Understanding Rack-Scale Disaggregated Storage

Sergey Legtchenko Hugh Williams Kaveh Razavi* Austin Donnelly
Richard Black Andrew Douglas Nathanaël Cheriere* Daniel Fryer* Kai Mast*
Angela Demke Brown† Ana Klimovic* Andy Slowey Antony Rowstron
Microsoft Research

Abstract

Disaggregation of resources in the data center, especially at the rack-scale, offers the opportunity to use valuable resources more efficiently. It is common that mass storage racks in large-scale clouds are filled with servers with Hard Disk Drives (HDDs) attached directly to each of them, either using SATA or SAS depending on the number of HDDs.

What does disaggregated storage mean for these racks? We define four categories of in-rack disaggregation: *complete*, *dynamic elastic*, *failure*, and *configuration* disaggregation. We explore the benefits and impact of these design points by building a highly flexible research storage fabric, that allows us to build example systems that embody the four designs.

1 Introduction

Resource disaggregation decouples resources such as storage, compute and memory, allowing independent optimization and resource scaling. The cost and efficiency benefits have attracted interest from both academia and industry [11, 18, 13, 14, 12, 6, 10]. Clouds already use high-level compute and storage disaggregation, with Amazon S3 [1] and Azure Storage [5] providing storage services that are independent of compute.

In contrast, at a rack-level, the storage racks used for cloud storage are not internally disaggregated. They are composed of independent servers, each physically attached to a set of Hard Disk Drives (HDDs) or Solid State Drives (SSDs) using SATA or SAS. The number of drives per server and their type can vary (between 10 and 40 is common), and the exact configuration is based on the workload, drive capacities, network link capacities and so forth. Large cloud providers can use multiple storage rack configurations to service different workloads. In these environments, data redundancy across a set of racks or (even) data centers is preferred to server-level redundancy (like RAID) [9].

This design implies explicit ownership of each drive by the server to which it is attached. Only that owning server can issue IOs to the attached drive. There is no sharing of the physical drive between servers at the hardware-level; there is no disaggregation since there is strict ownership of physical resources. The higher layers of the software stack control data placement and choose when data written to one drive is migrated to another drive, based on age, IO rate, etc.

What could disaggregation at the rack-level mean for HDD-based cloud storage? Disaggregation breaks this fundamental strong ownership principle currently used; breaking will incur some pain, but is it worth the gain? To help us understand this we propose four different disaggregation design points for cloud storage:

Complete disaggregation: What happens if we move to an extreme view of disaggregation and assume that, at the limit, any drive can be connected to any server *per IO*? The frequency of reconfiguration will be high, potentially per-IO.

Dynamic elastic disaggregation: What happens if we assume that a drive will be connected for long enough to service multiple IOs, but the number of drives connected to a server can vary over time? The frequency of reconfiguration will be minutes to hours.

Failure disaggregation: What happens if we migrate drives to servers only on failures, to make failure handling more efficient? The frequency of reconfiguration will be days (hopefully!).

Configuration disaggregation: Can disaggregation enable service-level configuration? Reconfiguration can occur at deployment time, or if a rack is repurposed to support a different service. The frequency of reconfiguration will be long, on the order of months to years. In contrast to the other three, this would not need to be controlled by an online controller monitoring the rack load

*Work done while on an internship.
†Work done while a visiting researcher.

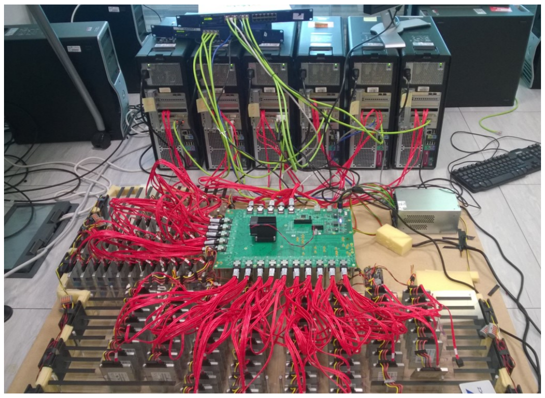

Figure 1: One of the Flexible Fabric testbeds.

or failures.

We created a new research storage fabric expressly to explore these disaggregation types. We refer to the storage fabric as the *Flexible Fabric*. It offers a generic, flexible, and reconfigurable (at a millisecond granularity) storage fabric supporting SAS and SATA PHYs. It uses a custom switch with 160 ports, and each port can be connected to a server Host Bus Adapter (HBA), a SAS expander, or a drive. The switch allows any pair of ports to be connected at the physical level. This flexibility allows us to build many different systems to explore the benefits of disaggregation. However, it should be noted that we are not advocating that the storage fabric we have built should be widely adopted or deployed (indeed, it should not be!). It is a research platform explicitly designed for efficient exploration of the functionality of future fabrics supporting disaggregation.

Before considering the four disaggregation types, we describe the Flexible Fabric and examine the core performance it provides.

2 The Flexible Fabric

The core of the Flexible Fabric is a 160-port switch, which implements a circuit switch abstraction. The switch allows *any* port to be *connected* to *any* other port. When any two ports are connected, we refer to them as being mapped; when mapped, the electrical signal received on each port is replicated at the other port. The switch supports both SAS and SATA PHYs and is transparent to all components connected to it. The switch has an on-board ARM microcontroller that runs the switch firmware and controls the switch. The firmware supports full and partial re-mapping. A full re-mapping contains all the port-to-port mappings required. A partial remapping allows incremental updates, only the ports to be remapped are specified. When doing partial remapping, any port-to-port mappings that remain unchanged experience no interruptions. Once a new configuration is loaded into the switch, it can execute a port remapping in less than 50 ns independent of whether it is a full or partial remapping. Figure 1 shows the switch in use in one of the testbeds. To control the switch, we use a simple, centralized controller that can communicate with all servers and the switch using a shared Ethernet network. The controller reconfigures and interacts with the switch. The storage fabric is a pure storage data plane.

We use two base testbed configurations: *SATA configuration* and *SAS configuration*. In the SATA configuration, SATA HDDs or SSDs and SATA HBAs are connected directly to the switch. This configuration is used in most of the experiments. In the SAS configuration, we connect a SAS HBA and SAS expanders to the switch. HDDs are connected to the SAS expanders.

As a research platform, the advantage of the Flexible Fabric is its simplicity: the switch does not need to handle PHY establishment, decoding or IO buffering. A consequence is that the PHY is established directly end-to-end between the components connected to the switch, e.g., between the HBA and the drive. This leads to PHYs being dropped when ports are remapped, which has several implications. First, the PHY needs to be re-established when two new ports are mapped together. We force the end-components to use SATA 3.0 or SAS 2.0 (or lower), which means the time taken to establish the PHY is only a few hundred microseconds (in contrast to more modern PHYs, like SAS 3.0, which can perform link quality measurements when establishing a PHY that can take as long as a second). Given the seek time for an HDD is in the order of milliseconds, a few hundred microseconds is a reasonable overhead. Second, the rapid and frequent switching of drives connected to an HBA is not a scenario widely tested by the manufacturers. We found this crashed some HBAs and drives, and others struggled to detect that the PHY had been dropped in a timely manner (delays of 5+ ms).

For the SATA configuration, after experimenting with several HBAs, we selected the Highpoint Rocket 640 Lite 4-port SATA 2.0 Internal PCI-e 2.0 x4 controller cards (which have firmware from Highpoint and a Marvell 9235 chipset). We experimented with a range of HDD and SSD manufacturers and models. For most configurations, we use OCZ TRION 100 SSDs, rather than an HDD, as this SSD handles the switching better than the other HDDs and SSDs that we tried.

For the SAS configuration used to understand configuration disaggregation, we use SuperMicro enclosures with LSI SAS 2.0 Expanders and Attotech and LSI 3008 Fury HBAs. The enclosures are populated with a mixture of Seagate and WDC Archival class HDDs [3]. We also added another SAS Expander with all its ports connected to the switch. This reconfigurable SAS topology allows

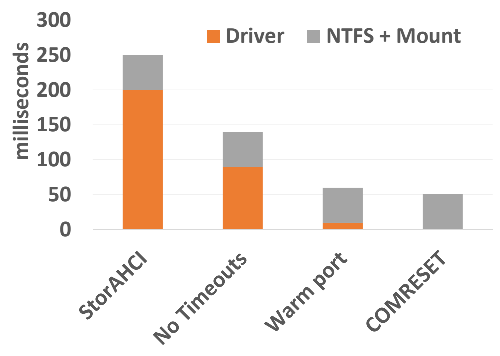

Figure 2: Switch and mount time.

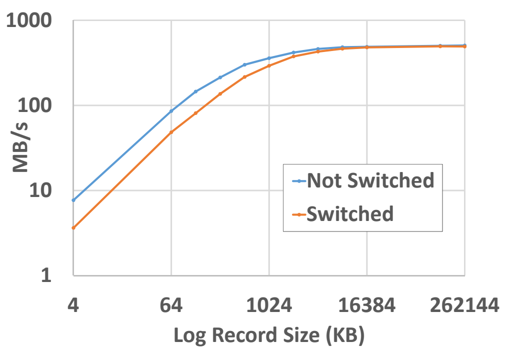

Figure 3: Switching versus static throughput.

interconnecting SAS components on demand and offers more flexibility than a traditional SAS switch could provide.

2.1 Base fabric performance

We now describe how we made the SATA configurations performant and show micro-benchmark results. The main challenge is to ensure that when a drive is attached to an HBA, we minimize the latency between remapping the ports and the drive becoming available to service IOs. To address this challenge, we modified the driver that interacts with the HBA controller card. We used the Windows StorAHCI driver, the default storport miniport driver that supports the SATA Advanced Host Controller Interface (AHCI). Figure 2 shows several configurations and the time taken for StorAHCI to register that a drive is online and mount the NTFS filesystem. In all columns, the time to mount NTFS is always approximately 50 ms; we made no modifications to NTFS. In Figure 2 *StorAHCI* shows the switch and mount time using a vanilla unmodified StorAHCI driver. It takes approximately 200ms for the drive to come online, four times as long as it takes to mount the file system. We changed StorAHCI to remove several timeouts to speed up detection of the failed link (shown as *No Timeouts*) which lowers the driver time to 90ms. In our experiments, we found that an HBA port that had a drive connected before switching to a new drive performed better. The column *Warm port* shows this, and the online latency drops to 10ms. We hypothesize that timeouts in the HBA or SSD firmware trigger faster if the drive or HBA port is already connected. The final optimization is to issue a COMRESET from the StorAHCI driver to the HBA when the server is informed by the controller that a drive is being switched. The column *COMRESET* shows that the latency drops to 1 ms. This is approximately 50% higher than the theoretical lower bound.

To understand the impact of this switching latency, we ran an experiment using a worst-case workload. It uses a single HBA SATA port and issues one read IO to an attached SSD, and when it completes, a second SSD is switched to the port. Another read IO is then issued, and the process then repeats. We compared this to running the same workload without swapping using a single SSD but again waiting for each read IO to complete. Note that we did not mount NTFS in either case.

Figure 3 shows the performance impact. On the x-axis, we show the block size of read IO issued, using a log scale. The y-axis shows the throughput in MB/sec, again using a log scale. At 4KB the throughout is approximately halved when we switch the SSD, but at 4MB and larger the throughput is comparable

It is important to remember that we are not proposing using the Flexible Fabric in a production environment. It is a research platform designed to allow us to evaluate scenarios enabled by storage disaggregation. The performance it offers is such that we can investigate and explore these scenarios. Next, we consider each disaggregation category in turn. Due to space constraints, we do not show detailed performance results.

3 Complete disaggregation

What happens if we adopt an extreme view of disaggregated storage, and assume that any IO for any drive can be serviced by any server?

Setup: Using the SATA configuration, we built a system that has n servers, each with k SATA ports connected to the switch. There are then y SSDs connected to the switch, such that $nk << y$. At any point in time, only nk SSDs can be connected to a server. An SSD can only service IO requests when it is connected to a server. The controller uses a policy to select which SSDs are connected to which servers on a *per-IO* basis. This explores the feasibility of fine-grained per-IO load balancing.

Summary: We discovered a number of key challenges with this extreme form of disaggregation. As shown in Figure 2 the file system mount times are high. Under the assumption that the server attached to the drive can change every IO, a very lightweight mount process is needed. We experimented with storing all the file system meta-data off the drive. Initially, we considered a single global shared service that would be queried when a drive was about to be attached to a server. For performance, this can be done asynchronously before the drive is physically attached, although care needs to be taken to ensure meta-data consistency if another server already has the drive attached. However, the rate of requests at this service can be on the order of IOs across the data center, and the meta-data transferred can be large.

We then designed a log-structured file system (a Lightweight File System or LWFS) that used a form of capabilities to control access. This design is block-orientated, based on the observation that many services in the data center use service-level meta-data to handle content identification. This still requires a central meta-data service, but storage servers no longer need to query the service at all. Instead, services using the storage interact with the meta-data service per-file, and then can independently issue multiple IOs to a storage server without contacting the service again. Whenever a storage server receives an IO for a drive, it can locally and independently check that the received IO is permitted using a capability token. The storage server still needs to ensure the correct drive is online and to read a key from the drive, but this requires only a single IO read when a drive comes online.

LWFS reduces the overhead, but the storage client still needs to dynamically identify the server to which the drive required for the IO is currently attached. To make this more efficient, we decided on a two-layer indirection approach. Each file is mapped to a specific rack and driveId. A rack-local service is then used to map from the driveId to the server that should be contacted to service the request. This means that a higher-level service can cache the rackId and driveId, and then only look up the driveId-to-storage-server mapping on a rack-local service.

The overhead of determining the mapping on the critical path incurs a high overhead per IO. It seems fundamental that such overheads will be needed. Looking at network bandwidth trends and hardware costs, it is difficult to come up with a convincing argument that the challenges and increased complexity and overheads justify such fine-grained per-IO remapping of drives.

4 Dynamic elastic disaggregation

What happens if we assume that a drive will be connected to a server for long enough to service several IOs, but the number of drives connected to a server can vary over time?

Setup: Using the SATA configuration, we built a system that has n servers, each with k SATA ports connected to a custom switch. There are then $nk/3$ SSDs connected to the switch. At any point in time, all drives need to be connected to a server or transitioning between servers. In this configuration, a third of the n servers are needed for all drives to be connected. So, we are examining if right-provisioning of CPU and memory resources to support a dynamic workload is useful.

Summary: This configuration significantly reduces the complexity compared to per-IO disaggregation. Drives are attached to storage servers for significant periods of time (e.g., minutes to hours). We can implement throughput proportionality of servers to workload; when the aggregate set of storage servers are underutilized (for example because of a diurnal traffic pattern), some servers can be detached from all their drives and used for non-storage tasks. The rate of drive migration between the storage servers is lower; services can cache information about which drives are attached to which storage server, reducing load and removing the need to query meta-data servers on the critical IO path. The NTFS mount overhead could also be more reasonable if the configuration frequency is on the order of hours.

Traditionally, in the cloud, isolation of storage and compute is an advantage. A cloud provider does not want random tenant workloads to potentially cause performance (or security) issues by sharing storage servers. Allowing under-utilized storage servers to be reallocated to compute could provide benefit. There is a small increase in complexity, but it can be mitigated, especially if the rate of change is dampened. Other advantages could be to reduce power-on-hours for servers that may reduce power costs and hardware failure rates.

5 Failure disaggregation

Can we use disaggregated storage to make failure handling more efficient?

Setup: Using the SATA configuration, we built a system that has n servers, each with k SATA ports connected to the switch. There are then $(nk/2)+\delta$ SSDs connected to the switch. Under no failures, there are $k/2$ SSDs connected to each server. If an SSD fails, then one of the δ unused SSDs can be used, providing rack-wide hot spares. On a server failure, the $k/2$ SSDs attached to it are redistributed across the remaining servers, either to one server, or distributed across the remaining servers. Due to growing HDD capacities, the amount of data to recover after an HDD or server failure is increasing. We explore whether disaggregation can address this issue.

Summary: The traditional approach to server failure in cloud storage is to simply assume (after some time window) that all the drives have failed and rebuild the lost files. As network bandwidth capacities increase, the number of drives per storage server will increase. Concurrently, the capacity per drive is increasing, but for HDDs the throughput is constant. Hence, the throughput per TB is dropping. If a server fails, rebuilding the data stored on it has an increasing overhead. Hence, it could be helpful to migrate the drives attached to a failed storage server to another storage server. To minimize the impact on normal operation, the traditional service infrastructure used today can be used to find the server to send an IO. On a server failure, this service can handle the change: the rate of updates to that server will be on the order of the number of storage server failures.

This approach seems interesting as there should be no real impact on the performance of the normal operation. When failures occur, the impact should be low. Given many designs of cloud storage, the drives of a failed server can be distributed across all the other storage servers in a rack, so the extra load per non-failed server can be small. As a concrete example, HDFS has DataNodes, which store data, and send Blockreports [8] describing the data blocks they store to a NameNode. On DataNode server failure, the inaccessible drives could be migrated to other DataNodes, who then just announce the blocks via a Blockreport in the usual way to the NameNode.

6 Configuration disaggregation

Can we use disaggregation to dynamically provision a rack, and infrequently reconfigure it during its lifetime?

Setup: Using the simple SAS configuration, we had n servers with k SAS ports connected to the switch, with $k/2$ SAS expanders, each connected to the switch with a single uplink. Each SAS expander can be connected to a single SAS port on any server. Each expander is always connected to a server or being transitioned between servers. Expanders are transitioned only when the rack is being reconfigured for a different role.

Summary: This is the least demanding scenario, the reconfiguration potentially happens only once when the rack is first deployed. This allows the ratio of drives to servers to be varied per service type using a single hardware rack configuration. When reconfigured the drives are assumed to be reformatted. This scenario does not require an online controller, as the configuration happens at deployment or redeployment time. We expect the configuration to hold across power cycling the rack.

The challenge is the additional cost of provisioning the storage fabric to be able to support reconfiguration, and how efficiently freed resources in the rack can be used. If a currently deployed storage stack can handle different ratios of storage servers to drives, then no software changes in the deployed environment are needed, and this would be the case, for example, with HDFS.

7 Related work

We are not aware of prior work looking specifically at building storage fabrics designed to support disaggregation *within* the rack for HDD storage. For the foreseeable future, it is expected that the majority of data will be stored in the cloud on HDDs [4]. Memory disaggregation at the rack scale has been considered extensively, from proposals for architectures [16, 17, 18, 14] to work looking at performance that would need to enable disaggregation [6, 7]. We see all this work as orthogonal to our work, where we have focused on what disaggregation could mean for HDD-based storage.

Some of the most related work is work by Klimovic et al. [11], looking at how to build efficient flash-based Network Attached Storage (NAS). This work is closely related to high-performance enterprise NAS, where the storage and the compute are separated. The focus of this work is exploring how disaggregation can be used within a rack, to support HDD-based storage for the cloud.

Some systems offer variants of failure disaggregation. For example, Pelican [2], a rack-scale design to support the storage of cool and cold data on HDDs (contrast with other cold storage rack-scale designs [15]). High-end enterprise class dual-ported SAS-drives are available, which can be used to allow access to the drive from multiple servers, to add failure resilience. However, this effectivity requires two independent storage fabrics in the rack. These two extremes motivated our interest in the benefits of low-cost disaggregation more broadly.

8 Conclusion

We have described our early experiences trying to understand rack-scale storage disaggregation. To achieve this, we have built a custom research flexible storage fabric that supports disaggregation. Our experiences with failure, dynamic elasticity, and configuration disaggregation have convinced us to design a new storage fabric that could be deployed in production storage racks.

Finally, although not in production, the switch has been deployed in a development lab to support configuration disaggregation and failure testing. This enables a single test rack to be used to instantiate multiple storage topologies, thus allowing a single rack to mimic multiple static racks. It also allows the repeatable testing of many different failure modes.

Finally, if you would like design details of our research switches, please contact us.

References

[1] Amazon s3. `https://aws.amazon.com/s3/`.

[2] Balakrishnan, S., Black, R., Donnelly, A., England, P., Glass, A., Harper, D., Legtchenko, S., Ogus, A., Peterson, E., and Rowstron, A. Pelican: A Building Block for Exascale Cold Data Storage. In *OSDI* (Oct. 2014).

[3] Black, R., Donnelly, A., Harper, D., Ogus, A., and Rowstron, A. Feeding the Pelican: Using Archival Hard Drives for Cold Storage Racks. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)* (Denver, CO, 2016), USENIX Association.

[4] Brewer, E., Ying, L., Greenfield, L., Cypher, R., and T'so, T. Disks for data centers. Tech. rep., Google, 2016.

[5] Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., et al. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 143–157.

[6] Gao, P. X., Narayan, A., Karandikar, S., Carreira, J., Han, S., Agarwal, R., Ratnasamy, S., and Shenker, S. Network Requirements for Resource Disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, 2016), USENIX Association, pp. 249–264.

[7] Han, S., Egi, N., Panda, A., Ratnasamy, S., Shi, G., and Shenker, S. Network Support for Resource Disaggregation in Next-generation Datacenters. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks* (2013), ACM, p. 10.

[8] HDFS architecture guide. `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`.

[9] Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J., Yekhanin, S., et al. Erasure coding in windows azure storage.

[10] Intel rack scale architecture. `http://www.intel.com/content/www/us/en/architecture-and-technology/rsa-demo-x264.html`.

[11] Klimovic, A., Kozyrakis, C., Thereska, E., John, B., and Kumar, S. Flash Storage Disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 29:1–29:15.

[12] Li, C.-S., Franke, H., Parris, C., and Chang, V. Disaggregated architecture for at scale computing.

[13] Lim, K., Turner, Y., Chang, J., Santos, J. R., and Ranganathan, P. Disaggregated Memory Benefits for Server Consolidation.

[14] Lim, K., Turner, Y., Santos, J. R., AuYoung, A., Chang, J., Ranganathan, P., and Wenisch, T. F. System-level implications of disaggregated memory. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture* (Washington, DC, USA, 2012), HPCA '12, IEEE Computer Society, pp. 1–12.

[15] Morgan, T. P. Facebook loads up innovative cold storage datacenter. `http://tinyurl.com/mtc95ve`, October 2013.

[16] Novakovic, S., Daglis, A., Bugnion, E., Falsafi, B., and Grot, B. Scale-out NUMA. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 3–18.

[17] Rao, P. S., and Porter, G. Is Memory Disaggregation Feasible?: A Case Study with Spark SQL. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems* (New York, NY, USA, 2016), ANCS '16, ACM, pp. 75–80.

[18] Reinhardt, S. K., and Wenisch, T. F. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *In International Symposium on Computer Architecture (ISCA*, p. 2009.