DDS: DPU-optimized Disaggregated Storage

Jiasheng Hu

Department of Computer Science
University of Toronto
jasonhu@cs.toronto.edu

1 Background & Challenges

A defining feature of cloud data systems is the disaggregation of storage and computation. In this architecture, application workloads execute on compute servers, while data is stored on dedicated storage servers that manage storage hardware and service storage requests via the network. However, existing approaches are incomplete solutions: (1) they dedicate CPU cores to issue requests and poll completions, and thus still suffer from high CPU cost; and (2) they do not reduce the overhead of network and storage routines within data systems, DPUs are SoC-based programmable NICs that are designed to offload host functionality. We characterize them as having these five components: High-speed network interface, power-efficient CPU cores, on-board memory, built-in hardware accelerators, and PCIe support.

DPU opportunities: It is possible to build an ultra efficient engine on the DPU that offloads arbitrary I/O operations, thereby saving host CPU cycles. **DPU limitations:** DPUs suffer from several constraints that prevent them from replacing the host servers in the design of cloud data systems. They have weaker and fewer cores than the host; they often have insufficient memory for data systems when serving large-scale workloads, which can lead to considerable slowdown for workloads such as page updates.

2 DDS Overview

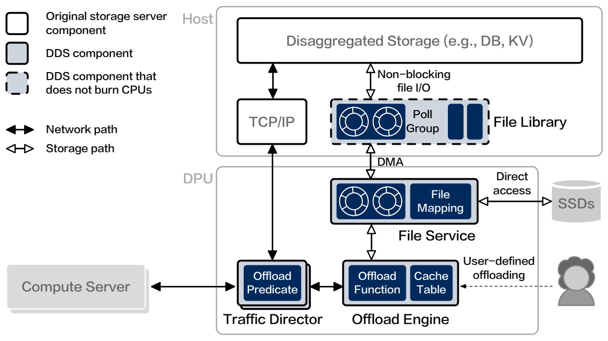

Figure 1: DDS architecture.

DDS therefore seeks to bridge the gap between the resource requirements of data system storage servers and the characteristics of DPUs with *partial offloading*. Our specific goals are: Minimal storage cost and latency, ease of adoption, and generality, shown in Figure 1. To this end, DDS introduces a storage path unifies the application's file operations on the host and those that are offloaded to the DPU. A network path then directs traffic between the host and the DPU to enable partial offloading. An offload engine supports customizable offloading of storage functions to harness the capability enabled by the storage and network paths, which takes as input a remote storage request from the traffic director, outputs a file read operation, and directly submits it to the DPU file service. This process is guided by a user-supplied *offload function*. When data is read, it responds to the client via the traffic director.

3 Evaluation

We evaluate DDS to find its CPU savings and latency reduction. We implement a storage-disaggregated application, and integrate DDS with production systems.

For reads, a storage-disaggregated application benefits from DDS by performing file I/O with its front-end library to replace the OS file system, which brings a noticeable CPU reduction—the baseline consumes 10.7 cores to achieve 390 K IOPS, but achieves 580 K IOPS using only 6.5 cores with DDS offloading. We show that DPU offloading effectively eliminates host CPU consumption as DDS can drive a sufficiently high read throughput.

Directly offloading reads to the DPU further decreases latency by avoiding the roundtrip to the host and its associated overhead. *When all of the host overhead is bypassed with DDS offloading, the latency of read requests is improved by an order of magnitude.* Specifically, the latency of the baseline achieving 390 K IOPS is 11 *ms*, while DDS incurs only 780 μs when achieving 730 K IOPS. Integration with SQL Hyperscale, a cloud-native DBMS, and FASTER, a KV-Store, also shows orders of magnitude latency reduction. This comes with the additional benefit of zero host CPU involvement, saving tens of CPU cores on the host.