Verifying Elasticity in a gray-scale conversion algorithm

Diana Popa
Computer Science Department, Faculty of Automatic Control and Computers
University Politehnica of Bucharest
Bucharest, Romania
diana.popa@cti.pub.ro

***Abstract*—The rise of the cloud services has triggered a significant interest in the area of cloud benchmarking, the focus lying mainly on either of the following metrics: scalability, efficiency, elasticity, throughput or multi-tenancy. Measuring performance results do not only influence future performance improvements, but they could also help in analysing provisioning algorithms and they could also offer a comparative view in terms of cost and performance over multiple similar systems. This paper aims to verify the cloud elasticity of a popular algorithm for converting coloured images to grey-scale ones by offering real-life measurement results. For the purpose of this paper, we chose a costly algorithm in terms of computation and we implemented the distributed version of it in order to be able to run it on a cluster consisting of multiple computational machines. Different stages of the algorithms are run independently on an increasing number of machines in order to record the execution time for each one of them. By analysing the variations in speed-up, we fix an optimal number of machines for every stage of the algorithm such that the hardware resource allocation is as close as possible to the demand in resources.**

***Index Terms*—elasticity, cloud computing, grey-scale, cluster, performance measurements.**

I. Introduction

In the following sections of this paper we present a measuring technique, discuss the experimental results obtained and analyse how well an arbitrary cloud infrastructure performs on an image gray-scaling process when it comes to elasticity.

One of the most appealing features when it comes to cloud computing and cloud services in general is the specific 'pay-as-you-go' payment method, or 'pay-as-you-consume' as per the authors of [10], where the user is charged based on the exact number of the resources (i.e. virtual machines) that are being used. Thus, the user of the cloud service or even the cloud system itself can vary the amount of resources based on the demand. While doing so, there are certain problems that can arise. One of them is under-provisioning where the demand in resources is higher than the actual resources allocated to the current workload. Another one is over provisioning where the consumer is being given more resources than are actually needed. The goal of any cloud provisioning algorithm is to find a way to minimize both of those two issues and to satisfy in a correct manner the type of elastic needs that the consumer has. By doing so, clouds services come as a great cost saving technique compared to some traditional service platform, which loads the available infrastructure at maximum capacity without taking into account the actual needs of the workload scheduled to run.

Given that elastic provisioning is one critical component of every cloud environment, dynamically detecting the needs in resource allocation and being able to cut off and add server instances such that the costs incurred are optimized represents the main goal of any resource provisioning technique as this is the secret to minimize overall cost for the cloud consumer. In our paper, we try to verify this important feature of the cloud, which is elasticity, by running a intensively computational algorithm on a cloud platform, dividing its logic work-flow into different steps and benchmarking the speed-up on each of these steps on an increasing number of physical machines. One of the first hardnesses that arises, when setting a goal such as measuring the elasticity of an algorithm, is actually finding the right algorithm. The number of computational steps has to be at least three in order to be able to note the variation of the optimal tasks needed on each of these steps. If, for example, we would use only two steps we would only be able to see either an increase or an decrease, without being able to see both of them manifesting simultaneously on that specific algorithm, and for proving that elasticity actually exists, we have to experimentally demonstrate the variation of resources allocated in both directions. Another challenge is that the steps involved have to allow panellization as we have to run them in distributed mode. This enables us to extract the most convenient number of machines for every step by observing where the speed-up curve becomes steady. One other aspect to take into account is that the steps perform CPU intensive operations. We are interested in proving that once a machine is fully loaded, the addition of another one reflects itself on the execution time and thus the cost of paying for it is plausible.

This paper is organized in the following manner: section II provides an overview on previous work done in fields that are related to this paper: presenting a definition to elasticity and popular tools for benchmarking cloud metrics, previous experiments on measuring elasticity in other fields and past work related to gray-scale algorithms. Section III deals with

the detailed description of the algorithm. Section IV describes the methodology used in measuring results for the elasticity tests. Section V presents the results related to verifying the elasticity of the proposed algorithm and finally section VI concludes the work done and suggests possible future work.

II. Related Work

As far as demonstrating elasticity in any area of concern, one essential aspect is actually understanding this concept and a lot of research exists on defining and benchmarking it and the first subsection of the next section will try to present previous work done in that direction. In the other two sections of the paper, some previous elasticity verifying experiments from other fields will be discussed and finally we will offer an overview of past work done on image gray-scaling algorithms.

A. Elasticity: Definition and Benchmarking Tools

One significant approach in presenting elasticity belongs to Herbst et al. [8]. Their goal is to deliver a concise explanation of the elasticity property and a mathematical characterization of the metrics involved. The authors try to highlight its fundamental properties by providing a comparative analysis to other related performance concepts which are very often mistakenly used as equivalents to elasticity: efficiency and scalability. Another interesting approach of presenting the characteristics of elasticity metrics was delivered by Weber et al in [15]. The authors set off by claiming the need of a new elasticity benchmarking tool and they motivate it by making a clear distinction between different cloud computing performance related terms like efficiency, scalability and throughput. By comparing different elasticity behaviours for different systems, by evaluating past approaches and by redefining elasticity related metrics the authors set an example of how a stable elasticity system should behave. One complex mathematical model for measuring the elasticity of cloud platforms has been proposed by Weinman [16]. A demand function $D(t)$ characterizes the resources needed by the computational system over time and a second function $R(t)$ represents the resource allocation over time. He also defines a loss function L which is computed by summing up all the resources that were not used over a period of time and the demands that were not full filled in the same period of time. A perfect scenario, form an elasticity point of view, would be when $D(t) = R(t)$ which means that the allocated resources exactly meet the demand requirements on all intervals of time and that the loss function is equal to 0. By making use of mathematical concepts, Weinman analyses the loss incurred on various theoretical scenarios which include: linearly, exponentially and randomly increasing and decreasing workload demands.

One of the most popular areas where research for benchmarking cloud performance, in terms of elasticity, scalability and efficiency measurements, has been carried out, is that of the distributed database services. The most famous tool for benchmarking systems, *Yahoo! Cloud Serving Benchmark (YCSB)*, is thoroughly described in [1]. The authors demonstrate its efficiency across various NoSQL database systems by running experimental tests on four very different commercial frameworks: *HBase*, *Casandra*, *PNUTS* and *MySQL*. One experimental process that the authors present is tracking down the performance of these cloud databases while the cluster increases the number of servers used (i.e. proof of elasticity). One clear advantage over other similar benchmarking tools is that new workloads can easily be added as extensions. A popular tool for measuring the elasticity performance of a cloud database is *BenchXtend* [4] which is build upon *YCSB* and which brings additional features like: presenting results from both consumer and provider perspectives, more metrics to the measurement process, increasing and decreasing the number of clients while running a specific workload.

B. Elasticity: Past Verifying Experiments

An interesting measurement example is described in [11] where they succeed in finding a mathematical formula for characterizing the penalty incurred by the difference in over-provisioning (allocate more hardware resources than needed to the system) and under-provisioning (having a larger demand than the allocated resources). In order to prove elasticity of the platform, the theoretical aspects are applied on a set of ten real-world TPC-W workloads which have different demand patterns and which trigger different peaks over time. In one section of the paper [2], the authors present the implementation of a new model of cloud framework build upon *IaaS*, the *MeT*. As part of their experimenting work, they demonstrate the elasticity of their cloud system by presenting comparative results obtained with another system of the same category named *TIRAMOLA*. The experimental set-up initially loads the two systems with five *YCSB* generated workloads. The authors prove that the *MeT* system performs better as it succeeds in keeping a higher throughput and a smaller number of active machines. Unlike our experiment methodology, which makes use of less intensive areas of the algorithm in order to show removal of nodes, in the *MeT* experiment, workloads start to get cut off and the system proves to be more responsive to the decreasing demand by releasing nodes. Another interesting approach of verifying the elasticity of a system was displayed in [9]. They gather performance results from five different workloads executed on a commercial cloud service platform, the *Amazon EC2*. Unlike us, who only use the pixels of a image as data input, they used as input for one of their benchmarking experiment a considerably larger amount of data: the tweets that were gathered from Twitter's storage cloud in three different sized chunks. The goal was to filter out an arbitrary category of these tweets and to observe the fluctuations in execution time, speed-up, efficiency and scalability as the number of instances used got increased.

Another research paper on cloud databases performance [12] aims to evaluate two popular such frameworks: *HBase* and *Casandra*. The authors succeed in replicating previous research results obtained using physical local hardware on *Amazon EC2* platform using the *Yahoo Cloud Serving Benchmark* as a benchmarking tool.

They measure elasticity results for three different workloads types: read-intense, scan-intense, write-intense on each of the two database types. They decide which database performed better by comparing the number of nodes the workloads occupied over the same periods of time. Another comparative approach for measuring elasticity on cloud databases [5] adds to the list of analysed databases the *MongoDB* platform and uses as input for their experiment *Wikipedia* articles. They observe behaviour of three databases by doubling at every step the size of the input and they use their own formula for measuring elasticity.

C. Gray-Scaling Algorithms

In this paper we try to verify the elasticity needs of a image gray-scaling technique proposed by Gooch et al. in [7]. This method implies that a source coloured image will undergo the process of color removal. There are a widely known variety of methods for greying out a image each of them with its advantages and disadvantages. The most obvious problem when it comes to eliminating the color of an image is finding a way to preserve as accurately as possible most of the visual differences from the initial coloured image to a image consisting mainly of combinations of gray pixels. Gooch et al. [7] succeeded in finding a method that became very popular over time and for which we will be implementing a parallel solution. In comparison to other similar algorithms, the one proposed in this paper consists of multiple costly computations and its complexity is one of the highest in this area. The steps of the algorithm will represent the workloads on which we will measure performance results. The variations in the demand patterns of the different steps in the algorithm will provide us with the opportunity of observing different resource allocation from the cloud service on which we run our experiment.

Throughout the time, there have been various other approaches to this kind of image processing technique. The methods implemented can be divided into two major categories: local gray-scale algorithms which take into account how the pixels are distributed locally, and so technique is a spatially dependent one; and global algorithms which apply the same mapping method on all the pixels, independently of their location. One local approach described in [14], is a linearly complex technique based on a psychometric experiment in which they analyse the human eye perception of the spectre of color. The authors claim that the linear method proposed by them has far better performances in terms of both execution time and quality of results than the one discussed in this paper. A more recent local approach to performing gray-scale conversion was discussed in [13]. The method progressively calculates the contribution of the red, green and blue components by analysing the localization of the pixels. This is also a faster technique than the one proposed in this paper and has the additional advantage of being fit for grey-scale transformation of video clips.

Besides the above mentioned sequential techniques, there have also been a few parallel implementations of this image processing method. More precisely, one of the existent parallel solution to the method proposed in this paper is presented in [3]. The authors developed it on a CUDA enabled graphics processing unit (GPU). The experimental results of the authors show a performance boost of up to 288 speed-up. In one of the section of their paper [17], amongst other image and video processing techniques, the authors also analyse the results obtained by applying gray scale conversion to a image using the MapReduce framework implemented on Hadoop.

As to our knowledge, there has not been any previous paper on presenting the "Color2Gray" MPI distributed version of the algorithm of Gooch et. al [7].

III. Algorithm Description

The "Color2Gray" algorithm for converting images to gray-scaled ones [7] consists of four main steps:

1) conversion to $CIE\ L*a*b*$ color space
2) computation of luminance differences
3) calculation of solution to optimization problem
4) conversion to RGB color space

A. Conversion to $CIE\ L*a*b*$ color space

The first step converts the data pixels from the RGB space to the CIE $L*a*b*$ space. In order to perform gray-scale conversion, the authors [7] claim that a color space with little or no-correlation between its axes is a more desirable one and $L*a*b*$ is such a color space. This conversion has an intermediary step, which is converting the RGB color space to the XYZ color space and only after that the XYZ color space to $CIE\ L*a*b*$ color space. The $L*$ component refers to the lightness of the image, such that a value of 0 means black while 100 means white. The $a*$ and $b*$ components hold the colouring information of a pixel: the $a*$ measures red/green and $b*$ represents yellow/blue. This means that a pixel that has both $a*$ and $b*$ components equal to 0 is a gray scaled pixel. The goal of the algorithm is to compute the luminance component of each pixel by making use of the chromatic differences between the $a*$ and $b*$ values. The equations describing the computations performed for the color space conversion can be found at [6]. At this point in the algorithm the image is represented by an array of CIE $L*a*b*$ pixels.

B. Calculate target differences

The next step in the method represents the most intensive computational part of the process. For the purpose of the algorithm description, we denote dL as being the luminance difference between a pair of pixels. Similarly, dA and dB are delta values of the other two components. The difference in chroma between two pixels will be represented by dC and is obtained by applying the Euclidean norm:

$$dC = \sqrt[2]{dA^2 + dB^2}$$

The luminance differences for every pair of pixels are computed by making use of three parameters that the user can vary and which are given as input to the program. The first one, the β is the one who influences the direction towards a darker or a brighter luminance by taking as input the chromatic difference. For example, you can choose to suggest the difference between a blue color spot and an orange one by making the blue brighter and the yellow darker or vice versa. This parameter is basically and angle which gives the sign of the chromatic difference and is calculated as follows:

$$v_\theta = cos(\theta) * dA + sin(\theta) * dB$$

The second parameter, the α parameter, dictates how much of the chromatic differences, represented by the $a*$ and $b*$ axes will be reflected in the final luminance differences. The range of values for a and b axes are [-200, 200] and [-500, 500] respectively, while the range for the luminance is [-100, 100]. In order to achieve these two steps described above, a crunch functions is introduced which makes use of the $\tanh$ function to the range and multiplies it with the alpha parameter in order to increase or decrease the impact of the chromatic difference, which is basically the parameter of this function.

$$crunch(x) = \begin{cases} 0, & \text{if } x = 0 \\ \alpha * \tanh(x/\alpha), & \text{otherwise} \end{cases}$$

The third parameter is the one that dictates the neighbourhood size. The locality of the Color2Gray algorithm is based on this radius parameter which basically tells how many pixel will be influencing the value being computed. The smallest the radius is, the fastest the algorithm is and the lower the quality obtained. This radius parameter is actually the one that makes this algorithm fit for parallelism. Basically, every computing machine has its own chunk of pixels from the initial image and the computing radius is equal to the full radius of the image divided by the number of nodes involved in the calculations. Given all these parameters the output at the end of the second step that represents the chromatic difference between two pixels, denoted as δ_{ij} is:

$$\delta_{ij} = \begin{cases} dL, & \text{if } fabs(dL) > crunch(dC) \\ crunch(dC), & \text{if } v_\theta * crunch(dC) > 0 \\ -crunch(dC), & \text{otherwise} \end{cases}$$

C. Resolve optimization problem

The next step after computing the luminance differences between pairs of target pixels, is to apply the difference on the source image, more exactly on its luminance component only. The chromatic components are equal to 0 in a gray-scale image and they were used only to find a way to express the variations in lightness of the final image in a way that preserves as much as possible the colouring dynamic of the initial image.

D. Conversion to RGB color space

The final step is to convert the image from the CIE $L * a * b*$ color space back to RGB space using the inverse equations which are presented in [6]. The inverse conversion algorithm is symmetric with the direct one, and consists also of an intermediary conversion to $CIE\ XYZ$ color space.

The overall performance of the algorithm is dependent upon the size of the radius and is equal to $O(\mu^2 * N^2)$, for a given μ radius. As far as the distributed implementation presented in this paper, we use a full radius size on each of the processing unit involved, the overall performance on each one of them is $O(n^4)$ only for computing the luminance differences. An additional factor of $3 * O(n^2)$ comes from converting the image to $L * a * b*$, back from $L * a * b*$ to RGB and matching the differences in luminance to the luminance component of the initial image.

The MPI version proposed in the article was obtained by giving every processing unit a chunk of the overall height of the image. If the height is not divisible to the size of the MPI world, the last process will be given an additional number of rows to process. At the end of each computation step, the root process will call the gathering function on all other process and will output the intermediary image to the disk.

IV. Experimental Methodology

The program we implemented in order to perform our elasticity tests is a distributed version of the sequential algorithm described above. It receives as input three parameters: the α and β which were defined in the previous sections, and the image that is to be transformed to monochromatic style. The executable offers support only for BMP pictures. The code executes on a $C++/MPI$ environment and does not use any additional external libraries or any additional compilation flags.

The experiment of validating the need for elasticity of the Gooch algorithm was conducted on multiple Intel Nehalem micro-architectures that are hosted on a public cluster. They are equipped with 24 Xeon processors which operate at 3.0 GHz CPU frequency. Every processor has 6 cores sharing a cache memory of 12 MB.

We divided our program into 4 different steps as per the above description and we measured the execution time on every step on an increasing number of machines. The maximum number of machines used was ??. For every computational part of the algorithm, we calculated the speed-ups by dividing the execution time obtained by running the program on 1 machine at the execution time obtained by running the workload in parallel. We analysed at every step if the further addition of multiple machines is relevant. In order to compare results on different sized workloads, we performed the above tests on four different image sizes: ... During our experimental tests we measured the CPU, RAM, Disk and Network usage. At the end of our experiment we obtained for every step we measure the number of machines needed so that the resource provisioning is balanced (we try to avoid over-provisioning and under-provisioning by measuring the speed-up and detect where the

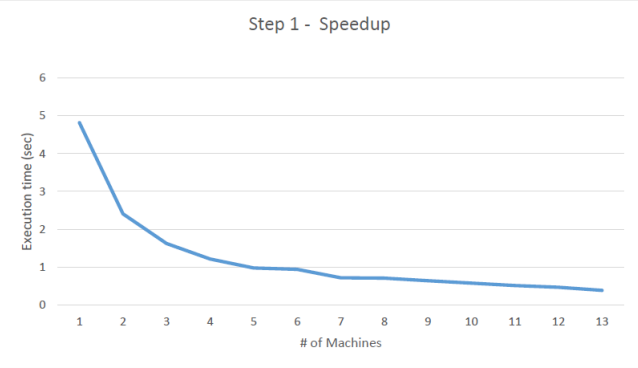


Fig. 1. Speedup variation for first step of Color2Gray

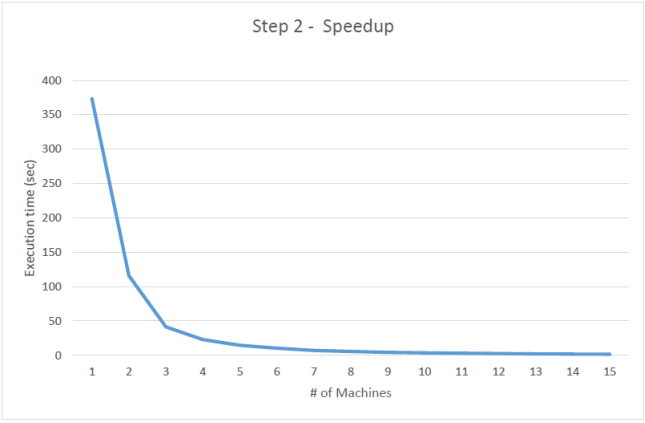


Fig. 2. Speedup variation for second step of Color2Gray

addition of hardware is unavailing), thus proving that different demand patterns require different resource allocation needs.

V. Experimental Results

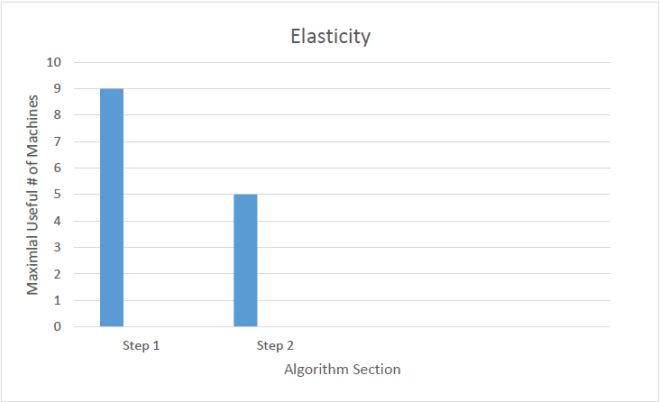
In order to obtain the speed-up for every section of the algorithm, we submitted experimental jobs on the cluster by increasing the number of machines on which to distribute the executable and we recorded the execution time with the MPI_Wtime function for each of the sections analysed. When measuring the elapsed time, we did not take into account the I/O operations which deal with reading and writing the image as well as writing log files to the disk.

Figure 1 shows a significant speed-up for the first 3 to 4 number of tasks. When starting to run with a number of tasks greater than 5 the speed-up begins to decline. We can observe that the execution time continues to decrease in a notable way for up to 8-9 tasks.

The variation for the second step of the algorithm 2 shows in a similar way as the first step, a very abrupt decrease in the execution time for the first part of the experiment. This time the speed-up settles more rapidly, as we can notice that at after running with more than 4 jobs, the over provisioning with hardware resources is not worth the cost.

In Figure 3 we provide a chart which suggests the optimal number of machines that should be allocated to the various steps of the algorithm such that the over-provisioning and under-provisioning be as near to 0 as possible.

Fig. 3. Verifying elasticity for Color2Gray



VI. Conclusions and Future Work

Throughout this paper, we focused our attention on the elasticity needs of grey-scale conversion algorithm. We started by motivating our work through previous work done on defining and benchmarking elasticity on other platforms and research fields. We motivated our choice of using a grey-scale algorithm by presenting each of the four computational steps which allow for parallelism and scalability. In the end, we have succeeded in presenting the results that show the elasticity needs of the costly computational distributed algorithm that performs grey-scale conversion on BMP images. The elastic behaviour demonstrated by this specific algorithm comes as a further motivation that good cloud provisioning algorithms are needed in order for the cloud service to be appealing to consumers. Apart from the elasticity proof, another indirect contribution of this paper is to certify the feasibility of a distributed implementation of the $Color2Gray$ algorithm proposed by Gooch et al [7] that proves the scalability behaviour of this intensively mathematical grey-scale algorithm.

There are some improvements that our current methodology could benefit from.....

References

[1] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[2] Francisco Cruz, Francisco Maia, Miguel Matos, Rui Oliveira, João Paulo, José Pereira, and Ricardo Vilaça. Met: workload aware elasticity for nosql. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 183–196. ACM, 2013.

[3] Wendell Fioravante da Silva Diniz, Eurípedes Guilherme de Agnus Azevedo Horta, Oliveira Nóbrega, and Luiz Otávio Saraiva Ferreira. Parallel implementation of grayscale conversion in graphics hardware.

[4] Rodrigo Felix de Almeida and Javam C Machado. Benchxtend: a tool to benchmark and measure elasticity of cloud databases. 2012.

[5] Thibault Dory, Boris Mejías, Peter Van Roy, and Nam-Luc Tran. Comparative elasticity and scalability measurements of cloud databases. In *Proc of the 2nd ACM symposium on cloud computing (SoCC)*, volume 11, 2011.

[6] Adrian Ford and Alan Roberts. Colour space conversions. *Westminster University, London*, 1998:1–31, 1998.

[7] Amy A Gooch, Sven C Olsen, Jack Tumblin, and Bruce Gooch. Color2gray: salience-preserving color removal. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 634–639. ACM, 2005.

[8] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, pages 23–27, 2013.

[9] Kai Hwang, Xiaoqing Bai, Yue Shi, Meng Li, Weijie Chen, and Yaowu Wu. Cloud performance modeling and benchmark evaluation of elastic scaling strategies. 2015.

[10] Shadi Ibrahim, Bingsheng He, and Hai Jin. Towards pay-as-you-consume cloud computing. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 370–377. IEEE, 2011.

[11] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 85–96. ACM, 2012.

[12] Jörn Kuhlenkamp, Markus Klems, and Oliver Röss. Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment*, 7(12):1219–1230, 2014.

[13] Wei Hong Lim and Nor Ashidi Mat Isa. A novel adaptive color to grayscale conversion algorithm for digital images. *Scientific Research and Essays*, 7(30):2718–2730, 2012.

[14] Laszlo Neumann, M Čadík, and Antal Nemcsics. An efficient perception-based adaptive color to gray transformation. In *Proceedings of the Third Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 73–80. Eurographics Association, 2007.

[15] Andreas Weber, Nikolas Herbst, Henning Groenda, and Samuel Kounev. Towards a resource elasticity benchmark for cloud environments. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud service Scalability*, page 5. ACM, 2014.

[16] Joe Weinman. Time is money: the value of "on-demand", 2011.

[17] Muneto Yamamoto and Kunihiko Kaneko. Parallel image database processing with mapreduce and performance evaluation in pseudo distributed mode. *International Journal of Electronic Commerce Studies*, 3(2):211–228, 2013.