

Evaluating the Performability of Systems with Background Jobs*

Qi Zhang¹, Alma Riska², Ningfang Mi¹, Erik Riedel², and Evgenia Smirni¹

¹Computer Science Dept., College of William and Mary, Williamsburg, VA 23187.

¹{qizhang, ningfang, esmirni}@cs.wm.edu

²Seagate Research, 1251 Waterfront Place, Pittsburgh, PA 15222.

¹{alma.riska, erik.riedel}@seagate.com

Abstract

As most computer systems are expected to remain operational 24 hours a day, 7 days a week, they must complete maintenance work while in operation. This work is in addition to the regular tasks of the system and its purpose is to improve system reliability and availability. Nonetheless, additional work in the system, although labeled as best effort or low priority, still affects the performance of foreground tasks, especially if background/foreground work is non-preemptive. In this paper, we propose an analytic model to evaluate the performance trade-offs of the amount of background work that a storage system can sustain. The proposed model results in a quasi-birth-death (QBD) process that is analytically tractable. Detailed experimentation using a variety of workloads shows that under dependent arrivals both foreground and background performance strongly depends on system load. In contrast, if arrivals of foreground jobs are independent, performance sensitivity to load is reduced. The model identifies dependence in the arrivals of foreground jobs as an important characteristic that controls the decision of how much background load the system can accept to maintain high availability and performance gains.

Keywords: *Foreground/background jobs; storage systems; idle periods; Markov Modulated Poisson Process, QBD.*

1 Introduction

Nowadays, computer systems are rarely taken off-line for maintenance. Even simple workstations, are in operation 24 hours a day, 7 days a week. Consequently, most systems schedule necessary maintenance that intends to assess system status and predict/avoid reliability and availability issues as background tasks [1, 2, 12, 17]. Very often, background activity is also associated with approaches that aim at enhancing system performance [4, 21, 3].

Although background activity is critical to system operation, it often has lower priority than foreground work, i.e., the work requested by the system users. Therefore, it is of paramount importance for system designers to better understand the trade-offs between minimizing the foreground performance degradation and maximizing completion of background tasks so that system reliability, availability, and performance are improved in the long-run and without compromising the short-term performance of foreground work. While facilitating and supporting background activity in a system is a general concept [20], its applicability differs among systems, i.e., distributed and clustered systems, storage systems, and communication systems. Consequently, efforts for utilizing idle time to improve reliability or performance are often system specific and are either based on prototyping and measurements [3, 1, 4, 21] or analytic models [2, 12, 13, 15].

In this paper, we propose an analytic model that addresses performance trade-offs between foreground and background work at the disk drive level of a storage system. There are numerous cases where storage systems and disk drives deal with background jobs¹. One widely accepted background task is data integrity check or media scrubbing in disk drives [17]. Disk scrubbing is a periodic checking

*This work was partially supported by the National Science Foundation under grants CCR-0098278, ACI-0090221, and ITR-0428330, and by Seagate Research.

¹The terms “task” and “job” are used interchangeably.

of disk media to detect unaccessible sectors. If a sector is not accessible then it is reported up to the file system for data recovery and it is remapped elsewhere on the disk. Another background activity in disk drives is the RAID rebuild process [19, 12], which happens when one disk in a RAID array fails and its data is reconstructed in a spare disk using the data in the remaining disks of the array. Other examples of background activities include flushing of write-back caches, prefetching, and replication [19].

Background tasks in a storage system may be periodic such as disk scrubbing, or may span over a long period of time, such as the RAID rebuild. Yet there are tasks where the background jobs have the same service demands as the foreground ones. For example, disk WRITE verification incurs one extra READ to detect any disk WRITE error. This process, known as READ-after-WRITE, degrades disk performance substantially and is not feasible if running in foreground, but is attractive as a low priority background activity. Nevertheless, its successful completion is tightly related to the reliability and consistency of the data.

In this paper, we propose a model, which consists of an infinite Markov chain with repetitive structure that captures the disk or storage system behavior under the background activity whose service demands are similar to the foreground activity. It differs from similar models proposed for storage systems [2] because it allows for bursty and autocorrelated arrivals, which are the case in storage systems [16]. The solution of the proposed model is tractable and can be solved using the well-known matrix-geometric method [10]. The model establishes that the relative performance of foreground and background jobs is similar for either independent or dependent arrivals. However, the saturation under dependent arrivals is very fast (for small changes in foreground workload), which actually effects more completion rate of background jobs rather than the latency of the foreground ones. For example, the non-preemptive background jobs delay in the worst case only 20% of all foreground jobs, with most delays remain below 5%. However, under highly correlated arrivals and medium load the completion of background tasks is minimal (close to zero), a non-desirable outcome when background activity intends to enhance long-term reliability, such as the case of WRITE verification.

This paper is organized as follows. Section 2 presents related work. Section 3 gives an overview of storage systems under background jobs. The proposed model is presented in Section 4. Performance evaluation results that are derived using the analytic model are presented in Section 5. Conclusions and directions for future work are given in Section 6.

2 Related Work

Multiple sources [4, 3, 16] indicate that computer system resources operate under bursty arrivals and while they have periods of high utilization, they may also have long stretches of idleness. For example, in average disk drives are only 20% utilized [16]. Given that a system operates in low utilization, a myriad of approaches have been proposed aiming at utilizing idle time to improve performance [4, 3], fault tolerance [1], and reliability [17]. The goal is to schedule performance/availability enhancing activities as low priority and minimize their impact on user performance [3].

The motivation of our work stems from storage systems, where traditionally a variety of tasks, mostly aiming at enhancing data reliability, are treated as background activity [2]. Storage system background functions that address reliability, availability, and consistency typically include data reconstruction [12], data replication [13], disk scrubbing [17], and WRITE verification [2]. Background jobs may also address storage performance issues including data replication in a cluster to improve throughput or data reorganization to minimize disk arm movement [4, 21].

Because background activity has often low priority, its service is completed only when there is no foreground activity in the system, i.e., at the end of a busy period. Vacation models have been proposed for the general performance analysis of systems where foreground/background jobs co-exist [20, 15, 15, 22, 23]. To the best of our knowledge, vacation models that are applied in storage systems or disk drives have been considered only in [2]. However the models in [2] attempt to model a system whose arrival process is strictly exponential and the background task results from sequential scanning of the data on a disk or part of it. In this paper, we explicitly model the performance effects of dependence (be it short range or long range) in the arrival process of background/foreground jobs on the disk, which has been detected in [7, 16, 5]. We examine the effects of both variability and dependence in the arrivals. We further assume that background and foreground jobs are drawn from the same distribution because we are interested in the set of background activities such as WRITE verification that have the same service demands as the user requests.

3 Storage System

In this section, we first identify the salient characteristics of IO workloads and we give an overview of the operation of the system with foreground and background tasks.

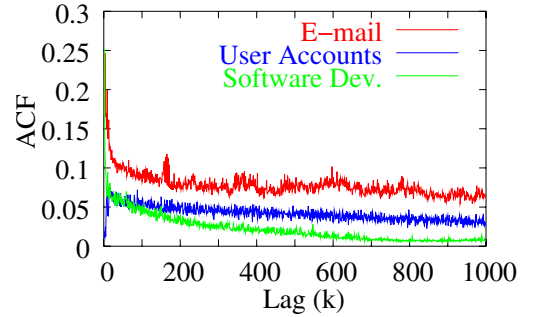
3.1 Workload Parameterization

In storage systems and disk drives the arrival process is bursty or self-similar [7, 16]. Here, we look at traces measured in different storage systems [16] that show high inter-dependence in the arrival streams of requests. Throughout this paper, we use the autocorrelation function (ACF) as a metric of the dependence structure of a time series and the coefficient of variation (CV) as a metric of variability. Consider a stationary time series of random variables $\{X_n\}$, where $n = 0, \dots, \infty$, in discrete time. The autocorrelation function (ACF) $\rho_X(k)$ and the coefficient of variation (CV) are defined as follows

$$\rho_X(k) = \rho_{X_t, X_{t+k}} = \frac{E[(X_t - \mu)(X_{t+k} - \mu)]}{\delta^2}, CV = \frac{\delta}{\mu},$$

where μ is the mean and δ^2 is the common variance of $\{X_n\}$. The argument k is called the lag and denotes the time separation between the occurrences X_t and X_{t+k} . The values of $\rho_X(k)$ may range from -1 to 1. If $\rho_X(k) = 0$, then there is no autocorrelation at lag k . If $\rho_X(k) = 0$ for all $k > 0$ then the series is independent, i.e., uncorrelated. In most cases ACF approaches zero as k increases. The ACF's decay rate distinguishes processes as short-range dependent (SRD) or long-range dependent (LRD).

Figure 1 presents the autocorrelation function (ACF) of the inter-arrival times of three traces that have been collected in three different systems, each supporting an e-mail server, a software development server, and user accounts server, respectively. These traces consist of a few hundred thousands entries each and are measured over a 12 to 24 hour period. As expected, for different applications the dependence structure of the arrivals is different and it is a result of multiple factors including the architecture of the storage system, the file system running on top of the storage system, and the I/O path hierarchy together with the resource managing policies at all levels of the I/O path. Nonetheless, independently of all these factors, all measurements show that arrivals at the storage system exhibit some amount of autocorrelation. The table in Figure 1 shows the mean and coefficient of variation (CV) for the inter-arrival times and the service times of all requests in the trace. The three traces represent systems under different loads. Specifically, the ‘‘User Accounts’’ trace comes from a lightly loaded system (only 2% utilized), while the ‘‘E-mail’’ and ‘‘Software Development’’ traces come from systems with modest utilizations also (‘‘E-mail’’ is 8% utilized and ‘‘Software Development’’ 6% utilized). These cases of underutilized systems naturally indicate that an opportunity exists for scheduling low priority jobs in the system and treating them as background work. Additionally, the low utilization levels in the above measurement traces allow to assume that the measured job response times are a close approximation of the



	Inter arrival times		Service times		Utilization
	mean	CV	mean	CV	
E-mail	56.93	9.01	5.59	0.75	8%
Soft. Dev.	88.06	12.38	6.34	0.84	6%
User Accs,	246.65	3.85	6.1	0.74	2%

Figure 1. ACF of inter-arrival times of three traces, the respective mean (in ms) and CV of the inter-arrival and service times.

workload service times. Because all storage systems in Figure 1 consist of similar hardware, the service process is similar across all traces and it actually has low variability, i.e., CV values are less than 1.

We propose models of the arrival and service processes in a storage system that reflect the characteristics of the various traces illustrated in Figure 1. We model the service process via an exponential distribution with mean service time of 6 ms. For the arrival process, we use a two-state Markovian Modulated Poisson Process (MMPP) [8, 11].² MMPPs are processes whose events are guided by the transitions of an underlying finite absorbing continuous time Markov chain and are described by two square matrices \mathbf{D}_0 and \mathbf{D}_1 , with dimensions equal to the number of transient states in the Markov chain. \mathbf{D}_0 captures the transitions between transient states and the variability in the stochastic process while \mathbf{D}_1 is a diagonal matrix that captures the dependence structure.

Let π_{MMPP} be the stationary probability vector of the underlying Markov chain for an MMPP, i.e., $\pi_{\text{MMPP}}(\mathbf{D}_1 + \mathbf{D}_0) = \mathbf{0}$, $\pi_{\text{MMPP}}\mathbf{e} = 1$, where $\mathbf{0}$ and \mathbf{e} are vectors of zeros and ones of the appropriate dimension. A variety of performance measures are computed using π_{MMPP} , \mathbf{D}_0 , and \mathbf{D}_1 , such as the mean arrival rate, the squared coefficient of variation, and the lag- k of its autocorrelation function ACF [14]:

$$\lambda = \pi_{\text{MMPP}}\mathbf{D}_1\mathbf{e}, \quad (1)$$

²MMPP can capture various ACF levels and inter-arrival times variabilities. Additionally, by using a 2-state MMPP for the arrival process and exponential service times, the resulting queuing system can be analyzed with matrix-analytic methods [10].

$$CV^2 = \frac{E[X^2]}{(E[X])^2} - 1 \quad (2)$$

$$ACF(k) = \frac{2\lambda\pi_{\text{MMPP}}(-\mathbf{D}_0)^{-1}\mathbf{e} - 1,}{\text{Var}[X]} \quad (3)$$

$$= \frac{E[(X_0 - E[X])(X_k - E[X])]}{\text{Var}[X]} \quad (3)$$

$$= \frac{\lambda\pi_{\text{MMPP}}((-\mathbf{D}_0)^{-1}\mathbf{D}_1)^k(-\mathbf{D}_0)^{-1}\mathbf{e} - 1}{2\lambda\pi_{\text{MMPP}}(-\mathbf{D}_0)^{-1}\mathbf{e} - 1},$$

where X_0 and X_k denote two inter-event times k lags apart.

We parameterize our MMPP models, using a simple moment matching approach that follows from the Eqs.(1), (2), and (3). The two 2×2 matrices of the MMPP model, \mathbf{D}_0 and \mathbf{D}_1 , have four parameters, i.e., v_1 , v_2 , l_1 , and l_2 as shown in Eq. (4).

$$\mathbf{D}_0 = \begin{bmatrix} -(l_1 + v_1) & v_1 \\ v_2 & -(l_2 + v_2) \end{bmatrix},$$

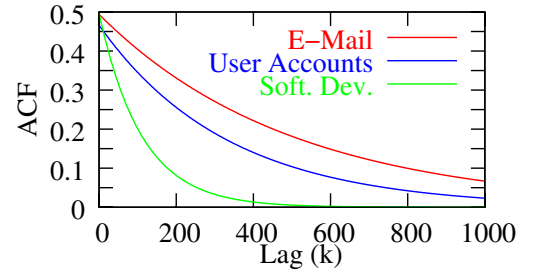
$$\mathbf{D}_1 = \begin{bmatrix} l_1 & 0 \\ 0 & l_2 \end{bmatrix}. \quad (4)$$

Our moment matching technique has one degree of freedom. We decide to set l_1 as the free parameter and adjust it to let the analytic model have the same mean response time as the real system. We parameterize three different MMPPs to model separately the three different arrival processes of our traces. The MMPPs are labeled as ‘‘E-mail’’, ‘‘User Accounts’’, and ‘‘Software Development’’ and are used as input to the analytic model that we develop here. We stress that these MMPP models do not represent an exact fitting of the traces in Figure 1, they only match the first two moments of the trace and provide a range of different ACF functions. Workload fitting such that the ACF is matched exactly, is outside the scope of this paper. In Figure 2, we show the ACF of the three MMPPs used here and their full parameterization.

3.2 Background Tasks in Storage Systems

We model a simple storage system with one service center, where foreground jobs are served in a first-come first-serve (FCFS) fashion. We assume that the amount of available buffer space is *always* large enough to store all data associated with waiting foreground tasks in the queue. Therefore, the above system is approximated by an infinite-buffer queue.

Foreground jobs consists of user arrivals *only*. Upon completion, a foreground job may either leave the system with probability $(1 - p)$, or generate a new background job with probability p , i.e., background tasks are only a portion of foreground tasks and have service demands with the same stochastic characteristics as the foreground jobs. Think of WRITE verification; only a portion of all user requests are



	v_1	v_2	l_1	l_2
E-mail	0.31e-5	0.69e-6	0.09	0.35e-3
Soft. Dev.	0.90e-6	0.19e-5	0.10e-3	0.35e-1
User Accs,	0.36e-4	0.13e-5	0.10e-1	0.49e-3

Figure 2. ACF of our 2-state MMPP models for the interarrival times of the three traces and their parameterization.

WRITES and they need to be verified once they are serviced by the disk. Background tasks are served in a ‘‘best-effort’’ manner: a background job will get served only if there is no foreground job waiting in the queue, i.e., during idle periods. Consequently, background tasks will ordinarily have longer waiting times than foreground tasks.

Neither foreground nor background tasks are preemptive, which is consistent with the nature of work in disk drives, where the service process consist of three distinct operations, i.e., seek to the correct disk track, position to the correct sector, and transfer data. The ‘‘seek’’ portion of the service time accounts in average for 50% of the service time and is a non-preemptive operation [9, 18]. Because of the non-preemptive nature of seeks, background activity inevitably impacts foreground work performance: if a background task starts service, then this precludes the existence of any foreground task in the system, but if a foreground job arrives during the service of a background job, it will have to wait in the queue and on the average experience longer delay than the delay it would have experienced if the system was not serving background tasks. To minimize this effect, background tasks do not start service immediately after the end of a foreground busy period, but after the system has been *idle* for some pre-specified period of time, which we refer to as ‘‘idle wait’’.

Background jobs, similarly to the foreground ones, require buffer space. Because the buffer is reserved for foreground jobs, background buffer is limited. As a result, some of background tasks will be dropped because the buffer is full. A practical setting in a disk drive would be to allocate 0.5-1MB of buffer space for background activity, which corresponds to approximately 50 background jobs of average size. Throughout the paper, we assume a buffer that stores a

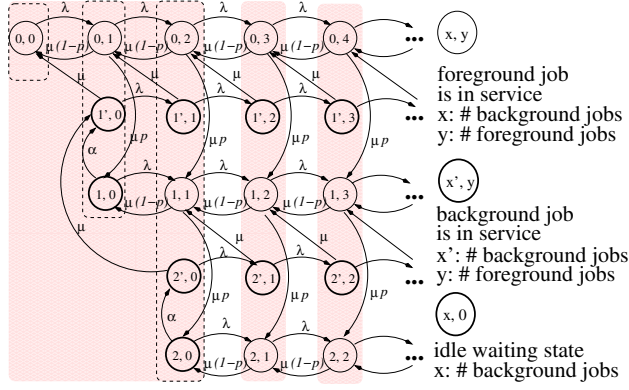


Figure 3. The Markov chain of the queuing system with infinite buffer size for foreground tasks and a buffer size of 2 for background tasks.

maximum of 50 background jobs. We also examined buffer sizes for up to 250 background jobs. Results are qualitatively same as those with buffer size 50 and are omitted due to lack of space.

4 The Markov Chain

In this section, we describe a Markov chain that models the queuing system with foreground/background activity as described in the previous section. To simplify the definition of the state space as well as transitions among states, we first assume exponential inter-arrival and service times with mean rates λ and μ , respectively. Later, in Subsection 4.1, we show how the exponential inter-arrival process is replaced by the 2-stage MMPP process. The Markov chain of the foreground/background activity is depicted in Figure 3. Because foreground jobs use an infinite buffer and the background jobs use only a finite one, the Markov chain is infinite in one dimension only. For presentation simplicity, Figure 3 shows the instance where the background buffer can store up to 2 background jobs only.

The state space is defined by a 2-tuple (x, y) , where x indicates the number of background tasks in the system (waiting or in service) and y indicates the number of foreground tasks in the system (waiting or in service). There are two sets of 2-tuples in Figure 3: (x, y) and (x', y) . States (x, y) indicate that a foreground job is being served. States (x', y) show that a background job is being served. The “idle wait” is represented by states $(x, 0)$, where $x > 0$ means that the background jobs wait for a time period, which is exponentially distributed with mean $1/\alpha$, before starting. We define levels in this Markov chain such that level j consists of the

set of states $\mathcal{S}^{(j)}$ defined as

$$\mathcal{S}^{(j)} = \{(x, y) \text{ and } (x', y) \mid 0 \leq x \leq j, 0 \leq y \leq j, x + y = j\}. \quad (5)$$

Let the maximum buffer size of the background jobs be X . Until there are X tasks in the system, the Markov chain has a tree-like structure. Beyond that point, the background buffer could be full and the levels of the Markov chain form a repetitive pattern. The form of the chain is that of a Quasi-Birth-Death process (QBD) which can be solved using matrix-analytic methods [10].

4.1 Modeling Dependence in the Arrival Process

Here, we enhance the simple Markov chain model to capture arrival streams with high variability and various degrees of dependence in their inter-arrival structure using a 2-state Markov Modulated Poisson Process (MMPP). Each state in the Markov chain of Figure 3 is now replaced by a set of sub-states, and scalars λ , μ and α are replaced by matrices \mathbf{F} , \mathbf{B} , and \mathbf{W} , respectively. An additional matrix \mathbf{L}_0 is used to describe transitions within a set of sub-states. Assume that $\mathbf{D}_0^{(A)}$ and $\mathbf{D}_1^{(A)}$ describe an A -state MMPP. Then \mathbf{L}_0 , \mathbf{F} , \mathbf{B} , and \mathbf{W} are $A \times A$ matrices computed by the following equations.³

$$\mathbf{F} = \mathbf{D}_1^{(A)}, \quad \mathbf{B} = \mathbf{I}_A \times \mu, \quad \mathbf{W} = \mathbf{I}_A \times \alpha, \quad \mathbf{L}_0 = (\mathbf{D}_0^{(A)})^{(*)}, \quad (6)$$

where \mathbf{I}_A is an $A \times A$ unit matrix and $(\mathbf{D}_0^{(A)})^{(*)}$ is equal to $\mathbf{D}_0^{(A)}$ except that diagonal elements are all 0. Therefore, we construct a new Markov chain and its corresponding infinitesimal generator \mathbf{Q} by replacing each state in Figure 3 with a set of A sub-states and use \mathbf{F} , \mathbf{B} , \mathbf{W} , and \mathbf{L}_0 to describe its state transitions. The resulting Markov chain is also a QBD process.

Figure 4(A) illustrates the transitions between the sub-states corresponding to states (x, y) , $(x, y + 1)$ and $(x + 1', y)$ in Figure 3. If we do not draw the detailed state transitions, but simply substitute λ , μ and α in Figure 3 with matrices \mathbf{F} , \mathbf{B} and \mathbf{W} , and add \mathbf{L}_0 to describe the local state transitions, we obtain the matrix-based transitions in Figure 4(B). According to Eq. (6), \mathbf{F} , \mathbf{B} , \mathbf{W} , and \mathbf{L}_0 of a system with two-state MMPP arrivals are computed as follows:

$$\mathbf{F} = \begin{bmatrix} l_1 & 0 \\ 0 & l_2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mu & 0 \\ 0 & \mu \end{bmatrix}, \\ \mathbf{W} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}, \quad \mathbf{L}_0 = \begin{bmatrix} 0 & v_1 \\ v_2 & 0 \end{bmatrix}, \quad (7)$$

³Note that the service time and the idle waiting time are exponentially distributed in our model. However, a similar method and Kronecker products can be used to generate the auxiliary matrices \mathbf{F} , \mathbf{B} , \mathbf{W} , and \mathbf{L}_0 when use a MMPP (or MAP) for the service and idle waiting processes.

where v_1, v_2, l_1 , and l_2 are the parameters of the 2-state MMPP model in Eq. (4). One can easily show the equivalence of state transitions in Figure 4(A) and Figure 4(B). The infinitesimal generator \mathbf{Q} of this new Markov chain can

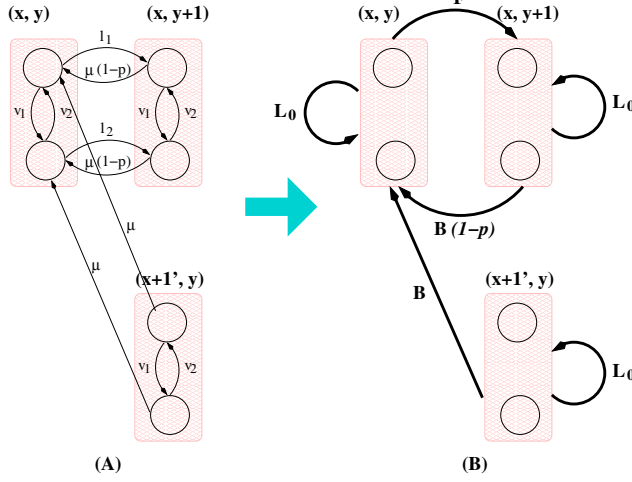


Figure 4. Changes in the Markov chain of Figure 3 when the arrival process is a 2-state MMPP.

be obtained from Figure 3. For each level i corresponding to the $(i + 1)^{th}$ column in Figure 3, the stationary state probabilities are given by the following vectors:

$$\begin{aligned} \boldsymbol{\pi}^{(i)} &= [\boldsymbol{\pi}_{(0,i)}^{(i)}, \boldsymbol{\pi}_{(1',i-1)}^{(i)}, \boldsymbol{\pi}_{(1,i-1)}^{(i)}, \dots, \boldsymbol{\pi}_{(i',0)}^{(i)}, \boldsymbol{\pi}_{(i,0)}^{(i)}], \\ &\text{for } 0 \leq i \leq X, \\ \boldsymbol{\pi}^{(i)} &= [\boldsymbol{\pi}_{(0,i)}^{(i)}, \boldsymbol{\pi}_{(1',i-1)}^{(i)}, \boldsymbol{\pi}_{(1,i-1)}^{(i)}, \dots, \boldsymbol{\pi}_{(X',i-X)}^{(i)}, \boldsymbol{\pi}_{(X,i-X)}^{(i)}], \\ &\text{for } i > X. \end{aligned}$$

$\boldsymbol{\pi}_{(x,y)}^{(i)}$ or $\boldsymbol{\pi}_{(x',y)}^{(i)}$ is a row vector of size A that corresponds to a set of sub-states under the MMPP arrival process. Then, $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$ and $\boldsymbol{\pi}\mathbf{e} = \mathbf{1}$ where $\boldsymbol{\pi} = (\boldsymbol{\pi}^{(0)}, \boldsymbol{\pi}^{(1)}, \dots, \boldsymbol{\pi}^{(X)}, \boldsymbol{\pi}^{(X+1)}, \dots)$.

We solve the QBD using the matrix geometric solution [10]. The state space is partitioned into boundary states and repetitive states. Boundary states in the QBD of Figure 3 are the union of all levels i for $0 \leq i \leq X$. We use $\boldsymbol{\pi}^{[0]}$ to denote the stationary probability vector of these states, i.e., $\boldsymbol{\pi}^{[0]} = (\boldsymbol{\pi}^{(0)}, \boldsymbol{\pi}^{(1)}, \dots, \boldsymbol{\pi}^{(X)})$. Each level i for $i > X$ represents a repetitive set of states. Key to the matrix geometric solution is that a geometric relation holds among the stationary probabilities of the repetitive states, i.e.,

$$\forall i > X, \quad \boldsymbol{\pi}^{(i)} = \boldsymbol{\pi}^{(X+1)} \cdot \mathbf{R}^{i-1}. \quad (8)$$

Here the matrix \mathbf{R} is a squared matrix of dimension equal to the cardinality of repetitive levels, and can be computed

using an iterative numerical algorithm [10]. By computing $\boldsymbol{\pi}^{[0]}$ and $\boldsymbol{\pi}^{(X+1)}$ as in [10] one can easily generate the entire infinite stationary probability vector for the QBD. Thanks to the geometric relationship in Eq. (8), several metrics can be computed in closed form formulas.

Let $\mathbf{e}^{(i)}$ be a column vector of 0's with appropriate dimension except the $(2i \cdot A + 1)^{th}$ to the $(2i \cdot A + A)^{th}$ elements that are equal to 1, and let $\mathbf{e}^{(i')}$ be another column vector of 0's except the $((2i - 1) \cdot A + 1)^{th}$ to the $((2i - 1) \cdot A + A)^{th}$ elements that are equal to 1, for $i \geq 0$. Note that all the elements of $\mathbf{e}^{(0)}$ are equal to 0. Both $\mathbf{e}^{(i)}$ and $\mathbf{e}^{(i')}$ are of size A , where A is the order of the arrival MMPP process. The average queue length of the foreground jobs $QLEN_{FG}$, the completion rate (or admission rate) of the background jobs $Comp_{BG}$, and the percentage of foreground jobs waiting behind background jobs $WaitP_{FG}$ can be calculated as follows.

$$\begin{aligned} QLEN_{FG} &= \sum_{i=1}^X \sum_{j=0}^{i-1} ((i-j) * (\boldsymbol{\pi}_{(j,i-j)}^{(i)} + \boldsymbol{\pi}_{(j',i-j)}^{(i)}) \mathbf{e}) \\ &+ \sum_{i=0}^X (X+1-i) \boldsymbol{\pi}^{(X+1)} (\mathbf{I} - \mathbf{R})^{-2} (\mathbf{e}^{(i)} + \mathbf{e}^{(i')}), \\ Comp_{BG} &= 1 - \frac{\boldsymbol{\pi}^{(X+1)} (\mathbf{I} - \mathbf{R})^{-1} \mathbf{e}^{(X)}}{1 - \sum_{i=0}^X \boldsymbol{\pi}_{(0,i)}^{(i)} \mathbf{e} - \boldsymbol{\pi}^{(X+1)} (\mathbf{I} - \mathbf{R})^{-1} \mathbf{e}^{(0)}}, \\ WaitP_{FG} &= \frac{\sum_{i=2}^X \sum_{j=1}^{i-1} \boldsymbol{\pi}_{(j',i-j)}^{(i)} + \sum_{i=1}^X \boldsymbol{\pi}^{(X+1)} (\mathbf{I} - \mathbf{R})^{-1} \mathbf{e}^{(i')}}{1 - \sum_{i=0}^X (\boldsymbol{\pi}_{(i,0)}^{(i)} + \boldsymbol{\pi}_{(i',0)}^{(i)}) \mathbf{e}}. \end{aligned}$$

5 Performance Evaluation Results

Here we use the analytic model to analyze the performance of a storage system that serves foreground and background jobs, as described in the previous section. The model is parameterized using the E-mail and Software Development traces (see Figure 1). This parameterization results in the MMPPs of Figure 2 which have different mean, CV, and dependence structure, and we consider representative.⁴

We evaluate the general performance of the system as a function of system load.⁵ Foreground load is a function of the mean of the arrival process in the system (i.e., the mean

⁴The User Account trace performs qualitatively the same as the E-mail trace because of its strong ACF structure. Results are not reported here due to lack of space.

⁵In this section we use interchangeably the terms "load" and "utilization".

of the MMPPs in Figure 2) while background load is a function of p , i.e., the probability that a foreground generates a background job upon its completion. We scale the mean of the two MMPPs in Figure 2 to obtain different foreground utilizations. We also scale the value of p between 0.1 and 0.9 to obtain different background loads. The mean “idle wait” time for a background job before starting service during an idle period is equal to the mean service time, unless otherwise stated. The background buffer size is 50.

5.1 Performance of foreground jobs

First, we report on the performance of foreground jobs. Figure 5 presents the average queue length of foreground jobs, which sharply increases as a function of foreground load. This increase is nearly insensitive to different p values, showing that foreground load determines overall system performance. Note that for long-range dependent arrivals (“E-mail” MMPP) the saturation is reached much faster than for arrivals with short-range dependence (“Software Development” MMPP). We will return to the question of how intensity in the dependence structure of the arrival process affects system performance later in this section. Figure 6 shows the percentage of foreground jobs that

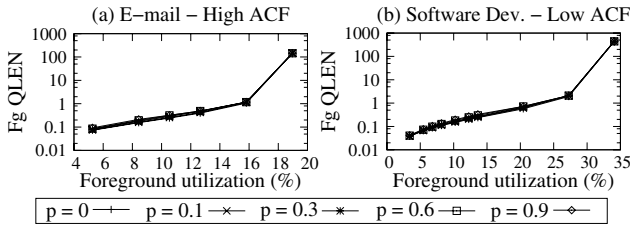


Figure 5. Average queue length of foreground jobs for the Email (a) and Software Dev. (b) traces as a function of foreground load.

are delayed because of background jobs. As background load increases, the portion of foreground jobs that are delayed increases, but as foreground load increases the portion of foreground jobs that are delayed decreases. In the worst case scenario that we present here, i.e., for $p = 0.9$, only 20% of foreground jobs are delayed, which shows that most foreground jobs maintain their expected performance. The most interesting point in Figure 6 is that when the (total) load increases beyond a certain point then the portion of foreground jobs that are affected decreases dramatically, which is explained by background jobs performance in the next subsection.

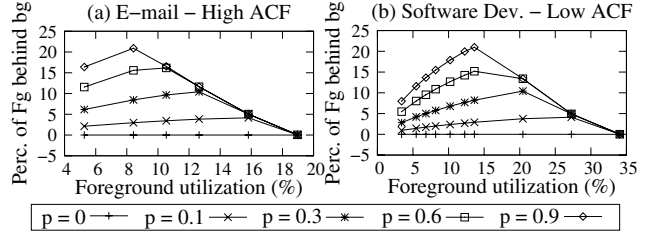


Figure 6. Portion of foreground jobs delayed by a background job for the Email (a) and Software Dev. (b) traces as a function of foreground load.

5.2 Performance of background jobs

We measure the performance of background jobs by the portion of background tasks that complete. This metric is directly related to reliability (or long term performance benefits) of background activity. Results are given in Figure 7, which shows that as load increases, the completion rate decreases to zero, independent of load or dependence structure. For arrivals with a strong dependence structure, (i.e., of “E-mail”), this point comes sooner than for arrivals with weak dependence structure, (i.e., the “Software Development”), see the range of the x-axis in Figure 7. Note that the completion rate of the background activity relates to the probability of the background buffer being full, which supports the observation that the strong dependence structure in arrivals increases the queue length of background jobs, as illustrated in Figure 8. Figure 8 shows the average queue

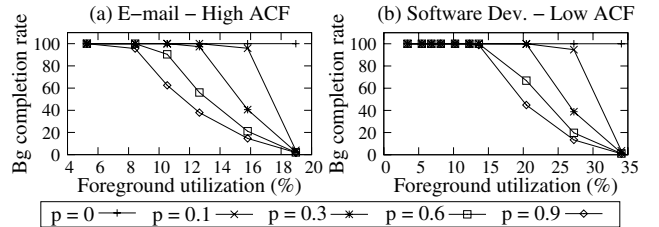


Figure 7. Completion rate for background jobs for the Email (a) and Software Dev. (b) traces as a function of foreground load.

length of background jobs. Consistent with results in Figure 7, Figure 8 shows a similar qualitative behavior across the two workloads. Quantitatively, the average queue length of the long-range dependent workload is smaller than that of the short-range dependent workload because more background jobs are dropped.

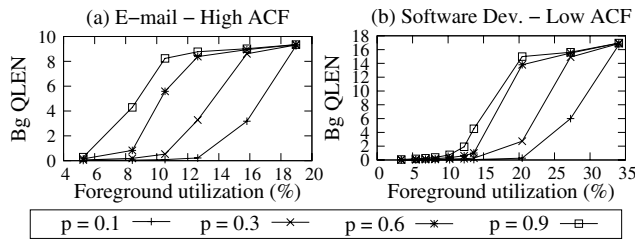


Figure 8. Average queue length of background jobs in the workloads Email (a) and Software Dev. (b) as a function of foreground load.

5.3 Effect of “idle wait” duration

An important design issue in a storage system that serves foreground and background jobs is the length of the “idle wait” period, i.e., the time that the system operates in non-work-conserving mode. The shorter the duration of “idle wait”, the higher is the performance degradation of foreground jobs.

In Figure 9, we show how the length of “idle wait” affects the average queue length of foreground jobs under different background loads. These experiments are conducted for the parameterization of the actual traces given in Figure 2. Increase in “idle wait” does improve foreground performance, because it reduces the number of foreground jobs delayed by servicing background jobs. However improve-

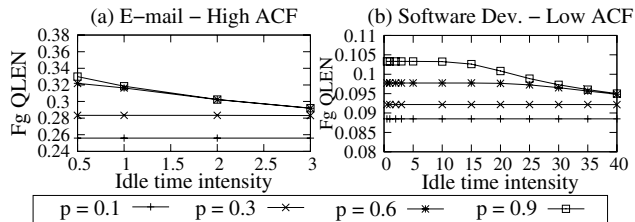


Figure 9. Foreground jobs average queue length for the Email (a) and Software Dev. (b) traces as a function of idle wait (in multiples of service time).

ment of foreground performance does come due to a considerable drop in background completion rate, as shown in Figure 10. For example in the case of “E-mail” parameterization under an “idle wait” of twice the service time and $p = 0.6$, the completion rate of background jobs drops by 20% compared to the completion rate when the idle wait is half of service time, but the foreground performance gains are as low as 6.5% (the average foreground queue lengths

are 0.32 and 0.30 when idle wait is twice of the service time and when idle wait is half of the service time, respectively). Given the long-term benefits of background activity, maintaining a small “idle wait” period, close to the average service time, is beneficial for sustaining foreground job performance and high background completion rate.

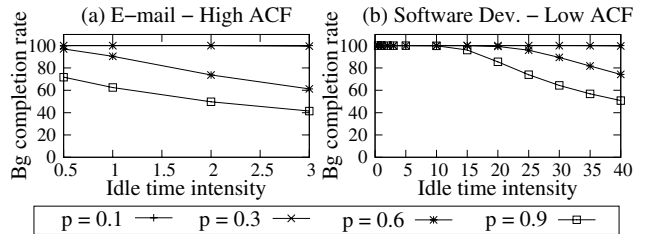


Figure 10. Completion rate for background jobs in the workloads Email (a) and Software Dev. (b) as a function of idle wait (in multiples of service time).

5.4 The impact of dependence in the arrival process

In this subsection, we analyze the effect that the arrival process has on a system with background jobs. Using only the “E-mail” workload parameterization, ⁶ we examine the performance effects of Poisson arrivals, of an Interrupted Poisson Process (IPP) (a process with high variability but no correlation [6]) and of two MMPP processes with low and high dependence structure. All these processes have the same mean and CV as the measured in the arrival process of “E-mail” trace, with the only exception of the Poisson arrival process that maintains the same mean only. Results show that the dependence structure of the arrival process determines the sensitivity of system performance toward load changes, that is, the stronger the dependence structure the higher the sensitivity toward system load.

Figure 11 shows the average queue length for foreground jobs under two different loads of background jobs, i.e., p equal to 0.3, and 0.9. There is a dramatic queue length increase under autocorrelated arrivals, that is orders of magnitude higher than the queue length increase with exponential inter-arrivals. Even at 19% foreground utilization under the strong correlated arrivals the foreground queue length reaches 100. Such queue length is reached only under 95% foreground utilization for the Poisson arrivals. For comparative purposes, we plot the results using different scales on the x-axis, separated by a vertical line. Consistent with the

⁶Qualitatively similar results can be obtained using the other two workloads and are omitted due to lack of space.

results in Figure 5, high foreground load rather than background load determines overall foreground performance. In

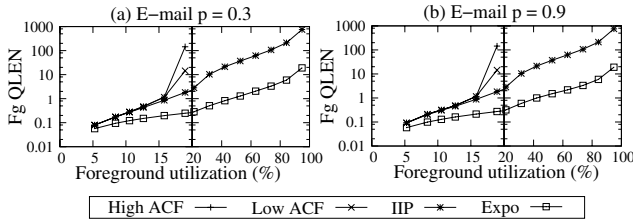


Figure 11. Average queue length for foreground jobs for the “E-mail” workload as a function of foreground load in the system.

Figure 12, we show completion rates for background jobs as a function of foreground load. There are cases when under high foreground load, there is nearly a 100% difference in performance between exponential and correlated arrivals. The system simply saturates faster under correlated arrivals and does not have the capacity to serve background tasks. Therefore, under correlated arrivals light background load should be sustained to ensure acceptable background completion rates. Finally, Figure 13 shows the percentage

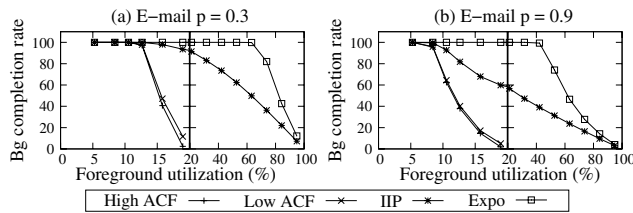


Figure 12. Completion rate of background jobs for the “E-mail” workload as a function of foreground load in the system.

of foreground jobs delayed by background jobs as a function of foreground load. Interestingly, the figure shows that the worst impact on foreground jobs is contained within a limited range which is reached faster under highly correlated arrivals than independent arrivals. In a dynamically changing environment with correlated arrivals, the system regulates itself faster to sustain foreground job performance than under independent arrivals.

To summarize, the results of this section indicate that, independent of workload characteristics, the non-preemptive background jobs minimally impact performance of foreground jobs. However sustained foreground performance under worst case scenarios is a result of low background completion rates, which suggests that background load must be kept modest to benefit system reliability or per-

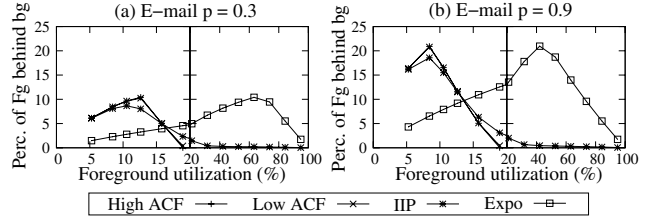


Figure 13. Portion of foreground jobs delayed by a background job for the “E-mail” workload as a function of foreground load in the system.

formance in the long-term. This sensitivity toward system changes, in particular for the background completion rate, is significantly higher for correlated than for independent arrivals, which indicates that workload burstiness is an important factor that should determine the amount of background work in the system.

6 Conclusions and Future Work

In this paper, we presented an analytic model for the evaluation of disk drives or storage systems with background jobs. Because of the non-preemptive nature of work (i.e., seeks) in disks, background work inevitably affects performance of foreground work. The proposed model allows to evaluate the trade-offs between foreground and background activities. Our model incorporates most important characteristics in storage systems workloads, including burstiness and dependence in the arrival process. The model results in a Markov chain of a QBD form that is solved using the matrix-geometric method.

Experiments show that system behavior can be qualitatively similar for independent or correlated arrivals, albeit for different utilization levels. This sensitivity to the system utilization levels strongly depends on the dependence structure in the arrival process. Although foreground performance is sustained at acceptable levels, the background completion rate suffers when background load is high under correlated arrivals. Our results suggest that the amount of background work is paramount for reliability and performance gains, and must be a function of the degree of dependence or burstiness in the arrival process. Currently, we are working on model extensions that capture more than one job priority level, i.e., different classes of background jobs.

References

- [1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Lazy verification in fault-tolerant distributed storage systems. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS)*. IEEE Press, October 2005.
- [2] E. Bachmat and J. Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *ACM Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pages 55–65. ACM Press, June 2002.
- [3] L. Eggert and J. D. Touch. Idle time scheduling with preemption intervals. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP)*, Brighton, UK, October 2005.
- [4] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *Proceedings of the Winter'95 USENIX Conference*, pages 201–222, New Orleans, LA, January 1995.
- [5] M. E. Gomez and V. Santonja. On the impact of workload burstiness on disk performance. In *Workload characterization of emerging computer applications*, pages 181–201. Kluwer Academic Publishers, 2001.
- [6] D. Green. *Departure Processes from MAP/PH/1 Queues*. PhD thesis, Department of Applied Mathematics, University of Adelaide, 1999.
- [7] Steven D. Gribble, Gurmeet Singh Manku, Drew S. Roselli, Eric A. Brewer, Timothy J. Gibson, and Ethan L. Miller. Self-similarity in file systems. In *SIGMETRICS98*, pages 141–150, Madison, Wisconsin, June 1998.
- [8] D. Heyman and D. Lucantoni. Modeling multiple IP traffic streams with rate limits. In *Proceedings of the 17th International Teletraffic Congress*, Brazil, December 2001.
- [9] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, HP Laboratories, 1991.
- [10] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. SIAM, Philadelphia PA, 1999. ASA-SIAM Series on Statistics and Applied Probability.
- [11] D. M. Lucantoni. The BMAP/G/1 queue: A tutorial. In L. Donatiello and R. Nelson, editors, *Models and Techniques for Performance Evaluation of Computer and Communication Systems*, pages 330–358. Springer-Verlag, 1993.
- [12] A. Merchant and P. S. Yu. An analytic model of reconstruction time in mirrored disks. *Performance Evaluation Journal*, 20(1-3):115–129, May 1994.
- [13] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failures. In *International Conference on Very Large Databases (VLDB)*, pages 162–173, August 1990.
- [14] M. F. Neuts. *Algorithmic Probability: A Collection of Problems*. Chapman and Hall, 1995.
- [15] Z. Niu, T. Shu, and Y. Takahashi. A vacation queue with setup and close-down times and batch markovian arrival processes. *Perform. Evaluation*, 54(3):225–248, 2003.
- [16] A. Riska and E. Riedel. Disk drive level workload characterization. In *USENIX Annual Conference*, Boston, MA, May 2006.
- [17] T. J. E. Schwarz, Q. Xin, E. L. Miller, D. D. E. Long, A. Hospodor, and S. Ng. Disk scrubbing in large archival storage systems. In *Proceedings of the International Symposium on Modeling and Simulation of Computer and Communications Systems (MASCOTS)*. IEEE Press, 2004.
- [18] M. Seltzer, P. Chen, and J. Osterhout. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Technical Conference*, pages 313–323, Washington, DC, 1990.
- [19] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID. In *Proceedings of the Third USENIX Symposium on File and Storage Technologies (FAST '04)*, San Francisco, CA, March 2004.
- [20] H. Takagi. *Queuing Analysis Volume 1: Vacations and Priority Systems*. North-Holland, New York, 1991.
- [21] E. Thereska, J. Schindler, J. Bucy, B. Salmon, C. R. Lumb, and G. R. Ganger. A framework for building unobtrusive disk maintenance applications. In *the 3rd USENIX Conference on File and Storage Technologies (FAST)*, August 2004.
- [22] A. Thomasian and V. F. Nicola. Performance evaluation of a threshold policy for scheduling readers and writers. *IEEE Transactions on Computers*, 42(1):83–98, 1993.
- [23] E. Xu and A. S. Alfa. A vacation model for the non-saturated readers and writers system with a threshold policy. *Performance Evaluation*, 50(4):233–244, 2002.