Analyzing the Energy Consumption of Synchronous and Asynchronous Checkpointing Strategies

Grant Wilkins*, Mikaila J. Gossman* Bogdan Nicolae†, Melissa C. Smith*, Jon C. Calhoun*
*Clemson University, Clemson, SC, USA
†Argonne National Laboratory, Lemont, IL, USA
Emails: {gfwilki, mikailg, smithmc, jonccal}@clemson.edu and bnicolae@anl.gov

***Abstract*—At the start of the exascale computing age, the number of components installed in high performance computing (HPC) systems has increased by more than 70 percent, leading to a shorter mean time between failure (MTBF) and large power budgets approaching and exceeding 20 MWs. In this context, resilience strategies such as checkpoint/restart (C/R), which are traditionally adopted by tightly coupled HPC applications, face the need to become energy efficient, in addition to the need to minimize the runtime overheads at scale. With respect to the latter aspect, C/R has evolved to employ asychronous multi-level resilience techniques that leverage heterogeneous storage. Thus, there are many factors that affect the performance and scalability of C/R. The impact of these factors on energy efficiency and the resulting trade-offs have been studied only to a limited degree so far in the literature. To address this gap, we propose to study two state-of-the-art C/R libraries, VELOC (local checkpoints to DRAM or SSD followed by asynchronous flushes to PFS) and GenericIO (optimized synchronous flushes to PFS). We perform weak and strong scalability experiments and show that DRAM provides $\approx 1.5\times$ greater throughput while utilizing $\approx 25\%$ less energy than SSDs. Further, we show that asynchronous C/R provides $\approx 4\times$ greater throughput while using practically a third of the energy than synchronous C/R. Further we show that data size and throughput are directly correlated to energy consumption, and therefore C/R developers should focus on ways to improve/maintain high throughput in order to effectively reduce energy consumption, rather than throttle hardware, to efficiently address exascale needs.**

***Index Terms*—high-performance computing, fault-tolerant computing, checkpoint-restart, energy-aware computing, performance analysis**

I. Introduction

Applications are facing ever-increasing runtimes and produce massive amounts of intermediate data [1] as production-ready HPC systems continue to scale. In June 2022, Frontier became the U.S.'s first Exascale capable machine, with a reported 8,730,112 compute cores, a 72% increase from its predecessor, Summit [2]. Historically, similar increases have led to difficulties in the form of: (1) high energy consumption due to relatively slower advancements in power technology and increasing complexity of software stacks [3]; and (2) lower mean-time-between-failures (MTBF), raising questions about hardware reliability [4].

In 2008, the scientific community set an ambitious goal to power future exascale-capable machines on 20 MW per year [5]. Even if we assume a very low cost of energy in the U.S. (<\$0.05/kWH), 1 MW of power results in almost \$1 million USD per year [6], meaning upwards of \$20 million USDs is required just to power the system. However, power budgets are subject to change based on new chip technology, cooling strategies, and facility sizes [6]. Thus, it is clear that power draw, and therefore energy, is a major expenditure. Such large energy costs raise problems for users as well, as they run the possibility of system usage being capped [7]. With energy being a key variable to control for both users and site administrators, it is pertinent to understand the impact different applications and middle-ware libraries have on energy consumption in the HPC stack.

On the other hand, the dramatic increase in number of components leads to lower mean-time-between-failures (MTBF) [4], which prompts the need for scalable, low-overhead resilience techniques. In the context of HPC applications, checkpoint-restart (C/R) is the most common resilience technique due to the tightly coupled nature of the processes and tasks. In a nutshell, C/R captures the global state of an application in a resilient fashion at a given point in time, and restarts from that state in case of failures, which dramatically reduces the amount of lost runtime compared to re-running the application from the beginning.

C/R approaches have evolved over time to leverage heterogeneous storage stacks (node-local memory hierarchies and SSDs, external repositories such as a parallel file system – PFS–, key-value stores, etc.) as part of multi-level resilience strategies that adapt to various classes of failures. Based on empirical observations, simple failures like application bugs happen more frequently than catastrophic failures like a large number of nodes going offline at once. Thus, multi-level resilience strategies employ lightweight "levels" to protect against simple failures more frequently (e.g. checkpoint to and restart from local storage) and more expensive "levels" to protect against catastrophic failures less frequently (e.g. checkpoint to and restart from a PFS that offers durability at the expense of high I/O overheads).

Even so, shorter mean time between failures (MTBF) means checkpointing needs to be performed more often at all levels of resilience. As a consequence, checkpoints are written to a PFS more frequently, which introduces unacceptable I/O overheads. To alleviate these overhead, asynchronous multi-level checkpointing techniques [1] block the application only until the checkpoints have been written to local storage (fastest level), then proceed with the other resilience strategies in the background (e.g., flush checkpoints to the PFS using

separate threads), while the application keeps running. This overlap hides the high I/O overhead of accessing a PFS, but is more complex to implement and introduces competition for resources with the application, which is non-trivial to predict and mitigate [8]–[10].

The combined effect of more frequent checkpointing using increasingly complex techniques, coupled with the need to increasingly emphasize energy efficiency, has prompted the need to design multi-level checkpointing techniques that are not only low-overhead and scalable, but also energy efficient. In this context, previous studies analyze the effects of the C/R software-stack (e.g. I/O design choices [8] and different storage layers [1] on application time memory, and bandwidth consumption. However, energy efficiency has comparatively received limited attention in the literature. There are recent efforts into exploring the energy *consumption* of checkpoint-restart, but these works only consider synchronous checkpointing libraries [11], and/or model certain aspects (e.g. checkpoint frequency [12], recovery mechanisms [13], or storage tier [14]) of C/R, not the libraries themselves (which has unaccounted for inefficiencies due to software implementation).

Thus, there is a need to analyze the energy efficiency in a *holistic* fashion that takes into account the overall complexity of checkpointing libraries. This is particularly important in the context of asynchronous multi-level checkpointing libraries, which have both complex implementations and runtime behaviors that involves competition for resources with the application. Just as understanding the interplay between applications and checkpointing runtimes is necessary to mitigate contention and reduce performance degradation, understanding the trade-off between performance, scalability, and energy consumption using various combinations of parameters and resilience strategies is a crucial step in the design of next-generation energy-efficient checkpointing libraries. In this paper we focus on the study of the aforementioned trade-off. We summarize our contributions below:

- We perform weak and strong scalability analysis of the trade-off between the performance overheads and energy consumption of VELOC [1], a production-ready asynchronous checkpointing library, as compared with GenericIO [15], an optimized synchronous checkpointing library for writing checkpoints directly to a PFS.
- We compare the impacts of C/R configurations such as storage hardware, I/O methods, and parallelism on the energy consumption and throughput of C/R applications.
- We detail both the CPU and DRAM energy consumption of C/R on different storage tiers at scale, combining these measurements to report aggregate results.
- We find asynchronous C/R writing to DRAM provides nominal improvement over SSDs in regards to throughput and energy consumption. Therefore, efforts should be taken to efficiently use both resources to provide high throughput and reduce contention for fast, local memory.

II. Background and Related Work

A. Checkpoint-Restart

HPC checkpoint-restart (C/R) captures the global state of multiple processes as a set of checkpoints that can be used to restart from in case of failures. Typically, C/R captures only the data structures residing in the memory of the processes, while other additional states are discarded and reconstructed on restart.

System-level vs. Application-level C/R: System-level C/R (such as DMTCP [16]) employs a transparent approach that captures the full set of in-memory data structures, while application-level C/R captures only the critical data structures and relies on the application to explicitly reconstruct the other data structures on restart. In this paper we study the latter approach. However, without loss of generality, our results can be used to reason about system-level checkpointing libraries too, as the same resilience techniques are applicable once checkpoint files have been produced by either approach.

Synchronous C/R: in this case, all multi-level resilience strategies are applied while blocking the application. A representative example is SCR [17]. In practice, it is often employed as a single level that involves blocking flushes directly to a PFS. In this context, several optimization are employed to reduce the I/O pressure on the PFS, such as aggregating checkpoints on a subset of compute nodes that are responsible for concurrent writes, as illustrated by GenericIO [15]. This type of C/R helps ensure a consistent global-state across a distributed application [11]. Further, synchronous C/R eliminates competition between the checkpointing runtime and the application, thus eliminating the need for interference mitigation, which is a non-trivial issue [8]. However, at scale, synchronous checkpointing strategies have high (and often unpredicatable) overheads, especially when concurrent writes to PFS(s) are involved, which is typically subject to I/O bottlenecks.

Asynchronous C/R: Asynchronous multi-level checkpointing techniques block the application only until the checkpoints have been written to local storage (fastest level), then proceed with the other resilience levels in the background, while the application keeps running (e.g. flush checkpoints to the PFS using separate threads). A representative example is VELOC [1], which supports several multi-level resilience strategies: (1) capture the checkpoints in-memory or to node-local storage; (2) partner replication, (3) peer-to-peer erasure coding, (4) flush to PFS(s), (5) flush to burst buffers or (6) flush to key-value stores. In this paper, we focus on a simple two-level scenario that is frequently used in practice: (1), which is blocking, followed by (4), which is performed asynchronously.

B. Energy-Aware Computing

The growing energy consumption of computing systems is immense and requires significantly greater costs and/or power capping users. In recent years, energy-aware computing (EAC) has become a focus of the HPC community at large, with efforts being made to reduce the environmental impact of

supercomputers. The main directions of research for energy reduction are through (1) analytic, (2) hardware, and (3) software optimizations/comparisons. Here we outline the differences of these approaches and discuss where our work falls.

Analytic EAC: In distributed computing, analytic approaches to optimizations are popular, as mathematical models of parallel systems yield interesting insights into potential pitfalls in performance. While not exhaustive, this section focuses on the analytic models of C/R that optimize energy and power models through aspects of C/R such as checkpoint intervals, scalability, and probabilistic execution. Some works [18]–[20] focus on optimizing checkpoint frequency via the Young-Daly equation [21], often modeling runtime speedup with this equation. Others find a more appropriate probabilistic model for C/R which is proven to allow a more fine-tuned model of energy consumption, resulting in better prediction and future optimization [22], [23]. We do not proceed with analytic methods, instead indirectly use the results as they are implemented by C/R application developers [1], [15].

Hardware-Focused EAC: Hardware-based EAC refers to optimizing the energy consumption of a node via controlling hardware variables such as CPU/GPU frequency and voltage or different storage locations/hardware. This level of control is achieved through dynamic voltage and frequency scaling (DVFS). Voltage and frequency are easily tunable variables through OS-specific kernel tools (e.g. `CPUfreq` for Linux) and directly lowers power consumption of a device.

In the context of C/R, previous works [24]–[26] suggest DVFS is useful in targeting different CPU frequencies to find an optimal power consumption without large performance loss. Other works look at how different hardware types (e.g. NVMe v.s. SSD) impacts the energy efficiency of C/R applications [27], [28]. In this paper, we present a greater focus on the different storage hardware types. Since resource contention introduced by asynchronous C/R is not well understood, we choose to leave DVFS work to more in-depth future studies, where it may have adverse effects on application runtime.

Software EAC: Another area of EAC optimizes energy consumption through software. Often times this comes in the form of analyzing similar applications to explore which configuration yields the greatest performance and lowest energy cost. The methods employed by these studies typically consist of power/energy monitoring over a series of different tests for a series of applications [29]. While studies currently exist that have looked at the trade-offs in energy efficiency of synchronous C/R such as [30], [31], none to our knowledge have explored the gap of comparing the energy consumption of synchronous and asynchronous C/R from a software perspective. We fill this gap in our study by analyzing the energy scalability of asynchronous C/R and the varying storage tiers compared to other synchronous, single-leveled C/R solutions.

C. Energy efficiency of C/R

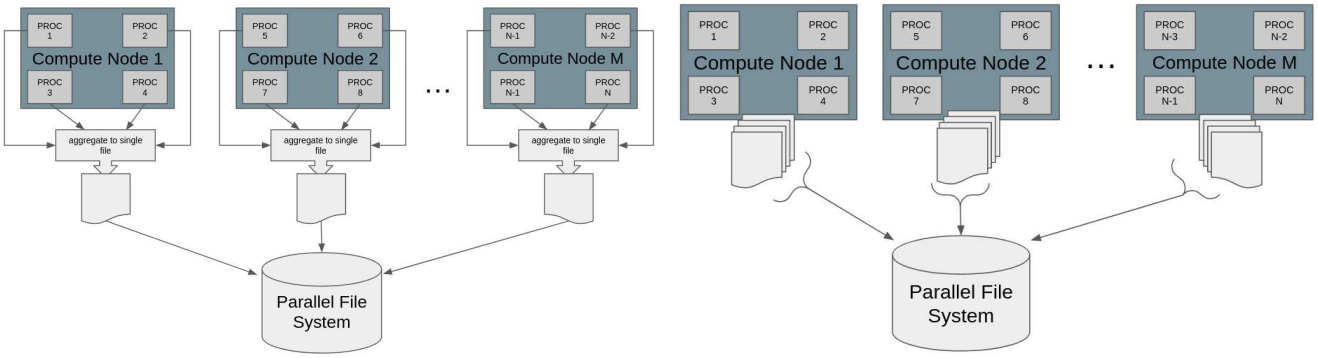
Moran et al. [11] modeled ways to predict the energy consumption of C/R in HPC, however they do not assume heterogeneous storage. Various other studies model energy consumption of C/R at Exascale; one such study by Dauwe et al. [12] in 2017 predicted Exascale hardware configurations and metrics rather well and took into account various storage tiers used by multi-level checkpointing. However, they analyze the energy efficiency of the storage levels independent of C/R. Amrizal et al. [14] assume checkpoint and restart times are constant, which does not apply for PFS(s) that show significant variability in aggregated I/O bandwidth. Miao et al. [13] looks at the interplay between certain scientific workloads and various synchronous resilience strategies, emphasizing various trade-offs. Other studies that seek to improve energy efficiency [13], [32] of C/R suggest throttling CPU power. However, such studies typically assume synchronous C/R, which already has unacceptable overheads at Exascale that would only be amplified by power CPU throttling. In asynchronous C/R, power throttling could also have adverse effects: while the I/O operations are overlapped with the application runtime, they need to finish by the time the next checkpoint request arrives, otherwise resilience is not guaranteed. If this does not happen naturally, the checkpointing library needs to guarantee it by blocking the application, which reduces the effectiveness of asynchronous techniques.

Overall, a comprehensive study of the trade-off between performance overhead, scalability and energy efficiency of C/R is necessary in order enable the design of efficient next-generation asynchronous C/R libraries. To our best knowledge, we are the first to address this gap.

III. Methodology to Study Energy Consumption of C/R

To study the overhead vs. energy efficiency trade-off for C/R, we investigate both synchronous and asynchronous resilience strategies under various write scenarios driven by a benchmark we implemented specifically for this purpose. The benchmark eliminates competition between the application and C/R library, allowing us to accurately identify C/R settings (i.e., I/O strategy, synchronicity, storage hardware) that inherently contribute to high energy overheads. If some configurations generate high energy overheads in isolated testing, it is reasonable to conclude that such overheads would be exacerbated in the presence of competitive applications.

We focus on both weak and strong scalability. Weak scalability illustrates how energy is impacted as the problem size and participating processes grow and provides valuable insight into how we predict the C/R libraries to perform on Exascale workloads and systems. Strong scalability is a corollary to Amdahl's law, allowing us to estimate how higher degrees of parallelism affects the upper limit of energy consumption. For application developers, interpreting our results provides insight for choosing C/R strategies and their optimal settings to reduce energy consumption without sacrificing performance; for C/R developers, this work helps identify areas of C/R that contribute to high energy overheads. We detail our methodology below.



(a) GenericIO: MPI collective I/O using $N \rightarrow M$ aggregation

(b) VELOC: POSIX I/O using one file-per-process flush strategy

Fig. 1: Checkpointing libraries: GenericIO vs. VELOC

A. Compared Approaches

We have chosen to focus on two representative checkpointing approaches: (1) GenericIO (GIO) [15], which is representative of an application-level, optimized synchronous checkpointing approach that directly writes the checkpoints to a PFS; (2) VELOC [1], which is representative of an application-level, optimized multi-level asynchronous checkpointing approach that first captures the checkpoints to local storage and then flushes them in the background to the PFS. GIO can use a variety of ways to flush the checkpoints to the PFS, including MPI I/O collective operations and POSIX read/write system calls. This allows GIO to be tuned to accommodate many system architectures and persistent storage options. GIO aggregates by default the checkpoints of N processes running on a large number of compute nodes into a smaller subset of M files. We call this N-M aggregation, which is illustrated in Figure 1a. M is tunable by the user to better match system composition such as storage targets, I/O servers, or other user-based constraints. For the purpose of this work, we configure GIO to use MPI I/O collective operations, and we let GIO automatically optimize M.

VELOC can capture the checkpoints on a variety of node-local memory hierarchies and storage devices (SSDs, HDDs). We study two POSIX options: in-memory using tmpfs (/dev/shm mount point) and SSDs using ext4. From here, we flush the checkpoints asynchronously to the PFS using a one-file-per process strategy (Figure 1b), which was chosen because of its high I/O performance under write concurrency (each file ends up on a single storage sever in its own dedicated stripe) and because it complements the I/O aggregation strategy used by GIO. A single active backend is responsible on each compute node for interacting with the PFS in an asynchronous fashion. We use the default settings, which results in a number of I/O threads on each active backend equal to the number of co-located application processes on each compute node. VELOC is modular and employs a wide array of other resilience strategies: partner replication, peer-to-peer erasure coding, flushing using burst buffers and key-value stores, etc. However, we have chosen to focus on a basic scenario that is the most frequently used in practice.

Compared Aspects: The comparison between GIO and VELOC uses a combination of alternative approaches that affect the resilience strategies: synchronous vs. asynchronous, I/O aggregation into a small set of files vs. one file-per-process, MPI collective I/O operations vs. POSIX I/O, direct writes to the PFS vs. leveraging a heterogeneous storage hierarchy. These have different behaviors and implications at scale both with respect to performance overheads and energy consumption, which results in a relevant multi-faceted comparison.

B. Measurements and Energy Monitoring

In this paper, our main focus is to characterize the energy scalability of asynchronous checkpointing, and different storage devices. To do this, we need the ability to capture the energy consumption of these processes. Thus, we require an energy monitoring tool that polls the hardware counters that exist at the kernel level for energy consumption of the CPU and DRAM. While there are various tools to accomplish this as mentioned in [29] we perform our tests on an Intel CPU and utilize the Running Average Power Limit (RAPL) [33] hardware counters. We measure these results with Performance Application Programming Interface (PAPI) [34].

Powercap and RAPL: Based on our cluster's kernel settings, our interface RAPL was through the *powercap* hardware counters. In Linux, these exist in the `/sys/devices/virtual/powercap` path and consists of several levels of telemetry data. These registers are divided into Zones 0 and 1 with a Subzone 0 and 1 in each Zone. These divisions represent different portions of the hardware. For example, Zones 0 and 1 capture the data for all cores of the CPU and the Subzones 0:0, 0:1, 1:0, 1:1 capture data for the DRAM on board [33]. For our case, with two zones and two subzones, if the energy of zones 0 and 1 are E_0, E_1 and subzones 0:0, 0:1, 1:0, 1:1 are $E_{0:0}, E_{0:1}, E_{1:0}, E_{1:1}$ then the total energy of the CPU and DRAM E_{CPU}, E_{DRAM} are:

$$E_{CPU} = E_0 + E_1 \tag{1}$$

$$E_{DRAM} = E_{0:0} + E_{0:1} + E_{1:0} + E_{1:1}. \tag{2}$$

We note that DRAM and CPU are the only available hardware counters on our system. Therefore, other presentation of total energy is based on these two terms. In this study we define Total Energy (kJ) to be

$$E_{total} = E_{CPU} + E_{DRAM}. \quad (3)$$

PAPI: To poll RAPL during application runtime, we use PAPI, an interface that allows users to collect program performance metrics via C library calls. This allows a user to determine what they would like to measure and when/where to measure it. In our case, we are interested in the energy consumption of the checkpoint operations of both synchronous and asynchronous C/R libraries. It is important to note that these constitute different methodologies, as C/R synchronicity changes the phases of a program, as touched on in Section II-A. Ignoring the setup of PAPI, in Algorithm 1 we demonstrate that synchronous C/R consists of a blocking checkpoint to persistent memory.

Algorithm 1 PAPI Measurement in C/R

Input: data_to_ckpt[], PAPI_EventSet, PAPI_results[]
Output: PAPI_values_arr
Note: PAPI_EventSet initalized with powercap events
1: PAPI_start(PAPI_EventSet);
2: CKPT_LIB.checkpoint(data_to_ckpt[]);
3: **if** VELOC_ASYNC **then**
4: *Local storage ckpt energy*
5: PAPI_read(PAPI_EventSet, PAPI_results[0]);
6: CKPT_LIB.wait();
7: **end if**
8: *Persistent storage ckpt energy*
9: PAPI_stop(PAPI_EventSet, PAPI_results[]);
10: **return** PAPI_results[]

On the other hand, asynchronous multi-level C/R first checkpoints to a local device of the user's choosing (in our case DRAM or an SSD), then flushes said checkpoints to persistent storage (PFS) in the background. For comparison, we are interested in the energy of both of these operations, therefore we read the results at two points as shown in lines 5 and 9 of Algorithm 1.

It is worth noting that we are unable to measure the energy consumption of the SSD and the HDD using PAPI or software available on the Palmetto Cluster. These storage drives are not equipped with hardware counters to poll the energy draw from the kernel-perspective, also we are unable to open the system and perform physical energy monitoring. We consider modeling these missing results by adding the average TDP of the devices multiplied by the runtime of the checkpoint to the total energy to model these results. However, in doing so we would scale everything by almost the same factor. Therefore, for clarity we choose to leave these modeled results out and instead only report the empirical data from PAPI and RAPL.

IV. Experimental Results and Discussion

In this section, we detail our experimental design for comparing the energy consumption of C/R libraries. We also present our results for energy usage of synchronous and asynchronous C/R writing to different hardware. In our tests, we evaluate these metrics along with throughput of the application in both weak and strong scaling contexts to present a more full evaluation of the merits of the different C/R options.

A. Experimental Platform

Hardware: We carry out our experiments on a multi-node HPC cluster with a 100 Gbps Infiniband interconnect. Each node is equipped with two Intel Xeon(R) Gold 6148 CPUs with a total of 40 cores, a max clock frequency of 2.40GHz, and 370GB of DDR4 DRAM. The system has various storage options, we summarize the storage locations we employ in Table I. Asynchronous C/R requires both a local memory location and a persistent storage location to write to once local checkpointing is complete. The persistent storage was chosen as it uses a PFS. Therefore the Usage column points out how each location is utilized.

TABLE I: Storage Hardware Summary

Location	Hardware	Size	File System	Usage
/dev/shm/	DRAM	370GB	tmpfs	Scratch
/localscratch/	SSD	1.8TB	ext4	Scratch
/scratch1/	HDD	1933TB	beegfs	Persistent

In our testing, we attempt to scale to four nodes using MPI for a total of 160 processes. However, the system in use provides only 24 I/O servers and fixed stripe settings of 4 storage targets and a stripe size of 512kB. The system prioritizes job compactness and throughput of small jobs versus large jobs. Since neither of the C/R libraries we use have any sort of node-level synchronization, it is unlikely that the performance degradation seen in the multi-node configuration as a result of these settings is indicative of the overall scalability of the C/R libraries. Thus, we only present single-node results.

In the following tests, the term local refers to VELOC's blocking write operation to local storage as a part of the multi-level checkpointing heuristic. GenericIO does not write to a scratch memory and is therefore not depicted in said figures.

Software: We utilize VELOC (v1.6) and GIO (Git tag 20190417) to perform C/R operations. To monitor energy, we use PAPI (v6.0.0) to read the RAPL counters on the aforementioned Intel CPUs and the cluster's operating system of Rocky Linux (v4.18.0). We interface with these hardware counters through PAPI (v6.0.0) and its C/C++ libraries. We compile our code using GCC 9.5.0 and OpenMPI 4.1.3.

B. Weak Scalability

Here we present the results for a series of weak scalability tests. For each approach and storage location from Table I, we spawn N ($1 \rightarrow 40$) processes. Each rank checkpoints a uniform size of either 1 or 2 Gigabytes (GiBs). We repeat each experiment 50 times and average the results, whose variability is depicted using errorbars.

Note that checkpointing ≥ 2 [GiB] per MPI rank experiences memory overflow errors due to MPI-IO aggregation in

the case of GIO. Therefore, we use their POSIX read/write implementation. VELOC is capable of using different I/O flushing methods: (1) *sendfile* system call that internally transfers data and (2) POSIX mmap/write; (3) regular read/write. For the purpose of this work, we use POSIX mmap/writes, since the sendfile system call is not efficiently implemented by our PFS.

Figures 2 and 3 illustrate how different local storage devices affect throughput and energy consumption when using VELOC. In Figure 2 we see that for local checkpoints, DRAM provides $\approx 25\%$ more throughput than the SSDs as processes begin to scale. DRAM supports higher throughput in general, but this actually has a direct impact on the energy use. Only as the throughput begins to differ significantly do we start to see a gap in energy consumption between the two devices. Thus, we conclude throughput dictates energy consumption.

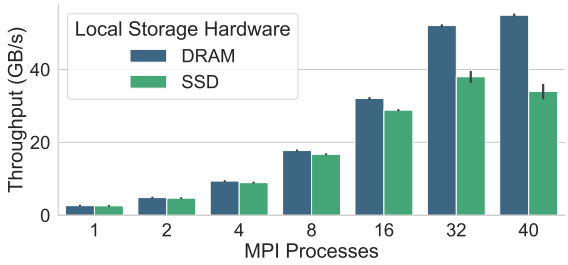
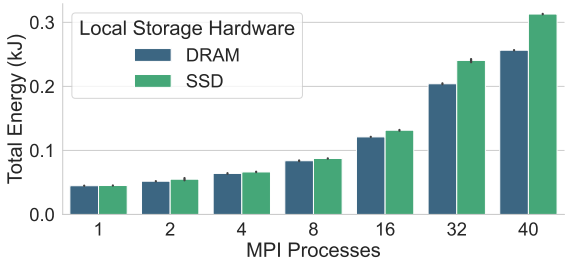


Fig. 2: Weak scalability of I/O throughput for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Higher is better.

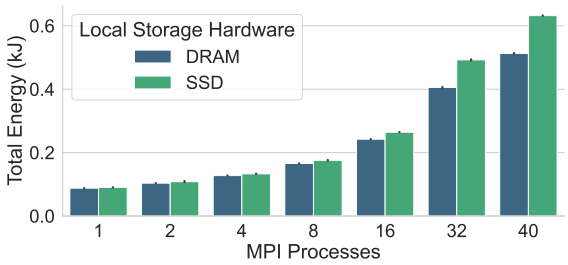
In Figure 3 the energy consumption is shown to be linearly correlated with the size of data, as illustrated in the difference between Figures 3a and 3b. Energy cost is dominated by the magnitude of data movement, as checkpointing 2 GiB per rank requires $\approx 2\times$ more energy than 1 GiB, regardless of the scratch memory location. On average, writing locally to DRAM takes $\approx 20\%$ less energy than to a local SSD.

Therefore, weak scaling reveals that asynchronous checkpoints to DRAM has a higher throughput and lower energy cost, incentivizing its use in C/R. However, this is really only a nominal difference. In a typical scientific application, DRAM is a heavily contested resource which may not be large enough to store both application data structures and checkpoints. Further, reduction techniques (such as compression) are becoming increasingly necessary to address the exponentially growing size of data, meaning it is unlikely that distributed pieces of a global checkpoint will be a uniform size. Therefore, future C/R development should work towards efficiently using both storage tiers by assigning larger data regions to DRAM.

Figures 4 and 5 compare the overall throughput and energy consumption of both C/R libraries when VELOC writes to DRAM locally. Both strategies use the PFS as their persistent storage tier. These results show that GIO consumes on average $\approx 4\times$ the amount of energy VELOC does. Similar to energy consumption of the local checkpointing phase, we see that



(a) 1 GB per Rank



(b) 2 GB per Rank

Fig. 3: Weak scalability of energy consumption for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Lower is better.

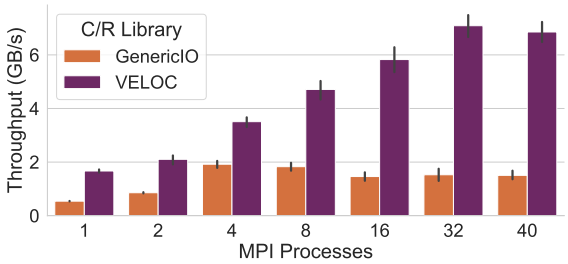


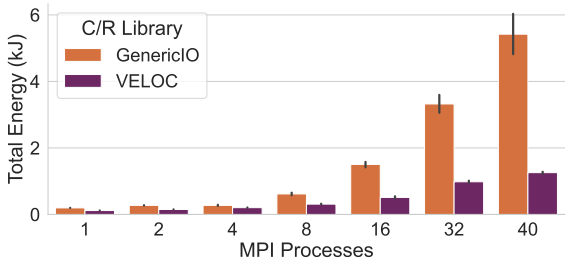
Fig. 4: Weak scalability of total I/O throughput (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Higher is better.

as the data assigned per rank doubles, so does the resulting energy consumption.

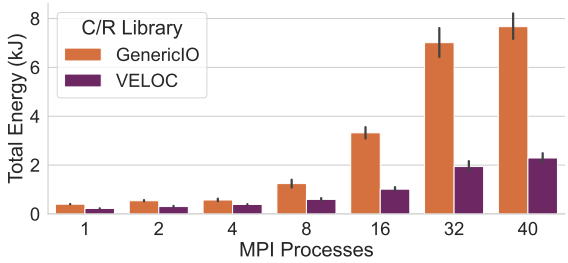
C. Strong Scalability

For our strong scalability tests, we evaluate workloads using N $(1 \rightarrow 40)$ processes on aggregate problem sizes of 40 and 80 GiB. Since we are consistently writing over 1 GiB, GIO uses POSIX read/writes and VELOC continues to use mmap/writes.

To evaluate how parallelism affects the throughput and energy of the local storage tiers, we analyze these metrics through a series of strong scalability experiments in Figures 6 and 7, respectively. In these experiments, we see the same overall trends in the weak scaling experiments: DRAM provides (1) higher throughput and (2) lower energy consumption than the SSD during local checkpointing. This similarity further supports the idea that better throughput directly leads



(a) 1 GB per Rank



(b) 2 GB per Rank

Fig. 5: Weak scalability of total energy consumption (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Lower is better.

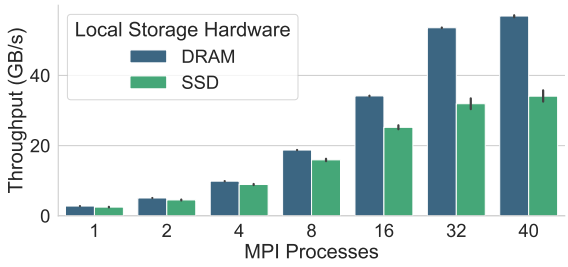
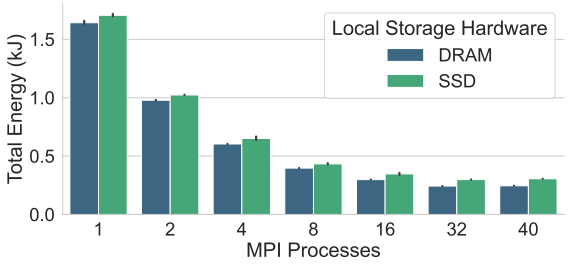
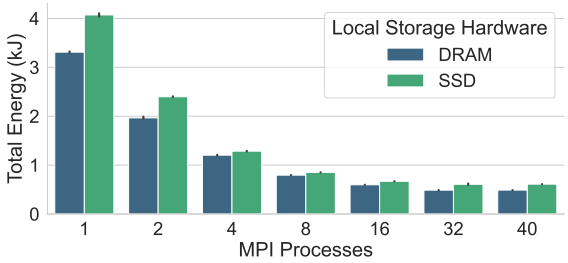


Fig. 6: Strong scalability of I/O throughput for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Higher is better.



(a) 40 GB Problem Size



(b) 80 GB Problem Size

Fig. 7: Strong scalability of energy consumption for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Lower is better.

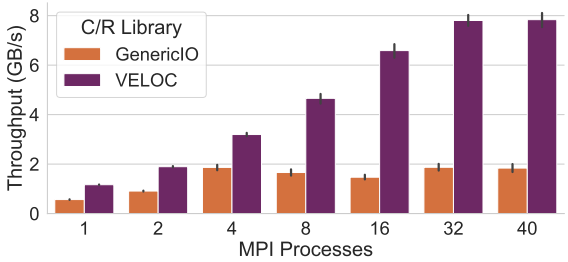


Fig. 8: Strong scalability of total I/O throughput (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Higher is better.

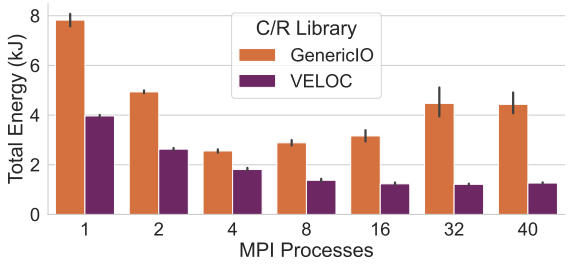
to more efficient energy usage. Therefore, future efforts to improve energy efficiency should focus on ensuring high throughput performance rather than throttling.

Figures 8 and 9 compare the throughput and energy consumption of the C/R libraries, respectively. In these experiments, we show that asynchronous file-per-process strategies continue to provide the highest throughput, without consuming extra energy. Even though the asynchronous strategy requires more operations to (1) write the checkpoint to a local storage device, and (2) buffering the checkpoint to transfer externally, this is more energy efficient than directly interacting with the PFS. There are a few possible explanations for this: (1) writing to a local storage tier and then asynchronously flushing spreads out writes to the PFS such that not all processes (or threads) are competing for I/O servers at the same time, thereby improving throughput and energy consumption; and (2) file aggregation suffers serialization at the I/O server level (in the case of poorly matched I/O strategies and stripe settings) due to false sharing.

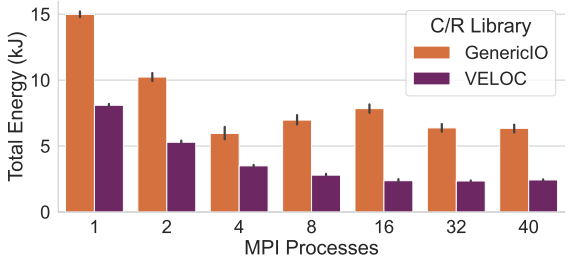
In figures 9a and 9b, after 8 or more processes are used, the energy consumed by GIO begins to rise again, despite the fact that the size per process is shrinking. We conclude that uncoordinated file aggregation, such as the case in POSIX writes, coupled with poor stripe settings causes CPUs to spend time (and therefore energy) waiting for access to the storage servers due to false sharing. Compared to VELOC, which is using a file-per-process strategy, energy consumption continues to decrease as more processes are added. Therefore, we conclude that the flush strategy has an impacting effect on energy consumption.

V. Future Works and Areas of Improvement

There are numerous ways this work can be expanded and improved upon.



(a) 40 GB Problem Size



(b) 80 GB Problem Size

Fig. 9: Strong scalability of total energy consumption (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Lower is better.

First, the experiments used to characterize the synchronous checkpointing library, GIO, were done using the POSIX implementation of C/R, which relies on uncoordinated file aggregation that issues regular read/write calls. While simple to set up and use, this leads to significant performance degradation as a side effect of file striping. Other techniques based on MPI I/O collectives are available in GIO and perform I/O reorganization and aggregation to better match the underlying stripe settings. This may improve performance and scalability at the expense of more energy consumption. In this context, our own previous work [35] also considers file aggregation in the context of asynchronous checkpointing, which introduces additional trade-offs. Due to time considerations, we did not explore such a comparison in this work, but plan to do so in the future.

Secondly, we plan to expand upon the methods of energy monitoring and recommendations we present. We aim to utilize a local cluster with the ability to physically measure the energy consumption of storage devices, allowing for more detailed recommendations about hardware energy-efficiency. By having this specific data, one could describe how the load of C/R software differences affect the hardware in use.

Finally, in this work our goal was to isolate the energy consumption of checkpointing from any other external influence in synthetic benchmarks. However, in a real-life application, the overlap between asynchronous multi-level checkpointing strategies and the application runtime may lead to interesting interference patterns that affect the energy consumption beyond the individual behaviors in isolation. Thus, in future work, we aim to study such effects in the context of real-life applications as well.

VI. Conclusions

In this paper, we evaluate two different C/R libraries and their various configurations in order to compare the energy consumption of multi-level asynchronous C/R and single-level synchronous C/R. We use this information to better help application and C/R developers identify configurations and areas of C/R that contribute to high energy overheads. Overall, we find that throughput heavily impacts energy consumption. Thus, asynchronous C/R to DRAM using file-per-process flushing strategies utilize the least amount of energy. We summarize our main observations below.

We note that energy consumption is directly tied to throughput. Thus, C/R should focus on providing high throughput in order to achieve low energy costs. In the context of asynchronous checkpointing, using DRAM as the local storage tier is shown to provide the highest throughput as compared to SSDs. However, in real-world scientific computing it is a heavily contended resource. Therefore, asynchronous C/R should focus on *efficiently* utilizing DRAM to balance the use of shared resources that reduce contention between applications and C/R to maximize throughput, thereby lowering energy consumption.

Furthermore, we show that flushing strategies like synchronous file aggregation have a significant impact on energy consumption. Our results at modest scale show that file-per-process strategies provide the highest throughput and consume the least amount of energy. However, file-per-process strategies may encounter I/O bottlenecks at scale due to metadata bottlenecks, which typically is not handled in a scalable fashion by many PFS implementations. Thus, considering the trade-offs between performance and energy efficiency that file aggregation introduces at scale both for synchronous and asynchronous checkpointing are non-trivial and need to be studied in further detail.

Encouraged by these observations, we plan to explore in future work several follow-up directions, as outlined in Section V.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation under Grants SHF-1910197 and SHF-1943114. This research was supported in part by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. This material was based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

[1] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello, "Veloc: Towards high performance adaptive asynchronous checkpointing at large scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 911–920.

[2] (2022) top500 results for june 2022. [Online]. Available: https://www.top500.org/lists/top500/2022/06/

[3] Z. Zong, R. Ge, and Q. Gu, "Marcher: A heterogeneous system supporting energy-aware high performance computing and big data analytics," *Big Data Research*, vol. 8, pp. 27–38, 2017, tutorials on Tools and Methods using High Performance Computing resources for Big Data. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221457961630048X

[4] J. Bent, B. W. Settlemyer, and G. Grider, "Serving data to the lunatic fringe: The evolution of hpc storage," *login Usenix Mag.*, vol. 41, 2016.

[5] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, p. 181, 2008.

[6] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *High Performance Computing for Computational Science – VECPAR 2010*, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–25.

[7] M. Schulz, D. Kranzlmüller, L. B. Schulz, C. Trinitis, and J. Weidendorfer, "On the inevitability of integrated hpc systems and how they will change hpc system operations," in *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, ser. HEART '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3468044.3468046

[8] S.-M. Tseng, B. Nicolae, F. Cappello, and A. Chandramowlishwaran, "Demystifying asynchronous i/o interference in hpc applications," *The International Journal of High Performance Computing Applications*, vol. 35, 2021.

[9] S.-M. Tseng, B. Nicolae, G. Bosilca, E. Jeannot, A. Chandramowlishwaran, and F. Cappello, "Towards portable online prediction of network utilization using mpi-level monitoring," in *EuroPar'19 : 25th International European Conference on Parallel and Distributed Systems*, Goettingen, Germany, 2019, pp. 47–60. [Online]. Available: https://hal.inria.fr/hal-02184204

[10] F. Isaila, J. García, J. Carretero, R. B. Ross, and D. Kimpe, "Making the case for reforming the i/o software stack of extreme-scale systems," *Adv. Eng. Softw.*, vol. 111, pp. 26–31, 2017.

[11] M. Morán, J. Balladini, D. Rexachs, and E. Luque, "Prediction of energy consumption by checkpoint/restart in hpc," *IEEE Access*, vol. 7, pp. 71 791–71 803, 2019.

[12] D. Dauwe, R. Jhaveri, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Optimizing checkpoint intervals for reduced energy use in exascale systems," in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, 2017, pp. 1–8.

[13] Z. Miao, J. Calhoun, and R. Ge, "Energy analysis and optimization for resilient scalable linear systems," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 24–34.

[14] M. A. Amrizal and H. Takizawa, "Optimizing energy consumption on hpc systems with a multi-level checkpointing mechanism," in *2017 International Conference on Networking, Architecture, and Storage (NAS)*, 2017, pp. 1–9.

[15] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W. keng Liao, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, jan 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1410.2805

[16] J. Ansel, K. Arya, and G. Cooperman, "Dmtcp: Transparent checkpointing for cluster computations and the desktop," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–12.

[17] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC '10: The 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, New Orleans, USA, 2010, pp. 1:1–1:11.

[18] N. El-Sayed and B. Schroeder, "To checkpoint or not to checkpoint: Understanding energy-performance-i/o tradeoffs in hpc checkpointing," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, pp. 93–102.

[19] M. Alfian Amrizal and H. Takizawa, "Optimizing energy consumption on hpc systems with a multi-level checkpointing mechanism," 08 2017, pp. 1–9.

[20] M. e. M. Diouri, O. Glück, L. Lefèvre, and F. Cappello, "Ecofit: A framework to estimate energy consumption of fault tolerance protocols for hpc applications," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013, pp. 522–529.

[21] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future generation computer systems*, vol. 22, no. 3, pp. 303–312, 2006.

[22] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 206–215.

[23] L. Wan, Q. Cao, F. Wang, and S. Oral, "Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems," *Journal of Parallel and Distributed Computing*, vol. 100, pp. 16–29, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731516301198

[24] M. Morán, J. Balladini, D. Rexachs, and E. Luque, "Prediction of energy consumption by checkpoint/restart in hpc," *IEEE Access*, vol. 7, pp. 71 791–71 803, 2019.

[25] L. Tan, Z. Chen, and S. L. Song, "Scalable energy efficiency with resilience for high performance computing systems: A quantitative methodology," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, nov 2015. [Online]. Available: https://doi.org/10.1145/2822893

[26] G. Wilkins and J. C. Calhoun, "Modeling power consumption of lossy compressed i/o for exascale hpc systems," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2022, pp. 1118–1126.

[27] T. Saito, K. Sato, H. Sato, and S. Matsuoka, "Energy-aware i/o optimization for checkpoint and restart on a nand flash memory system," in *Proceedings of the 3rd Workshop on Fault-Tolerance for HPC at Extreme Scale*, ser. FTXS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 41–48. [Online]. Available: https://doi.org/10.1145/2465813.2465822

[28] P. Llopis, M. Dolz, J. Blas, and et al., "Analyzing the energy consumption of the storage data path," *Journal of Supercomputing*, vol. 72, pp. 4089–4106, 2016.

[29] P. Czarnul, J. Proficz, and A. Krzywaniak, "Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments," *Scientific Programming*, vol. 2019, p. 8348791, Apr 2019. [Online]. Available: https://doi.org/10.1155/2019/8348791

[30] N. El-Sayed and B. Schroeder, "Understanding practical tradeoffs in hpc checkpoint-scheduling policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 2, pp. 336–350, 2018.

[31] E. Stafford and J. L. Bosque, "Performance and energy task migration model for heterogeneous clusters," *The Journal of Supercomputing*, vol. 77, no. 9, pp. 10 053–10 064, Sep 2021. [Online]. Available: https://doi.org/10.1007/s11227-021-03663-1

[32] R. R. Chandrasekar, A. Venkatesh, K. Hamidouche, and D. K. Panda, "Power-check: An energy-efficient checkpointing framework for hpc clusters," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 261–270.

[33] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 189–194. [Online]. Available: https://doi.org/10.1145/1840845.1840883

[34] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173.

[35] M. J. Gossman, B. Nicolae, J. C. Calhoun, F. Cappello, and M. C. Smith, "Towards aggregated asynchronous checkpointing," *ArXiv*, vol. abs/2112.02289, 2021.