JackRabbit: Improved agility in elastic distributed storage

James Cipar*, Lianghong Xu*, Elie Krevat*, Alexey Tumanov*
Nitin Gupta*, Michael A. Kozuch†, Gregory R. Ganger*
*Carnegie Mellon University, †Intel Labs

1. INTRODUCTION

Distributed storage can and should be elastic, just like other aspects of cloud computing. When storage is provided via single-purpose storage devices or servers, elasticity is useful for reducing energy usage. For storage provided via multi-purpose servers, however, such elasticity is needed to provide the cloud infrastructure with the freedom to use those servers for other purposes, which may be particularly important for increasingly prevalent data-intensive computing activities (e.g., data analytics).

Unfortunately, most distributed storage is not elastic. The Hadoop Distributed File System (HDFS) [1], for example, uses a psuedo-random data layout mainly for data availability and load balancing. But, such a data layout prevents elasticity by requiring that almost all nodes be active—no more than one node per rack can be turned off without a high likelihood of making some data unavailable.

Recent research has provided new data layouts and mechanisms for enabling elasticity in distributed storage. Most notable are Rabbit [2] and Sierra [5]. Both organize replicas such that one copy of data is always on a specific subset of servers, termed "primaries", so as to allow the remainder of the nodes to be powered down without affecting availability. Writes directed at powered-down servers are instead written to other servers—an action called "write offloading" [4]—and then later reorganized (when servers are turned on) to conform to the desired data layout.

Ideally, an elastic storage system should provide high performance, good fault tolerance, flexibility to shrink to a small fraction of servers, and the ability to quickly resize system footprint (termed "agility") with minimal data migration overhead. However, achieving all these goals at the same time is very difficult. As a result, state-of-the-art elastic storage systems usually have to make painful tradeoffs. For example, Sierra balances load across all active servers for good performance, but its system footprint can never go below one third of the total cluster size, and the migration overhead to restore the orginal data layout is significant when turning servers back on. Rabbit provides good agility by using an equal-work data layout with a very small primary set (about 10% of the cluster size). Unfortunately, however, it suffers from write performance degradation, because each primary server has to service many more writes than the others, which can them performance bottlenecks. The write performance problem can be mitigated by using the offloading technique described in Everest [4]. However, offloading writes from primaries to all the active servers incurs significant migration overhead when shrinking system size. As a result, the improved performance comes at a high cost in system agility.

We briefly overview a new elastic distributed storage system, called JackRabbit, focusing on its new policies designed to maximize the agility of elastic storage while accommodating performance and fault tolerance goals. JackRabbit builds on the Rabbit system for elastic sizing down to a small percentage of the cluster, and introduces new policies for data placement, workload distribution and data migration. For additional details and further evaluation, see [3].

2. JackRabbit POLICIES

This section describes the read/write offloading and passive migration policies used in JackRabbit.

When applications simultaneously read and write data, JackRabbit can coordinate the read and write requests so that reads are preferentially sent to higher numbered servers that naturally handle fewer write requests. By taking read work away from the low numbered servers (which are the bottleneck for writes), JackRabbit can increase write throughput without changing the offloading factor. We call this technique *read offloading*.

Write offloading policy consists of two parts—write performance offlaoding and write availabiliity offloading. Write performance offloading is a technique used when the target server is active, but is deemed to be a performance bottleneck. Rather than being sent to the server chosen by the equal-work layout agent, offloaded writes are sent to a server with low load. JackRabbit's write offloading policy has two key features. First, it bounds the set of servers for which any cleanup is

needed before extraction to as small a set as possible, given a target maximum write throughput. Second, it provides peak write throughput controlled by a parameter called the *offloading factor*, for which the value can be dynamically tuned to trade-off between maximum write throughput and cleanup work. When the equal-work layout policy tries to write a block to a host that is currently inactive, JackRabbit must still maintain the target replication factor for the block. To do this, another host must be selected to receive the write. JackRabbit load balances availability offloaded writes together with the other writes to the system. This technique is called *availability offloading*. Figure 1 illustrates the write offloading policies used in JackRabbit, when the offload set consists of $m > p$ servers.

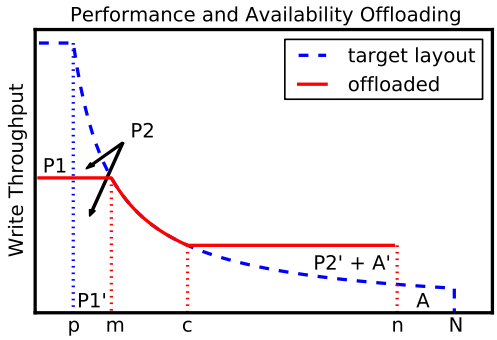

Figure 1: JackRabbit write offloading policy

Reintegrating a server that is previously deactivated is more than restarting its software. While the server can begin servicing its share of the write workload immediately, it can only service reads for blocks that it stores. Thus, filling it according to its place in the equal-work layout is part of full reintegration, which involves migrating data blocks from other servers. Instead of agressively fixing data layout before reintegrated servers begin servicing reads, JackRabbit activates more servers than needed to satisfy the read throughput requirement and utilizes the extra bandwidth for migration work. To avoid migration work building up indefinitely, the migration agent sets a time threshold so that whenever a migration takes place, it is garanteed to finish within T minutes. With T set to be larger than 1 (the default resizing interval), JackRabbit delays part of the migration work while satisfying throughput requirement. Because higher numbered servers receive more writes than their equal-work share due to write offloading, the delayed migration work can be amortized during future writes, which reduces the overall amount of data migration. This technique is called *passive migration*.

3. EVALUATION

JackRabbit is implemented as a modification of Rabbit, which itself is a modified instance of the Hadoop Distributed File System (HDFS), version 0.19.1. We evaluate JackRabbit by comparing it to Rabbit, Sierra and an imaginary "ideal" system where elastic resizing requires no cleanup work, with a trace from a real Hadoop deployment at Facebook.

Figure 2 shows the number of active servers needed as a function of time by each system, using passive migration with a migration threshold T set to be 10 minutes. The area under each curve represents the machine hour usage. As expected, JackRabbit exhibits better agility than Rabbit, especially when shrinking the size of cluster, since it needs no cleanup work until resizing down to the offload set. JackRabbit also outperforms Sierra because its minimum system footprint is well below one third of the cluster size, and it requires less migration work when restoring reintegrated servers. The read, write offloading combined with passive migration policies maximizes JackRabbit's agility and enable it to achieve a "close-to-ideal" (within 4%) machine hour usage, improving over state-of-the-art elastic storage systems by approximately 20%.

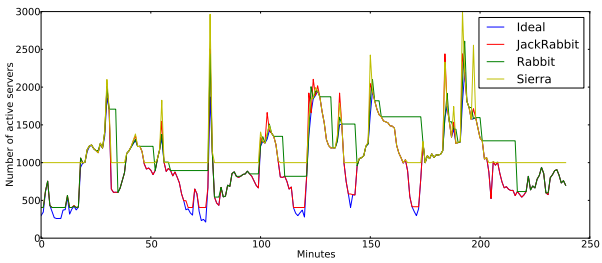

Figure 2: Machine hour usage (with Facebook trace)

4. REFERENCES

[1] Hadoop. http://hadoop.apache.org.

[2] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. *ACM Symposium on Cloud Computing*, pages 217–228, 2010.

[3] J. Cipar, L. Xu, E. Krevat, A. Tumanov, N. Gupta, M. A. Kozuch, and G. R. Ganger. Jackrabbit: Improved agility in elastic distributed storage. *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-12-112*, 2012.

[4] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through I/O off-loading. In *USENIX Conference on Operating Systems Design and Implementation*, 2008.

[5] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: practical power-proportionality for data center storage. In *EuroSys*, pages 169–182, 2011.