Cache Provisioning for Interactive NLP Services

Jaimie Kelley and Christopher Stewart
The Ohio State University

Yuxiong He and Sameh Elnikety
Microsoft Research

Abstract

Search engines and question-answer systems support interactive queries against natural language corpora. As their corpora grows, these *interactive natural language processing (NLP) services* use large portions of their IT budget to cache data in costly main memory. These caches ensure fast access to existing and recently added data. However, recently added data often overlaps with existing data in terms of informative content. Users will not perceive a loss in quality if such redundant data is excluded from cache. For this paper, we quantified cost savings when caches are rightly provisioned so that they are just large enough to avoid quality loss. We set up two NLP services, a search engine and a question-answering system, that supported growing corpora from Wikipedia and the New York Times (up to 88MB and 30GB per month, respectively). First, we studied the effect of keeping cache size fixed as new data arrived. Our results show that some NL corpora survived with fixed-size cache for months without incurring much quality loss. We also studied novel provisioning triggered by quality loss—rather than data growth. Depending on the quality-loss thresholds, our approach reduced costs by 45–98% relative to provisioning resources up front and 6-52% relative to provisioning each month.

1 Introduction

Unstructured, natural language (NL) corpora are large and growing fast. As of this writing, Twitter receives more than 300M tweets per day, a 2X increase over 2010 [20]. TripAdvisor holds over 100M user reviews, a 2X increase since 2011 [10]. Search engines, question-answer systems, and other big-data services process user queries against such data. To meet tight response time limits, these services cache data in the main memory of large clusters. For example, TripAdvisor uses a MemCache cluster on Amazon EC2, and this cluster comprises 52% of its online storage costs [10, 14]. As data grows, these costly caches require additional resources.

Given the costs of long response times, many services that process natural language data are designed to compute partial results quickly rather than full results slowly [12]. These services impose processing timeouts; a query that times out accesses only a fraction of its data. The difference between results returned with timeouts enabled (i.e., constrained resources) and results with infinite resources is *quality loss*. Users are often satisfied as long as quality loss is small. Large, in-memory caches prevent quality loss by allowing queries to access a lot of data within processing timeouts. However, NL corpora present a challenge: Documents contain redundant information. Services can over provision caches when the corpora grows faster than its informative content. Over provisioned caches inflate operating costs by forcing managers to expand capacity sooner than needed. With memory prices dropping by an average of 30% per year [16], it is cost effective to wait as long as possible before buying resources.

This paper argues that caches for NL workloads should be provisioned for quality loss, not data growth. These workloads permit some quality loss because NL concepts, e.g., synonyms and noisy results, introduce redundancy into query results. We present an approach to measure quality loss that captures these concepts. First, a query's baseline results were defined as those computed under a fully provisioned cache with no timeouts. We computed quality loss by comparing the baseline results with results observed under smaller caches. Queries had access to the same available data within a quality-loss test, but between tests we replayed data growth.

We set up two systems: Apache Lucene [1], an open source search engine, and OpenEphyra, an open source question answering system like IBM's

Watson [8] that uses unstructured data. We used two NL datasets: Wikipedia and The New York Times. We organized each corpus into monthly snapshots, allowing us to measure quality loss over time as data grew. The portion of the Wikipedia corpus used grew by at most 30GB per month. The New York Times corpus added at most 88MB per month. From 2006–2008, our Wikipedia dataset exploded by more than 3X in raw size. We used a Redis [2] cluster as a main memory cache in our setup, and Google Trends to create a sequence of queries that were popular during periods studied. We replayed queries one-by-one under processing timeouts.

Quality loss varied based on 1) the corpus and 2) cache management policy. Cache under provisioning almost always caused quality loss, but often the effects were small. However, if our search engine permitted some quality loss among the top K query results, it could provision 50% fewer cache resources on both Wikipedia and New York Times. We further observed that the New York Times corpus permitted a greater degree of cache under provisioning.

We also studied the impact of well known cache management policies. In *term-based LRU*, we stored only Lucene's inverted index in our main memory cache. When the cache was under provisioned, least recently used terms were swapped out of memory. This policy is widely used in search engines that provide pointers to content, rather than the actual content. In contrast, question-answer systems and online review engines often provide actual content. These workloads may prefer *content elision*, in which certain documents are elided from the indexes. Content elision is commonly used when new data replaces old data and the active size of the corpora is fixed. Terms in the resulting inverted index references fewer documents compared to inverted-index terms derived from the full corpora. In the worst case scenario of content elision, which we analyzed, new data is indexed only after a quality loss threshold is exceeded. Term-based LRU incurred less than 30% quality loss on both corpora. This result held even when the cache was severely under provisioned. Content elision incurred less than 30% quality loss on only the New York Times corpus. We hypothesize that content elision required more data redundancy to be effective.

We also used our framework to study the following policy: When quality loss exceeds a threshold, add more servers to expand the cache. We compared this approach to other approaches, including naively provisioning enough resources to fully provision the cache for the full corpus. Our approach reduced costs in two ways. First, it provisioned resources on demand, reducing operating costs. Second, it would enable managers to buy hardware later rather than sooner, taking advantage of falling DRAM prices. Compared to buying enough memory servers upfront to handle 3 years of data growth, our approach reduced costs by 92%. Compared to an on-demand approach driven by monthly data growth, our quality-aware approach reduced costs by up to 48.8%.

The remainder of this paper is organized follows: Section 2 defines quality loss in the context of NL workloads. Section 3 describes our experimental results. Section 4 discusses related work. Section 5 concludes.

2 NLP Workloads

We interact with NL throughout our lives. We have learned to tolerate imprecise typographical errors, grammar, accents, and idioms. Services that process NL corpora also benefit from precision tolerance. We classify two key types of precision tolerance based on our experience with search engines.

Synonyms: Words and word sequences often have the same meaning within the context of a query. The precise output of a search engine with fully provisioned cache may output links to many of these synonyms. However, users are satisfied when a subset appears on their screen. For example, a Bing search for "Flowers in Washington State" returns results on florists, gardening, and the Coast Rhododendron (state flower). With a smaller cache, some of these results would be elided, but as long as the categories are represented (on the top answer page), many users will be satisfied.

Noise Tolerance: Continuing the example above, adjacent search results on the answer page represent different categories. Users are often willing to parse unrelated categories to find the desired content. In other words, a certain degree of noisy results are okay as long as users can find good answers.

2.1 Defining Quality Loss

Quality loss (QL) is a metric to determine answer dissimilarity between an underprovisioned system and a fully provisioned baseline. To compute quality loss, we use the equation:

$$QL(x,\hat{x},D,Q) = 1 - S(x,\hat{x},D,Q) \ , \qquad (1)$$

where x is our current underprovisioned hardware, software, data, and settings configuration, and $\hat{x}$ is the same configuration with enough cache resources to avoid timeouts. The function S is a measure of answer set similarity. We issue a set of queries Q and, for each query q_i, we compare its answers under x to its answers under $\hat{x}$.

Our similarity function is based on recall of the top-k results. We perform k-pairwise string comparisons, matchings top results under $\hat{x}$ (i.e., $R_i(\hat{x},D)$) to results under x. When we find a match, we count it and move on to the next result string from the $\hat{x}$ answer set. At most one match for a single answer from a single question will be counted. The total number of matches is divided by K and averaged across all questions in Q. Equation 2 captures this base model and extends it to handle synonyms and noisy data.

$$S(x,\hat{x},D,Q) = \frac{\sum_Q \sum_K \phi(\sum_{k_2} |R_{q,k}(\hat{x},D) \cap R_{q,k_2}(x,D)|)}{|Q| \cdot K} \qquad (2)$$

Capturing Synonyms: We specify a parameter K to use in a top-k analysis of quality, and thereby only look for matches of the first K result from the answer set. For example, in web search, K can be set as the number of results on the first page; these $K = 10$ results are the most critical to deciding result quality. As K decreases, the number of potential matches decreases and the denominator decreases; but as K increases, the difference between the current quality loss and the quality loss at $K-1$ decreases until quality loss stays within 5% of the quality loss at the previous K.

Support for Noisy Results: Users are willing to look through some number of results to find what they were searching for. We add a parameter k_2 to capture this and revise the top-k analysis to top-2k analysis. Similar as top-k analysis, the top-2k analysis uses the K number of the top baseline results as denominator; but differently, it uses the number of matches between the top-K baseline results (from $\hat{x}$) and the top-k_2 test results (from x) as nominator, where $k_2 \geq K$. The relative difference of k_2 and K reflects users' tolerance level to noise. When users cannot tolerate any noise, we require $k_2 = K$; with higher tolerance of noise, k_2 can be more significantly larger than K.

Note, quality loss depends on the full specification of the above parameters and varies across services and users. A key contribution in our study is empirical analysis across a wide range of quality-loss settings.

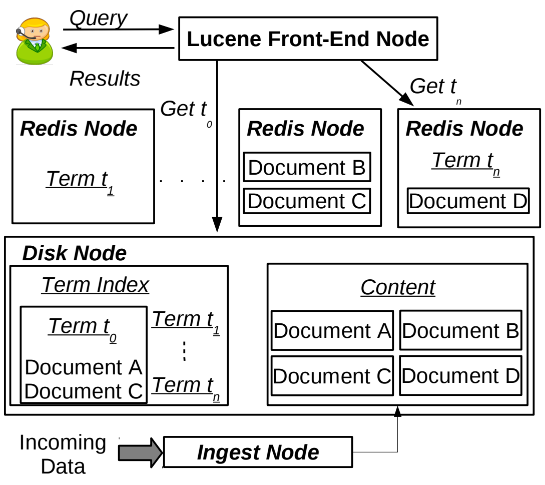
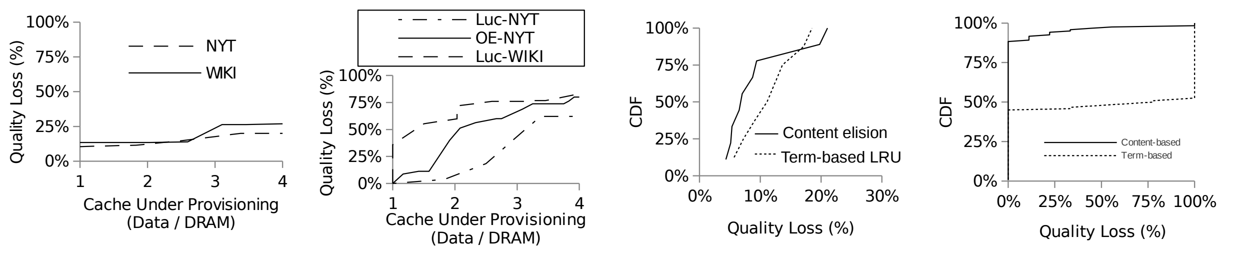


Figure 1: Our system setup for experimentation, including service logic for both applications we used.

3 Experimental Results

Figure 1 shows our system setup. For a given a query, Lucene's front-end nodes first look up query terms in a distributed Redis cache. Each Redis node stores up to 9 GB of data in its main memory. When more DRAM cache is needed, the cache scales out via additional Redis nodes. Terms not found in the Redis cache are looked up on two dedicated disk nodes that store 3 TB each. For each query, Lucene waits until all term data is found or a timeout occurs. Results are then analyzed, aggregated, and returned to the user. If the service logic in the application layer analyzes content, this content is also cached in Redis. For each



(a) Quality loss of NYT vs Wiki (b) Content elision caching (c) Distribution of quality loss by replacement policy (d) Quality loss per question

Figure 2: Cache under provisioning on quality loss.

experiment, we set query timeout, active Redis and disk nodes, and disk access times by broadcasting a configuration file to all nodes.

Our experiments run on a 112-node local cluster with EC2-like cloud provisioning. Each node has a 2.66GHz processor, 1 Gb Ethernet, and 100 GB local disk storage. Two dedicated disk nodes with the same specifications also have access to their own 3 TB external hard disk. The nodes described in Figure 1 communicate through software-defined image names.

We use Google Trends to capture 2,000 popular web searches representative of queries from 2004–2008. For each experiment, we replay these queries one by one. Typical response times are 500ms. We use Equation 1 to define quality loss. Our default configuration sets $K = 10$, $k_2 = 30$, and query timeout equal to 10 seconds.

3.1 Comparing NLP Datasets

Our data from The New York Times (NYT) spans articles published from 2004 to 2006. Over our trace, the corpus doubles in size to about 3GB indexed. However, new articles often repeat informative content from prior articles, reflecting follow-up stories and opinions pieces based on recent news articles.

Our data from Wikipedia (Wiki) spans articles published from 2001 to 2013, including revision data. We use two 3TB disk nodes to store the entire data set. Unlike New York Times, Wikipedia has less repeated content. Revisions often extend articles rather rephrasing existing content. However, links between entries can be repetitive, carrying over terms and copying definitions.

Figure 2(a) shows the observed quality loss across each data set as we increasingly under provision the cache, i.e., as data grew, we updated and increased the size of the term index on disk, but did not provision additional cache resources. Over time, the under provisioned cache pushed less popular terms to disk, increasing the probability that these terms would not be retrieved within the timeout. Both data sets handled under provisioned caches well; neither exceeded a 30% quality loss threshold.

3.2 Cache Replacement Policies

Figure 2(a) showed results where we updated the term index at each data snapshot. Under provisioned caches used LRU policies (a part of Redis) to manage growing data. For this section, we studied an alternative approach called content elision, in which the index size is kept static. In the worst case of content elision, the term index is not updated. Referring to Figure 1, we configured the ingest node to hold incoming data, instead of forwarding to the disk node. For services that store both term indexes and content in main memory caches, each new piece of data can use a lot of space. These services may prefer content elision because it prevents data growth. However, content elision can lead to high quality loss when incoming data is not highly redundant with existing data.

Figure 2(b) shows that quality loss under content elision varies depending on dataset and application. With Lucene on the NYT data, data growth can double the original cache size before hitting 10% quality loss. However, the less-redundant Wiki data suffers with quality loss starting at 35%. To analyze content elision, we set up an additional service, which accesses content and term indices stored within the Redis cache. OpenEphyra is question-answer system in the mold of IBM Watson [8]. It uses Lucene to identify documents related to a NL question and scans the top documents' contents for an answer. OpenEphyra uses the NYT workload. Our results show that for OpenEphyra up to 1.5 of the original data size can be added before hitting a 12% quality loss threshold. We suspect that the difference between OpenEphyra and Lucene on the NYT dataset is the effect of cache pressure from actual content access.

3.3 Whole Distribution Analysis

For Figure 2(c), we ran tests at a fixed under provisioning ratio (i.e., $\frac{data}{dram} = 2$). Each test used different data snapshots. We observe significant variance across the snapshots; quality loss increased by more than 3X for both caching policies. However, content elision has a significantly heavier tail relative to term-based LRU because when key documents are elided, the quality loss from content elision can affect many queries with terms described within the document. For Figure 2(d), we plot quality loss for one test on each question in our trace. Under content elision and NYT, we observe that most questions incur no quality loss at all, but the outliers that experience 100% loss (i.e., none of their results are the same) pull average quality loss up to 10%. Under term-based LRU with Wiki, we selected one of the worst data points (average quality loss was 45%) to highlight the on-off behavior of the replacement policy. If a query's terms are totally on disk, quality loss is high despite potential redundancy in the data.

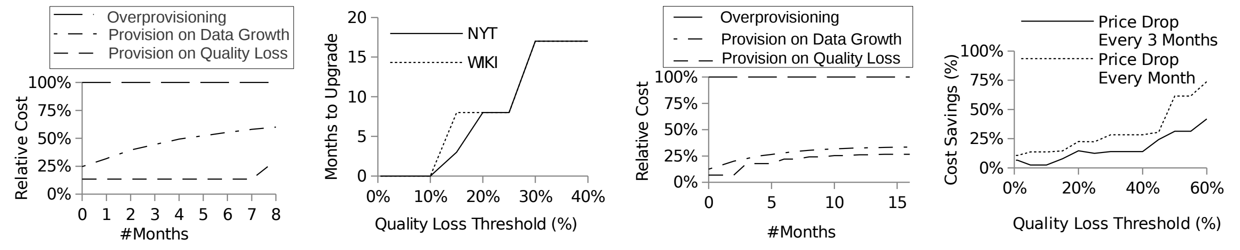
3.4 Cache Provisioning on Quality Loss

In this section, we propose a new cache provisioning policy: *Expand the cache when when quality loss exceeds a threshold.* When quality loss does exceed a threshold, we add enough DRAM Redis nodes to fully provision the cache. Then we wait for quality loss to exceed the threshold again. By default, we set the threshold to 20%, but we explore the impact of all threshold settings. We call our approach *provision on quality loss*.

We compare our approach to two alternative provisioning policies. *Over provisioning* avoids any quality loss by provisioning enough resources to cache the entire NYT corpus up front. This policy has increased operating costs; since the average cost of DRAM is steadily decreasing, this policy also pays more per bit for cache resources. *Provision on data growth* provisions resources at each data snapshot, avoiding the increased price per bit from overprovisioning. As in our approach, this approach avoids the initial cash outlay. We assume all unspent cache budget is invested at 0.5% APR. Cost savings occur as interest gained from this investment plus the difference between the original price and the reduced price for DRAM.

We assume that DRAM prices drop on average by 2% per month, and simulate cost savings using our NYT and Wiki data for price drops every month and for price drops every three months. Figure 3(a) shows the cost of our approach relative to over provisioning and provisioning on data growth under the NYT dataset on Lucene, using term-based cache elision. Over 8 months, when our approach first provisions cache resources, our costs are 30% of the over provisioning case and half of the provisioning on data growth approach. Figure 3(b) shows the number of months that we can go before provisioning as a function of the quality loss. Here, we show results for both NYT and Wiki. For the New York Times data set under a DRAM price drop every three months, we save 19.45% compared to upgrading every time we add data. For the Wikipedia data set under a 3-month DRAM price drop, we save 14.37% compared to upgrading at every data add. For the New York Times data set, we save 51.19% compared to upgrading every month when we simulate a price drop every month; for the Wikipedia data set, we save 24.31% compared to upgrading at every data add when we simulate a price drop every month.

Figure 3(c) uses the same methodologies as the above but for content elision instead of term-based LRU elision. Our provisioning on quality loss approach saves more relative to the over provisioning approach, costing only 20%, but the provision on data growth approach is more competitive. This is because content elision requires updates to DRAM more frequently than term-based LRU. Figure 3(c) shows that when the DRAM cost drops every month, we save 22.51% of the cost of provisioning based on



(a) Term-based LRU cache policy

(b) Effect of quality loss threshold on term-based LRU

(c) Content elision

(d) Effect of quality loss threshold on content elision

Figure 3: Cost savings of cache provisioning approaches.

data growth and 80.44% of the cost of over provisioning.

As Figure 3(d) shows, the cost savings from increasing the quality loss threshold at an interest rate of 0.5% increases modestly when we we compare provisioning based on quality loss to provisioning based on data growth. With a quality loss threshold of 20%, we save 14.64% using the New York Times data set and 11.15% using the Wikipedia corpus for an every three month cost decrease. When the DRAM cost drops every month, we save 6.14% using Wikipedia and 22.51% using the New York Times dataset. This savings will grow as the threshold is relaxed; cost savings also increase as data is added. The numbers presented in this graph are subject to small fluctuations dependent upon the point at which we add data.

3.5 Additional issues

One of the parameters that affects quality loss regardless of caching policy used is the choice of presentation. All of the Lucene quality loss numbers presented in this paper use the top-2k method of comparison, with a k of 10 and a k_2 of 30. As Figure 4 shows, the choice of k matters for the results coming from Lucene. A k less than 10 will result in showing higher quality loss than is average for the run, and a k greater than or equal to 10 will be result with quality loss within 5% of the average quality loss over all values of k.

Instead of changing the apportioned DRAM, we could modify the parameter that specified the timeout allowed by the system to analyze the effect of this timeout on quality loss.

Figure 5 shows the results of changing the timeout threshold over multiple different values of Data / DRAM. The lowest timeout threshold shown, at 1 second, resulted in a very high number of timeouts and very few results returned as compared to the other timeout thresholds shown. A timeout threshold of 5 seconds resulted in fewer timeouts and a correspondingly lower quality loss. A ten second 10 second threshold is slightly worse than the threshold with the lowest quality loss, which was 30 seconds.

4 Related Work

Our work intersects information retrieval, natural language processing, and storage systems. We exploit imprecision inherent in NL workloads to reduce caching costs. Our experiments with real NL workloads suggests that caches can be significantly under provisioned without incurring much quality loss.

Approximate computing also focuses on workloads that tolerate imprecision. For example, anytime algorithms [24] define a class of problems that can be solved incrementally. If the algorithm is interrupted during its execution, an imprecise result is returned. In contrast, compilers that support loop perforation [13] accept total running time as input. This approach elides loop iterations to complete within preset running times. Similarly, web content adaptation [4, 9] degrades image quality and webpage features to meet response time goals. Our own prior work [11] studies approximate computing within search engines, where a request may return partial results to complete within processing timeouts. These works, for the most part, trade off response time and

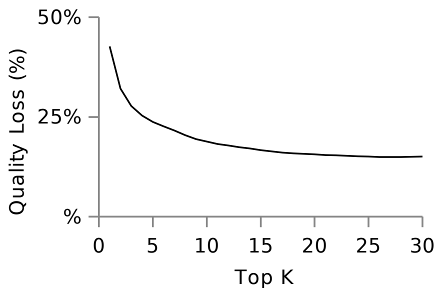


Figure 4: The effects of varying *k* on quality loss for a single experiment.

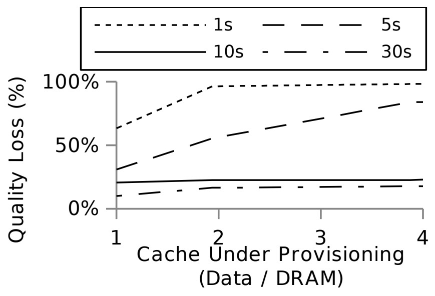


Figure 5: The effects of changing threshold on varied DRAM configurations over the same amount of New York Times data on Lucene using term-based caching.

imprecision. In contrast, our goal in this paper is to trade imprecision for reduced cache costs. Baek and Chilimbi [3] present a general framework to support approximated computation of different applications to tradeoff between quality and energy consumption.

Cache replacement and compression share our goal of provisioning fewer resources without incurring quality loss or high response times. SILT [17] is a key-value store that spans main memory, SSD, and disk. It combines diverse data structures across these materials, trading access time overhead with compression. Chockler et al. have begun studying caching as cloud service to achieve improved cache replacement under diverse, consolidated workloads [5].

Several recent works profile application access patterns to reduce cache contention between competing applications [7, 19, 22]. Processing timeouts are akin to service level objectives. Recent work has shown that meeting strict objectives requires novel designs [6, 15, 21].

Capacity planners traditionally provision resources based on models of data growth, in part because non-NL workloads are less permissive to imprecision. Recent work from Google [23] models the growth of data. Their approach profiles specific services and achieves predictably low error. Mackie [18] provides an earlier, macro-analysis forecasting approach.

5 Conclusion and Future Considerations

Cache provisioning for interactive services can be made more cost effective by becoming quality-aware. In this paper we use Lucene, a search engine which processes data from either a New York Times data set or data from Wikipedia, to show how much cost savings can be effected by setting a quality loss threshold. We examine two caching policies, content-based and term-based elision, and show that while both can provide a cost savings under a quality loss threshold, each has advantages. Content-based caching works best for low values of data/DRAM, while term-based caching scales better as data is added. Content-based caching can save 22.51% using the New York Times data set and 6.14% using the Wikipedia data set at a 20% quality loss threshold, while term-based caching can save 41.62% using the Wikipedia data set and 52.47% using the New York Times data set under the same 20% quality loss threshold. We identify variations on computing quality loss regarding synonyms and noisy data, and examine how system parameters can effect quality loss.

So far our calculations of quality loss has been done offline by automated perl scripts. One aspect that we have so far omitted from the discussion is how this metric can be used to detect quality loss in systems in an offline or online system. We are currently pursuing options that include subsampling data and queries for a minimal offline system to detect quality loss, extending timeouts linearly in an offline system to enable disk storage to mimic cache, and shadow querying for online quality loss detection. We are also considering the problem of how to automate the acquisition and integration of additional cache into the online system case.

References

[1] Apache lucene. `http://lucene.apache.org/core/`.

[2] Redis. `http://redis.io/`.

[3] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.

[4] Y. Chen. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW*, 2003.

[5] G. Chockler, G. Laden, and Y. Vigfusson. Design and implementation of caching services in the cloud. In *IBM Technical Report*, 2012.

[6] J. Dean and L. Barroso. The tail at scale. In *Communications of the ACM*, 2013.

[7] C. Delimitrou, N. Bambos, and C. Kozyrakis. Qos-aware admission control in heterogeneous datacenters. In *IEEE ICAC*, 2013.

[8] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Hyberg, J. Prager, N. Schlaerfer, and C. Welty. The ai behind watson—the technical article. In *The AI Magazine*, 2010.

[9] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to network and client variation using infrastructural process proxies: lessons and perspectives. *Personal Communications*, 5:10–19, 1998.

[10] A. Gelfond. Tripadvisor architecture - 40m visitors, 200m dynamic page views, 30tb data. `http://highscalability.com`, June 2011.

[11] Y. He, S. Elnikety, J. Larus, and C. Yan. Zeta: scheduling interactive services with partial execution. In *SOCC*, 2012.

[12] Y. He, S. Elnikety, and H. Sun. Tians scheduling: Using partial processing in best-effort applications. In *ICDCS*, 2011.

[13] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. C. Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, 2001.

[14] S. Hsiao, L. Massa, and V. Luu. An epic tripadvisor update: Why not run on the cloud? the grand experiment. `http://highscalability.com`, Oct. 2012.

[15] J. Hwang and T. Wood. Adaptive performance-aware distributed memory caching. In *IEEE ICAC*, 2013.

[16] International Technology Roadmap for Semiconductors. The itrs dram cost is the cost per bit (packaged microcents) at production. http://www.itrs.net/.

[17] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. Silt: a memory-efficient, high-performance key-value store. In *SOSP*, 2011.

[18] S. Mackie. How fast is our data volume growing. Storage Strategies Inc., 2009.

[19] H. Madhyastha, J. McCullough, G. Porter, R. Kapoor, S. Savage, A. Snoeren, and A. Vahdat. scc: Cluster storage provisioning informed by application characteristics and slas. In *FAST*, 2012.

[20] C. Roe. The growth of unstructured data: What to do with all those zettabytes? `www.dataversity.net`, 2012.

[21] C. Stewart, A. Chakrabarti, and R. Griffith. Zoolander: Efficiently meeting very strict, low-latency slos. In *IEEE ICAC*, 2013.

[22] C. Stewart, K. Shen, A. Iyengar, and J. Yin. Entomomodel: Understanding and avoiding performance anomaly manifestations. In *IEEE MASCOTS*, 2010.

[23] M. Stokely, A. Mehrabian, C. Albrecht, F. Labelle, and A. Merchant. Projecting disk usage based on historical trends in a cloud environment. In *ScienceCloud*, 2012.

[24] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3), 1996.