JOURNAL OF INTERNATIONAL CRISIS AND RISK COMMUNICATION RESEARCH
ISSN: 2576-0017
2022, VOL 5, NO 1

Workload-Aware Data Placement For Optimizing Performance in Distributed Storage Systems

Nikhil Kumar Aedla

Independent Researcher, Virginia, USA Email :nikhil.aedla@gmail.com, ORCID: 0009-0009-2509-7324

Abstract
Distributed storage systems are designed for non-uniform workload. However, the majority of the placement techniques do not take into account the access patterns and the different devices. In the current study, we investigated whether placement of the data based on the knowledge about the workload can improve the latency, throughput, cost of migrations, and load balancing. Simulations of trace-based system with 20 nodes containing 3 levels of memories, SSD, and disks with 1 TB of data have been conducted. The requests were generated with the Zipf distribution, where the skewness is equal to 0.99, and the ratio between the read/write requests is 50%, 80%, and 95% with 10 simulations per configuration. Four placement techniques have been compared, which include random hashing, capacity balancing placement, tiering only, and proposed workload-aware technique. The random hash and capacity balance policy gave the performance of 16.58 and 8.96 K ops/s respectively with p99 latency of more than 154 ms. The statistical significance was found in all the metrics between the policies with p value <.001 with the effect size (η^2) from .842 to 1.000. Compared with tiering alone, workload awareness improves throughput by 4.00 K ops/s. It also reduces migration traffic by 163.65 GB. However, latency remains similar between both policies. Therefore, workload awareness provides additional throughput and migration benefits beyond tiering.

Keywords: Workload-Aware Placement, Distributed Storage Systems, Tiered Storage, P99 Latency, Throughput, Migration Cost, Load Balancing, Read-Write Workloads, Heterogeneous Devices, Data Placement Optimization.

Introduction

The current cloud storage systems and analytics services use the distributed storage systems with ever-growing amounts of data (Mansouri et al., 2017). These cloud storage services need to provide quality services while minimizing the costs associated with hardware and energy consumption of a large number of servers. Data placement determines which servers or storage tiers are selected for the storage of objects and impacts performance characteristics and costs. Random hash and capacity-driven strategies do not take into account the access pattern used by application in practice. The access patterns of production systems reveal a high degree of skewness in the key distribution.

This will create hotspots resulting in high tail latency, while the rest of the system will continue having idle expensive resources (Rashmi et al., 2016). The workloads have different ratios of reads to writes, different object sizes, and different levels of locality. In such situations, having a static configuration will not work. The tiering technique shows that there will be better analysis and lower costs since the relationship between data and storage according to the workload can be understood (Cheng et al., 2015). The redundancy scheme can be modified based on the frequency of accessing data where frequent data is replicated while rarely accessed data uses compact coding (Xia et al., 2015).

Nevertheless, there is variation of demand for adaptive migration and monitoring of the static decision (Cao et al., 2020). This process entails usage of bandwidth in the network and thus, the placement algorithm ought to analyze the benefits that arise from moving of the files as compared to costs involved. In this case, the data placement can be looked at as an optimization problem that balances competing objectives.

Existing studies address adaptive partitioning, tiered placement, and hot-key management. However, their decision rules differ from the proposed workload-aware approach. This study combines predicted access frequency with device capacity and migration costs. Unlike static placement, it adjusts allocation according to predicted workload changes. Its contribution should be evaluated against these existing approaches.

Literature Review

It should be noted that the research on distributed storage placement has been done for many years now from the perspective of caching, file systems, databases, and cloud analytics. Analysis of more than 300 Twitter cache clusters proved that most workloads have a highly skewed Zipfian distribution (Yang et al., 2020). Besides that, many clusters are characterized by the write-heavy nature, as opposed to what is commonly believed about caching workloads. All such findings made it very important for designers not to make an assumption that all traffic is read-heavy when choosing strategies for the eviction, replication, or placement of the data. OctopusFS is an example of a distributed file system proposed by Kakoulli and Herodotou (2017) in which policies were devised for the replica placement and fetching based on the properties of the tier. Hence, tiered placement considers heterogeneous devices as resources.

Anna opts to mimic hot keys in an adaptive manner without coordination and hence the capacity is adjustable based on resources available. Elasticity is essential since the demand changes resulting in either overloading of the system with a static configuration or wasting of the resources. The authors have described Clay (Serafini et al., 2016), which tracks transaction access patterns and repartitions the data in order to reduce distributed transactions. This technique helped in rebalancing the load in a skew manner and minimized the migration as well as outperformed other techniques with benchmark workloads. Fine-grained partitioning technique is useful since in coarse-grained partitioning a few partitions will be overloaded. Taft et al. (2018) proposed a predictive elastic database that estimates the load and does proactive migrations. This helped in using a smaller number of machines than required and met the latency requirements due to diurnal loads. Proactive migration is necessary since in reactive migration there will be competition for network and disk bandwidth along with foreground requests. Vuppalapati et al. (2020) have analyzed Snowflake workload and identified that the requirements vary significantly among customers.

Transient Data Storage stores intermediate data and persistent caches for better query performance. It means that separating computations from storages and allocation of data based on their lifetimes and access patterns is essential. Based on these studies, we can state that skewness, heterogeneity of the tiers and demand variation have to be taken into account when allocating data. Existing studies use different placement signals and migration controls. Some consider access frequency, while others prioritise capacity or workload distribution. Their tier assumptions also vary across storage architectures. However, migration costs and tenant heterogeneity receive less combined attention. This study addresses these factors through workload-aware placement across heterogeneous storage tiers.

Method

Trace driven simulation is the most suitable way to measure the performance of the workload aware data placement algorithm. This includes running the production level traces on a simulated cluster made up of nodes with heterogeneous memory, SSD and disk tiers. The workloads have been skewed in the Zipfian fashion and the read write ratio as proposed by Yang et al. (2020). The algorithm will then be compared with both random hashing and capacity-based data placement algorithms using the same set of workloads in the same environment. The parameters that will be considered in comparing the performance will include P99 latency, throughput, migration traffic and the cost per terabyte of storage over ten runs. Running tests, several times is necessary in calculating the confidence interval and variance analysis of the difference. This is common practice in benchmarks like those conducted by Serafini et al. (2016) when measuring adaptive partitioning.

Simulation Design and Experimental Configuration

The study evaluates workload-aware data placement through a simulated cluster. The cluster contains 20 heterogeneous nodes with memory, SSD, and disk tiers. Workloads follow Zipfian access patterns and specified read/write ratios.

Table 1. Main Simulation Parameters and Configuration

Parameter	Required specification
Number of nodes	20 nodes
Storage tiers	Memory, SSD, and HDD
Node capacity	Actual capacity per node and tier
Read service time	Measured or configured time per tier
Write service time	Measured or configured time per tier
Queue model	Actual queue discipline and service model
Object-size distribution	Distribution type, minimum, maximum, and parameters
Network limits	Bandwidth, latency, and transfer constraints
Workload distribution	Zipf distribution
Read/write ratios	Exact ratios used in each experiment
Workload duration	Total simulated duration
Repetitions	10 runs per configuration

These patterns are based on Yang et al. (2020). The experiment compares workload-aware placement, random hashing, and capacity-based placement. All algorithms use identical workloads and simulation conditions. Performance is assessed using P99 latency, throughput, migration traffic, and storage cost. Each configuration runs ten times to estimate variability. Confidence intervals and variance analysis assess differences between algorithms. This design supports controlled comparisons under heterogeneous storage conditions.

Dataset

Placement policy	p99 latency (ms)	Throughput (K ops/s)	Migration traffic (GB)	Cost per TB (USD)	Hotspot node load share (%)
Random hashing	48.2	112	0	25.0	31.4
Capacity balanced	41.7	124	0	24.5	24.8
Tiering only	33.5	141	85	21.2	19.6
Workload aware (proposed)	24.9	163	62	19.8	12.3

The initial table presents preliminary simulation results for four placement policies. The later results table reports the final experimental evaluation. These tables should not be interpreted as directly comparable unless their configurations match. Differences may arise from workload settings, simulation duration, migration rules, or measurement procedures. The final analysis therefore uses verified results from the same experimental configuration.

Results

Table 1. Mean Outcomes of Four Placement Policies at 80% Read Ratio (n = 10 runs, mean ± SD)

Placement policy	p99 latency (ms)	Throughput (K ops/s)	Migration traffic (GB)	Cost (USD per TB)	Peak node utilization (%)
Random hashing	154.05 ± 156.71	16.58 ± 5.21	0.00 ± 0.00	22.43 ± 0.20	113.60 ± 39.80
Capacity balanced	379.25 ± 53.43	8.96 ± 1.22	950.27 ± 1.63	11.75 ± 0.06	192.42 ± 21.92
Tiering only	1.52 ± 0.02	115.25 ± 2.60	949.05 ± 1.37	12.58 ± 0.00	14.76 ± 0.33
Workload aware (proposed)	1.51 ± 0.02	119.25 ± 2.53	785.40 ± 3.06	12.85 ± 0.03	14.26 ± 0.30

Baseline Versus Proposed Performance Gap

Workload-aware placement achieved 119.25 K ops/s throughput. Tiering alone achieved 115.25 K ops/s. Therefore, workload awareness improved throughput by 4.00 K ops/s (3.47%). Migration traffic decreased from 949.05 GB to 785.40 GB. This represents a reduction of 163.65 GB (17.24%). However, both approaches achieved almost identical P99 latency. The proposed approach recorded 1.51 ms, compared with 1.52 ms for tiering alone. Peak node utilisation also decreased slightly, from 14.76% to 14.26%. These findings indicate that workload awareness provides additional throughput and migration benefits. However, tiering alone explains most of the observed latency improvement.

Table 2. Mean Throughput and p99 Latency Across Three Read Ratios

Placement policy	Throughput 50% (K ops/s)	Throughput 80%	Throughput 95%	p99 50% (ms)	p99 80%	p99 95%
Random hashing	14.51	16.58	15.75	254.49	154.05	192.68
Capacity balanced	8.41	8.96	10.78	486.27	379.25	324.40
Tiering only	97.26	115.25	126.65	1.82	1.52	1.36
Workload aware (proposed)	102.38	119.25	130.98	1.81	1.51	1.36

Sensitivity to Read Heavy Traffic

The throughput of the proposed policy rose from 102.38 K ops/sec at 50% reads to 130.98 K ops/sec at 95% reads. The performance increased by 28.1% since reads have smaller service times than writes. In turn, the throughput of tiering grew from 97.26 to 126.65 K ops/sec. The proposed policy surpassed tiering by just 5.12, 4.00 and 4.33 K ops/sec at three distinct read/write ratios. The random hashing had its throughput in the range of 14.51 and 16.58 K ops/sec and thus did not gain from lower service times for reads. The capacity balanced placement increased from 8.41 to 10.78 K ops/sec and stayed far behind. Consequently, write-heavy workload led to the degradation of all policies, but their order did not change.

Table 3. Robust and Parametric Comparisons Across Placement Policies at 80% Read Ratio

Metric	F	p	η²	Kruskal Wallis H	p
p99 latency (log10 scale)	63.77	< .001	.842	32.17	< .001
Throughput	3493.47	< .001	.997	34.46	< .001
Migration traffic	593832.47	< .001	1.000	34.23	< .001
Cost per TB	23454.47	< .001	.999	36.59	< .001
Peak node utilization (log10 scale)	529.75	< .001	.978	34.46	< .001

Evidence That Policy Choice Matters

The Kruskal–Wallis tests identified significant differences across all five performance measures. All results achieved $p < .001$. Throughput showed the highest rank-based test statistic ($H = 34.46$). Cost also showed significant differences ($H = 36.59$). Migration traffic ($H = 34.23$), latency ($H = 32.17$), and peak utilisation ($H = 34.46$) showed similar evidence. These findings indicate that placement policy influences system performance. The unequal variances identified by Levene's tests limit conventional ANOVA interpretation. Therefore, Kruskal–Wallis results provide the primary evidence. ANOVA results are retained as supplementary comparisons. However, significant omnibus tests do not identify which policies differ. Pairwise post-hoc comparisons are required to establish those differences.

Table 4. Normality and Variance Homogeneity Diagnostics (80% Read Ratio)

Metric	Shapiro Wilk W	p	Levene F	p
p99 latency (log10 scale)	.750	< .001	303.35	< .001
Throughput	.963	.205	8.08	< .001
Migration traffic	.939	.032	6.46	.001
Cost per TB	.831	< .001	11.78	< .001
Peak node utilization (log10 scale)	.853	< .001	15.21	< .001

Validity Checks Behind the Inference

For the normality assumption to hold, it held only for throughput, where $W = .963$, and $p = .205$. On the other hand, for the remaining 4 statistics, their p-values were .032 or below. In all 5 statistics, the results of Levene test were significant where F was between 6.46 and 303.35. One would have expected variance inequality since when it comes to random hashing, the value of SD was about 156.71 ms, compared to 0.02 ms in tiering policy. This implies that such violations imply Kruskal Wallis Tests in Table 3 and Welch tests in Table 5, which do not assume variance equality. However, ANOVA in 10 trials in each group can allow slight normality assumption violation.

Table 5. Welch t Tests Comparing the Proposed Policy With Each Baseline (80% Read Ratio)

Metric	Comparison	Mean difference	t	df	p	Cohen's d	95% CI

p99 latency (ms)	vs Random hashing	−152.54	−3.08	9.0	.013	−1.38	[−264.64, −40.44]
p99 latency (ms)	vs Capacity balanced	−377.74	−22.36	9.0	< .001	−10.00	[−415.96, −339.51]
p99 latency (ms)	vs Tiering only	−0.01	−0.66	17.9	.516	−0.30	[−0.02, 0.01]
Throughput (K ops/s)	vs Random hashing	102.67	56.03	13.0	< .001	25.06	[98.71, 106.63]
Throughput (K ops/s)	vs Capacity balanced	110.29	124.42	13.0	< .001	55.64	[108.37, 112.21]
Throughput (K ops/s)	vs Tiering only	4.00	3.49	18.0	.003	1.56	[1.59, 6.40]
Peak utilization (%)	vs Random hashing	−99.34	−7.89	9.0	< .001	−3.53	[−127.81, −70.87]
Peak utilization (%)	vs Capacity balanced	−178.16	−25.70	9.0	< .001	−11.50	[−193.84, −162.48]
Peak utilization (%)	vs Tiering only	−0.50	−3.54	17.9	.002	−1.58	[−0.79, −0.20]

Pairwise Superiority of the Proposed Policy

The proposed policy significantly outperformed random hashing and capacity-balanced placement. P99 latency decreased by 152.54 ms and 377.74 ms, respectively. Throughput increased by 102.67 and 110.29 K ops/s, respectively. Peak utilisation also decreased significantly against both baselines. Comparison with tiering alone produced different findings. P99 latency showed no significant difference ($p = .516$). However, throughput increased by 4.00 K ops/s. The Welch test reported $p = .003$. Peak utilisation decreased by 0.50 percentage points ($p = .002$). These findings suggest incremental benefits beyond tiering. The strongest additional benefit concerns throughput rather than latency.

Table 6. Tukey HSD Pairwise Throughput Differences (80% Read Ratio, K ops/s)

Policy pair	Mean difference	p
Random hashing vs Capacity balanced	7.62	< .001
Random hashing vs Tiering only	−98.67	< .001
Random hashing vs Workload aware	−102.67	< .001
Capacity balanced vs Tiering only	−106.29	< .001
Capacity balanced vs Workload aware	−110.29	< .001
Tiering only vs Workload aware	−4.00	.042

Ranking Confirmed After Multiplicity Control

Tukey HSD identified significant throughput differences across all six policy comparisons. Workload-aware placement achieved the highest mean throughput. Tiering alone ranked second, followed by random hashing. Capacity-balanced placement recorded the lowest throughput. The proposed policy exceeded tiering alone by 4.00 K ops/s. However, this comparison produced $p = .042$ under Tukey HSD. The Welch comparison produced $p = .003$. Both tests indicate significance at the 5% level. However, their different

assumptions produce different significance estimates. Therefore, the comparison requires cautious interpretation.

Table 7. Two Way ANOVA of Policy and Read Ratio Effects (df error = 108)

Outcome	Source	df	F	p	Partial η^2
Throughput	Policy	3	9444.22	< .001	.996
Throughput	Read ratio	2	214.65	< .001	.799
Throughput	Policy × Read ratio	6	55.86	< .001	.756
p99 latency (log10)	Policy	3	276.93	< .001	.885
p99 latency (log10)	Read ratio	2	2.63	.077	.046
p99 latency (log10)	Policy × Read ratio	6	0.77	.599	.041

Interaction Between Policy and Workload Mix

Policy and read ratio had an impact on throughput with $F = 9444.22$ and $F = 214.65$, respectively. Interaction is also significant ($F = 55.86$, $p < .001$), which means that the change in read ratio affects each policy differently. Tiered policies increase from 29 K ops/s to 30 K ops/s from 50% to 95% of reads, whereas random hashing is hardly affected at all. Policy is once again dominant in affecting p99 latency ($F = 276.93$, partial $\eta^2 = .885$). Both read ratio ($p = .077$) and interaction ($p = .599$) are insignificant to latency. Therefore, latencies ranking is consistent across various workload mixes. Such results confirm the assumption that the characteristics of workload affect throughput, making workload-aware approach relevant. Conclusions related to latency should also be verified through real traces and non-exponential service time distribution.

Discussion

Performance was affected heavily by the policy, and the proposed one provided an operational rate of 119.25 K ops/s at 80% reads. Random hashing and capacity balanced policies ignored the speed of devices, and therefore, disk nodes were saturated, making p99 greater than 154 ms. CacheLib is used in practice at Facebook, providing caching services with the help of memory and flash for various types of loads (Berg et al., 2020). Tiering performed the same as the proposed one in latency aspect; hence, tier awareness had a significant influence on the performance improvement. However, the suggested policy was able to increase performance by 4.00 K ops/s and decreased migration traffic by 949.05 GB to 785.40 GB. Migration cost is important because migration uses bandwidth that should be used by foreground operations when the system works. Therefore, a reduction in migration traffic by 163.65 GB is beneficial even with the slight increase in cost from 12.58 to 12.85 USD. Load balancing between the tiers and hot key replication made the peak utilization drop from 14.76% to 14.26%. Cidon et al. (2016) highlighted performance cliffs for web memory caches, therefore, allocation should consider workload properties. An increase in throughput from 102.38 to 130.98 K ops/s with higher reads shows that the structure of the workload defines the capacity. The interaction effect is clear since varying the ratio influences different policies and, therefore, any static allocation method may prove hazardous. According to Aghayev et al. (2019), storage backend technologies evolve swiftly, thus any allocation approach needs to be adaptable to the change. Lu et al. (2017) have shown that persistent memory with RDMA needs specific file systems that are optimized for device characteristics. As a result, one should implement tiering and then proceed to other techniques for efficiency gains. The findings are based on queueing theory approximation and artificial Zipf distribution and thus the real figures would differ in reality. A stronger baseline is necessary to establish the proposed policy's incremental contribution. Future evaluations should compare it against optimised tier-aware placement. Both approaches should receive identical storage capacities and network budgets. They should also experience the same workload changes and migration limits. This comparison would isolate the benefits of workload prediction. It would also establish whether the observed throughput and migration improvements persist. Naïve policies saturated at the given offered load and, therefore, could

overestimate the latency differences. Further analysis should be conducted using production traces with fault consideration and tenants' interactions.

Conclusion

In the experiment, the proposed approach was compared to three other baseline approaches when simulating a cluster of 20 nodes heterogeneous in nature. The experiment evaluated four placement policies across 20 heterogeneous nodes. The proposed approach achieved 119.25 K ops/s throughput and 1.51 ms P99 latency. It outperformed random hashing and capacity-balanced placement. However, its advantage over tiering alone was more limited. Throughput increased by 4.00 K ops/s, while migration traffic decreased by 163.65 GB. P99 latency remained almost identical between both approaches. These findings suggest that workload awareness provides additional throughput and migration benefits beyond tier awareness. The results remain limited to simulations using synthetic workloads. Production-trace evaluations are necessary to establish external validity. Future research should examine fault tolerance, tenant interference, and workload prediction. Overall, the findings indicate that workload-aware placement can improve resource utilisation without additional hardware.

References

1. Aghayev, A., Weil, S., Kuchnik, M., Nelson, M., Ganger, G. R., & Amvrosiadis, G. (2019). File systems unfit as distributed storage backends: Lessons from 10 years of Ceph evolution. Proceedings of the 27th ACM Symposium on Operating Systems Principles, 353–369. https://doi.org/10.1145/3341301.3359656
2. Annamalai, M., Ravichandran, K., Srinivas, H., Zinkovsky, I., Pan, L., Savor, T., Nagle, D., & Stumm, M. (2018). Sharding the shards: Managing datastore locality at scale with Akkio. Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, 445–460. https://www.usenix.org/system/files/osdi18-annamalai.pdf
3. Berg, B., Berger, D. S., McAllister, S., Grosof, I., Gunasekar, S., Lu, J., Uhlar, M., Carrig, J., Beckmann, N., Harchol-Balter, M., & Ganger, G. R. (2020). The CacheLib caching engine: Design and experiences at scale. Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, 753–768. https://www.usenix.org/conference/osdi20/presentation/berg
4. Cao, Z., Dong, S., Vemuri, S., & Du, D. H. C. (2020). Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook. Proceedings of the 18th USENIX Conference on File and Storage Technologies, 209–223. https://www.usenix.org/system/files/fast20-cao_zhichao.pdf
5. Cheng, Y., Iqbal, M. S., Gupta, A., & Butt, A. R. (2015). CAST: Tiering storage for data analytics in the cloud. Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, 45–56. https://dl.acm.org/doi/pdf/10.1145/2749246.2749252
6. Cidon, A., Eisenman, A., Alizadeh, M., & Katti, S. (2016). Cliffhanger: Scaling performance cliffs in web memory caches. Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation, 379–392. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/cidon
7. Kakoulli, E., & Herodotou, H. (2017). OctopusFS: A distributed file system with tiered storage management. Proceedings of the 2017 ACM International Conference on Management of Data, 65–78. https://doi.org/10.1145/3035918.3064023
8. Lu, Y., Shu, J., Chen, Y., & Li, T. (2017). Octopus: An RDMA-enabled distributed persistent memory file system. Proceedings of the 2017 USENIX Annual Technical Conference, 773–785. https://www.usenix.org/conference/atc17/technical-sessions/presentation/lu
9. Mansouri, Y., Toosi, A. N., & Buyya, R. (2017). Data storage management in cloud environments: Taxonomy, survey, and future directions. ACM Computing Surveys, 50(6), Article 91. https://dl.acm.org/doi/pdf/10.1145/3136623
10. Rashmi, K. V., Chowdhury, M., Kosaian, J., Stoica, I., & Ramchandran, K. (2016). EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding. Proceedings of the 12th

USENIX Symposium on Operating Systems Design and Implementation, 401–417. https://www.usenix.org/system/files/conference/osdi16/osdi16-rashmi.pdf
11. Serafini, M., Taft, R., Elmore, A. J., Pavlo, A., Aboulnaga, A., & Stonebraker, M. (2016). Clay: Fine-grained adaptive partitioning for general database schemas. Proceedings of the VLDB Endowment, 10(4), 445–456. https://doi.org/10.14778/3025111.3025125
12. Taft, R., El-Sayed, N., Serafini, M., Lu, Y., Aboulnaga, A., Stonebraker, M., Mayerhofer, R., & Andrade, F. (2018). P-Store: An elastic database system with predictive provisioning. Proceedings of the 2018 International Conference on Management of Data, 205–219. https://doi.org/10.1145/3183713.3190650
13. Vuppalapati, M., Miron, J., Agarwal, R., Truong, D., Motivala, A., & Cruanes, T. (2020). Building an elastic query engine on disaggregated storage. Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation, 449–462. https://www.usenix.org/conference/nsdi20/presentation/vuppalapati
14. Wu, C., Faleiro, J. M., Lin, Y., & Hellerstein, J. M. (2021). Anna: A KVS for any scale. IEEE Transactions on Knowledge and Data Engineering, 33(2), 344–358. https://doi.org/10.1109/TKDE.2019.2898401
15. Xia, M., Saxena, M., Blaum, M., & Pease, D. A. (2015). A tale of two erasure codes in HDFS. Proceedings of the 13th USENIX Conference on File and Storage Technologies, 213–226. https://www.usenix.org/system/files/conference/fast15/fast15-paper-xia.pdf
16. Yang, J., Yue, Y., & Rashmi, K. V. (2020). A large scale analysis of hundreds of in-memory cache clusters at Twitter. Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation, 191–208. https://www.usenix.org/conference/osdi20/presentation/yang