(c)$ via multiple replications.
  - Compute $\hat{C}(c)$ and a 95% CI using the CI for $W_q(c)$.
- Choose the configuration with the smallest $\hat{C}(c)$, but report uncertainty:
  - If CIs for $C(c)$ overlap, it may be safer to report a set of near-optimal options.
- Optional: add constraints, e.g. $P(W_q > w^*) \leq \alpha$ (service-level constraint), and pick the cheapest feasible c.

From Numbers to Recommendations

- Translate optimisation results into plain-language recommendations:
  - "With current demand, 2 agents minimise expected cost; 3 agents only improve waiting times slightly."
  - "If demand increases by 20%, the system becomes unstable with 2 agents; we recommend adding a third."
- Emphasise uncertainty: use CIs and scenario analysis to show robustness.
- Document assumptions: model structure, stationarity, independence, stability conditions.

Course Pipeline Recap

- **Model**: choose a queueing framework and assumptions informed by data and context.
- **Estimate**: infer parameters from data (Lecture 4) and quantify input uncertainty.
- **Simulate**: implement discrete-event simulations with error control (Lecture 3).
- **Evaluate**: compare model predictions to observed performance (Lecture 5).
- **Optimise**: use costs and constraints to recommend configurations (Lecture 6).

This is the workflow you should apply in your final projects.

For Your Projects

- Clearly state:
  - Modelling assumptions and chosen queueing framework.
  - How parameters were estimated and with what uncertainty.
  - How simulation was used to compare policies or configurations.
  - What decision or recommendation you make and how robust it is.
- Focus on a coherent modelling story rather than perfectly tuned parameters.