Markovian queue with garbage collection

Illés Horváth[1], István Finta[2], Ferenc Kovács[2], András Mészáros[3], Roland Molontay[4], and Krisztián Varga[2]

[1] MTA-BME Information Systems Research Group, Hungary
[2] Nokia, Bell Labs
[3] Budapest University of Technology and Economics
Department of Networked Systems and Services
[4] Budapest University of Technology and Economics
Department of Stochastics

Abstract. Garbage collection is a fundamental component of memory management in several software frameworks. We present a general two-dimensional Markovian model of a queue with garbage collection where the input process is Markov-modulated and the memory consumption can be modeled with discretisation. We derive important performance measures (also including garbage collection-related measures like mean garbage collection cycle length). The model is validated via measurements from a real-life data processing pipeline.
Keywords: memory management, garbage collection, stochastic modelling, Markovian modelling.

1 Introduction

Some of the most popular languages such as Java and C# require efficient memory management including garbage collection (GC): the automated process of identifying and recovering the storage space that is occupied by objects that are no longer required. The physical memory is a limited resource so designing more sophisticated garbage collectors and providing a theoretical framework are of particular research interest.

A huge number of different garbage collection techniques have been proposed throughout the years. For surveys and evaluation of garbage collection algorithms we refer the reader to [30], [12] and [23]. More recent GC techniques include Garbage First for multi-processors with large memory [10], Metronome, a real-time GC integrated with the scheduling system [4], MMTk, a memory management toolkit for Java [6], the concurrent-copy collector, a real time garbage collector for Java [25], and FeGC, an efficient GC scheme for flash memory based storage systems [14]. Virtual machine garbage collection optimization was addressed in [5]; the model shares the main idea with the present paper (discretisation of the memory), with an overall simpler, essentially 1-dimensional model with focus on optimising parameters for garbage collection.

A more recent tendency has been to consider formal models therefore provide a rigorous method to characterize GC algorithms and analyze their performance

independently of the programming system. In particular, analytical modeling of garbage collection algorithms in flash-based solid-state drive (SSD) systems has received significant research interest [28], [16], [31], [29], [9].

The majority of the analytical studies on garbage collection process have focused on the following specific algorithms: greedy GC, FIFO GC, Windowed GC, d-choices GC. For more details, see [28], [31].

An important issue regarding garbage collection in SSD systems is the so-called write amplification phenomenon. For more details, see [31].

A number of analytical frameworks have been proposed by the abstraction of the block state space and the stochastic modeling of the selection process. In [9] a Markov chain model is provided to characterize the performance of SSD operation for uniformly distributed random small user writes and considering the greedy scheme. They find that write amplification increases as the system occupancy increases as the number of pages per block increases but decreases as the number of block increases. In [16] a Markov chain model is employed to capture the dynamics of large-scale SSDs, and mean-field theory is applied to derive the asymptotic steady state, the performance/durability trade-off of GC algorithms is analyzed. Yang et al. also apply mean field analysis and show that the system dynamics can be represented by a system of ordinary differential equations and the steady state of the write amplification can be predicted for a class of GC algorithms (including d-choices) [31].

Another modeling approach is providing a theoretical framework of distributed garbage collection [17], [22], [8]. The increasing use of distributed systems implies that distributed garbage collectors should be considered. A formal model of distributed garbage collection is Surf [8] that can describe a wide range of GCs and is amenable to rigorous analysis.

In our work, we focus on a Markovian approach that models of the effect of garbage collection on memory management. We present the model in two steps. In Section 2.1, we present a 2-dimensional Markov-modulated fluid description of the model. The fluid approach is easy to define but difficult to solve analytically. Then we present the corresponding Markovian model in Section 2.2, which is essentially a discretisation of the memory level. The Markovian model can be solved efficiently numerically, with the analysis and performance measures derived in Section 3.

Section 4 contains an application to an actual data processing system. The model of Section 2.2 is then validated by comparison to performance measurements of the actual system.

2 Queue with garbage collection

In the model, data arrives at a server and is stored in the memory. When it is processed, it does not flush (empty) immediately from the memory, but is only flushed when the memory reaches a certain level.

Memory level is described by two variables: in-use memory (V) and junk memory (U). In-use memory contains all data that has not been processed yet

(that is, the queue), while junk memory contains data that has been processed since the last GC period. Data processing may generate extra memory usage; we assume that this extra memory usage is proportional to the size of the data with multiplicative constant C (that is, processing 1 byte of data creates a total of C bytes of memory usage in addition to the original 1 byte). We also make the assumption that the service time of data is proportional to the amount of data; this assumption means that V is proportional to the service queue length. These assumptions typically hold for systems with relatively simple processing.

When the total memory $U+V$ reaches a certain level M, GC turns on. During GC, the junk memory flushes at a fixed rate g, but data may keep arriving (and stored entirely in in-use memory). For simplicity, we assume that there is no service during GC. When GC finishes, service is resumed.

We assume that arrivals are Markov-modulated with a finite state space S and generator Q. The arrival process itself is denoted by $X(t)$. The arrival rate in state $i \in S$ is r_i, and the service rate is constant s.

In Section 2.1, we present a fluid approach to model the memory level. While the model definition is relatively straightforward and tidy from the behaviour of the system, it leads to a 2-dimensional fluid queue with special behaviour on the boundaries.

2.1 Fluid description

Fluid modeling approach is an efficient way of describing and analyzing a wide range of real systems for domains as diverse as job scheduling [20] and battery life [13]. An overview of the basic concepts of fluid models with the potential usage in performance analysis can be found in [11].

A fluid description of the queue is obtained when data is assumed to be continuous; in this case, U (junk memory level) and V (queue) are fluid variables governed by the arrival process (a continuous time Markov chain) and the switch between service and GC modes.

The behaviour of the system is governed by the equations

$$\left.\begin{array}{l} \mathrm{d}U(t)/\mathrm{d}t = Cs \\ \mathrm{d}V(t)/\mathrm{d}t = r_{X(t)} - s \end{array}\right\} \text{ if } V(t) > 0 \text{ during service}$$

$$\left.\begin{array}{l} \mathrm{d}U(t)/\mathrm{d}t = Cs \\ \mathrm{d}V(t)/\mathrm{d}t = r_{X(t)} - s \end{array}\right\} \text{ if } V(t) = 0 \text{ and } r_{X(t)} > s \text{ during service}$$

$$\left.\begin{array}{l} \mathrm{d}U(t)/\mathrm{d}t = Cr_{X(t)} \\ \mathrm{d}V(t)/\mathrm{d}t = 0 \end{array}\right\} \text{ if } V(t) = 0 \text{ and } r_{X(t)} < s \text{ during service} \qquad (1)$$

$$\left.\begin{array}{l} \mathrm{d}U(t)/\mathrm{d}t = -g \\ \mathrm{d}V(t)/\mathrm{d}t = r_{X(t)} \end{array}\right\} \text{ during GC}$$

and the forced transitions:

- when $U(t) + V(t)$ reaches M during service, we switch to GC mode;
- when $U(t)$ reaches 0 during GC, we switch to service mode.

In (1), $V(t) = 0$ corresponds to no queue; if $r_{X(t)<s}$, all incoming data is processed immediately, while if $r_{X(t)>s}$, the queue starts growing. As long as $V(t) > 0$, the server is working at a full service rate. During garbage collection, there is no service, so all incoming data goes in the queue.

The above system is difficult to solve analytically. For 2-dimensional fluid queues, very few results available. Instead, we present a discretised Markovian version of the model in Section 2.2 where U and V are both discretised; stationary analysis of the Markovian model is carried out in Section 3. A detailed analysis of the original fluid model is subject to further research.

The Markov model of Section 2.2 is applied to a data processing application in Section 4 with the performance measures predicted by the model compared with measurements from the real system. We note that only some of the measures derived in Section 3 are measured in the application. We nevertheless included these and other measures as well in Section 3, with possible different future applications in mind.

2.2 Markovian description

Markovian queuing theory is a well-established topic with diversified domains of application. For a detailed introduction to queuing theory with computer science and telecommunication applications we refer the reader to [7], [27] and [19].

In this approach, we replace the fluid queues U and V by a discrete memory level to obtain a Markovian model with a discrete state space. We note that we allow U and V to be discretised with different granularity; assume the possible values of U are divided into N_U different sections, while the possible values of V are divided into N_V different sections. The reason to allow a different granularity lies in the fact that the behaviour of the system depends highly on whether $V = 0$ or $V > 0$: as long as $V > 0$ (there is a queue), the system will work at full capacity. Thus it makes sense to select N_V relatively high in order to be able to identify $V > 0$ more precisely. Since the exact value of U is less relevant in the behaviour of the system (apart from the total memory reaching M), the granularity of U may be allowed to be less fine. For simplicity, we assume N_V is an integer multiple of N_U.

The maximal possible memory level M corresponds to a full memory, while the value 0 corresponds to an empty memory that contains no data. In correspondence with this, in the Markovian description U and V refer to the level of junk memory and in-use memory, respectively, and can only take (non-negative) integer values such that $0 \leq U \leq N_U$ and $0 \leq V < N_V$. Note that in applications, the memory level corresponds to memory used exclusively for the processing of data; memory usage by other system processes is not included.

Altogether, the following parameters define the system:

- M, the value of the memory cap;
- the state space $\mathcal{S}$, the generator Q and the rate vector $\{r_i : i \in \mathcal{S}\}$ define the arrival process;
- s is the service rate;

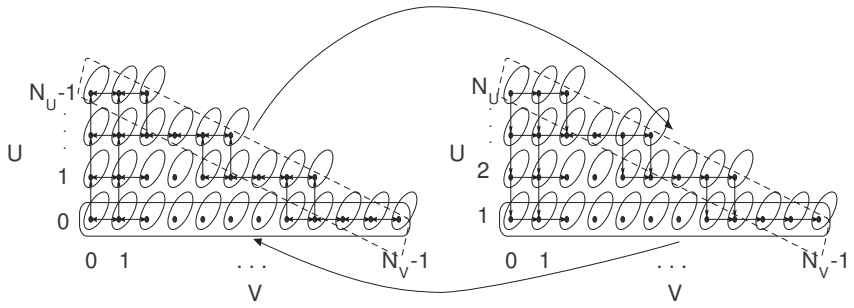

Fig. 1: State space (left-hand side: service, right-hand side: GC)

- g is the rate of garbage collection;
- C is the ratio of memory usage generated during processing compared to the size of the data;
- N_U and N_V describe the granularity of the memory.

We obtain a finite 4-dimensional state space Ω with

- dimension 1 representing the arrival process, and
- dimension 2 representing the value of V (in-use memory),
- dimension 3 representing the value of U (junk-memory),
- dimension 4 representing GC or service mode.

We assume GC starts immediately when $U/N_U + V/N_V$ reaches 1 during service, and service restarts immediately when U reaches 0 during GC. Thus, for the service states $U/N_U + V/N_V < 1$ holds, which can be depicted as a triangular shaped array. For the GC states, the value of U is essentially higher by 1: $1 \leq U \leq N_U$, and service restarts immediately when U reaches 0. The state space is depicted in Figure 1. Dimension 1 is depicted only in small bubbles; dimensions 2 and 3 are represented by the large triangular arrays, and dimension 4 only has size 2, which is depicted as the two triangles on the left and right. Altogether,

$$\begin{aligned}\Omega = &\{(i,j,k,\text{service}) : i \in S, j \geq 0, k \geq 0, j/N_V + k/N_U < 1\} \cup \\ &\{(i,j,k,\text{GC}) : i \in S, j \geq 1, k \geq 0, (j-1)/N_V + k/N_U < 1\}\,. \end{aligned} \tag{2}$$

In the following notations, i denotes the state of the background process, j denotes the value of V, k denotes the value of U, and l denotes the service mode (either service or GC). The type of transitions possible from a state depend slightly on where the state is situated within the two triangles; three main types of transitions are present: transitions corresponding to (1) the changes of the background process, (2) arrivals, and (3) service. Arrivals are suppressed when memory is full. Service increases the junk memory and decreases the queue (decreasing is suppressed when the queue is empty). Also, the forced transitions

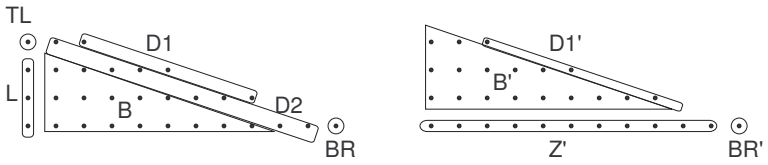

Fig. 2: States grouping

are present at the diagonal border of the service triangle and the bottom row of the GC triangle.

For clarity, we group the states according to Figure 2. For the service states (left triangle), TL is the top left corner, L is the left border (except TL). BR is the bottom right corner. D1 is the rightmost nodes in the diagonal (except BR), D2 is the rest of the diagonal (the "uppermost" states except D1 and TL). The remaining states are grouped together in B, the "bulk" of the service states. For the GC states, D1' and BR' are the counterparts of D1 and BR, Z' is the bottom row (the states from which service may resume) and B' is the bulk of the GC states. We omit a more formal definition of the groups. Note that the groups are understood within the 3-dimensional subspace of (U, V) and the service type.

We have the following types of transitions.

The background process may change at any state regardless of memory levels or service mode:

- for any (i, j, k, l), we may transition from (i, j, k, l) to (i', j, k, l) according to the generator Q of the arrival process;

For $(j, k, l) \in B$, that is, the bulk of the service states, we have different types of transitions depending on whether $r_i < s$ or $r_i > s$. If $r_i < s$, we have the following transitions:

- from $(i, j, k, \text{service})$ to $(i, j, k+1, \text{service})$ with rate $N_U \cdot C \cdot s/M$,
- from $(i, j, k, \text{service})$ to $(i, j-1, k, \text{service})$ with rate $N_V(s - r_i)/M$;

while if $r_i > s$, we have the transitions:

- from $(i, j, k, \text{service})$ to $(i, j, k+1, \text{service})$ with rate $N_U \cdot C \cdot s/M$,
- from $(i, j, k, \text{service})$ to $(i, j+1, k, \text{service})$ with rate $N_V(r_i - s)/M$.

The above values ensure that the horizontal and vertical drifts in the Markovian model are in correspondence with the fluid model (1); the factors N_U, N_V and $1/M$ are due to the discretisation.

For $(0, k, l) \in L$, that is, the queue is empty during a service period, the types of transitions depend on whether $r_i > s$ or $r_i < s$. If $r_i < s$, we have the following transitions:

- from $(i, 0, k, \text{service})$ to $(i, 0, k+1, \text{service})$ with rate $N_U \cdot C \cdot r_i/M$,

while if $r_i > s$, we have the following transitions:

- from $(i, 0, k, \text{service})$ to $(i, 0, k+1, \text{service})$ with rate $N_U \cdot C \cdot s/M$,

– from $(i, 0, k, \text{service})$ to $(i, 1, k, \text{service})$ with rate $N_V(r_i - s)/M$;

For $(j, k, l) \in D_2$, $r_i > s$, we also have some transitions corresponding to the forced transitions from service to GC. If $r_i < s$, we have the transitions:

– from $(i, j, k, \text{service})$ to $(i, j, k + 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,
– from $(i, j, k, \text{service})$ to $(i, j - 1, k, \text{service})$ with rate $N_V(s - r_i)/M$;

while if $r_i > s$, we have the transitions:

– from $(i, j, k, \text{service})$ to $(i, j, k + 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,
– from $(i, j, k, \text{service})$ to $(i, j + 1, k, \text{service})$ with rate $N_V(r_i - s)/M$.

$(j, k, l) \in D_1$ is similar; if $r_i < s$, we have the transitions:

– from $(i, j, k, \text{service})$ to $(i, j, k + 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,
– from $(i, j, k, \text{service})$ to $(i, j - 1, k, \text{service})$ with rate $N_V(s - r_i)/M$;

while if $r_i > s$, we have the transitions:

– from $(i, j, k, \text{service})$ to $(i, j, k + 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,
– from $(i, j, k, \text{service})$ to $(i, j + 1, k, \text{GC})$ with rate $N_V(r_i - s)/M$ (note that $N_U \leq N_V$ ensures that $(i, j + 1, k, \text{GC}) \in \Omega$).

For $(j, k, l) \in TL$ (which contains a single element, $(j, k, l) = (0, N_U - 1, \text{service})$), for $r_i < s$, we have

– from $(i, 0, N_U - 1, \text{service})$ to $(i, 0, N_U, \text{GC})$ with rate $N_U \cdot r_i/M$;

and for $r_i > s$, we have

– from $(i, 0, N_U - 1, \text{service})$ to $(i, 0, N_U, \text{GC})$ with rate $N_U \cdot S \cdot s/M$,
– from $(i, 0, N_U - 1, \text{service})$ to $(i, 1, N_U - 1, \text{service})$ with rate $N_V(r_i - s)/M$.

For $(j, k, l) \in BR$ (which is again a single element, $(j, k, l) = (N_V - 1, 0, \text{service})$), for $r_i < s$ we have

– from $(i, N_V - 1, 0, \text{service})$ to $(i, N_V - 1, 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,
– from $(i, N_V - 1, 0, \text{service})$ to $(i, N_V - 2, 0, \text{service})$ with rate $N_V(s - r_i)/M$,

while for $r_i > s$, we have the transitions:

– from $(i, N_V - 1, 0, \text{service})$ to $(i, N_V - 1, 1, \text{GC})$ with rate $N_U \cdot C \cdot s/M$,

and the transition increasing V is suppressed (this corresponds to data loss in the system).

For $(j, k, l) \in B'$, that is, the bulk of the GC states, we have the following transitions:

– from (i, j, k, GC) to $(i, j, k - 1, \text{GC})$ with rate $N_U \cdot g/M$,
– from (i, j, k, GC) to $(i, j + 1, k, \text{GC})$ with rate $N_V \cdot r_i/M$;

For $(j, k, l) \in D_1'$, we have the following transitions:

– from (i, j, k, GC) to $(i, j, k-1, \text{GC})$ with rate $N_U \cdot g/M$,

and the transition only increasing V is suppressed; this corresponds to data loss in the system.

For $(j, k, l) \in Z'$, we have the transitions:

– from $(i, j, 1, \text{GC})$ to $(i, j, 0, \text{service})$ with rate $N_U \cdot g/M$;
– from (i, j, k, GC) to $(i, j+1, k, \text{GC})$ with rate $N_V \cdot r_i/M$.

For $(j, k, l) \in BR'$, we have the following transitions:

– from $(i, N_V - 1, 1, \text{GC})$ to $(i, N_V - 1, 0, \text{service})$ with rate $N_U \cdot g/M$

and transitions increasing V are suppressed; these contribute to data loss.

The collection of the above transitions define a CTMC on the state space Ω.

3 Stationary Analysis

From the stationary analysis of such a system, it is possible to derive the following parameters:

– distribution and mean of memory level (both in-use and junk memory);
– mean period length (of an entire service + GC cycle, or the two separately);
– mean time spent with GC;
– mean utilisation (along with the ratio of CPU usage spent on GC and service);
– effective long-term rate of service;
– mean loss ratio and mean loss rate;
– average response time (in Section 3.1).

We calculate them as follows. If $v_{st}(i, j, k, l)$ denotes the stationary distribution of the system, then the mean memory levels can be calculated as follows:

$$\bar{M}_{\text{in-use}} = \sum_i \sum_j \sum_k \sum_l k v_{st}(i, j, k, l) \qquad \bar{M}_{\text{junk}} = \sum_i \sum_j \sum_k \sum_l j v_{st}(i, j, k, l)$$
$$\bar{M}_{\text{total}} = \sum_i \sum_j \sum_k \sum_l (j + k) v_{st}(i, j, k, l) \tag{3}$$

CPU utilisation rates can be calculated as

$$\rho_{\text{service}} = \sum_i \sum_{j \geq 1} \sum_k v_{st}(i, j, k, \text{service}) \quad + \sum_i \sum_k v_{st}(i, 0, k, \text{service}) \min(1, r_i/s)$$
$$\rho_{\text{GC}} = \sum_i \sum_j \sum_k v_{st}(i, j, k, \text{GC}) \qquad \rho_{\text{total}} = \rho_{\text{service}} + \rho_{\text{GC}} \tag{4}$$

In order to calculate the mean time of garbage collection intervals, we first need to calculate the average in-use memory level at the beginning of a garbage collection period.

$$\bar{M}_{\text{in-use at GC start}} = \sum_i \sum_k k v_{st}(i, N-k, k, \text{GC}) / W_{\text{GC start}}, \text{ where} \tag{5}$$

$$W_{\text{GC start}} = \sum_i \sum_k v_{st}(i, N-k, k, \text{GC}); \tag{6}$$

then the mean time of garbage collection intervals is simply calculated as

$$\bar{T}_{\text{GC}} = \bar{M}_{\text{in-use at GC start}} / g \tag{7}$$

and the mean time of an entire cycle of service plus garbage collection can be calculated as

$$\bar{T}_{\text{total period}} = \bar{T}_{\text{GC period}} / \rho_{\text{GC}}. \tag{8}$$

For mean loss rate, we use the formula

$$\begin{aligned} \bar{L} = & \sum_i \max((r_i - s), 0) v_{st}(i, N-1, 0, \text{service}) + \\ & \sum_i \sum_{j<N-1} v_{st}(i, j, N-j, \text{GC}) \max((r_i - g), 0) + \\ & \sum_i \max((r_i - g), 0) v_{st}(i, N-1, 1, \text{GC}), \end{aligned} \tag{9}$$

and the mean loss ratio is

$$\bar{l} = \bar{L} / \bar{r}, \tag{10}$$

where $\bar{r}$ is the average rate of arrival.

The *effective rate of service* is

$$s_e = \frac{gs}{g+s}. \tag{11}$$

since each arrival needs to be served with rate s and (after some time) flushed with rate g.

Analysis of the average response time requires a more involved calculation.

3.1 Analysis of average response time

For analysis of average response time, we assume the system is FIFO. Average response time is the total time spent in the system (spent with either service or waiting for service). It will also be referred to as delay.

The main idea is the following: when a tagged unit of data ("job") arrives during state (i, j, k, l), it will enter the queue. We consider this job as in position

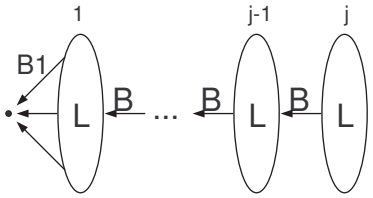

Fig. 3: QBD representation

j within the queue, where each position corresponds to a unit segment within the queue. As the jobs are served, the tagged job will move ahead in the queue, eventually reaching position 1 and then being served.

The position of the tagged job within the queue as the system progresses is not included in the state of the system in the previously defined Markov chain. Instead, we represent it as the level in a quasi birth-death process (QBD) (see [15]), where the states are $((i,j,k,l),m)$, with m denoting the position of the tagged job within the queue (the level). (We remark that matrix-geometric methods are also a possible alternative to the QBD approach presented, see [21].) Initially, a job arrives in state $((i,j,k,l),j)$ with probability

$$\pi(i,j,k,l) = \frac{v_{st}(i,j,k,l)r_i}{\sum_{i,j,k,l} v_{st}(i,j,k,l)r_i} \tag{12}$$

since in state (i,j,k,l) jobs arrive with rate r_i. π is understood as a row vector of size $|\Omega|$.

All transitions of the original generator are partitioned into matrices B and L (of size $|\Omega| \times |\Omega|$), with B corresponding to transitions that decrease the level, that is, the transitions corresponding to service. L corresponds to the rest of the transitions. In order to avoid listing all transitions again, we refer to Section 2.2; from among all transitions listed there, all the transitions where *the third coordinate increases* go to B, while the rest of the transitions go to L (including the negative values in the diagonal). The corresponding QBD represents the progress of the tagged job along with the state of the entire system. See Figure 3. The process does not contain actual 'births', since the level may only decrease. In such a system, let T_1 denote the (random) time it takes to go down one level to some state (i',j',k',l'), assuming we started from state (i,j,k,l), and H denotes its Laplace-transform:

$$\begin{aligned} &H_{T_1}(s)_{(i,j,k,l),(i',j',k',l')} = \\ &E(e^{-T_1 s}1(\text{first backwards transition is to state } (i',j',k',l')) \\ &\quad |\text{starting from state } (i,j,k,l)) \end{aligned} \tag{13}$$

H can be calculated as follows [15]:

$$H_{T_1}(s) = (sI - L)^{-1}B.$$

The total delay T of the tagged job is equal to the time it takes to cross j levels from initial distribution π and regardless of the end state (see also Figure

3), which has Laplace transform

$$H_T(s) = \sum_{i,j,k,l} \pi_{(i,j,k,l)} \cdot H_{T_1}^j(s) \cdot \mathbb{1} \tag{14}$$

where $\mathbb{1}$ denotes the constant 1 column vector of size $|\Omega|$. Thus

$$\begin{aligned}
E(T) = &- \frac{\mathrm{d}}{\mathrm{d}s} H_T(s)\Big|_{s=0} = -\frac{\mathrm{d}}{\mathrm{d}s} \sum_{i,j,k,l} \pi_{(i,j,k,l)} \cdot H_{T_1}^j(s)\Big|_{s=0} \cdot \mathbb{1} = \\
&- \frac{\mathrm{d}}{\mathrm{d}s} \sum_{i,j,k,l} \pi_{(i,j,k,l)} \cdot ((sI-L)^{-1}B)^j\big|_{s=0} \cdot \mathbb{1} = \\
&- \sum_{i,j,k,l} \pi_{(i,j,k,l)} \cdot \sum_{m=0}^{j-1} ((sI-L)^{-1}B)^m (sI-L)^{-2} B((sI-L)^{-1}B)^{j-1-m}\Big|_{s=0} \cdot \mathbb{1} = \\
&- \sum_{i,j,k,l} \pi_{(i,j,k,l)} \cdot \sum_{m=0}^{j-1} ((-L)^{-1}B)^m (-L)^{-2} B((-L)^{-1}B)^{j-1-m} \cdot \mathbb{1}.
\end{aligned} \tag{15}$$

Similarly,

$$\begin{aligned}
E(T^2) = &\frac{\mathrm{d}^2}{\mathrm{d}s^2} H_T(s)\Big|_{s=0} = \sum_{i,j,k,l} \Bigg[\pi_{(i,j,k,l)} \cdot \sum_{m=1}^{j-1} \sum_{l=0}^{m-1} ((-L)^{-1}B)^l (-L)^{-2} \times \\
&\times B((-L)^{-1}B)^{m-1-l} (-L)^{-2} B((-L)^{-1}B)^{j-1-m} + \\
&\sum_{m=0}^{j-1} ((-L)^{-1}B)^m (-2)(-L)^{-3} B((-L)^{-1}B)^{j-1-m} + \\
&\sum_{m=0}^{j-2} ((-L)^{-1}B)^m (-L)^{-2} B \sum_{l=0}^{j-2-m} ((-L)^{-1}B)^l (-L)^{-2} B((-L)^{-1}B)^{j-2-m-l} \Bigg] \cdot \mathbb{1}.
\end{aligned} \tag{16}$$

(15) and (16) are explicit for $E(T)$ and $E(T^2)$, thus the mean and variance of the delay can be calculated. However, L and B are sparse matrices of size $|\Omega| \times |\Omega|$, which is typically large, so the actual calculations need special care. In the rest of this section, we sketch an efficient algorithm for the calculation of the formulas (15) and (16) for large Ω.

The first main point is that for large Ω, we only make calculations with vectors. To calculate (15), we start with the rightmost vector $\mathbb{1}$. Then, apart from summations, only 2 steps are repeated: either multiplication by B, which is feasible, or multiplication by $(-L)^{-1}$. The calculation of $(-L)^{-1}$ is infeasible, so to calculate $(-L)^{-1}v$ for some v, we solve $(-L)x = v$ instead. L has a special structure; we show that with a proper reordering of the states, L it will be upper block diagonal (with small block sizes), which allows $(-L)x = v$ to be solved block by block.

The ordering is as follows:

– The states for the same values of j, k, l will form blocks of size $|Q|$. The order within the block is irrelevant.
– For each value of j, k, the block for $l = 1$ comes before the block for $l = 2$.
– Then, for each value of j, the blocks are ordered in an increasing manner according to k (without changing the order of the blocks belonging to the same value of k).
– Then the blocks are ordered in a decreasing manner according to the value of j (without changing the order of the blocks belonging to the same value of j).

According to Figure 1, this means that the last block is the bottom right corner of the GC triangle, preceded by the bottom right corner of the service triangle. Then the bottom rows of the two triangle follow from right to left, with blocks from the GC triangle and blocks from the service triangle alternating. Then the left of the rows follow from bottom to top.

The first two blocks (corresponding to the bottom right corners of each triangle) are special in the sense that L contains transitions between them in both directions. However, from all other blocks, L only contains transitions that go to later blocks (according to the above ordering), so in the above ordering, L is indeed block-upper-triangular, with a single diagonal block of size $2|Q|$ and all other diagonal blocks of size $|Q|$.

This allows us to solve $(-L)x = v$ for any v efficiently.

Starting from the vector $\mathbb{1}$, we keep multiplying by B and $(-L)^{-1}$ until we obtain the vectors $((-L)^{-1}B)^m(-L)^{-2}B((-L)^{-1}B)^{j-1-m} \cdot \mathbb{1}$ (from (15)). This process can be sped up by storing the vectors $((-L)^{-1}B)^j \cdot \mathbb{1}$ for separate values of j. Then the final summation can be made more efficient by pre-splitting π into vectors $\pi = \sum_j \pi_j$, where π_j only contains the elements of π whose second coordinate is j (that is, π_j corresponds to a single row in the service and GC triangles). Then

$$E(T) = \sum_j \pi_j \cdot \sum_{m=0}^{j-1} ((-L)^{-1}B)^m(-L)^{-2}B((-L)^{-1}B)^{j-1-m} \cdot \mathbb{1}. \tag{17}$$

(16) can be calculated efficiently using similar techniques (albeit with more steps). We do not go into further details due to lack of space here.

4 Experimental results

4.1 Calculating network performance KPIs

The basis of the experimentation is a storm-based data processing system that uses reports from a large number of network elements (e.g. base stations) to calculate higher level network performance KPIs (key performance indicators). The topology of the processing system is a four-stage pipeline. We examine the first stage, called Parser, which parses the reports and retrieves the measurements from them.

The experimentation took place in the lab environment of Nokia, Bell Labs, with status reports stored in an HDFS storage and played back with real traffic timing. The processing software is implemented within the Apache Storm framework [2]. For monitoring, the Ganglia monitoring system was used [1]. Measurements were registered at intervals of length 500ms.

4.2 Application of the model

We apply the model to the Parser unit. First, the input data stream was approximated by a stationary Markov-modulated fluid model using k-means clustering to obtain the background Markov process with generator Q. Technically, input is given in discrete units (files), but the file size is relatively small compared to the total memory size.

Initial measurements showed that file size of the input data is proportional to both the amount of memory used during service, and also to the service time necessary. The corresponding constant factors were measured and are used as an input to the model. Service rate was also measured.

First, we are interested in the effect of discretisation: we model the same input process with several different setups of (N_U, N_V) pairs. The input parameters are (r and Q are not included in their entirety; input was clustered to 6 clusters):

$$s = 14.6MB/s,\ g = 64440MB/s,\ C = 40.81, M = 252MB \qquad (18)$$
$$\bar{r} = 0.78MB/s,\ \max(r) = 20.9MB/s.$$

Service rate was measured using an artificially overloaded system, while the constant C was obtained by comparing the junk memory and the size of the incoming data. We note that the parameters in (18) reflect a relatively low load of the system. With a high load, certain processes such as memory swapping may be initiated which are not included in the model.

Table 1 contains the values of several performance measures obtained from the stationary analysis of the model for various (N_U, N_V) pairs.

	(4,8)	(10,20)	(20,40)	(10,50)
mean period length	7.8282	7.8196	7.8193	7.8195
utilisation	0.05421	0.05429	0.05429	0.05433
mean loss ratio	1.17e-6	6.51e-10	1.69e-12	2.87e-13

Table 1: Effect of discretisation

The mean period length and the utilisation change very little as (N_U, N_V) are increased. On the other hand, the mean loss ratio is small and decreases rapidly as (N_U, N_V) increases. For the above input, it should be considered 0.

Analysis showed that the effect of the discretisation is relatively small, in other words, the model performs well with moderately large values of N_U and N_V

(at least for utilisation and mean period length); from now on, we set $(N_U, N_V) = (10, 20)$ but with various inputs for actual validation.

From among the performance measures calculated in Section 3, we use the mean period length for validation with real life data. Mean period length is the mean time of an entire cycle of a service plus garbage collection period. The mean period length is easy to measure reliably: the real life monitoring system keeps count of the number of garbage collections over a sustained period of time. Several other performance measures are difficult to measure reliably: CPU usage relates to utilisation but may be distorted by other system processes. Loss ratio is known to be 0 from the actual monitoring, and this is approximated fairly well by the model, but relative error does not make sense in this case. Delay and the length of the queue was not possible to measure with the monitoring system.

The memory cap is slightly different for each run, ranging between 190MB and 252MB. The input process also varies slightly, with the minimal input rate 0, maximal input rate changing between 17 MB/s and 21 MB/s, and average input rate changing between 0.72 MB/s and 0.78 MB/s.

input run	1	2	3	4	5	6	7
from model	7.8196	8.2805	7.6990	7.2829	6.8598	6.6874	6.7157
monitored	7.6585	8.1437	7.5303	7.1277	6.7030	6.5090	6.5090
relative error	1.02%	1.02%	1.02%	1.02%	1.02%	1.03%	1.03%

Table 2: Validation of mean period length

Overall, the relative error is around 1%, with the model consistently overestimating the mean period length according to actual monitored results. The exact explanation and correction to the model is subject to further research, along with a more direct validation of the model. We also believe that the model presented models garbage collection on a realization level (not just stationary behaviour), but again, this is difficult to validate due to the fact that measurements made too often will distort the results themselves.

Close results in the literature are due to [5], but differences in the models (for example, description of the arrival process) make a direct comparison difficult.

5 Conclusion

The model is only applicable for a certain region of parameters. Under certain conditions, processes like memory swapping may be initiated. These are not included in the model.

The current model only includes one "type" of memory. However, in many memory management applications, there are "young" and "old" sections of the memory to store data for short and long term calculations. Such sections may be integrated in the model naturally with the expansion of the state space. This is subject to future work.

The computation of the stationary distribution (and the derived performance measures of Section 3) may be infeasible for very large values of N_U and N_V. Possible future work includes the application of dimension-reduction techniques based on tensor decomposition [18].

The model is sophisticated enough to allow modelling of a process on a realization level. This may be explored further.

Another natural option is to examine a transient version of the model; this would allow the examination of unstable systems as well.

An explicit solution for the original fluid model of Section 2.1 is also an interesting challenge.

Acknowledgment

We would like to thank Miklós Telek and Gábor Horváth for their valuable help and insight.

References

1. Ganglia monitoring system. `http://ganglia.sourceforge.net/` . Accessed: 2017-05-08.
2. Apache Storm. `http://storm.apache.org/` . Accessed: 2017-05-08.
3. S. Ahn, V. Ramaswami: Efficient algorithms for transient analysis of stochastic fluid flow models. *J. Appl. Probab.*, 42(2):531–549, 2005.
4. Bacon, David F., Perry Cheng, and V. T. Rajan. "The Metronome: A simpler approach to garbage collection in real-time systems." OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". Springer Berlin Heidelberg, 2003.
5. S. Balsamo, G. Dei Rossi, and A. Marin. "Optimisation of virtual machine garbage collection policies." In LNCS 6751/2011, proceedings of the International Conference ASMTA, pages 7084. Springer, Venice, IT, 2011. ISBN 978-3-642-21712-8. ISSN 0302-9743.
6. Blackburn, Stephen M., Perry Cheng, and Kathryn S. McKinley. "Oil and water? High performance garbage collection in Java with MMTk." Proceedings of the 26th International Conference on Software Engineering. IEEE Computer Society, 2004.
7. Bolch, Gunter, et al. Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. John Wiley & Sons, 2006.
8. Brodie-Tyrrell, William. Surf: an abstract model of distributed garbage collection. Diss. 2008.
9. Bux, Werner, and Ilias Iliadis. "Performance of greedy garbage collection in flash-based solid-state drives." Performance Evaluation 67.11 (2010): 1172-1186.
10. Detlefs, David, et al. "'Garbage-first garbage collection"'. Proceedings of the 4th international symposium on Memory management. ACM, 2004.
11. Gribaudo, Marco, and Miklós Telek. "Fluid models in performance analysis." International School on Formal Methods for the Design of Computer, Communication and Software Systems. Springer Berlin Heidelberg, 2007.
12. Jones, Richard, and Rafael D. Lins. "Garbage collection: algorithms for automatic dynamic memory management." (1996).

13. Jones, Gareth L., et al. "Fluid queue models of battery life." 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE, 2011.
14. Kwon, Ohhoon, et al. "FeGC: An efficient garbage collection scheme for flash memory based storage systems." Journal of Systems and Software 84.9 (2011): 1507-1523.
15. G. Latouche, V. Ramaswami: Introduction to matrix analytic methods in stochastic modeling, ASA-SIAM, 1999.
16. Li, Yongkun, Patrick PC Lee, and John CS Lui. "Stochastic modeling and optimization of garbage collection algorithms in solid-state drive systems." Queueing Systems 77.2 (2014): 115-148.
17. Lowry, Matthew C. A new approach to the train algorithm for distributed garbage collection. Diss. 2004.
18. D. Kressner and F. Macedo. "Low-Rank Tensor Methods for Communicating Markov Processes." Proceedings of Quantitative Evaluation of Systems: 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. 25-40. Springer, 2014.
19. Medhi, Jyotiprasad. Stochastic models in queueing theory. Academic Press, 2002.
20. Nazarathy, Yoni, and Gideon Weiss. "A fluid approach to job shop scheduling: theory, software and experimentation." J Sched 13 (2009): 509-529.
21. M. Neuts: "Matrix-geometric solutions in stochastic models. An algoritheoremic approach." The Johns Hopkins University Press, Baltimore, MD, 1981.
22. Norcross, Stuart J. "Deriving Distributed Garbage Collectors from Distributed Termination Algorithms." Diss. University of St Andrews, 2004.
23. Plainfossé, David, and Marc Shapiro. "A survey of distributed garbage collection techniques." Memory Management. Springer Berlin Heidelberg, 1995. 211-249.
24. H. Kobayashi, Q. Ren: Transient solutions for the buffer behavior in statistical multiplexing, *Performance Evaluation*, 23(1):65–87, 1995
25. Schoeberl, Martin. "Real-time garbage collection for Java." Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06). IEEE, 2006.
26. Sericola, Bruno. "Transient analysis of stochastic fluid models." *Performance Evaluation*,32(4):245–263, 1998
27. L. Lakatos, L. Szeidl, and M. Telek, Introduction to Queueing Systems with Telecommunication Applications. Springer, 2013.
28. Van Houdt, Benny. "A mean field model for a class of garbage collection algorithms in flash-based solid state drives." ACM SIGMETRICS Performance Evaluation Review. Vol. 41. No. 1. ACM, 2013.
29. Van Houdt, Benny. "Performance of garbage collection algorithms for flash-based solid state drives with hot/cold data." Performance Evaluation 70.10 (2013): 692-703.
30. Wilson, Paul R. "Uniprocessor garbage collection techniques." Memory Management. Springer Berlin Heidelberg, 1992. 1-42.
31. Yang, Yue, and Jianwen Zhu. "Analytical modeling of garbage collection algorithms in hotness-aware flash-based solid state drives." 2014 30th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, 2014.