Processor modelling using queueing theory

Erik Leonards

April 24, 2023

Bachelor thesis Mathematics and Computer Science

Supervisor: prof. dr. (Rude)sindo Núñez Queija, prof. dr. Anuj Pathania

Informatics Institute
Korteweg-de Vries Institute for Mathematics
Faculty of Sciences
University of Amsterdam

Abstract

Schrijf een samenvatting van hoogstens een halve bladzijde waarin je kort uitlegt wat je hebt gedaan. De samenvatting schrijf je als laatste. Je mag er vanuit gaan dat je docent de lezer is.

Title: Processor modelling using queueing theory
Authors: Erik Leonards, erik.leonards@student.uva.nl, 13170007
Supervisors: prof. dr. (Rude)sindo Núñez Queija, prof. dr. Anuj Pathania
Second grader: dr. Florian Speelman, prof. dr.
End date: April 24, 2023

Informatics Institute
University of Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.ivi.uva.nl`

Korteweg-de Vries Institute for Mathematics
University of Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.kdvi.uva.nl`

Contents

1. Introduction

Vertel iets over je artikel, noem je vraagstelling. Richt je tekst op medestudenten. (Richtlijn 2 bladzijden).

2. A computer system

A computer system is an electronic device that is used for storing and processing data according to computer instructions. A task is a long list of instructions. The tasks can be very complicated and many tasks require input from the user. Users can be very unpredictable and the response time of a user can be very long. This is why we will only consider user independent tasks. Furthermore, repeatability is an important concept in scientific research, so we will only consider deterministic tasks. For those tasks, the most important components of a computer are the processor and the memory. Section 2.1 and Section 2.2 describe respectively the processor and memory. Section 2.3 specifies how the tasks are executed and which resources they require.

2.1. The processor

The processor is a hardware component of the computer system that is responsible for the execution of instructions. The processor comprises a set of processor cores, which can independently execute tasks. This is called task parallelism. Moreover, each core can execute multiple instructions at once. This is called instruction level parallelism.

2.2. The memory

There are many different types of memory and there exists an entire memory hierarchy.

2.3. Task execution

As said earlier, we will only consider deterministic tasks.

2.4. Task simulation software

3. Queueing theory and Markov chains

3.1. Continuous Markov chains

3.2. A queueing network

3.2.1. The customers

A closed queueing network comsists of a finite number of service centers, a finite number of different classes of customers and a finite number of customers of each class. Let M be the number of service centers and let R be the number of different classes of customers. We will index the service centers from 1 to M and often i and j to denote an index of a service center and k to denote the position in the queue. Furthermore, classes are indexed from 1 to R and r, s often denote the class. For $1 \leq r \leq R$ let N_r be the total number of customers of class r.

The customers travel through the network according to transition probabilities. Those transition probabilities are equal for all customers within the same class. Let $p_{i,j}^{(r)}$ denote the transition probability of a customer of class r to travel from service center i to j. Notice how the sequence of service centers being visited by a customer forms a Markov chain, because the next service center only depends on the current service center. This means that a customer's path is a Markov chain on the set

$$\{(i, r) \ : \ 1 \leq i \leq M, 1 \leq r \leq R\},$$

where i denotes the service center and r denotes the class of the customer.

3.2.2. The service centers

The service center is defined as a queue exhibiting the First In First Out (FIFO) server discipline. Each service center is determined by its capacity and service rate. The capacity denotes the number of customers a service center can serve at once. A service rate of $\mu \in \mathbb{R}_{>0}$ means that the service times of the customers being served are $E(\mu)$ (exponentially with parameter μ) distributed. Notice how the service times distributions are equal for all classes. For $1 \leq i \leq M$ let $C_i \in \mathbb{Z}_{>0}$ and μ_i be respectively the capacity and service rate of service center i. Now define

$$\min\{C_i, k\}\, \mu_i,$$

to be the total rate at which customers leave service center i.

3.2.3. The state of the model

For a network with M service centers the state of the network is denoted by $\mathfrak{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M)$, where $\mathbf{x}_i$ denotes the state of service center i. The state $\mathbf{x}_i$ of service center i is denoted with $\mathbf{x}_i = \left(x_{i,1}, x_{i,2}, \ldots, x_{i,n_i(x)}\right)$ where $n_i(x) = |\mathbf{x}_i|$ is the number of customers at service center i and $x_{i,k}$ is the class of the customer at the k-th position in the queue. Formally, the state space is given by

$$\chi \coloneqq \left\{ \mathtt{x} : \mathtt{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_M), 1 \leq x_{i,k} \leq R, 1 \leq i \leq M, 1 \leq k \leq n_i(x), \sum_{i=1}^{M} \sum_{k=1}^{n_i(x)} \delta_{r,x_{i,k}} = N_r \right\},$$

where the last condition ensures that exactly N_r number of class r customers are in the system and

$$\delta_{i,j} = \begin{cases} 1 & : i = j \\ 0 & : i \neq j \end{cases},$$

denotes the Dirac delta function. When the state is obvious from the context, we will often write n_i instead of $n_i(x)$.

It is important to observe that the queue contains both the waiting and served customers. To be more precise, the customers $x_{i,1}, \ldots, x_{i,\min\{C_i, n_i(x)\}}$ are being served by service center i with rate μ_i. Furthermore, the next service center of a customer is determined by the transition probabilities $p_{i,j}^{(r)}$. This means that the state space and transition probabilities induce transition rates between the states.

3.2.4. Aggregate states

The number of customers of a class at the service centers is often regarded as the most relevant information of the state. This is what aggregate states describe. An advantage of aggregate states is that each aggregate state will represent a large number of states in χ, which will significantly decrease the size of the state space and the computational complexity. An aggregate state is denoted by $\mathfrak{n} = (\mathbf{n}_1, \ldots, \mathbf{n}_M)$ with $\mathbf{n}_i = (n_{i,1}, \ldots, n_{i,R})$ where $n_{i,r}$ denotes the number of class r customers at service center i. The aggregate state space is then formally defined by

$$\mathcal{N} \coloneqq \left\{ \mathfrak{n} : \mathfrak{n} = (\mathbf{n}_1, \ldots, \mathbf{n}_M), n_{i,r} \geq 0, 1 \leq i \leq M, 1 \leq r \leq R, \sum_{i=1}^{M} n_{i,r} = N_r \right\}.$$

For a state $\mathfrak{n} \in \mathcal{N}$ we will use $n_i(\mathfrak{n})$ to denote the total number of customers at service center i, i.e. we have

$$n_i(\mathfrak{n}) = \sum_{r=1}^{N} n_{i,r}.$$

Similar to $n_i(x)$, we will often write n_i instead of $n_i(\mathfrak{n})$ when the state is clear from the context.

3.2.5. The model as continuous Markov chain

As observed in Section 3.2.3, the state space, transition probabilities $p_{i,j}^{(r)}$, capacities C_i and rates μ_i induce transition rates between the states. To be more precise, a customer from class r being served by service center i induces a rate of $p_{i,j}^{(r)}\mu_i$ between the current state and the state in which this customer is removed from queue $\mathbf{x}_i$ and added at the end of queue $\mathbf{x}_j$. Furthermore, note how the transition rates between states in χ is zero if two or more customers travel at once between service centers. This is due to the fact that two or more customers travelling at once has zero probability of occurring. Let $Q = (q_{\mathtt{xy}} : \mathtt{x}, \mathtt{y} \in \chi)$ be the transition rate matrix.

3.2.6. The invariant distribution

Let $\pi = (\pi_{\mathtt{x}} : \mathtt{x} \in \chi)$ be the invariant distribution. This distribution can be found by solving the balance equations

$$\pi_{\mathtt{x}} \sum_{\mathtt{y} \in \chi} q_{\mathtt{xy}} = \sum_{\mathtt{y} \in \chi} \pi_{\mathtt{y}} q_{\mathtt{yx}}, \quad \mathtt{x} \in \chi.$$

Informally, these equations equate the total rate of outflow to the total rate of inflow.

Before presenting the solution to the balance equations, we must first introduce the **one customer invariant measure terms**.

Definition 3.2.1. *The **one customer invariant measure terms** are given by*

$$e = (e_{i,r} : 1 \leq i \leq M, 1 \leq r \leq R),$$

which satisfy the equations

$$\sum_{j=1}^{M} e_{j,r} p_{j,i}^{(r)} = e_{i,r}. \tag{3.1}$$

Observe for $1 \leq r \leq R$ that $(e_{i,r} : 1 \leq i \leq M)$ is an invariant measure for the Markov chain of the jump chain of a class r customer.

Theorem 3.2.2 gives the general solution to the balance equations. The theorem and its proof originates from [Baskett et al., 1975].

Theorem 3.2.2. *For a closed queueing network with service centers exhibiting the FIFO server discipline, it holds that the invariant distribution $\pi = (\pi_{\mathtt{x}} : \mathtt{x} \in \chi)$ is given by*

$$\pi_{\mathtt{x}} = C \prod_{i=1}^{N} \prod_{k=1}^{n_i(\mathtt{x})} \frac{e_{i,x_{i,k}}}{\min\{C_i, k\}\,\mu_i}, \tag{3.2}$$

where C is a normalising constant such that $\sum_{\mathtt{x} \in \chi} \pi_{\mathtt{x}} = 1$.

Proof. The theorem will be proved by checking the balance equations.

As mentioned in Section 3.2.5, it holds for a state $\mathtt{x} \in \chi$ that the only non-zero transition rates $q_{\mathtt{xy}}$ correspond to the transition of exactly one customer moving to another service center. This gives reason for defining

$$F_{\mathtt{x}} \coloneqq \left\{(k,i,j) \in \mathbb{N}^3 \ : \ 1 \leq k \leq \min\left\{C_i, n_i(\mathtt{x})\right\}\right\}, \quad \mathtt{x} \in \chi$$

to be the set of all non-zero probability transitions from state $\mathtt{x}$. Here $(k,i,j) \in F_{\mathtt{x}}$ corresponds to a customer at the k-th position of service center i to move to service center j. Notice how k must be less than both C_i and n_i, because those are the only customers that are served by service center i when the system is in state $\mathtt{x}$.

Now for $\mathtt{x} \in \chi$ and $(k,i,j) \in F_{\mathtt{x}}$ let $\mathtt{y} \in \chi$ correspond to the new state of the system when the k-th customer of service center i moves to service center i. Notice that we have

$$q_{\mathtt{xy}} = p_{i,j}^{(\mathtt{x}_{i,k})} \mu_i.$$

The left hand side of the balance equations can now be calculated by summing over all elements of $F_{\mathtt{x}}$ and for $\mathtt{x} \in \chi$ we obtain

$$\begin{aligned}
\pi_{\mathtt{x}} \sum_{\mathtt{y} \in \chi} q_{\mathtt{xy}} &= \pi_{\mathtt{x}} \sum_{i=1}^{M} \sum_{k=1}^{\min\{C_i, n_i(\mathtt{x})\}} \sum_{j=1}^{M} \mu_i p_{i,j}^{(x_{i,k})} \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \mu_i \sum_{k=1}^{\min\{C_i, n_i(\mathtt{x})\}} \sum_{j=1}^{M} p_{i,j}^{(x_{i,k})} \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \mu_i \sum_{k=1}^{\min\{C_i, n_i(\mathtt{x})\}} 1 \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \mu_i \min\left\{C_i, n_i(\mathtt{x})\right\},
\end{aligned}$$

which corresponds to the total flow out of center i.

Similarly to $F_{\mathtt{x}}$, we can define

$$T_{\mathtt{x}} \coloneqq \left\{(k,j,i) \in \mathbb{N}^3 \ : \ 1 \leq k \leq \min\left\{C_j, n_j + 1\right\}, n_i(\mathtt{x}) \geq 1\right\},$$

as the set of all non-zero probability transitions that result in state $\mathtt{x}$. Here (k,j,i) corresponds to the k-th customer of service center j moving to service center i such that the resulting state is $\mathtt{x}$. Notice how $n_i(\mathtt{x}) \geq 1$ is a necessary condition, because a customer cannot move from center j to center i in order to arrive at state $\mathtt{x}$ if there is no customer at center i in state $\mathtt{x}$.

Now for $\mathtt{x} \in \chi$ and $(k,j,i) \in T_{\mathtt{x}}$ let $\mathtt{y} \in \chi$ correspond to the state such that the k-th customer moving center j to center i results in state $\mathtt{x}$. Then

$$q_{\mathtt{yx}} = p_{j,i}^{(r)} \mu_j,$$

and notice that in state $\mathtt{y}$ there is one more customer in center j and one less in center i, thus

$$\pi_{\mathtt{y}} = \pi_{\mathtt{x}} \frac{e_{j,r}}{\min\{C_j, n_j(\mathtt{x})+1\}\mu_j} \frac{1}{\frac{e_{i,r}}{\min\{C_i,n_i(\mathtt{x})\}\mu_i}} = \pi_{\mathtt{x}} \frac{e_{j,r}}{\min\{C_j, n_j(\mathtt{x})+1\}\mu_j} \frac{\min\{C_i, n_i(\mathtt{x})\}\mu_i}{e_i, r}.$$

The right hand side of the balance equations can now be calculated by summing over all elements of $T_{\mathtt{x}}$ and for $\mathtt{x} \in \chi$ we obtain

$$\begin{aligned}
\sum_{\mathtt{y}\in\chi} \pi_{\mathtt{y}} q_{\mathtt{yx}} &= \pi_{\mathtt{x}} \sum_{i=1, n_i(\mathtt{x})\geq 1}^{M} \sum_{j=1}^{M} \sum_{k=1}^{\min\{C_j, n_j(\mathtt{x})+1\}} \frac{e_{j,x_{i,n_i(\mathtt{x})}}}{\min\{C_j, n_j(\mathtt{x})+1\}\mu_j} \frac{\min\{C_i, n_i(\mathtt{x})\}\mu_i}{e_i, r} p_{j,i}^{(x_{i,n_i(\mathtt{x})})} \mu_j \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \frac{\min\{C_i, n_i(\mathtt{x})\}\mu_i}{e_{i,x_{i,n_i(\mathtt{x})}}} \sum_{j=1}^{M} \frac{e_{j,x_{i,n_i(\mathtt{x})}}}{\min\{C_j, n_j(\mathtt{x})+1\}} p_{j,i}^{(x_{i,n_i(\mathtt{x})})} \sum_{k=1}^{\min\{C_j, n_j(\mathtt{x})+1\}} 1 \\
&= \pi_x \sum_{i=1}^{M} \frac{\min\{C_i, n_i(\mathtt{x})\}\mu_i}{e_{i,x_{i,n_i(\mathtt{x})}}} \sum_{j=1}^{M} e_{j,x_{i,n_i(\mathtt{x})}} p_{j,i}^{(x_{i,n_i(\mathtt{x})})} \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \frac{\min\{C_i, n_i(\mathtt{x})\mu_i\}}{e_{i,x_{i,n_i(\mathtt{x})}}} e_{i,x_{i,n_i(\mathtt{x})}} \\
&= \pi_{\mathtt{x}} \sum_{i=1}^{M} \min\{C_i, n_i(\mathtt{x})\}\mu_i,
\end{aligned}$$

which is equal to the left hand side of the balance equations. This implies that $\pi = (\pi_{\mathtt{x}} : \mathtt{x} \in \chi)$ is a invariant distribution. □

The invariant distribution of Equation 3.2 enables us to calculate the equilibrium state probabilities of the aggregate states $\mathcal{N}$. Theorem 3.2.3 gives these probabilities and originates from [Baskett et al., 1975].

Theorem 3.2.3. *The equilibrium state probabilities for $\mathfrak{n} \in \mathcal{N}$ are given by*

$$\pi_{\mathfrak{n}} = \left[\prod_{i=1}^{N} \binom{n_i}{n_{i,1}, \ldots, n_{i,R}} \right] \pi_{\mathtt{x}},$$

for any $\pi_{\mathtt{x}} \in \chi$ that gives rise to the aggregate state $\mathfrak{n}$. The constant C is the constant from Theorem 3.2.2.

Proof. For $\mathfrak{n} \in \mathcal{N}$ define $\chi_{\mathfrak{n}}$ as the set of states that give rise to the aggregate state $\mathfrak{n}$. Formally, this is defined as

$$\psi_{\mathfrak{n}} := \left\{ \mathtt{x} \in \chi \ : \ \sum_{k=1}^{n_i(\mathfrak{n})} \delta_{r,x_{i,k}} = n_{i,r}, 1 \leq i \leq M, 1 \leq r \leq R \right\}.$$

Notice how

$$|\chi_{\mathfrak{n}}| = \prod_{i=1}^{M} \binom{n_i(\mathfrak{n})}{n_{i,1}, \ldots, n_{i,R}},$$

because only the number of customers of each class at service station i is known, which means that there are

$$\binom{n_i(\mathfrak{n})}{n_{i,1}, \ldots, n_{i,R}}$$

different queues possible at service station i. Also notice for $\mathtt{x} \in \chi_{\mathfrak{n}}$ and service center i that

$$\prod_{k=1}^{n_i(\mathfrak{n})} e_{i,x_{i,k}} = \prod_{r=1}^{R} e_{i,n_{i,r}},$$

only depends on $\mathfrak{n}$ and not on the exact element within $\chi_{\mathfrak{n}}$. This implies that $\pi_{\mathtt{x}}$ is the same for all $\mathtt{x} \in \chi_{\mathfrak{n}}$. We can use this to obtain an equilibrium state probability of

$$\pi_{\mathfrak{n}} = \sum_{\mathtt{x} \in \chi_{\mathfrak{n}}} \pi_{\mathtt{x}} = |\chi_{\mathfrak{n}}| \, \pi_{\mathtt{x}} = \left[\prod_{i=1}^{M} \binom{n_i(\mathfrak{n})}{n_{i,1}, \ldots, n_{i,R}} \right] \pi_{\mathtt{x}},$$

as desired. □

4. Model description and analysis

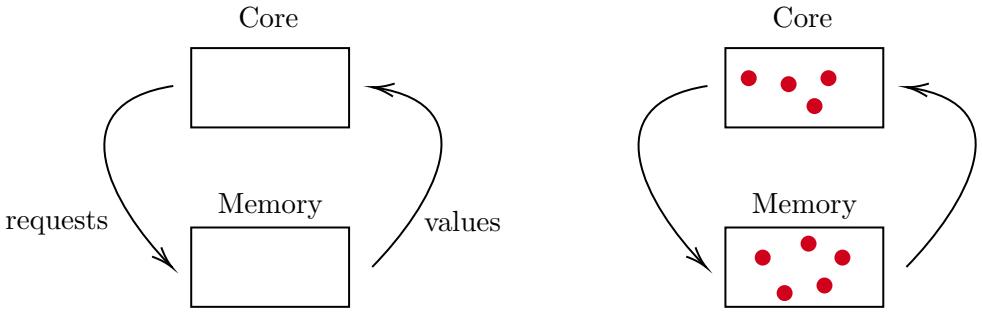
The queueing theory from Chapter 3 can now be used to create a model which imitates the dynamics of the processor as described in Chapter 2. Section 4.1.1 describes the model and the approach taken to arrive at the model, while Section 4.2 analyses the model and Section 4.3 presents derivations for the model parameters.

Note that the model is highly abstract and only aims at recreating the processor dynamics. This can give a non-trivial relation between the model parameters and the components and processes in the processor.

4.1. Model description and explanation

4.1.1. The model dynamics overview

The most relevant interaction in the processor is the sending of memory requests by the core to the memory. There are two types of memory requests: local and DRAM requests. The local requests require the memory inside the core, and thus won't get effected by an increase in the number of tasks run in parallel. Therefore, those requests won't be considered in the model. The DRAM requests require the DRAM, which is the shared memory between the cores. The service rate of these requests is effected by the other tasks due to the finite memory bandwidth. This gives reason to consider those requests in the model. The dynamic induced by those requests is visualised in Figure 4.1a.

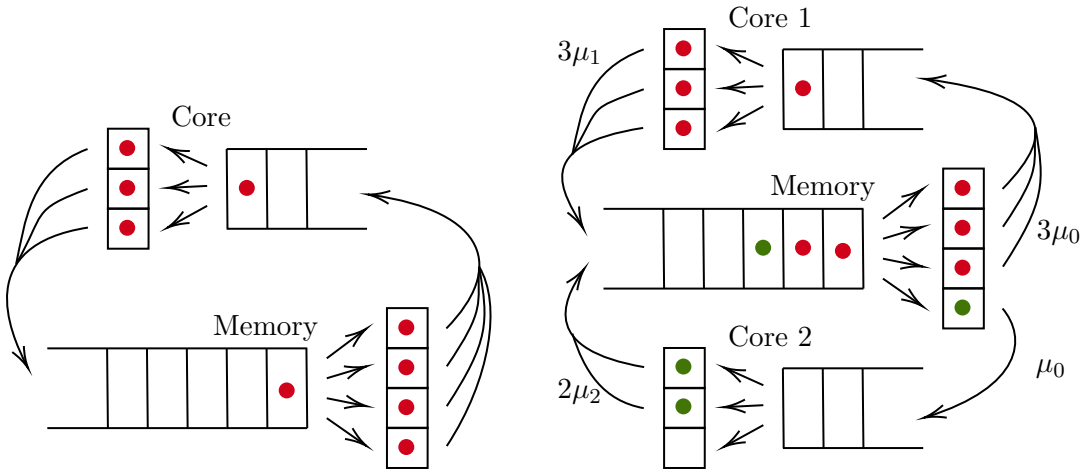


(a) The core-memory interaction. (b) Resource requesting tokens in the system.

Figure 4.1.: The system dynamics modelled with resource requesting tokens travelling through the system, where the number of tokens correspond to the degree of parallelizability of a task.

However, there is a much more useful way of perceiving the core-memory interaction. This is done by viewing a task in execution as a task that requests resources from the

different components of the system. The amount of attention a task wants at once is determined by the parallelizability of a task. A task with a high degree of parallelizability is able to perform many different computations and requests many different resources at once. This degree of parallelizability can be modelled with memory requests travelling through the network. A memory request in the memory does actually correspond with a real life memory request, while the interpretation of memory requests in the core is not trivial. The network is visualised in Figure 4.1b and it gives an abstract framework of reasoning about task execution and its interference with the memory.



(a) Single core.

(b) Two cores.

Figure 4.2.: The core and memory depicted as queues serving tokens in a system with one and two cores.

The model can be further concretized by realising that both the core and memory can only serve a finite number of tokens at once. This can be incorporated into the model by replacing the core and memory rectangles in Figure 4.1b with a queue to obtain Figure 4.2a. The resulting network is a closed queueing network and multiple cores can easily added to the network. The case of two cores is given in Figure 4.2b. The formal definition of the model is given in Definition 4.1.1, where service center 0 corresponds to the memory and centers $1 \leq i \leq M$ correspond to the cores.

Definition 4.1.1. *The M-core single memory model is defined as the triple $(\mathbf{N_M}, \mu, C)$, where*

$$\mathbf{N_M} = (N_i \in \mathbb{N} : 1 \leq i \leq M)$$

is the population vector,

$$\mu = (\mu_i : 0 \leq i \leq M)\,,$$

is the service rate vector and

$$C = (C_i : 0 \leq i \leq M)$$

is the capacity vector. There are $R = M$ classes of customers and

$$p_{i,0}^{(i)} = p_{0,i}^{(i)} = 1, \quad 1 \leq i \leq M.$$

4.2. Model analysis

The theorems from Chapter 3 can now be used to get an expression for the terms $e_{i,r}$ and the invariant distribution. For the terms $e_{i,r}$ and $1 \leq r \leq M$ it must hold that

$$e_{i,r} = \sum_{j=0}^{M} e_{j,r} p_{j,i}^{(r)} = \begin{cases} \sum_{j=1}^{M} e_{j,r} & : i = 0 \\ e_{0,r} & : i \neq 0 \end{cases},$$

which means that we can simply let $e_{0,i} = 1$ and $e_{i,i} = 1$ for all $1 \leq i \leq M$, and all other terms are equal to 0. This directly implies that any $\mathtt{x} \in \chi$ with $x_{i,k} \neq i$ for $1 \leq i \leq M$ has 0 probability of occurring in equilibrium, because then $e_{i,x_{i,k}} = 0$. In other words, in equilibrium there are only class i customers in core i, which should be the case. We can use this to obtain for $\mathtt{x} \in \chi$ with $x_{i,k} = i$ for all $1 \leq i \leq M$ that

$$\begin{aligned} \pi_{\mathtt{x}} &= C \prod_{i=0}^{M} \prod_{k=1}^{n_i} \frac{e_{i,x_{i,k}}}{\min\{C_i, k\}\,\mu_i} \\ &= C \prod_{i=0}^{M} \prod_{k=1}^{n_i} \frac{1}{\min\{C_i, k\}\,\mu_i} \\ &= C \prod_{i=0}^{M} \frac{1}{\min\{C_i, n_i\}! C_i^{\min\{C_i - n_i, 0\}} \mu_i^{n_i}}. \end{aligned}$$

Furthermore, for the equilibrium state probabilities of the aggregate state $\mathfrak{n} \in \mathcal{N}$ we obtain

$$\begin{aligned} \pi_{\mathfrak{n}} &= C \left[\prod_{i=1}^{M} \frac{n_i!}{\prod_{r=1}^{M} (n_{i,r}!)} \right] \prod_{i=0}^{M} \frac{1}{\min\{C_i, n_i\}! C_i^{\min\{C_i - n_i, 0\}} \mu_i^{n_i}} \\ &= C \frac{n_0!}{\prod_{r=1}^{M} (n_{0,r}!)} \prod_{i=0}^{M} \frac{1}{\min\{C_i, n_i\}! C_i^{\min\{C_i - n_i, 0\}} \mu_i^{n_i}} \\ &= C \frac{n_0!}{\prod_{r=1}^{M} ((N_r - n_r)!)} \prod_{i=0}^{M} \frac{1}{\min\{C_i, n_i\}! C_i^{\min\{C_i - n_i, 0\}} \mu_i^{n_i}}, \end{aligned}$$

where we used the fact that only class r customers are in service center r for $1 \leq r \leq N$. This implies that

$$n_{0,r} = N_r - n_r, \quad 1 \leq r \leq N$$

The constant C can then be calculated by summing over all $\mathfrak{n} \in \mathcal{N}$. Notice that those states are completely determined by $n_1, n_2, \ldots, n_M$. This means we have

$$C = \left[\sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} \cdots \sum_{n_N=0}^{N_M} \frac{n_0!}{\prod_{r=1}^{M} ((N_r - n_r)!)} \prod_{i=0}^{M} \frac{1}{\min\{C_i, n_i\}! C_i^{\min\{C_i - n_i, 0\}} \mu_i^{n_i}} \right]^{-1},$$

where

$$n_0 = \sum_{r=1}^{M} N_r - n_r.$$

4.3. Model parameter derivation

This section concerns the execution of a single application and the data it generates. The obtained execution data of an application can then be used to determine the parameters of the model. Note that an application might invoke multiple cores, thus let M be the number of cores that the application invokes. The values to be determined are then N_i, μ_i, C_i for $1 \leq i \leq M$ and μ_0, C_0.

4.3.1. The memory parameters

The `sim.cfg` file is the system configuration file that the HotSniper simulation software uses. It contains information about the DRAM memory such as the DRAM service time. This service time must be equal to the expected service time in the model of $\frac{1}{\mu_0}$, which means that

$$\mu_0 = \frac{1}{\text{DRAM service time}}.$$

There are then two different ways of computing the memory capacity C_0.

1. The first approach concerns using the `sim.cfg` file. This file contains the
   - number of DRAM memory controllers;
   - bandwidth in B/s of each DRAM memory controller;
   - average service time of a request.

   This information, together with the number of bytes a request receives, can be used to estimate the bandwidth of the DRAM by

   $$C_0 \approx \frac{\text{total bandwidth}}{\text{number of received bytes of a request}} \times \text{average service time},$$

   where

   $$\text{total bandwidth} = \text{number of controllers} \times \text{bandwidth of a controller}.$$

2. The second approach can be done by simulating an application. For each core, HotSniper will return the number of requests, utilisation of the DRAM and response time of the core. This can be used to estimate the DRAM bandwidth by

$$C_0 \approx \frac{\text{average number of requests per second}}{\text{utilisation}} \times \text{average service time},$$

where

$$\text{average number of requests per second} = \frac{\text{total number of requests}}{\text{response time}}.$$

4.3.2. The core and task parameters

Due to the fact that the core doesn't serve memory requests in real life, it is certainly not trivial how to derive the values of N_i, μ_i, C_i for $1 \leq i \leq M$. The most important restriction on the parameters is that the total number of memory requests visits must be equal to the total number of DRAM requests that the core sends within the execution of a task. We will call this the **total memory visit criteria**. We will start by stating the intuition behind the different parameters.

- For core i the value of μ_i relative to μ_0 relates to the proportion of time that a memory request spends in the core and memory. As an example, if μ_i is very big relative to μ_0, then you are expected in equilibrium to see almost all memory requests in the memory. The value of μ_i can thus be used to satisfy the **total memory visit criteria**.
- For core i the value of N_i is the number of requests in the systems and thus can be used to satisfy the **total memory visit criteria**.
- For core i the value of C_i determines the rate at which requests can be send back to the memory and thus can also be used to satisfy the **total memory visit criteria**.

Notice how all three parameters can be used to satisfy the **total memory visit criteria**. This might indicate that we have two degrees of freedom that can be used to make sure that some other execution data, such as the utilisation or queueing delay, aligns with the model.

There are now a few approaches that can be taken to estimate the parameters.

1. If we only care about the **total memory visit criteria** we can make the value of C_i and μ_i constant, i.e. $C_i = C_1$, $\mu_i = \mu_1$ for all $1 \leq i \leq M$. This might not even be an unrealistic approach, because the hardware of each core is the same. We can then set N_i equal to total number of DRAM requests. We can then set C_i to an arbitrary value and choose μ_i such that we expect each request only to visit the memory once. This approach makes sure the **total memory visit criteria** is satisfied.
2. We can also use the CPI stack and utilisation to create two more criteria.

5. Model accurateness and shortcomings

5.1. Task simulation results

5.2. Model parameters and parallel simulation predictions

5.3. Comparisons

6. Conclusion

Bibliography

[Baskett et al., 1975] Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G. (1975). Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2):248–260.

Populair summary

A. Proofs