# A Data Aware Admission Control Technique for Social Live Streams (SOLISs)

*Sumita Barahmand and Shahram Ghandeharizadeh*

Database Laboratory Technical Report 2012-03

Computer Science Department, USC

Los Angeles, California 90089-0781

July 20, 2012

**Abstract**

A SOcial LIve Stream, SOLIS, is a live stream produced by a device whose owner is sharing the stream with her friends, granting each friend to perform time shifted viewing for a pre-specified duration. The system buffers this chase data to facilitate its browsing and display. In the presence of many SOLISs, memory may overflow and prevent display of some chase data. This paper presents a novel data-aware admission control, DA-AdmCtrl, technique that summarizes chase data pro-actively to maximize the number of admissible SOLISs with no memory overflow. It is designed for use with multi-core CPUs and maximizes utility of data whenever the user's level of satisfaction (utility) with different data formats is available. We use analytical models and simulation studies to quantify the tradeoff associated with DA-AdmCtrl.

## A Introduction

Social networking sites such as RAYS [5], Qik [23] or Bambuser [4] enable their members to invite their friends to view a live stream from their devices. A streaming device might be a smartphone such as an iPhone or an inexpensive wireless consumer electronic device by a vendor such as Linksys or Panasonic. With RAYS, the owner of a Device $i$ may specify a chase duration ($\Delta_{i,j}$) for an Invitee $j$. This allows Invitee $j$ to time shift view [12, 16, 17] data for the specified duration $\Delta_{i,j}$. The term SOcial Live Stream, *SOLIS*, refers to device produced streams with this characteristic [6], see Figure 1.

The motivation for using streaming devices with a social networking web site is two folds. First, to share a SOLIS with a large number of invitees. Second, to support functionality such as chase display and recording of streams for future recall and sharing. With the first, a device typically supports a handful of simultaneous viewers due to either limited network bandwidth, processing capability, or both. For example,

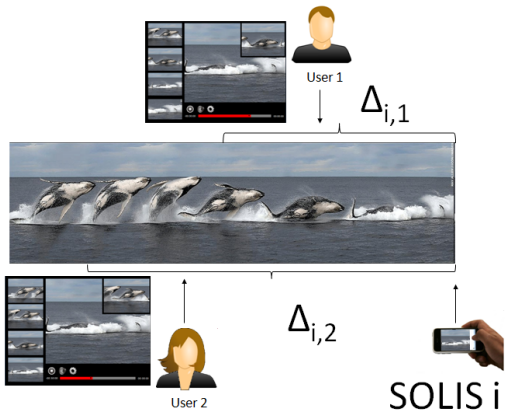



Figure 1: A SOLIS with two chase displays.

the Panasonic BL-C131 camera configured to use its wired network connection supports a maximum of 9 to 12 simultaneous displays. When used with a social networking web site, a stream might be shared among thousands of viewers.

With the second, the constantly evolving chase data corresponding to the past $\Delta_{i,j}$ time units must be buffered on behalf of Invitee $j$. The infrastructure of a social networking site provides this memory[1]. Moreover, it may provide sophisticated indexing techniques (using tapestries [7]) to enable the invitee to retrieve the relevant portions of chase data. With a single SOLIS $i$ and many invitees with different $\Delta_{i,j}$ values, RAYS maintains data pertaining to the maximum $\Delta_{i,j}$ value. All viewers share this copy of data and the software enforces the value of $\Delta_{i,j}$ specified (by the owner of the device) for Invitee $j$. This means invitees of a SOLIS may join an in progress SOLIS late without requiring additional memory. However, a new SOLIS request $q$ requires memory equal to the maximum $\Delta_{q,r}$ specified by the owner of Device $q$ for invitee $r$.

A key deployment question is the configuration of the server and its memory size. One may over provision for the maximum number of SOLISs and their $\Delta$ values. This is a reasonable approach given today's inexpensive price of memory. However, if either the anticipated number of SOLISs is unknown or the size of the device formatted data changes dynamically, then a deployment may run out of memory and evict chase data. Every time an invitee references this evicted data, the system may either ignore the request and/or report the data is missing, reducing the quality of service.

One approach to this limitation is to require the system administrator to detect memory overflows and extend the system with additional memory. This detective approach has its own limitations. First, from the first time an overflow is detected until additional memory is obtained, SOLIS requests may overflow memory repeatedly. Second, configuring a server with additional memory may require some down time.

[1]An alternative is to buffer this data on the client device. A limitation of this approach is that the client device may have insufficient physical memory to buffer $\Delta$ time units of data.

An alternative approach is to employ a Data Aware Admission Control, DA-AdmCtrl, technique. When a new SOLIS request arrives to the system and there is insufficient memory to support its chase data, DA-AdmCtrl starts to summarize past and incoming data pro-actively in order to admit the new request. A *Summarization Technique, ST,* may either reduce the resolution (quality) of data in order to reduce its memory requirement or convert device produced data into a format that can be used by another ST to reduce its size, see discussions of Section E. DA-AdmCtrl admits a new SOLIS request as long as there is sufficient memory to accomodate its chase data. Otherwise, it rejects the SOLIS request. The system may allow this request to proceed as a live stream with no chase display.

With multi-core CPUs, DA-AdmCtrl employs several ST processes to summarize different data chunks simultaneously where each data chunk corresponds to a portion of chase display of one or more SOLISs. There may exist dependencies between the STs such that the summary format produced by one ST is the input to another ST. For example, Figure 3.a shows one inter-dependency named Deep-Desc. It consists of four different data formats: $F_1$ produced by a device with average size of 1000 KB per unit of time, and three summarized formats $F_2$, $F_3$, and $F_4$ with their sizes shown in each node (number on an edge denotes the average service time of ST to process its input data format to produce its output data format). DA-AdmCtrl is flexible enough to support arbitrarily complex inter-dependencies between STs as long as they are trees.

The primary contribution of this paper is a data aware admission control technique that admits SOLIS requests with the objective to maximize utility of data with no memory overflow. Utility quantifies the user's level of satisfaction with a data format (produced by either the device or an ST). When the utility for all data formats is identical (chase viewer does not discriminate between the different data formats), the admission control selects those data format(s) that prevent memory overflow. In addition to considering the available memory at one instance in time, the admission control ensures the summarization rate for the chase data is either equal to or higher than the rate of data arrival by the currently active SOLISs, preventing memory over-flow. This analysis considers the number of cores, the service time of STs, and their inter-dependencies. The following example illustrates these concepts.

**Example 1**: Consider three SOLISs, each with $\Delta = 5$ time units. Each SOLIS produces 1000 KB data in format $F_1$ in each time unit, $C_{F_1}$=1000 KB (see Table 1). The system is configured with one ST that summarizes the incoming high resolution data to a low resolution format $F_2$ that is ten times smaller, $C_{F_2}$=100 KB. Assume the summarization process requires 2 time units. At first glance, the amount of required memory for these 3 SOLISs might appear to be 1500 KB, $3 \times \Delta \times C_{F_2}$. However, as shown in Figure 2, assuming a system with more than six cores, the total amount of required memory during steady state is 7200 KB. At any instance of time, say 7th time unit, the occupied memory must accommodate three different SOLISs producing $F_1$ data ($3 \times 1000$ KB), $F_1$ data produced at time 6 that is being summarized actively ($3 \times 1000$ KB), $F_2$ data that is being produced actively by the summarization technique ($3 \times 100$ KB), and $F_2$ data pertaining to summary data corresponding to Time 5, 4, and 3 ($3 \times 3 \times 100$ KB). The

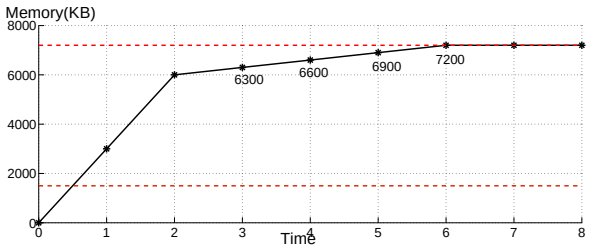



Figure 2: Memory usage for three SOLISs of Example 1.

later must be maintained in memory because $\Delta$=5.

The numbers in this simple example scale. The amount of required memory with 3000 SOLISs is approximately 7 GB assuming a system with 6000 cores. If the system is configured with less memory than what is required, then the admission control might be forced to reject some of the arriving SOLIS requests even though the system is configured with many cores to summarize data. At least 6000 cores are required because the time to summarize $F_1$ formatted data is twice the inter-arrival time for data. Fewer than 6000 cores requires more memory than 7 GB, see discussions of Equation 1. □

The rest of this paper is organized as follows. Section B presents related work. Details of DA-AdmCtrl are provided in Section C. We quantify the tradeoffs associated with DA-AdmCtrl in Section D. Section E describes flexibility of DA-AdmCtrl and its use in two different contexts. Section F offers brief conclusions and future research directions.

## B Related Work

Admission control techniques prevent a newly arrived request from disrupting currently active requests and their observed quality of service [11, 25]. They do so by monitoring either the amount of available memory, disk [26, 18, 28, 1] and network [2, 8] bandwidth, or a combination of these resources. For example, admission control techniques developed for video-on-demand servers consider disk bandwidth in combination with the available memory to control how many simultaneous requests may stream data from the disk without overflowing memory [13, 10, 22, 9, 15, 20]. Similarly, multi-layer encoding techniques are employed to control the bandwidth requirements of a stream in the presence of bursty background load that exhausts the available network bandwidth [21, 24, 27]. These studies strive to deliver continuous media in a timely manner to prevent a client's display from starving for data. Our proposed technique is different because it focuses on live streaming data and how to summarize this data proactively in order to admit requests without overflowing memory. Note that once data is lost (due to memory overflows), the original cannot be re-constructed. Similarly, once a lossy summarization technique reduces the resolution of data to reduce the memory requirement of a SOLIS, the original data is lost permanently.

Several studies focused on recording of continuous media produced by cameras used by news organizations [19, 3, 14]. Their proposed techniques write data to magnetic disk drives without overflowing memory of a server as a temporary staging area. These studies highlight the limited bandwidth of magnetic disk drives, restricting the number of admitted requests severely. While the bandwidth of disks continues to be a limiting factor today, the cost of memory has dropped significantly to deliver inexpensive servers with tens of Gigabytes of memory[2]. These trends are the primary motivation for our proposed admission control technique. By using memory only, we can support a much larger number of concurrent requests than [19, 3]. Our admission control technique is novel for two reasons. First, it quantifies memory requirements of alternative summarization techniques with multi-core CPUs. Second, it is flexible enough to consider arbitrarily complex inter-dependencies between different summarization techniques as long as they are trees.

We introduced the concept of SOLIS, its user interface, and a buffer replacement technique that summarizes (instead of evicting) data when memory becomes fully occupied in [6]. The replacement technique cannot control the number of SOLISs and, once the number of SOLISs exceeds a threshold, is forced to evict a significant amount of chase data from memory. The proposed admission control technique address this limitation in two ways. First, it summarizes data pro-actively. Second, it does not admit a SOLIS request if it results in memory overflow.

# C Admission Control

Admission control is invoked every time a new SOLIS request arrives to the system. It can be configured to either prevent memory overflow or maximize utility of data while preventing memory overflow. Memory overflow occurs when data pertaining to chase display portion of active SOLISs exceeds the available memory, forcing the system to drop chase data that might be referenced by a viewer. It is undesirable because whenever the viewer references this data, the viewer's display becomes blank as the referenced data is no longer available. Data *utility* refers to the user's satisfaction with the quality of displayed data. An ST may reduce the size of incoming data by producing a lower resolution (quality) format. This data format might be perceived as undesirable by a user who may stop viewing it. The user's satisfaction is application specific and we assume a system administrator quantifies it by assigning a utility to each data format. A higher utility value reflects a better quality of service (a more satisfied user). We assume the device produced data has the highest utility.

Inputs to DA-AdmCtrl include the amount of available memory, the alternative STs supported by the server and their inter-dependencies, and the characteristics of each ST such as its compression factor, anticipated service time, and utility of its output data. Given $F$ data formats, each format is represented as a node

[2]At the time of this writing, one may purchase a ZT Desktop PC with 16 GB of memory and a 2 TB disk drive for less than $900.

and an edge represents an ST that consumes one data format to produce another. Thus, an inter-dependency is a directed acyclic graph consisting of nodes and edges annotated with meta-data. Figure 3 shows four inter-dependencies describing the average size of a data format (number in the node) and service time of an ST (value assigned to an edge). While the inter-dependencies of Figure 3 are simple, our proposed admission control supports arbitrarily complex inter-dependencies as long as they are trees, i.e., directed acyclic graphs with one root, $F_1$.

DA-AdmCtrl must either admit or reject SOLIS requests such that it meets its configured objective using its available resources. The server starts buffering the chase data for a SOLIS as soon as it is admitted by the DA-AdmCtrl. The chase portion of a rejected SOLIS is not buffered and the request may proceed as a livestream. The decision to admit requests is trivial when the available memory exceeds the chase display requirements of the active SOLISs and their device produced data. The problem becomes interesting once the number of active SOLISs is such that a newly arrived SOLIS request exhausts the available memory. Now, DA-AdmCtrl must decide which STs to employ pro-actively to accommodate the incoming SOLIS. In essence, DA-AdmCtrl is solving a two dimensional bin packing problem. On one dimension, it strives to meet the memory requirements of the active SOLISs and STs used to compact them. On a second dimension, it solves how to schedule STs using the available CPU cores.

We solve the problem using a two step heuristic. The first step identifies how many SOLISs are supported given a *combination* of ($\omega$) data formats. Each combination results in *mixes* of data for the identified data formats. A mix defines the number of SOLISs in each format within the $\omega$ possible formats, $\omega \leq F$. Each mix supports a fixed number of SOLISs and has a utility. Different mixes are stored in a sorted array, identifying the maximum number of SOLISs ($\mathcal{N}$) supported by system resources (memory and cores).

In the second step, DA-AdmCtrl uses this array to decide whether to admit or reject a new request. At any instance in time, DA-AdmCtrl is aware of the number of active SOLISs. If the maximum number ($\mathcal{N}$) is reached and a new SOLIS request arrives, DA-AdmCtrl rejects this new request to prevent memory overflow for the existing SOLISs. Otherwise, it computes the new number of SOLISs in the system and looks up the table to identify the mix that supports this new number. It proceeds to schedule CPU resources to realize the identified mix. Similarly, when a SOLIS leaves the system, DA-AdmCtrl decrements the number of SOLISs in the system and looks up the table to identify the most suitable mix. It may stop scheduled summarizations that are either not necessary or reduce the utility of data unnecessarily.

Below, we elaborate on these two steps in turn.

### C.1 Step 1: Mix of STs

The first step of the heuristic computes different mixes of $\omega$ data formats. Each mix supports a fixed number of SOLISs and may offer a different utility. To simplify discussion and without loss of generality, we assume each SOLIS has a fixed chase display duration $\Delta$. Section C.1.1 extends this discussion to SOLISs

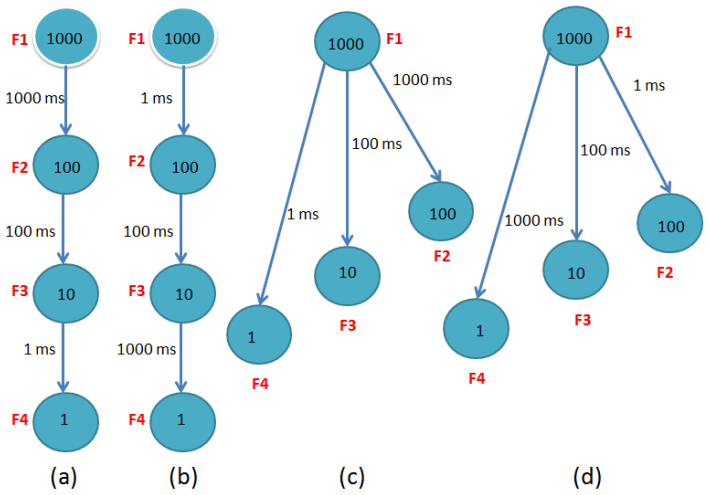



Figure 3: Example inter-dependencies: (a) Deep-Desc, (b)Deep-Asc, (c)Shallow-Desc, (d) Shallow-Asc. Numbers in each node denote the size of a data format per unit of time. Numbers next to an edge denote the service time of a summarization technique, chosen to highlight the tradeoffs associated with DA-AdmCtrl (see discussions of Figure 6).

with varying $\Delta$ values.

The value of $\omega$ dictates the number of data formats that DA-AdmCtrl may employ at an instance in time. It is an integer greater than or equal to zero and less than or equal to $F$. If $\omega = 0$ then DA-AdmCtrl must admit requests based on device produced data format and may not use an ST. If $\omega = 1$ then DA-AdmCtrl may admit requests by using one of the $F$ data formats. If $\omega = 2$ then DA-AdmCtrl may admit requests by using a combination of either one or two (out of the $F$ data) formats. And when $\omega = F$, DA-AdmCtrl may utilize all available data formats simultaneously.

The number of STs used to generate a combination of $\omega$ data formats is dependent on the inter-dependency

| Term | Definition |
|---|---|
| $\Delta$ | Chase display duration. |
| $D$ | Number of cores, i.e., simultaneous summarizations. |
| $D_{f_i}$ | Number of cores to summarize data format $f_i$. |
| $M$ | Total available memory. |
| $F$ | Number of data formats. |
| $F_1$ | Device produced data format. |
| $F_i$ | An available data format $i$. |
| ST | A summarization technique. |
| $C_{F_i}$ | Size of data format $F_i$ in one time unit. |
| $T_{F_i}$ | Time required to generate $F_i$, $i > 1$. |
| $\omega$ | Number of data formats used simultaneously, $0 \leq \omega \leq F$. |
| $\mathcal{N}$ | Maximum number of admissible SOLISs. |

Table 1: List of terms and their definitions.

between the data formats. To illustrate, consider the Deep-Desc inter-dependency of Figure 3.a. When $\omega = 1$, DA-AdmCtrl may utilize any one of these data formats to admit requests. To produce the most compact representation $F_4$ with average size of 1 KB, DA-AdmCtrl must utilize 3 STs and apply them in succession. With the Shallow-Desc inter-dependency (see Figure 3.c), DA-AdmCtrl may use 1 ST to produce the most compact representation $F_4$.

A *combination* is a set consisting of $\omega$ data formats: $f_1$, $f_2$, ..., $f_\omega$, $f_i \in \{F_1, F_2, ..., F_F\}$, $f_i \neq f_j$. With $\omega = 0$, the set only consists of $F_1$ and DA-AdmCtrl must use $F_1$. With $\omega > 0$, the number of possible combinations is $\sum_{i=1}^{\omega} \binom{F}{i}$. Larger values of $\omega$ increase the number of combinations.

When $\omega = 0$, the number of admissible SOLISs is $\frac{M}{C_{F_1} \times \Delta}$ where $M$ is the size of available memory, $C_{F_1}$ is the size of device produced data in one time unit, and $\Delta$ is the duration of chase display in the same time unit granularity. When $\omega = 1$, there are $F$ possible combinations, each set with one unique data format $f_i$. Each combination has one mix consisting of one $s$ value, the number of SOLISs supported by its corresponding data format. The exact value of $s$ depends on the available memory, the summarization service time $T_{f_i}$, and the number of cores, see Equation 1.

When $\omega > 1$, each combination supports a set of admissible requests: $\{s_1, s_2, ..., s_\omega\}$ where $s_i$ is the number of SOLISs supported by its corresponding data format. Each set is named a *mix* and supports $\sum_{i=1}^{\omega} s_i$ admissible SOLISs. For example, with one of the inter-dependencies of Figure 3 and $\omega = 2$, a combination might be $F_2$ and $F_3$. Assuming a system with 10,000 KB of memory configured with the granularity of time set to 1 second and $\Delta$=1 second, a mix of this combination might be $\{91, 90\}$, supporting a total of 181 SOLISs. This mix requires DA-AdmCtrl to commit approximately 90% of the memory to be occupied by data in format $F_2$ while the remaining 10% is occupied by data in format $F_3$. The two extreme mixes for this combination are (1) $\{100,0\}$: 100 SOLISs in format $F_2$ and 0 SOLISs in $F_3$, and (2) $\{0, 1000\}$: 0 SOLISs in format $F_2$ and 1000 SOLISs in $F_3$. Thus, the mixes produced for one $\omega$ value (say $j$) are a superset of those with a lower $\omega$ value (less than $j$). In this example, the mixes produced for $\omega$=2 are a superset of those produced with $\omega$=1.

A mix may not be feasible given the available memory, the service time of different STs, and the number of CPU cores. An ST allocates memory for its produced data prior to starting its summarization. Thus, invocation of an ST increases the required memory during its service time, see Example 1 in Section A. The average increase by an ST is the average output size of that ST. To quantify the amount of required memory, we break time into fix sized intervals, say 100 milliseconds. Moreover, we represent $\Delta$ and service time $T_{f_i}$

of each ST in these units. The amount of required memory is:

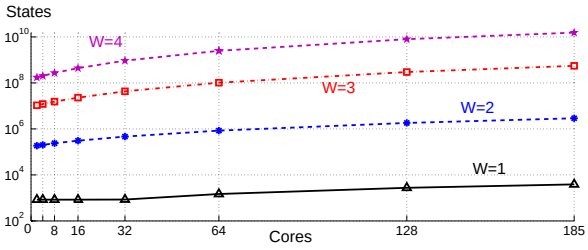
$$
\begin{aligned}
M_{req} \;=\; & (N \times C_{F_1}) + \\
& \sum_{i=1}^{\omega} \sum_{0}^{\Delta - T_{f_i}} (\frac{D_{f_i}}{T_{f_i}} \times C_{F_{f_i}}) + \\
& (\sum_{0}^{\Delta-2} N - \sum_{i=1}^{\omega} \sum_{0}^{\Delta - T_{f_i}} \frac{D_{f_i}}{T_{f_i}}) \times C_{F_1}
\end{aligned} \tag{1}
$$

This equation consists of 3 terms and we explain them in turn. During each time unit, a device produces a fixed amount of data, $C_{F_1}$. With $N$ SOLISs, the amount of required memory is $N \times C_{F_1}$. During one time unit, the number of in-progress summarization for format $f_i$ is estimated to be $N - \lceil \frac{D_{f_i}}{T_{f_i}} \rceil$ where $D_{f_i}$ is the number of employed cores (independent summarization processes) to produce summary format $f_i$. For the remaining $\Delta - T_{f_i}$ time units, the summarized data must be maintained in memory. If $D > N$ then $D$ is reset to $N$. This prevents the last term from producing negative values.

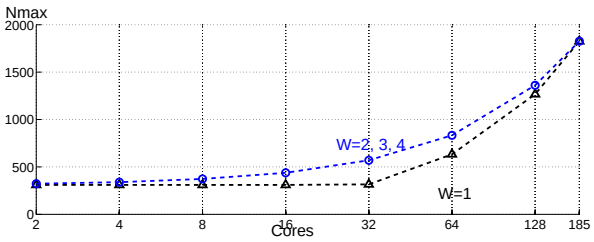
A mix is valid as long as (1) its required memory is less than or equal to the available memory, $M_{req} \leq M$, and (2) its required number of cores is less than or equal to the available cores, $\sum_{i=1}^{\omega} D_{f_i} \leq D$. Step 1 discards invalid mixes.

Given a combination, we enumerate its mixes starting with the least compact representation among all the formats within a combination. This in combination with the size of memory limits the choice of values for the compact formats, reducing the number of enumerated mixes. The enumeration process terminates either once a mix requires memory in excess of $M$ or all possible mixes have been enumerated. As an example, recall the earlier example with a combination consisting of $F_2$ and $F_3$ with a 10,000 KB memory size, we enumerate the possible combinations starting with the highest number of SOLISs supported by $F_2$. This would be $\{100,0\}$. Next, we decrease the number of SOLISs for $F_2$ and compute how many are supported by $F_3$. This produces $\{99,10\}$ and discards values 1 to 9 for $F_3$. This process repeats until either the mix $\{0,1000\}$ is encountered or an enumerated mix has a memory requirement in excess of 10,000 KB. Once the memory requirement of a mix, say $\{99,10\}$, exceeds 10,000 KB then the available memory in combination with CPU resources cannot support more than 10 summarizations. Hence, there is no point in considering 20 summarizations. These early pruning of mixes expedites Step 1, see discussions of Figure 4 in the following paragraphs.

Valid mixes are inserted as key-value pairs in a hash table. The hash table maintains the mix with the highest utility for $n$ SOLISs. The hash table key is the unique value $n$ that is a candidate value for $N$, the maximum number of SOLISs. Its value is a data structure containing a representation of a mix and its utility rating (if any). Before each key-value insert for a key $n$, Step 1 performs a hash lookup for $n$. If no entry exists then in proceeds to insert the new key-value pair in the hash table. Otherwise, the new mix overwrites




4.a) Number of visited states.




4.b) Maximum admissible SOLISs.

Figure 4: Impact of $\omega$ on number of visited states and $N$ with Shallow-Asc and $\Delta$=2 minutes.

the existing one only if its utility is higher.

The space complexity of this algorithm is O($N$) due to its use of a simple hash table to store key-value pairs that maintain different mixes for each unique value of $N$. DA-AdmCtrl uses this table to either admit or reject SOLIS requests, see discussions of Step 2 in Section C.2.

An upper bound on the complexity of this step is O($F^{\omega} \times N^{\omega+1}$). This is an upper bound because it ignores the inter-dependency between the different data formats, the number of cores, and our pruning heuristic. Figure 4.a shows the number of visited states with different $\omega$ values when pruning heuristics are applied as a function of the number of CPU cores (1 GB of memory and the Deep-Asc inter-dependency graph). The y-axis denotes the number of visited states and is log-scale. The number of visited states increases as a function of the number of cores because additional cores support a larger mix of requests. 185 cores are required to summarize data to the lowest format possible to realize the maximum number of admitted SOLISs, 1800. With an exhaustive search, this step would have visited 14 thousand, 76 million, 182 billion, 163 trillion states with $\omega$=1, 2, 3, and 4, respectively. Our technique visits significantly fewer states.

Figure 4.b shows the maximum number of admissible SOLISs with different number of cores and $\omega$ values. While increasing the value of $\omega$ from 1 to 2 enhances the number of admitted SOLISs, values of $\omega$ beyond 2 provided no improvements. This means incurring the complexity overhead of $\omega$ set to 3 and 4 is not beneficial when the objective is to maximize $N$. Note that when the number of SOLISs is below $N$, with $\omega$ values higher than 2, the mixes in the hash table may consist of more than 2 formats, see discussions of Figure 5 in Section D.

### C.1.1 Varying $\Delta$ values

A new SOLIS request may specify the value of its $\Delta$ upon its arrival and this value may not be available in advance. Moreover, different SOLIS requests may use a different $\Delta$ value. Typically, the system has sufficient memory to accommodate chase data of a few SOLISs in their device produced data format, $F_1$. This enables DA-AdmCtrl to admit the first few arriving SOLISs without computing the mix of format

combinations. As free memory falls below a certain threshold (say 10% of $M$) then DA-AdmCtrl computes the average $\Delta$ value ($\overline{\Delta}$) used by the active SOLISs to compute the table of mixes. This establishes both the table and its assumed $\Delta$, i.e., $\Delta_T$. As SOLISs arrive (leave) the system, $\overline{\Delta}$ may become different than $\Delta_T$. $\overline{\Delta}$ values smaller than $\Delta_T$ are undesirable because they force DA-AdmCtrl to use STs more aggressively than necessary, reducing utility of data. $\overline{\Delta}$ values greater than $\Delta_T$ are less desirable because they mislead DA-AdmCtrl to utilize STs in a manner that may exhaust available memory.

To prevent data loss, every time $\overline{\Delta}$ exceeds $\Delta_T$, we re-compute the table. To avoid frequent computation of the table, we use an inflation value to establish the new value of $\Delta_T$, $\Delta_T$=$\alpha \times \overline{\Delta}$, used to compute the table. The value of $\alpha$ may inflate $\Delta_T$ by 10%, $\alpha$=1.1. This means $\overline{\Delta}$ must increase by 10% in order to trigger computation of a new table.

To prevent aggressive use of STs and their impact on data utility, we re-compute the table every time the value of $\overline{\Delta}$ differs from $\Delta_T$ significantly. For example, we re-compute the table when $\frac{\Delta_T}{\overline{\Delta}}$ is more than a factor of two.

With the inter-arrival time of SOLISs anticipated to be in the order of minutes (and hours), $\overline{\Delta}$ should not change that often. Thus, the re-computation of table of mixes is not expected to be either frequent or taxing of CPU resources.

### C.2 Step 2: Combination Lookup

The hash table of Step 1 is used by DA-AdmCtrl to either admit or reject SOLIS requests as follows. DA-AdmCtrl maintains the number of active SOLIS requests, $n$. When a new SOLIS request arrives, DA-AdmCtrl increments $n$ by 1 and uses this value to probe the hash table for a mix (which provides the highest utility). If the hash table produces no output then the new request is rejected because $n$ is the maximum number of supported SOLISs, $n$=$N$, and $n$ is restored to its original value. Otherwise, DA-AdmCtrl proceeds to admit the request and realizes the new mix by scheduling the latest arriving data chunks for summarization first. These chunks reside in the memory for the longest amount of time and Equation 1 assumes they are summarized first.

## D Evaluation

We used a simulation study to characterize the behavior of DA-AdmCtrl with different amount of memory, CPU cores, summarization techniques and their inter-dependencies. This section reports on one experimental setting and our obtained results. Table 2 shows the assumed data and system settings. We focus on inter-dependencies of Figure 3 due to their simplicity. The numbers shown in each node of an inter-dependency are at the granularity of blocks where the size of a block is 250 KB. The first three reported experiments assume the utility of data is the same for all data formats, characterizing DA-AdmCtrl and

| Data | |
|---|---|
| SOLIS Bit Rate | 2 Mbps |
| Chase display duration, $\Delta$ | 2 minutes |
| System | |
| Cache Memory Size, $M$ | 1 GB |
| Number of Data formats, $F$ | 4 |
| Granularity of time unit | 1 second |

Table 2: Simulation parameters for the experiments.

| $D$ (#of Cores) | Simple | DA-AdmCtrl |
|---|---|---|
| 32 | 33 | 573 |
| 256 | 256 | 1864 |
| 1024 | 1024 | 2259 |
| 2048 | 2048 | 2787 |
| 3574 | 3574 | 3574 |

Table 3: Number of admitted SOLISs with Simple and DA-AdmCtrl ($\omega$=2) using Shallow-Asc inter-dependency of Figure 3.d, $\Delta$=2 minutes.

its behavior with different settings. The last experiment reports on utility of a SOLIS as the DA-AdmCtrl admits a larger number of SOLIS requests.

The first experiment compares our admission control with an alternative that utilizes the most compact data representation always, named *Simple*. It assumes the utility of all data formats is identical. Simple provides the same[3] performance as DA-AdmCtrl with the Shallow-Desc inter-dependency, see Figure 3.c. With other inter-dependencies, Simple admits fewer SOLISs because it is too time consuming to produce the most compact representation and the system may have insufficient number of cores to free memory at a fast enough rate to meet the production rate of data by the active SOLISs. This is shown in Table 3 with the Shallow-Asc inter-dependency as a function of the number of cores, i.e., number of simultaneous summarizations. With 32 cores, Simple admits less than 10% of SOLISs supported by DA-AdmCtrl because the time to summarize $F_1$ data to Format $F_4$ is 1000 msec, see Figure 3.d. DA-AdmCtrl utilizes a combination of formats $F_2$ and $F_3$ with service time of 1 and 100 msec, respectively. This ten fold difference in summarization time enables AdmCtrl to admit more than ten times as many SOLISs. As we increase the number of cores, the gap between Simple and DA-AdmCtrl decreases because many more summarization processes may proceed simultaneously. With 3574 cores, both techniques admit the same number of SOLISs. No additional SOLISs may be admitted because the memory (1 GB, see Table 2) is fully occupied with data in format $F_4$.

[3]Simple is inferior when the objective is to maximize utility of data and the most compact representation provides the lowest utility.

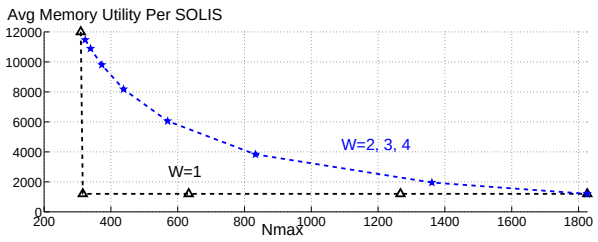



Figure 5: Average utility of data occupying memory per admitted SOLIS, Deep-Asc, $\Delta$=2 minutes.

When memory is not fully occupied, the number of cores plays an important role in the number of admitted SOLISs. Typically, the number of admitted SOLISs increases monotonically as a function of the number of cores. This is the case with Simple which exhibits a linear relationship because it summarizes $F_1$ to $F_4$ always, incurring a 1000 msec service time for each summarization. However, this function is not necessarily linear with DA-AdmCtrl. Its characteristic depends on the service time of the different STs used by a mix picked by AD-AdmCtrl. For example, in Table 3, we increase the number of cores by a factor of eight from 32 to 256 cores while the number of SOLISs with AdmCtrl increases by only a factor of three. This is because DA-AdmCtrl employs a combination of $F_2$ and $F_3$ with 32 cores and a combination[4] of $F_3$ and $F_4$ with 256 cores. The average service time of STs do not exhibit a linear relationship with these mixes.

When both the memory and the number of cores are limited, increasing the value of $\omega$ from 1 to 2 enhances the number of SOLISs admitted by DA-AdmCtrl, see Figure 4.b. When $\omega = 1$, the admission control technique is forced to use only one format. With a handful of cores, it selects $F_2$ which reduces the size of device produced data by a factor of 10. With tens of cores, it selects $F_3$ and applies two STs linearly to produce this data format, admitting a larger number of SOLISs.

With $\omega = 2$ and a handful of cores, DA-AdmCtrl employs a combination of $F_1$ and $F_2$. As the number of cores increases from 2 to 8, DA-AdmCtrl uses the additional processing capability to summarize more data into format $F_2$, admitting more SOLISs. With 32 cores, $\omega = 2$ supports approximately twice as many SOLISs when compared with $\omega = 1$. With additional cores, the gap between $\omega = 1$ and $\omega = 2$ closes because the cores are no longer the limiting factor. With both $\omega$ values, DA-AdmCtrl employs $F_3$ to maximize the number of admitted SOLISs.

Figure 5 shows the average utility of data occupying memory per admitted SOLIS as we increase the number of SOLISs, i.e., system load. When $\omega$ is set to 1, with Deep-Asc, DA-AdmCtrl is forced to use the most compact representation, dropping the utility of data dramatically. With $\omega$ set to 2, DA-AdmCtrl uses a mix of two data formats to realize a drop in data utility as a function of the number of admitted SOLISs.

In the third experiment, we analyzed the inter-dependency of summarization techniques and their impact

[4]With 32 cores, DA-AdmCtrl allocates 45% of memory to $F_2$ and 55% to $F_3$. With 256 cores, DA-AdmCtrl allocates 77% to $F_3$ and the remaining 23% to $F_4$.

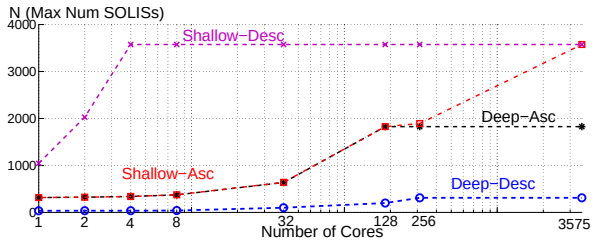



Figure 6: Number of admissible SOLISs ($N$) as a function of the number of cores for all four techniques

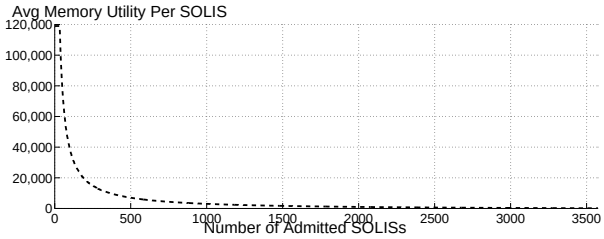



Figure 7: Utility of data as a function of system load, admitted SOLISs. Shallow-Asc, $\Delta$=2 minutes, $\omega$=2.

on DA-AdmCtrl. Figure 6 shows the number of admissible SOLISs by the four inter-dependencies of Figure 3 with different number of cores. Shallow-Desc supports the highest number of SOLISs and realizes this with the fewest number of cores, $D$=4. This is due to the fast summarization time (1 msec) to produce the most compact representation, $F_4$. With all other inter-dependencies, the time to produce this data format is either 1000 msec or higher. Deep inter-dependencies support the fewest number of admitted SOLISs because they prevent DA-AdmCtrl from freeing memory fast enough to meet the rate of data production by SOLISs.

Our final experiment analyzed the utility of data as a function of the number of SOLISs admitted in the system. In this experiment, STs are very fast and their processing time is not a limiting factor. We report on the results with the Shallow-Asc inter-dependency graph with utility of data set to 1000, 100, 10, and 1 for Formats $F_1$, $F_2$, $F_3$, and $F_4$, respectively. Thus, the user perceives device produced data to be a thousand times better than the most compact representation $F_4$. We report the average memory utility per SOLIS, computed by dividing the total utility of data in memory by the number of active SOLISs. Figure 7 shows this metric as a function of the number of SOLIS requests issued to the system. With a few ($\leq$33) SOLISs, DA-AdmCtrl realizes the highest utility per SOLIS by maintaining chase data produced by all SOLISs in their device produced data format. With more than 33 SOLISs, DA-AdmCtrl employs a combination of $F_1$ with one of $F_2$ or $F_3$ (either $\{F_1, F_2\}$ or $\{F_1, F_3\}$) because utility of $F_1$ is ten times higher. With 308 SOLISs, DA-AdmCtrl has insufficient memory to use format $F_1$ and switches to use a combination consisting of $F_2$ with one of the other two data formats, $F_3$ and $F_4$. It abandons use of $F_2$ with more than 1827 SOLISs, using a combination of $F_3$ and $F_4$. Finally, with 3574 SOLISs, it switches to use $F_4$ only.

Beyond 3574 SOLISs, there is insufficient memory with $F_4$ and DA-AdmCtrl rejects additional requests.

The smooth decline in data utility shown in Figure 7 is because DA-AdmCtrl picks mixes with the objective to maximize utility of data while preventing memory overflow.

# E Discussion

Inter-dependencies of Figure 3 highlight the flexibility of DA-AdmCtrl to support arbitrarily complex summarization DAGs (directed acyclic graphs). We now present two concrete example uses of the deep inter-dependencies shown in Figures 3.a and 3.b. While the first is implicit and due to arrival of a new SOLIS request, the second is due to format mismatch between different software packages. We describe these in turn:

1. Consider a scenario in which the available memory supports 5 SOLISs in their device produced format (F1), 6 in format F2 and 7 in format F3. With 5 active SOLISs, DA-AdmCtrl maintains all in their original device produced format (F1). Now when the 6th SOLIS request arrives, DA-AdmCtrl must summarize all chase data of the SOLISs to F2. When the 7th SOLIS request arrives, once again DA-AdmCtrl is invoked and it starts to summarize data into F3. So apart from the incoming chunks, the previously summarized chunks in F2 are summarized to F3. In essence, this is traversing the tree of Figure 3.a/b on-demand (when a new SOLIS request arrives) to summarize chase data into a more compact representation.

2. Xuggler API provides tools for manipulating media streams. Xuggler supports Adobe's RTMP streams as an input whereas some network cameras create an RTSP stream. In order to manipulate video streams using Xuggler, the RTSP streams must be converted to RTMP streams, i.e., using FFMpeg and the RED5 media server. This conversion can be considered as an intermediate summarization step converting F1 (the RTSP stream generated by the device) to F2 (RTMP stream) which can then be used by Xuggler to create other lower resolution formats. With FFMpeg, one may specify the number of frames and their resolution in order to ensure the RTMP formatted output is not greater than the original RTSP format.

# F Conclusions and Future Research Directions

DA-AdmCtrl manages the format of chase data occupying main memory with the objective to maximize the number of admissible SOLISs while preventing overflow of memory. When the user's satisfaction with different data formats (utility of data) is available, DA-AdmCtrl selects data formats with the objective to maximize data utility. It considers arbitrary inter-dependencies between different data formats as long as

they are trees. In addition, it considers the number of cores and schedules summarizations in a manner that prevents memory overflow. Our evaluation section shows different inter-dependencies have an impact on the number of admissible SOLISs. In addition, it highlights the importance of the service time to produce a data format. Use of the most compact representation may not be feasible if the service time of the summarization technique(s) to produce it is too high.

Currently, we are undertaking an implementation of DA-AdmCtrl in RAYS. This implementation envisions a modular and configurable component that can be plugged into different systems that require an admission control technique to manage their memory. It must consider different policies that might be defined by an application. For example, consider an application that assigns a negative cost to both references for missing chase data and SOLIS request rejection. DA-AdmCtrl must intelligently reject (admit by dropping existing) SOLISs to maximize the overall utility observed by the application.

## References


[1] D. Ahmed, N. Chowdhury, and M. Akbar. Admission control algorithm for multimedia server: a hybrid approach. *International Journal of Computers and Applications*, 29(4):414–419, September 2007.

[2] N. Alon, Y. Azar, and S. Gutner. Admission control to minimize rejections and online set cover with repetitions. *ACM Trans. Algorithms*, 6(1):11:1–11:13, December 2009.

[3] W. G. Aref, I. Kamel, and S. Ghandeharizadeh. Disk Scheduling in Video Editing Systems. *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[4] Bambuser. Live From Your Mobile, http://bambuser.com.

[5] S. Barahmand and S. Ghandeharizadeh. RAYS: Recall All You See, Grace Hopper Celebration of Women in Computing, Oregon 2011.

[6] S. Barahmand and S. Ghandeharizadehd. Chase Display of Social Live Streams. In *Proceedings of the 3rd ACM SIGMM International Workshop on Social Media*, November 2011.

[7] Connelly Barnes, Dan B Goldman, Eli Shechtman, and Adam Finkelstein. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(3), August 2010.

[8] A. Blum, A. Kalai, and J. Kleinberg. Admission control to minimize rejections. In *In Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS 2001), LNCS 2125*, pages 155–164. Springer, 2001.

[9] E. Chang and H. Garcia-Molina. BubbleUp: Low Latency Fast-Scan for Media Servers. In *Proceedings of the fifth ACM International Conference on Multimedia,* pages 87–98, November 1997.

[10] M. Chen, D. D. Kandlur, and P. S. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. In *Proceedings of the First ACM International Conference on Multimedia,* August 1993.

[11] S. Cheng, C. Chen, and I. Chen. Performance Evaluation of an Admission Control Algorithm: Dynamic Threshold with Negotiation. *Performance Evaluation*, 52(1), March 2003.

[12] M. Christel, M. Smith, C. Taylor, and D. Winkler. Evolving Video Skims into Useful Multimedia Abstractions. In *CHI*, April 1998.

[13] J. Gemmell and S. Christodoulakis. Principles of Delay-Sensitive Multimedia Data Storage and Retrieval. In *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[14] S. Ghandeharizadeh, L. Huang, and I. Kamel. A Cost Driven Disk Scheduling Algorithm for Multimedia Object Retrieval. *IEEE Transactions on Multimedia*, 5(2):186–196, 2003.

[15] S. Ghandeharizadeh and R. Muntz. Design and Implementation of Scalable Continuous Media Servers. In *Parallel Computing*, 24:91–122, 1998.

[16] L. He, E. Sanocki, A. Gupta, and J. Grudin. Auto-Summarization of Audio-Video Presentations. In *Proceedings of ACM Multimedia*, November 1999.

[17] K. Inkpen, R. Hegde, S. Junuzovic, C. Brooks, and J. Tang. AIR Conferencing: Accelerated Instant Replay for In-Meeting Multimodal Review. In *Proceedings of ACM Multimedia*, October 2010.

[18] X. Jiang and P. Mohapatra. Efficient admission control algorithms for multimedia servers. *Multimedia Syst.*, 7(4), July 1999.

[19] I. Kamel, T. Niranjan, and S. Ghandeharizedah. A Novel Deadline Driven Disk Scheduling Algorithm for Multi-Priority Multimedia Objects. In *Proceedings of IEEE Data Engineering Conference*, 2000.

[20] J. Lui and X. Wang. An Admission Control Algorithm for Providing Quality-of-Service Guarantee for Individual Connection in a Video-on-Demand System. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, 2000.

[21] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *SIGCOMM*, pages 117–130. ACM, 1996.

[22] R. T. Ng and J. Yang. Maximizing Buffer and Disk Utilizations for News On-Demand. In *VLDB*, pages 451–462, 1994.

[23] qik. Record and Share Video Live From Your Mobile Phone, http://qik.com.

[24] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet. In *INFOCOM*, pages 1337–1345, 1999.

[25] P. Shenoy, P. Goyal, and H. Vin. Issues in Multimedia Server Design. *ACM Comput. Surv.*, 27(4):636–639, December 1995.

[26] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: a disk array management system for video files. In *Proceedings of the first ACM international conference on Multimedia*, pages 393–400, 1993.

[27] D. Turner and K. Ross. Adaptive Streaming of Layer-Encoded Multimedia Presentations. *Journal of VLSI Signal Processing*, 34(1):83–99, May 2003.

[28] H. Vin, P. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of the second ACM international conference on Multimedia*, MULTIMEDIA '94, pages 33–40, 1994.