# Queuing Models of Tertiary Storage[1]

**Theodore Johnson**
Dept. of CISE, University of Florida
AT&T Research
ted@cis.ufl.edu

## Abstract

Large scale scientific projects generate and use huge amounts of data. For example, the NASA EOSDIS project is expected to archive one petabyte per year of raw satellite data. This data is made automatically available for processing into higher level data products and for dissemination to the scientific community. Such large volumes of data can only be stored in robotic storage libraries (RSLs) for near-line access. A characteristic of RSLs is the use of a robot arm that transfers media between a storage rack and the read/write drives, thus multiplying the capacity of the system.

The performance of the RSLs can be a critical limiting factor of the performance of the archive system. However, the many interacting components of an RSL make a performance analysis difficult. In addition, different RSL components can have widely varying performance characteristics. This paper describes our work to develop performance models of a RSL. We first develop a performance model of a RSL in isolation. Next, we show how the RSL model can be incorporated into a queuing network model. We use the models to make some example performance studies of archive systems.

The models described in this paper, developed for the NASA EOSDIS project, are implemented in C with a well-defined interface. The source code, accompanying documentation, and also sample JAVA applets, are available at:

***http://www.cis.ufl.edu/~ted/***

## Introduction

Large scale scientific projects generate and use huge amounts of data. For example, the NASA EOSDIS project is expected to archive one petabyte per year of raw satellite data [KBCH94]. This data is made automatically available for processing into higher level data products and for dissemination to the scientific community (see, for example, the reports in [ESDIS]). Automatic management of such large data sets requires the use of tertiary storage, typically implemented using *robotic storage libraries* (RSLs). In addition to EOSDIS and related projects, many organizations and scientific disciplines make use of mass storage archives (for example high energy physics [Lu95] and digital libraries [CoHu93]).

1 This research was funded by a grant from NASA #10-77556

The database community has also become interested in the use of RSLs [DSF94,CHL93,Sequoia2k,Sa95]. This interest is motivated in part by scientific database problems such as EOSDIS. Another motivation for integrating RSLs with on-line database systems is to facilitate data warehousing.

Tertiary storage is required when the managed data set becomes too large to store economically with conventional magnetic disk devices. The point at which tertiary storage becomes necessary is an economic tradeoff. Currently, it seems that tertiary storage is needed to manage more than a terabyte of data. A RSL is much slower than magnetic disk storage, and data access latencies can run into minutes even on unloaded systems. However, RSL-resident data can be accessed automatically. Hierarchical storage management systems, such as Unitree, Filestore, and Amass, provide the illusion that the RSL is an extension of the file system. Access to archived data incurs a short delay. The storage capacity of a data system can also be increased by using off-line storage -- i.e. tape racks with human operators. Access latencies with off-line storage can be very large, ranging into hours or days, but the data storage capacity is limited only by the size of the warehouse that one can afford to rent. Since RSL provides data volumes and access latencies between those provided by on-line and off-line storage, it is often referred to as *near-line storage*. A cost analysis of on-line, near-line, and off-line archives can be found in [KGT90].

A characteristic of RSLs is the use of removable media and a robot arm. The removable media (e.g. magnetic tape, optical disk, etc. are normally located in a storage rack. To service a request for a file, the robot arm fetches the proper media from the storage rack and delivers it to a read/write drive. The media is accessed in the normal way to fetch the file. Finally, the media is returned to the storage rack. The capacity of RSL is the product of the capacity of the media and the size of the storage rack. Recent magnetic tapes have a data capacity on the order of 10 Gbytes, and storage rack sizes range from 10 to 1000 media (approximately). The time to fetch and mount the media which holds the requested file can be a large component of the access latency.

The performance of the RSLs can be a critical limiting factor of the performance of the archive system. Given the high data request rates expected for EOSDIS, attention to handling these requests efficiently is critical [KBCH94,ESDIS]. However, the many interacting components of a RSL make a performance analysis difficult. In addition, different RSL components can have widely varying performance characteristics.

This paper describes our work to develop performance models of tertiary storage. We first develop a performance model of a RSL in isolation. Next, we show how the RSL model can be incorporated into a queuing network model. Finally, we model fork-join jobs to study the tradeoffs of using multiple devices. We use the models to make some example performance studies of archive systems.

The models described in this paper, developed for the NASA EOSDIS project, are implemented in C with a well-defined interface. The source code, accompanying documentation, and also example JAVA applets, are available through:

***http://www.cis.ufl.edu/~ted/***

## Previous Work

Considerable work has been done to develop performance models of mass storage. Rahm [Rh92] presents a simulation study of a database system with a hierarchy of storage devices. Ramakrishnan and Emer [RE89] present a queuing model of a client/server file system. Drakopoulos and Merges [DM92] present a closed queuing model of a client/server storage system with hierarchical storage. Kelly, Haynes, and Ernest [KHE91] discuss a benchmark for network storage systems. Hauser, Rivera, and Thoma [HRT91] discuss the performance of their networked WORM server.

Some work has been done to characterize the performance of mass storage devices. Waters [Wa74] presents a validated model of seek times in hard disk drives. More recently, Ruemmler and Wilkes [ReWi94] present a detailed model of a modern disk drive, and discuss the difficulties inherent in I/O modeling. Christodoulakis and Ford [CF88] and Christodoulakis [Ch87] present analytical performance models of optical drives. Chinnaswamy [Ch92] presents performance models of a streaming tape drive to investigate the benefit of a cache.

Models of disk arrays resemble the models presented in this paper in several aspects. Burkhard, Claffy, and Schwarz [BCS91] present a simulation study of a disk array scheme. Lee and Katz [LK93] and Yang, Hu and Yang [YHY94] present analytical models of disk arrays. Chen et al. [CLGKP94] and Thomasian [Th95] present surveys of research in RAID modeling.

Several authors have modeled a RSL. Butturini [Bu88] presents the results of a simulation study of an optical disk jukebox system. Hevner [He85] presents a model of an optical jukebox that is used for a database application. Howard [Ho92] gives a performance model for data duplication from an archive. Finestead and Yeager [FY92] give performance measurements of a Unitree file server at the National Center for Supercomputer Applications. Hull and Ranade [HR93] present measurements of tape loading and unloading, and of data throughput, in a tape silo. Bedet et al. [Be93] discuss the results of a detailed simulation model of the Goddard DAAC. Pentakalos, Menasce, Halem, and Yesha [PMHY95] develop a queuing network model that incorporates a RSL. Daigle, Kuehl, and Langford [DKL90] present a queuing model of an optical disk jukebox. Golubchik, Muntz and Watson [GMW95] analyze tape striping on an RSL.

The analyses most closely related to the one in this paper are [PMHY95,DKL90,GMW95]. The analysis in [DKL90] gives a detailed model of access times to data on an optical platter. However, only one drive is permitted and contention for the robotic arm is not modeled. In [PMHY95], the authors present a detailed model of a data center, incorporating RAID disk caches and user computation. However, the authors assume that contention for the drives in the

RSL is negligible, and model the RSL as a delay server. Contention for the robotic arm due to batch arrivals is modeled in [GMW95], but contention between jobs is not modeled.

The contribution of this work is to present a validated model of a RSL that accounts for batch arrivals, multiple drives, contention for the robotic arm, and realistic operation. We show how the model can be used to make a variety of data layout and device comparison studies. Finally, we show how to incorporate the RSL model into a queuing network model.

## Model of a Robotic Storage Library

Our model of a RSL is illustrated in Figure 1. Previous studies of mass storage archive log files (see, for example, [Jo95a,DKL90]) indicate that requests to a mass storage device come in batches. This study has been corroborated by our studies of access to preliminary versions of the EOSDIS archives (the V0 archives) [Bedet96, DunhamNorth96]. As a result, our RSL model uses batch arrivals.

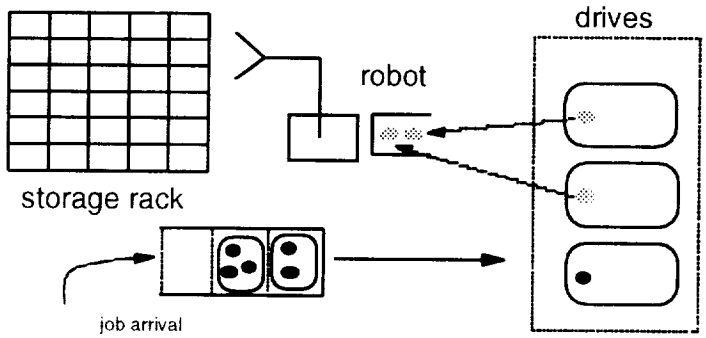


**Figure 1**

A user requests that f files be loaded into on-line storage, and these files are distributed over m media in the RSL. The request is satisfied when every file has been loaded into on-line storage. So, a user *request* consists of *m jobs*, each of which must be completed before the request in finished. A RSL consists of $n_d$ drives, each of which can read or write any of the media in the RSL[2], a storage rack containing the removable media, and a robot arm for transferring the media between the drives and the storage rack. The model of a RSL is illustrated in Figure 1.

---

2 In some installations, a subset of the drives are designated as read-only or write-only. We will address this complication in a later version of the model.

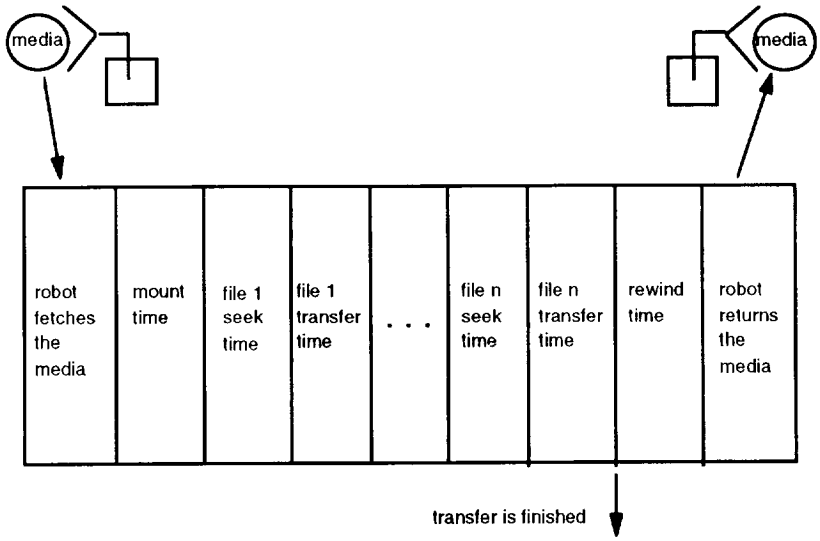


**Figure 2**

The steps taken by a drive in retrieving files from a media is illustrated in Figure 2. When a request arrives, its jobs are placed in the job queue. If there are jobs in the RSL queue and a drive is idle, the drive allocates one of the jobs for execution. First, the robot arm fetches the appropriate media from the storage area and loads it into the drive. If the robot arm is busy serving other drives, the drive must wait for service. After the media is brought to the drive, it must be mounted. For every file of interest on the media, the drive must seek to the start of the file, spend a settling time for precise positioning and opening communications channels, and then transfer the file to on-line storage. After all files have been transferred, the media is rewound and returned to the storage rack by the robot arm. However, the job is finished once all of the files have been transferred.

In the next section, we briefly discuss our analytical performance model of a RSL system (a more detailed discussion can be found in [Johnson96]). In this preliminary model, we make the following assumptions:

- Requests arrive in a Poisson process.
- The distribution of the number of media per request and the number of files per request must be specified. In the model discussion, we assume that the number of media per request and the number of files per request have geometric distributions. These can be replaced by user-specified distributions (e.g. empirically determined), but at the cost of requiring the user to specify more parameters.

- The RSL can contains one robot arm. The robot arm can access every media, and every drive.
- Every drive can read and write every media.
- Requests (i.e., jobs) are serviced first-come-first-serve[3].
- The service for a request is completed when the last file of the batch has been read (written).
- Network or communication channel contention is not significant[4].
- Service times at the drives are independent.

## Analytical model

A RSL presents many difficulties for performance modeling, including batch arrivals, multiple servers, derived parameters, and interacting components. The primary component of the RSL model, the $M^X/G/c$ queue, has been studied and solved in the literature [Tijms94]. Solving the actual $M^X/G/c$ queue is intractable, so the solution technique is to interpolate between the results for the $M^X/M/c$ queue and the $M^X/D/c$ queue using the coefficient of variation of the service time as the interpolation parameter. Because of the potential complexity of the batch arrival distributions, we do not use explicit (i.e., generating function) formulas. Instead, we numerically solve the recurrence equation that defines the state occupancy probabilities. If the occupancy probabilities of the first N states must be computed for an error bound of e, then solving the $M^X/M/c$ queue requires $O(N^2)$ time and solving the $M^X/D/c$ queue requires $O(N^3)$ time.

Fortunately, we can take advantage of the nature of the problem to speed up the solution times. The state occupancy distributions eventually converge to a geometric distribution (i.e., $p_N = t_0 p_{N-1}$). Therefore the recurrence equations only need to be solved up the first $N_0$ states, and the remainder can be computed using the $t_0$ ratio (or perhaps the performance metrics can be computed directly). $N_0$ depends primarily on the distribution of the size of the batch arrival. Fortunately, the batch arrival distribution will have a short tail -- one cannot request that more media than exist in the storage rack be mounted, and usually only a few media are required to

---

3 A simple optimization is to load files for all requests once a media has been mounted. We assume this situation has a negligible impact on performance in this model.

4 Potential model users indicated that communication channel contention is not a problem for their systems. Communication contention can be incorporated into the seek times or mount times using standard techniques [Ka92].

satisfy a request. By using these tricks, we implemented batch queue solvers that are fast enough to be incorporated into a higher level model which calls them many times.

We model the RSL as an $M^X/G/c$ queue -- that is, a queue with Poisson batch arrivals, general service time distribution, and $c$ servers. The parameters of a $M^X/G/c$ queue are:

- Arrival rate
- Mean service time
- Coefficient of variation of service time
- Batch size distribution
- Number of servers

All but the service time distribution are input parameters, so our analysis is focused on how to compute the expected service time, $E_d$ and the coefficient of variation $cv_d$. To compute queue length distributions and expected waiting times properly, we need to compute the time that a drive is unable to serve other jobs per media that it serves. This period includes the time to fetch the media, mount it, seek to each file, transfer each file, rewind and eject the media, and return it to the storage rack. We will incorporate the time to return the media as part of the media fetch time, so we have:

drive service = (robot fetch)+(mount time)+(seek time)+(transfer time)+(rewind time)

Since we are interested in the response time of the last job in the batch to finish (i.e., instead of the average job), we need to modify the response time computation. An efficient algorithm for computing the response time of the last job in the batch is given in [Ka92]. The modified $M^X/G/c$ queue provides the batch response time $R_{batch}$, the drive utilization $r_d$, and $p_d(0),...,p_d(n_d-1)$, the probability that 0,... , $n_d-1$ servers are busy on a request arrival (where $n_d$ is the number of drives in the RSL). The effective drive service time (and $p_d(.)$) depends on the robot response time, which in turn depends on $p_d(.)$. Finally, the job is finished when the last file has been transferred, there is no need to wait for the tape rewind. Therefore:

$$R_{request} = R_{batch} - E_{rwd}$$

For more information about the derivations in the model, please see our other report [Johnson96].

## Interfaces

The RSL solver consists of a number of modules, mostly consisting of procedures to solve $M^x/G/c$ queue. Figure 3 shows a map of the procedure calls. The functions rslsolve and rslsolve_f are the entries to the RSL solver.

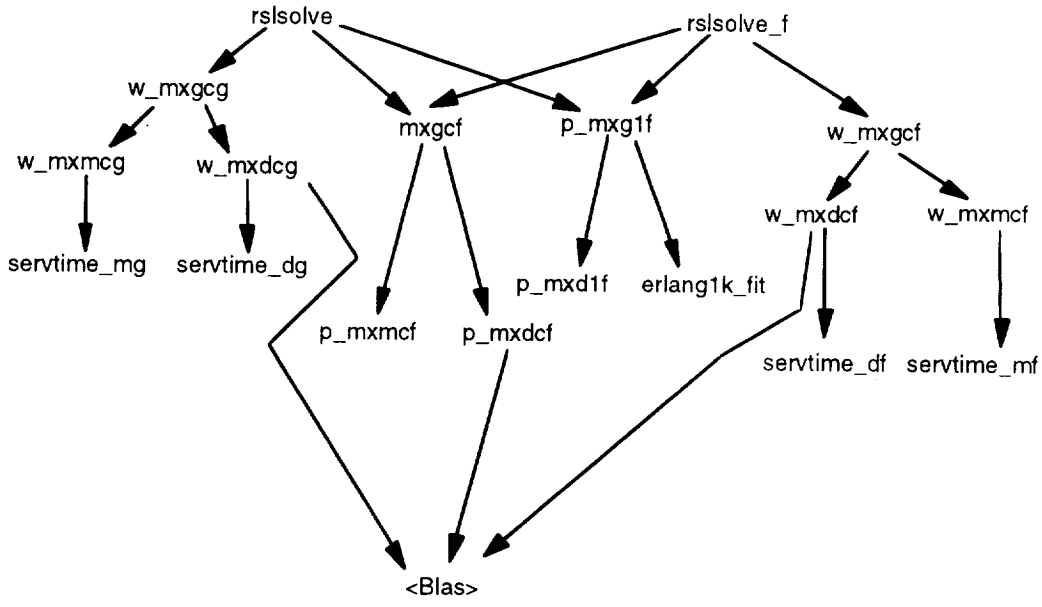


**Figure 3**

The prototype for the rslsolve function is

```
void rslsolve(float l,float fr,float mr,int nd,int nr,float,Etr,
 float Vtr,float tmt,float Xb,float Esz,float Vsz,float Erwd,
 float Vrwd,float (* seekfun)(int,float*,float*),int ncust,
  float *drho, float *dR, float *dW, float* dRv, float *dWv,
  float *rrho,float *rR, float *rW,float *dmu, float *dV,
  float *basemu,float *drbusy,int DEBUG)
```

where the input parameters are:

- l is the arrival rate.
- fr is the average number of files per request.
- mr is the average number of media per request
- nd is the number of drives in the RSL.
- nr is the number of robot arms.
- Etr is the is the average robot fetch time.
- Vtr is the variance of the robot fetch time.
- tmt is the media mount time.
- Xb is the transfer rate.
- Esz is the average file size.
- Vsz is the variance in the file size.
- Erwd is the average tape rewind and unmount time
- Vrwd is the variance in the average tape rewind and unmount time.
- moments of the seek time, given that nfiles are loaded.
- DEBUG is set true to print a trace.

And the output parameters are:

- drho is the drive utilization
- dR is the batch job response time
- dW is the batch job waiting time
- dRv is the variance in job response time
- dWv is the variance in job waiting time
- rrho is the robot utilization
- rR is the avg. robot response time
- rW is the avg robot waiting time
- dmu is the drive service time
- dV is the variance of drive service
- basemu is the base drive service time
- drbusy is an array where drbusy[i] is the long-term probability that *I* drives are busy, *0* <= *I* <= *nd-1* (drbusy must point to the storage location for the array when the call is made).

The rslsolve function assumes that the number of media per request and the number of files per media have a geometric distribution. In the rslsolve_f function, the user supplies the distribution of the number of media per request, but the number of files per media has a geometric distribution. The prototype for the rslsolve_f function is:

```
void rslsolve_f(float l,float fr, int nd,int nr,float,Etr,
 float Vtr,float tmt,float Xb,float Esz,float Vsz,float Erwd,
 float Vrwd, float *bd, int bmax,
 float (* seekfun)(int,float*,float*),int ncust,
  float *drho, float *dR, float *dW, float* dRv, float *dWv,
  float *rrho,float *rR, float *rW,float *dmu, float *dV,
  float *basemu,float *drbusy,int DEBUG)
```

where

- bd is an array where bd[i] is the probability that a request requires *I* media.
- bmax is the largest number of media required to service a request.

## Validation Study

We wrote a simple RSL simulator. The simulation accepts batch arrivals, requires that a robot unload and fetch a media before a drive can service a job, handles multiple drives, and accounts for media rewind times. The drive service time, except for the robot arm component, is sampled from an Erlang distribution.

We used the following values of the parameters in the validation study:

- fr = 20.
- bd[.] : Geometric distribution.
- nd = 4.
- Etr = 10.0 seconds.
- Vtr = 10.0.
- tmt = 10.0 seconds.
- X_b = 1.0 Mbyte/sec.

We ran four sets of experiments to test the model. In the ``large files" experiments, Esz=50, Vsz=100, Tfs=50, and Tsl=1. In the ``small files" experiments, Esz=5, Vsz=10, Tfs=20, and Tsl=2. We tested the model with mr=2 and mr=6.

The results of the validation study are shown in Figures 4 through 7. In each case there is close agreement between the analytical and the simulation models. The most difficult case is when the files are small and distributed over an average of six media, because the robot fetch times constitute a large portion of the drive service times (about 22% of the total drive service time when the robotic arm waiting time is added). However, the analytical model is accurate enough to predict response times and drive utilizations. Charts comparing analytical and simulation drive utilizations are shown in Figures 8 and 9.

## Performance Study

A performance model is useful for studying implementation alternatives. In this section, we present two sample performance studies based on the RSL model.

### Clustering

Conventional wisdom holds that striping or declustering is necessary for obtaining high transfer rates from tertiary storage (by making use of parallel I/O). So, one should spread the files of a typical request around as many media as possible. Conventional wisdom also holds that swapping media is a source of great inefficiency in RSL access, so that one should try to ensure that the files of a typical request are placed on as few media as possible.

Neither argument is convincing, unless one has a predictive performance model. We ran the "small files" experiment with mr ranging between 1.2 and 8. In Figure 10, we plot the response time of a request against the number of media per request for varying arrival rates (A similar chart can be found in an analysis of tape striping [GMW95]). For low arrival rates, setting mr to approximately nd produces the best results. When lambda=.0001, setting mr =3 results in a 22% lower response time than setting mr=1.2. For high arrival rates, setting mr=2 gives lower response times than other choices.

In Figure 11, we plot the drive utilization against mr for varying arrival rates. Increasing mr causes a linear increase in the drive utilization. As the arrival rate increases, it becomes less likely that all nd drives are available to service the request. So, distributing the files over a smaller number of media reduces queuing delays. If the demand on the RSL is expected to be close to the device's capacity, then mr should be small to increase the maximum throughput of the device.

The question of whether to cluster or decluster the files on the media can be summarized as:

- If the expected drive utilization is low [PMHY95] and fast response is important, then declustering can be a good strategy. However, the decrease in transfer times must be larger than the increase in queuing delays.

- If high throughput is important, clustering is a good strategy.

### Number of Drives

Many RSLs allow the user to install a ranging number of drives. Adding drives to a RSL can improve the performance of the device. But after a threshold, adding drives does not significantly improve performance.

We ran a sample study using the ``small files" parameters and four media per request. In Figure 12, we plot the response time versus the arrival rate for a number of drives varying between 2 and 8. Adding a drive significantly improves performance up to four drives, but gives less benefit after four drives. In Figure 13, we plot the drive utilization against the arrival rate. Adding a drive to the RSL increases the capacity of the device. However, the robot arm will start to become a bottleneck. This can be seen in the non-linear increase in utilization of some of the curves, for example for nd=8.

## Computing Response Times for Particular Jobs

The rslsolve function computes the response time for an average request. However, it is often necessary to compute the expected response time for a particular request (with a particular number of media to be accessed, etc.). The servtime* routines use information computed by rslsolve to compute response times for particular requests. The prototype for the servtime_mf function is:

```
void servtime_mf(float* bd,float bmax,float* pr,float m, int c,
  float* Eserv, float* M2serv)
```

where the input parameters are:

- bd - batch arrival distribution.
- bmax - largest batch arrival.
- pr - pr[k] is the probability that k servers are busy, $0 <= k < nd$
- m -service rate.
- c - number of servers.

And the output parameters are:

- Eserv - average request service time.

- M2serv - 2nd moment of request service time.

The function servtime_mf assumes that the service time has an exponential distribution. A similar routine, servtime_df, assumes that the service time is deterministic. Results for general service time distributions are obtained through interpolation.

We consider the following application. A large scale data center is likely to have multiple RSLs. The devices might be acquired to handle data center growth, or multiple small devices might be less expensive than a single large device. In this section, we discuss an approximation to the request response time when the request is served by multiple RSLs.

If a request is services by two different RSLs, the request is finished when both devices have completed their part of the request. Since we assume that requests are independent, we need to analyze a fork-join queue with interfering traffic. Thomasian and Tantawi [ThTa94,Th96] have found that a good approximation to the response time of the fork-join job is to take the maximum of the response times of each device (we assume an Erlang distribution on the response time when computing order statistics).

For an experiment, we applied the ``large files'' workload to two RSLs, with both receiving the same arrival rate. We considered a request that required files from six media. In Figure 14, we plot the response time of this request against the arrival rate, and varied the number of media that must be loaded from each device. The results show that when the load on the tertiary storage devices is low, it is better to divide the request evenly between the two devices. But, when the load is high it is better to use one device only. The reason for this result is that splitting the request between the two devices provides parallel I/O, but if the request load is high, then the variance in the response times becomes large. Thus, the decision to allocate files so that most requests use only one device or that most requests use both devices depends on the expected load placed on the devices.

## Queuing Network Model

A mass storage data system consists of many components in addition to the RSLs. Typical hierarchical storage management systems use a database to track file to media location mappings, and maintain a sizeable staging and caching area. The computing centers that use tertiary storage often have large scale computing tasks. For example, EOSDIS archives must perform *product generation* to filter, correct, remap, and fuse satellite images (see the reports in [ESDIS], and also the discussion in [PMHY95]).

To capture the effects of RSLs in computing systems, we need to integrate the RSL model into a queuing network model. The typical approach for incorporating devices with unusual response time characteristics into a queuing network model is to use mean value analysis (MVA), and develop a MVA recurrence for the device in question [Ka92]. However, it is difficult to develop

such a recurrence even for multiple server devices. Therefore, we take the approach of integrating the open RSL queue into a MVA model.

Although the RSL model solver is fast (about 2 seconds of execution time), an exact MVA solver requires an iteration over every possible population vector. If the population is large and there are many job classes, solution times become intolerably large. We instead used an approximate MVA solver, making use of Schweitzer's approximation on queue lengths and Bard's approximation for the load dependent servers [Ka92]. The approximate MVA solver built using these approximations compute the throughput at each iteration, which we use as the arrival rate at the RSL (after scaling by the visit ratio).

The function that solves the queuing network model is closedqn_rsl. Its prototype is:

```
int closedqn_rsl( int M,  int K,  double D[MaxM][MaxClass], double* N,
 int* servtype, double alpha[MaxLD][MaxPop], struct rslparamstr* rslparam,
 double visit[MaxM][MaxClass],
   double  lam_out[MaxClass], double  Rloc[MaxM][MaxClass], double* U)
```

where the input parameters are:

- M - number of servers.
- K - number of classes.
- D - D[k][r] is the service demand of a class r job at server k.
- N - N[r] is the number of customers of class r.
- servtype - servtype[k] encodes server k's type, and possibly points to additional parameters.
- alpha - alpha[l][*] are the service rate multipliers of load dependent server l.
- rslparam - rslparam[l] is a structure containing the service parameters of RSL l.
- visit - visit[k][r] is the number of times a class r job visits server k.

And the output parameters are:

- lam_out - lam_out[r] is the throughput of class r jobs.
- Rloc - Rloc[k][r] is the residence time of a class r job at server k, per visit.
- U - U[k] is the utilization of server k.

Incorporating an open queuing model into a closed queuing network can introduce inaccuracies (we implemented some heuristic corrections). To test the accuracy of the approximate MVA model, we simulated a computer system with a RSL and three other queuing devices. The requests to the RSL used the ``large file" workload, and every customer submits a single request

to the RSL per task execution. There are three queuing devices, with per-task service demands of 250, 400, and 350 units of work, respectively.

We plot the response time of the RSL against the number of customers in the system in Figure 15 for a sleep time of 5000 and 9000. The model is accurate even for a small number of customers. However, the accuracy declines when the number of customers is large and the sleep time is small. This problem is occurring because one of the queuing devices is saturated, and the approximate MVA solver becomes inaccurate in these situations.

## Conclusions

We have developed an analytical model of a robotic storage library and validated the model by comparison to simulations. The RSL consists of a storage rack for removable media, a set of drives that read and write the media, and a robotic arm that transfers the media between the storage rack and the drives.

The RSL model can be used for many useful studies. We provide examples of data layout and device selection studies. A RSL is used as a part of a larger computing system. We incorporated the RSL solver into an approximate MVA queuing network model, and validated the model by comparison to a simulation.

We have developed this model to support NASA's EOSDIS on-line archiving efforts. Future work will be directed towards refining the model and providing studies useful to archive sites. This work includes further model refinements, and encapsulating the queuing model solvers into Java applets that perform particular analyses.

## Acknowledgments

We'd like to thank Ben Kobler and Chris Daly of NASA GSFC, and Bob Howard of Hughes for their comments, and Alex Thomasian for his advice regarding fork-join jobs.

## Bibliography

**[Bedet96]** T. Johnson and J.J. Bedet. Analysis of the access patterns at the GSFC Distributed Active Archive Center. In *Proc. Fifth Goddard Conf. on Mass Storage Systems and Technologies, 1996*

**[Be93]** J.J. Bedet et al. Simulation of a data archival and distribution system at GSFC. In *Goddard Conference on Mass Storage Systems and Technology*, NASA CP 3262, pages 139-160, 1993.

**[BCS91]** W.A. Burkhard, K.C. Claffy, and T.J.E. Schwarz. Performance of balanced array schemes. In *Mass Storage Systems Symposium*, pages 45-50, 1991.

**[Bu88]** R.S. Butturini. Performance simulation of a high capacity optical disk system. In *Mass Storage Systems Symposium*, pages 147-153, 1988.

**[CHL93]** M.J. Carey, L.M. Haas, and M. Linvy. Tapes hold data too: Challenges of tuples on tertiary store. In *Proc. ACM SIGMOD*, pages 413-418, 1993.

**[CLGKP94]** M.P. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson. RAID: High-performance reliable secondary memory. *Computing Surveys*, 26(2):145-185, 1994.

**[Ch92]** V. Chinnaswamy. Analysis of cache for streaming tape drive. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, NASA CP-3198, Vol. I, pages 299-310, 1992.

**[Ch87]** S. Christodoulakis. Analysis of retrieval performance for records and objects using optical disk technology. *ACM Transactions on Database Systems*, 12(2):137-169, 1987.

**[CF88]** S. Christodoulakis and D.A. Ford. Performance analysis and fundamental performance tradeoffs for clv optical disks. In *ACM SIGMOD*, pages 286-294, 1988.

**[CoHu93]** R.A. Coyne and H. Hulen. Toward a digital library strategy for a national information infrastructure. In *Proc. 3rd NASA Godddard Conf. on Mass Storage Systems and Technologies*, NASA CP-3262, pages 15-18, 1993.

**[DKL90]** J.N. Daigle, R.B. Kuehl, and J.D. Langford. Queuing analysis of an optical disk jukebox-based office system. *IEEE Trans. on Computers*, 39(6):819-828, 1990.

**[DSF94]** J. Dozier, M. Stonebraker, and J. Frew. Sequoia 2000: A next-generation information system for the study of global change. In *Proc. 13th IEEE Mass Storage Systems Symposium*, pages 47-53, 1994.

**[DM92]** E. Drakopoulos and M.J. Merges. Performance analysis of client-server storage systems. I*EEE Transactions on Computers*, 41(11):1442-1452, 1992.

**[DunhamNorth96]** J. Dunham and B. North. EOSDIS statistics collection and reporting system, 1996. Available by anonymous FTP at eos.nasa.gov:/EosDis/Daacs/Statistics.

**[ESDIS]** ESDIS document catalog. http://spsosun.gsfc.nasa.gov/ESDIS_Docs.html.

**[FY92]** A. Finestead and N. Yeager. Performance of a distributed superscaler storage server. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, NASA CP- 3198, Vol. II, pages 573-580, 1992.

**[GMW95]** L. Golubchik, R.R. Muntz, and R.W. Watson. Analysis of striping techniques in robotic storage libraries. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 225-238, 1995.

**[HRT91]** S.E. Hauser, C. Rivera, and G.R. Thoma. Factors affecting the performance of a dos-based WORM file server. In *Mass Storage Systems Symposium*, pages 33-37, 1991.

**[He85]** A.R. Hevner. Evaluation of optical disk systems for very large database applications. In *ACM SIGMETRICS* conference, pages 166-172, 1985.

**[Ho92]** K. Howard. High speed data duplication/data distribution - an adjunct to the mass storage equation. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, pages 123-133, 1992.

**[HR93]** G. Hull and S. Ranade. Performance measurements and operational characteristics of the Storage Tek ACS 4400 tape library with the Cray Y-MP EL. In *Proc. NASA Goddard Conference on Mass Storage Systems and Technology*, pages 111-122, 1993.

**[Jo95a]** T. Johnson. Analysis of the request patterns to the nssdc on-line archive. In Proc. 4th *NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1995.

**[Johnson96]** T. Johnson. An Analytical Performance Model of Robotic Storage Libraries. In *Performance '96*, 1996.

**[Ka92]** K. Kant. *Introduction to Computer System Performance Evaluation.* McGraw Hill, 1992.

**[KHE91]** S.M. Kelly, R.A. Haynes, and M.J. Ernest. Benchmarking a network storage service. In *Mass Storage Systems Symposium*, pages 38-44, 1991.

**[KGT90]** K.F. Klenk, J.L. Green, and L.A. Treinish. A Cost Model for NASA Data Archiving. Technical Report 90-08, National Space Science Data Center, NASA Goddard Space Flight Center, 1990.

**[KBCH94]** B. Kobler, J. Berbert, P. Caulk, and P.C. Hariharan. Architecture and design of storage and data management for the NASA Earth Observing System Data and Information System (EOSDIS). In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 65-78, 1995.

**[LK93]** E.K. Lee and R.H. Katz. An analytic performance model of disk arrays. In *ACM SIGMETRICS*, pages 98-109, 1993.

**[Lu95]** L. Lueking. Managing and serving a multi-terabyte data set at the Fermilab D0 experiment. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 200-208, 1995.

**[PMHY95]** O.I. Pentakalos, D.A. Menasce, M. Halem, and Y. Yesha. Analytical performance modeling of hierarchical mass storage systems. Technical Report TR-CS-96-01, Dept. of Computer Science, University of Maryland, 1995. A short version appears in the *14th IEEE Mass Storage Symposium* proceedings.

**[Rh92]** E. Rahm. Performance evaluation of extended storage architectures for transaction processing. In *ACM SIGMOD*, pages 308-317, 1992.

**[RE89]** K.K. Ramakrishnan and J.S. Emer. Performance analysis of mass storage service alternatives for distributed systems. *IEEE Trans. on Software Engineering*, 15(2):120-133, 1989.

**[ReWi94]** C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17-28, 1994.

**[Sa95]** S. Sarawagi. Database Systems for Efficient Access to Tertiary Memory. In *Proc. 14th IEEE Mass Storage Systems Symposium*, pages 120-126, 1995.

**[Sequoia2k]** Sequoia 2000 home page. http://s2k-ftp.cs.berkeley.edu:8000/.

**[Th95]** A. Thomasian. Surveyor's forum: High performance secondary memory. *Computing Surveys*, 27(2):292-295, 1995.

**[Th96]** A. Thomasian. Approximate analyses for fork/join synchronization in RAID 5. *Computer Systems: Science and Engineering*, 1996. To appear.

**[ThTa94]** A. Thomasian and A.N. Tantawi. Approximate solutions for M/G/1 fork/join synchronization. In *Proc. 1994 Winter Simulation Conference*, 1994.

**[Tijms94]** H.C. Tijms. *Stochastic Models: An Algorithmic Approach*. Wiley, 1994.

**[Wa74]** S.J. Waters. Estimating disk seeks. *The Computer Journal*, 18(1):12-17, 1974.

**[YHY94]** T. Yang, S. Hu, and Q. Yang. A closed-form formula for queuing delays in disk arrays. In *Proc. Intl. Conf. on Parallel Processing*, pages II:189-192, 1994.

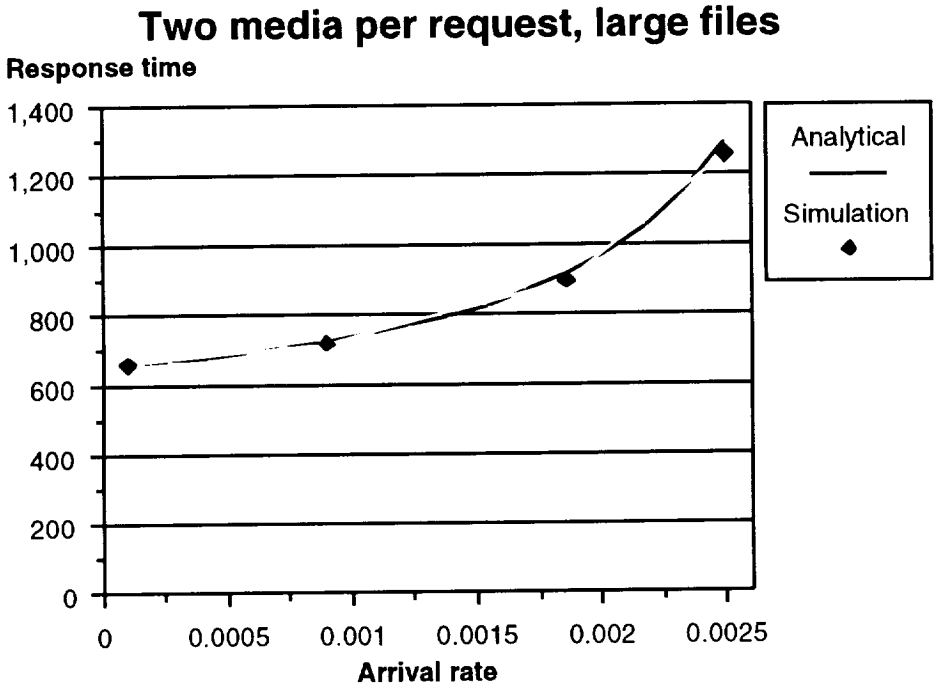


Figure 4

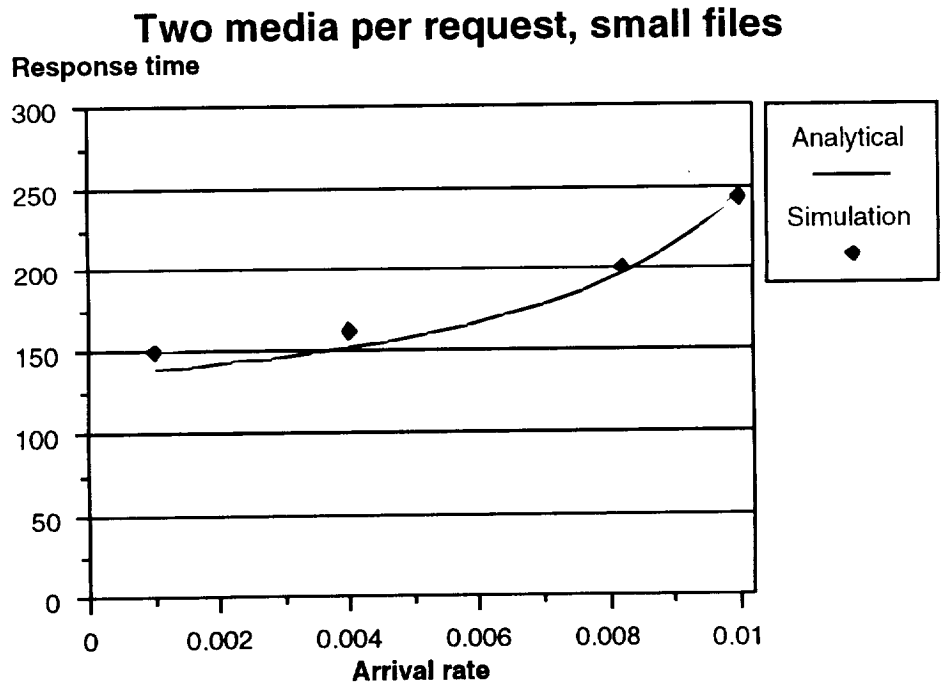


Figure 5

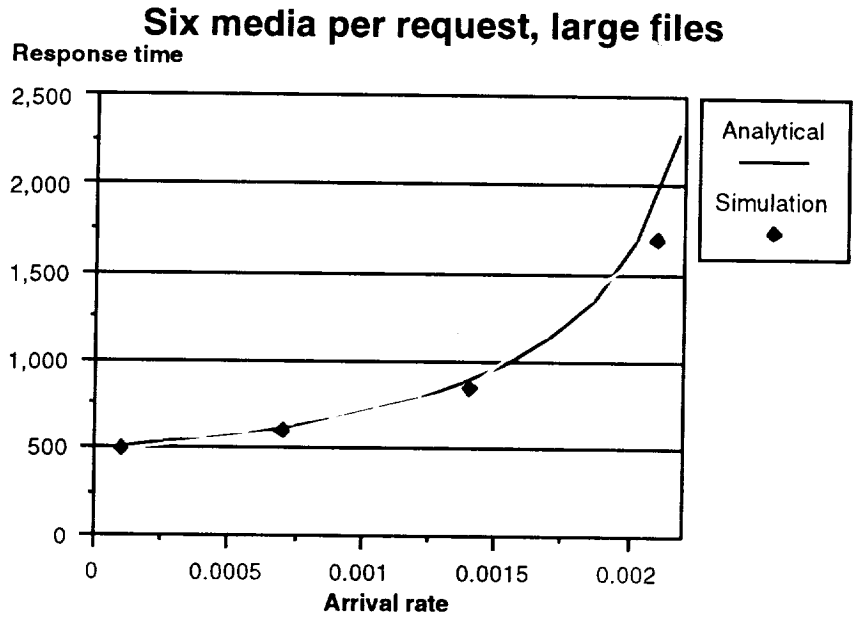


Figure 6

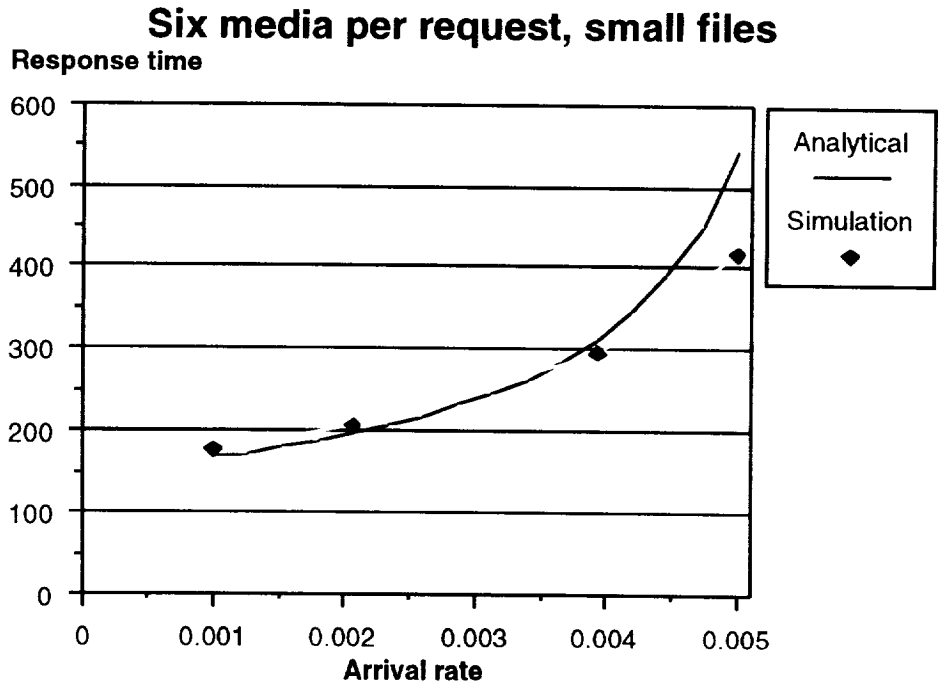


Figure 7

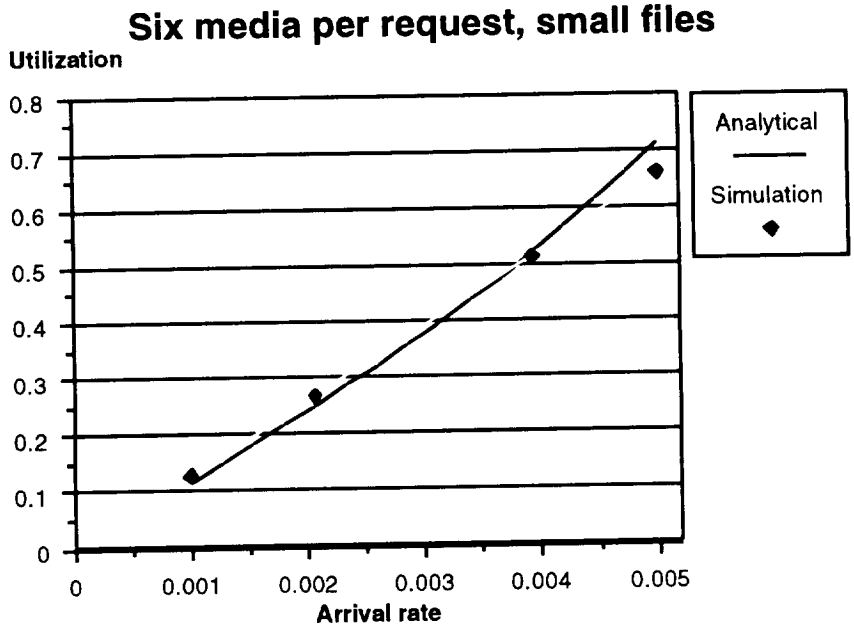


Figure 8

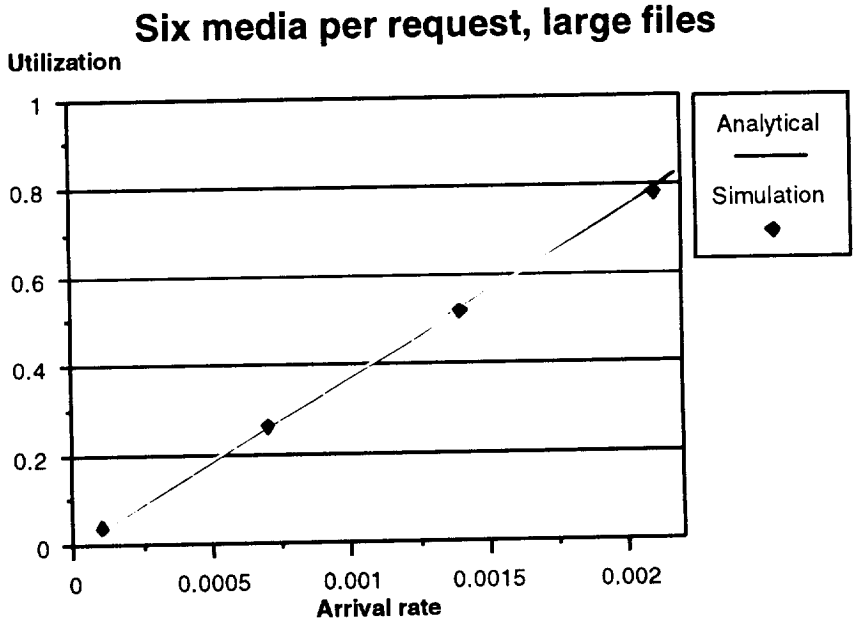


Figure 9

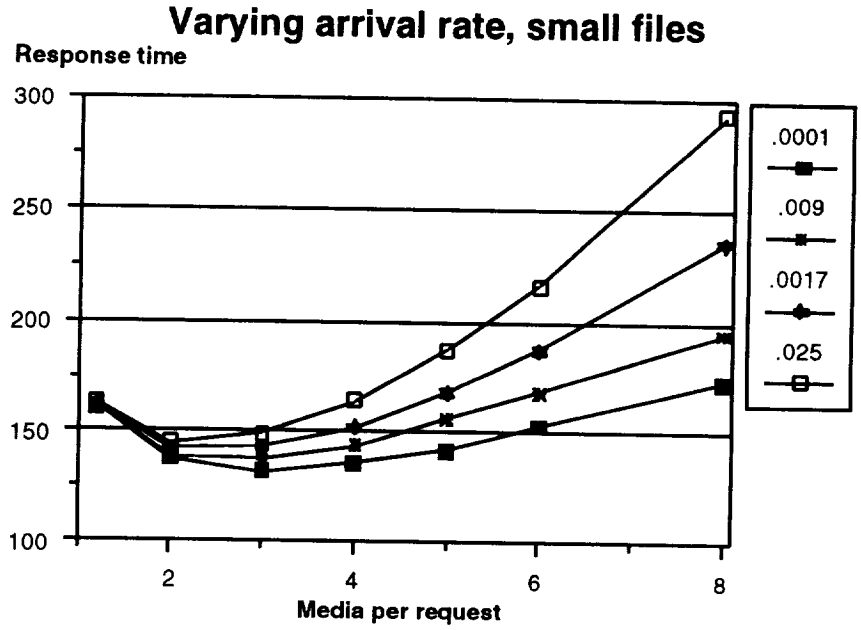


Figure 10

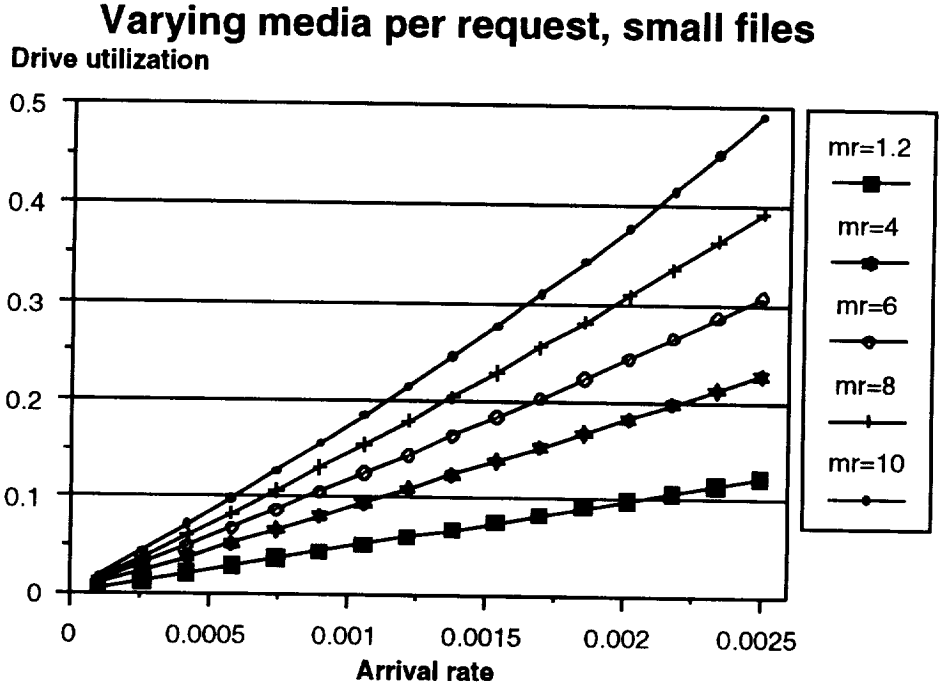


Figure 11

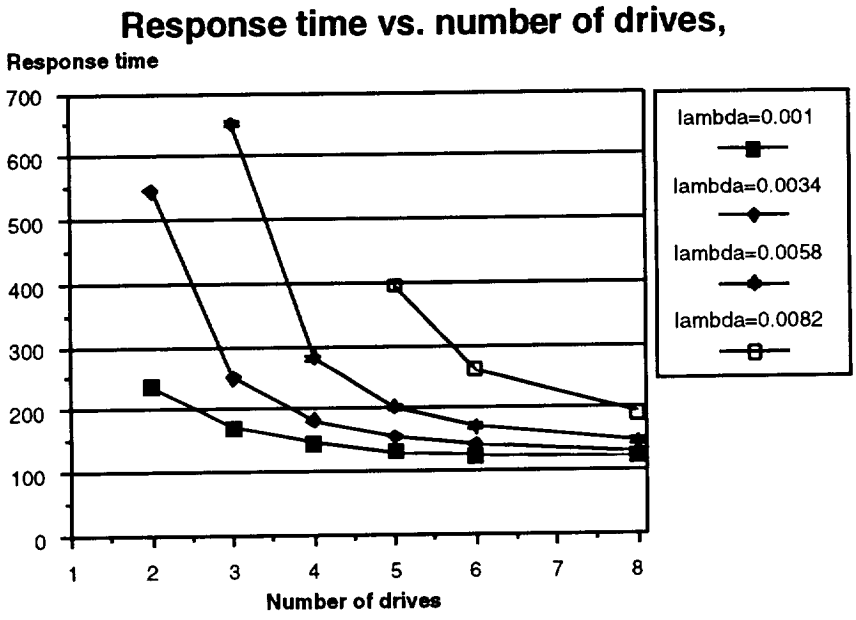


Figure 12

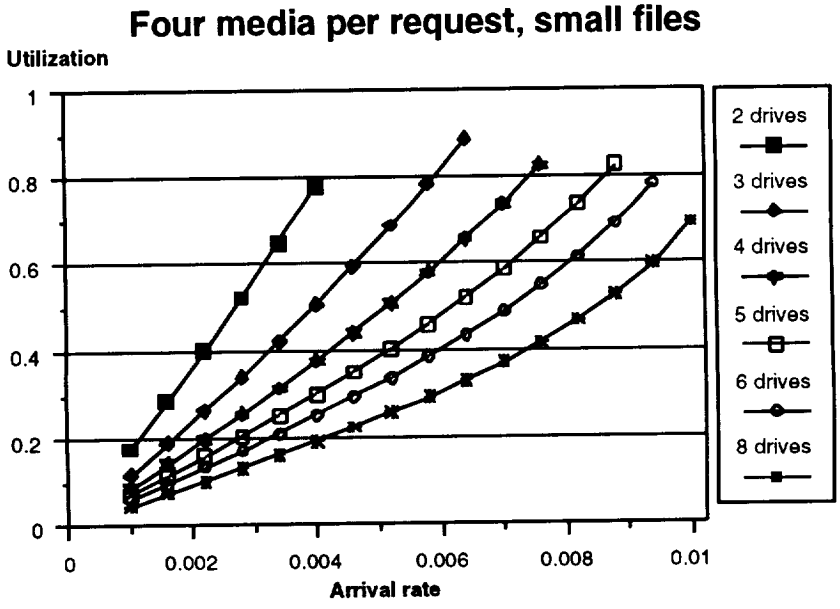


Figure 13

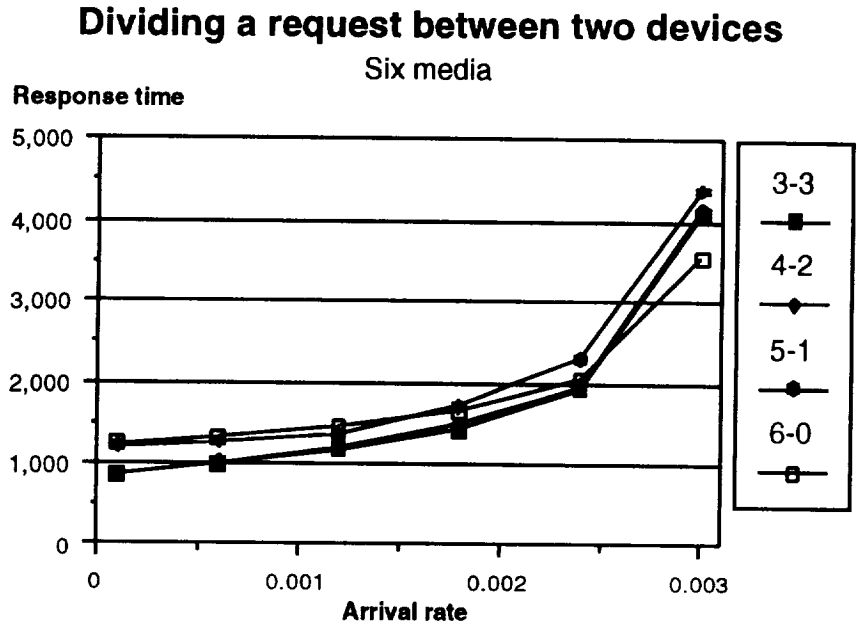


Figure 14

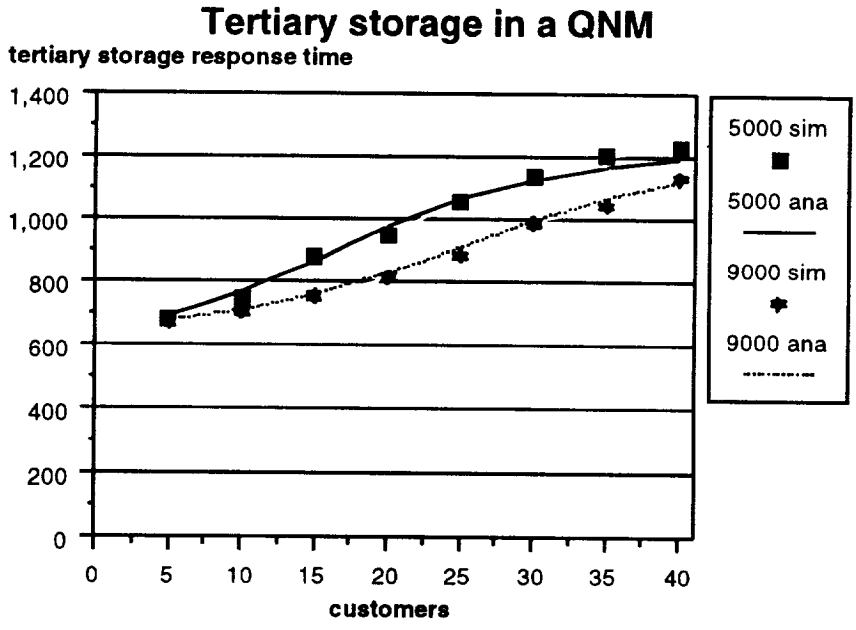


Figure 15