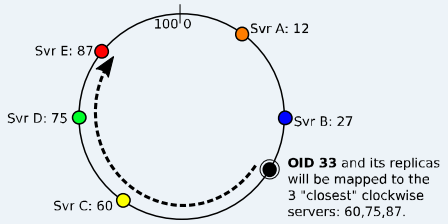
Consistent hashing distance metrics for large-scale object storage

Philip Carns, Kevin Harms, John Jenkins, Misbah Mubarak, Robert B. Ross: Argonne National Laboratory
Christopher Carothers: Rensselaer Polytechnic Institute

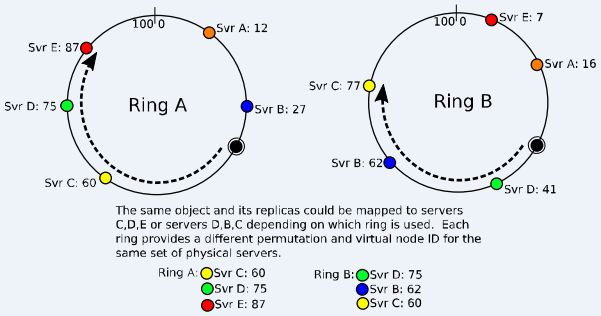
Background: Large-scale storage systems often use object-level replication to protect against server failures. Efficient algorithms are needed to map replicas to servers at scale for this approach to be effective. Consistent hashes are ideal for this purpose: they map each object to the "K closest" servers based on a distance metric and require minimal data movement on membership change. If membership is stable, then consistent hashes can be calculated locally with no communication.

***libch-placement* is a modular, open-source consistent hashing library.** We use it to investigate *distance metrics* for consistent hashing. The choice of distance metric determines:

- **Replica declustering** -> rebuild time and MTTDL
- **CPU efficiency** -> latency and bulk calculation cost
- **Object set sharding** -> aggregate HPC bandwidth



Consistent hashes are often conceptually depicted as a ring (as in Chord), with proximity measured as numerical difference between IDs. Virtual nodes can be added to improve load balancing. Hashing (e.g., in CRUSH) can be used as an alternative distance metric.

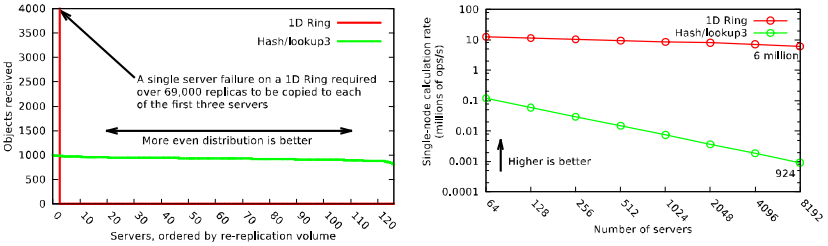


Virtual nodes are usually placed on a single conceptual ring. **Multiring hashing** instead places each virtual node for a server on a separate ring. Example shows 5 physical servers with 2 virtual rings.

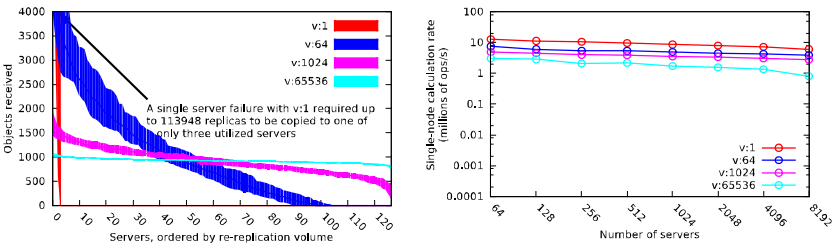
Findings:

- Virtual node approaches can dramatically increase declustering, but tens of thousands of virtual nodes may be required at scale.
- Multiring hashing balances declustring and CPU efficiency while also supporting optimal object set sharding.
- Consistent hashes are highly amenable to parallelism.
- Per-object granular placement is possible on large-scale systems.

The libch-placement library, including additional modular distance metric implementations not shown in this work, is freely available at https://xgitlab.cels.anl.gov/codes/ch-placement/

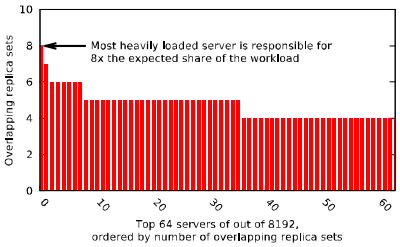


Declustering and CPU efficiency in simple placement schemes: The left graph shows how many replicas would be rebuilt on each server in a 128-server system, assuming 5 million objects, 3-way replication, and one server fault. The right graph shows the placement calculation rate on a 2 GHz AMD Opteron as system size increases. The 1D ring exhibits poor declustering but excellent CPU efficiency. The hash metric exhibits excellent declustering but poor CPU efficiency, bound by an *O(n)* per-object calculation.

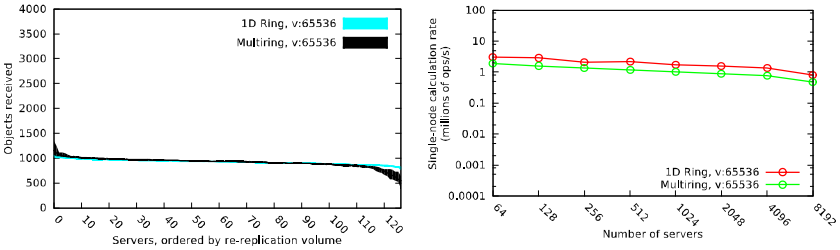


Improving declustering with virtual nodes: Virtual nodes increase the number of replica set permutations and thus increase declustering. This example shows the effect of adding V pseudo-random IDs for each server in a 1D ring. The declustering graph plots a range of values from 30 random object populations per configuration (left). 65,536 virtual nodes per physical node on a 1D ring provides similar declustering to the hash distance metric with three orders of magnitude higher CPU efficiency (right).

Object set sharding: HPC applications cannot complete a synchronous I/O phase until all operations have finished. This calls for an even distribution of data across storage resources to keep all resources engaged during I/O. Simply choosing random objects is inadequate. The figure (right) illustrates the allocation of N/R random objects in a 1D ring, each with 3 replicas, using 8,192 servers and 65,536 virtual nodes (V) per server. The resulting hot spots would hinder aggregate throughput.



Multiring hashing: a balanced approach: Unlike other distance metrics, multiring hashing can be extended to enable allocation of sharded sets of objects made up of coordinated, non-overlapping replica sets, regardless of the virtual node ratio. This is done by walking a single ring, selecting an ID from each range and skipping every R ranges, where R is the replication factor. The resulting objects are distributed evenly across all participating servers. It also retains strong declustering and CPU efficiency properties (below).



Intra-node parallelism: CPU efficiency for bulk calculation (e.g. for rebuild or scrubbing operations) can be further increased using OpenMP for intra-node parallelism. Applying this optimization to multiring hashing yields a per-server calculation rate of 4.9 million objects per second, or an aggregate calculation rate of nearly 40 billion objects per second across 8,192 servers.

The work was supported by the Office of Advanced Scientific Computer Research (ASCR) under contract DE-AC02-06CH11357. The research used resources from Argonne Leadership Computing Facility (ALCF).