Benjamin Byholm | Fareed Jokhio | Adnan Ashraf |
Sébastien Lafond | Johan Lilius | Ivan Porres

Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud

Turku Centre *for* Computer Science

TUCS Technical Report
No 1121, August 2014

Cost-Efficient, Utility-Based Caching of Expensive Computations in the Cloud

Benjamin Byholm
Fareed Jokhio
Adnan Ashraf
Sébastien Lafond
Johan Lilius
Ivan Porres

TUCS Technical Report
No 1121, August 2014

Abstract

We present a model and system to decide on computing versus storing trade-offs in the Cloud using von Neumann-Morgenstern lotteries. We use the decision model in a video-on-demand system providing cost-efficient transcoding and storage of videos. Video transcoding is an expensive computational process that converts a video from one format to another. Video data is large enough to cause concern over rising storage costs. In the general case, our work is of interest when dealing with expensive computations that generate large results that can be cached for future use. Solving the decision problem entails solving two sub-problems: how long to store cached objects and how many requests we can expect for a particular object in that duration. We compare the proposed approach to always storing and to our previous approach over one year using discrete-event simulations. We observe a $72\,\%$ cost reduction compared to always storing and a $13\,\%$ reduction compared to our previous approach. This reduction in cost stems from the proposed approach storing fewer unpopular objects when it does not regard it cost-efficient to do so.

Keywords: Cache storage, Markov processes, Simulation, Transcoding, Utility theory

TUCS Laboratory
Embedded Systems Laboratory
Software Engineering Laboratory

1 Introduction

In this paper we present a decision model for caching expensive computations producing large amounts of data in a cloud computing setting. This model is applicable to cloud services that produce large amounts of data that can be reused in subsequent requests. As a concrete example, we shall study its application to a cloud-based video transcoding service.

A video transcoding service converts a digital video from one format to another. The need for such a service arises due to the existence of a large number of video compression techniques as well as packaging formats. On the other hand, client devices, and especially mobile devices, can decode and play only a limited number of video formats. If a video is not supported at the client-side device, it needs to be converted into one of the supported formats before playing it. This process is known as video transcoding [19] and it is a CPU-intensive operation. Transcoded videos need to be stored in the server while they are being streamed to the client but they can be deleted from the storage once the streaming operation is completed. Still, there may be new requests for a previously transcoded video in the near future. By storing a transcoded video for an additional amount of time, we can avoid repeating CPU-intensive transcoding operations, thereby saving relatively large amounts of money. After the additional amount of time, we may evaluate the circumstances anew and make a new decision on whether or not to continue with storing the video.

In the context of a pay-per-use cloud computing infrastructure, each transcoding operation has a monetary cost due to use of CPU resources, while video storage has a cost based on the amount of data and time to be stored. In order to reduce the operating costs of the service, we need to decide when and for how long each transcoded video should be cached in the storage. A service that stores data that will not be requested in the future will incur unnecessary storage costs. On the other hand, a service that discards data too eagerly may incur unnecessary computing costs. We require the proper balance.

In this paper we study this problem and propose a decision model for cloud-based caches with the objective to reduce operating costs. In previous work on video transcoding we developed a transcoding–storage cost trade-off strategy called cost and popularity score (CPS) [10], which resulted in significantly lower operating costs compared to always storing. This paper improves the decision process for whether to cache data by applying utility theory through von Neumann-Morgenstern lotteries, which we have previously used for cost-efficient, reliable, utility-based session management in the Cloud [6].

Our utility model for decision making requires three unknown parameters: the storage duration t, the mean number of arrivals $m(t)$ over the storage duration and the popularity distribution p_i of cached objects o_i in the system. We present a natural way of obtaining a good value for the storage duration t, having nice properties that help evaluate the performance of the decision algorithm. We obtain

the number of arrivals $m(t)$ over the storage duration t by solving a subproblem consisting of predicting future arrival counts through singular value decomposition. Finally, we use the Simple Good-Turing frequency estimator to estimate the popularity p_i of each cacheable object.

We evaluate the decision making approaches through discrete-event simulations and find that the proposed approach offers $72\,\%$ lower cost compared to always storing all requested objects. Compared to our previous approach [10], we see $13\,\%$ less cost. This cost reduction stems from the proposed approach storing fewer unpopular objects when it determines that doing so would lead to unnecessary costs compared to not storing.

1.1 Related Work

There are only a few works in the area of computation and storage trade-off analysis for cost-efficient usage of cloud resources. One of the earlier attempts is Adams et al. [1], which addressed the problem of maximizing efficiency by trading storage for computation. It highlighted some of the important issues involved in constructing a cost-benefit model, which can be used to analyze the trade-offs between computation and storage. However, it did not propose a strategy to find the proper balance between computation and storage resources.

Deelman et al. [7] studied cost and performance trade-offs for an astronomy application using Amazon Elastic Compute Cloud (EC2) [1] and Amazon Simple Storage Service (S3)[2] cost models. It also examined the trade-offs between three different data management models for cloud storage, namely remote I/O, regular, and dynamic cleanup. The paper concluded that, based on the likelihood of reuse, storing popular datasets in the Cloud can be cost-efficient. However, it did not provide a concrete strategy for cost-efficient computation and storage of scientific datasets in an actual, cloud-based environment.

The Nectar system [9] is designed to automate the management of data and computation in a data center. It initially stores all the derived datasets when they are generated. However, when the available disk space falls below a threshold, all obsolete or least valued datasets are garbage collected to improve resource utilization. Nectar makes use of the usage history of datasets to perform cost-benefit analysis, which determines the usefulness of each dataset. The cost-benefit analysis considers the size of the dataset, the elapsed time since it was last used, the number of times it has been used, and its cumulative computation time. The datasets with the largest cost-to-benefit ratios are deleted. Although Nectar provides a computation and storage trade-off strategy, it is not designed to reduce the total cost of computation and storage in a cloud-based service which makes use of infrastructure as a service (IaaS) resources.

[1] `http://aws.amazon.com/ec2/`

[2] `http://aws.amazon.com/s3/`

Yuan et al. [22] proposed two strategies for cost-efficient storage of scientific datasets in the Cloud, which compare the computation cost and the storage cost of the datasets, and a cost transitive tournament shortest path (CTT-SP) algorithm to find the best trade-off between the computation and the storage resources. The strategies are called cost rate based storage strategy [21, 24] and local-optimization based storage strategy [23]. The cost rate based storage strategy compares computation cost rate and storage cost rate to decide storage status of a dataset. Whereas, the local-optimization based storage strategy partitions a data dependency graph (DDG) of datasets into linear segments and applies the CTT-SP algorithm to achieve a localized optimization. The local-optimization based storage strategy tends to be more cost-efficient than the cost rate based storage strategy. However, due to the overhead introduced by the CTT-SP algorithm, it is less efficient and less scalable. The DDG-based local-optimization based storage strategy of Yuan et al. [23], which provides cost-efficient results for scientific datasets, is not much relevant for video transcoding, as the latter involves few data dependencies.

Jokhio et al. [10, 14] presented a computation and storage trade-off strategy called CPS. It estimates an equilibrium point on the time axis where the computation cost and the storage cost of a transcoded video become equal. It also estimates video the popularity of the individual transcoded videos to differentiate among videos based on their levels of popularity.

Kathpal et al. [15] analyzed compute versus storage trade-off for transcoded videos. It proposed an elimination metric to decide which transcoded videos can be removed from the video repository. However, in contrast to the cost and popularity score based strategy of Jokhio et al. [10], it did not account for the video popularity score. Moreover, although the results are also based on Amazon EC2 and Amazon S3, it used relatively short videos, which comprise of clips with durations up to 60 s.

2 A Video On-Demand System

The system architecture of the cloud-based, on-demand video transcoding service is shown in Fig. 1. It consists of a *streaming server*, a *video splitter*, a *video merger*, a *video repository*, a dynamically scalable cluster of *transcoding servers*, a *load balancer*, a *master controller*, and a *load predictor*. The video requests and responses are routed through the *streaming server*. Since our main focus in this paper is on computation and storage trade-off for video transcoding, we assume that the *streaming server* will not become a bottleneck.

The video streams are stored in the *video repository* in various compressed formats. The *streaming server* accepts video requests from users and checks if the required video is available in the *video repository*. If it finds the video in the desired format and resolution, it starts streaming the video. However, if it finds that the requested video is stored only in another format or resolution than the one desired

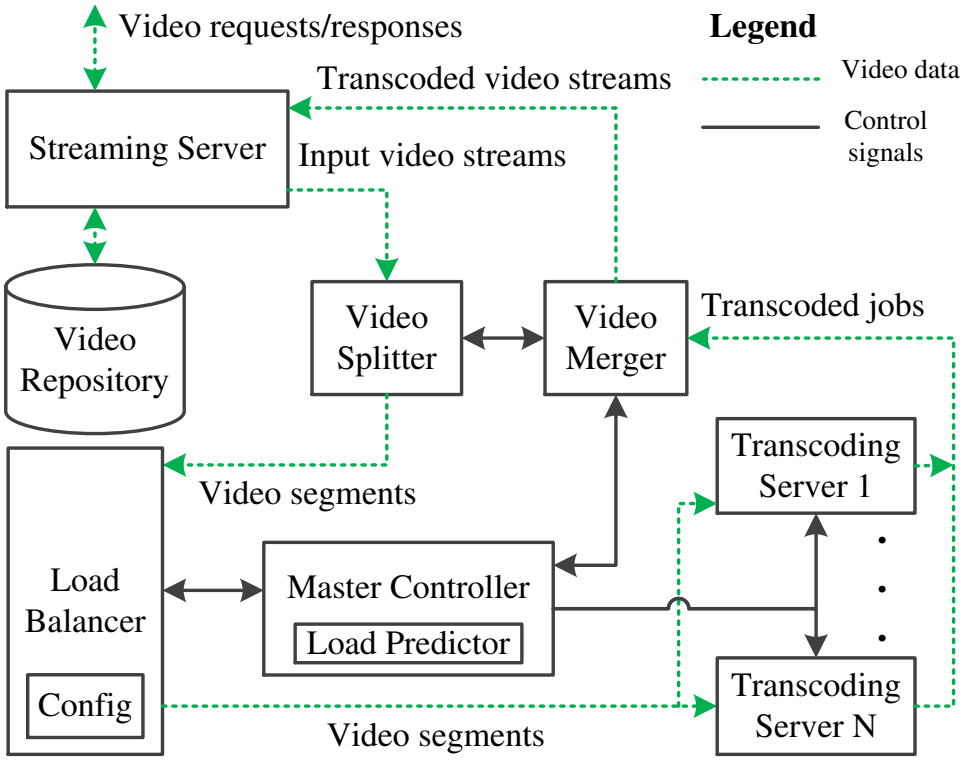


Figure 1: Architecture of the video transcoding service.

by the user, it sends the video for segmentation and subsequent transcoding. Then, as soon as it receives the transcoded video from the *video merger*, the *streaming server* begins streaming the video.

After each transcoding operation, the computation and storage trade-off strategy determines if the transcoded video should be stored in the *video repository* or not. Moreover, if a transcoded video is stored, then the trade-off strategy also determines the duration for which the video should be stored. Therefore, it allows us to trade computation for storage or vice versa in order to reduce the total operating costs and to improve the overall performance of the video transcoding service.

The *video splitter* splits the video streams into smaller segments called jobs, which are placed into the job queue. Further discussion on video segmentation at the group of pictures (GOP) level is provided in Jokhio et al. [12, 13].

The *load balancer* distributes load on the transcoding servers. In other words, it routes and load balances transcoding jobs on the *transcoding servers*. It maintains a configuration list of active *transcoding servers*. This list is updated often as a result of dynamic virtual machine (VM) allocation and deallocation. The *load balancer* serves the jobs in FIFO (First In, First Out) order. It implements one or more job scheduling policies, such as, the *shortest queue length* policy, which selects a *transcoding server* with the shortest queue length and the *shortest queue waiting time* policy, which selects the *transcoding server* that currently has the shortest available queue waiting time.

The actual transcoding is performed by the transcoding servers. They get compressed video segments, perform the required transcoding operations, and return the transcoded video segments for merging. A *transcoding server* runs on a dynamically provisioned VM. Each *transcoding server* processes one or more simultaneous jobs. When a transcoding job arrives at a *transcoding server*, it is placed in the server's queue, from where it subsequently will be processed.

The *master controller* acts as the main controller and resource allocator. It implements prediction-based dynamic resource allocation and deallocation algorithms [11] and one or more computation and storage trade-off strategies. The resource allocation and deallocation is mainly based on the target play rate of the video streams and the predicted transcoding rate of the transcoding servers. The *master controller* uses the *load predictor* for load prediction [2]. The *video merger* merges the transcoded jobs into video streams, which form video responses. Our resource allocation and load prediction algorithms are described in detail in Jokhio et al. [11] and Ashraf et al. [2]. In this paper, our primary focus is on a cost-efficient computation and storage trade-off strategy.

3 Utility Model for Computing Versus Storing

In this section we present the utility model that will be used to govern the caching strategy of the service. We proceed by introducing the basics of von Neumann-Morgenstern lotteries.

3.1 Von Neumann-Morgenstern Lotteries

A von Neumann-Morgenstern lottery [20] consists of mutually exclusive outcomes that may occur with a given probability. The sum of probabilities in a lottery should be equal to one. For example, the simple lottery L described by the equation

$$L = 0.20A + 0.80B \tag{1}$$

denotes a scenario where the probability of event A, $P(A) = 0.20$, the probability of event B, $P(B) = 0.80$, and exactly one of the possible outcomes will occur. In general, a lottery L with n outcomes A_i and probabilities p_i is expressed as:

$$\begin{aligned} L &= \sum_{i=0}^{n} p_i A_i \\ \text{subject to } &\sum_{i=1}^{n} p_i = 1 \end{aligned} \tag{2}$$

According to the von Neumann-Morgenstern utility theorem [20], an agent faced with the problem of choosing between a set of lotteries has a utility function, provided that the four axioms of the theorem are satisfied. The four axioms of the utility theorem on lotteries L, M and N are:

- *completeness* (L or M is preferred, or they are equal)
  $L \preceq M \vee M \preceq L$
- *transitivity* (consistent preference across 3 operations)
  $(L \preceq M \wedge M \preceq N) \rightarrow L \preceq N$
- *continuity* (transitive preference is continuous)
  $(L \preceq M \wedge M \preceq N) \rightarrow \exists p \in [0,1] pL + (1-p)N = M$
- *independence* (independence of irrelevant alternatives)
  $L \prec M \rightarrow \forall N \forall p \in (0,1] pL + (1-p)N \prec pM + (1-p)N$

If an agent satisfies these axioms, it has a utility function u, assigning a real value $u(A)$ to every possible outcome A, so that for any two lotteries L and M, $Eu(L)$ is the expected value of u in L, $Eu(M)$ is the expected value of u in M and

$$L \prec M \leftrightarrow Eu(L) < Eu(M) \tag{3}$$

By using the utility function we can determine which lotteries to play. In this paper we will model the choice between computing and storing as von Neumann-Morgenstern lotteries. By choosing among lotteries, we can make the most cost-efficient decisions.

There are, however, some limitations to von Neumann-Morgenstern utility. Von Neumann and Morgenstern [20] acknowledged that nested gambling is ignored. An example of nested gambling with lotteries L and M would be $pL + (1-p)M$, which gets treated as a lottery itself. Another limitation is that utilities cannot be compared between agents X and Y with different utility functions u_X and u_Y. Expressions like $u_X(L) + u_Y(L)$ are undefined. As we use neither nested gambling nor multiple agents, these limitations do not affect us. We may design a utility function that incorporates risk aversion or diminishing returns, which could be beneficial in an environment with a relatively high degree of uncertainty.

Assuming i.i.d. arrivals, we can model requests arriving to the system as an inhomogeneous Poisson process $N(t)$ with rate $\lambda(t)$, where $N(t)$ is the number of arrivals by time t . The mean number of arrivals $m(t)$ by time t is given by

$$m(t) = \int_0^t \lambda(u)\,\mathrm{d}u \tag{4}$$

$N(t)$ has a Poisson distribution with parameter $m(t)$:

$$P(N(t) = k) = \frac{m(t)^k}{k!} e^{-m(t)} \tag{5}$$

Using Poisson splitting, we can model the requests arriving for each video as independent Poisson processes $N_i(t)$ with mean number of arrivals $p_i m(t)$, where p_i is the relative frequency of requests for video i. Each process $N_i(t)$ has a Poisson distribution with parameter $p_i m(t)$, given by the equation

$$P(N_i(t) = k) = \frac{(p_i m(t))^k}{k!} e^{-p_i m(t)} \tag{6}$$

Requests may arrive to the system at any time. Whenever a video is requested, we check if a cached copy is available. If a cached copy is available, we serve the request from the cache at no extra cost. If there is no cached copy available, we need to transcode the corresponding source video into the correct format, this will incur a fixed cost c_t of transcoding. After the video has been transcoded, we have the option of storing the result in the cache for duration t at cost $c_s t$, after which we can decide to continue storing it for a new duration t at cost $c_s t$ or to delete it from the cache at no further cost.

If we decide to cache a transcoded video for duration t it will always cost $c_s t$, regardless of whether any further requests arrive for the video or not. If we decide not to cache the transcoded video, one of two possible outcomes will occur: either further requests for that video arrive, say $n_i \geq 1$ requests, at which point we will have to transcode the source video again at cost $n_i c_t$, or no further requests arrive, costing us nothing. Thus, we have the following set of possible outcomes:

A: Delete, no requests arrive

B: Delete, requests arrive

C: Store, no requests arrive

D: Store, requests arrive

Using (6), we can compute the probability of 0 arrivals, $P(N_i(t) = 0) = e^{-p_i m(t)}$. Conversely, the probability of more than 0 arrivals is $1 - P(N_i(t) = 0) = 1 - e^{-p_i m(t)}$. Assuming that n_i is approximately equal to to the mean of the corresponding zero-truncated Poisson process $n_i \approx \frac{p_i m(t)}{1-e^{-p_i m(t)}}$, we obtain the following utilities for each outcome:

$$
\begin{aligned}
u(A) &= 0 \\
u(B) &= -C_{ti}\frac{p_i m(t)}{1-e^{-p_i m(t)}} \\
u(C) &= -C_{si}t \\
u(D) &= -C_{si}t
\end{aligned}
$$

We can then formulate the alternatives as von Neumann-Morgenstern lotteries, L_d for deleting and L_s for storing:

$$L_d = P(N_i(t) = 0)A + (1 - P(N_i(t) = 0))B \tag{7}$$
$$L_s = P(N_i(t) = 0)C + (1 - P(N_i(t) = 0))D \tag{8}$$

The two lotteries have the following expected utilities:

$$Eu(L_d) = -p_i m(t) C_{ti} \tag{9}$$
$$Eu(L_s) = -C_{si}t \tag{10}$$

Always choosing the lottery with the highest expected utility should give us the best result in the long run. However, if we are to compute the expected utility of each lottery, we need actual values for $m(t)$ as well as p_i, none of which are directly observable. In Sections 4 and 5 we present possible methods for estimating these parameters. If necessary, it would be possible to use a set of estimators better suited for a particular problem.

3.2 Determining the Duration for Caching

To decide whether to cache the results of a computation, we first need to determine when to make this decision. If we decide not to store a transcoded video, we naturally cannot make further decisions for that video until it has been transcoded again, as the data is discarded. If we decide to store a transcoded video, we also

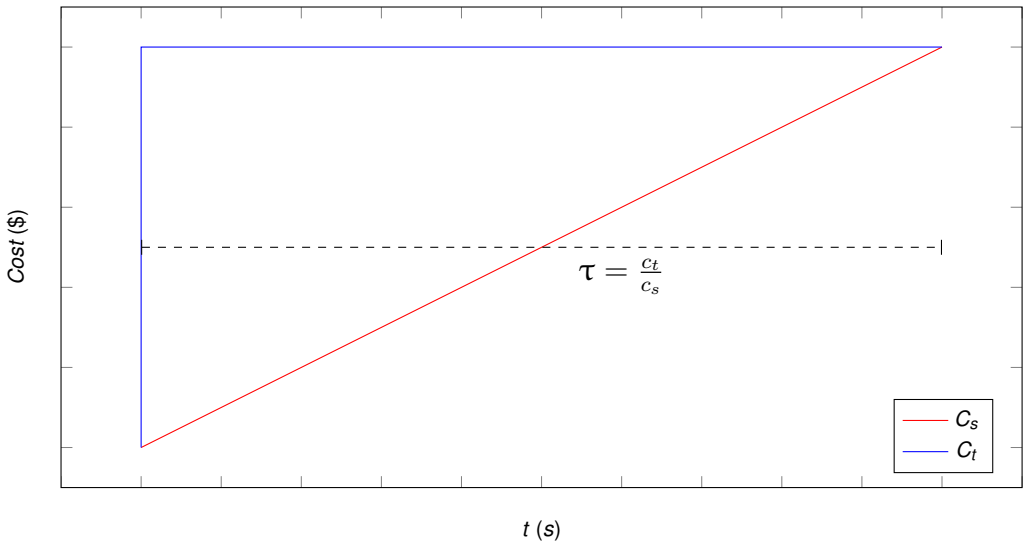


Figure 2: Storage cost C_s approaches transcoding cost C_t.

need to determine how long to store it. Storing indefinitely is not viable. Jokhio et al. [10, 14] determined that there is an equilibrium point τ, where the cost of storing over time C_s becomes equal to the cost of transcoding C_t. Figure 2 illustrates how storage cost and transcoding cost are related. For example, if the cost of transcoding a given video v_i is $C_{ti} = \$1$ and the cost of storing that video is $C_{si} = \$0.5\,\mathrm{d}^{-1}$, storing for $\frac{\$1}{\$0.5\,\mathrm{d}^{-1}} = 2\,\mathrm{d}$ results in the same cost as transcoding once: $C_{ti} = \tau C_{si} = C_i$. This means that if we store a video v_i for duration τ and get at least one request for it, say n_i requests, we will break even or even save money compared to transcoding it n_i times, as $C_i \leq n_i C_i, n_i \geq 1$. Setting $t = \tau$ in (9) results in a simplified expression, in which the costs are now equal:

$$Eu(L_d) = -p_i m(\tau) C_i \tag{11}$$
$$Eu(L_s) = -C_i \tag{12}$$

Thus, we can conclude that we should decide to store whenever

$$p_i m(\tau) >= 1 \tag{13}$$

4 Arrival Rate Prediction

Obtaining $m(\tau)$ in (13) requires knowledge of the future arrival rate up to time τ. We have decided to use a time-series prediction approach based on truncated singular value decomposition, as outlined by Shen and Huang [18], which presented methods for predicting future arrivals to a call center.

4.1 Truncated Singular Value Decomposition

Let $\boldsymbol{X}$ be an $n \times m$ matrix recording the number of requests for n days, each day having m time periods. The rows of this matrix constitute a vector-valued time series in $\mathbb{R}^m$. We wish to build a time-series model and forecast future values. The dimensionality of this time series is large. We can significantly reduce the dimensionality through the following decomposition:

$$\boldsymbol{x}_i = \beta_{i1}\boldsymbol{f}_1 + \cdots + \beta_{iK}\boldsymbol{f}_K + \boldsymbol{\epsilon}_i, \quad i = 1, \ldots, n \tag{14}$$

where $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_K \in \mathbb{R}^m$ are basis vectors and $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_n \in \mathbb{R}^m$ are the corresponding error terms.

The singular value decomposition (SVD) of $\boldsymbol{X}$ is given by

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\intercal \tag{15}$$

and gives the solution for minimizing the error terms in (14):

$$\boldsymbol{x}_i \simeq s_1 u_{i1}\boldsymbol{v}_1 + \cdots + s_K u_{iK}\boldsymbol{v}_K \tag{16}$$

where $s_1, \ldots, s_K$ are the K largest singular values obtained from the diagonal of $\boldsymbol{S}$, $u_{i1}, \ldots, u_{iK}, \quad i = 1, \ldots, n$ are the entries of column i in $\boldsymbol{U}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K$ are the corresponding columns of $\boldsymbol{V}$. We can now forecast each series $\{\beta_{ik}\}$ separately, which we do by extrapolating from linear regression. We can then predict the number of arrivals in future days according to

$$\hat{\boldsymbol{x}}_{n+h} = \hat{\beta}_{n+h,1}\boldsymbol{f}_1 + \cdots + \hat{\beta}_{n+h,K}\boldsymbol{f}_K \tag{17}$$

However, to keep predictions non-negative, we employ the root-unroot method of Brown et al. [5]: we transform $\boldsymbol{X}$ into $\sqrt{\boldsymbol{X} + \frac{1}{4}}$ before the SVD and restore the prediction as $\hat{\boldsymbol{x}}^2 - \frac{1}{4}$.

SVD generally requires $\mathcal{O}\left(m^2 n + n^3\right)$ operations on an $m \times n$ real matrix. However, as we only want the K most important singular values, we can use efficient algorithms for truncated singular value decomposition. We therefore use the implicitly restarted Lanczos bidiagonalization method of Baglama and Reichel [4], implemented in the irlbpy library [3]. In practice, this method requires as little as $\mathcal{O}\left(mnK\right)$ operations [3].

5 Frequency Estimation

Suppose there are five kinds of videos: $\{A, B, C, D, E\}$. You observe the number of times each video is requested and find 2 requests for A, 2 requests for B, 3 requests for C and 1 request for D. A naïve frequency estimator, like the maximum likelihood frequency estimator, will assign a probability p_i of .25 to A,

.25 to B, .375 to C and .125 to D. Notice that no requests for E were observed, resulting in the maximum likelihood frequency estimator assigning a probability of 0 for requests to the video. However, this must be false, because we know that it is possible to request the video. Video E should therefore have a probability greater than zero. The maximum likelihood estimator does not consider missing samples.

5.1 The Good-Turing Frequency Estimator

Since we are most concerned about low expected numbers of requests, we are dealing with rare events of which we might not have made any observations. Similarly, when new videos are made available, we will again be dealing with a lack of information. This is why we propose to use the Simple Good-Turing frequency estimator [8], which accounts for unobserved events. The estimator tends to underestimate frequent items, but this can be mitigated by using the empirical estimate for them [16]. However, as we are really only interested in infrequent items where $p_i m(t) \approx 1$, it is not necessary for our particular case. The Simple Good-Turing frequency estimator assigns these probabilities to the example with the five videos: .22 to A, .22 to B, .285 to C, .15 to D, and .125 goes to E.

The Good-Turing frequency estimators assume that the observed items follow a binomial distribution [8], but we assume that they follow a Poisson distribution. However, this is not a problem in our case, as the binomial distribution converges to the Poisson distribution as $np = \lambda, n \to \infty, p \to 0$. With a large number of observable items, the probabilities will be small for infrequent items. The Good-Turing frequency estimators also assume that the underlying frequency distribution is static. We postulate that we can relax this assumption by using a sliding window. We only need to ensure that the window is large enough, compared to the mean number of arrivals $m(t)$, to give a suitable number of observations n within the window.

6 Evaluation Using Discrete-Event Simulations

We used SimPy[3] to develop a discrete-event simulation of a video transcoding service. We decided to compare the proposed, utility-based approach with the previously developed CPS policy and a reference policy based on always storing.

6.1 Setting up the Experiment

We simulated a video transcoding service with 10 000 videos over one year. Video sizes were randomly assigned according to a double Pareto-lognormal distribution [17] with parameters $\alpha = 2, \beta = 4, \mu = 0, \sigma = 1$ scaled by a factor of 256 MiB. The probability of a request to the system belonging to a particular

[3] `http://simpy.readthedocs.org/`

video is given by a truncated Pareto distribution with parameters $x_m = 1, \alpha = 2$. Transcoding cost $\$1.7 \times 10^{-5}\,\mathrm{s}^{-1}$, the cost of a medium instance in Amazon EC2. Transcoding rate was $2.4\,\mathrm{MiB\,s}^{-1}$. Storage cost $\$3.6 \times 10^{-11}\,\mathrm{MiB}^{-1}\,\mathrm{s}^{-1}$, as in Amazon S3. Thus, the time to store each video was

$$\tau = \frac{\$1.7 \times 10^{-5}\,\mathrm{s}^{-1}}{2.4\,\mathrm{MiB\,s}^{-1} \times \$3.6 \times 10^{-11}\,\mathrm{MiB}^{-1}\,\mathrm{s}^{-1}} = 55\,\mathrm{h} \tag{18}$$

The arrival process was a randomly generated inhomogeneous Poisson process. We constructed the arrival process by thinning a homogeneous Poisson process with rate $\lambda = 20\,\mathrm{s}^{-1}$ with a Bernoulli trial with a time-dependent probability $p(t)$. We generated the probability vector by dividing the simulation duration into a random number X of parts according to a Poisson distribution with mean $\mu = 26$. We then generated an exponentially distributed duration with mean $\omega = \frac{t_{\text{total}}}{X}$ for each interval. With this information we finally generated a linear spline with random coefficients, assuming values between 0 and 1. Figure 3 shows the resulting mean arrival rate over time.

We sought to compare the proposed, utility-based approach with the the established CPS approach. We also included the always store policy and a version of the utility-based approach with perfect knowledge of the mean number of arrivals $m(t)$ and popularity p_i in the benchmarks, acting as references. To enable direct comparison of the approaches by classifying each decision as good, bad or neutral, we used a minimum storage duration of $SD_{\tau_i} = 55\,\mathrm{h}$ for the CPS approach. This resulted in marginally higher cost for this approach, compared to when using a minimum duration of $24\,\mathrm{h}$, but this increase was not significant enough to alter the outcome of the experiments. The reference policy made no decisions, as it always chose to store.

For the utility-based approach, the rate predictor used a sliding past window of $7\,\mathrm{d}$. The popularity estimator also used a sliding window of $7\,\mathrm{d}$. Because the system cannot make good predictions before historical data have been collected, all requested videos were always stored during the first $7\,\mathrm{d}$. Samples of request counts were collected every hour: $t_s = 1\,\mathrm{h}$.

6.2 Results

Figure 3 also shows the predicted arrival rate obtained through truncated singular value decomposition. The prediction accuracy appears good considering the simplicity of the used approach. The root-mean-square error (RMSE) was $3.3\,\mathrm{s}^{-1}$. The standard deviation of a Poisson distributed variable is equal to the square root of the mean parameter, which varied between $0.8\,\mathrm{s}^{-1}$ and $4.4\,\mathrm{s}^{-1}$, with a mean of $3.3\,\mathrm{s}^{-1}$. The RMSE was thus within one standard deviation. The total operating cost over time is shown in Fig. 4. Always storing cost $\$605$, CPS cost $\$196$ and the utility-based approach cost $\$171$ ($\$164$ with perfect information). The utility-based approach clearly operates at a lower cost than the other approaches.

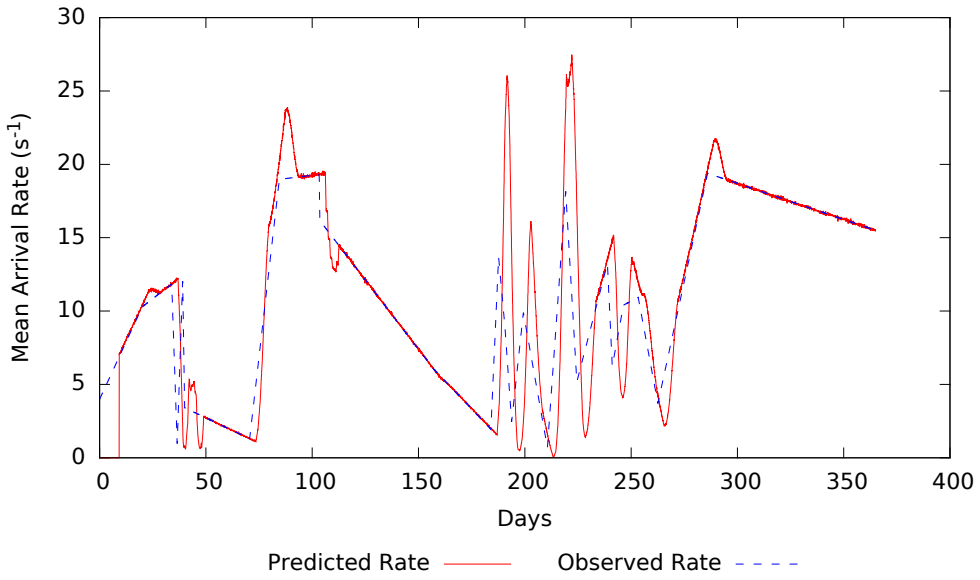


Figure 3: Observed and predicted arrival rate for the system.

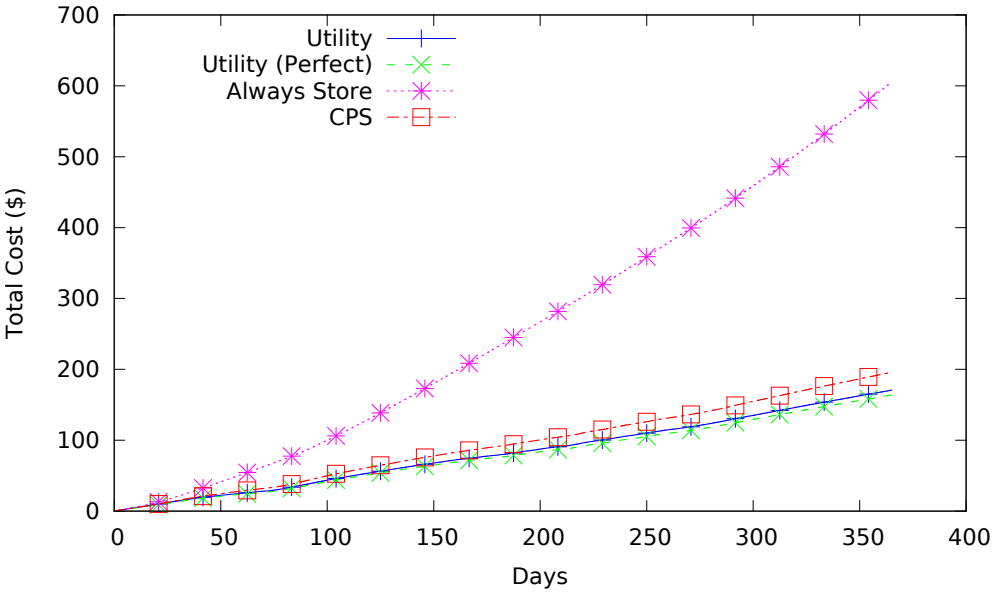
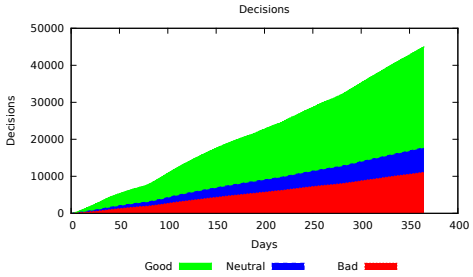
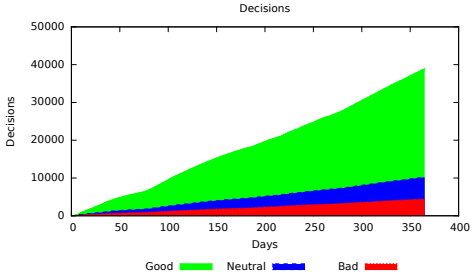


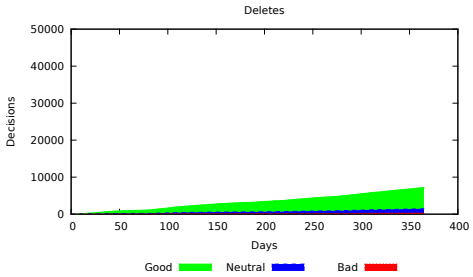
Figure 4: Cost of operation for the approaches over one year.



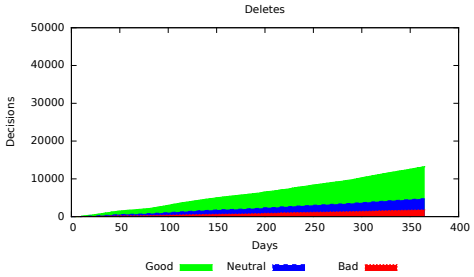
(a) Decisions by the CPS approach.



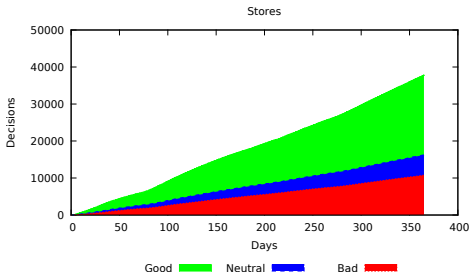
(b) Decisions by the utility-based approach.



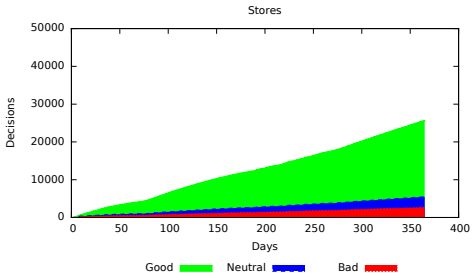
(c) Deletes by the CPS approach.



(d) Deletes by the utility-based approach.



(e) Stores by the CPS approach.



(f) Stores by the utility-based approach.

Figure 5: Classification of decisions made by the two actual approaches as good, bad or neutral.

Table 1: Bad, neutral and good operations per approach

Operation	Approach	Bad	Neutral	Good
Delete	CPS	4 %	16 %	80 %
	Utility	14 %	22 %	64 %
Store	CPS	28 %	15 %	57 %
	Utility	10 %	11 %	79 %
All	CPS	25 %	15 %	60 %
	Utility	11 %	15 %	74 %

Compared to always storing, the utility based approach operated at $1 - \frac{\$171}{\$605} = 72\,\%$ less cost. Compared to the CPS approach, the utility based approach operated at $1 - \frac{\$171}{\$196} = 13\,\%$ less cost. With perfect information, the utility-based approach operated at $1 - \frac{\$164}{\$171} = 4\,\%$ less cost than the actual implementation. This is a small difference, indicating good decision making by the actual implementation. The reason why the proposed approach fares better than the previous can be seen by classifying each decision as good, bad or neutral, as can be seen in Fig. 5.

6.3 Analysis

The utility-based approach operates at less cost than the CPS approach because it did fewer bad decisions, as seen in Figs. 5a and 5b. A bad decision is either a bad decision to store or a bad decision to delete. A bad decision to store means that no requests arrived for the video in question during duration τ, meaning that it cost us C_i when it could have cost us \$0 if we had chosen the other alternative. Conversely, a bad decision to delete means that more than one request arrived for the video in duration τ, which cost us more than storing at cost C_i would have done. When only one request arrives in duration τ, it does not matter whether we store or delete, as storing for duration τ costs the same as transcoding once: C_i. These decisions are thus neutral. Table 1 shows the proportion of bad, neutral, and good delete and store operations for each approach. CPS made $44\,973$ decisions, comprising 7172 deletes and $37\,801$ stores. The utility-based approach made $38\,894$ decisions, comprising $13\,214$ deletes and $25\,680$ stores. While the utility-based approach did slightly more erroneous deletes than the CPS approach, it still did far fewer erroneous stores. The CPS approach deletes too rarely.

7 Conclusions

We presented a utility-based decision strategy for caching expensive computations in a cloud-computing setting. We exemplified our approach with a video transcoding service, but we believe that a similar approach can be used in other services that transcode, uncompress or index large amounts of data based on user

requests. Our model requires three unknown parameters: the storage duration t, the mean number of arrivals $m(t)$ over the storage duration and the popularity distribution p_i of objects to store in the cache o_i. We presented a formally justified way of obtaining a good value for the storage duration t, having good properties that helped evaluate the performance of the decision algorithm. We obtained the mean number of arrivals $m(t)$ over the storage duration t by solving a subproblem consisting of predicting future arrival counts through singular value decomposition. Finally, we used the Simple Good-Turing frequency estimator to estimate the relative popularity p_i of each available video in the system.

We evaluated our approach using discrete-event simulations. The utility-based approach incurred $72\,\%$ less cost than always storing and $13\,\%$ less cost than the CPS approach [10] over one simulated year. Prediction accuracy was high, as evidenced by only obtaining $4\,\%$ less cost when using perfect information. This made the utility-based approach better. We only evaluated the system with a static popularity distribution, but have accounted for non-static distributions by using a sliding window for popularity estimation. Determining the sizes of the sliding windows used by the arrival rate predictor and popularity estimator is a domain-specific problem which we have only touched upon lightly. Given a good estimator for non-static popularity distributions, the utility-based approach should work well. We have assumed independent arrivals, which is not always true of a real-world service. However, our approach can also be evaluated against actual request logs if made available.

Acknowledgments

This work was supported by the N4S research project. Adnan Ashraf and Fareed Jokhio were partially supported by the Foundation of Nokia Corporation and by doctoral scholarships from the Higher Education Commission (HEC) of Pakistan.

References

[1] Ian F. Adams, Darrell D. E. Long, Ethan L. Miller, Shankar Pasupathy, and Mark W. Storer. Maximizing efficiency by trading storage for computation. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[2] Adnan Ashraf, Benjamin Byholm, and Ivan Porres. A session-based adaptive admission control approach for virtualized application servers. In *Utility and Cloud Computing (UCC), 5th IEEE/ACM International Conference on*, pages 65–72, 2012.

[3] James Baglama, Michael Kane, Bryan Lewis, and Lothar Reichel. *irlbpy: Truncated SVD by Implicitly Restarted Lanczos Bidiagonalization for NumPy*, 2013.

[4] James Baglama and Lothar Reichel. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42, 2005.

[5] Lawrence Brown, Noah Gans, Avishai Mandelbaum, Anat Sakov, Haipeng Shen, Sergey Zeltyn, and Linda Zhao. Statistical analysis of a telephone call center: A queueing-science perspective. *Journal of the American Statistical Association*, 100(469):36–50, 2005.

[6] Benjamin Byholm and Iván Porres. Cost-efficient, reliable, utility-based session management in the cloud. In *14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 102–111. IEEE Computer Society, 2014.

[7] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[8] William A. Gale and Geoffrey Sampson. Good-Turing frequency estimation without tears. *Journal of Quantitative Linguistics*, 2(3):217–237, 1995.

[9] Pradeep Kumar Gunda, Lenin Ravindranath, Chandramohan A. Thekkath, Yuan Yu, and Li Zhuang. Nectar: automatic management of data and computation in datacenters. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[10] Fareed Jokhio, Adnan Ashraf, Sébastien Lafond, and Johan Lilius. A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud. In *39th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 365–372. IEEE Computer Society, 2013.

[11] Fareed Jokhio, Adnan Ashraf, Sébastien Lafond, Ivan Porres, and Johan Lilius. Prediction-based dynamic resource allocation for video transcoding in cloud computing. In *Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on*, pages 254–261, 2013.

[12] Fareed Jokhio, Tewodros Deneke, Sébastien Lafond, and Johan Lilius. Bit rate reduction video transcoding with distributed computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 206–212, feb 2012.

[13] Fareed A. Jokhio, Tewodros Deneke, Sébastien Lafond, and Johan Lilius. Analysis of video segmentation for spatial resolution reduction video transcoding. In *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposium*, dec 2011.

[14] Fareed Ahmed Jokhio, Adnan Ashraf, Sébastien Lafond, Ivan Porres, and Johan Lilius. Cost-efficient dynamically scalable video transcoding in cloud computing. Technical Report 1098, Turku Centre for Computer Science (TUCS), dec 2013.

[15] Atish Kathpal, Mandar Kulkarni, and Ajay Bakre. Analyzing compute vs. storage tradeoff for video-aware storage efficiency. In *Proceedings of the 4th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'12, pages 13–13. USENIX Association, 2012.

[16] Alon Orlitsky, Narayana P. Santhanam, and Junan Zhang. Always Good Turing: Asymptotically optimal probability estimation. *Science*, 302(5644):427–431, 2003.

[17] William J. Reed and Murray Jorgensen. The double Pareto-lognormal distribution—A new parametric model for size distributions. *Communications in Statistics-Theory and Methods*, 33(8):1733–1753, 2004.

[18] Haipeng Shen and Jianhua Z. Huang. Interday forecasting and intraday updating of call center arrivals. *Manufacturing and Service Operations Management*, 10(3):391–410, 2008.

[19] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: An overview. *Signal Processing Magazine, IEEE*, 20(2):18–29, mar 2003.

[20] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, third edition, 1953.

[21] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, 2010.

[22] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. Computation and storage trade-off for cost-effectively storing scientific datasets in the cloud. In *Handbook of Data Intensive Computing*, pages 129–153. Springer New York, 2011.

[23] Dong Yuan, Yun Yang, Xiao Liu, and Jinjun Chen. A local-optimisation based strategy for cost-effective datasets storage of scientific applications in the

cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 179–186, 2011.

[24] Dong Yuan, Yun Yang, Xiao Liu, Gaofeng Zhang, and Jinjun Chen. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. *Concurrency and Computation: Practice and Experience*, 24(9):956–976, 2012.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN ?
ISSN 1239-1891