An Analytical Model-based Capacity Planning Approach for Building CSD-based Storage Systems

HONGSU BYUN, SAFDAR JAMIL, JUNGWOOK HAN, and SUNGYONG PARK, Department of Computer Science and Engineering, Sogang University, South Korea
MYUNGCHEOL LEE, CHANGSOO KIM, and BEONGJUN CHOI, Electronics and Telecommunications Research Institute, South Korea
YOUNGJAE KIM, Deptartment of Computer Science and Engineering, Sogang University, South Korea

The data movement in large-scale computing facilities (from compute nodes to data nodes) is categorized as one of the major contributors to high cost and energy utilization. To tackle it, in-storage processing (ISP) within storage devices, such as Solid-State Drives (SSDs), has been explored actively. The introduction of computational storage drives (CSDs) enabled ISP within the same form factor as regular SSDs and made it easy to replace SSDs within traditional compute nodes. With CSDs, host systems can offload various operations such as search, filter, and count. However, commercialized CSDs have different hardware resources and performance characteristics. Thus, it requires careful consideration of hardware, performance, and workload characteristics for building a CSD-based storage system within a compute node. Therefore, storage architects are hesitant to build a storage system based on CSDs as there are no tools to determine the benefits of CSD-based compute nodes to meet the performance requirements compared to traditional nodes based on SSDs. In this work, we proposed an analytical model-based storage capacity planner called *CsdPlan* for system architects to build performance-effective CSD-based compute nodes. Our model takes into account the performance characteristics of the host system, targeted workloads, and hardware and performance characteristics of CSDs to be deployed and provides optimal configuration based on the number of CSDs for a compute node. Furthermore, *CsdPlan* estimates and reduces the total cost of ownership (TCO) for building a CSD-based compute node. To evaluate the efficacy of *CsdPlan*, we selected two commercially available CSDs and four representative big data analysis workloads.

CCS Concepts: • **Information systems** → **Storage architectures**; • **Software and its engineering** → *Software system models*; • **Computer systems organization** → *Distributed architectures*; Embedded software;

Additional Key Words and Phrases: Computational storage drives, solid state drives, in-storage processing, near-data processing, analytical modeling, distributed processing

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00136, Development of Big Blockchain Data Highly Scalable Distributed Storage Technology for Increased Applications in Various Industries).
Authors' addresses: H. Byun, S. Jamil, J. Han, S. Park, and Y. Kim (Corresponding author), Department of Computer Science and Engineering, Sogang University, 35, Baekbeom-ro, Mapo-gu, Seoul, South Korea, 04107; e-mails: {byhs, safdar, immerhjw, parksy, youkim}@sogang.ac.kr; M. Lee, C. Kim, and B. Choi, Electronics and Telecommunications Research Institute, 218 Gajeong-ro, Yuseong-gu, Daejeon, South Korea, 34129; e-mails: {mclee, cskim7, bjchoi92}@etri.re.kr.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1539-9087/2024/09-ART92 $15.00
https://doi.org/10.1145/3623677

ACM Reference format:
Hongsu Byun, Safdar Jamil, Jungwook Han, Sungyong Park, Myungcheol Lee, Changsoo Kim, Beongjun Choi, and Youngjae Kim. 2024. An Analytical Model-based Capacity Planning Approach for Building CSD-based Storage Systems. *ACM Trans. Embedd. Comput. Syst.* 23, 6, Article 92 (September 2024), 25 pages.
https://doi.org/10.1145/3623677

1 INTRODUCTION

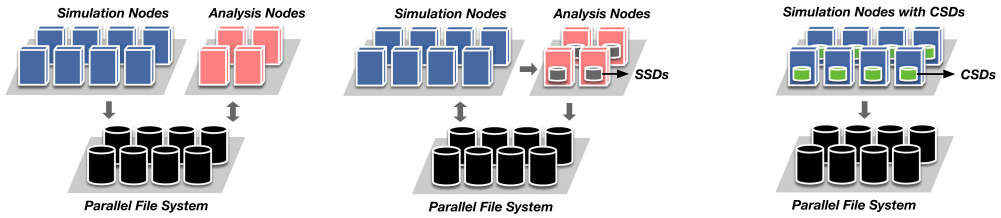
High-performance computing (HPC) simulations on large-scale supercomputers (e.g., the exascale Frontier machine [1], No. 1 on the Top500 list as of December 2022) routinely produce vast amounts of result output data [4]. Examples of such applications include astrophysics, climate, combustion, and fusion. The data generated from these applications are managed by a **parallel file system (PFS)** such as Lustre [34], as shown in Figure 1(a). Deriving insights from output data stored at PFS often involves performing a sequence of *data analysis tasks*. The data analysis tasks are performed either by a single server or a small cluster (Analysis nodes in Figure 1(a)) in an offline manner. The critical attributes required by these tasks include parallel I/O for high performance in accessing the data from storage systems. However, these tasks suffer from huge data movement costs, leading to both performance and energy inefficiencies.

To overcome this, a few solutions have been proposed to perform data analysis on a set of dedicated analysis nodes, where in-transit output data is analyzed before being written to the PFS, as shown in Figure 1(b) [46]. Although it reduces the redundant I/Os, it might cause interference at the simulation nodes[1] which leads to slowing down the simulation jobs. Importantly, it still suffers from massive data transfer between the simulation node and the analysis node. Therefore, HPC facilities have started looking at the potential of adopting storage devices within the simulation nodes, which provides an opportunity for adopting in-storage processing solutions [23]. **In-Storage Processing (ISP)** is one of the state-of-the-art paradigms that use internal resources (e.g., CPU, FPGA, and DRAM) to run data analysis tasks inside a storage device [13, 22, 24, 39, 41]. ISPs not only improve the energy efficiency of the system but also reduce the data movement between the host and storage devices. A prime example of a commercially available ISP is the **Computational Storage Drive (CSD)**. Recently, SK Hynix and **Los Alamos National Laboratory (LANL)** have demonstrated the world's first **Key-Value Computational Storage Device (KV-CSD)** to accelerate data analysis tasks of HPC simulations [2].

Moreover, several vendors have introduced commercial CSDs, including Samsung's SmartSSD [36], NGD system's Newport CSD [32], and ScaleFlux's Computational Storage [37]. The adoption of CSD within simulation nodes will play a vital role in analysis nodes where data analysis tasks can be offloaded to CSDs. Figure 1(c) shows a representative HPC system where each simulation node has local CSD(s). However, adopting CSDs naively does not benefit due to the distinct hardware and performance characteristics of commercially available CSDs. A typical hardware architecture of a CSD embeds an accelerator (FPGA) or an embedded CPU within a storage device to perform analysis tasks. CSDs can be classified based on the support of the operating system on top of the device. For instance, Newport SSD of the NGD systems [32] runs an embedded OS, whereas SmartSSD [36] does not. A CSD with built-in support of an embedded OS runs the analysis task from user space and benefits from the ease of programmability and manageability through exploiting the traditional features of OS, such as supported libraries, multitasking, and well-defined hardware abstractions.

On the other hand, a CSD, without OS support, benefits from executing the analysis kernel directly on the FPGA accelerator, just like a bare-metal application, and avoiding the software

[1]Simulation nodes are the compute nodes, and we will use these terms interchangeably from here after.



(a) Traditional approach (b) In-transit approach (c) In-situ approach using CSD

Fig. 1. Three approaches for HPC workload processing: (a) traditionally using simulation node and separate analysis node, (b) in-transit approach where analysis nodes play a role of staging area and perform data analysis tasks, (c) integrating CSD into simulation node instead of analysis node [39].

overhead caused by OS. The kernel developers with these CSDs can design and develop their analysis kernels with the supported platform. For instance, an analysis kernel for SmartSSD [36] to be executed on the FPGA accelerator is implemented using OpenCL programming at the Vitis platform [11] provided by Xilinx. CSDs are packaged in the same form factor as regular SSDs and can easily replace traditional block-based SSDs. Several works [40–42] attempted to build a CSD-based storage system and executed big data applications using CSDs and showed their performance benefits. However, the performance characteristics of CSDs vary from vendor to vendor thus, adopting CSDs becomes a challenging task for storage architects within HPC facilities. For instance, storage architects have to decide whether to adopt CSDs with an embedded ARM processor (Newport CSD) or an FPGA-based accelerator (SmartSSD), as both CSDs have different computational power and programming interfaces. Moreover, the internal and external I/O bandwidth vary significantly depending on the interconnect and network protocol implementation (for more details, refer to Section 2.3). Furthermore, the performance efficiency of CSDs is highly dependent on the nature of the workload (from being compute-intensive to I/O-intensive). Previous works put strenuous efforts into identifying the optimal number of CSDs to meet performance requirements for specific workloads [40, 41], thus, making the adoption of CSDs even more challenging.

Therefore, in this work, we propose a model-based storage capacity planning tool (*CsdPlan*) which allows storage designers/architects to find the **break-even point (BEP)** effectively without having to run experiments on all storage combinations manually. To the best of our knowledge, this work is the first to find workload-specific BEP using real commercial CSDs. This paper makes the following specific contributions.

- We propose an analytical model-based capacity planner for *CsdStore*, called *CsdPlan*. *CsdPlan* finds the optimal number of CSDs in *CsdStore*, where *CsdStore* outperforms a traditional approach. Our model can be extended and adopted for large-scale systems with multiple compute nodes over the network.
- We developed a mathematical model for the *CsdPlan*, which takes the performance characteristics of the CSD and the workload patterns of the applications as input and provides the optimal number of CSDs required to outperform the traditional compute node with SSDs. The *CsdPlan* can give the "rule-of-thumb" to storage architects/administrators of *CsdStore* while making storage capacity planning decisions.
- We performed an extensive evaluation of our proposed *CsdPlan* to account for various hardware characteristics, such as computing power and I/O bandwidth with several real-world workloads. Specifically, *CsdPlan* finds the optimal break-even point (BEP) for big data analysis workloads. For instance, the BEP for Vector Addition workload would be 5 for SmartSSDs, while it would be 1 if the computing power of the FPGA accelerator is significantly improved, like 5×.

- We studied the **total cost of ownership (TCO)** savings in *CsdStore* using *CsdPlan*. For example, in the Array Merge workload, *CsdPlan* suggests that the traditional compute node with 12 SSDs and the *CsdStore* with 2× slower CPU and 6 Newport CSDs have the same throughput. According to our CSD pricing assumption, *CsdStore* can reduce CPU and storage costs by up to 55%, compared to a traditional approach.

This paper is organized as follows. Section 2 introduces the background knowledge of CSD and the motivation for our proposed *CsdPlan*. Section 3 shows an overview of *CsdStore* and the mathematical analysis model of *CsdPlan*, and Section 4 analyzes the performance characteristics of CSD and extensive evaluation results of *CsdPlan*. Finally, Section 5 gives the conclusion.

2 BACKGROUND AND MOTIVATION

In this section, we present the background of **computational storage drives (CSDs)**, the related work for storage capacity provisioning, and the motivation for this study.

2.1 Computational Storage Drives

In-Storage Processing (ISP) uses the SSD's internal hardware resources such as CPU and memory for out-of-core execution inside the SSD [15, 18, 21, 22, 27, 28, 30, 35, 44, 45]. The ISP not only frees up CPU and memory resources on the host, but also reduces the cost of moving data between the host and the device. SSDs that support ISPs are called CSDs. In the meantime, there have been many studies on the design and performance optimization of CSDs. Biscuit [18] facilitates development by defining protocols for near-data processing using the ISP and supporting a full-featured standard library and the latest C++ standards. Willow [38] has extended the meaning of SSD to a function that applications can use without damaging the file system by adding a programmable feature to the storage device. Specifically, several researches investigated to accelerate database applications with ISP on SSDs. FCAccel [44] integrated a column-oriented field-programmable-gate-array-based acceleration engine into an SSD to offload SQL operators to the SSD. Aquoman [45] is an SSD with a general analytic query processor, prototyped in an FPGA for ISP on the SSD. Smart SSD [22] provides MapReduce [16] framework that can execute a user's customized job or database query inside SSD.

2.2 Storage Capacity Provisioning

There have been several storage capacity provisioning studies that cost-effectively design compute nodes using SSDs instead of HDDs [14, 25, 26, 31]. Among these, especially Narayanan et al. [31] investigated the role of SSDs in enterprise compute nodes using multiple real-world data-center traces. Their work explores the cost-benefit trade-offs of various SSD and HDD configurations. Kim et al. [25] investigated the problem of finding the optimal storage configuration for compute nodes employing both SSDs and HDDs while meeting performance requirements. They also studied the issue of designing the dynamic placement of workload in the hybrid storage configurations. On the other hand, our work is different from these works. We explored the problem of cost-benefit trade-offs of various CSD configurations for big data workloads. Since CSDs are much more complex devices than SSDs, finding the design requirements, such as the number of CSDs when building the CSD-array-based computational node is not an easy task. We explored the computational and I/O processing performance of commercial CSDs. We found that (i) the performance spectrum of CSDs is very broad (not comparable to SSDs), and (ii) even CSDs have different performance trends. More details on these are given in the next section.

Table 1. Hardware Specifications of Representative CSDs

	ISP Engine	External I/O BW	Internal I/O BW
Smart SSD[2] [43]	ARM Cortex R4 @ 400 MHz	0.55 GB/s	1.5 GB/s
Willow [38]	FPGA @ 125 MHz	2 GB/s	4 GB/s
Biscuit [18]	ARM Cortex R7 @ 750 MHz	3.2 GB/s	4 GB/s

2.3 Motivation

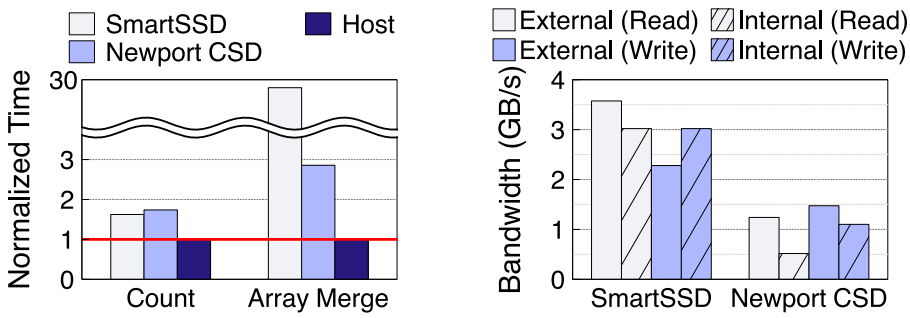
Table 1 shows the hardware specifications of representative CSDs. In Table 1, all CSDs have different computing engines (e.g., FPGAs or low-power CPUs), but they all have much higher internal I/O bandwidth than external I/O bandwidth. However, the real performance of the kernel of applications is dependent on the workload patterns, that is, depending on whether the kernel is compute-intensive or I/O-intensive. Additionally, the application's I/O bandwidth also depends on whether it is external I/O or internal I/O. To verify these, we conducted several experiments and measured the execution times and I/O bandwidths over two commercial CSDs (SmartSSD and Newport CSD).

Figure 2 shows the results of the execution times for two represented analysis kernels, Array Merge and Count. A detailed description of the analysis kernels and the input data size are provided in Section 4. In Figure 2(a), the execution times are normalized with the host system using a single CPU core and equipped with traditional SSD. For Count, we observe that SmartSSD exhibits a slightly lower execution time than Newport CSD. Also, we observe that the execution times of SmartSSD and Newport CSD were about 1.6× and 1.7× worse than that of the host system, respectively. Through this, it can be seen that the performance difference between FPGA and ARM processors is small for the Count kernel. On the other hand, for Array Merge, we observe that the execution times of SmartSSD and Newport CSD were 28× and 2.8× higher than that of the host, respectively. SmartSSD showed significantly higher execution time compared to the host as well as the Newport CSD. In the Array Merge kernel, the reason why CSD has a much higher execution time than the host system is that Array Merge is a more CPU-bounded workload than Count and requires a system with high computational power. On the other hand, CSD is a system with significantly lower computational power than the host CPU. Since the Count kernel is an I/O-intensive workload, the execution time is not significantly different from that of the host system.

Next, we measured the internal and external bandwidths of Newport CSD and SmartSSD. To measure the external I/O bandwidth of each CSD, we ran the FIO benchmark [12] on the host. However, due to the difference in hardware design between SmartSSD and Newport CSD, the internal I/O bandwidth measurement method is as follows: For the Newport CSD, to measure internal and external I/O bandwidth, we performed an FIO benchmark [12] on the host and CSD side. The FIO benchmark was configured by using the libaio engine[3], direct option on, 1 MB request size, 64 queue depth, and sequential pattern. For the SmartSSD, for internal I/O bandwidth measurement, we used a bandwidth measurement kernel program [6] with a request size of 64 MB. Figure 2(b) shows the I/O bandwidth measurements for SmartSSD and Newport CSD. We define R_{tx} as the ratio of the internal I/O bandwidth to the external I/O bandwidth ($R_{\text{tx}} = \frac{BW_{\text{Internal}}}{BW_{\text{External}}}$). If R_{tx} is greater than 1, it means that the internal I/O bandwidth is higher than the external I/O bandwidth.

In the results for SmartSSD, we observe that for reads, the external bandwidth is about 1.18× higher than the internal bandwidth ($R_{\text{tx}} = 0.84$), but, for writes, the internal bandwidth is about

[2]Here, Smart SSD is the research prototype name of the cited paper, which is different from SmartSSD, a commercial CSD.
[3]Linux-native asynchronous I/O access library



(a) Execution Time **(b)** I/O Bandwidth

Fig. 2. (a) Comparison of execution times of Count and Array Merge workloads against CSDs and host CPU. Execution time is normalized to the execution time of the host system.(b) Comparison of internal and external I/O bandwidths of CSDs.

1.36× higher than the external bandwidth (R_{tx} = 1.36). This is due to the hardware limitations of the SmartSSD, the maximum internal bandwidth is bound to the bandwidth of the PCIe bus connecting the SSD and the FPGA. In the results for Newport CSD, we observe, surprisingly, that the external bandwidth is 2.28× and 1.33× higher than the internal bandwidth for both read and write workloads, respectively (R_{tx} = 0.43, 0.75). The NGD system explained that the internal components of Newport CSD (DRAM, NAND, etc.) are connected through high-speed interconnect but did not disclose detailed hardware specifications [17]. These results show a different observation from the observations in the literature where the internal I/O bandwidth of CSD is higher than the external I/O bandwidth [18, 38, 39, 43].

To summarize our observations, each CSD shows different performances depending on their computing power, I/O bandwidth, and workload characteristics. Therefore, when building a compute node based on a CSD-array, storage architects should design in careful consideration of each SSD's hardware and workload characteristics. Therefore, this study aims to offer a software tool that provides storage architects with guidance when building the CSD-array-based node in terms of the optimal number of CSDs for CSD-based compute nodes.

3 CAPACITY PLANNING FOR *CSDSTORE*

This section presents an overview of *CsdStore* and details of how to build a *CsdPlan* for *CsdStore* and how system architects/administrators can use it.

3.1 Overview of *CsdStore*

CsdStore is a cluster of CSDs and leverages the compute capabilities of each CSD to provide better performance than the traditional storage server approach. Several recent studies have shown the potential of CSD-array for HPC, big data and AI workloads [17, 40, 42]. However, one of the main problems for system architects with CSD-arrays is the lack of planning tools for how to efficiently build CSD-array based on their requirements. For instance, with varying computation resources, system architects are not able to identify the number of CSDs to be installed in a CSD-array. Thus, it is possible that a system architect might over- or under-provision the CSD-array's resource when designing a cluster.

CsdStore provides *CsdPlan* as a software tool to the system architects. *CsdPlan*, which provides guidelines on how to efficiently build a compute node using CSDs, on a limited budget. We envision *CsdPlan* to be a tool that would enable system architects to provision the CSD-array

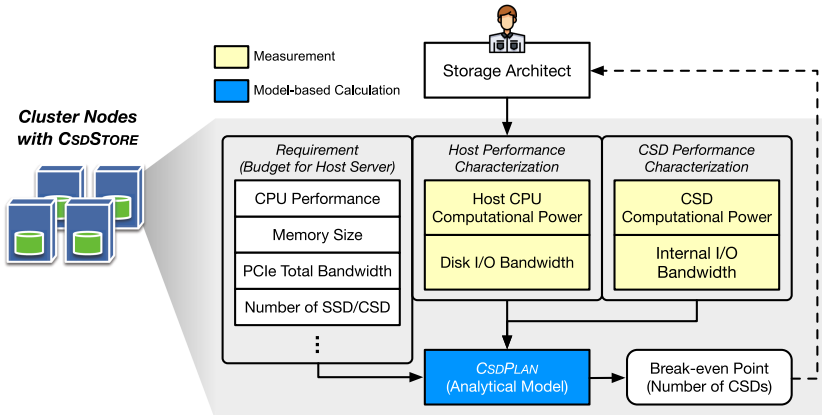


Fig. 3. *CsdStore* and *CsdPlan*'s overview description. Yellow boxes indicate values that system architects should measure.

based computational node in performance-effective ways. Figure 3 depicts an overview of the *CsdPlan* where a system architect is required to measure the initial performance characteristics of the host and CSDs. *CsdPlan* employs a break-even point (BEP) decision-maker based on mathematical formulations to make its storage provisioning decisions. *CsdPlan* finds the BEP where the performance of data analysis in a CSD-array configuration outperforms the traditional storage server approach. system architects can understand the effectiveness of the CSD-array system by considering both the BEP found with *CsdPlan* and the budget to be used for the storage server implementation. For example, with *CsdPlan*, a *CsdStore* with higher performance than a traditional storage server can be built at a lower cost.

3.2 *CsdPlan*: Capacity Planning

CsdPlan is a software module that provides guidelines to system architects when building a computational node. *CsdPlan* takes input as the computation and I/O performance parameters of a CSD for applications and outputs the minimum number of required CSDs for *CsdStore*. *CsdPlan* required the following two steps to be performed by the system architect:

- **Step 1:** The system architect selects the CSD to be deployed at the CSD-array and targets applications that will be running on that cluster. As the *CsdPlan* takes the performance characteristics as input, thus a system architect is required to measure the performance of the CSD to obtain the computational and I/O processing capabilities of the corresponding CSD.
- **Step 2:** Once the performance characteristics are obtained from Step 1, the system architect inputs them to the *CsdPlan*, which determines the minimum number of CSDs required to achieve optimal performance on a CSD-array that is higher than the traditional storage server approach.

Our *CsdPlan*'s solver is built on top of mathematical system modeling and is described in detail in the following subsection.

3.2.1 System Modeling. In this subsection, we provide details of performance modeling of *CsdPlan* for SSD system and CSD system as follows: We first define a system equipped with a single device:

- **SSD system (n):** A traditional compute node where a host is equipped with a single block-based SSD and uses the host n cores and memory for analysis.

- **CSD system:** A compute node equipped with commercial CSDs where it utilizes the CSD's resources for analysis.

The execution time is considered the performance metric in our model.

The execution time of the SSD system (n) ($T_{\text{SSD}(n)}$) for the kernel (W) can be modeled as the sum of the data transfer time ($T_{\text{SSD-tx}}$) and the computation time ($T_{\text{SSD}(n)\text{-comp.}}$). Thus,

$$T_{\text{SSD}(n)} = T_{\text{SSD-tx}} + T_{\text{SSD}(n)\text{-comp.}} \tag{1}$$

We assume that the host system is comprised of n cores and the workload is equally divided between all the cores. The data transfer time stays the same regardless of the number of cores in the host system. The sequential execution time is presented as $T_{\text{SSD-comp.}}$, thus according to Amdahl's law [19], the execution time over n cores would be:

$$T_{\text{SSD}(n)\text{-comp.}} = \frac{1}{n} \cdot T_{\text{SSD-comp.}}, \ n \leq \text{Max Cores} \tag{2}$$

On the other hand, the performance model for the CSD system will be:

$$T_{\text{CSD}} = T_{\text{CSD-tx}} + T_{\text{CSD-comp}}. \tag{3}$$

Moreover, the resources of the host system are classified as *normal* and *overloaded* based on the workload. For instance, if the kernel is being executed in parallel to other applications at the host system/machine and the execution time of the kernel is greatly affected. This is due to resources being shared between the kernel and other applications and thus leading to resource contention. We call this situation an *overloaded* condition. On the other hand, if the kernel is being executed with the desired resources from the host machine, it will be considered a *normal* condition.

Applications running parallel to the kernel are categorized as: CPU-, data-, or memory-intensive. If an application is CPU-intensive, then the computational resources are exhausted, thus affecting the computation time of the kernel. On the other hand, if an application is data-intensive, then the I/O resources are being shared, thus leading to significant increase in data transfer time. However, if an application is memory-intensive, then both computation and data transfer times are affected due to high I/O overhead by frequent disk swapping in the virtual memory system. Therefore, we extend the execution time model of each system in overloaded conditions by applying the *slow-down factor* (the rate of increase in time) to the computation time and data transfer time. Assume that the slow-down factors of data transfer time and computation time are sd_{tx} and $sd_{\text{comp.}}$, respectively. The execution time of the SSD system can be modeled as follows:

$$T_{\text{SSD}(n)} = sd_{\text{tx}} \cdot T_{\text{SSD-tx}} + sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}} \quad (sd_{\text{tx}}, \ sd_{\text{comp.}} \geq 1) \tag{4}$$

The CSD systems, unlike SSD systems, run the kernel on CSD, so their execution time is not affected by overload conditions. Therefore, the slow-down factor for each term of the CSD system is always 1. Thus, the execution time of the CSD system is simply modeled as to Equation (3).

Now, we extend this performance model to a system comprised of an array of devices. We assume that the data required for workload execution is uniformly distributed and stored in the device array. We consider two systems as follows:

- **SSD-array system (n)** ($T_{\text{SSD}(n)\text{-array}}$): The host is equipped with an array of block-based SSDs and uses the host CPU's n cores and memory to run analysis kernels.
- **CSD-array system** ($T_{\text{CSD-array}}$): The host is equipped with an array of CSDs and uses the CSD's CPU and memory instead of the host's resources to run the analysis kernel.

An array of m SSDs can theoretically reduce the data transfer time to $\frac{1}{m}$ until the bus connected to the host becomes a bottleneck [29] Therefore, we set the number of SSDs (M_{SSD}) as k_{limit} so that

the disk I/O becomes a bottleneck in the SSD-array system. Thus, the SSD-array system model is extended as follows:

$$T_{\text{SSD}(n)\text{-array}} = \frac{1}{M_{\text{SSD}}} \cdot sd_{\text{tx}} \cdot T_{\text{SSD-tx}} + sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}} \tag{5}$$

$$M_{\text{SSD}} = \begin{cases} m & \text{if } (m < k_{\text{limit}}) \\ k_{\text{limit}} & \text{else} \end{cases} \tag{6}$$

Unlike the SSD-array system, since each CSD has the computational capability, both the data transfer time and computation time of the CSD-array system are reduced to $\frac{1}{m}$. In addition, since each CSD does not share the connected bus, there is no bottleneck as the number of CSDs increases. The CSD-array system model is extended as follows:

$$T_{\text{CSD-array}} = \frac{1}{M_{\text{CSD}}} \cdot T_{\text{CSD-tx}} + \frac{1}{M_{\text{CSD}}} \cdot T_{\text{CSD-comp.}} \tag{7}$$

3.2.2 Solver: Finding the Break-Even Point. CsdPlan deploys a solver to find the BEP for the number of CSDs in a CSD-array-based compute node. Our proposed solver takes the performance characteristics of the CSDs as input and generates an optimal number of CSDs as output. This optimal number of CSDs is referred to as the BEP, where the CSD-array will outperform the traditional compute node. Therefore, we derive the mathematical model of ($T_{\text{SSD}(n)\text{-array}} > T_{\text{CSD-array}}$) as follows:

$$T_{\text{SSD}(n)\text{-array}} > T_{\text{CSD-array}}$$

$$\Rightarrow \frac{sd_{\text{tx}}}{M} \cdot T_{\text{SSD-tx}} + sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}} > \frac{1}{M} \cdot T_{\text{CSD-tx}} + \frac{1}{M} \cdot T_{\text{CSD-comp.}}$$

$$sd_{\text{tx}} \cdot T_{\text{SSD-tx}} + M \cdot sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}} > T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} \quad \text{(Multiply both sides by } M\text{)}$$

$$M \cdot sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}} > T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - sd_{\text{tx}} \cdot T_{\text{SSD-tx}} \quad \text{(Subtract } sd_{\text{tx}} \cdot T_{\text{SSD-tx}} \text{ from both sides)}$$

$$M > \frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - sd_{\text{tx}} \cdot T_{\text{SSD-tx}}}{sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}}} \quad \text{(Divide both sides by } sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}}\text{)}$$

$$M_{\text{CSD}} > \frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - sd_{\text{tx}} \cdot T_{\text{SSD-tx}}}{sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}}} \quad \text{(To find BEP, } M = M_{\text{SSD}} = M_{\text{CSD}}\text{)}$$

Therefore,

$$M_{\text{CSD}} \geq \left\lceil \frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - sd_{\text{tx}} \cdot T_{\text{SSD-tx}}}{sd_{\text{comp.}} \cdot T_{\text{SSD}(n)\text{-comp.}}} \right\rceil^{4} \tag{8}$$

If the host resource is not overloaded, $sd_{tx} = 1$, $sd_{comp.} = 1$, then the following holds.

$$M_{\text{CSD}} \geq \left\lceil \frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} \right\rceil \tag{9}$$

Impact of Computing and I/O Performance: The increase or decrease of the computational power of CSD determines the change in kernel execution time. Additionally, the internal I/O bandwidth of the CSD can be higher or lower than the external I/O bandwidth. CSD's computing power and internal I/O bandwidth are determined by how the device is manufactured. In Table 1, the internal I/O bandwidth of CSD is higher than the external I/O bandwidth. On the other hand, as shown

[4]($\lceil\ \rceil$) is least integer function

in Figure 2(b), the internal I/O bandwidth of CSD can be lower than the external I/O bandwidth. Therefore, we model the BEP (N_{CSD}) according to the change of CSD's computational power and internal I/O bandwidth as follows.

$$S(T_{\text{CSD-tx}},\ T_{\text{CSD-comp.}}) = \left\lceil \frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} \right\rceil \tag{10}$$

To simplify the formula, we define the following ratios:

$$R_{(n)\text{comp.}} = \frac{T_{\text{SSD}(n)\text{-comp.}}}{T_{\text{CSD-comp.}}},\ R_{(n)\text{SSD}} = \frac{T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}}$$

Then,

$$\frac{T_{\text{CSD-tx}} + T_{\text{CSD-comp.}} - T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} = \frac{T_{\text{CSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} + R^{-1}_{(n)\text{comp.}} - R_{(n)\text{SSD}}$$

We assumed that the SSD system uses the CSD as a block device.

$$R_{\text{tx}} = \frac{BW_{\text{Internal}}}{BW_{\text{External}}} = \frac{\text{Data Size}/BW_{\text{External}}}{\text{Data Size}/BW_{\text{Internal}}} = \frac{T_{\text{SSD-tx}}}{T_{\text{CSD-tx}}},\quad T_{\text{CSD-tx}} = R^{-1}_{\text{tx}} \cdot T_{\text{SSD-tx}}$$

Then,

$$\begin{aligned}
&\frac{T_{\text{CSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} + R^{-1}_{(n)\text{comp.}} - R_{(n)\text{SSD}} \\
&= \frac{R^{-1}_{\text{tx}} \cdot T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}} + R^{-1}_{(n)\text{comp.}} - R_{(n)\text{SSD}} \quad \left(T_{\text{CSD-tx}} = R^{-1}_{\text{tx}} \cdot T_{\text{SSD-tx}}\right) \\
&= R^{-1}_{\text{tx}} \cdot R_{(n)\text{SSD}} + R^{-1}_{(n)\text{comp.}} - R_{(n)\text{SSD}} \quad \left(R_{(n)\text{SSD}} = \frac{T_{\text{SSD-tx}}}{T_{\text{SSD}(n)\text{-comp.}}}\right) \\
&= \left(R^{-1}_{\text{tx}} - 1\right) \cdot R_{(n)\text{SSD}} + R^{-1}_{(n)\text{comp.}} \quad \text{(Distributive Law)}
\end{aligned}$$

Therefore, Equation (8) is transformed as follows.

$$S(R_{\text{tx}},\ R_{(n)\text{comp.}}) = \left\lceil \left(R^{-1}_{\text{tx}} - 1\right) \cdot R_{(n)\text{SSD}} + R^{-1}_{(n)\text{comp.}} \right\rceil \tag{11}$$

Finally, the BEP cannot be smaller than 1, so set the minimum value to 1 as follows.

$$S(R_{\text{tx}},\ R_{(n)\text{comp.}}) = \max\left(\left\lceil \left(R^{-1}_{\text{tx}} - 1\right) \cdot R_{(n)\text{SSD}} + R^{-1}_{(n)\text{comp.}} \right\rceil, 1\right) \tag{12}$$

By adjusting the two variables in the above function, we can estimate the change in BEP when building a CSD-array-based compute node.

Impact of Ratio Parameter: We have found the BEP as a function of the computational power and I/O bandwidth of the CSD system. Meanwhile, since R_{tx} and $R_{(n)\text{comp.}}$ are the performance ratio of the SSD system and the CSD system, we can also find the BEP as the performance of the SSD system changes. The computing power and I/O bandwidth of an SSD system depend on the host CPU and mounted SSD, respectively. For example, if $R_{(n)\text{comp.}}$ and R_{tx} become smaller in an SSD system by using a more powerful CPU or an SSD with higher I/O bandwidth, the BEP will increase.

4 EVALUATING *CSDPLAN*

This section presents an evaluation of CSD and *CsdPlan*. To this end, in Section 4.2, CSD performance characteristics are first evaluated to obtain the parameters required for *CsdPlan* use, and then *CsdPlan* is evaluated from Section 4.3.

Table 2. Host Server Specifications

CPU	AMD EPYC™ 7352, 24 Cores (48 Threads), 2.3 GHz (Up to 3.2 GHz)
Socket	2 NUMA Node
Memory	256 GB (64 GB × 4) DRAM DDR4 3200 MHz
OS	Centos 7.92.2009 (Core) / Linux Kernel 4.14

Table 3. CSD Specifications

	SmartSSD [36]	**Newport CSD** [32]
Storage Capacity	3.84 TB	8 TB
Host Interface	PCIe Gen3×4 (U.2)	PCIe Gen3×4 (U.2)
In-Storage Processing Engine	Xilinx Kintex Ultrascale+ KU15P	ARM Cortex-A53 1.0 GHz, 4 Cores
	4 GB DDR4 DRAM, 4.325 MB BRAM	8 GB DDR4 DRAM
	Clock : 300 MHz	OS : Linux Kernel 4.14

4.1 Experiment Setup

We implemented our proposed BEP solver, *CsdPlan*, using Python, and the source code is less than 100 lines. *CsdPlan* takes an extremely short time (in terms of seconds) to find an optimal BEP for CSD-array. However, the initial evaluation of CSD for the workload may require some effort, depending on the accuracy of the CSD's performance characterization.

For evaluation, we build two systems with two AMD EPYC™ 7352 CPUs with 24 cores and 256 GB DRAM, running CentOS 7 where each of the systems has SmartSSD and Newport CSD, respectively. SmartSSD does not have an OS installed and runs the kernel using an FPGA accelerator. On the other hand, Newport CSD runs a Linux-based embedded OS using an ASIC-based 64-bit general-purpose CPU and runs the kernel on it. Detailed specifications of the host server and each CSD are shown in Table 2 and 3[5].

To evaluate the efficiency of our proposed *CsdPlan* for *CsdStore*, we selected four widely adopted analysis kernels from big data applications. The analysis kernel and their corresponding workload working set size (in parenthesis) are listed below:

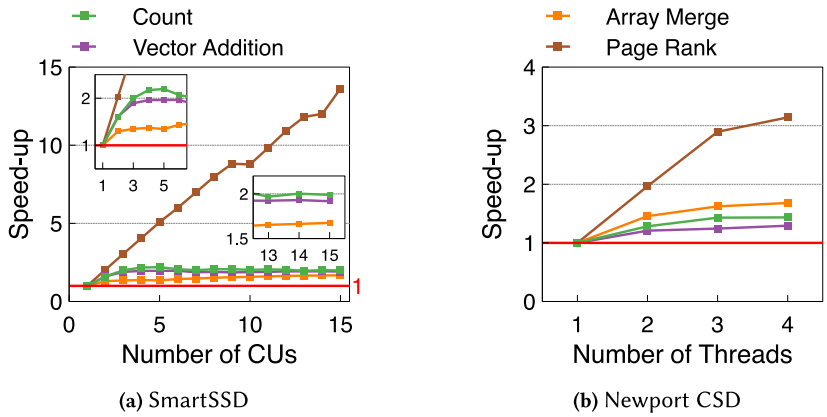
- Count (4.8 GB): Counts specific values in one integer array
- Vector Addition (4.8 GB): Calculates the sum of each element of two integer arrays and creates one large integer array
- Array Merge (4.8 GB): Takes two sorted integer arrays as input, removes duplicate elements, merges them, and creates a new merged array
- Page Rank (0.2 GB[6]): Performs Page Rank algorithm [33] for graph data processing using the rank values of pages stored in one float-type two-dimensional array and one float-type one-dimensional vector

4.2 Performance Characterization of CSDs

Our proposed *CsdPlan* relies on system architects to evaluate the performance characterization of the CSDs. Thus, in this subsection, we present the evaluation steps for the performance

[5]The memory used for In-Storage Processing is separate from the memory required by the controller for the NAND flash operation of SSD.

[6]The execution time of Page Rank is extremely long and increases exponentially as the working set increases, so the working set is smaller than other workloads.



(a) SmartSSD **(b)** Newport CSD

Fig. 4. Impact of parallel computation by either multiple CUs used or multi-threading. All experiments were normalized to the throughput when using one CU or thread.

characterization of CSD and the process of analyzing the results for selected big data workloads. To this end, the throughput was shown by measuring the data transfer time and computation time when offloading the analysis kernel in CSD. The kernel offloading overhead from the host side to CSD is considered to be not that significant in our experiment setup, thus, we did not take into account that overhead. Data transfer time is the time for loading the working set from NAND in the CSD to memory, and computation time is the time for computing the working set loaded in the memory of the CSD. When processing a working set of 4.8 GB, Newport CSD has 8 GB of memory, so it processes the workload with 1 I/O, and SmartSSD has 4 GB of memory, so it processes with 2 I/O.

4.2.1 Impact of Parallel Computation. The CSDs in our system, SmartSSD, and Newport CSD, have the capability to perform parallel computation. SmartSSD is an FPGA-based CSD and employs multiple **computation units (CUs)**, up to 15 CUs. A CU is a computing instance created within the FGPA to execute the kernel. Meanwhile, the Newport CSD is equipped with a quad-core ARM processor, enabling multi-kernel execution by multi-threading on Linux OS. We first present the evaluation results of SmartSSD with an increasing number of CUs with selected big data workloads, and then we discuss the performance of Newport CSD in detail.

SmartSSD's Results: Figure 4(a) shows the throughput improvement (Speed-up) of SmartSSD with an increasing number of CUs (up to 15 CUs in our case). To fully exploit the performance characteristics of SmartSSD's FPGAs, we applied several performance optimization techniques, such as local memory buffers, loop pipelining, and array partitioning, to evaluate big analysis kernels [7–11]. In Figure 4(a), the throughput of the Count and Vector Addition kernels improve up to 5 CUs by 2.2× and 2.0× respectively, after which the throughput is saturated. The main reason for saturating throughput is that the CU executes the kernel after the data loaded in DRAM is copied to **BRAM (Block RAM)**, and the memory copy from DRAM to BRAM becomes the bottleneck. BRAM [5] is an FPGA's on-chip RAM, which can process data loaded in local memory faster than DRAM. Therefore, using BRAM is more effective than using DRAM at the expense of memory copy overhead. However, when many CUs access BRAM at the same time, BRAM access becomes a bottleneck. On the other hand, Page Rank is a CPU-intensive workload, and there is almost no bottleneck caused by the memory copy mentioned above. Thus, the Page Rank shows perfect scaling as the number of CUs increases, improving up to 13.6×. Array Merge shows a gradual increase,

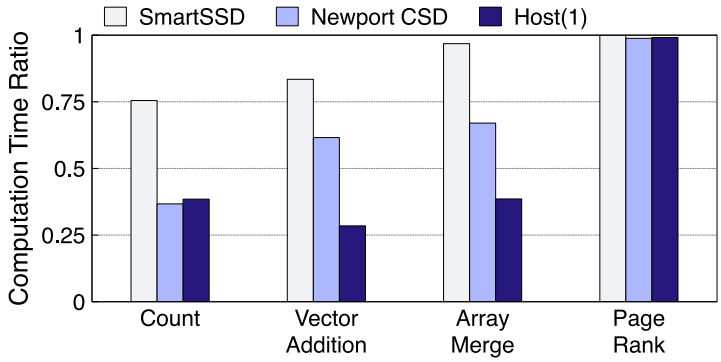


Fig. 5. Computation time ratio of execution time for each workload of SmartSSD, Newport CSD and host CPU(1).

but only up to 1.6×. Like Page Rank, Array Merge is a CPU-intensive workload, but the algorithm has a lot of if-else statements for merge operations, which is the main impediment to performance gains.

Newport CSD's Results: Figure 4(b) shows the evaluation results of the Newport CSD with multithreading enabled within analysis kernels. Since Newport CSD is equipped with 4 CPU cores, we conducted experiments with up to 4 threads by mapping each execution thread to each core (one-to-one mapping). All workloads used in each evaluation were written in parallel programs to enable multithreading. The results in Figure 4(b) are similar to those in Figure 4(a). As shown in Figure 4(b), throughput scales up to 4 threads for all workloads. The throughput of Count, Vector Addition, Array Merge, and Page Rank has been improved up to 1.4×, 1.3×, 1.7×, and 3.1×, respectively. However, a notable observation is that Page Rank linearly scaled in SmartSSD (as shown in Figure 4(a)), but it only linearly scaled up to 3 threads in Newport CSD (as shown in Figure 4(b)), and thereafter, throughput improvement is slightly reduced (Figure 4(b)).

Furthermore, in SmartSSD, throughput is improved in the order of Page Rank, Count, Vector Addition, and Array Merge. In contrast, in Newport CSD, throughput is improved by Page Rank, Array Merge, Count, and Vector Addition, showing different results. The reason is that SmartSSD's FPGA needs to apply various optimization techniques to achieve optimal throughput. The kernel code of an FPGA is more complex that is different from the code that runs on a typical ASIC-based processor. Therefore, the trend of throughput improvement may vary between kernels/workloads.

4.2.2 Workload Classification. As observed in Figure 4, the more parallel processing of the workload computation, the higher the throughput. However, as explained in Amdahl's law [19], the throughput improvement has a limit bound to the data transfer time that cannot be further reduced. Therefore, we analyze computation and I/O ratios for each workload. Figure 5 shows the **computation time ratio to total execution time (CTR)** for each workload. CTR values vary depending on the system. For the convenience of explanation, based on the CTR value of the host system, we classify workloads with a CTR of less than 0.5 as I/O-intensive workloads and otherwise as compute-intensive workloads. For example, Page Rank is a completely compute-intensive workload.

4.2.3 CSD vs. Host System. Now we compare the throughput of a single CSD with the host system[7]. Figure 6 shows the evaluation results for all workloads. Throughput was normalized to

[7]Hereafter, we use the host system and SSD system interchangeably.

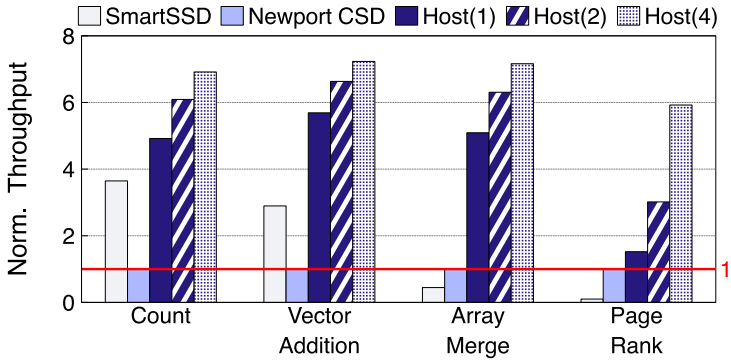


Fig. 6. Throughput comparison for each workload on SmartSSD, Newport CSD, and host systems. Throughput is normalized to the throughput of Newport CSD. In Host(n), n represents the number of active cores on the host system.

Newport CSD. Each CSD is configured to achieve maximum throughput using multiple computational units (refer to Figure 4). The host system used a SmartSSD as a block device. We limit the number of cores for the host system to 4, and each configuration is represented as Host(1), Host(2), and Host(4) in Figure 6.

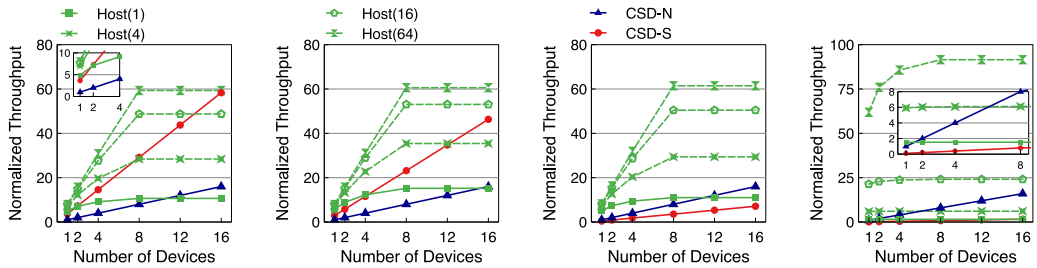
In Count and Vector Addition, SmartSSD has about 3× higher throughput than Newport CSD. This is because both Count and Vector Addition are I/O intensive workloads, and although Newport CSD's computation latency is lower than SmartSSD's, throughput is more affected by internal I/O bandwidth (SmartSSD's internal I/O BW is higher, refer to Figure 2). In addition, Host(1) has a higher throughput of about 5× than Newport CSD. When compared to SmartSSD and Host(1), Host(1) has up to 1.3× and 2× higher throughput than SmartSSD. On the other hand, as the number of active cores increases in the host system, the workload throughput improves by about 20% per core. This is because it is bound to the data movement time between the host and the SSD.

In Array Merge and Page Rank, unlike observed in Count and Vector Addition, Newport CSD shows 2.2× and 10× higher throughput than SmartSSD because Newport CSD has higher computational power than SmartSSD. Although Array Merge is an I/O-intensive workload, it requires sufficient computational power as well. Thus, we can categorize the Array Merge kernel as compute- and I/O-intensive workload. Also, as expected, the workload throughput of the host system is higher than that of CSD. However, it is noteworthy that the throughput of the host system is observed to be significantly higher in Page Rank. This is because Page Rank is a completely compute-intensive workload, and computational performance has the greatest impact on throughput. Therefore, the host system, which has the highest computational power, exhibits a significantly higher workload throughput than any CSD.

4.3 Evaluating *CsdPlan* Solver

Finding the Break-Even Point (BEP): *CsdPlan* finds different BEPs depending on the workload, CSD-array, and host configurations. To evaluate *CsdPlan*, we assumed that the host's system supports PCIe 3.0 up to 32 lanes (1.0 GB/s × 32 = 32 GB/s), and each SSD uses 4 GB/s of bandwidth [29]. So, k_{limit} of Equation (6) is set to 8.

Figure 7 shows each workload's throughput for various host configurations (1, 4, 16, or 64 active cores) and CSD-array systems with SmartSSD or Newport CSD. Throughput was normalized to a single Newport CSD. For all workloads, CSD-array systems increase throughput linearly as the number of devices increases. The host, on the other hand, has higher throughput with an increasing



(a) Count (b) Vector Addition (c) Array Merge (d) Page Rank

Fig. 7. Evaluation to find the break-even point for normal conditions. In all results, throughput is normalized to the throughput of a single Newport CSD system for each workload. In Host(n), n represents the number of active cores on the Host. CSD-N and CSD-S denote CSD-array system (Newport CSD) and CSD-array system (SmartSSD), respectively.

number of cores and storage devices (shown in x-axis of Figure 7) except Page Rank, but the host's throughput improvement was limited to 8 SSDs due to k_{limit} being 8. Since Page Rank is a compute-intensive workload, an increase in I/O throughput with an increasing number of storage devices does not lead to an improvement in throughput (more details on Page Rank are provided below).

Before analyzing the results, for convenience of explanation, we assume that Host('a', 'b') is a host system with 'a' cores and 'b' devices, and CSD-N('c') is a CSD-array system composed of 'c' Newport CSDs, and CSD-S('c') is a CSD-array system composed of 'c' SmartSSDs. In Count, Host(1, 1) has higher throughput than CSD-S(1). However, when the number of storage devices is 2, Host(1, 2) and CSD-S(2), the throughput meets for both configurations. This is because CSD-S has higher internal bandwidth, which leads to higher throughput with an increasing number of devices. On the other hand, in CSD-N, the increase in throughput is lower than CSD-S as the number of devices increases, but the throughput increases gradually. Thus, CSD-N meets Host(1) when the number of devices reaches 12. As expected, Host(1) faces a limit in throughput scalability due to the PCIe bandwidth bottleneck, and eventually, CSD-N becomes higher than Host(1). In addition, the host is equipped with a high-performance CPU, as shown in Table 2, and when the workload is running a single thread, the CPU is very under-utilized. Therefore, as the number of cores increases at the host side, the throughput increases and the BEP value also increases. Host(4) meets CSD-S(8), and Host(16) meets CSD-S(13). However, the degree of improvement in throughput is reduced. This is because the workload execution time is bound to the data transfer time.

Array Merge is a mix of compute and I/O intensive workloads. That is, Array Merge requires a system with high computational power as well as high I/O throughput. Therefore, as shown in Figure 7(c), in both CSD-N and CSD-S, the increase in throughput is not higher than that of the host system as the number of devices increases. In Figure 7(c), Host(1) and CSD-N meet when the number of devices is 12. CSD-S shows lower throughput than CSD-N. This is because the computational power of SmartSSD is lower than that of Newport CSD. If the number of cores in the host increases, the host and CSD-array do not meet in 16 devices (refer to Host(4), Host(16), and Host (64) in Figure 7(c)). In other words, for Array Merge, 16 or more CSDs are required for the CSD-array system to achieve higher throughput than the host system. In summary, for I/O-intensive workload (Count, Vector Addition and Array Merge), the host system's performance is limited by the PCIe bandwidth, where CSD-array catches up with host.

Finally, Page Rank is a completely compute-intensive workload. As shown in Figure 7(d), in the host system, the throughput increases with the number of cores, whereas the number of devices has little effect. However, in 64 cores, the throughput increases with the number of devices. This

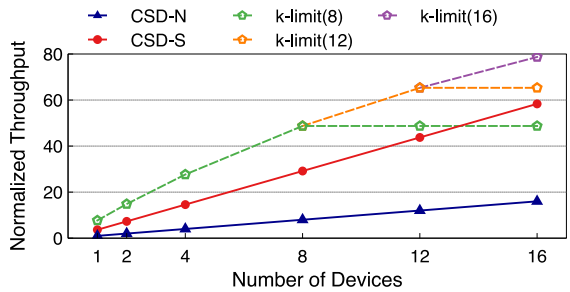


Fig. 8. Example of change in maximum throughput of Host(16) according to k_{limit} in Count.

is because the computation time is so low that the data transfer time affects the throughput. On the other hand, CSD-N and CSD-S increase the workload throughput as the number of devices increases. As mentioned earlier, BEP increases as the number of cores in the host increases.

Figure 8 shows an example of the maximum throughput of Host(16) according to k_{limit} of Equation (6) in Count. In the legend, 'n' in k-limit(n) means the k_{limit} value. The 'n' in k-limit(n) represents the value of k_{limit}. As the number of devices increases, the throughput of $k_{limit}(8)$ improves to 8 devices, while the throughput of $k_{limit}(12)$ and $k_{limit}(16)$ improves to 12 and 16 devices, respectively. Therefore, the BEP value also increases. The $k_{limit}(8)$ meets CSD-S(13), while $k_{limit}(12)$ and $k_{limit}(16)$ meet after CSD-S(16). As such, *CsdPlan* can find BEP changes according to the number of devices that cause PCIe bottlenecks in various host systems.

Impact of CSD Parameters: The factors that determine the workload throughput in CSD are computational power and internal I/O bandwidth. *CsdPlan* can estimate the change in the BEP for Host(n) according to the change in the values of these two factors. In Equation (12), an increase in R_{tx} or $R_{(n)\text{comp.}}$ means an increase in internal I/O bandwidth or computational power of CSD, respectively. In this experiment, we assumed a Host(n) system where n is 1 (one active core on the server) and conducted experiments and analysis.

Figure 9(a)-(d) shows the results for SmartSSD. SmartSSD has internal I/O bandwidth that meets the needs of each workload to some extent but has lower computational power. Therefore, increasing I/O bandwidth does not impact the BEP for compute-intensive workloads, such as Page Rank, while increasing the computational power does. On the other hand, for I/O intensive workloads, such as Count, SmartSSD shows sufficient internal I/O bandwidth and computational power. Thus, changing any factor does not impact much in throughput. Vector Addition is also an I/O intensive workload, but increasing computational power does impact the BEP because SmartSSD has low computational power. Furthermore, Array Merge is a combination of compute and I/O intensive workloads. Thus, increasing both computational and I/O bandwidth will have a great effect on lowering the BEP.

Figure 9(e)-(h) shows the result for Newport CSD. Newport CSD has relatively higher computational power than SmartSSD but has lower internal I/O bandwidth. As shown in Figure 9, in I/O-intensive workloads such as Count, Vector Addition, and Array Merge, an increase in the internal I/O bandwidth has a great effect on lowering the BEP. However, in Vector Addition and Array Merge, computational power also tends to be affected. That is, Newport CSD has higher computational power than SmartSSD, but it is still lower than the host. Finally, Page Rank, again, is a completely compute-intensive workload, so increasing I/O bandwidth has little effect on lowering BEP.

CsdPlan can find the BEP values that change according to the hardware parameters (computational power and internal I/O bandwidth) of the CSD. Figure 10 visually shows the BEP values that

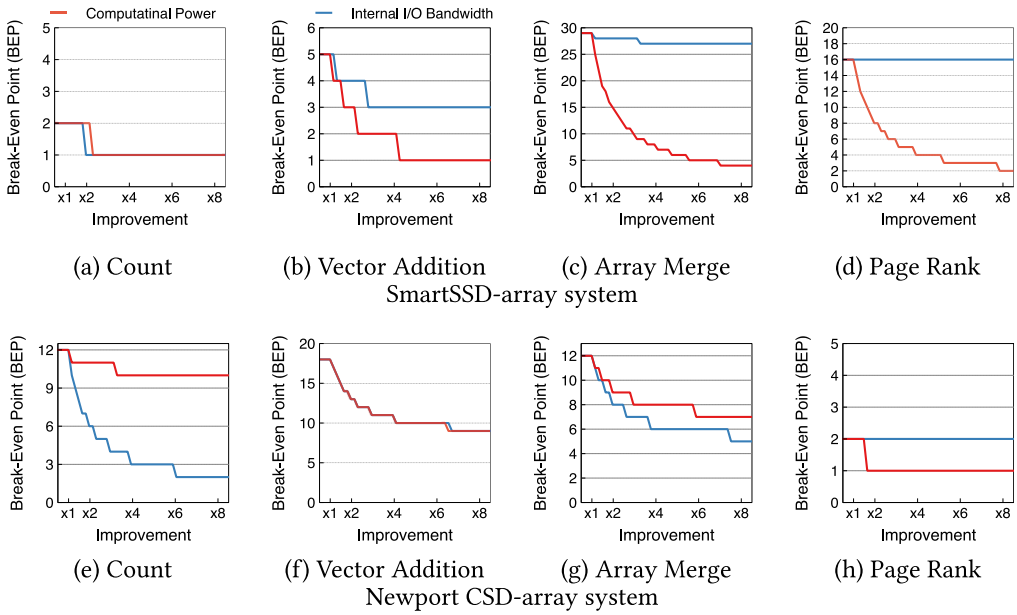


Fig. 9. Analysis of reduction in the number of devices (BEP) by varying CSD's computational power or internal I/O bandwidth.

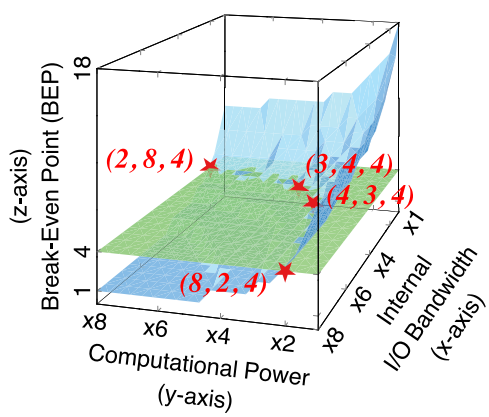
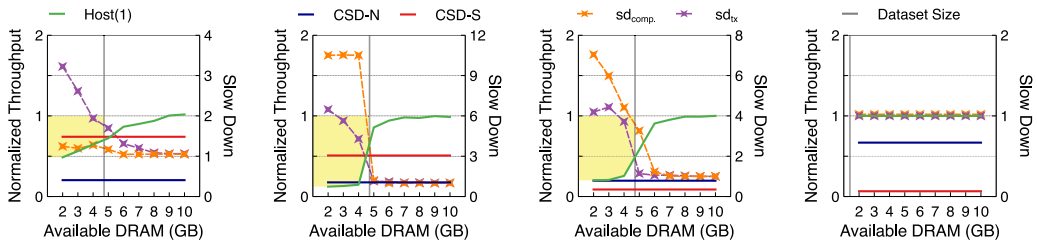


Fig. 10. Finding the values of BEP according to the change of internal I/O bandwidth and computation power of CSD for Vector Addition with the Newport CSD-array.

CsdPlan finds. The x-axis means internal I/O bandwidth, the y-axis means computational power, and the z-axis means BEP (the number of devices) that *CsdPlan* finds. As shown in Figure 10, a 3D plane (drawn by a 3D function of $z = f(x, y)$) shows the changes in BEP values for x and y (blue plane). Also, the points where the green plane and the blue plane intersect in the figure are all combinations of hardware parameters corresponding to BEP $= C$ where C is a constant. For example, in the Figure 10, C is 4. In the figure, four combinations that satisfy BEP = 4{(2, 7, 4), (3, 4, 4), (4, 3, 4), (8, 2, 4)} are marked with red stars.

Moreover, the results presented in Figure 10 can be considered as the design guideline for CSD manufacturers. The two factors impacting the performance of CSD are: internal I/O bandwidth



(a) Count (b) Vector Addition (c) Array Merge (d) Page Rank

Fig. 11. Comparison of the performance of analysis kernels for SSD system and CSD system under overload condition. In order to simulate the overload condition of the host machine, the amount of physical memory available to the analysis kernel is limited.

and computational power, and both are required to be improved. CSD has very low computational power compared to the host CPU. In order to improve the performance of CSD, it is necessary to install a processor with higher computational power.

4.4 *CsdPlan* Solver in Overload Situations

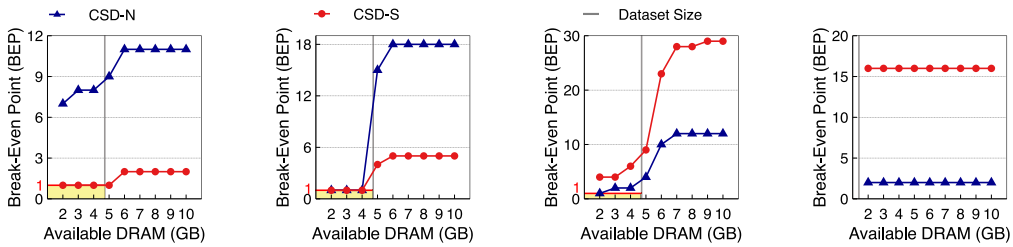
As mentioned in Section 3.2, the host system can be overloaded due to excessive resource usage by applications co-located on the host. In this section, we show how *CsdPlan* finds the BEP of a CSD-array-based system under such an overloaded host system.

First, as described in Section 4.2, the system architect should perform a performance characterization of the host system under overloaded conditions. An overload situation can occur for some reasons; lack of CPU cycles, insufficient memory, I/O bandwidth, or a mixture of these. In big data applications, overloaded situations often occur due to insufficient memory, thus, we consider it as the main cause here as well. In this subsection, we provide guidelines to system architects for the experiments to be performed in overloaded conditions.

The goal of the experiment is to find the optimal values of slow-down factors (mentioned in Equation (8)) and find BEP in various overloaded conditions. For this experiment, we simulate the overloaded condition on the host system by controlling the amount of physical memory by adopting *mlock*()[8] to limit the available memory to the analysis kernel. We simulated with 2 GB to 10 GB of available memory for analysis kernels.

Results: Figure 11 shows the comparison results of throughput according to the size of the host's available memory for the three systems and shows the host's two slow-down factors ($sd_{\text{comp.}}$, sd_{tx}) from Equation (8). An increase in $sd_{\text{comp.}}$ and sd_{tx} means the degradation of computational power and data transfer time of the host, respectively. Here, we use the Host(1), CSD-N, and CSD-S notations used in Section 4.3. All system's throughputs were normalized to the Host(1)'s throughput under normal conditions. In Figure 11(a)-(c), the Host(1) throughput is significantly reduced when the host's available memory is less than the dataset size, and host's slow-down factors are significantly increased. In Count, only sd_{tx} increases significantly because it is an I/O-intensive workload. Vector Addition and Array Merge are also I/O-intensive workloads, but $sd_{\text{comp.}}$ and sd_{tx} grow together. This is because both have much computation compared to Count and is fatally affected when the host's memory is insufficient. On the other hand, CSDs throughput are not affected by the availability of resources on the host. In Figure 11(d), the Host(1)'s throughput and

[8]mlock() locks part or all of the calling process's virtual address space into RAM, preventing that memory from being paged to the swap area.



(a) Count (b) Vector Addition (c) Array Merge (d) Page Rank

Fig. 12. Results of finding deceleration factors and break-even points according to available DRAM size.

slow-down factors does not change at all, no matter how much memory is available. This is because the size of the dataset for the Page Rank workload is small. All datasets are loaded into memory, so disk swapping does not occur at all in the virtual memory system.

Figure 12 shows the change of BEP according to the host's available memory for each workload. The result was calculated by substituting the slow-down factor used in Figure 11 into Equation (8).

In the Count, Vector Addition, and Array Merge (except for Newport CSD's Count and SmartSSD's Array Merge), the BEP is 1 when the host's available memory is smaller than the dataset size and increases rapidly when the host's available memory begins to exceed the dataset. This shows that the host's resource (memory) actually has a significant effect on the slow-down factor, and our proposed modeling from Equation (4) is well applied.

The parts marked in yellow in Figure 11 and 12 correspond to the case where the BEP is 1. In all cases, the BEP included in the parts marked in yellow in Figure 11 is equally included in Figure 12. This means that our proposed modeling works well. Through this result, it is possible to analyze the effectiveness of CSD according to the change of host resources using *CsdPlan*.

Impact of Host's Overloading: *CsdPlan* can find the BEP value according to the change in the host's slow-down factors. For this, *CsdPlan* uses the following function:

$$S_{\text{overload}}(sd_{\text{tx}},\ sd_{\text{comp.}}) = \max\left(\left\lceil sd_{\text{comp.}}^{-1} \cdot \left\{\left(R_{\text{tx}}^{-1} - sd_{\text{tx}}\right) \cdot R_{(n)\text{SSD}} + R_{(n)\text{comp.}}^{-1}\right)\right\}\right\rceil, 1\right) \tag{13}$$

The above function extends Equation (8) and has a form similar to Equation (12). Here we show that *CsdPlan* uses the above function to find the value of BEP according to host's two slow-down factors. For the experiment, we assumed a Host(n) system where n is 1.

Figure 13(a)-(d) shows the results for SmartSSD. The result is similar to Figure 9(a)-(d). In this evaluation, the decrease in the computational power of the host almost coincides with the increase in the computational power of the SmartSSD in Figure 9 (red line). On the other hand, for Vector Addition and Array Merge in Figure 9, the increase in SmartSSD's internal I/O bandwidth has a limit to the BEP reduction, but the decrease in the host's I/O bandwidth has an almost linear effect on the BEP. In Figure 9, since BEP reduction is evaluated when the total execution time of the SmartSSD is reduced, the total execution time of the workload is bound to the computation time. However, in this evaluation, since BEP reduction was evaluated when the total execution time of the host's workload increases, the total execution time of the workload continues to increase according to the decrease in the I/O bandwidth of the host. Page Rank is completely compute-intensive, so BEP is not affected by the reduced host's I/O bandwidth.

Figure 13(e)-(h) shows the results for Newport CSD. Unlike SmartSSD, this result is not similar to Figure 9(e)-(h). This is because Newport CSD has lower I/O bandwidth and computing power compared to the host. Note that the host system uses SmartSSD as a block device, so the I/O bandwidth is similar to SmartSSD. Overall, BEP is exponentially affected by the reduction in host

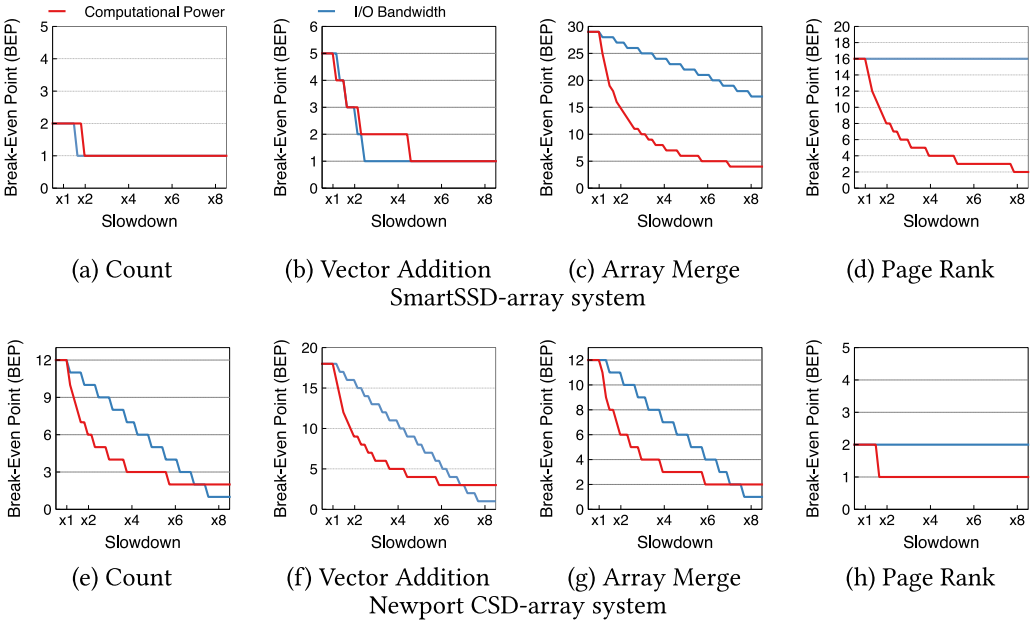


Fig. 13. Analysis of reduction in the number of devices (BEP) by varying host's slow-down factors.

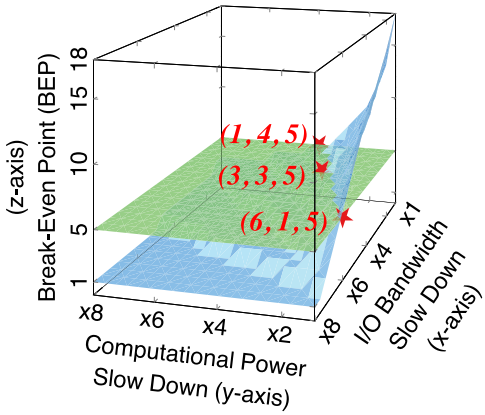


Fig. 14. Finding the values of BEP according to the change of host's slow-down factors for Vector Addition with the Newport CSD-array.

computational power, while it is linearly affected by the reduction in I/O bandwidth. The reason is that, in Equation (13), the value of the S_{overload} function is inversely proportional to $sd_{\text{comp.}}$, whereas it is in direct proportional relationship to sd_{tx}. In Count, Vector Addition, and Array Merge, BEP is more affected by the reduction in computational power than the reduction in I/O bandwidth of the host and reverses after ×7 in Figure 13 The inversion value is dependent on the performance difference between CSD, host, and workload characteristics. Page Rank is completely compute-intensive, so BEP is not affected by the reduced host's I/O bandwidth.

In addition, *CsdPlan* finds BEP for the combination of two factors (sd_{tx}, $sd_{\text{comp.}}$), as shown in Figure 10. Figure 14 visually shows the BEP values that *CsdPlan* finds. The x-axis means I/O

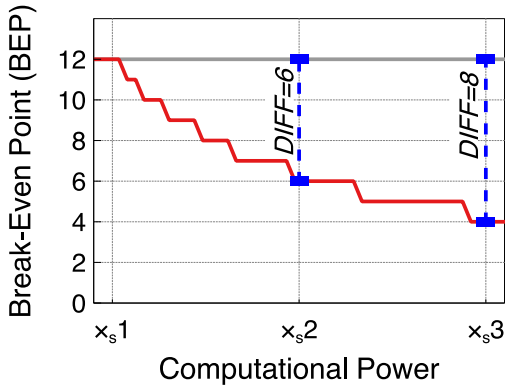


Fig. 15. In Array Merge of Newport CSD, BEP reduction is according to the computational power of the host CPU. On the x-axis, 'n' in $\times_s n$ indicates n times slower.

Table 4. The Total Cost of Ownership when Building a Compute Node with Baseline SSDs and CSD

System	CPU-Part				Storage-Part		Total	Perf. /	Capacity /
	Name	Perf. (Mark)	Slow Down	Cost	SSD/CSD	Cost	Cost	$	$
SSD-system	AMD EPYS 7351	39999	-	$1828	12×SSD	$1200	$3028	13.2/$	8.10 GB/$
CSD-system (A)	AMD EPYS 7452	20471	Appx. ×2	$1399	6×CSD	$660	$2059	19.4/$	5.96 GB/$
CSD-system (B)	AMD EPYS 7251	14935	Appx. ×3	$485	8×CSD	$880	$1365	29.3/$	12.00 GB/$

bandwidth (sd_{tx}), and the y-axis means computational power ($sd_{\text{comp.}}$), and the z-axis means BEP (the number of devices) that *CsdPlan* finds. In the figure, the explanation of BEP is the same as that of Figure 10. For example, sketch red stars.

4.5 Analysis of Total Cost of Ownership

In this subsection, we will discuss the total cost of ownership (TCO) for building a compute node using *CsdPlan*. The cost of building a server consists of installation costs and recurring costs. We compare the cost of building a compute node using SSDs and CSDs where we only consider the cost of host CPU and storage among the installation costs and ignore the recurring cost. Also, we consider a system with the same throughput for processing workloads for both systems. CSD-system can improve throughput through internal resources, so using a low computational power CPU for the host would reduce the cost.

Figure 15, in Array Merge of Newport CSD, shows the difference (*DIFF*) between the BEP and the initial BEP value (12) according to the decrease in computational power of the host CPU. The x-axis represents the slowdown factor of computational power, and the *DIFF* represents the difference in the number of CSDs required to meet the performance requirement of the SSD system for the CSD system. For instance, the CSD-system with 2× slower host CPU only requires 6 Newport CSDs to meet the performance requirements, while the CSD-system with 3× slower host CPU only requires 8 Newport CSDs.

Table 4 shows an example of a cost comparison between the SSD-system and CSD-system (A and B) related to Figure 15. The first part of Table 4 represents the adopted CPUs in all three systems. "Mark" in the table represents the score evaluated using Passmark, a widely used software tool for CPU benchmarking [3]. We considered a system with AMD EPYC 7351 and 12 SSDs as the baseline compute node. For CSD-systems, we considered the low-power CPUs from the same lineup, AMD

EPYC 7452 (approximately 2× slower) and AMD EPYC 7251 (approximately 3× slower), as CSDs have considerable computation resources.

The second part of Table 4 shows the storage devices employed in all three systems. The SSD-system is equipped with 12 SSDs while CSD-system (A) and CSD-system (B) have a varying number of CSDs based on the BEP values from *CsdPlan*. The commercially available CSDs are relatively expensive compared to SSDs. Thus, we assume that the prices of CSDs would become affordable once adopted by the system architect actively. For this comparison, we consider the following cost model for SSD and CSD: (i) The price of the 2 TB SSD is $100 [20], and (ii) The cost of processor-equipped SoC for CSD is 10% higher, so the price of CSD is $110. Comparing the total cost according to our model, the performance/$ for CSD-system (A) and CSD-system (B) are 19.4/$ and 29.3/$, respectively, which is about 1.47× and 2.21× higher than the SSD-system. On the other hand, in terms of Capacity/$, CSD-system (A) shows a decrease to 5.96 GB/$, which is 0.27× less than the SSD-system, while CSD-system (B) exhibits an increase to 12 GB/$, which is 1.48× higher. CSD-system (A) experiences a decrease in storage capacity's efficiency compared to the total cost, despite reduced CPU cost. Therefore, storage architects should comprehensively consider both performance/$ and capacity/$ when configuring compute nodes, enabling them to evaluate the effectiveness of CSD and reduce the Total Cost of Ownership (TCO).

5 CONCLUSION AND FUTURE WORK

HPC facilities have started looking at the potential of adopting storage devices within the simulation nodes which provides an opportunity for adopting in-storage processing solutions within simulation nodes to perform data analysis tasks. With the advent of CSDs, there are opportunities for building CSD-array-based computing nodes called *CsdStore*. With CSDs, data analytic tasks are offloaded to the device where data resides and, reducing the cost of data movement optimizing the performance, energy utilization, and total cost of ownership. However, adopting CSDs naively does not benefit due to the distinct hardware and performance characteristics of commercially available CSDs. Therefore, in this work, we formulated and implemented a storage capacity planner, called *CsdPlan*, that takes into account the hardware and performance characteristics of CSDs, host systems, and workloads to provision *CsdStore* in a cost-effective manner. *CsdPlan* finds the optimal number of CSDs (BEP) to outperform a traditional compute node with block-based SSDs. We demonstrated the efficacy of our proposed *CsdPlan* through two commercial CSDs – SmartSSD and Newport CSD – and showed how *CsdPlan* effectively finds optimal BEP. Our proposed solution also tracks changes in BEP according to the change in hardware parameters of host and CSD systems (i.e., computational power and I/O bandwidth).

The simulation node can adopt a CSD and SSD combination system (CSD-SSD system). In this case, *CsdPlan*'s capacity planner alone is not sufficient to find the optimal BEP according to the workload. In the CSD-SSD system, the degree of performance improvement due to parallel processing varies according to the number of SSDs and CSDs. In addition, the size of the workload executed by the CSD-SSD system can be dynamically changed depending on the situation, and several different workloads can be executed simultaneously. Therefore, in this case, sophisticated workload analysis is required considering the number of SSDs as well as the performance characteristics of CSDs. We will expand *CsdPlan* as future work to explore technologies that allow CSD-SSD systems to have optimal performance in dynamic workloads.

REFERENCES

[1] 2022. Frontier - Exascale Supercomputer. (2022). https://www.olcf.ornl.gov/frontier Last Accessed: December 1, 2022.
[2] 2022. Los Alamos National Laboratory and SK hynix to demonstrate first-of-a-kind ordered Key-value Store Computational Storage Device. (2022). https://discover.lanl.gov/news/0728-storage-devic Last Accessed: November 28, 2022.

[3] 2022. PassMark - CPU Mark. (2022). Retrieved Nov. 10, 2022 from https://web.archive.org/web/20221024093010/https://www.cpubenchmark.net/high_end_cpus.html
[4] 2022. Top500 Supercomputer site. https://www.top500.org/lists/top500/list/2022/11/. (2022). Last Accessed: November 28, 2022.
[5] ARM Xilinx. 2018. BRAM and Other Memories. (2018). Retrieved Nov. 10, 2022 from https://www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/jbt1504034294480.html
[6] ARM Xilinx. 2021. P2P bandwidth Example. (2021). Retrieved Nov. 10, 2022 from https://github.com/Xilinx/Vitis_Accel_Examples/tree/master/host/p2p_bandwidth
[7] ARM Xilinx. 2021. Vitis Accel Examples. (2021). Retrieved Nov. 10, 2022 from https://github.com/Xilinx/Vitis_Accel_Examples
[8] ARM Xilinx. 2021. Vitis Accel Examples Documentation. (2021). Retrieved Nov. 10, 2022 from https://xilinx.github.io/Vitis_Accel_Examples/2021.2/html/index.html
[9] ARM Xilinx. 2022. UG1416-Vitis-Documentation. (2022). Retrieved Nov. 10, 2022 from https://docs.xilinx.com/v/u/en-US/ug1416-vitis-documentation
[10] ARM Xilinx. 2022. Vitis High-Level Synthesis User Guide (UG1399). (2022). Retrieved Nov. 10, 2022 from https://docs.xilinx.com/r/en-US/ug1399-vitis-hls
[11] ARM Xilinx. 2022. Vitis Unified Software Platform Documentation: Application Acceleration Development (UG1393). (2022). Retrieved Nov. 10, 2022 from https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration
[12] Axboe, J. 2021. GitHub—axboe/fio: Flexible I/O Tester. (2021). Retrieved Nov. 10, 2022 from https://github.com/axboe/fio
[13] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H. Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-data processing: Insights from a MICRO-46 workshop. *IEEE Micro* 34, 4 (2014), 36–42. https://doi.org/10.1109/MM.2014.55
[14] Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, and Laurent Lemarchand. 2019. Optimizing the cost of DBaaS object placement in hybrid storage systems. *Future Generation Computer Systems* 93 (2019), 176–187.
[15] Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, Zhenjun Liu, Feng Zhu, and Tong Zhang. 2020. POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*. USENIX Association, USA, 29–42.
[16] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492
[17] Jaeyoung Do, Victor C. Ferreira, Hossein Bobarshad, Mahdi Torabzadehkashi, Siavash Rezaei, Ali Heydarigorji, Diego Souza, Brunno F. Goldstein, Leandro Santiago, Min Soo Kim, Priscila M. V. Lima, Felipe M. G. França, and Vladimir Alves. 2020. Cost-effective, energy-efficient, and scalable storage computing for large-scale AI applications. *ACM Trans. Storage* 16, 4, Article 21 (Oct. 2020), 37 pages. https://doi.org/10.1145/3415580
[18] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: A framework for near-data processing of big data workloads. *SIGARCH Comput. Archit. News* 44, 3 (2016), 153–165. https://doi.org/10.1145/3007787.3001154
[19] John L. Gustafson. 2011. *Amdahl's Law*. Springer US, Boston, MA, 53–60. https://doi.org/10.1007/978-0-387-09766-4_77
[20] John C. McCallum. 2022. Flash Memory and SSD Prices. (Oct. 22, 2022). https://jcmit.net/flashprice.htm
[21] Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2015. BlueDBM: An appliance for big data analytics. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, 1–13.
[22] Yangwook Kang, Yang-suk Kee, Ethan L. Miller, and Chanik Park. 2013. Enabling cost-effective data processing with smart SSD. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–12. https://doi.org/10.1109/MSST.2013.6558444
[23] Awais Khan, Hyogi Sim, Sudharshan S. Vazhkudai, Ali R. Butt, and Youngjae Kim. 2021. An analysis of system balance and architectural trends based on Top500 supercomputers. In *The International Conference on High Performance Computing in Asia-Pacific Region*. 11–22.
[24] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, Sang-Won Lee, and Bongki Moon. 2016. In-storage processing of database scans and joins. *Inf. Sci.* 327, C (Jan. 2016), 183–200. https://doi.org/10.1016/j.ins.2015.07.056
[25] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *Proceedings of the 19th International*

Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 227–236.

[26] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2014. HybridPlan: A capacity planning technique for projecting storage requirements in hybrid storage systems. *The Journal of Supercomputing* 67, 1 (2014), 277–303.

[27] Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50'17)*. 219–231.

[28] Dongup Kwon, Dongryeong Kim, Junehyuk Boo, Wonsik Lee, and Jangwoo Kim. 2021. A fast and flexible hardware-based virtualization mechanism for computational storage devices. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. 729–743.

[29] Karol Latecki and Maciej Wawryk. 2022. SPDK NVMe BDEV Performance Report release 22.01. (February 2022), 11–12. https://ci.spdk.io/download/performance-reports/SPDK_nvme_bdev_perf_report_2201.pdf

[30] Shengwen Liang, Ying Wang, Youyou Lu, Zhe Yang, Huawei Li, and Xiaowei Li. 2019. Cognitive SSD: A deep learning engine for in-storage data retrieval. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. USENIX, 395–410.

[31] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating server storage to SSDs: Analysis of tradeoffs. In *Proceedings of the Fourth European Conference on Computer Systems (EuroSys'20)*.

[32] NGD Systems. 2022. Newport CSD. (2022). Retrieved Nov. 10, 2022 from https://www.ngdsystems.com/solutions#NewportSection

[33] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. Stanford InfoLab.

[34] Philip Schwan. 2003. Lustre: Building a file system for 1,000-node clusters. In *Proceedings of the Linux Symposium*.

[35] Zhenyuan Ruan, Tong He, and Jason Cong. 2019. INSIDER: Designing in-storage computing system for emerging high-performance drive. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 379–394. https://www.usenix.org/conference/atc19/presentation/ruan

[36] Samsung Electronics. 2022. SmartSSD. (2022). Retrieved Nov. 10, 2022 from https://semiconductor.samsung.com/ssd/smart-ssd/

[37] Scaleflux Inc. 2022. Scaleflux. (2022). Retrieved Nov. 10, 2022 from http://www.scaleflux.com/

[38] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. 2014. Willow: A user-programmable SSD. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, Broomfield, CO, 67–80. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/seshadri

[39] Devesh Tiwari, Simona Boboila, Sudharshan Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter Desnoyers, and Yan Solihin. 2013. Active Flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*. USENIX Association, San Jose, CA, 119–132. https://www.usenix.org/conference/fast13/technical-sessions/presentation/tiwari

[40] Mahdi Torabzadehkashi, Ali Heydarigorji, Siavash Rezaei, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. 2019. Accelerating HPC applications using computational storage devices. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 1878–1885.

[41] Mahdi Torabzadehkashi, Siavash Rezaei, Ali Heydarigorji, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. 2019. Catalina: In-storage processing acceleration for scalable big data analytics. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 430–437. https://doi.org/10.1109/EMPDP.2019.8671589

[42] Mahdi Torabzadehkashi, Siavash Rezaei, Ali HeydariGorji, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. 2019. Computational storage: An efficient and scalable platform for big data and HPC applications. *Journal of Big Data* 6, 1 (2019), 1–29.

[43] Jianguo Wang, Dongchul Park, Yang-Suk Kee, Yannis Papakonstantinou, and Steven Swanson. 2016. SSD In-storage computing for list intersection. In *Proceedings of the 12th International Workshop on Data Management on New Hardware (DaMoN'16)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2933349.2933353

[44] Satoru Watanabe, Kazuhisa Fujimoto, Yuji Saeki, Yoshifumi Fujikawa, and Hiroshi Yoshino. 2019. Column-oriented database acceleration using FPGAs. In *Proceedings of 2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 686–697.

[45] Shuotao Xu, Thomas Bourgeat, Tianhao Huang, Hojun Koim, Sungjin Lee, and Arvind. 2020. AQUOMAN: An analytic-query offloading machine. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 386–399.

[46] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. 2010. PreDatA–preparatory data analytics on peta-scale machines. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–12.

Received 9 December 2022; revised 27 July 2023; accepted 2 September 2023