Offload (Only) the Right Jobs: Robust Offloading Using the Markov Decision Processes

Esa Hyytiä
Aalto University, Finland

Thrasyvoulos Spyropoulos
EURECOM, Sophia-Antipolis, France

Jörg Ott
Aalto University, Finland

***Abstract*—We consider a dynamic offloading problem arising in the context of mobile cloud computing (MCC). In MCC, three types of tasks can be identified: (i) those which can be processed only locally in a mobile device, (ii) those which are processed in the cloud, and (iii) those which can be processed either in the mobile or in the cloud. For type (iii) tasks, it is of interest to consider when they should be processed locally and when in the cloud. Furthermore, for both type (ii) and (iii) tasks, there is typically two ways to access the cloud: via a (costly) cellular connection or via intermittently available WLAN hotspots. The optimal strategy involves multi-dimensional considerations such as the availability of WLAN hotspots, energy consumption, communication costs and the expected delays. We approach this challenging problem in the framework of Markov decision processes and derive a near-optimal offloading policy.**

I. Introduction

We consider the task assignment problem arising in the context of the mobile cloud computing (MCC). In MCC, the key idea is to *offload* computationally demanding tasks to the cloud, process them there, and then transmit the results back to a mobile device [1], [2], [3] This can be faster and also consume less energy in the mobile device. The cloud could be accessed using the cellular network, or via a WLAN hotspot. The latter option is only intermittently available, but can offer significant advantages in terms of cost and energy [1]. The offloading scenario for a mobile user is illustrated in Fig. 1.

In MCC, three types of tasks can be identified: (i) those which can be processed only locally in a mobile device, (ii) those which are processed in the cloud, and (iii) those which can be processed either in the mobile or in the cloud. A first interesting question is when flexible type (iii) tasks should be processed locally and when in the cloud. Furthermore, for type (ii) and type (iii) tasks, there may exist multiple access links, with different characteristics (e.g., WLAN, LTE, 3G) to use for accessing the cloud. This leads to multi-dimensional considerations where, energy consumption, transmission costs, task type (e.g., background or foreground), network availability and rates, must be taken into account.

We approach this problem in the framework of Markov decision processes (MDP). The different options (process locally, or offload to a cloud, either at a hotspot or via a cellular network) are modelled as single server queues. The resulting dynamic decision problem is a specific type of a task assignment problem, where the policy seeks to minimize a given cost function (e.g., the mean response time). In contrast to the more common setting, in our case the servers corresponding to (WLAN) hotspots are available only intermittently due to the assumed mobility of a user. Moreover, some options can be expensive to use. For example, service time (but not the queueing delay) in a cellular connection can cost actual money, in particular, if the user is roaming abroad. Therefore, often well-functioning task assignment policies such as the join-the-shortest-queue (JSQ) and the least-work-left (LWL) may make sub-optimal decisions leading to poor performance. In contrast, the policies developed in this paper take into account the actual cost factors, the current state of the system (backlogs in different queues and the currently available networks), and the anticipated future tasks when making the decisions.

First we derive a new theoretical result, the so-called value function, for an M/G/1-queue with intermittently available server. The intermittently available server corresponds to the WLAN hotspots: tasks offloaded via WLAN may need to wait until the "server" appears again. Then, the value function derived for this particular type of M/G/1-queue, together with value functions for the other two options (local processing, offloading to the cloud through the cellular network), are used to apply the policy improvement step of the MDP framework. This yields a robust and efficient offloading policies that take into account the intermittent presence of WLAN hotspots.

In addition to the offloading of computational tasks, mobile offloading is also seen as a promising candidate to deal with the rapidly increasing traffic volumes (see, e.g. [4], [5], [6]), as well as, for backup and synchronization traffic (e.g. Dropbox & Flicker) [7]. Typically the offloading mechanisms are heuristic and chosen with a particular scenario in mind. [8] proposes a simple model to study computation and transmission energy tradeoffs, while [9] formulates an optimization problem involving various costs. Neither work considers queueing effects. Lee et al., in [6], develop a similar queueing model for

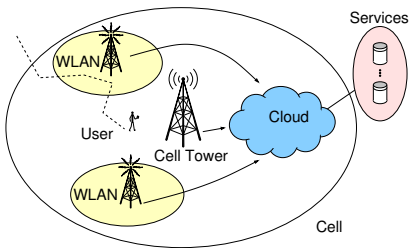


Fig. 1. Offloading in mobile cloud computing. The cloud services can be accessed via a cellular network or intermittently available WLAN hotspots.

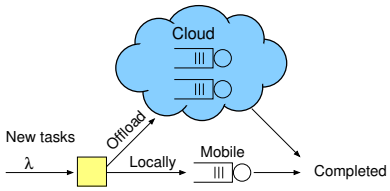


Fig. 2. Offloading in mobile cloud computing: some tasks can be processed either locally in a mobile device or in the cloud, where the latter involves also transmission costs in terms of energy, delay and money.

the intermittently available server to analyze the gain from offloading. However, no policy optimization is carried out.

The main contributions of this paper are as follows:

1) We analyze the offloading problem in MCC and propose a multi-queue model that captures the essential elements.
2) We derive the size-aware value function for M/G/1-queue with an *intermittently available server*.
3) We give new state- and cost-aware offloading policies that take into account the mobility, and the existing and anticipated tasks. By means of numerical examples, we further demonstrate that significant gains are available.

The rest of the paper is organized as follows. Section II introduces the model and notation. The value function for M/G/1 with intermittently available server is derived in Section III. The new offloading policies are given and evaluated in Section IV, and Section V concludes the paper.

II. Model and Notation

Fig. 2 illustrates the basic setting, where three performance metrics are important to us: (i) energy consumption at the mobile device, (ii) monetary costs (from the access network), and (iii) the delay until the results are available for the user (i.e., the response time). Should a transfer of a content be carried out in the vicinity of WLAN hotspots, the transfer time can vary a lot depending on if an WLAN connection is currently available or not, whereas the use of a cellular connection is often slower and costs money. The cloud itself is assumed to be scalable and thus free from capacity concerns.

A. Task and Network Types

Different types of tasks emerge in a mobile device according to some process, which we, for simplicity, assume to be a Poisson process with rate λ. These tasks can be classified into the following three classes:

(i) **Local-only:** A fraction p_m are unconditionally processed locally in the mobile device, e.g. because transferring the relevant information would take more time and energy than when processed locally or because these tasks must access local components (e.g. sensors, user interface, etc.) [1]. Local processing consumes the CPU power of the device and, in particular, the battery power. We assume that jobs processed locally are served in the *first-come-first-server* (FCFS) order and for each task we know the expected service time (depends on the task and the CPU). There are no communication costs or delays.

(ii) **Cloud-only:** A fraction p_c must always be processed in the cloud, e.g. because necessary data or the software is only present in the cloud. Backup or synchronization of files and images (e.g. Dropbox, Google+, etc.) could also be seen as such an "offloading task". The user can still decide which network to use for offloading.

(iii) **Flexible:** A fraction $p = 1 - p_m - p_c$ are flexible tasks that can be processed either in a mobile device or in the cloud. The device can dynamically decide where they are processed based on the urgency of the task, available networks and the state of the system (e.g., the current backlogs). Many tasks fall in this category, and the offloading decision depends on whether the communication costs balance the local processing costs [8].

All "cloud-only" tasks as well as some of the "flexible" ones, will need to access the cloud over some wireless interface. We assume that the following options are available:

(a) **Cellular network:** When accessing the cloud via a cellular network, there is queueing due to the transmission speed of the cellular link. The service time consists of the upload time, cloud processing time, and the download time. For simplicity, we lump these three phases together and model them as a single FCFS server. Costs arise in terms of transmission delays (queueing and actual transmission times), energy and money costs related only to the transmission and reception (but not to processing).

(b) **WLAN:** WLAN is similar with two distinctive features. First, the communication costs are generally lower (or even zero). Second, the hotspot availability is intermittent and the offloading may encounter long random delays until the device arrives to a vicinity of the next hotspot. This intermittent availability of hotspots corresponds to a FCFS queue with a server that disappears occasionally. We assume that the sojourn time in a hotspot and the time between two hotspots are exponentially distributed with parameters ν_A and ν_D. However, this assumption is relaxed in numerical examples. (The cellular connection and local CPU processing are always available.)

B. Cost Structure and Objective

We take the users' perspective and aim at maximizing his or her satisfaction. This involves energy consumption (unless the device is currently connected to a charger), monetary costs, and the delay. We further assume a *time scale separation* so that each task takes a short time when compared to a typical recharging cycle. The dynamic optimization problem is to strike a balance between the different costs factors and the perceived user experience in terms of delay (a.k.a., latency and response time). This boils down to two kinds of dynamic task assignment decisions as illustrated in Fig. 3: (i) choosing the cloud and the access link, or local processing for the flexible tasks, and (ii) choosing the cloud access link for the cloud-only tasks. The intermittent availability of the WLAN hotspots further complicates the problem.

We have different costs depending on what kind of action a queue models (local CPU, or cloud via WLAN/cellular):

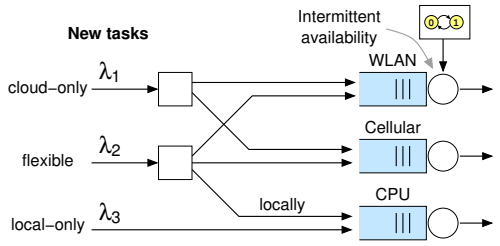


Fig. 3. Model: some tasks can be processed only in cloud, some locally and some are flexible.

- **Waiting cost** in queue while server is ON or OFF, which is just a delay. No energy or money is spent while waiting, and this cost factor is also independent of the interface. The same holding cost applies also for the service time, as the user does not care where the delay occurs, just about the time it takes to complete a task.
- **Energy cost**, measured per bit, spent only when transmitting or receiving a task to/from a cloud. This cost depends on the interface. An energy cost is present also when a task is processed locally.
- **Monetary cost**, also measured per bit and spent only when transmitting or receiving a task to/from a cloud. This can be assumed to be zero for WLAN and some positive constant for the cellular access.

The complete model, illustrated in Fig. 3, is as follows:

- For each new task, the system must take an *action* among the available options (e.g., offload task i via WLAN).
- Each task j has a task-specific *holding cost* h_j, which characterizes its urgency (e.g., emails may have a lower priority than a response to the user activity).
- Cost parameters of job j for each action i are known:
  i) Energy consumption rate $e_{j,i}$,
  ii) Transmission cost rate $c_{j,i}$, and
  iii) Service time $s_{j,i}$ (excluding queueing delays).
- We are also aware of the full state, i.e., the files under transmission via different access links and the current state of the local processing queue, including the remaining service times of the jobs currently receiving service.
- The queues are assumed to operate independently at constant rate according to FCFS scheduling discipline.
- The objective is to minimize the costs the system incurs,

$$\min \sum_{j=1}^{n} \left((c_{j,d(j)} + e_{j,d(j)}) S_j + h_j T_j \right),$$

where $d(j)$ is the decision for job j, S_j is its service time and T_j the latency, i.e., the sum of the waiting time and service time, $T_j = W_j + S_j$.

Note that the unit holding cost, $H = 1$, corresponds to minimization of the mean latency. In general, with this cost structure each job, after the assignment, has a certain holding cost while waiting, h_W, and another while receiving service, h_S. We utilize this convention later in Section III-D. For simplicity, we assume that $S_j = s_{j,d(j)}$, i.e., the actual service time is known exactly for each possible action. Similarly, in the size-aware setting with FCFS, also the tentative waiting time W_j is known exactly for each decision (later arriving tasks will not cause any additional delays to the present tasks). That is, we know exactly the *immediate cost* of each action (waiting and service time of the new task), while the consequences on later arriving tasks have to be also taken into account.

III. Analysis

In this section, we consider a single size-aware M/G/1-FCFS queue, where the (remaining) service times of the existing jobs are known. Our aim is to derive the *value function*, which characterizes the expected difference in costs a system initially in state $\mathbf{z}$ incurs when compared to a system initially in equilibrium. The single queue results are later utilized to evaluate the costs from assigning a new task to different queues in the offloading setting depicted in Fig. 3.

A. Value function and policy improvement

The notion of value functions is a fundamental element in solving MDP problems [10], [11], [12]. Formally, let $V_{\mathbf{z}}(t)$ denote the costs a system incurs during time $(0, t)$ when the system is initially in some state denoted by $\mathbf{z}$. Obviously, many systems incur costs at every time step and $\lim_{t\to\infty} \mathrm{E}[V_{\mathbf{z}}(t)] = \infty$. However, for ergodic systems the expected difference in costs incurred between two initial states, $\mathbf{z}_1$ and $\mathbf{z}_2$, is finite,

$$\lim_{t\to\infty} \mathrm{E}[V_{\mathbf{z}_1}(t) - V_{\mathbf{z}_2}(t)] < \infty.$$

Instead of choosing an arbitrary state (such as $\mathbf{z}_2$ in above) as a reference state, it is customary to compare the costs incurred from a given state $\mathbf{z}$ to the long-run average costs, leading to a formal definition of the value function (without discounting)

$$v_{\mathbf{z}} \triangleq \lim_{t\to\infty} \mathrm{E}[V_{\mathbf{z}}(t) - rt],$$

where r denotes the (long-run) mean cost rate. Both the value function and the mean cost rate depend on the policy, $v_{\mathbf{z}} = v_{\mathbf{z}}(\alpha)$ and $r = r(\alpha)$.

The standard application of the value function is the policy improvement. Suppose that we have determined $v_{\mathbf{z}}(\alpha_0)$ for a basic policy α_0. At some state, we have n possible actions (e.g., n possible ways to process a new task), having immediate costs x_i and resulting states $\mathbf{z}_i$, $i = 1, \ldots, n$. The n actions can be evaluated by $x_i + v_{\mathbf{z}_i}$, as

$$\overbrace{(x_i - x_j)}^{=\text{immediate}} + \overbrace{(v_{\mathbf{z}_i} - v_{\mathbf{z}_j})}^{=\text{future}},$$

clearly describes the relative expected cost of action i in comparison to action j. The policy improvement step defines a new policy α by choosing the action i^* that minimizes the expected long-term costs,

$$i^* = \operatorname*{argmin}_i \, x_i + v_{\mathbf{z}_i}.$$

That is, for the current action one deviates from the basic policy α_0, while the consecutive actions are according to it so

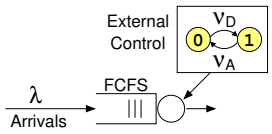


Fig. 4. M/G/1-FCFS queue with an external Markov ON/OFF process controlling the availability of the server.

that $v_{\mathbf{z}}(\alpha_0)$ can be utilized. The constant offset in the value function is immaterial, and equally

$$\operatorname*{argmin}_i x_i + (v_{\mathbf{z}_i} - v_0),$$

where v_0 denotes the value of an empty system. In policy iteration, one repeats the same steps for the new policy α iteratively until the policy converges and an optimal policy has been obtained.[1] For more details on the value functions and MDP in general, we refer to [10], [11], [12], and on the use in the task assignment setting, see e.g., [13], [14], [15].

B. M/G/1-FCFS with intermittently available server

The value function for the size-aware M/G/1-FCFS queue with an always available server has been derived in [15].

In this section, we consider the M/G/1-FCFS queue where the availability of the *server* is governed by an external ON/OFF process. In our context, the queue corresponding to offloading via WLAN is served only when the mobile device is in the vicinity of a WLAN hotspot, as illustrated in Fig. 1.

More specifically, the server is either in ON-state processing the existing jobs in FCFS order (if any), or in OFF-state during which no job gets service. The server's activity cycle is assumed to follow an external Markov ON/OFF process with independent activity and inactivity periods (A_i, D_i), both obeying exponential distributions with parameters ν_A and ν_D, respectively. When a server returns, it resumes to serve the customer whose service was interrupted (if any), i.e., the work already conducted is not lost (cf. file transfers with resume).

The system is illustrated in Fig. 4. Hence, the fraction of time the server is available to process jobs is

$$\mathrm{P}\{\text{available}\} = \frac{\nu_D}{\nu_A + \nu_D}.$$

As $\nu_D \to \infty$, the availability of the server tends to 1.

1) Busy periods: A busy period begins when a job arrives to an empty system, and ends when the system becomes empty again. When a busy period ends, the server is obviously unconditionally active. Let q denote the probability that the server is active when a new busy period starts. Due to the memoryless property of the Markov processes, we have

$$q = \frac{\lambda}{\lambda + \nu_A} + \frac{\nu_A}{\lambda + \nu_A} \cdot \frac{\nu_D}{\lambda + \nu_D} \cdot q,$$

which gives

$$q = \frac{\lambda + \nu_D}{\lambda + \nu_A + \nu_D}. \tag{1}$$

[1]To be exact, this holds, e.g., for ergodic systems with finite state-spaces, whereas with continuous state-space the situation is more complicated.

The mean remaining busy period in a work-conserving M/G/1-queue is given by

$$\frac{u}{1-\rho}, \tag{2}$$

where u is the initial backlog and $\rho = \lambda \mathrm{E}[S]$ the offered load with $\mathrm{E}[S]$ denoting the mean service time. We can utilize (2) to give an expression for the mean busy period in a system with the intermittently available server as follows. First, we associate the interruptions occurring during the service time of job i to be part of the corresponding job,

$$Y_i = X_i + D_1^{(i)} + \ldots + D_{N_i}^{(i)},$$

where $N_i \sim \mathrm{Poisson}(\nu_A X_i)$ and $D_j^{(i)} \sim \mathrm{Exp}(\nu_D)$. Thus, with the chain rule of expectation one obtains

$$\mathrm{E}[Y] = \left(1 + \frac{\nu_A}{\nu_D}\right) \mathrm{E}[X],$$
$$\mathrm{E}[Y^2] = \left(1 + \frac{\nu_A}{\nu_D}\right)^2 \mathrm{E}[X^2] + \frac{2\nu_A}{\nu_D^2} \mathrm{E}[X].$$

In this sense, the offered load *during* the busy period is

$$\rho^* \triangleq \lambda \mathrm{E}[Y] = \lambda \mathrm{E}[X](1 + \nu_A/\nu_D).$$

For stability $\rho^* < 1$ is assumed. The mean busy period can be obtained by utilizing (2), which gives the mean remaining busy period in an M/G/1-queue with an initial backlog of u:

$$\mathrm{E}[B] = q \cdot \frac{\mathrm{E}[Y]}{1-\rho^*} + (1-q) \cdot \frac{\mathrm{E}[Y+D]}{1-\rho^*}$$
$$= \frac{\mathrm{E}[Y]}{1-\rho^*} + \frac{1}{1-\rho^*} \cdot \frac{\nu_A/\nu_D}{\lambda + \nu_A + \nu_D} = \frac{\mathrm{E}[Y] + \beta}{1-\rho^*},$$

where β denotes the mean time until the server becomes available at the start of the busy period,

$$\beta = \frac{\nu_A/\nu_D}{\lambda + \nu_A + \nu_D}.$$

2) Mean delay: The mean delay in an ordinary M/G/1-queue is given by the Pollaczek-Khinchine formula,

$$\mathrm{E}[T] = \frac{\lambda \mathrm{E}[S^2]}{2(1-\rho)} + \mathrm{E}[S]. \tag{3}$$

The following holds for the intermittently available server:

Theorem 1: The mean delay in an M/G/1-queue with intermittently available server is given by

$$\mathrm{E}[T] = \frac{\lambda \mathrm{E}[Y^2]}{2(1-\rho^*)} + \frac{\mathrm{E}[Y] + \beta(1 + \lambda(\mathrm{E}[Y] + 1/\nu_D))}{1 + \lambda\beta}. \tag{4}$$

Proof: In an M/G/1-queue, where the first customer of a busy period receives *exceptional service*, the mean delay is [16]

$$\mathrm{E}[T] = \frac{\lambda \mathrm{E}[S^2]}{2(1-\rho)} + \frac{2\mathrm{E}[S^*] + \lambda(\mathrm{E}[(S^*)^2] - \mathrm{E}[S^2])}{2(1 + \lambda(\mathrm{E}[S^*] - \mathrm{E}[S]))}, \tag{5}$$

where S^* denotes the service time of the first customer, S the service time of the other customers, and $\rho = \lambda \mathrm{E}[S]$. Our system is a special case of the above, where $S = Y$ and

$$S^* = Y^* = \begin{cases} Y, & \text{with probability of } q, \\ Y + D, & \text{otherwise.} \end{cases}$$

The first two moments of Y^* are

$$\begin{aligned}\mathrm{E}[Y^*] &= \mathrm{E}[Y] + \frac{\nu_A/\nu_D}{\lambda+\nu_A+\nu_D} = \mathrm{E}[Y] + \beta,\\ \mathrm{E}[(Y^*)^2] &= q\,\mathrm{E}[Y^2] + (1-q)\,\mathrm{E}[(Y+D)^2]\\ &= \mathrm{E}[Y^2] + 2\frac{\nu_A/\nu_D}{\lambda+\nu_A+\nu_D}\left(\mathrm{E}[Y]+\frac{1}{\nu_D}\right)\\ &= \mathrm{E}[Y^2] + 2\beta\left(\mathrm{E}[Y]+\frac{1}{\nu_D}\right).\end{aligned}$$

Substituting these into (5) gives (4). □

Note that at the limit $\nu_D \to \infty$, we have $\beta \to 0$ and (4) reduces to the Pollaczek-Khinchine mean value formula (3). Another interesting limit is obtained when both $\nu_A, \nu_D \to \infty$ in such way that $\nu_D/(\nu_A+\nu_A)$, corresponding to the fraction of time the server is available, is finite (cf. duty cycle).

C. Size-aware value function

As mentioned, we assume a *size-aware* system, where the service times of the jobs become known upon arrival. Let Δ_i denote the (remaining) service time of job i, which are ordered so that job 1 (if any) receives first service, then job 2, etc. We also know the state of the server, i.e., whether it is available or not, but not how long each time period is going to last. The parameters of the availability process, (ν_A, ν_D), however, are known. The state of the system is denoted by vector $\mathbf{z}$,

$$\mathbf{z} = (\Delta_1, \ldots, \Delta_n; e),$$

where $e \in \{0, 1\}$ indicates if the server is available or not.

Our aim is to derive the size-aware value function with respect to delay. The mean number of jobs served during a busy period in an ordinary M/G/1-queue is [17]

$$\mathrm{E}[N_B] = \frac{1}{1-\rho}. \tag{6}$$

Utilizing the above, it is easy to show by conditioning that the mean number of jobs served during a busy period that starts with a job having a service time of x is

$$\mathrm{E}[N_B \mid X_1 = x] = 1 + \frac{\lambda x}{1-\rho}. \tag{7}$$

Now we are ready to state our main theorem:

Theorem 2: The value function with respect to delay in an M/G/1-queue with intermittently available server is

$$\begin{aligned}v_{\mathbf{z}} - v_0 = \frac{\lambda\gamma^2 u^2}{2(1-\rho^*)} + 1_{\mathrm{off}}\frac{\lambda\gamma u}{(1-\rho^*)\nu_D}\\ + \gamma\sum_{i=1}^{n}(n+1-i)\Delta_i + 1_{\mathrm{off}}\frac{n}{\nu_D},\end{aligned} \tag{8}$$

where $\gamma = 1+\nu_A/\nu_D$ and $\rho^* = \lambda\,\gamma\,\mathrm{E}[X]$, and $1_{\mathrm{off}} = 1$ if the server is not currently available ($e = 0$) and zero otherwise.

Proof: See Appendix. □

We note that the terms with 1_{off} factor correspond to the additional penalty due to the currently unavailable server. Note also that the reference state v_0 corresponds to an empty system with server initially in the same state as in $\mathbf{z}$. This is sufficient for task assignment as inserting new jobs to a queue does not change the state of the server, the availability of which was governed by an external independent ON/OFF process. The parameter $\gamma = 1+\nu_A/\nu_D$ can be seen as the effective mean service rate, as γu corresponds to the mean time to process a backlog of u (given the server is initially available).

The value function (8) is with respect to the delay. Next we will consider how much it costs to add a new job to a queue, i.e., we give an expression for the admission cost. We also generalize the cost structure to arbitrary and separate holding costs that apply for the waiting time and service time.

D. Admission costs and general cost structure

Next we consider how much it costs in terms of additional delay all later arriving jobs concurrently incur if a job with size x is admitted to the queue. The cost difference with respect to delay is simply the difference in the relative values,

$$a(x; \mathbf{z}) = v_{\mathbf{z}\oplus x} - v_{\mathbf{z}}, \tag{9}$$

where $\mathbf{z}$ denotes the initial state and $\mathbf{z} \oplus x$ the state after including a new job with size x,

$$\mathbf{z} \oplus x = (\Delta_1, \ldots, \Delta_n, x; e).$$

Corollary 1: The admission cost of a job with size x with respect to delay in an M/G/1-queue with intermittently available server is

$$a(x; u) = \frac{\lambda\gamma x}{1-\rho^*}\left(\frac{\gamma(2u+x)}{2} + \frac{1_{\mathrm{off}}}{\nu_D}\right) + \gamma(u+x) + \frac{1_{\mathrm{off}}}{\nu_D}. \tag{10}$$

Proof: With FCFS, the admission cost depends only on the current backlog $u = \Delta_1 + \ldots + \Delta_n$. Substituting (8) to (9) yields (10). □

Note that $\gamma \to 1$ and $\rho^* \to \rho$ when $\nu_D \to \infty$, and the above reduces to the admission cost in an ordinary M/G/1 [15],

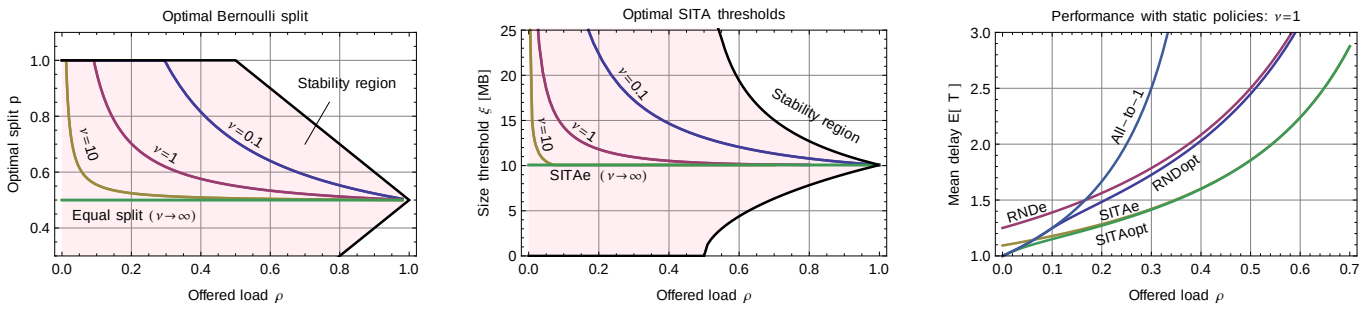
$$a(x; u) = \frac{\lambda x}{2(1-\rho)}(2u+x) + u + x.$$

The above value function is with respect to delay, corresponding to unit holding cost during both the waiting and service time. A generalization to arbitrary and separate holding costs for the waiting and service time is straightforward. This is important for us as it **allows specifying energy, delay, and money costs per task** once the assignment decision has been done and, e.g., the cost per bit transmitted gets fixed (operators charge only for the bits transmitted).

Corollary 2: The admission cost of a job with size x and holding cost rates (h_W, h_S) for the waiting and service time to an M/G/1-queue with an intermittently available server is

$$\begin{aligned}a(x; u) = \frac{\lambda\,\mathrm{E}[H_W]\gamma x}{(1-\rho^*)}\left(\frac{\gamma(2u+x)}{2} + \frac{1_{\mathrm{off}}}{\nu_D}\right)\\ + h_W\left(\gamma(u+x) - x + \frac{1_{\mathrm{off}}}{\nu_D}\right) + h_S\,x.\end{aligned} \tag{11}$$

where $\mathrm{E}[H_W]$ is the mean waiting time holding cost rate for jobs arriving to the queue.



(a) Optimal RND split (b) Optimal SITA threshold (c) Mean delay (in minutes) with static policies

Fig. 5. Example #1: Server 1 always available with rate $r_1 = 100\,\text{kB/s}$; Server 2 is intermittently available with $\nu_A = \nu_D$ (i.e., 50%) and $r_2 = 200\text{kB/s}$.

Proof: First we note that the first two terms in (10) correspond to the additional delay the later arriving jobs experience on average if the new job is admitted to the queue. With non-preemptive scheduling disciplines, the service times of the jobs are not affected by the state of the system (at any moment), and thus the first two terms (10) must correspond to the increase in the waiting time, which we simply multiply by the mean holding cost while waiting, $\mathrm{E}[H_W]$. The costs the given (x, h_W, h_S)-job incurs on average is

$$(u(1+\nu_A/\nu_D) + x(\nu_A/\nu_D) + 1_{\text{off}}/\nu_D)\, h_W + x h_S.$$

Combining these yields (11). □

The first term corresponds to the mean additional costs the later arriving jobs will incur (while waiting), and the last two terms to the costs the given new job incurs on average. The same generalization to job- and state-specific (waiting or in service) holding cost rates can be done also for the value function (8), which we omit here for brevity.

The obtained result (11) is satisfactory in many respect. First, the expression is compact and depends only on few parameters: the new task, (x, h_W, h_S), the state of the server (available or not) and the current backlog u. The rest are constant, i.e., (11) is clearly easy to compute in online fashion. Second, (11) is *insensitive* to the distribution of service time X and depends only on its mean. Third, considering (11) as a conditional admission cost on condition that $U = u$, we note that the expected admission cost is *also insensitive* to the distribution of U and depends only on its mean $\mathrm{E}[U]$. The insensitivity is a desirable property, e.g., towards the *robustness and predictability* of a control algorithm [18].

IV. Dynamic Mobile Data Offloading

In this section, we utilize the just derived value functions and admission costs to derive sophisticated dynamic policies for the offloading problem. The general approach is as follows:

1) We start with a static policy α_0 (actions are made independently of the past decisions and the state of the queues). Note that the actions of α_0 may still depend on the parameters of a task (e.g., size or holding costs).
2) Given a Poisson arrival process, the static α_0 feeds each server jobs also according to a Poisson process, the system of parallel queues *decomposes*, and the sum of queue specific value functions gives the value function for the whole system, $v_{\mathbf{z}} = \sum_i v^{(i)}_{\mathbf{z}(i)}$.
3) Consequently, we can carry out the policy improvement step and simply choose for each task the queue with the smallest admission cost using (11),

$$\alpha_1 : \qquad \operatorname*{argmin}_j a_j((x, h_W, h_S); (u_j, e_j)),$$

where (x, h_W, h_S) in general also depend on Queue j (e.g., the service time depends on the link speed when offloading a task to a cloud), and (u_j, e_j) denote the (relevant) state information of Queue j.

We refer to the the improved policy as FPI (first-policy-iteration). The FPI policy is no longer static but dynamic, and in practice it is often also near optimal (given the starting point was a reasonable). Let us next illustrate the concept and different heuristic policies with two numerical examples.

A. Example #1: Cellular vs. WLAN

Consider first an elementary system of two servers:

- Cellular: always present; service rate $r = 100\text{kB/s}$
- WLAN: intermittently available, $\nu_A{=}\nu_D{=}\nu$; $r{=}200\text{kB/s}$

The mean rate of both servers is thus equal. Jobs are large files, $X \sim \mathrm{Exp}(\mu)$ with the mean size of $1/\mu = 6\text{MB}$

The optimal Bernoulli split between the two queues, determined using (4), is depicted in Fig. 5(a) for $\nu = 0.1, 1, 10$ min^{-1}. Parameter p on the y-axis corresponds to the fraction of jobs assigned to Server 1. For very small λ, it is advantageous to assign all jobs to Server 1 ($p = 1$). As λ increases, it first becomes advantageous and later necessary ($\rho \geq 0.5$) to utilize both servers ($p < 1$). When $\nu \to \infty$, the duty cycle in Server 2 becomes infinitely small and the two servers are equivalent.

The size-interval-task-assignment (SITA) policy assigns jobs shorter than threshold ξ to Cellular, and the rest to WLAN [19], [20]. This way, the short jobs are segregated from the long ones and processed in the "faster" server (cf. SRPT). Hence, SITA controls both the load and the type of jobs each server receives. With SITAe, the threshold is chosen so that both servers receive equal load; $\xi_e \approx 10\,\text{MB}$. The optimal threshold for SITA can be determined using (4), and we refer to that policy as SITAopt. Fig. 5(b) depicts these for $\nu = 0.1, 1, 10$ min^{-1}. Observations are similar as with RND.

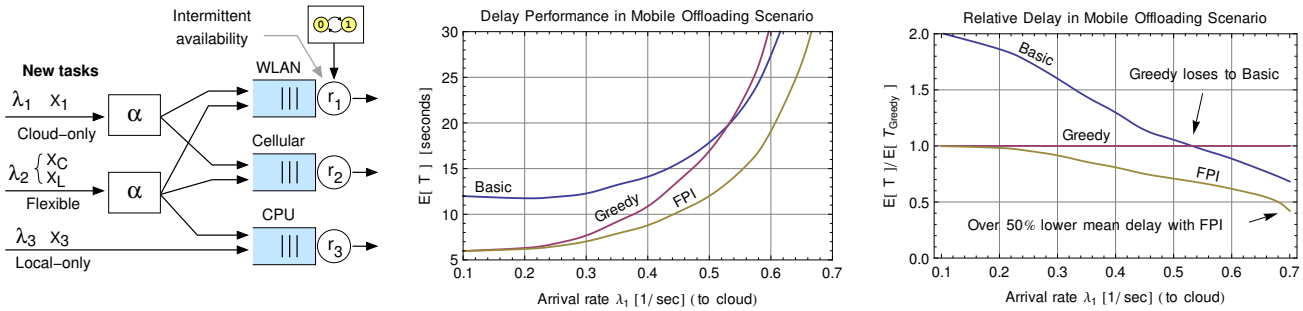


Fig. 7. Example #2: WLAN is available 50% of the time ($\nu^{-1} = 40$ sec) and has service rate $r_1 = 200$ kB/s. Cellular network has service rate $r_2 = 100$ kB/s and the CPU is the "slowest" with $r_3 = 50$ k ticks/s. The mean delay is depicted for fixed $\lambda_2 = \lambda_3 = 0.1$ [1/sec]. Offloading by FPI yields significant gains.

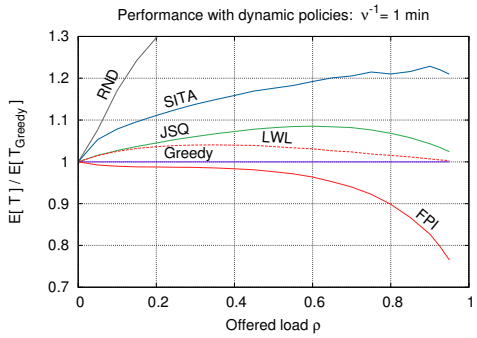


Fig. 6. Relative mean delay with dynamic policies. FPI outperforms all other policies even in this very elementary setting. $\nu^{-1} = 1$ min.

Fig. 5(c) depicts the mean delay (in minutes) with the different policies. All-to-1 sends all jobs to Server 1 and it is unstable if $\rho \geq 0.5$. We observe that the SITA policies are strong and the difference between them is negligible when $\rho > 0.1$ in this case. (It is the optimal static policy for ordinary FCFS servers [21].)

Let us then consider the following dynamic policies:

- JSQ chooses the queue with the least number of jobs.
- LWL chooses the queue with the shortest backlog.
- Greedy is the same as LWL, but includes the availability.
- FPI based on the SITAe.

Possible ties are broken in favor of Server 1. The numerical results are depicted in Fig. 6. On the x-axis is the offered load ρ, and the y-axis corresponds to the mean delay relative to the mean delay with Greedy. We can make two observations: (i) the optimal static policies are inferior to dynamic policies, and (ii) FPI, the only policy that *consciously* takes into account the anticipated future arrivals, shows a superior performance.

Experimental studies [5], [6] suggest that both ON and OFF periods tend to be *heavy-tailed*. Obviously, everything depends on the mobility pattern of the particular user. Therefore, we also run the simulations when the ON/OFF periods obey bounded Pareto (with $\alpha = 0.6$) and uniform distributions. In all cases, FPI turned out to be similarly superior to others, suggesting that the obtained **FPI policy is relatively insensitive to the shapes of the ON/OFF distributions**.

B. Example #2: Offloading vs. local processing

Let us next consider the scenario depicted in Fig. 7 (left). The rate of WLAN and cellular are 200kB/s and 100kB/s, respectively, whereas the rate of CPU is 50k "ticks"/s. The always offloaded tasks $X_1 \sim \text{Exp}(\mu_1)$ with mean $1/\mu_1 = 250$kB arrive at rate λ_1. Flexible jobs (X_C, X_L) arrive at rate λ_2, where X_C denotes the size if offloaded (bytes), and X_L if processed locally (ticks). We assume that X_C and X_L are i.i.d., $X_C \sim X_L \sim \text{Exp}(\mu_2)$ with $1/\mu_2 = 500$k. Jobs that CPU must process arrive at rate λ_3 and are uniformly distributed, $X_3 \sim \text{U}(0, 500\,\text{k})$ (ticks). Then $1/\nu_A = 1/\nu_D = 40$ sec (i.e., a vehicular setting) and the objective is to minimize the mean delay. We consider the following policies:

- **Basic** policy comprises two heuristic decisions. First, the flexible jobs are processed locally if $X_C > X_L$, and otherwise offloaded to the cloud. Offloaded tasks are split with SITAe: short jobs are transmitted via a cellular network, and long ones via WLAN hotspots.
- **Greedy** policy chooses the action which minimizes the expected delay of the given task. The current system state and tentative service times are utilized in myopic fashion.
- **FPI** policy is obtained by computing the parameters of the arrival process to each queue with the above basic policy, and then using (10) to evaluate each action.

As Basic is static, its mean delay can be computed analytically using the Pollaczek-Khinchine formula and (4). The other two must be evaluated by means of simulations.

We set $1/\lambda_2 = 1/\lambda_3 = 10$ sec, and vary λ_1. With these, Basic is stable when $\lambda_1 < 0.75$. Fig. 7 (middle) depicts the mean delay (in seconds) as a function of λ_1, and Fig. 7 (right) the relative mean delay when compared to Greedy. Initially, both Greedy and FPI offer a low mean delay, as expected. As the load increases, Greedy starts to suffer from its *short-sighted* decisions, and eventually it falls even behind Basic. The fact that Basic focuses on minimizing the workload (the $\min$ operation with flexibile jobs) pays off under a heavy load. In contrast, FPI shows superior performance and outperforms the other two especially when the load is high: the mean delay with Greedy is more than two times longer than with FPI.

It is clearly a non-trivial task to dynamically decide which tasks to process locally and which to offload. It seems that the **FPI based policy does a very good job also in this respect**.

V. Conclusions and Future Work

In this paper, we have developed a stochastic model to study the dynamic offloading in the context of MCC. The model captures various performance metrics and also the intermittently available access links (WLAN hotspots). The problem itself is non-trivial and to facilitate the analysis, we have, e.g., assumed that a job set to be offloaded via WLAN hotspot cannot be later re-assigned to cellular. The analysis is carried out in the MDP framework. First we derived a new queueing theoretic result, which enables derivation of near-optimal dynamic offloading policies that take into account the different performance metrics (delay, energy, offloading costs), current state of the queues, and the anticipated future tasks. In the preliminary numerical experiments, the resulting policy gave very good results, it was robust to shape of distributions defining the lengths of the ON/OFF durations, and outperformed other policies by a significant margin. The gain is expected to be even higher with more diverse cost structures, file size distributions etc.

The approach is robust and, e.g., scales to an arbitrary number of offloading channels. The future work includes experiments in more realistic scenarios, where, e.g., energy consumption and transmission costs of cellular data plan are explicitly included. Moreover, we plan to investigate the possibility to develop schemes with jockeying, where it is possible to re-assign tasks later on. This is useful, e.g., when a user unexpectedly arrives to the vicinity of a hotspot.

References

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. of ACM MobiSys*, 2010.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. of EuroSys '11*, 2011.

[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of IEEE INFOCOM*, 2012.

[4] A. J. Nicholson and B. D. Noble, "Breadcrumbs: forecasting mobile connectivity," in *ACM MobiCom'08*, 2008, pp. 46–57.

[5] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3G using WiFi," in *ACM MobiSys'10*, 2010, pp. 209–222.

[6] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: How much can wifi deliver?" *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 536–550, 2013.

[7] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *Proc. of IEEE INFOCOM*, 2013.

[8] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, 2010.

[9] Y. Im, C. J. Wong, S. Ha, S. Sen, T. Kwon, and M. Chiang, "AMUSE: Empowering users for cost-aware offloading with throughput-delay tradeoffs," in *Proc. of IEEE Infocom Mini-conference*, 2013.

[10] R. Bellman, *Dynamic programming*. Princeton University Press, 1957.

[11] R. A. Howard, *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. Wiley Interscience, 1971.

[12] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[13] K. R. Krishnan, "Joining the right queue: a state-dependent decision rule," *IEEE Transactions on Automatic Control*, vol. 35, no. 1, 1990.

[14] ——, "Markov decision algorithms for dynamic routing," *IEEE Communications Magazine*, pp. 66–69, Oct. 1990.

[15] E. Hyytiä, A. Penttinen, and S. Aalto, "Size- and state-aware dispatching problem with queue-specific job sizes," *European Journal of Operational Research*, vol. 217, no. 2, pp. 357–370, Mar. 2012.

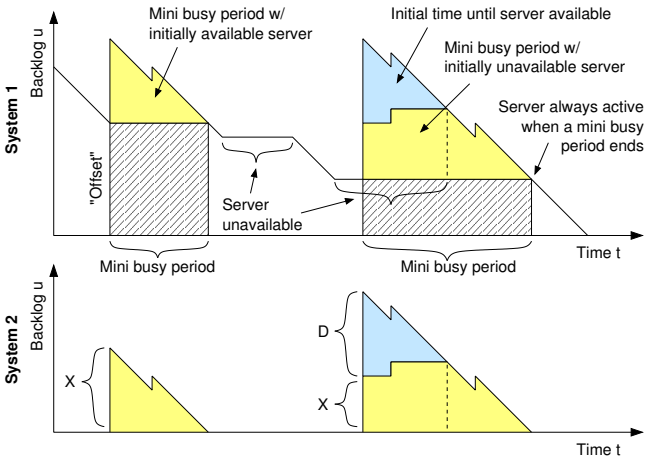


Fig. 8. Derivation of the value function by analyzing mini busy periods.

[16] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, 4th ed. John Wiley & Sons, 2008.

[17] L. Kleinrock, *Queueing Systems, Volume I: Theory*. Wiley, 1975.

[18] T. Bonald and A. Proutière, "Insensitivity in processor-sharing networks," *Perform. Eval.*, vol. 49, no. 1-4, pp. 193–209, Sep. 2002.

[19] M. E. Crovella, M. Harchol-Balter, and C. D. Murta, "Task assignment in a distributed system: Improving performance by unbalancing load," in *SIGMETRICS'98*, Madison, Wisconsin, USA, Jun. 1998, pp. 268–269.

[20] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[21] H. Feng, V. Misra, and D. Rubenstein, "Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems," *Performance Evaluation*, vol. 62, no. 1-4, pp. 475–492, 2005.

Appendix

Proof of Theorem 2

Proof: Fig. 8 illustrates a sample realization when the server is initially available, which happens with the probability of q, see (1). The upper System 1 has an initial backlog of u, and the lower System 2 is initially empty. The idea is that by assuming the same arrival sequences and server availabilities for both systems, we can deduce the mean difference in costs incurred until System 1 becomes empty, after which the two systems will behave identically. In what follows, we utilize the lack of memory property of Markov processes repeatedly without explicitly mentioning it.

Suppose a new job with size x arrives when the backlog in System 1 is u. This causes the backlog to jump from u to $u+x$. We refer to the time period until the backlog in System 1 is again at level u as a *mini busy period*. During the other time periods, i.e., outside the mini busy periods, the "original" backlog is served and we refer to this as *the main process*. First we observe that two types of mini busy periods can emerge: (i) those where the server is initially available (Type 1), and (ii) those where it is not (Type 2). Fig. 9 illustrates a high-level view on System 1. The two states on the left, active and unavailable, correspond to the main process, and the possible mini busy periods are on the right. We note that when a mini busy period ends, the server is always in active state.

Next we refer to Fig. 8, and observe that the durations of the mini busy periods and the new jobs involved are identical in both systems, and in fact a mini busy period in System 1 corresponds to a busy period in System 2. The only important difference is that the new jobs arriving during a (mini) busy period experience an additional delay in System 1 equal to the amount of backlog there was in System 1 at the beginning of the mini busy period (see the "offset" in Fig 8).

The number of Type 1 mini busy periods obeys Poisson process with mean λu. Associating the unavailability periods to the job currently under service, the remaining busy period is equivalent to one in an ordinary M/G/1-queue with service time Y. In particular, the mean number of jobs served during such a busy period is given by (6)

$$\mathrm{E}[N_{B1}] = \frac{1}{1-\rho^*}.$$

The cost difference these jobs experience between the two systems is equal to the offset indicated in Fig. 8. These offsets are independently and uniformly distributed on $(0, u)$ (cf. Poisson arrival process), i.e., the average additional delay in System 1 for each mini busy period is $\gamma\frac{u}{2}$, where $\gamma = 1+\nu_A/\nu_D$ is the constant factor by which the nominal mean service time $\mathrm{E}[X]$ is multiplied to include also the interruptions occurring during the service of the job, $\mathrm{E}[Y] = \gamma\,\mathrm{E}[X]$. Combining these gives the additional delay System 1 incurs in total on average during the Type 1 mini busy periods until System 1 becomes empty,

$$d_1 = \lambda u \cdot \frac{1}{1-\rho^*} \cdot \gamma\frac{u}{2} = \frac{\lambda\gamma u^2}{2(1-\rho^*)}.$$

Consider next the time periods when the server in the "main process" is unavailable (i.e., excluding the mini busy periods, see Fig. 8). The key observation here is that these time periods end according to a Poisson process with rate $\lambda + \nu_D$; either a new job arrives and a mini busy period starts, or the server becomes available. In particular, at the end of Type 2 mini busy period the server is also available, and the unavailable time period has thereby ended, as illustrated in Fig. 9.

The number of times the server in the "main process" becomes unavailable obeys Poisson distribution with mean $\nu_A u$. With probability of $\lambda/(\lambda+\nu_D)$ these time periods change into the mini busy periods of Type 2. Thus, the mean number of Type 2 mini busy periods is

$$\nu_A u \cdot \frac{\lambda}{\lambda+\nu_D}.$$

With Type 2 mini busy periods, we can associate the remaining time until the server becomes available to the job that started the mini busy period, corresponding to an effective service time of $Y + D$, where $D \sim \mathrm{Exp}(\nu_D)$. With aid of (7), it follows that the mean number of jobs served during a Type 2 mini busy period is

$$\mathrm{E}[N_{B2}] = 1 + \frac{\lambda\,\mathrm{E}[Y+D]}{1-\rho^*} = \frac{1+\lambda/\nu_D}{1-\rho^*}.$$

The mean additional delay in System 1 (the offsets) are statistically the same as earlier, $\gamma u/2$, for each mini busy period. Thus the mean additional delay incurred during the Type 2 mini busy periods before System 1 becomes empty is again the product of the above: the mean number of Type 2 mini busy periods, times the mean number of jobs in each of them, times the average additional delay for jobs in System 1 in each mini busy period,

$$d_2 = \nu_A u \cdot \frac{\lambda}{\lambda+\nu_D} \cdot \frac{1+\lambda/\nu_D}{1-\rho^*} \cdot \gamma\frac{u}{2} = \frac{\lambda\gamma u^2}{2(1-\rho^*)}\nu_A/\nu_D.$$

The average additional delay the later arriving jobs experience in System 1 is then

$$d_1 + d_2 = \frac{\lambda\gamma^2 u^2}{2(1-\rho^*)}, \tag{12}$$

Additionally, we have the n jobs present only in System 1. The average delay they incur is

$$d_3 = \gamma\,(n\Delta_1 + (n-1)\Delta_2 + \ldots + \Delta_n) = \gamma\sum_{i=1}^{n}(n+1-i)\Delta_i. \tag{13}$$

Combining (12) and (13) gives the difference in the value function between state $\mathbf{z}$ and an empty system, $v_{\mathbf{z}} - v_0$, for the states where the server was initially available.

The case where the server is not initially available requires only two minor correction terms. If the server becomes available before a next job arrives, the remaining behavior is identical to what we just discussed with Type mini busy periods. However, with probability of $\lambda/(\lambda+\nu_D)$, the initial period when the server is unavailable ends to a specific (additional) Type 2 mini busy period. The offset for this mini busy period is u, since the server has been unavailable from the start. Thus the additional delay incurred in System 1 during this specific mini busy period is on average

$$\frac{\lambda}{\lambda+\nu_D} \cdot \gamma u \cdot \frac{1+\lambda/\nu_D}{1-\rho^*} = \frac{\lambda\gamma u}{(1-\rho^*)\nu_D}.$$

Similarly, the present n jobs experience on average an additional delay of n/ν_D before the server becomes available for the first time, after which the situation is the same as with Type 1 and given by (13). Adding these two gives the additional mean delay incurred if the server is initially unavailable,

$$d_4 = 1_{\mathrm{off}}\left(\frac{\lambda\gamma u}{(1-\rho^*)\nu_D} + \frac{n}{\nu_D}\right).$$

The sum $d_1 + d_2 + d_3 + d_4$ then gives the desired result. □

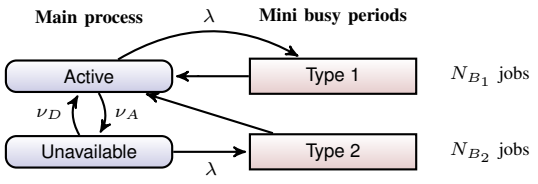


Fig. 9. State diagram illustrating how both Type 1 and Type 2 mini busy periods end with the server being available and active. The total time the main process spends in the "Active" state is u.