2019 IEEE International Conference on Autonomic Computing (ICAC)

Quality-Elasticity: Improved resource utilization, throughput, and response times via adjusting output quality to current operating conditions

Lars Larsson*, William Tärneberg†, Cristian Klein*, and Erik Elmroth*
* Department of Computing Science, Umeå University, Sweden
Email: {larsson,cklein,elmroth}@cs.umu.e
† Department of Electrical and Information Technology, Lund University, Sweden
Email: william.tarneberg@eit.lth.se

***Abstract*—This work addresses two related problems for online services, namely poor resource utilization during regular operating conditions, and low throughput, long response times, or poor performance under periods of high system load. To address these problems, we introduce our notion of quality-elasticity as a manner of dynamically adapting response qualities from software services along a fine-grained spectrum. When resources are abundant, response quality can be increased, and when resources are scarce, responses are delivered at a lower quality to prioritize throughput and response times. We present an example of how a complex online shopping site can be made quality-elastic. Experiments show that, compared to state of the art, improvements in throughput (57% more served queries), lowered response times (8 time reduction for 95th percentile responses), and an estimated 40% profitability increase can be made using our quality-elastic approach. When resources are abundant, our approach may achieve upwards of twice as high resource utilization as prior work in this field.**

***Index Terms*—cloud computing, service delivery, adaptive software, brownout**

I. Introduction

We address the problem of poor service throughput and resource utilization by varying response output quality based on current operating conditions to provide better end-user utility. Unlike prior work [1], our approach encompasses *not only quality* ***reductions*** *but also* ***increases*** along a **fine-grained spectrum** without an explicit set-point. That work focused only on quality reductions as a binary decision, to either respond with regular quality responses or to degrade them in order to expedite processing.

We expand on the problem formulation in Section II and present our case of how the current state of affairs is wasteful for both service and cloud infrastructure providers. Related work in Section III shows that the main contemporary way of addressing the problem is to either *degrade response output quality* to make processing simpler or to *scale up the underlying cloud infrastructure*. We argue that neither is a satisfying solution to the problem, and compromises poorly on output quality (failing to use available resources) while also failing to address the utilization and throughput problem properly (by e.g. letting requests time out). Instead, **we propose a quality-elastic solution** in Section IV, including a running example in the "online shopping site" domain targeted by prior work [1]. We show how output quality can be modified along a fine-grained spectrum rather than as a binary decision whether to show (optional) product recommendations or not. We vary output quality by choosing different algorithms for generating recommendations, described in Section IV-A. Which algorithm to choose is determined by taking current conditions into account, using thresholds on current resource utilization. Our experiments in Section V show that: (a) in a fair apples-to-apples comparison, making a binary choice whether to serve optional content based on utilization thresholds, we achieve **50% higher throughput**, **8 times lower 95th percentile response time**, and an estimated profitability increase by 20%; (b) when using different output quality levels, performance in terms of throughput and response time is similar, but improved ability to serve product recommendations **increases profitability by 40%**; and (c) an **upward of 100% higher resource utilization** in times of resource abundance, compared to prior work [1].

We conclude that these results motivate further research, of which some directions are given in Section VI. Our contributions in this paper are as follows:

- we define and explore the concept of quality-elasticity and argue that it is required to optimize service delivery with regard to objectives of both infrastructure and service providers; and
- we conduct experiments comparing our operational condition-aware quality-elastic approach with state of the art, demonstrating vast improvements in terms of lowered response times, higher throughput, higher profitability, and improved resource utilization.

This work has been funded by the Swedish Government's strategic effort eSSENCE, the Swedish Research Council (VR) under contract number C0590801 (Cloud Control), the Excellence Center Linköping—Lund in Information Technology (ELLIIT), the SEC4FACTORY project, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

2474-0756/19/$31.00 ©2019 IEEE
DOI 10.1109/ICAC.2019.00017

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

II. Problem formulation

Online software services are predominantly designed to produce responses of a fixed quality level to incoming requests. They are (intentionally?) oblivious to (a) the properties of their execution environment (i.e. CPU core count, speed, and amount of RAM), and (b) current operating conditions, such as current resource contention. We regard this as the core problem: that *services consistently produce results of the same quality, without regard to either current operating conditions* or to whether the output quality matches consumer expectations of what the service should do, i.e. the service utility.

Ability to serve requests is heavily impeded when resources are scarce. Cases of immediate resource scarcity are signified by high resource contention in already overloaded execution environments. Because applications are increasingly deployed as distributed systems, performance fluctuations in one component can cause large ripple effects [2]. There is also no practical way of instantly scaling up the underlying infrastructure, even when using e.g. a platform such as AWS Lambda [3]. Requests are served either slowly, or, due to client timeouts, perhaps not even at all. Thus, service utility, resilience, and even throughput suffer.

In cases of resource abundance, the problems are two-fold. Firstly, execution environment resources that have been paid for (e.g. virtual machines or containers of fixed sizes) are poorly utilized, leading to higher operational cost for both service and cloud infrastructure providers. Secondly, not using resources also implies a missed opportunity to potentially provide higher quality results at no additional expense.

For service providers, the impact is palpable. Online software services are an important part of the global economy, and consumer satisfaction depends primarily on how useful services are perceived to be. For example, availability, resilience, and throughput contribute to the experienced performance. Amazon reportedly determined that a 100ms latency increase costs them 1% in lost sales.

Poor compute resource utilization is costly for both service and cloud infrastructure providers, with average utilization reaching only around 15% [4]. Hence, resources are wasted to essentially keep the lights on [5], [6], to the tune of ability to host 40% more servers with the same power budget if properly optimized [7]. Cloud providers are also under market pressure to provide seemingly infinite resources to service providers, which requires large buffers of available resources to deal with request peaks. This is costly, and constitutes a barrier to entry for smaller providers. Thus, both service providers and cloud infrastructure providers have clear incentives to make better use of available resources.

Eventual consistency, explored at length in [8], may itself be regarded as a form of quality-elasticity for databases[1]. This mode of operation and the constantly evolving models used by modern services such as Google and Amazon have taught users to not expect consistent results to imprecise queries: what products are recommended changes based on factors and proprietary algorithms that users cannot inspect. Thus, they cannot judge whether these results are actually accurately modelled to their particular preference, instead just trusting that they have been generated "somehow". This non-strictness is opens possibilities for optimization. For service delivery, we can use this to our advantage, as intuitively, *some* result (of some quality) is better than *none*.

[1]A scheme wherein quick and hopefully mostly-correct responses are delivered in response to queries, instead of using additional time and resources to ensure that the freshest possible data is used.

III. Related Work

The main source of inspiration for our work is the pioneering work of Klein et al. [1]. It reformulated the concept of brownout for cloud applications. The term brownout originates from electrical power distribution, where in case of low power output, overall output voltages are lowered to give subscribers at least some power, rather than none, as would be the case of a full blackout. In the cloud computing case, it was instead used to *not* serve some aspects of a response (content marked as optional) in case of resource scarcity. The running example in their work, and the work that has followed, was that of an online store, where recommendations for related products are optional when a user views a product detail page. The intuition is that the most important pieces of information, such as product image, description, price, and whether the product is is stock or not, are more crucial than determining which other products might be related to the current one. In contrast to our example of a wide variety of different response output qualities in Section IV-A, their approach is thus to either include (default) or drop product recommendations entirely.

Various algorithms and controllers have been employed to extend the brownout-concept as originally envisioned, including how it can be used to optimize both service delivery [9], [10], [11], [12], [13] and cloud infrastructure power consumption [14], [15], [16]. These results validate the claim that great utility and efficiency increases can be made using a quality-*reduction* approach.

Application brownout has hitherto been formulated to employ binary output quality *reductions*: either include an *optional* part or not. The goal of application brownout has been to smoothly *handle capacity shortages* instantly. Our position is that output quality should be truly **elastic**: possible to *reduce or increase* as current conditions dictate. We also propose that output quality be adjusted on a **fine-grained spectrum** without an explicit setpoint, rather than as a binary decision.

Brownout has not gained widespread adoption outside the research domain since its inception in 2014. A much more common alternative is scaling the underlying virtual cloud infrastructure in response to fluctuations in demand. Scaling can be either in the size of software execution environments handling ongoing requests (vertical scaling) or their number (horizontal scaling). Vertical scaling may cause service disruption. In the case of VMs, resizing them may cause delays [17]. More severe delays can be incurred if migration is required. In the case of containers orchestrated by e.g. Kubernetes, other Pods of containers may be adversely affected as they are

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

evacuated to make space for the one that shall increase in capacity. Horizontal scaling, on the other hand, is typically rather slow. New VMs can take several minutes to become fully operational[2]. Starting additional containers in a cluster with abundant resources is faster [18], but such fast speed depends on surplus resources during normal operating conditions. If the cluster is near or beyond capacity, and needs to be resized by adding more virtual machines to it, scaling time is obviously on parity with virtual machine scaling. Also, as evident from both experience and via literature studies, both types of scaling embody a highly simplified approach to dealing with load peaks. Cloud resources are subject to significant variations in delivered performance [19]. It is, therefore, non-trivial to assess what the effects of scaling will be, ahead of time.

It is of particular value to mention that both types of scaling may also disrupt stateful applications [20], as they may require state to be replicated and clustering algorithms to handle new members. Because of these disruptions, capacity scarcity is actually worsened as a result of cluster size increases during state replication periods and until the new cluster size starts to contribute positively to overall service performance. Because of these issues, the most common approach is to tolerate temporary resource starvation to establish that a resource increase was not a spike, but rather, a sustained increase.

Our proposed quality-elastic approach offers an attractive complementary path and alternative. Since it adapts to current operating conditions immediately, services need not suffer starvation. Instead, they adjust to emit lower-quality responses during load spike conditions. If these turn out to be the beginning of a new higher usage trend, scaling can be used to more confidently scale to match demand. It therefore operates on not only a completely different time scale, as quality adjustments can be done even mid-processing of requests, but on the correct level to address the core problem of poor matching to utilization (of resources) and utility (to customers).

Suresh et al. targeted the problem of collaboratively meeting deadlines in a service-oriented architecture [21]. Their approach hinges on rate limiting and scheduling on a per-service basis to optimize overall ability to meet deadlines. Similarly, Wang et al. determine three heuristics for dealing with large response time fluctuations in n-tier systems, the first two of which are precisely related to scheduling to maximize throughput and rate limiting (the third is to increase buffers to avoid repeated requests) [2]. Neither approach does, however, modify the output quality of the services. If a service is at risk of not meeting a deadline, scheduling the most at-risk requests (Suresh et al.) or lightest transactions (Wang et al.) for priority processing helps meet overall system performance targets, rather than employing output quality adjustments. Such a scheduling-assisted approach should be possible to use together with ours and help achieve further improved results. Quality-elastic variations can then help impact the choice of which request to schedule and thus service first, not just its arrival time into the system.

[2]The largest public cloud provider, AWS, still in 2018 on their FAQ state that an on-demand VM may take up to 10 minutes to become operational.

Leveraging approximations carefully can help trade output quality for higher throughput. By reducing the input set via intelligently sampling massive input material and thus working only on statistically determined *representative* samples, large bodies of recent work has shown great performance increases, and thereby increased ability to handle significantly larger input sets [22], [23], [24]. By also allowing users to provide approximating versions of their tasks, and/or dropping entire tasks, Goiri et al. showed great improvement in MapReduce efficiency. Their results show up to 32 times run time reduction when users can tolerate an error of 1% with 95% confidence [25]. The authors claim that such an approach is useful for a large variety of use cases involving statistical data processing, e.g. data analytics, machine learning, and media processing. They did not study online/interactive work, focusing instead on offline data processing.

Bridging the gap between approximations for offline and online processing, Kelley et al. presented Ubora, a system that learns which components respond slowly and transparently circumvents them from slowing down the system as a whole [26]. This is done by duplicating some incoming requests: the first elides data from slow responses, the second is allowed to take a long time, and is in return expected to provide a high-quality response. This response is memoized, and can be used later to provide overall higher quality responses while still keeping throughput high, processing 37% more queries than a competing controller guided by the rate of timeouts [26]. The approach of Kelley et al. works on queries that yield "mature answers", i.e. ones that result from a full execution without time constraints. Such answers are memoized at the cost of increased bandwidth and memory in order to reduce repeated computation and yield faster results in the future. The utility of such an approach is therefore highly dependent on how often identical queries are repeated. The notion of quality is in the work of Kelley et al. a measure of what percentage of requests get mature answers as response. Requests that cannot get mature answers are timed out instead. Our view is that a system is more robust, as a whole, if all requests get some response instead of none.

Our quality-elastic concept operates on a level of abstraction akin to that of a scheduler, although currently, without the algorithmic sophistication expected by one. It can choose whether to drop work (as Goiri et al.), to use an applicable cached response (as Kelley et al.), or even to do more work than usual, based on current resource and time availability. As such, it makes assumptions neither about online or offline data processing, nor does it primarily provide benefit when identical queries are repeated. Crucially, the quality-elastic approach aims at giving a system a way to instantaneously adapt to current operating conditions on a per-request basis. Thus, it makes no claims about how future requests are handled, as operating conditions will be different.

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

IV. Quality Elasticity

We define quality-elasticity as letting software services adapt their mode of operation to current operating conditions by providing results of varying output quality. Adaptions are based both on *basic properties* of their execution environments[3], such as opting for a more memory-intensive algorithm when memory is more readily available than CPU time, and on *current conditions* such as system load and instantaneous contention effects by other execution environments. Crucially, they may provide **lower-quality results in cases of resource scarcity**, and **higher-quality ones when resources are abundant**.

For service providers, the implication is that service instances can more predictably handle load spikes and avoid client timeouts by reducing output quality of responses. This instantly lessens load and thus helps achieve higher throughput. As stated in Section II, scaling the underlying infrastructure is both slow and may adversely affect performance, due to adjusting clusters of (stateful) components. Note that while scaling up the most trivial of stateless services has gotten substantially faster as we have progressed beyond VMs and containers to functions in FaaS platforms, these neither offer instant performance issue remediation, nor do they constitute a large fraction of deployed services, due to the large amount of still-deployed legacy services and that the Function-as-a-Service paradigm is still in its relative infancy (AWS Lambda was introduced late 2014).

For infrastructure providers, higher utilization and an improved ability to adapt deployed services to resource contention means that aforementioned additional readily available spare capacity can be reduced. Output quality *reductions*, a subset of quality-elasticity, have been shown to optimize cloud infrastructure in this way [15]. Thus, from both perspectives, cost reductions can be made in addition to more robustly dealing with software execution in non-deterministic environments.

Current trends in service delivery point to a higher level of responsibility being shifted to the cloud infrastructure provider. Container- and Function-as-a-Service both alleviate service providers from operational responsibilities, and enable infrastructure providers to optimize according to whatever criteria they see fit. This includes making decisions that greatly impact delivered service performance [27]. We therefore see it as at least *feasible* that quality-elasticity could become a way to deliver service in the future. The current alternative, which is to just let service performance and throughput suffer, makes deployment of e.g. mission-critical software impossible [28].

We assume that services are request/response based, as common in contemporary web-based services. When a request arrives it is either queued at the server or, if the queue is empty, the server serves it directly. Next, the server needs to decide *how* to serve it, based on some metadata (as is done today) or by inspecting the current operating conditions (as suggested in this paper). Once the server starting working on a request, it can no longer change how it is served, as part of the work required to serving the request may depend on other services, as well.

A. Running Example

Assume that we wish to employ quality-elasticity to an online shopping site. The most crucial part of a product page is listing the details about the product itself. Recommendations for similar products are common, and help increase sales [29], [30]. Producing these recommendations can be done in many ways, with large disparity in how computationally intensive they are and what output quality they offer. Some of these quality-elastic approaches are:

1) `drop`: drop the recommendations altogether (as in prior related work [1]).
2) `cached-prior`: use a previously cached recommendation set, calculated at a time of higher resource availability, regardless of what it may be.
3) `equal-tags`: query a database for products with the same tags as the current product of focus. This is assumed to be fast, if tags are indexed in the database.
4) `similar-general`: query a database for similar products often bought together by customers in general.
5) `similar-specific`: query a database for products often bought together by customers similar to the one visiting the page, as based on particular previous purchases.
6) `individual-model`: running a machine learning model to establish what products the current customer may be interested in, based on all data available at the time.

The further down the list we go, output quality demonstrably increases, as the recommendations are more finely tuned to the particular product and user. We claim that such a more fine-grained output quality spectrum approach can be used to achieve the optimal per-user utility, balanced against resource usage. It is well-known that product recommendations increase sales, and the quality of such recommendations matter greatly [29], [30].

During normal operating conditions, it is feasible that either `equal-tags` (Approach 3) (as in the Sock Shop demo application[4]) or `similar-general` (Approach 4) may be chosen, as they may provide good enough output quality. However, if resources are available in abundance, the online shopping site may opt to use one of the latter ones to A/B test whether providing higher quality results cause customers (of a particular set) to buy more products.

B. Implementation

To implement and experiment with our proposed approach, we have modified the simulator used to obtain results in [1]. Thus, we are able to highlight differences between our work and theirs, while still keeping the results comparable. As previously stated, Brownout decides whether to serve "optional content", i.e. product recommendations in the running example, using a control theoretical approach to regulate *response times*.

[3]Such as a VM, container, or language runtime.

[4]https://github.com/microservices-demo/microservices-demo

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

We modified the simulator in two ways. Firstly, we added algorithms that consider *current resource utilization* instead of response times. These otherwise behave as Brownout and choose in a binary fashion whether to add or drop optional content altogether. These operate using a threshold, and serve content only if current resource utilization is below a configured level. Serving optional content takes the same amount of simulated work, regardless of which approach is chosen. In control theoretic terms, the controller and sensor are different, but the actuator is the same.

Secondly, we added the concept of multiple algorithms for creating optional content. The algorithms are the ones listed in Section IV-A above. By introducing these to the simulator, we can let a simple utilization threshold-based algorithm determine whether which output quality level to use for the response to a given request.

The simulator lets a configurable number of simulated clients (e.g. 200) make requests for a given length of time (100 seconds). In a closed-loop manner [31], the simulated clients "think" for a Poisson-distributed number of seconds with a configurable average (1.0) before making a new request. All these requests are timed and logged. A simulated server decides whether to respond with optional content or not. The difference in processing time is around an order of magnitude (cf. Table II, `drop` and `similar-general` algorithms). This difference in processing (that is, not counting network delay) is realistic, as we assume that responses without optional content can be cached heavily and that those that require optional content are dynamically generated for each request. The simulator also logs whether the response included optional content or not and at what output quality level.

Because simulated closed-loop clients need to "think" before making a new request (similar to how a person would need to wait for a page to load before choosing where to navigate next), lower response time immediately relates to throughput, as that makes it possible for a client to sooner make a new request.

V. Comparison to Brownout

In this section, we evaluate whether a spectrum of different output qualities, chosen based on current operating conditions, can simultaneously minimize response times, maximize throughput, and increase profitability in times of resource scarcity **and** whether it can increase utilization to produce higher quality responses in case of resource abundance.

To do so, we have devised three experiments that build upon each other. First, we verify whether basing output quality decisions on current operating conditions via simple thresholds yields reasonable results in terms of throughput, response times, and service profitability, compared to prior work. In the first experiment, the output quality choice is binary: with or without optional quality-enhancing content (product recommendations). Second, we introduce a broader spectrum of output quality levels to see if throughput, response times, and service profitability can be improved further by making not a binary choice, but one from a set of choices. This uses the quality-elastic approach, as was described in the previous section. Third, we see how the different approaches handle resource abundance, as increased utilization in such cases is desirable when computational resources have already been paid for.

A. Utilization Thresholds

The first experiment is concerned with the effects on response times, throughput, and profitability of responses by leveraging a resource utilization threshold approach, rather than one that aims to adaptively regulate response times as in prior work [1]. Profitability is in this experiment calculated as in [1] by employing the results in [29], i.e. that responses that include product recommendations (optional content) generate 50% more profit than those that do not. This comparison is therefore intended to be as fair as possible to original *Brownout* [1] in an apples-to-apples way.

Table I shows the results with regard to served requests, the fraction of requests for which optional content was served, as well as the profitability of various approaches. The reader is asked to regard only the top part of the table in this section, as the *Brownout-tuned* and *q-e* approaches are discussed further in Section V-A1 and Section V-B, respectively. The figures in this section should be treated similarly.

Table I includes resource utilization thresholds of 10% up to 90% in steps of 10, as well as 95% (marked *ut=x*, where *x* is the threshold level). In addition to these, we have also included the *never* and *always* approaches, which are essentially resource utilization thresholds of 0% and 100%, respectively. That is, they will only choose to serve optional content if current utilization is less than or equal to the configured level. Similar to *Brownout*, their choice is binary: to serve or not to serve optional content, in the form of product recommendations for an online shopping site. In the terms defined in the previous section, the recommendation algorithm used is `similar-general`, as in the original *Brownout* paper [1].

To make comparisons easier, and to choose a realistic baseline, Table I shows the profitability factor compared to the *always* approach of each of the other approaches. This is a reasonable baseline, since contemporary services typically operate in this fashion (i.e. always serving recommendations, regardless of current operating conditions).

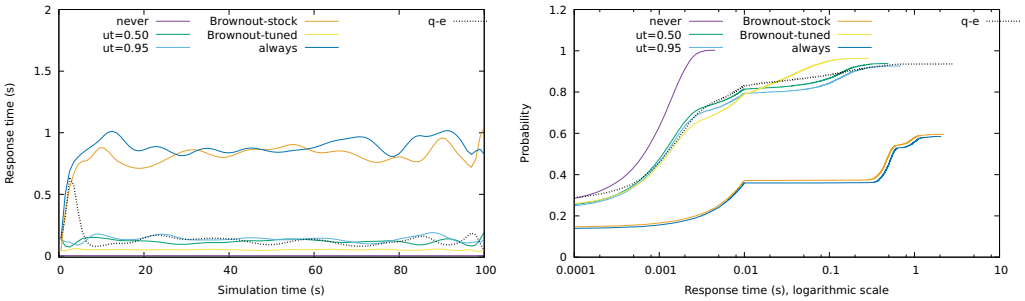
A perhaps unexpected result is that resource utilization thresholds of 50–80% all behave in the same way. However, given that their choice of product recommendation algorithm is limited to just `drop` or `similar-general`, this behavior is perfectly understandable. If processing using *similar-general* would push resource usage to e.g. 85% for a short while, then the approaches using these resource utilization thresholds *should* behave similarly. For compactness and graph readability, we choose the *ut=0.50* and *ut=0.95* resource utilization threshold approaches as examples for further study.

Table I shows that the stock *Brownout* approach serves optional content for almost every request (99% of all served requests). Thus, more time is spent per request, as shown

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

Table I: Request breakdown and profitability. Most utilization threshold-based approaches outperform stock Brownout by about 50% in terms of total served requests, but fail to serve optional content often, impeding potential profitability. Never serving optional content results in the highest throughput, but the highest profitability is achieved by the *q-e* approach, which serves both a large number of requests in total and with optional content. How *Brownout* was tuned is explained in Section V-A1. See Section V-B for discussion about the profitability model used for the *q-e* approach, as it differs from the others (to be more conservative).

	Served requests	**With optional content**	**(fraction)**	**Profitability**	**(factor)**
never (ut=0.00)	20053	0	0.0	20 053	1.14
ut=0.10	20038	69	0.03	20 072.5	1.14
ut=0.20	19193	4238	0.22	21 312.0	1.21
ut=0.30	19074	5016	0.26	21 582.0	1.23
ut=0.40	18891	6250	0.33	22 016.0	1.25
ut=0.50	18757	6567	0.35	22 040.5	1.26
ut=0.60	18757	6567	0.35	22 040.5	1.26
ut=0.70	18757	6567	0.35	22 040.5	1.26
ut=0.80	18757	6567	0.35	22 040.5	1.26
ut=0.90	18867	6636	0.35	22 185.0	1.26
ut=0.95	18529	6908	0.37	21 983.0	1.25
always (ut=1.00)	11699	11 699	1.0	17 548.5	1.00
Brownout-stock	11904	11 836	0.99	17 822.0	1.02
Brownout-tuned	19277	9186	0.48	23 870.0	1.36
q-e	18732	14 548	0.77	25 092.1	1.43



(a) 95th percentile response times.

(b) Empirical CDF plot of response times.

Figure 1: Response times for the various approaches. Both the utilization threshold-based approaches (*ut=0.50* and *0.95*) achieve about 8 times lower 95th percentile response times compared to stock *Brownout*. The eCDF confirms that response times are significantly lower on average, as well. The theoretical minimum (*never*) and maximum (*always*) are also shown for reference.

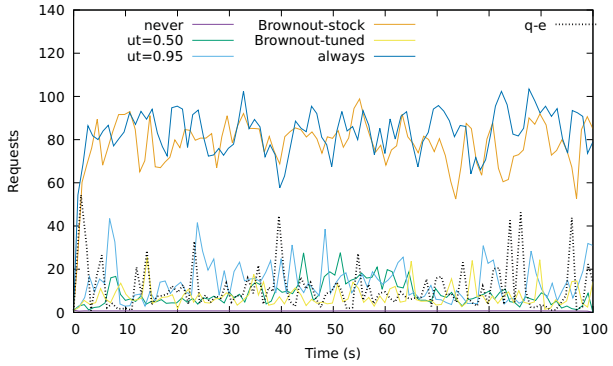


Figure 2: Active ongoing requests. If a request cannot be fully served during a time slot, it is put in a queue. Shown here is queue length at given points in time during the simulation.

in Figure 1. In turn, this also means that more requests are queued up (Figure 2), as the server becomes overloaded. This makes the server unable to keep up with demand, which further exaggerates the problem as more requests come in.

In contrast, both the *ut=0.50* and *ut=0.95* approaches achieve significantly higher throughput (Table I), lower response times (Figure 1), keep their queues less full (Figure 2), and can therefore serve more than 55% additional requests during the simulation period (Table I).

The two extremes, *never* and *always* show the theoretical lower and upper bounds of policies related to never or always serving optional content.

Profitability comparisons show that the resource utilization threshold approaches do better than original *Brownout*, by more than 20% for both cases (Table I). In spite of serving fewer total requests than the *never* approach, both *ut=50* and *ut=0.95* are more profitable by the chosen profitability model, due to the 50% increase in profitability for requests served with optional

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

content. They are conservative in doing so (doing so only in about 35% of cases), as each such response is more resource-intensive to generate. Note that this profitability increase is actually a side-effect, as utilization thresholds are not explicitly designed to maximize profitability.

Interestingly, *always* serving optional content *is the least profitable*, and yet this is the approach that contemporary services take.

The results of these simple experiments show that we by merely considering utilization may be able to get improved results compared to the stock *Brownout* approach as presented in [1]. The comparison has been made as fair as possible, pitting the approaches against each other on *Brownout*'s own terms.

1) Tuning Brownout for this use case : From Table I, it is clear that the stock configuration of *Brownout* does not perform well for this particular use case, where there is a large number of simultaneous clients. Indeed, it instead behaves similarly to the *always* approach, the worst choice.

The stated goal of *Brownout* is to keep response times acceptably low, and it uses control theory to do so. The *setpoint* for the controller is the acceptable time to wait for a response, expressed in seconds, and the goal is to keep the response times below that number. The signal that the controller uses as input is the 95th percentile response time, which by definition is a very noisy signal. The point, however, is that if objectively difficult cases like controlling the 95th percentile response times can be controlled within reasonable bounds, the simpler cases are also covered. The *pole* represents a value trading off responsiveness (fast reaction to—possibly incorrectly measured—disturbances) and safety (conviction that actions taken will not negatively impact the system). To compensate for the noise, stock *Brownout* is configured with a setpoint of 1 second, and a pole value of 0.9. For web browsing, 1 second is indeed a reasonable upper bound site visitor patience [32].

However, a simple sensitivity analysis with setpoints in $[0.05, 1.0]$ and poles in $[0.1, 0.9]$ on profitability yields the results shown in Figure 3 as well as a local maxima at setpoint=0.05 and pole=0.7. These values are used for *Brownout-tuned* in the tables and figures of this chapter.

With these parameters, *Brownout-tuned* performs much better on this particular use case, and is therefore a better adversary for comparison with the *q-e* approach.

B. Quality-Elasticity

The previous section shows that using utilization thresholds is a promising alternative to stock *Brownout*. However, the comparison was made on *Brownout*'s terms with a binary choice dictating whether to serve optional content or not. We now turn our attention to the main contribution of the paper, namely offering a broader **spectrum of quality adjustments** in a true quality-elastic fashion. Thus, there are multiple output quality levels, and we use utilization thresholds to determine which one to target (quality levels are described in Section IV-A).

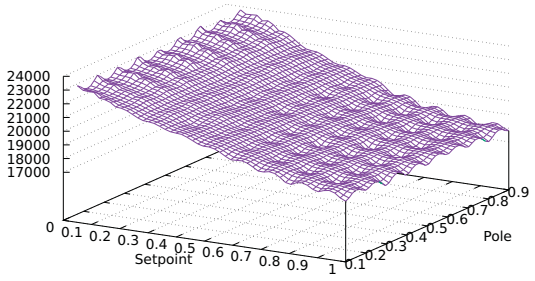


Figure 3: Profitability as a function of Brownout parameters *setpoint* and *pole* [1]. Setpoint dictates acceptable time to wait for a response, and pole values closer to 1 means faster (but less safe) controller reactions. Local maxima at setpoint=0.05 and pole=0.7.

This experiment is conducted using the same simulator and scenario as the one in the previous section. However, the quality-elastic approach must be parameterized. The parameters are shown in Table II. Values for the service time and variance parameters were determined via experimentation using a single-server deployment of an online shopping system. We scaled the service times and variances such that they would be comparable to those of the `drop` and `similar-general` algorithms, which were already present in the Brownout simulator. Although there is a risk that such numbers are highly implementation-specific, readers familiar with database systems will recognize the relative relationship that e.g. `equal-tags` can be answered very quickly using indices, whereas more complicated queries such as in `similar-specific` will require more processing time.

Determining profitability levels for requests is complex [29], [30]. For simplicity, we scaled the profitability for the different algorithms as shown in Table II. For easy comparison with *Brownout*, `drop` is the baseline at profitability 1 and `similar-general` is at profitability 1.5. Getting significantly more individualized product recommendations is regarded to be twice (`similar-specific`) or thrice (`individual-model`) as profitable as `similar-general`. Recommendations of lesser quality are also scaled accordingly.

It is obvious that *how* the profitability factors are calculated will greatly impact the overall results. The numbers we have chosen here are likely to be as wrong as assuming that all product suggestions, regardless of their output quality and individualization, would contribute to 50% higher profits. If, however, we do assume that profitability is uniformly 50% higher for easy comparison to the other approaches that cannot choose other recommendation algorithms, the *q-e* approach gets an **even higher** profitability of 26006 (1.48 times that of *always*). We did, however, regard that as an unfair advantage, because if `cached-prior` would be as profitable

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

Table II: Parameters for the quality-elastic approach. Service times and variance have been determined experimentally. Profitability factors are estimated based on, and in relation to, prior work [1], [29].

Algorithm	Service time (ms)	Variance (ms)	Profitability factor
`drop`	0.67	1	1.0
`cached-prior`	0.85	1.5	1.2
`equal-tags`	2.5	8	1.4
`similar-general`	7	10	1.5
`similar-specific`	14	15	3.0
`individual-model`	30	100	4.5

Table III: Configuration of utilization thresholds for the quality-elastic approach (*q-e*). The values were determined experimentally.

Algorithm	Utilization threshold
`drop`	∞
`cached-prior`	0.99
`equal-tags`	0.75
`similar-general`	0.35
`similar-specific`	0.25
`individual-model`	0.05

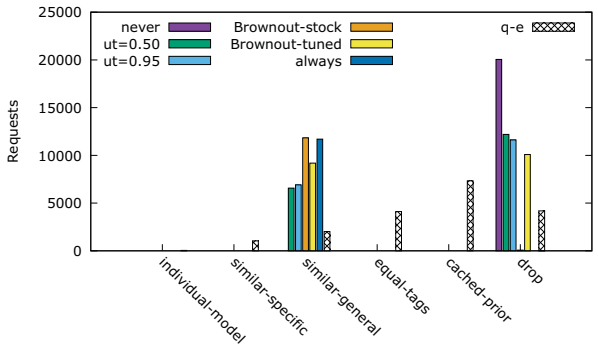


Figure 4: Frequency of chosen output quality levels during resource scarcity (Section V-B). Note that all but *q-e* are limited to making a binary choice between `similar-general` and `drop` by design.

as `individual-model`, not only would there never be a point to individualizing product recommendations, but also, the other approaches (including *Brownout*) should then just use `cached-prior` instead.

Determining actual profitability levels experimentally rather than estimation, thereby getting a measure of actual customer utility, is regarded as future work (Section VI).

The quality-elastic approach uses utilization thresholds to determine which algorithm to choose. The utilization thresholds were determined experimentally, not via optimization by a solver. Doing so in a practical way that does not merely trivially choose the `cached-prior` algorithm all the time is good future work, but outside the scope for this paper. The algorithms and utilization thresholds are shown in Table III.

Figure 4 shows that the quality-elastic service, represented by the *q-e* bars, serves more requests with optional content of at least some kind, compared to the others. In particular, it only decides to *drop* optional content for 23% of requests (Table I) and instead makes use of all quality levels available.

Table IV shows that the *q-e* service serves a large amount of total requests compared to stock *Brownout*, and a comparable amount (3% less) to the tuned version. Also, more than 75% of requests get optional content of some output quality level ($14548/18732 \approx 0.77$). With our previously discussed estimated profitability model, this leads to an overall profit level which is 40% higher than that of stock *Brownout*. A more naive model where "any served recommendation, regardless of quality, increases profitability by 50%" puts profit at 26006, or 46% higher than *Brownout-stock* and 10% higher than *Brownout-tuned*. We prefer the more conservatively estimated due to fairness. Although *Brownout-tuned* served more requests, total profitability was higher for the *q-e* approach.

While the purpose of this experiment is a direct comparison of *q-e* to *Brownout*, we note that the *q-e* approach also fares reasonably compared to the other approaches in terms of response time (Figure 1) and queued up requests (Figure 2). That it achieves a significantly higher increase in profitability compared to the binary utilization threshold-based approaches over *Brownout* (40% increase rather than about 20%, compared to the stock configuration) makes it the most interesting approach overall when resources are scarce.

C. Resource Utilization

There are two goals concerning utilization for our quality-elastic approach: (a) in cases of resource abundance, increase utilization by using more of the allocated resources; and (b) in cases of resource scarcity, prioritize throughput by emitting results of lower quality. Previous sections have explored effects of utilization thresholds and of quality-elasticity when resources are scarce. This section is concerned with resource abundance, and whether using quality-elasticity can make use of additional resources, thereby keeping utilization high.

Figure 5 shows average utilization as a function of the number of simultaneous clients. The *always* and *never* approaches behave as expected and are the theoretical bounds. The main differences between the other approaches are:

- The quality-elastic *q-e* approach takes advantage of idle resources much better than other approaches when resources are abundant (i.e. client count is low), at about a factor 2. Thus, it improves utilization as intended. It cannot spend too much time or resources on each request,

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

Table IV: Request breakdown and profitability comparing our quality-elastic approach (*q-e*) to *Brownout*. By being able to choose computationally cheaper algorithms to create optional content, it can do so more often while still serving significantly more total requests than stock Brownout, and thus achieves a higher profitability level.

	Served requests	**(factor)**	**With optional content**	**(factor)**	**Profitability**	**(factor)**
Brownout-stock	11 904	1.00	11 836	1.00	17 822.0	1.00
Brownout-tuned	19 277	1.61	9186	0.77	23 870.0	1.33
q-e	18 732	1.57	14 548	1.23	25 092.1	1.40

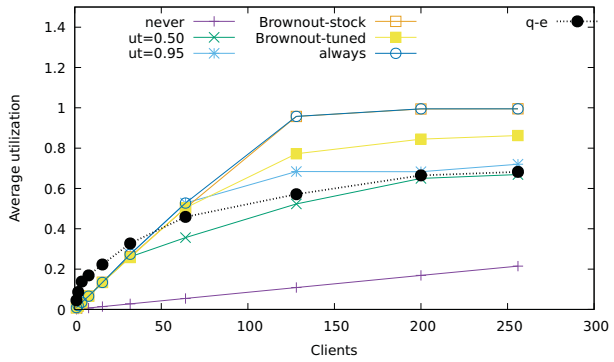


Figure 5: Average utilization as a function of number of clients. The number of clients increases in powers of 2 up to 256 (200 also included for reference). Resource abundance (low number of clients) shows significantly higher utilization for the *q-e* approach than others, and resource scarcity (high number of clients) shows that all approaches using utilization thresholds defensively avoid high utilization, opting instead to provide higher throughput than *always* and *Brownout*.

however, as that would adversely impact response times for each served request.

- All utilization threshold-based approaches (*ut=0.50*, *ut=0.95*, and *q-e*) keep average utilization quite far below the maximum, even during resource scarcity (i.e. higher numbers of clients). This is directly related to their ability to provide lower response times and better throughput than *always* and *Brownout*, as shown in Section V-A.

Figure 6b shows utilization only for the case when resources are abundant to make the data easier to see. Resource abundance in this case means that only 16 concurrent clients need service. The figure shows that the goals regarding making use of additional resources in times of abundance is best met by the *q-e* approach. However, we must investigate the output quality levels further to understand what is going on. Figure 6a shows the quality levels chosen by the different approaches during resource abundance. As in Section V-B, only the *q-e* approach can choose between all algorithms. The others may choose only to `drop` or to serve optional content using the `similar-general` algorithm. As expected, the *q-e* approach chooses the more computationally expensive ones (`similar-specific` and `individual-model`) most of the time, as there abundant resources to be used. Sometimes, due to variance (Table II), results come back so slowly and resources are in use for such a long time, that output quality must be lowered to maintain good throughput and response times in general. Relying on utilization thresholds makes this choice quick and easy to reason about.

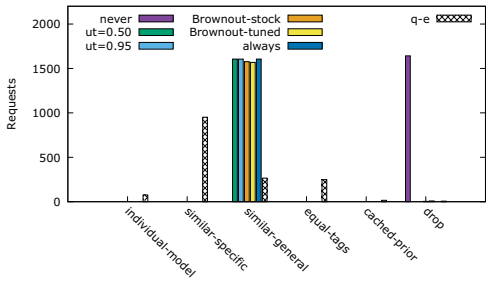
To summarize, our results show both that taking current operating conditions into consideration *and* that quality-elasticity improves response times, throughput, and service profitability. Quality-elasticity gives a service the ability to make decisions that can maximize profitability and end-user utility even further. In addition, it also helps ensure that resources are not wasted when request rates are low enough to permit more resources to be used per request.

VI. Future Work

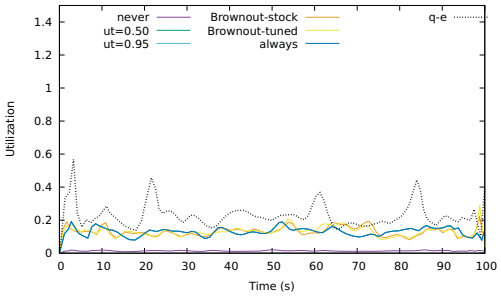
The contributions and positions stated in this paper are part of a larger body of related work (see Section III). We aim to contribute to the research area in the following ways based on the quality-elastic approaches described in this paper, motivated by the positive experimental results in Section V:

- Practical implementation of a quality-elastic demonstration application. This will help carry out experimental validation of the results presented in this paper, as they were obtained via simulation. Of particular interest is to investigate how quality-elasticity impacts service availability, resilience, and throughput in practice. We would also then like to implement and compare with more recent work on Brownout, e.g. [9], [10], [11], [12], [13].
- Study the relationship between utilization and utility, as a measure of how useful customers find a service. The profitability measures used both in [1] and in this paper are intentionally simple.
- Related to the previous point: customers experience utility differently, implying that per-user target quality levels may yield better results than treating all customers the same. A statistical method such as machine learning may help in determining these levels.
- Study the effects of collaborative quality-elasticity between service and infrastructure provider on scheduling, throughput, and resource utilization. Our hypothesis is that quality-elasticity enables collaboration to circumvent resource exhaustion due to load peaks. If so, cost savings should be possible, permitting more dense co-location of software execution environments. We have intentionally left out whether quality-elasticity is initiated by service or

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.



(a) Frequency of chosen quality levels during resource abundance. Like in Figure 4, only the *q-e* approach can choose algorithms other than `similar-general` and `drop`.



(b) Resource utilization for 16 concurrent clients.

Figure 6: Resource abundance use case, where 16 concurrent clients are used, rather than 200.

infrastructure providers in this work as it is an interesting and complex problem in itself.

- We also envision an API for collaborative quality-elasticity to carry out the previous item. This should be coupled with exploration of the question of how a micro-service, or the underlying cloud infrastructure, determines how much processing time to devote to each of the other micro-services it depends on to return a result to the user.

VII. Conclusions

Computational cloud resources are currently poorly utilized during normal operation and yet, in cases of high system load, service throughput suffers and response times increase. Both of these issues are rooted in the same core problem, namely that *services currently do not adjust their mode of operation to current operating conditions*. Our *main contribution* in this paper is that we have proposed quality-elasticity as a solution to this problem. A quality-elastic service adjusts the quality of its output on a fine-grained (or continuous) spectrum, thereby varying the amount of work that is needed to get results. When resources are abundant, more time can be spent on producing high-quality responses. When they are scarce, quicker and lower-quality responses must be given to keep throughput and response times at satisfying levels.

The quality-elastic concept builds and extends upon previous work on application *Brownout* [1], which was focused on a binary choice of whether to reduce quality or not. It neither captured increasing output quality when resource availability allows, nor did it consider more than two output quality levels.

Through experiments using the same simulator developed for *Brownout*, we have shown that by adding these aspects in a quality-elastic way, throughput can be increased by 50%, 95$^{\text{th}}$ percentile response times can be lowered by a factor 8, profitability increased by 40%, and utilization can be doubled in times of resource abundance, thus reducing idle resource waste. Based on these findings, we conclude that this quality-elastic approach and basing output quality decisions on current usage shows some promise, and motivates further research within this field.

Open Source and Open Data Notice

The modified code and experiment data used in the final version of the paper is available publicly on Github. The address is https://github.com/llarsson/brownout-simulator/tree/icac2019, which signifies that the `icac2019` branch is where the modifications reside.

References

[1] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

[2] Q. Wang, Y. Kanemasa, M. Kawaba, and C. Pu, "When average is not average: Large response time fluctuations in n-tier systems," in *Proceedings of the 9th International Conference on Autonomic Computing (ICAC '12)*. ACM, 2012.

[3] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017.

[4] A. Shehabi, S. Smith, N. Horner, I. Azevedo, R. Brown, J. Koomey, E. Masanet, D. Sartor, M. Herrlin, and W. Lintner, "United states data center energy usage report," Lawrence Berkeley National Laboratory, Berkeley, California, Tech. Rep. LBNL-1005775, 2016.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," Intel Science and Technology Center for Cloud Computing, Tech. Rep., 2012.

[6] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[7] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *SIGARCH computer architecture news*, no. 2. ACM, 2007.

[8] H.-E. Chihoub, S. Ibrahim, Y. Li, G. Antoniu, M. Pérez, and L. Bougé, "Exploring energy-consistency trade-offs in Cassandra cloud storage system," in *SBAC-PAD'15 — The 27th International Symposium on Computer Architecture and High Performance Computing*. DBLP, 2015.

[9] T. Nylander, M. T. Andrén, K.-E. Årzén, and M. Maggio, "Cloud application predictability through integrated load-balancing and service time control," in *International Conference on Autonomic Computing (ICAC)*. IEEE, 2018.

[10] A. V. Papadopoulos, C. Klein, M. Maggio, J. Dürango, M. Dellkrantz, F. Hernández-Rodriguez, E. Elmroth, and K.-E. Årzén, "Control-based load-balancing techniques: Analysis and performance evaluation via a randomized optimization approach," *Control Engineering Practice*, 2016.

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.

[11] M. Maggio, C. Klein, and K.-E. Årzén, "Control strategies for predictable brownouts in cloud computing," *IFAC proceedings volumes*, vol. 47, no. 3, 2014.

[12] C. Klein, A. V. Papadopoulos, M. Dellkrantz, J. Dürango, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, and E. Elmroth, "Improving cloud service resilience using brownout-aware load-balancing," in *33rd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2014.

[13] J. Dürango, M. Dellkrantz, M. Maggio, C. Klein, A. V. Papadopoulos, F. Hernández-Rodriguez, E. Elmroth, and K.-E. Årzén, "Control-theoretical load-balancing for cloud applications with brownout," in *53rd Annual Conference on Decision and Control (CDC)*. IEEE, 2014.

[14] A. V. Papadopoulos, J. Krzywda, E. Elmroth, and M. Maggio, "Power-aware cloud brownout: Response time and power consumption control," in *56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017.

[15] M. Xu, A. V. Dastjerdi, and R. Buyya, "Energy efficient scheduling of cloud application components with brownout," *Transactions on Sustainable Computing*, vol. 1, no. 2, 2016.

[16] M. Xu and R. Buyya, "Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions," *Computing Surveys*, 2018.

[17] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. ACM, 2013.

[18] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *International Conference on Advances in Computer Engineering and Applications*. IEEE, March 2015.

[19] P. Leitner and J. Cito, "Patterns in the chaos — a study of performance variation and predictability in public IaaS clouds," *Transactions on Internet Technology (TOIT)*, vol. 16, no. 3, 2016.

[20] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: Elastic distributed resource scaling for infrastructure-as-a-service," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. USENIX, 2013.

[21] L. Suresh, P. Bodik, I. Menache, M. Canini, and F. Ciucu, "Distributed resource management across process boundaries," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017.

[22] M. T. A. Amin, S. Li, M. R. Rahman, P. T. Seetharamu, S. Wang, T. Abdelzaher, I. Gupta, M. Srivatsa, R. Ganti, R. Ahmed, and H. Le, "SocialTrove: A self-summarizing storage service for social sensing," in *Proceedings of the 12th International Conference on Autonomic Computing (ICAC '15)*. IEEE, 2015.

[23] Z. Wen, D. L. Quoc, P. Bhatotia, R. Chen, and M. Lee, "ApproxIoT: Approximate analytics for edge computing," in *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[24] D. L. Quoc, I. E. Akkus, P. Bhatotia, S. Blanas, R. Chen, C. Fetzer, and T. Strufe, "ApproxJoin: Approximate distributed joins," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC '18)*. ACM, 2018.

[25] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. ACM, 2015.

[26] J. Kelley, C. Stewart, N. Morris, D. Tiwari, Y. He, and S. Elnikety, "Measuring and managing answer quality for online data-intensive services," in *International Conference on Autonomic Computing*. IEEE, 2015.

[27] W. Tärneberg, A. Mehta, J. Tordsson, M. Kihl, and E. Elmroth, "Resource management challenges for the infinite cloud," in *10th International Workshop on Feedback Computing at CPSWeek*, 2015.

[28] P. Skarin, W. Tärneberg, K.-E. Årzen, and M. Kihl, "Towards mission-critical control at the edge and over 5G," in *International Conference on Edge Computing (EDGE)*. IEEE, 2018.

[29] D. M. Fleder and K. Hosanagar, "Recommender systems and their impact on sales diversity," in *Proceedings of the 8th Conference on Electronic commerce*. ACM, 2007.

[30] D. Fleder and K. Hosanagar, "Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity," *Management science*, vol. 55, no. 5, 2009.

[31] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation*, ser. NSDI'06, vol. 3. USENIX Association, 2006.

[32] F. Nah, "A study on tolerable waiting time: How long are web users willing to wait?" *Behaviour and Information Technology*, vol. 23, 01 2003.

Authorized licensed use limited to: Lunds Universitetsbibliotek. Downloaded on May 30,2020 at 11:37:24 UTC from IEEE Xplore. Restrictions apply.