# Rack Aware S3G

## Introduction

In a large ozone deployment, the cluster will have several hundred datanodes storing petabytes of data. When using the S3 protocol, the cluster can’t only use a few S3G servers to serve data to thousands of compute servers as the network interface (NIC) might quickly be saturated and the read network path might not be efficient.

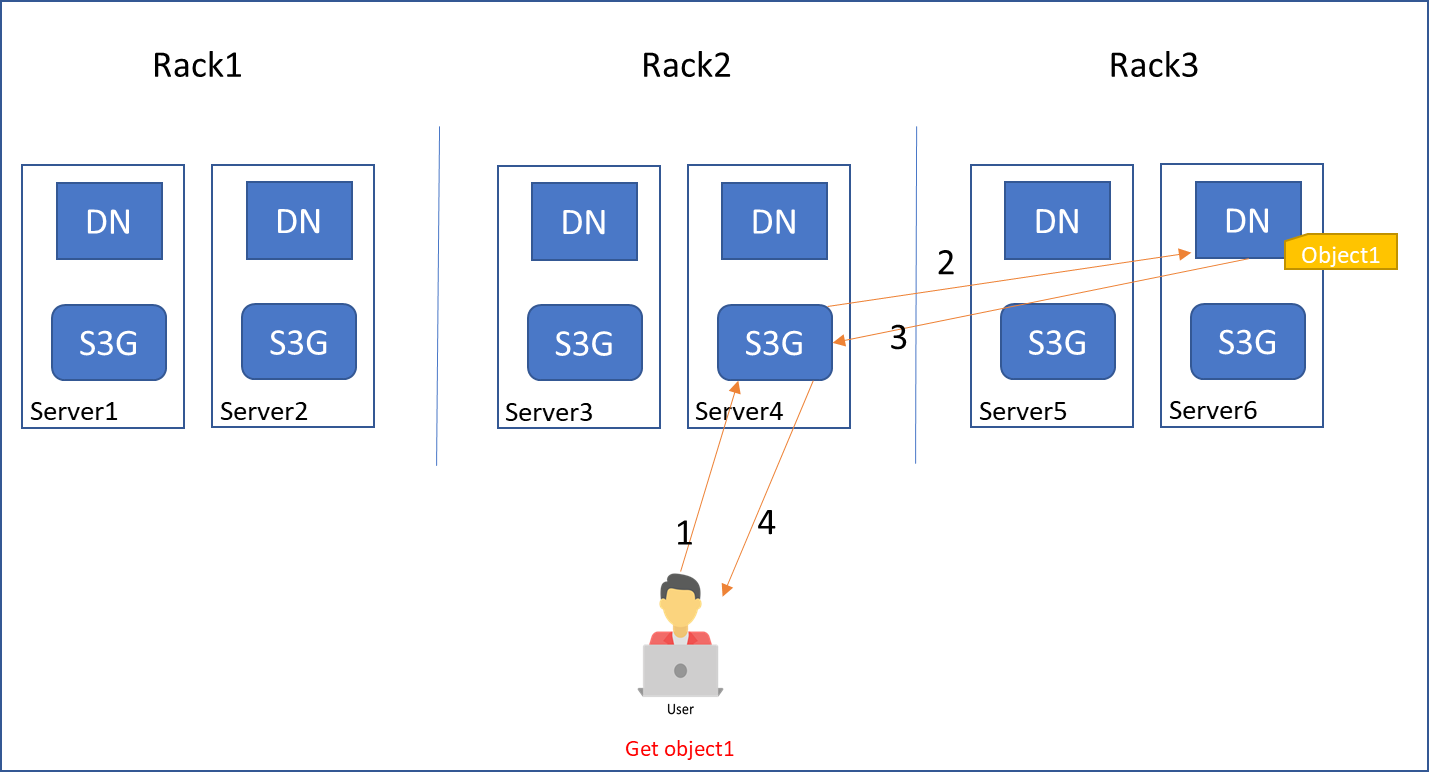
In a large production deployment:

* We will/could have an S3G installed on each datanode to be able to use the network interface bandwidth of each server.
* The servers will be spread over several racks and even over multiple halls into the data centre.

Because of the second point, we need to optimize the network reads in order to limit the network usage between halls and if possible try to limit it to intra-rack.

In the current version of ozone, when a client reaches out to an S3G to get an object, the S3G will get the data from a datanode. That datanode might be in another rack or in the worst case in another hall of the DC.

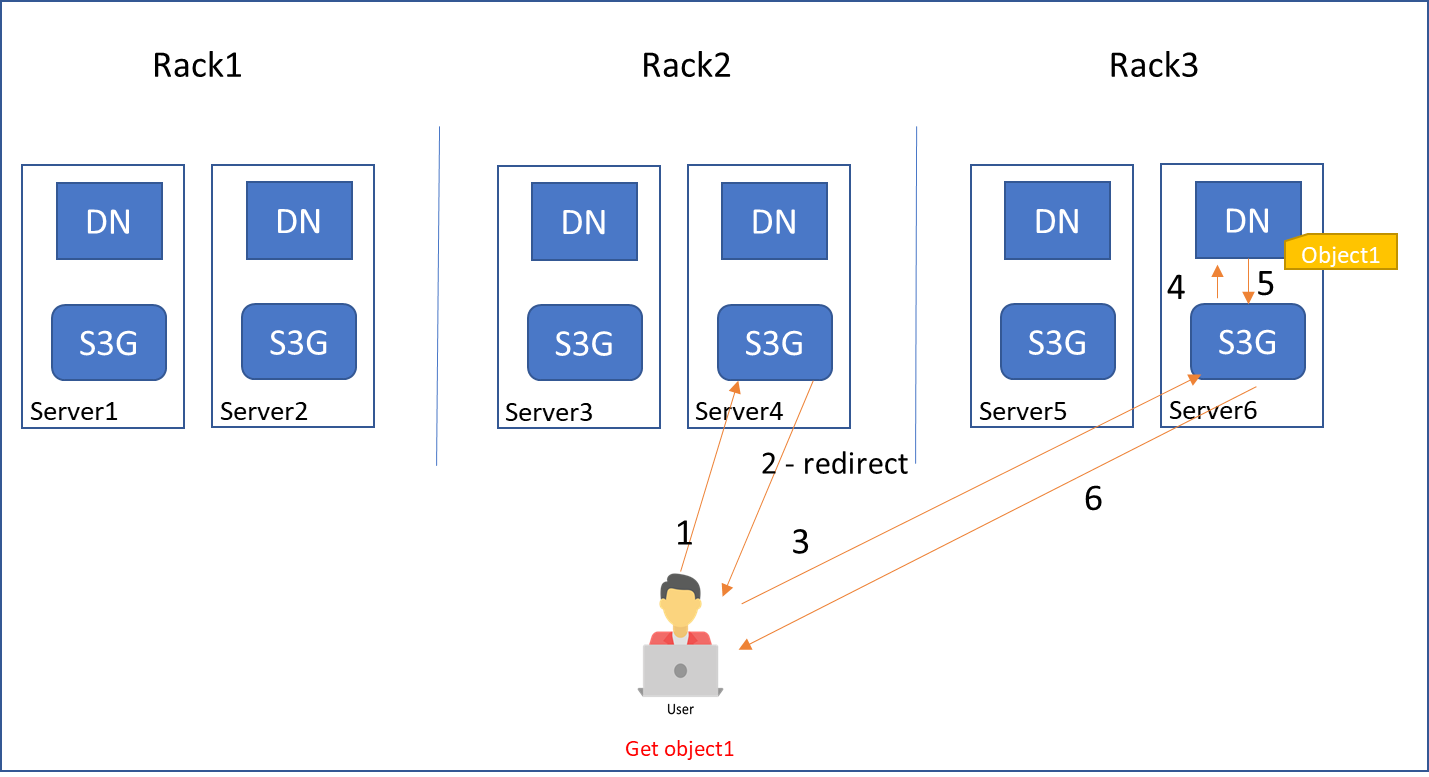
The network path of the data during a read could look like the following diagram, using the bandwidth of the network interface (NIC) of the datanode and twice the NIC of the S3G (in and out) finally arriving at the client. This is quite inefficient and generates useless network traffic.



A way to improve this, could be to use the data locality awareness and the redirection of the client request. Ozone has a notion of “rack-awareness”, and it would be nice if s3g processes were also rack-aware to reduce unnecessary intra-cluster traffic.

As shown on the following diagram,

* The client will do a get object from any S3G (here server4)
* The S3G will look at which datanode has the block for the object and redirect the request to the S3G the closest to that datanode. (here server6)
* The S3G on server6 will receive the request, look at the datanode with the object and find that it’s himself.
* The S3G on server6 will read the data locally from the datanode and forward it to the client



This provides the benefit of avoiding the network traffic between the S3G and the datanode.

If the data needs to be read from a datanode that has no S3G, then the client will be redirected to a S3G in the same rack as the datanode. If there is no S3G in the same rack then there will be no redirect of the request.

[HTTP redirect support](https://docs.aws.amazon.com/AmazonS3/latest/userguide/RESTRedirect.html) is required of s3 rest clients, so we should be able to use it in this manner.

## Implementation

### S3G registry

Enabling this feature requires a registry of the s3g processes. This registry would track the locations/racks of those processes. Each s3g could then compare incoming read requests with the location of the other s3g processes to determine whether/where to redirect the request.

The registry could be implemented in a number of ways, e.g. a dedicated “Bucket” registry, or as an extension to the OzoneManager protocol.

##### Bucket registry

There will be a dedicated bucket for the s3g registry. Each s3g process registers itself by creating a bucket entry This entry will be in the same [format](https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/RackAwareness.html) used for datanode locations, (e.g /myrack/myhost).

The entries will have no data blocks associated, (i.e. the OmKeyInfo::keyLocationVersions field will be empty.) The s3g processes will refresh them periodically to keep the last modification time up to date.

Each s3g will periodically read the bucket to track s3g processes, and to know on which rack they exist. These reads will be cached so that they don't need to happen during an s3 get request. Any older entries, (based on the modification time,) will be ignored, (and removed from the cache if they are present.)

The s3g processes can also delete older entries from the bucket, to eliminate outdated entries. If they happen to delete an entry just as the corresponding s3g comes back online, the entry will be recreated the next time that S3G refreshes.

##### Protocol-based registry

The bucket based registry requires no mods to the OM. If we are willing to extend the OM protocol, a s3g heartbeat method could be added. It could return the registry which it maintains in an in-memory cache, adding/removing entries based on the heartbeats. It could also keep the OM-followers up-to-date over ratis.

### S3 object get()

With rack-awareness enabled, the s3g will do a lookupKey() call on receiving an s3 object-get request. That call shows the datanodes on which the request’s first data blocks reside. Those datanodes would be randomized, then the get request processed as follows:

if any of the datanodes are hosting the current s3g,  
 process the request locally  
  
else if any of the datanodes is hosting an s3g,  
 redirect to it  
  
else if any of the datanodes are on the same rack as one hosting the current s3g,  
 process the request locally  
   
else if any of the datanodes are on the same rack as an s3g,  
 redirect to that s3g  
   
else process locally

## Access

The registry bucket will need an acl to limit access to s3g processes.

## Limitations

## Only uses location of first block

The approach proposed above uses the datanode of the request’s first block to target the redirect. Ozone does not appear to collocate subsequent blocks on the same datanode. If most requests are significantly larger than a block, this approach won’t reduce intranet traffic much. (The default block size is 256 megs.)

## Heartbeat induced delay

The fact that the S3G's use heartbeats to maintain their status means that there will be a delay before the peers learn one has become unavailable. During that time, requests will continue to be redirected to the unavailable node and potentially timeout if it doesn't come back online fast enough.

We felt that was an acceptable tradeoff to maintain the simplicity of the design. If not, an alternative will have to be considered. Perhaps using a dedicated registry process. All S3G's could connect to it permanently and be immediately notified should one of their peers become unavailable.

## Redirection Count Limit

Multiple redirections for a single request are not expected, but it would probably be a good idea to limit the possibility. This could be done by adding/incrementing a query parameter with each redirection. The S3G's would not redirect after the parameter reaches a threshold.

## Runtime Configuration Parameters

We probably want a way to disable redirection at runtime for maintenance/test purposes. This could be done with config data in a special bucket entry, or returned as a flag in the OM protocol.

Other possible runtime configuration parameters should probably include:

* heartbeat period
* file size threshold, (below which redirection doesn't take place).
* redirection count threshold, (number of redirections allowed for a single request.)

## What If's

* What if a client gets redirected to an S3G that went down (between 2 heartbeats)?

The client will timeout, (just like it would if the initial S3G is down.)

* What if a client gets redirected to an S3G where the datanode failed / is down?

If multiple redirections are allowed, it will redirect again. Otherwise, it will read from one of the other replicas

* What if there are 3k/5k S3G’s with a heartbeat period of 3sec, will it generate an intensive load on the OM?

Maybe. As mentioned above, the heartbeat period should probably be configurable at runtime. If the OM load is a real concern, we could introduce a separate process to maintain the registry.

* What if we need to decommission a S3G? Does it unregister itself before the shutdown?

We could do that, but the others still won't know until the end of the heartbeat period.

* What if 2 S3G's try to register themselves with the same identifier? (any security impact? An attacker trying to redirect a request to his server?)

The acl's should prevent an attacker from doing that. If it happens because one of the S3G's is incorrectly configured, the incorrectly configured node won't receive any redirects.

* What if an S3G registers himself with a timestamp from the future? (NTP issue on the S3G server)?

The last modification time is set by the OM, so the times will be consistent.