Software Architecture Modeling

Arnav Gupta

December 8, 2024

Contents

1 Architectural Modeling

Architectural Model: artifact that captures some or all design decisions that comprise a system's architecture

Architectural Modeling: reification and documentation of design decisions

Architectural Modeling Notation: language or means of capturing design decisions

Architects and other stakeholders must make critical modeling decisions (what, how detailed, formality), weighing cost and benefit.

Model basic architectural elements like components, connectors, interfaces, configurations, and rationale (reasoning behind decisions). Also model constraints on interactions, behaviour, and concurrency.

Aspects:

- **static**: do not change as system runs
- **dynamic**: change as system runs
- **functional**: something the system does directly
- **non-functional**: something about the qualities of the system

2 Software Performance Models

Includes Execution Graphs, Queueing Networks, and Machine learning-based performance needs.

Formal representations: capture aspects of software performance, help understand non-functional requirements

Software Performance Models allow:

- estimating performance of software
- estimating resource needs
- identify performance issues as early as possible
- simulate execution of software under certain conditions (number of users and size of infrastructure)
- establish performance for average, best, and worst-case scenarios

Design models of system capture static aspects, performance models capture dynamic aspects.

3 Software Execution Models

Constructed early in development process to ensure that chosen software architecture can achieve required performance objectives.

Provides static analysis of mean, best, and worst case response times.

Generally sufficient for identifying serious performance problems at architectural and early design phases.

Absence of problems in software model does not mean there are none.

3.1 Execution Graphs

Execution graphs represent sequence of operations of system, and are constructed for each performance scenario.

Not sufficient for a complete analysis of software performance but work well for understanding software and non-functional requirements. Special annotation can give idea of performance. Combine graphs with other models to complement analysis.

3.1.1 Graph Notation

Basic nodes: represent processing steps at lowest level of detail that is appropriate for current development stage

Expanded nodes: represent processing steps elaborated in another subgraph

Repetition nodes: subsequent nodes repeated n times, last node has an edge to the repeat node

Case node: represent conditional execution of processing steps, each attached node has a probability of execution

Pardo node: attached nodes run in parallel: all nodes must complete (join) before proceeding

Division node: attached nodes represent new processing threads, need not all complete before proceeding

3.2 Execution Model Analysis

Software execution model analysis helps:

- make quick check of best-case response time to ensure architecture and design will lead to satisfactory performance
- assess performance impact of alternatives
- identify critical parts of system for performance management
- derive params for system execution model

Algorithms are formulated for evaluating graphs.

Basic solution algorithms are easy to understand by examining graphs, identifying basic structure, computing time of basic structure and reducing basic structure to a computed node, and continuing until only one node left.

Basic structures are sequences, loops, and cases.

Computation for case nodes differs for:

- shortest path: time for case node is minimum of times for conditionally executed nodes
- longest path: time for case node is maximum of times for conditionally executed nodes
- average analysis: time is multiplying each node's time by execution probability

3.3 Software Resource Requirements

Each basic node has specified software resource requirements A_j for each service unit j.

Processing Overhead Matrix: chart of computer (hardware) resource requirements for each of the software resource requirements

To compute total execution time:

1. use processing overhead matrix to calculate total computer resources required per software resource for each node in the graph
2. compute total computer resource requirements for the graph
3. compute best-case elapsed time

4 Queuing Network Models

Software execution models provide a static analysis of the mean, best, and worst case response times for software and characterize resource requirements of proposed software alone.

Queuing Network Models characterize software's performance in the presence of dynamic factors (like other workloads or multiple users) and aims to solve resource contention.

If software execution model indicates no problems, ready to construct and solve queuing networks to account for contention efforts.

Benefits of QNM:

- more precise metrics that account for resource contention

- sensitivity of performance metrics to variations in workload composition
- scalability of hardware and software to meet future demands
- effect of new software on service level objectives of other systems
- identification of bottleneck resources
- comparative data on performance improvement options

Resource contention can come from:

- multiple users of an application or transaction executing at once
- multiple applications or systems executing on the same hardware at once
- concurrent processes
- multi-threaded application

Key computer system resources in QNM are server and queue:

- **server**: component of the environment that provides some service to the software
- **queue**: jobs waiting for service, consists of waiting line and server

Kendall notation for queues is A/S/m/B/K/SD:

- A is interarrival time distribution
- S is service time distribution
- m is number of servers
- B is number of buffers (service capacity)
- K is population size
- SD is service discipline

Distributions can be:

- M: exponential
- E: Erlang with param k
- H: hyperexponential with param k
- D: deterministic

- G: general

Performance metrics of interest:

- **residence time (RT)**: average time jobs spend in server, in service and waiting
- **utilization (U)**: average percentage of time server is busy
- **throughput (X)**: average rate at which jobs complete service
- **queue length (N)**: average numbers of jobs at the server (receiving service and waiting)

Value of metrics depends on:

- number of jobs
- amount of service they need
- time required for server to process individual jobs
- policy used to select next job from queue

Can know performance from Kendall notation values.

For busy time B, completed jobs C, and total period T (and the above performance metrics):

- utilization: $U = B/T$
- throughput: $X = C/T$
- mean service time: $S = B/C$
- area under graph: $W = \sum_{time}(\text{number of jobs})$
- residence time: $RT = W/C$
- queue length: $N = W/T$

In early phases of development, can't make measurement of software to derive execution profile.

Workload intensity: measure of number of requests made by a workload in given time interval

Service requirement: amount of time that workload requires from each device in processing facility

Jobs-flow balance: assume that system is fast enough to handle arrivals and thus completion rate or throughput equals arrival rate

Let λ be arrival rate (workload intensity) and S be mean service time (service requirements). Then:

- throughput: $X = \lambda$
- utilization law: $U = XS$
- residence time: $RT = \frac{S}{1-U}$
- queue length: $N = X * RT$ (Little's Law)

To find **steady-state probabilities** of birth-death process, define the rate at which events occur (birth rate and death rate):

- let λ_n be the birth rate when the system is in state n
- let μ_n be the death rate when the system is in state n
- let ρ_n be the steady-state probability of being in state n
- then rate of flow into state n is rate of flow out of state n (since steady-state) $$\rho_{\mathrm{n}}\ \lambda_{\mathrm{n}} = \rho_{\mathrm{n+1}}\ \mu_{\mathrm{n+1}}$$

For general ρ_n:

$$\rho_n = \frac{\prod_{i=0}^{n-1} \lambda_i}{\prod_{i=1}^{n}} \rho_0$$

Probability of having n jobs in the system for M/M/1 queue:

$$\rho_n = \left(\frac{\lambda}{\mu}\right)^n \rho_0$$

where $\frac{\lambda}{\mu}$ is the traffic intensity, average number of customers arriving compared to the number being served.

Probability of having n or more jobs in the system for M/M/1 queue:

$$\left(\frac{\lambda}{\mu}\right)^n$$

Mean number of jobs in the system:

$$\frac{\frac{\lambda}{\mu}}{1 - \frac{\lambda}{\mu}}$$

For M/M/m queue, birth rate is arrival rate which is λ and death rate is service rate which is $\mu_n = n\mu$ for $n = 1, \ldots, m-1$ and $\mu_n = m\mu$ for $n \geq m$:

- utility is $\rho = \lambda / m\mu$
- probability of having n jobs in the system is

  $$\frac{(m\rho)^n}{n!} p_0$$

  for $n = 1, \ldots, m-1$ and

  $$\frac{\rho^n m^m}{m!} p_0$$

  for $n \geq m$
- probability that an arrivign job has to wait in the queue is

  $$p_0 \frac{(m\rho)^m}{m!} \sum_{n=m}^{\infty} \rho^{n-m}$$

Queuing network consists of 2+ queues connected together and serve requests sent by clients.

Request routing in queuing network is specified by **probability matrix**.

Types of QNM:

- **open models**: requests come from a source external of queuing network and leave network after service completion
  - appropriate for systems with external arrivals and departures
  - specify workload intensity and service requirements
    - workload is arrival rate
    - service requirements are number of visits for each device and average service time per visit (or total demand for that device)
- **closed models**: no external source of requests and no departing requests (population of requests in queuing network remains constant)
  - needs number of users and think time (average delay between receipt of response and submission of next)
- **mixed models**: open for some workload classes and closed for others

To derive system model params from software model results:

1. use queue-servers to represent key computer resources or devices specified in the software execution model and add connections between queues to complete model topology
2. decide whether system is best modeled as an open or closed QNM
3. determine workload intensities for each scenario
4. specify service requirements

Modeling hints:

- multiple users and workload
- average vs peak performance (basic QNMs calculate average values)
- sensitivity: if small change in one param causes large change in computed metrics, model is sensitive to that quantity
- scalability: improves response times for anticipated future loads
- bottlenecks: bottleneck device is one with highest utilization