Three Highly Available Data Streaming Techniques and Their Tradeoffs

Sumita Barahmand, Shahram Ghandeharizadeh, Anurag Ojha and Jason Yap
USC Computer Science Department
SAL 102, 941 W 37th Place, Los Angeles, CA 90089-0781, USA
barahman@usc.edu, shahram@usc.edu, aojha@usc.edu, jyap@usc.edu

ABSTRACT

The Recall All Your Senses (RAYS) project envisions a social networking system that empowers its users to store, retrieve, and share data produced by streaming devices. An example device is the popular Apple iPhone that produces continuous media, audio and video clips. This paper focuses on the stream manager of RAYS, RAYS-SM, and its peer-to-peer overlay network. For a request that streams data from a device, RAYS-SM initiates more than one stream in order to minimize loss of data when nodes in its network fail. We present the design of 3 data availability techniques, quantifying their throughput and Mean Time To Data Loss (MTTDL). These two metrics highlight the tradeoff between the resource usage of each technique during normal mode of operation in order to minimize loss of data in the presence of node failures.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Reliability

Keywords

Data availability; System throughput; Markov model

1. INTRODUCTION

Advances in computing, networks, and mass storage devices have enabled wireless devices that stream audio and video clips. Apple's iPhone is one example device. There are numerous comparable devices including the inexpensive wireless camcorders from vendors such as Panasonic, see Figure 1. Users of these devices may share their streams with others in real time and store them for future recall. This is the primary functionality of systems such as RAYS, Qik and Bambuser. A recent study made a compelling case for these systems using "The five Rs": Recollecting, Reminiscing, Retrieving, Reflecting, and Remembering intentions [4].

Figure 1 shows the main components of RAYS. It consists of a user interface, RAYS-UI, that enables users to register streaming devices, stream and record data from them, and share a device with others. The persistent fragment manager, RAYS-PFM, is a scalable storage subsystem that stores data produced by the different streams and enables users to retrieve them. The stream manager, RAYS-SM, initiates streams from different devices on behalf of users, producing data for either RAYS-PFM or RAYS-UI.

To understand the interaction between the different components of RAYS, consider a user who employs RAYS-UI to store the stream produced by a Linksys WVC54GC camera that is monitoring a laboratory. This request is directed to RAYS-PFM which detects if other users are recording the stream by the same device. If this is the case, there is a physical stream from the device to RAYS-PFM. RAYS-PFM represents the new request as a virtual request that maintains its start and end time stamps relative to the physical stream from the target device. Otherwise, RAYS-SM initiates a physical stream from the referenced device to RAYS-PFM.

We have designed RAYS-SM as a decentralized peer-to-peer network of commodity PCs that are distributed in different geographical locations. This overlay network is in the spirit of CAN [3] and Chord [5]. To scale, RAYS-SM allows nodes to join and leave the network.

A node may leave the network abruptly when it encounters power failures or an unscheduled system restart. Such failures are undesirable because the active streams assigned to these nodes are terminated. While the overlay network is able to repair itself in the presence of such failures and re-start the streams of the failed node on other nodes, the data produced by a streaming device is lost during the repair time. To minimize the likelihood of this data loss, we initiate backup streams for each physical stream in the system, with different nodes of RAYS-SM processing these requests. When a node fails disrupting an active stream from a device, its backup on a different node continues to stream from the device, producing data for RAYS-PFM. While the backup streams enhance availability of data, they reduce the number of simultaneous streams (termed throughput) supported by RAYS-SM during normal mode of operation. In this study, we quantify this tradeoff using analytical models.

A key design consideration is how to schedule a request across the available nodes of RAYS-SM. We describe two paradigms. The first generates *sticky* requests that utilize

AVSTP2P'10, October 29, 2010, Firenze, Italy.

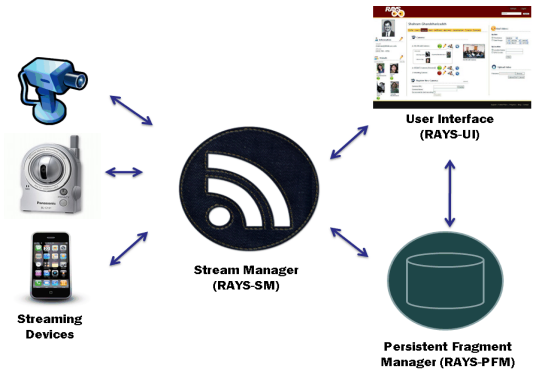


Figure 1: Components of RAYS.

the resources of specific nodes during their life time. *Nomad* requests of the second paradigm visit nodes of RAYS-SM during regular time intervals, distributing their load across all the nodes. We present two variants of nomad requests: The first maintains the dependency between the primary and backup streams of a request while the second avoids this dependency.

The rest of this paper is organized as follows. Section 2 presents the design of sticky requests and its key parameters. Section 3 describes straightforward extensions of this technique to support dependent nomad requests. Subsequently, Section 4 presents independent nomad requests. A comparison of these techniques is presented in Section 5. Section 6 contains brief future research direction.

2. STICKY REQUEST MIRRORING(SRM)

This section presents processing of sticky requests. We start with the basics, establishing the key parameters of request processing during normal mode of operation. Subsequently, we present a data availability technique that initiates α secondary streams on behalf of a single stream. This technique is named Sticky Request Mirroring (SRM). This section concludes by quantifying MTTDL of a single request with SRM.

2.1 Key Parameters

Once RAYS-PFM directs a request to the RAYS-SM overlay network, it is assigned to a node. A sticky request utilizes the resources of its assigned node for the entire duration of its life time (until all users who share the stream click their "stop recording" buttons). A request operates in rounds with a fixed duration, d. During a round, the request initiates a stream from its referenced device, storing data in memory. We assume memory is allocated in fix-sized chunks named blocks. Let P (say 4 Kilobytes) denote the size of a block which is significantly smaller than the amount of data required to store d units of recording from a device. For example if d is set to a value ranging from 5 to 10 Seconds, Panasonic BL-C131 camera produces chunk sizes in the order of hundreds of Kilobytes. Let β denote the size of chunk that corresponds to d units of recording from a device. At the end of a round, a request transmits β bytes to RAYS-PFM for permanent storage. We assume this transmission time, δ, satisfies the following constraint: The amount of data produced by a streaming device during δ is smaller than P. (It is trivial to remove this assumption by introducing yet another variable; however, to simplify discussion and without loss of generality, we decided to make this simplifying assumption). Thus the amount of memory required by a the stream is β+P.

In our experiments with both the Panasonic BL-C131 and Linksys WVC54GC cameras using an HP i7 quad core PC, we observed the physical memory of the PC to be the limiting resource. This means that the number of simultaneous streams captured by a node of RAYS-SM is dictated by the size of that node's available memory. Given N nodes with each node providing m_i amount of memory, the total amount of memory in the system is defined as M=$\sum_{i=0}^{N-1} m_i$. Hence, the theoretical upper-bound on system throughput is: $\frac{\sum_{i=0}^{N-1} m_i}{\beta+P}$.

2.2 Handling Node Failures

To prevent loss of data in the presence of node failures that cause an active stream to disappear from the system, RAYS-SM utilizes other nodes to start redundant streams from a device. When a node failure causes the primary stream to fail, its redundant streams have a copy of the data and provide it to RAYS-PFM for permanent storage. The details are as follows. RAYS-SM represents a request, say R_i, as one primary, denoted $R_{i,p}$, and α backups, $R_{i,b1}$, $R_{i,b2}$, ..., $R_{i,b\alpha}$. Each of these is an independent stream. Thus, for each physical stream in RAYS, there are α+1 streams from the referenced device to α+1 distinct nodes of RAYS-SM. For each request R_i, we use the term *primary* node to denote the node that is processing $R_{i,p}$ and the other α nodes as *secondary*.

There are two key differences between a primary request and its backups: First, during normal mode of operation, the primary transmits β units of data every d unit of time while the backup makes no data transmission. Second, the backup consumes a different amount of memory than the primary node. The exact amount of memory required by a backup stream depends on the delays incurred to recover from a node failure. There are several ways to recover from node failures. Here, we describe one. With sticky requests, RAYS-SM may require each node to maintain the identity of backup and primary requests assigned to its neighbors (in addition to the metadata specific to the overlay network, see [3, 5, 1] for details). When a node fails, its neighbors activate the primary requests assigned to this node first followed by its backup requests. One may utilize a variety of policies to assign these requests to the neighboring nodes as there is no affinity for data. Once a primary request is re-started, it: a) contacts one of its backups to transmit its cached β units of data to RAYS-PFM and promotes this backup to be the primary, b) demotes itself to be a secondary. RAYS-PFM employs a numbering mechanism to detect duplicate β transmissions produced by different streams that constitute a request, storing it once.

This recovery process entails repairing the overlay network (see [3, 5, 1] for details) and re-starting the primary[1] requests of the failed node on its neighbors (or neighbors of neighbors when immediate neighbor(s) have insufficient memory). Let T denote the amount of data pertaining to

[1]Secondary requests of the failed node can be started in the background. Once activated, a secondary notifies its primary of its new location.

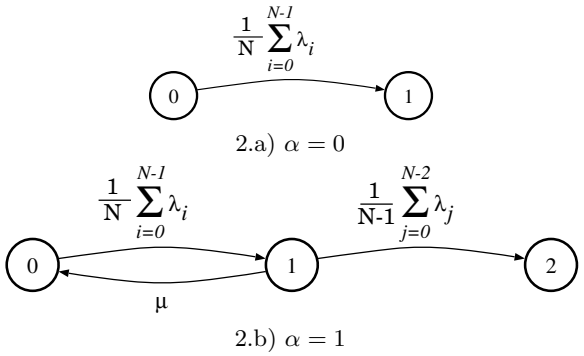


Figure 2: Markov model.

this maximum incurred delay. Thus, each secondary request must maintain (β+T) data in main memory. Hence, the throughput of the system with α backups is defined as:

$$Throughput = \frac{M}{\beta + P + \alpha * (\beta + T)} \quad (1)$$

2.3 Mean Time To Data Loss of a Request

The concept of fault tolerance involves a large number of factors concerning software, hardware, network, environmental failures, etc. The focus of this study is on node failures. A node failure is defined as when a node stops participating in the overlay network of RAYS-SM abruptly. Such a failure might be caused by power-failures, un-scheduled shut-down of a node, etc. The overlay network incurs a Mean Time To Repair (MTTR) to recover from such a failure. The recovery process includes the time to repair the overlay network and restart the streams of the failed node. MTTR defines the duration of data loss for the active streams assigned to the failed node.

In this section, we employ redundant streams to minimize loss of data for a request and use MTTR to establish the Mean Time to Data Loss, MTTDL, of *a* request. We assume the maximum MTTR in order to establish the worst MTTDL.

Markov models offer a useful tool to model the reliability of a system with discrete processes where the next state of the system depends only on the current state. We use a Markov model to determine the MTTDL of the system for RAYS-SM with sticky requests. The probability of transition from one state to another is based on the estimated failure rate of each node. To describe the Markov model for the streaming approach recall that $\alpha + 1$ nodes are streaming data on behalf of a single request, see Section 2.2. In the case where these $\alpha + 1$ nodes of request R_i fail simultaneously before any repairs complete, R_i will no longer be recording and streaming data, resulting in data loss.

We first consider a deployment with no backups ($\alpha = 0$). In this case, failure of the node processing $R_{i,p}$ will result in data loss. Figure 2.a shows the two possible states of the system. Each represents the number of node failures in RAYS-SM. During normal mode of operation, there are no (zero) failures. The sum of failure rates of the nodes (λ_i) quantifies the rate of transitioning from zero failures to one failure in the system, see Figure 2.a. We assume a uniform probability, $\frac{1}{N}$, for $R_{i,p}$ residing on the failed node.

The rate of failure for node i (λ_i) is the inverse of its Mean Time To Failure (MTTF), see [6, 2]. Figure 2 assumes nodes fail independent of one another. Beginning with a given state i the expected time until the first transition into a different state j can be determined as (2).

$$\begin{aligned} \text{E [state } i \text{ to state } j] &= \text{E[time in state } i \text{ per visit]} \\ &\quad + \Sigma_{k \neq i} \text{P(trans state } i \text{ to state } k) \\ &\quad \times \text{E[state } k \text{ to state } j] \end{aligned} \quad (2)$$

$$\text{E [time in state } i \text{ per visit]} = \frac{1}{\Sigma \text{ rates out of state } i} \quad (3)$$

and

$$\text{P(trans state } i \text{ to state } k \text{)} = \frac{\text{rate of trans to state } k}{\Sigma \text{ rates out of state } i} \quad (4)$$

So the MTTDL for this case will be the expected time beginning at state 0 with no failures and ending on transition into state 1 where one failure occurs that can be calculated by solving the Markov model in 2.a using formulas (2,3,4) as below:

$$\begin{aligned} \text{E[state 0 to 1]} &= \text{E[time in state 0 per visit]} \\ &\quad + \text{E[state 1 to 1]}\Sigma\text{P(trans state 0 to 1)} \\ &= \frac{1}{\frac{1}{N}\Sigma_0^{N-1}\lambda_i} + \frac{\frac{1}{N}\Sigma_0^{N-1}\lambda_i}{\frac{1}{N}\Sigma_0^{N-1}\lambda_i} \text{ E[state 1 to 1]} \\ &= \frac{N}{\Sigma_0^{N-1}\lambda_i} \end{aligned} \quad (5)$$

$$\text{E[state 1 to 1]} = 0 \quad (6)$$

The resulting Mean Time To Data Loss :

$$\text{MTTDL} = \frac{N}{\Sigma_0^{N-1}\lambda_i} \quad (7)$$

One may enhance the MTTDL of RAYS-SM using backup streams ($\alpha > 0$). This MTTDL can be quantified by extending the Markov model of Figure 2.a. The evaluation of this model with a large values of α is complex. To simplify the discussion and without loss of generality, we quantify MTTDL with one backup stream ($\alpha = 1$). Figure 2.b shows the Markov model for N independent nodes. The different states signify the number of node failures in the system.Here the MTTDL can be expressed as the expected time beginning at state 0 with no failures and ending on the transition into state 2 with two failures (primary node and the secondary node for the request) which can be computed by solving the Markov model presented in 2.b. Once again, the discussion is focused on one request and the probability of a request being assigned to a node is the same for all nodes. μ is the rate of repair in the overlay network of RAYS-SM and is the inverse of MTTR, $\mu = \frac{1}{MTTR}$.

$$\begin{aligned} \text{E[state 0 to 2]} &= \text{E[time in state 0 per visit]} \\ &\quad + \Sigma\text{P(trans state 0 to 1) E[state 1 to 2]} \\ &= \frac{1}{\frac{1}{2}\Sigma_0^1\lambda_i} + \frac{\frac{1}{2}\Sigma_0^1\lambda_i}{\frac{1}{2}\Sigma_0^1\lambda_i}\text{E(state 1 to 2)} \end{aligned} \quad (8)$$

$$\begin{aligned}\text{E[state 1 to 2]} &= \text{E[time in state 1 per visit]} \\ &\quad +\Sigma\text{P(trans state 1 to 0) E[state 0 to 2]} \\ &\quad +\Sigma\text{P(trans state 1 to 2) E[state 2 to 2]} \\ &= \frac{1}{\mu+\frac{1}{1}\Sigma_0^0\lambda_j}+\frac{\mu}{\mu+\frac{1}{1}\Sigma_0^0\lambda_j}\text{E[state 0 to 2]} \end{aligned} \quad (9)$$

$$\text{E[state 2 to 2]} = 0 \quad (10)$$

$$\text{MTTDL= E[state 0 to 2]} = \frac{\mu+\frac{1}{2}\Sigma_0^1\lambda_i+\frac{1}{1}\Sigma_0^0\lambda_j}{\frac{1}{2}\Sigma_0^1\lambda_i\times\frac{1}{1}\Sigma_0^0\lambda_j} \quad (11)$$

3. DEPENDENT NOMAD REQUESTS(DNR)

Similar to a sticky request, a nomad request operates in rounds of fixed-duration d. A nomad request migrates to a different node of RAYS-SM after each round, distributing the load of the request across the available nodes.

With a backup stream ($\alpha = 1$), there is dependency between the primary and its backup, see Figure 3.b. In each round, the primary migrates to the node containing its backup, initiating a new backup stream on a different node. This migration is a logical continuation of the backup, requiring only a role change[2]. With more than one backup ($\alpha > 1$), the primary may migrate to any one of its backups. It may utilize a variety of policies to select its target backup. We speculate that a policy that selects the backup with the most available memory to maximize the throughput of the system in practice.

With nomad requests and small values of d, it may not be practical to require each node to maintain the identity of requests assigned to its neighbors. This is because the identity of these requests may change frequently (with small d values), causing the exchange of this metadata to impose a high overhead on the overlay network. In this case, primary and backup requests of R_i may monitor each other using periodic heart-beat messages. When a primary ($R_{i,p}$) detects the failure of a backup, it starts a new secondary on a different node. Once a secondary detects the failure of a primary, it (a) promotes either itself or one of the other backups to become the primary, and (b) starts a new backup.

We schedule $R_{i,p}$ and $R_{i,b}$ requests in advance to prepare nodes for recording from the device. This minimizes the impact of process activation time on a stream, ensuring that the backup node begins recording in a timely manner. Without advanced scheduling, activation of recording on the backup might be delayed. If the primary fails during this delay, the backup would not be able to assume the responsibilities of the primary, resulting in data loss.

The throughput of RAYS-SM with dependent nomad requests is similar to the sticky requests. Assuming that the failure of the backup and primary nodes of a request are independent then the MTTDL of a request is similar to that of Section 2.

4. INDEPENDENT NOMAD REQUESTS(INR)

INR breaks the dependency between the primary and backup node(s) of a request by utilizing twice as many backup streams (2α), see Figure 3.c. With INR, a primary migrates to a new node every d units of time while a backup utilizes the resources of its assigned node for $2d$. This enables the primary to migrate to a node independent of the identity of nodes used by its backups. Moreover, this enhances availability of data, see Section 3. Both are at the expense of a reduced system throughput during normal mode of operation. This is because twice as much memory is used by each α backup. The throughput of INR is:

$$Throughput = \frac{M}{\beta + P + 2*\alpha*(\beta+T)} \quad (12)$$

Similar to the discussions of Section 3, requests might be scheduled in advance to minimize the likelihood of race conditions between the primary and its backups.

With $\alpha = 1$, in order for a request to incur data loss, its primary and two, 2α backups must fail. Hence, there must be three, 2α+1 failures in the system to incur data loss, see Figure 4.

$$\begin{aligned}\text{E[state 0 to 3]} &= \text{E[time in state 0 per visit]} \\ &\quad +\Sigma\text{P(trans state 0 to 1)E[state 1 to 3]} \\ &= \frac{1}{\frac{1}{N}\Sigma_0^{N-1}\lambda_i}+\frac{\frac{1}{N}\Sigma_0^{N-1}\lambda_i}{\frac{1}{N}\Sigma_0^{N-1}\lambda_i}\text{E[state 1 to 3]} \end{aligned} \quad (13)$$

$$\begin{aligned}\text{E[state 1 to 3]} &= \text{E[time in state 1 per visit]} \\ &\quad +\Sigma\text{P(trans state 1 to 0)E[state 0 to 3]} \\ &\quad +\Sigma\text{P(trans state 1 to 2)E[state 2 to 3]} \\ &= \frac{1}{\mu+\frac{1}{N-1}\Sigma_0^{N-2}\lambda_j} \\ &\quad +\frac{\mu}{\mu+\frac{1}{N-1}\Sigma_0^{N-2}\lambda_j}\text{E[state 0 to 3]} \\ &\quad +\frac{\frac{1}{N-1}\Sigma_0^{N-2}\lambda_j}{\mu+\frac{1}{N-1}\Sigma_0^{N-2}\lambda_j}\text{E[state 2 to 3]} \end{aligned} \quad (14)$$

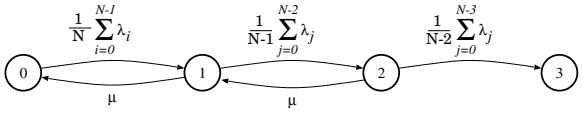


Figure 4: Markov model of INR with $\alpha = 1$.

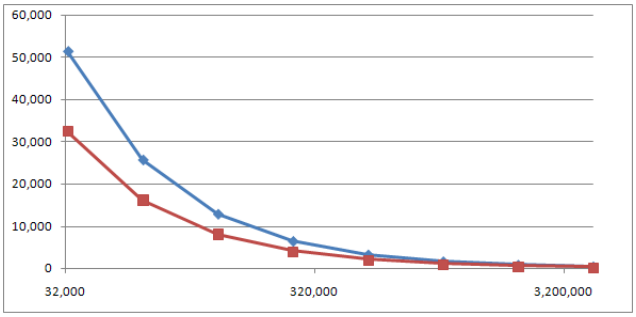
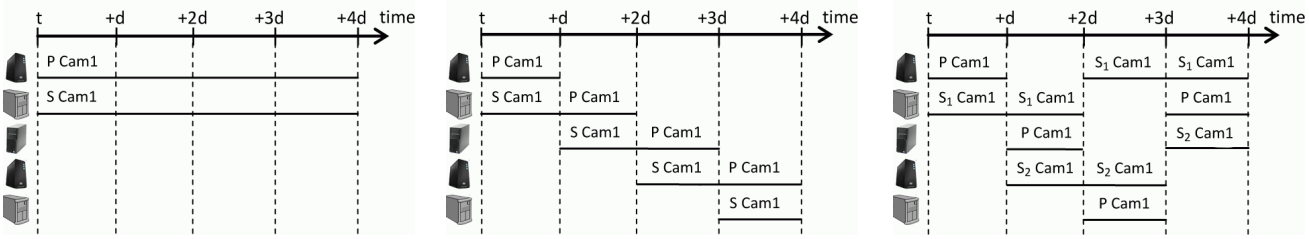


Figure 5: Throughput.

[2]There is no need to stop and restart the active backup stream from the device.



3.a) Sticky Request Mirroring, SRM 3.b) Dependent Nomad Request, DNR 3.c) Independent Nomad Request, INR

Figure 3: Alternative request processing paradigms

$$\begin{aligned}
\text{E[state 2 to 3]} &= \text{E[time in state 2 per visit]} \\
&\quad +\Sigma\text{P(trans state 2 to 1)E[state 1 to 3]} \\
&\quad +\Sigma\text{P(trans state 2 to 3)E[state 3 to 3]} \\
&= \frac{1}{\mu + \frac{1}{N-2}\Sigma_0^{N-3}\lambda_k} \\
&\quad + \frac{\mu}{\mu + \frac{1}{N-2}\Sigma_0^{N-3}\lambda_k}\text{E[state 1 to 3]} \quad (15) \\
\text{E[state 3 to 3]} &= 0 \quad (16)
\end{aligned}$$

Solving the Markov model equations for Figure 4, results in a linear system of three equations with three unknowns.

MTTR	MTTDL	
	SRM/DNR	INR
1 minutes	1,226 Years	34,160,559 Years
5 minutes	245 Years	1,366,842 Years
10 minutes	123 Years	341,841 Years
15 minutes	82 Years	151,988 Years
30 minutes	41 Years	38,041 Years
1 hour	21 Years	9,532 Years
5 hours	4 Years	388 Years
10 hours	2 Years	99 Years
20 hours	1 Years	26 Years

Table 1: MTTDL with SRM, DNR, and INR.

5. A COMPARISON

To compare the alternative data availability techniques, we assumed an overlay network consisting of ten nodes for RAYS-SM. The total available memory of these nodes is 40 Gigabytes. We assumed one of the nodes is a supernode with high availability, MTTF of 100,000 hours. The MTTF of the other nine nodes are: 256, 300, 350, 350, 400, 400, 400, 500 and 500 hours.

Figure 5 shows the throughput of the system with SRM, DNR and INR as a function of the bit rate of the stream produced by a device. The value of d is fixed at 10 seconds in these experiments. The value of P, size of a block, is a function of 500 milliseconds and the bit rate produced by a device (x-axis of Figure 5). The amount of memory required to stream a request (values of β and P, see Section 2.1) increases as a function of the resolution of its referenced camera (higher bit rate requirement), explaining the decrease in throughput of the system. SRM and DNR provide an identical throughput because they require an identical amount of memory. INR requires more memory on behalf of each request, providing a lower throughput during normal mode of operation.

Table 1 shows the MTTDL of the system with $\alpha = 1$ for different MTTR values. With MTTR less than a minute, the MTTDL of SRM and DNR is in the order of thousands of years. INR provides a MTTDL in the order of tens of millions of years because it has 2 backup streams instead of 1, see Section 4. With higher MTTR values, the MTTDL drops for all techniques. With SRM and DNR, the rate of drop in MTTDL is almost the same as the rate of increase in MTTR. This is not the same for INR because it has 2 backups. Note that SRM and DNR would behave the same as INR when configured with $\alpha = 2$. Of course, this would reduces the throughput of these techniques to be identical to that of INR.

We anticipate the MTTR of RAYS-SM to be in the order of seconds (minutes in the worst case scenario). This is because recovery of a failed node involves repairing the overlay network and re-starting the streams residing on the failed node. This work is dominated by the network round trip times (RTT) between the neighbors of the failed node, with each node performing a fraction of the required work in parallel with other nodes.

6. FUTURE RESEARCH DIRECTIONS

Techniques to schedule primary and backup requests intelligently is an immediate future research direction for RAYS-SM. Such a schedule should consider the current load of a node, its available memory, and its historical MTTF characteristics (if available). The throughput analysis of this study is a theoretical upper bound. If requests are not scheduled intelligently, the realized throughputs might fall significantly lower than this bound. This is because the backup requests consume more resources when compared with the primary requests. If they are scheduled naively, the available memory of a node may be exhausted prematurely, forming bottlenecks. This is specially true in the presence of failures.

7. REFERENCES

[1] A. Daskos, S. Ghandeharizadeh, and R. Shahriari. SkiPeR: A Family of Distributed Range Addressing Spaces for Peer-to-Peer Systems. In *15th International Conference on Software Engineering and Data Engineering*, July 2006.

[2] G. A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage.* Ph.D. Dissertation, University of California at Berkeley, Berkeley, California, December 1991.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications,*

Technologies, Architectures, and Protocols for Computer Communications, pages 161–172, Aug. 2001.

[4] A. Sellen and S. Whittaker. Beyond Total Capture: A Constructive Critique of Lifelogging. *Communications of the ACM*, 53(5), May 2010.

[5] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 149–160, San Diego, California, Aug. 2001.

[6] R. Zimmermann and S. Ghandeharizadeh. Highly Available and Heterogeneous Continuous Media Storage Systems. *IEEE Transactions on Multimedia*, 6(6):886–896, 2004.