Ring: unifying replication and erasure coding to rule resilience in KV-stores

Konstantin Taranov
ETH Zurich
ktaranov@inf.ethz.ch

Torsten Hoefler
ETH Zurich
htor@inf.ethz.ch

1 POSTER SUMMARY

There is a wide range of storage schemes employed by KV-stores to ensure reliability of stored keys: from basic r-way replication, to complex erasure codes such as Reed-Solomon [1, 5]. However, previous implementations do not allow choosing the storage scheme dynamically, thereby forcing developers to commit to a single scheme [2, 3, 6]. Such inefficient data management wastes cluster resources since the way how data is stored implies completely different trade-offs between cluster resources such as memory usage, network load, latency, availability and many others. To efficiently exploit these trade-offs, we have designed a strongly consistent KV-store Ring that empowers its users to explicitly manage storage parameters like other resources, such as memory or processor time.

The key feature of Ring is that all keys live in the same strongly consistent namespace, and a user does not need to specify the resilience when looking up a key. Users can update a key's resilience requirements during the key insertion or whenever needed. We achieve this by employing a collection of containers, called memgests, offering different storage schemes. Memgests range from simple storage (no replication for fastest access), through flexible forms of Reed-Solomon (*RS*) coding, to full replication.

Architecture. The key to achieving high performance is to avoid communications during the lookup of keys with unknown resilience parameters. Therefore, Ring is decentralized by having no dedicated server for managing data allocation. In a traditional store with multiple substores with different resilience parameters, the programmer would need to contact one server in each sub-store to determine if the key exists there. We introduce our Stretched Reed-Solomon (*SRS*) codes which are constructed on top of *RS* codes. The crux of *RS* codes is inflexibility of the access interface, which is strictly defined by the number of data blocks. When the number of data blocks is changed, the client's interface has to be updated as well, since the set of coordinators serving requests has been changed. The main idea behind *SRS* codes is to ensure the same key shard allocations for a range of *RS* codes, which could have different numbers of data blocks. *SRS* coding is the key component of Ring, ensuring that a primary copy of each key only exists on a single node, even with a multitude of different resilience parameters. Therefore, even if the resilience scheme is modified, the key will be stored on the same server, which allows moving data across schemes without notifying all users about the performed operation.

API. Clients' queries are simple read and write operations to data items that are uniquely identified by a key. Ring exposes three basic KV-store operations: `get(key)`, `put(key, object)`, and `delete(key)`. It also supports `move(key, memgestID)`, which provides additional flexibility in storing the data by allowing moving keys across storage schemes. Moving the object has lower latency then putting since the client does not send the object again and its copied from local memory. Here we benefit from the fact that each node stores all primary copies of all storage schemes. Ring also allows writing the object to specified coding scheme directly with `put(key, object, memgestID)` request. To manage memgests, clients also may dynamically add and remove memgests from the system.

Applications. Ring can meet the needs of numerous applications, by allowing users to alter the storage scheme of each key-value pair on the fly at low cost. it can be run on an RDMA cluster/supercomputer as storage layer and uncovers a wide range of opportunities for optimizing performance and resource costs which lay in between known storage schemes. Clients can achieve considerable memory reduction by employing the unreliable storage scheme which does not involve any replication. The cost of the proposed memory reduction is just one move request which takes about $5\mu s$ over InifiniBand QDR network.

The unreliable memgest has the highest throughput and the lowest latency among all coding schemes since requests are not replicated within unreliable memgests. We can thus achieve a considerable speedup in throughput by moving objects to unreliable memgest and performing all write requests there. A client can deliberately move objects to a faster resilience option when it predicts high workload for them to reduce memory consumption, CPU load and network traffic. Furthermore, the client does not sacrifice reliability when the data is moved to less reliable storage, since the previous version of data can be stored in a reliable memgest. Finally, the importance of data may dynamically change over time according to the intrinsic nature of the data. For instance, in PageRank algorithm [4], the time to recover data increases with an iteration counter, since losing data at iteration n would require recomputing it from the first iteration.

We believe that explicitly managing resilience will become a standard for performance-critical applications and applicable in wider contexts as well.

REFERENCES

[1] Navin Budhiraja, Keith Marzullo, Fred B Schneider, and Sam Toueg. 1993. The primary-backup approach. *Distributed systems* 2 (1993), 199–216.
[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.* 41, 6 (2007), 205–220. https://doi.org/10.1145/1323293.1294281
[3] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. 2013. Scaling memcache at facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 385–398.
[4] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.
[5] James S Plank et al. 1997. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Softw., Pract. Exper.* 27, 9 (1997), 995–1012.
[6] Jeremy Zawodny. 2009. Redis: Lightweight key/value store that goes the extra mile. *Linux Magazine* 79 (2009).