Hyrax: Fail-in-Place Server Operation in Cloud Platforms

Jialun Lyu[1,2] Marisa You[1] Celine Irvene[1] Mark Jung[1] Tyler Narmore[1] Jacob Shapiro[1]
Luke Marshall[1] Savyasachi Samal[1] Ioannis Manousakis* Lisa Hsu* Preetha Subbarayalu[1]
Ashish Raniwala[1] Brijesh Warrier[1] Ricardo Bianchini[1] Bianca Schroeder[2] Daniel S. Berger[1,3]
[1]*Microsoft Azure* [2]*University of Toronto* [3]*University of Washington*

Abstract

Today's cloud platforms handle server hardware failures by shutting down the affected server and only turning it back online once it has been repaired by a technician. At cloud scale, this all-or-nothing *operating model is becoming increasingly unsustainable. This model is also at odds with technology trends, such as the need for new cooling technology.*

This paper introduces Hyrax, a datacenter stack that enables compute servers with failed components to continue hosting VMs while hiding the underlying degraded capacity and performance. A key enabler of Hyrax is a novel model of changes in memory interleaving when deactivating faulty memory modules. Experiments on cloud production servers show that Hyrax overcomes common hardware failures without impacting peak VM performance. In large-scale simulations with production traces, Hyrax reduces server repair requirements by 50-60% without impacting VM scheduling.

1 Introduction

Server hardware failures are quite frequent in cloud platforms. For example, a typical cloud server relies on at least 24 DIMMs, six SSDs, six fans, and two CPU sockets [48]. Even assuming optimistic annual failure rates[1] of 0.1% per DIMM and 0.2% per SSD, *22% of servers will have at least one failure* during the typical 6-year lifetime of a cluster. In practice, repair rates are typically even higher.

The common approach to dealing with hardware failures in today's cloud platforms is to evict all virtual machines (VMs) and stop using the affected server. The server goes back into production only once a technician has replaced all faulty components. This maintains server homogeneity, which simplifies scheduling and operation [4, 24, 32, 41, 42, 46, 61, 64, 69]. We call this the *"all-or-nothing"* operating model.

Recent technology trends make all-or-nothing operations increasingly unsustainable in cloud platforms. First, server power consumption increasingly requires liquid cooling, which offers performance, efficiency, and sustainability benefits [33, 63, 72]. Liquid cooling significantly increases the time and effort required to repair servers. Second, the share of total costs that are due to repairs are increasing (§2). This is in part due to servers staying in datacenters for longer[2]. Third, all-or-nothing requires a continuous supply of spare components, which is increasingly hard to procure. Component supply chains have emerged as a barrier to further extending server lifetimes and reducing carbon emissions [7]. Fourth, the human repair process can cause interruptions to nearby servers [32], which is becoming an obstacle in cloud provider's pursuit to improve the availability of their servers.

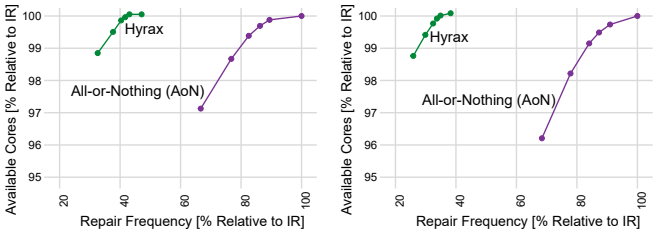
This paper advocates that cloud providers should move toward a *fail-in-place* paradigm where servers with faulted components continue to host VMs without requiring repairs. Fail-in-place operation would significantly reduce repair needs, improving costs, carbon emissions, and availability. However, fail-in-place faces multidimensional challenges in practice. First, it requires a form of graceful degradation where individual faulty components are deactivated instead of decommissioning the entire server. Unfortunately, we find that mechanisms to deactivate components are largely undocumented. Furthermore, deactivating the right component requires accurate fault diagnostics and it is unclear whether this can be achieved in practice. Second, deactivating common components such as DIMMs can significantly impact server performance due to reduced memory interleaving. This performance loss should not be exposed to VM customers. Third, the cloud platform must be able to actually use the capacity on servers with deactivated components. This requires algorithmic changes in VM scheduling and changes to adopt the cloud control plane to support heterogeneous servers.

We introduce Hyrax— the first implementation of the fail-in-place paradigm for cloud compute servers. In a multi-year study of component failures across five server generations, we find that sufficient redundancy in existing servers can overcome the most common memory and SSD device failures. While existing diagnostics can only identify a subset of component types, we empirically find that they are 95% accurate. We identify hooks in deployed firmware that enable deactivating components in ways that overcome many failure possibilities (e.g., dirty or corroded connectors or chip failures). Finally, Hyrax adds a *degraded* server state and corresponding scheduling rules to a production control plane to

*Formerly at Microsoft Azure

[1]Prior work reported 0.09% [58, 59], 0.12% [12], and 1.6% [57, 66] for DIMMs and 0.22% [43, 44] to 1.2% [3, 54] for SSDs.

[2]Major cloud providers have moved to a minimum server lifetime of six years [7, 26, 53] for cost and sustainability reasons.



(a) US Region. **(b)** EU Region.

Figure 1: Hyrax dominates all-or-nothing (AoN) operations along the entire trade-off spectrum between available resources (core hours) and the number of required server repairs (repair tickets). Different points on the trade-off spectrum are generated by varying the repair schedule, ranging from immediate repairs (IR) to performing repairs in batches at periodic intervals ranging from 1-12 months long. All numbers are normalized to those for AoN with immediate repairs, which is the common approach in today's cloud platforms.

support servers with deactivated components.

Hyrax overcomes the reduced performance of degraded servers by exploiting existing heterogeneity in VM sizes and configurations. Specifically, we find that the peak performance expectation of small and old VM types matches the performance offered by degraded servers. Further, we find that there are sufficiently many small and old VM types to effectively utilize the capacity of degraded servers. Hyrax also introduces scheduling optimizations for efficiency at scale.

Hyrax has been deployed for a few months on a subset of Azure clusters and a small set of component types. We report on its effectiveness on real failures and use microbenchmarks and large-scale trace-driven simulations to extrapolate a full deployment over six years. Our experience demonstrates that the fail-in-place paradigm is practical under real-world platform constraints.

To evaluate the benefits of at-scale deployment, we simulate 66 compute clusters from two geographic regions over a period of six years. Overall, Hyrax reduces the number of server repairs in a region by 50-60% depending on the region (Figure 1), while offering the same resource availability and scheduling the same VMs as today's all-or-nothing operation. Figure 1 also shows that Hyrax's benefits carry over to different repair schedules, including Azure's existing repair schedule (immediate repairs) as well as previously-suggested batching of repairs [4, 5], where repairs are scheduled at periodic intervals (e.g. once per year). Furthermore, Hyrax reduces replacement rates by 40% for fans, 50% for SSDs, and 75% for memory, which enables extending server lifetimes for multiple years to amortize server costs and carbon emissions.

We hope that, by sharing our journey towards the fail-in-place paradigm, we motivate the community to invest in future cross-stack systems research to make degraded mode and fail-in-place operation significantly more efficient.

Contributions:

- The first description of design goals and constraints for fail-in-place and feasibility analysis of degraded mode operation at a large public cloud platform (§3).
- The design and implementation of Hyrax, the first fail-in-place system at a cloud provider. Hyrax's implementation includes novel mechanisms to deactivate component pathways and a novel model of memory interleaving when memory modules are deactivated (§4, §5, and § 6).
- Experimental results that show Hyrax's effectiveness, performance, and cost impacts (§7).
- A discussion of deployment experience, broader impacts, and research avenues (§9).

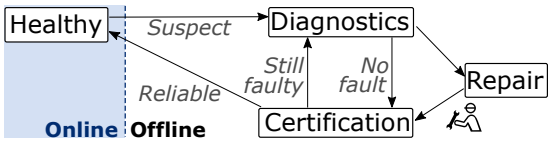


Figure 2: At Azure, servers are either online and serving VMs, or offline and being repaired. Repairs take between 3 and 190 days at the 50-th and 99-th percentile, respectively.

Limitations. Hyrax is not applicable to all repair operations. The following assumptions underpin our work.

- Hyrax focuses on server repairs, which account for the majority of technician hours in Azure datacenters. Hyrax does not reduce other technician duties, such as power, network, and cooling maintenance.
- Hyrax focuses on compute servers, where degraded operation is challenging. Storage servers often already implement variants of degraded mode (§8).

2 Background

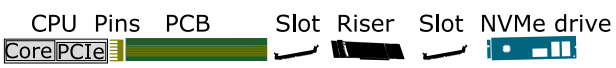
This section reviews repair workflows and costs, typical server configurations, and cloud workloads.

Repair workflow. A software agent called Server Health Monitor (SHM) checks server error logs and component types, counts, and capacity for deviations from the expected (homogeneous) configuration. If the SHM suspects any kind of fault, the server is marked as "offline", which signals the VM scheduler to filter out this server (Figure 2). VMs are migrated away or gracefully evicted. The server is then rebooted into a diagnostics environment. If diagnostics finds a hardware problem, it immediately creates a *repair ticket* [4, 32, 41, 66].

Repair tickets can point to a specific component pathway (like DIMM #4, Figure 3a) or require a manual diagnosis. After a technician resolves a ticket, e.g., by reseating connectors or swapping out components, the server is tested again to certify reliability (certification step). A reliable server is marked "online" and again becomes a candidate for hosting VMs.

Impact of all-or-nothing repairs on TCO. Server repairs are a significant component of total cost of ownership (TCO). The main components of TCO are CapEx (capital expendi-

(a) Memory component pathway.



(b) SSD component pathway.

Figure 3: A component can appear faulty due to other component faults along the path between a core and the actual component. We call this a *pathway* that typically spans the socket and pins, printed circuit board (PCB), and slots/risers to the actual component like the NVMe SSD or memory DIMM.

ture for the purchase of servers, networking, cooling, and power infrastructure) and operational costs due to energy and power (estimated at 6% of CapEx per year [19, 20, 62]), and maintenance (estimated at 5% of CapEx per year for each server [4, 66]). Maintenance costs are largely made up by technician salaries and cover maintenance of all datacenter components. At Azure, server repairs account for about half of technician work hours in the all-or-nothing operating model. Server repairs thus account for 9% and 12% of total cost (TCO) for server lifetimes of 6 and 10 years [7], respectively.[3]

Repairs are also known to be slow [69]. At Azure, 2% of servers are waiting for repairs at any given time in the all-or-nothing operating model.

Server hardware. Figure 4 shows a typical cloud server configuration [48]. Variants of this base architecture include one or two NICs and 24-32 DIMMs; most servers use a single NIC. We note that the component count for some component types is larger than one (marked in green in Figure 4). We refer to these as *degradable* components as they do not represent a single point of failure.

We note that hardware components internally contain redundancy, such as spare blocks in SSDs [8, 25, 34, 44, 52, 55, 67, 68]. Moreover, the operating system and hypervisor at Azure employ an aggressive policy for offlining memory pages to mask faulty cachelines. A repair ticket is generated for a component only when the above mechanisms cannot resolve the problem.

Cloud workload. All workloads run within virtual machines (VMs) for security and ease of management. Resources for each VM are typically preallocated at its start time to improve performance and facilitate the use of virtualization accelerators [2, 39, 60, 70, 71]. VMs come in hundreds of different types with many combinations of the number of virtual cores, memory capacity, local and remote storage options, NIC and GPU configurations.

The cloud provider has no introspection into the workloads that a customer is running inside their VMs and does not know their performance requirements. Hence, performance goals

[3] We calculate TCO based on the three dominant cost factors: Deployment years (y), CapEx (C), Maintenance ($y \times C \times 5\%$), and Energy/Power ($y \times C \times 6\%$). This leads to $TCO(y) = C + y \times 5\% \times C + y \times 6\% \times C = C(1 + 0.11 \times y)$.

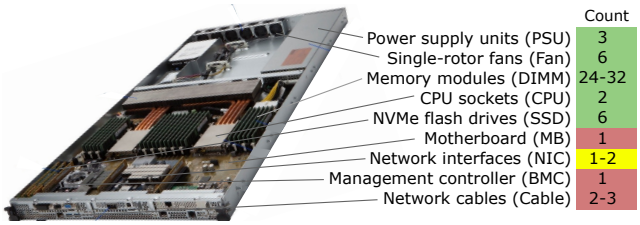


Figure 4: A typical cloud server configuration and its component counts. We refer to the component types marked in green as *degradable*, as their component count is large enough that they are not a single point of failure.

are defined in terms of peak performance, e.g., bandwidth and latency for memory and IOPS and bandwidth for SSDs. For older VM types that are scheduled on newer servers, their performance goals are defined for the server generation they were originally introduced on.

Azure's distributed VM scheduler is called Protean [1, 9, 22, 37, 61]. Protean first forwards VM requests to a compute cluster within the specified region based on hardware requirements and available capacity. At the cluster level, Protean places VMs following a series of rules that balance tightly packing resources with spreading workloads across racks for high availability. Filter rules select which servers are considered candidates for placing each VM. They ensure that only servers are considered that can ensure the SLAs associated with the requested VM type. Preference rules rank these candidates to find the best placement. Similar to other schedulers [4, 22, 24, 32, 41, 42, 46, 61, 64, 69], Protean assumes identical hardware configurations for all servers within a cluster.

3 Fail in Place

The "all-or-nothing" operating model and the associated high repair frequency is costly and at odds with multiple server and data center trends. This paper pursues an alternative paradigm, which we term Fail-in-Place (FIP). In FIP, servers are allowed to exist with failed components for prolonged periods of time, sometimes forever. The main goal of FIP is to *reduce repair tickets* while continuing to offer the same user experience to VMs and minimal impact on cluster capacity and scheduling.

FIP is motivated by our observation that the majority of hardware repair tickets are due to the failure of *degradable* components. Consider Figure 5, which breaks down repair tickets at Azure into the component type that triggered them. We see that, for example, in Generation 3 clusters[4] more than 65% of tickets are due to degradable components. Recall from Section 2 that degradable components do not represent a single point of failure as their component count per server is larger than one.

[4] Higher server generations reflect newer server and component architectures. Generation 3 is a currently highly utilized hardware generation.

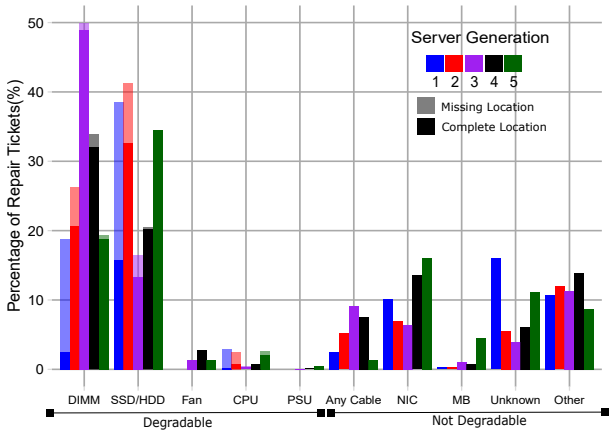


Figure 5: Breakdown of repair tickets at Azure into the component type responsible for the ticket. The repair tickets were recorded on dozens of production clusters spanning regions across two continents and clusters from five different hardware generations, which have been deployed between 2018 to 2022. Most repair tickets in server generations later than 2 are for degradable components and diagnostics indicates a specific pathway.

For degradable component types, Figure 5 further marks in a darker shade the share of tickets that also identify a specific pathway, rather than just the component type. For example, for DIMMs, these tickets would include the specific DIMM slot (recall Figure 3a). We observe that for generations above 2, almost all repair tickets among degradable components also indicate the specific pathway.

The key idea behind FIP is to avoid repair tickets by deactivating (rather than repairing) a faulty degradable component and allowing the server to continue to host customer VMs, albeit with reduced capacity. We refer to this new server state as *degraded* servers.

While Figure 5 illustrates FIP's potential to reduce repair tickets, a real FIP implementation must also satisfy the following constraints.

- $C_{\text{Performance}}$ VMs placed on degraded servers must still be able to achieve the same peak performance (e.g. memory bandwidth) expected for this VM type (§2).
- $C_{\text{Efficiency}}$ A FIP system must be able to effectively use the capacity on degraded servers. For example, it must not strand one resource (e.g., CPUs) because another resource is degraded (e.g., memory).
- C_{Capacity} A FIP system must continue to be able to satisfy a region's demand for VM resources. In particular, VMs must not be turned away from a region because of server degradation or disrepair.

For cloud platforms, FIP system design can be guided by the following observations based on real-world cloud workloads and failure patterns.

First, a majority of VMs that customers are running belong to smaller VM types that can be accommodated on a degraded

Requested Cores	Core-hours v3	Core-hours pre-v3
≤ 2	27.7%	26.9%
(2,4]	26.8%	16.9%
(4,8]	21.5%	18.9%
(8,16]	10.6%	16.6%
>16	13.4%	20.7%

Table 1: Core counts for VMs introduced with 3rd-generation servers (v3) and with previous-generation servers (pre-v3).

mode server without impacting their performance. For example, Table 1 shows a breakdown of core hours by VM type at Azure. VMs with four or fewer cores account for 40-50% of all core hours and are small enough that they require only a small fraction of a server's full capacity to achieve their expected performance.

Second, our study of server repair tickets at Azure reveals that the number of component failures per server is typically small compared to a server's total component count. For example, for servers in Generation 3, 90% of servers that develop SSD and/or DIMM failures in a one-year period exhibit two or fewer failures. The most common failure patterns among those servers are one failed DIMM (36.5%) followed by one failed SSD (10.3%). Hence for the bulk of servers with failures, deactivating the affected components would reduce the server's capacity by only a small fraction (recall that typical server configurations include 24-32 DIMMs and 6 SSDs) and not cause a significant amount of resource fragmentation. We note however that over long time periods, more than a few components will fail. To prevent resource stranding, any FIP system must thus control how many components can be deactivated in any degraded server.

Third, we find that FIP systems will still have to accommodate some repairs (albeit at a greatly reduced frequency) in order to satisfy capacity requirements. While servers with failures of degradable components are returned to online status, the capacity loss due to servers with failures of undegradable components (which will stay offline in the absence of repairs) is not acceptable.

The design and implementation of a complete FIP system pose multiple open challenges not captured in the simple vision above. For example, FIP requires accurate diagnostics, mechanisms to deactivate component pathways, a detailed understanding of how component deactivation impacts performance, policies to determine when to degrade (versus repair) a server, and a control plane that supports FIP (including the VM scheduler and automated diagnostics).

4 Hyrax System Design

Hyrax is a concrete implementation of the FIP idea and the first FIP system at a cloud provider. Hyrax implements a new "degraded" online server state on servers and in the control plane and changes multiple aspects of the offline workflow at Azure. Currently, Hyrax supports three degradable component

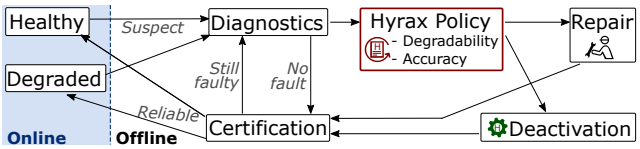


Figure 6: Server states in Hyrax.

types: memory, SSDs, and fans.

Figure 6 provides an overview of server states in Hyrax. After a server is marked as suspect, results from Diagnostics are used by the Hyrax Policy ( ) to decide whether to degrade or repair. This policy applies first filters for degradable component types. Second, it verifies that diagnostics points to a specific pathway within this component type. Third, it applies a threshold on how many components of each type can be degraded. Degraded servers are created by deactivating the faulty component pathway (, §5.1). Repairs are scheduled for undegradable component types, when diagnostics cannot identify the faulty component pathway, or if deactivation would cross the policy's threshold. Degraded and repaired servers are subject to extensive testing (called Certification in §2 and Figure 2) before becoming available for hosting VMs (online).

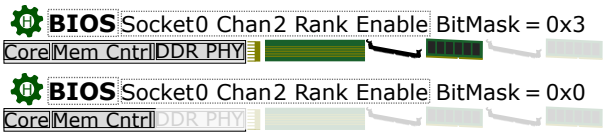
Hyrax achieves $C_{\text{Efficiency}}$ via the policy's thresholds. Currently, we never deactivate more than two components of any type. Empirically, we find that this is sufficient to prevent resource stranding. We provide a detailed sensitivity analysis in Section 7.4.

Hyrax achieves $C_{\text{Performance}}$ by characterizing how deactivating components affects VM performance for different VM types. This allows Hyrax to decide whether the remaining healthy components are sufficient for the server to continue serving VMs and which VM types it can serve without impacting user experience. Hyrax modifies the VM scheduler such that *only the VM types whose performance requirements can be met* are scheduled on the degraded server.

Hyrax minimizes repair tickets because many servers that are degraded instead of repaired will not encounter another fault during their deployed period. If degraded servers encounter another fault that cannot be degraded, Hyrax issues a single repair ticket and technicians repair all faults on the server at once. We call this technique *"mini-batching"*. Mini-batching effectively amortizes technician work like the journey to the server's rack, identifying and opening the server, manual diagnosis, and record keeping.

Hyrax achieves C_{Capacity} in two ways. First, the capacity an individual degraded server can lose is limited via the policy's thresholds. Second, undegradable servers are not permanently left offline without repairs. We consider a range of different repair schedules (§7).

We discuss technical details of the Hyrax server design in Section 5 and the Hyrax policy and control plane in Section 6.



(a) Deactivating a failed memory component pathway.

(b) Deactivated SSD component pathway.

Figure 7: Component deactivation takes care of entire paths of error sources, such as memory controller, DDR phy, PCB, connector, and DIMM itself. This has the potential to improve over repairs where reseating or exchanging the DIMM often does not resolve the problem.

5 Hyrax Servers

Hyrax seeks to convert an offline server with one or multiple faulty component pathways into a degraded server that can host VMs. Hyrax focuses on memory, SSD, and fans as the most common degradable components (§3). This section describes how to deactivate these three component pathways and associated performance implications.

5.1 Component Pathway Deactivation ( )

The key challenge is making component deactivation comprehensive enough so that faults are effectively hidden. Hyrax achieves this by deactivating components using combined firmware and software mechanisms.

Memory pathway. Hyrax targets memory errors that cannot be resolved by existing, fine-grained mitigations [8, 13, 35, 55, 67]. Common causes are uncorrectable errors across a DIMM's banks/ranks, connector problems, or too many faulty rows. Hyrax exploits a rarely-documented firmware (BIOS) feature, called `Rank Enable BitMask`. On Azure servers, this setting offers a bitmask for each channel, on each memory controller (MC), and on each socket. Each bitmask controls which of the DIMM's ranks on this channel are included in memory interleaving. Azure diagnostics currently only provides DIMM-level information, so Hyrax deactivation always excludes all ranks on a DIMM. Excluding an entire DIMM means that this DIMM's memory is not assigned an address. Furthermore, the MC will not attempt to control or refresh any data on that DIMM's memory chips. Figure 7a shows examples of deactivating one DIMM (0x3) as well as the whole memory pathway (0x0).

Hyrax has two ways to set the `Rank Enable BitMask`. If the server is able to boot a minimal OS, Hyrax software directly sets the bitmask in the BIOS configuration flash. If the server does not boot, Hyrax can set the bitmask via the Baseboard Management Controller (BMC) on the management network.

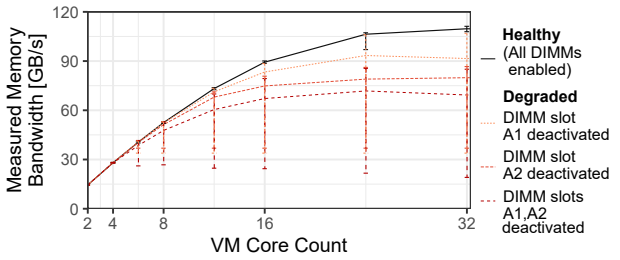


Figure 8: Peak memory bandwidth of a naïve implementation of Hyrax as measured from VMs on servers with all DIMMs enabled, one DIMM or two DIMMs deactivated, respectively.

SSD pathway. Server-local storage for VMs is striped across six NVMe drives. This configuration improves peak performance for IO-intensive VM types and facilitates bin packing hundreds of VM types with server-local storage. We modify the striping software module to read a list of serial numbers to include into the stripe. To deactivate an SSD pathway, Hyrax deletes the drive's serial number from the striping configuration file. Additionally, we deactivate the SSD component pathways in the BIOS using an option called `PCIe Port Config` (Figure 7b).

Fan pathway. No explicit deactivation is needed for fans. They are monitored by the BMC which emits frequent error messages in case of faults (e.g., zero or low RPM). Hyrax changes BMC firmware to filter out fan error messages for deactivated fan slots.

5.2 Achieving High Performance on FIP Servers

We describe performance challenges when deactivating memory and SSD pathways and how Hyrax overcomes them.

Memory pathway. Cloud servers maximize achievable memory bandwidth by interleaving cachelines across DIMM ranks on all memory channels on the same socket. Deactivating a DIMM limits the processor's interleaving options and can significantly reduce VM memory bandwidth. Unfortunately, the resulting configuration is almost always outside CPU specifications, known as DIMM population rules [11, 31, 38].

To understand the performance impact of undocumented interleaving from deactivating DIMMs, we experiment with a common production server configuration. This server has two memory controllers per socket (MC0 and MC1), three memory channels per controller (A-C on MC0 and D-F on MC1), and two DIMMs per channel (e.g., A1, A2).

Figure 8 shows memory bandwidth for this server configuration measured in four scenarios: all DIMMs enabled, only DIMM A1 deactivated, only DIMM A2 deactivated, or DIMM A1 and A2 deactivated. We measure the memory bandwidth with a Memory Latency Checker (MLC) [30] for VMs ranging from 4-32 cores and show averages across 10 runs for each VM size. Error bars indicate the worst-performing run

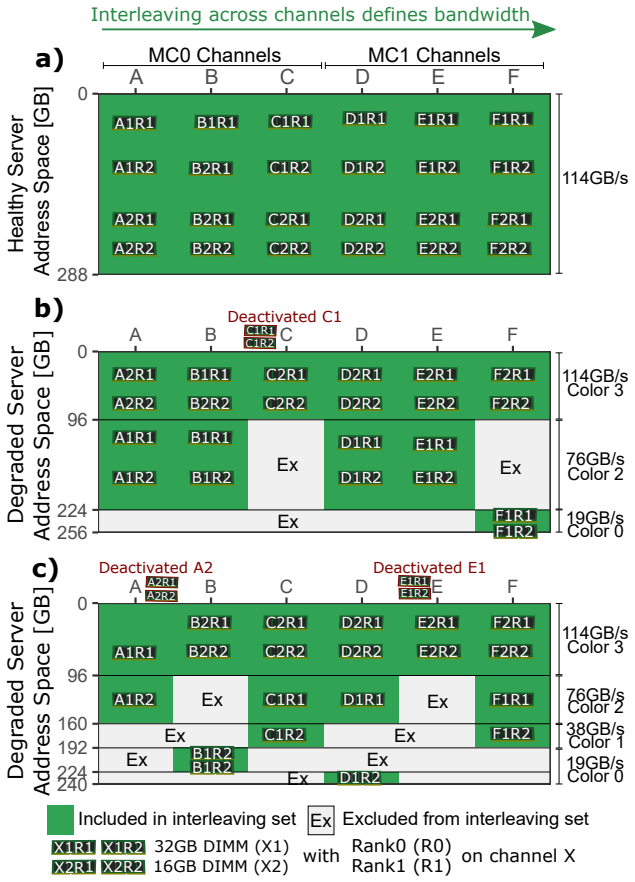


Figure 9: Channel interleaving with deactivated DIMMs. The top image shows interleaving for a healthy server, the middle image with DIMM C1 deactivated and the bottom image with two DIMMs (A2, E1) deactivated. Under degraded mode different regions of the address space experience different memory bandwidths, ranging from 19GB/s to 114GB/s.

for each VM size. We observe mean bandwidth loss between 0 to 36% depending on which and how many DIMMs are deactivated. Additionally, we observe that even for the same configuration, there is a significant variance between runs with worst case bandwidth loss up to 82%. Such outliers are not acceptable for deployment. We next explain the underlying reasons and then explain our mitigation.

We find that the inflexibility inherent in channel interleaving is the reason for the bandwidth loss. While a server can have multiple interleaving configurations for different ranks (called sets), each set must either alternate between MCs or just focus on a single MC. Consequently, cross-MC-interleaving requires the same capacity in participating channels on both MCs. To better understand the subtleties involved in interleaving we use a custom firmware debug mode that prints interleaving sets and participating channels. Figure 9 compares the interleaving we observe on healthy versus degraded servers for a single CPU socket on a common platform.

Figure 9a shows interleaving for a healthy server, which

contains a 32GB and a 16GB DIMM per channel[5]. For example, channel A on MC0 contains the 32GB DIMM A1 with ranks A1R1 and A1R2 and a 16GB DIMM A2. Cachelines are interleaved across all six 32GB DIMMs and across all six 16GB DIMMs. Interleaving across all channels creates a uniform address space with 114GB/s, i.e., a sixfold increase over a single channel (19GB/s).

Figure 9b shows a degraded server with C1 (32GB) deactivated. Since symmetry is required within an interleaving set, both C1 and F1 are removed from the first set and as a result, the server interleaves only across the four remaining 32 GB DIMMS. On the other hand, since all 6 16GB DIMMs are still active, the processor continues to interleave across all 6 DIMMs achieving the full 114 GB/s for their part of the address space (note that the 16GB DIMMs now make up the top part of the address space). As F1 is active, but not part of any set so far its capacity remains as non-interleaved (19GB/s). This creates a non-uniform address space with 38% of pages at 114GB/s, 50% at 76GB/s, and 12% at 19GB/s.

A degraded server with two deactivated DIMMs further complicates interleaving sets. Figure 9c shows the interleaving that results when A2 (16GB) and E1 (32GB) are deactivated. With deactivated DIMMs having different sizes, the resulting interleaving sets do not align with full DIMMs and instead use individual ranks (1/2 of a DIMM). The first interleaving set uses A1's first rank (A1R1) and five 16GB DIMMs (B2-F2) achieving the full 114GB/s. The second set uses A1's second rank (A1R2) and the first ranks from DIMMs C1, D1, F1 achieving 76GB/s. The third set uses the second rank from C1 and F1. Two final sets interleave across only a single channel using both ranks from B1 and D1's second rank. This results in an address space with 40% of pages at 114GB/s, 27% at 76GB/s, 13% at 38GB/s, and 20% at 19GB/s.

The main problem with varying peak bandwidth in different address ranges is that it makes VM memory performance on these servers unpredictable. As the OS and hypervisor are unaware of bandwidth differences across the address space the performance of a VM will vary depending on where in the address space its memory gets allocated. A naïve implementation of Hyrax would allocate VMs with a mix of pages leading to low-bandwidth outliers as shown in Figure 8.

To mitigate bandwidth variance on a degraded server, we must know the exact address map that maps address ranges to their achievable peak bandwidth. Unfortunately, reading the interleaving configuration usually requires debugging output that is typically not available. While we can test the memory bandwidth of the entire address space, we found this to be slow and inaccurate. Instead, we conceptually group different deactivation scenarios into equivalence classes, where scenarios in the same class result in the same address map, and store the resulting address map in a distributed database (§6). For example, deactivating a single DIMM leads to two equivalence classes depending on the DIMM size of the deactivated DIMM: The first class includes all scenarios where any one of the 32GB DIMMS fails (and the resulting map would be the image in Figure 9b) and the second class includes all scenarios where one of the 16GB DIMMs fails. Deactivating two DIMMs leads to ten equivalence classes, in addition to two DIMM sizes, interleaving changes with the two DIMMs being on the same channel, within the same MC, in a symmetric or asymmetric position on another MC.

Note that our discussion above focused on only one socket. Since interleaving on different (cache-coherent) CPU sockets happens independently, it is sufficient to characterize one socket. We validated equivalence classes by testing almost all 276 possible combinations. Deactivating three DIMMs leads to 2024 combinations and a multitude of equivalence classes — Hyrax thus deactivates at most two DIMMs and repairs three or more DIMM failures. A sensitivity analysis in § 7.4 will show that disabling larger numbers of DIMMs does also not provide significant gains in terms of repair savings.

Once we know the address map, we employ *page coloring* in the OS/hypervisor memory manager (MM) to assign the same color to pages that are in address regions with equal bandwidth. For example, in Figure 9 we assign colors 0, 1, 2, 3 to pages within a 19, 38, 76, 114 GB/s region, respectively. Each VM type comes with a preferred page color, which is set based on core count. Figure 8 shows that color 0 is sufficient for 2-core VMs. Color 1 is sufficient for 4-6 cores, color 2 for 8-12 cores, and color 3 for above 16 cores. Older generations of VMs sometimes run on new servers, while originally being created for servers with a lower per-channel bandwidth and four (instead of six) channels. Thus, old VM types do not even require color 3 and often use colors 0 and 1.

One could use this coloring scheme to guarantee performance at all times by exposing the amount of available memory for each color to the scheduler. However, to reduce coupling between control plane services, we do not expose this level of detail to the VM scheduler. So, large VMs may be allocated using colors below their bandwidth expectation if no higher colors are currently available on the server. Thus, Hyrax offers only a best-effort guarantee. Empirically, we find that this is sufficient since these cases are exceedingly rare (§7).

SSD pathway. The SSD pathway is simple compared to memory. In a fully healthy server, local VM storage is striped across six NVMe drives. VM types are capacity and rate limited (IOPS and bandwidth). When deactivating one NVMe drive, aggregate throughput remains sufficient for even the largest VM type. Deactivating two NVMe drives leads to sufficient throughput for all except the largest VM type. Hyrax thus never schedules this VM type on degraded servers with only four active NVMe drives. Hyrax never deactivates more

[5]The combination of 32GB and 16GB DIMM within one channel is a common configuration to reach target memory-to-core ratios in cloud compute servers of recent years. We discuss this configuration since our experiments with custom firmware happened to run on it. Interleaving on a 32GB/32GB server behaves similarly.

than two NVMe drives and this failure case is rare.

Fan pathway. Due to cooling overprovisioning, deactivating up to two fans leads to no performance loss.

6 Hyrax Control Plane

The Hyrax control plane consists of two new distributed services that implement the Hyrax policy and many changes to existing control plane services, including the VM scheduler.

6.1 Hyrax Policy

The Hyrax Policy has two roles (recall Figure 6). First, it interprets diagnostics and sets constraints on which components are degradable. Second, it ensures that degraded servers meet C_{Capacity}.

To perform the first role, the Hyrax Policy specifies for each server type how many component pathways of each type can be deactivated at once. While Hyrax can adapt to a wide range of thresholds, for our purposes we use two DIMMs, two SSDs, and two fans. These thresholds are guided by common failure scenarios (Section 3) and performance observations (Section 5). Hyrax schedules repairs for any server with more than two faulty component pathways of the same type or any other failure diagnosis. The Policy also includes an extensive mapping list of diagnostic results to valid component pathways. For example, SSD pathways can appear as IO errors, timeouts, and PCIe errors. Diagnostics for SSD failures can sometimes point to PCIe ports and slots that have different (non-SSD) devices or even no device — for these the Hyrax Policy would just schedule the server for repair.

To ensure $C_{\text{Performance}}$, the Hyrax Policy maps every degraded server configuration to a capacity and performance profile (CPP). The CPP defines the exact server capacity and performance equivalence class (§5). Based on the CPP, Hyrax defines the set of allowable VM types that can run on a degraded server and still meet their SLAs. For example, servers with two DIMMs deactivated on the same channel do not have any page of color 3. This server thus cannot host latest-generation VMs with more than 16 cores. A server with two deactivated SSDs cannot host the largest VM type.

6.2 Control Plane

Deactivating component pathways leads to heterogeneous server configurations within a cluster. This requires changes across service and team boundaries. Figure 10 shows a simplified view of Azure's control plane. We change three and add two new control plane systems.

Let's consider a server that starts in healthy state and encounters an SSD failure. (1) The Server Health Monitor (SHM) detects NVMe read errors and follows the offlining workflow (§2). (2) Diagnostics reports the SSD component

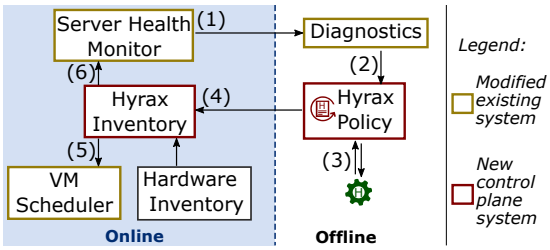


Figure 10: Simplified overview of Hyrax's control plane.

pathway to the Hyrax Policy. (3) The policy decides to start the deactivation workflow and communicates with a server-local daemon to deactivate that SSD. The deactivated SSD's serial number is also passed to the new Hyrax Inventory system. (4) After deactivation, the server is tested in the certification step. In the rare event that diagnostics leads Hyrax to deactivate the wrong pathway (§3), it would be detected in this step, e.g., during load testing. After passing certification, the server is onlined. As multiple control plane services might cache the server's capacity, onlining requires Hyrax to invalidate caches throughout the control plane including the VM scheduler. (5) The Hyrax Inventory shares the server's capacity and performance profile (CPP) with the VM scheduler (§6.3). Internally, our inventory tracks server state as a delta to the existing Datacenter Inventory. The delta consists of the serial numbers and slots of deactivated components, which remains small enough to fit into a single inventory server's memory. (6) The Hyrax Inventory sends active serial numbers and slots to the SHM. The SHM only checks for these active components, which prevents the SHM from triggering warnings over missing components which have been deactivated.

There are additional changes in downstream services not shown here. For example, it was previously uncommon for servers to have multiple concurrent failures, so repair tickets used to be issued only for a single component type. With Hyrax, it is common for repair tickets to include multiple different component types. For example, there are no tickets for a server with two DIMM failures. However, if the two DIMM failures are later followed by any failures for an undegradable component (e.g., the NIC) the repair ticket will involve two different component types. To minimize repair tickets, Hyrax changed the ticket workflow and retrained technicians to repair multiple different component types at once, with a single ticket (mini-batching).

6.3 VM scheduling policy

Hyrax requires three changes to VM scheduling and an optional optimization. First, the VM scheduler consumes Hyrax Inventory to calculate hardware resources for individual servers instead of a single lookup to obtain a cluster's homogeneous server type. The overhead of this lookup is negligible as servers moving from offline to online state is

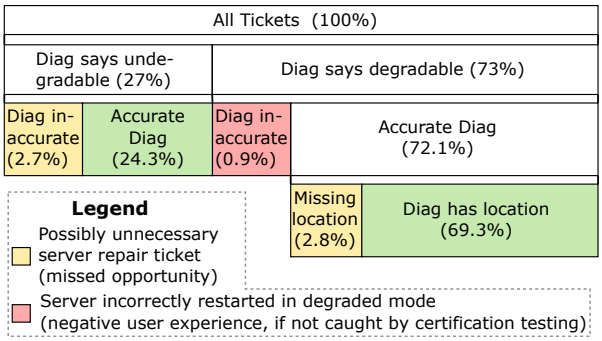


Figure 11: Accuracy of automated fault diagnostics at Azure and their impact on Hyrax.

rare compared to VM scheduling events.

Second, we extend filter rules (§2) to enforce Hyrax's CPP, i.e., which VM types can be placed on every server.

Third, we change the definition of a cluster's *"capacity reserve"*. The capacity reserve exists for multiple reasons, including to have a target to migrate VMs to when a server shows signs of failing soon. A key component of the capacity reserve is to have some healthy empty servers (HES) that are able to host any kind of VM, including full-server VMs that use all of a server's capacity. Degraded servers are not able to host all full-server VMs. We thus exclude them from being counted as HES.

Finally, we change a preference rule to optimize scheduling. Since degraded servers cannot be counted as HES, we prefer fully-healthy servers to become empty and stay empty. Our change updates rules to prefer placing VMs on degraded servers over healthy servers, provided no other rule takes precedence. By doing this we increase HES counts which allows placing more VMs into clusters.

6.4 Hyrax Diagnostics

Hyrax builds on an existing automated monitoring and diagnostics system. This system's output is targeted at humans and includes information on which component type is faulty and its location. To use this system, we add an interpreter that maps diagnostic results to valid Hyrax component pathways. As part of this design, we analyzed four years of repair ticket logs at Azure. This analysis shows the *accuracy* of the diagnostic system and how Hyrax handles inaccurate or incomplete diagnoses. Specifically, we rely on notes from human technicians, who worked on the tickets in our history of repair logs. These notes indicate whether the diagnosis was correct, including whether the right component was identified.

At a high level, we find that diagnostic accuracy is high. For example, across all tickets in 2021, 96.4% accurately identify the component type at fault. For a more detailed view, Figure 11 shows a breakdown of all diagnoses made in 2021, outlining the different scenarios that arise and how they impact Hyrax's operation.

We make two interesting observations: First, diagnostic accuracy is lower for diagnoses pointing to an *undegradable* component: 10% of tickets labelled with an undegradable component are inaccurate (accounting for 2.7% of *all* tickets). Fortunately, this type of misdiagnosis is relatively benign. Hyrax will take the server offline (for potential later repair), which is the intended behavior if the actual faulty component is indeed undegradable. It is however a *missed opportunity* to keep the server running in degraded mode if the true fault is in a degradable component.

Second, diagnostic accuracy is very high for diagnoses pointing to a degradable component: 98.8% of tickets labelled as degradable do accurately identify the component type at fault. Within these, some diagnoses are *incomplete*, where the correct component type is specified, but location information is missing (e.g. the diagnosis indicates a DIMM problem, but does not specify a DIMM slot). More precisely, 3.8% of the accurately diagnosed degradable tickets (corresponding to 2.8% of *all* tickets) are missing location information which leads Hyrax to offline the server despite the fact that the faulty component is degradable. These tickets thus also represent a *missed opportunity* for degraded mode operation.

The last scenario we need to consider is the 1.2% of degradable tickets that contain an inaccurate diagnosis pointing to the wrong component type. These make up only 0.9% of all tickets, but their impact on Hyrax is less obvious. In the best case, Hyrax will try to deactivate the specified pathway and certification testing (recall §4) fails since this is not the faulty component. Failing certification testing with any degraded component automatically triggers an investigation both by a technician and by the Hyrax on-call team. In the worst case, the server passes certification testing and returns to serve customer VMs despite the fact that the true faulty component has not been degraded or repaired. This can lead to negative user experience as VMs may be scheduled on the server and they may get interrupted if the server is offlined again. Such repeat offlining of the same server also happens for technician repairs. In fact, our preliminary data indicates that the rate at which repaired servers are offlined again is comparable to such inaccurate decisions by Hyrax. This is likely because technicians rely on the same automated diagnostics and certification process as Hyrax.

We conclude by noting that diagnostic accuracy has continuously improved over the past years. Figure 12 shows the breakdown of repair tickets for three different hardware generations (Gen 2-4) by year since deployment. We observe that accuracy has improved from generation to generation, and also that accuracy improves over time within a particular hardware generation. Both the fraction of tickets with missing location and tickets with inaccurate fault code have decreased over the years. The reason is a concerted effort by the diagnostic team at Azure to add more coverage of various fault codes as well as improvements based on technician feedback.

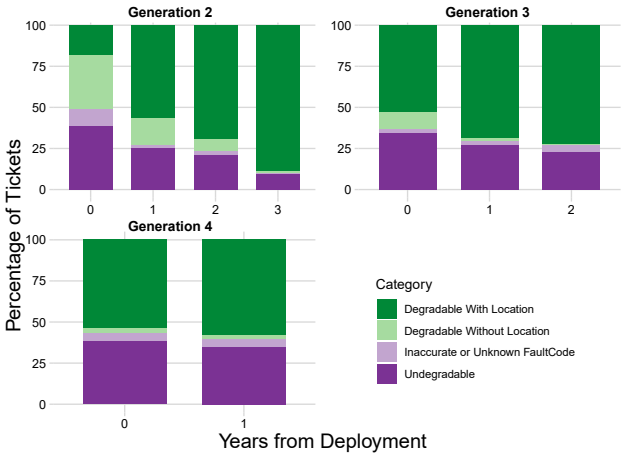


Figure 12: The progression of the breakdown of repair tickets at Azure by deployment year for three generations of servers.

7 Evaluation

7.1 Evaluation Setup

We use two types of setups in our evaluation of Hyrax. First, we evaluate Hyrax on production servers to characterize its performance and ability to mitigate faulty components. Our evaluation focuses on 3rd-generation servers which have been deployed for 2-3 years. Second, to measure cluster-level impacts on repairs and VM scheduling over six years, we use trace-driven large-scale cluster-level simulations.

7.1.1 Server experiments

We use production server hardware and *synthetically inject failures* using a commercial memory error injector (MEI) that interposes on the DDR memory bus [29]. We also perform *real failure* tests by intercepting nodes after Diagnostics flags a memory fault, but before a repair ticket is issued (§4).

We measure latency and bandwidth with Intel MLC [30] from inside VMs on healthy and degraded servers. MLC characterizes worst-case performance as it is more sensitive to deteriorated latency and bandwidth than any real-world application we've tested. We compare three implementations.

- **Hyrax:** Coloring approach based on 1GB hypervisor page table entries (§4)
- **Naïve:** Hypervisor randomly allocates VM memory among free pages
- **Interleaving:** 4kB-interleaving in hypervisor page tables

Our tests cover Intel servers from generations 3-5 and a subsequent (not yet deployed) generation. We report measurements from the 3rd generation as results from other generations are qualitatively the same. A typical 3rd-generation server uses two Intel Skylake processors (96 threads total). Each socket is equipped with six DDR4 channels with a 32GB and a 16GB DIMM per channel. Memory interleaving is enabled across all ranks on the same socket; thus, the OS/hypervisor sees two NUMA nodes. There are six data SSDs using 960GB NVMe drives. The server runs Azure's production-grade hypervisor and software stack. VMs are allocated with a 1GB page size.

7.1.2 Large-scale simulations

We replay VM, failure and repair ticket traces in a simulated environment, using the Azure production VM scheduler code base. The traces span 66 clusters that host general-purpose VMs from regions in the US and Europe. With only 2-3 years of real failure traces for 3rd-generation servers, we model future failures with the help of 1st and 2nd-generation failure traces. The simulator models Hyrax's control plane components (Figure 10) including Hyrax and all server states (Figure 6).

We compare two designs.

- **Hyrax:** Hyrax enables degraded server states and repairs servers with undegradable components and above thresholds (§4).
- **AoN:** All-or-Nothing repairs all hardware faults.

We simulate four possible repair schedules: issuing an immediate repair ticket (**IR**) and scheduling batch repairs every 3, 6, or 12 months (**3m**, **6m**, **12m**). For IR, we sample actual repair delays from Azure production datacenters. For batch repairs, we assume a hypothetical schedule where repairs are immediately effective at 3, 6, or 12 months. This batch repair schedule is unlikely how batch repairs would actually be implemented in practice. Instead, its purpose is to show a hypothetical and simplified schedule that could also reduce repair work, to highlight the impact of degraded mode operation. For each repair schedule, we compare the number of repair tickets, repair trips, resource availability and impact on arriving VMs under Hyrax and AoN.

We cross-validate the simulator for AoN relative to real-world clusters with the same failures and VM workloads. Due to the inherent randomness in placement decisions, repeated runs have small deviations. Across runs on 10 clusters, simulation of AoN and real-world metrics are within 0.25%. Overall, our simulations required more than 80,000 CPU hours.

7.2 Correctness

In this section, we use production server measurements to demonstrate that Hyrax can correctly deactivate component paths and thereby avoid future faults on a path. Due to space constraints, we focus on memory faults and omit qualitatively-similar SSD experiments.

Synthetic failures. We measure memory error rates with the MEI placed on a given DIMM slot and either activate all ranks (no-Hyrax) or deactivate the corresponding slot (Hyrax). We target the MEI to corrupt bits matching a single row address and start a VM on the same CPU socket. The VM runs MLC in the peak bandwidth setting. Under no-Hyrax, we observe

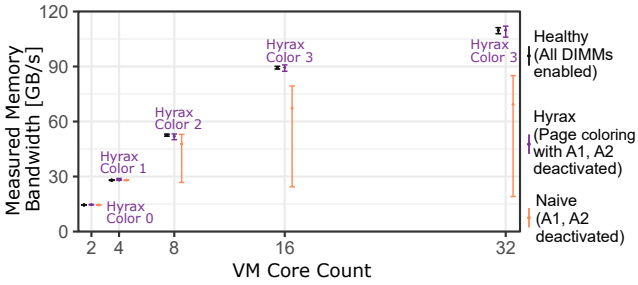


Figure 13: Peak memory bandwidth of a healthy server, a Hyrax server with page coloring, and a naïve implementation of degraded servers with two DIMMs deactivated.

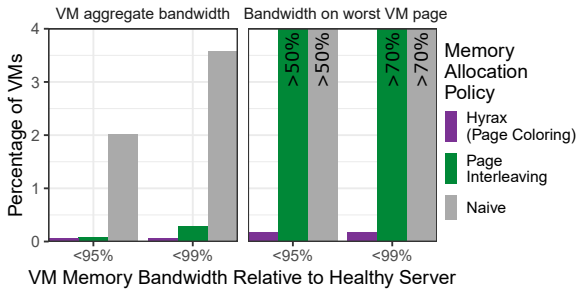


Figure 14: Hyrax almost always achieves the same VM memory bandwidth on degraded nodes as VMs would on healthy servers.

a high rate of correctable memory errors. There are bursts of uncorrectable errors that lead to both VM and host crashing within minutes. With Hyrax, there are no memory errors throughout the duration of a 48 hour test; the VMs and the host run without errors or crashes.

Real-world failures. We identify a server in a test cluster that was diagnosed with a high rate of uncorrectable memory errors on one DIMM. Diagnostics is able to boot its minimal OS and reproduce these memory errors. Hyrax recognizes that this server can be degraded and deactivates the correct DIMM. Certification testing does not find any memory errors and issues a "pass" that qualifies this server for hosting VMs.

7.3 Performance

In this section, we demonstrate that Hyrax can successfully mitigate any VM performance impact of degraded mode operation. For space reasons, we focus on the more complex case of memory performance (memory latency and bandwidth).

Server-level experiments. Figure 13 compares VM memory bandwidth of Hyrax and Naïve on a degraded server to a healthy server. The degraded server has A1 and A2 deactivated. Hyrax allocates the VM using colors 0-3, depending on VM core count (§4). We find that memory bandwidth under Hyrax is within 1% of the healthy server. In contrast, Naïve's performance is highly variable with mean bandwidth up to 36% lower and worst-case bandwidth up to 82% lower than on the healthy server.

We also tested memory latency. In all three systems, and across all experiments, the unloaded memory latency reported by MLC for the degraded server remains within 5% of the healthy server.

Large-scale page coloring simulations. The previous experiment focused on a single VM in isolation for one particular failure pattern. For a more complete view of VM performance under Hyrax we use simulations that are driven by actual traces of VM arrivals and departures to capture the effect of VM churn and also simulate component deactivation based on real failure traces to capture the rich set of failure patterns that arises in practice. (The traces come from our cluster simulations in §7.4). We play back these VM events in server-level simulations of the three memory allocation policies: Hyrax page coloring, page interleaving, and Naïve.

Figure 14 shows the percentage of VMs with less than 95% and 99% of the bandwidth of a healthy server, both for VM aggregate bandwidth (left) and bandwidth of the VM's *worst* page (right). With Hyrax, fewer than 0.16% of VMs see bandwidth on their worst page that is lower than 99% of the worst-page bandwidth achieved on a healthy server. VM *aggregate* bandwidth under Hyrax is even closer to that of a healthy server.

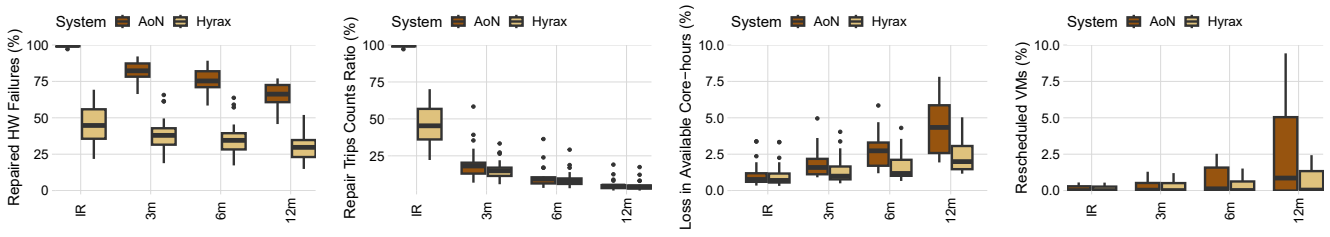
Page interleaving also results in a low percentage of VMs that achieve less than 95-99% of the aggregate memory bandwidth of a healthy server. However, more than half of VMs include at least one memory page with significantly lower bandwidth. We also note that page interleaving increases a VM's page table by orders of magnitude. This leads to a high rate of TLB misses and increased memory access latency. In practice, we know that memory access latency is even more important than bandwidth — internal production workloads lose 5-15% of performance for small page sizes. Thus, interleaving is not practical.

Naïve is compatible with large page sizes but more than 2% of VMs achieve less than 95% of the aggregate bandwidth goal. This grows to 3.5% for a goal of 99% and above 50% when considering the worst page in a VM. While Naïve performs well on average, tail performance matters at scale.

7.4 Large-scale Cluster Simulations

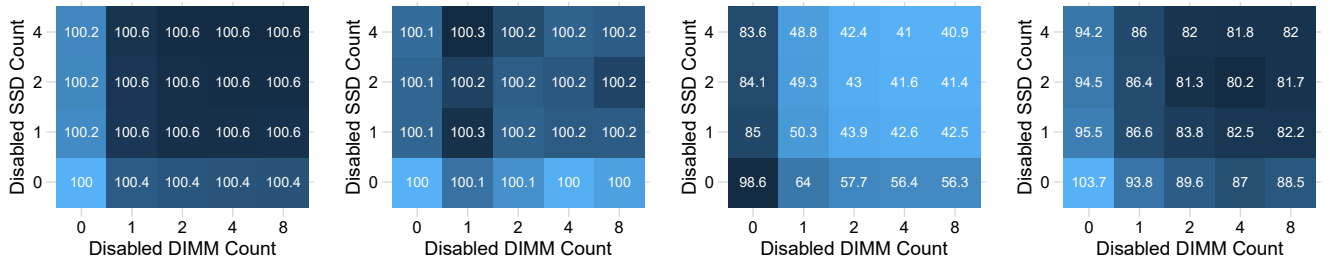
We turn to large-scale cluster simulations to characterize Hyrax's impact on repair tickets, repair trips, cluster resource availability and user impact. We consider four different repair modes: Azure's process of immediately scheduling a repair ticket (IR) and hypothetical repair batching policies with three different intervals (3 months, 6 months, 12 months).

Repair tickets. We begin by measuring for each repair mode the percentage of all hardware failures that result in a repair ticket, i.e. the failures that require a technician to perform



(a) Repair tickets. **(b)** Repair trips. **(c)** Lost available core-hours. **(d)** Rerouted VMs.

Figure 15: Results from a simulated deployment of Hyrax across two regions with 66 compute clusters and four repair schedules: immediate repair tickets (IR) and batch repairs (3m, 6m, 12m). The figures compare key metrics under Hyrax with all-or-nothing server operation (AoN).



(a) Available core-hours. **(b)** Available memory-hours. **(c)** Repair Tickets. **(d)** Rerouted VMs.

Figure 16: Ratio between Hyrax and AoN for different threshold settings with fixed batch repair interval (3m). Stranded availability is not included when computing core-hours and memory-hours availability.

physical examinations and repairs. Figure 15a shows the results for Hyrax and AoN using boxplots, where each data point in the distribution represented by the boxplot corresponds to one of the 66 clusters.

We observe that Hyrax reduces the number of repair tickets by more than a factor of 2 across all repair modes. Both mean and median are consistently around 55% lower under Hyrax than under AoN. A significant contributor to Hyrax's effectiveness is mini-batching. Specifically, under Hyrax, we find that 56% of repair tickets contain more than one component, compared to single-digit fractions for AoN.

Repair trips. We compare the number of repair trips required under Hyrax and AoN, i.e. the number of times when a technician needs to travel to a cluster. Figure 15b shows the number of repair trips normalized by the number of hardware failures.

We observe that Hyrax significantly reduces repair trips under immediate repairs (IR). While under AoN every hardware failure results in a repair trip, under Hyrax, on average 55% of these repair trips can be avoided by deactivating the affected component.

Under batch repairs, the number of repair trips to a cluster is upper-bounded to once every x months, where x is the repair interval. Interestingly, Hyrax still provides improvements over AoN, albeit smaller than for IR. For example, for a batch repair interval of 3 months Hyrax reduces repair trips by around 20% (mean and median across clusters). Every saved repair trip results from a 3 month interval in which Hyrax was able to handle all failures with component deactivation.

Lost available core-hours. This metric quantifies the impact of the repair operating model on the availability of cluster hardware resources. In particular, we consider the percentage of a cluster's total core-hours (i.e. number of cores in the cluster multiplied by cluster lifetime) that are *lost*, i.e., a core physically exists in the cluster, but is not available to run VMs due to one of two reasons: (1) A server is offline for repairs; (2) Due to resource fragmentation some of a server's cores cannot be allocated to VMs because of limited availability of another resource (DIMMs or SSDs) [40]. Hyrax might exacerbate resource fragmentation as it might deactivate multiple components of one type, making it harder to utilize the remaining components.

Figure 15c shows that under a batch repair schedule Hyrax significantly reduces the loss of available core-hours. Hyrax keeps servers running (albeit with reduced capacity) after degradable component failures, rather than taking the entire server offline until the next scheduled batch repair. The improvement in the median lost core-hours of Hyrax over AoN ranges from 38% for a 3m interval to 55% for a 12m interval.

Interestingly, we observe that Hyrax improves loss in available core-hours even in the IR repair schedule. The median loss in core-hours is 9% lower under Hyrax than AoN. The reason is that immediate repairs are not truly immediate - typical repair times are on the order of days, but can sometimes take much longer, depending on component availability. In

contrast, deactivating components is consistently fast.

Rerouting of VMs. When a cluster's available resources are insufficient to host an arriving VM, the VM is rerouted to a different cluster. Rerouting of VMs can negatively impact user experience as it increases the time until a VM gets started. Figure 15d shows the percentage of arriving VMs that are being rerouted under Hyrax versus AoN.

Under IR, the fraction of VMs that get rerouted is very small for both Hyrax and AoN. While it is identical (zero) in the median for both policies, the mean is slightly lower (4% reduction) under Hyrax as it decreases lost core hours in some clusters with many failures.

When moving to batch repair schedules, Hyrax provides clear improvements over AoN, ranging from an average 38% reduction in rerouted VMs for 3 month batch repairs to an average 64% reduction for 12 month batch repairs. These improvements are a direct consequence of the reduced loss in system capacity (core-hours) under Hyrax compared to AoN.

TCO impacts. Server repairs account for 9 to 12% of TCO (§2), and Hyrax reduces repair tickets by an average of 55% across the simulated clusters (§7.4). However, repairs frequently involve multiple components as well as undiagnosed failures, which extends repair times by about 15%. Thus, Hyrax reduces technician time by about 48%, which translates to a 4.5 to 6% reduction in TCO.

Sensitivity to Hyrax's deactivation thresholds. Our implementation of Hyrax chooses its deactivation threshold of two per component type to reduce complexity. Figure 16 shows how different choices of thresholds impact available core hours, available memory hours, repair tickets and rerouted VMs. The numbers in the figure represent the ratio of Hyrax to AoN for a batch interval of 3m. Darker color shading corresponds to better results.

We observe that available system capacity (core-hours as well as memory hours) does not improve/change significantly beyond a threshold of one DIMM and one SSD. The reduction in number of repair tickets and rerouted VMs under Hyrax continues to increase as thresholds increase, however, returns are diminishing past a threshold of two DIMMs and two SSDs. One of the reasons is that it is very rare that more than two DIMMs and/or more than two SSDs fail in the same server, so these scenarios have little impact on key metrics.

In conclusion, increasing the thresholds beyond two per component type provides very limited gains while increasing system complexity, e.g. in handling servers with very low performance due to a large number of degraded components.

Sensitivity to different regions. Figure 1 shows that Hyrax performs similarly across the US and EU region.

Sensitivity to server generation. We also simulated a full deployment of Hyrax on 4th-generation servers. Figure 12 shows that this generation has an overall lower percentage of degradable components. Hyrax's benefits are thus slightly less pronounced on this server generation. However, as diagnostics has improved over time for 2nd and 3rd-generation servers, 4th-generation servers may also improve in the future.

8 Related Work

Our work is the first work to explore degraded mode operation in the context of VM compute servers and at the scale of a cloud platform.

Datacenters that fail-in-place. Related to our work are the general efforts toward lights-out data centers such as containerized datacenters [23, 65], underwater datacenters [10], and zero-maintenance storage systems [49, 50]. In our evaluation, AoN with high batch repair intervals (12m) represents these approaches. Unfortunately, the loss in availability or cost (hardware, power, space) to make up for this loss is prohibitive without degraded mode.

Mechanisms to implement fail-in-place. We borrowed the term degraded mode from RAID systems [51], where upon failure of a drive, the system seamlessly continues to operate until the failed drive is replaced, however at *reduced capacity and reduced performance*.

There are many existing fault-tolerance approaches that use component-internal redundancy [8, 25, 34, 44, 52, 55, 67, 68]. Hyrax targets the left-over failures not already covered by these approaches. It can be viewed as taking degraded mode to the extreme and applied to even combinations across different devices. As such, Hyrax has different requirements that raises novel challenges (§4).

Improving repairs and redundancy. Recent efforts for reducing the reliance on human technicians in lights-out datacenters explore the use of robots to replace hardware components [56]. Currently, this technology is not sufficiently capable, versatile and economical to be employed at scale. Our work presents a solution that can be deployed immediately in today's systems.

Finally, systems that require no or minimal repairs throughout their lifetime are common in the context of embedded systems, for example, as part of autonomous vehicles, airplanes or satellites [6, 14, 47, 73]. However, these are special purpose systems with specialized components and significant redundancy. In contrast, we are exploring whether a cluster based on commodity data center components can operate with no or minimal repair throughout its lifetime through the use of fail-in-place.

9 Deployment Experience and Discussion

Hyrax reduces repair tickets while maintaining cluster capacity, VM scheduling, and VM performance. We discuss deployment experience and broader issues.

Deploying incrementally. Hyrax requires changes across teams that have not previously interfaced, including hypervi-

sor software engineers, hardware validation teams, and datacenter staffing. Such a large project requires years to achieve visibility and alignment. During this process, we developed variants of Hyrax that could be deployed incrementally and fly largely under the radar. Our first increment focused on deactivating a single SSD in clusters without impacts on VM scheduling due to spare capacity and bandwidth. Further, we shortcut offline state changes as we could pinpoint some SSD failures without diagnostics. Our shortcut quickly migrated VMs away, rebooted the server, and deactivated the faulty SSD without leaving the online server state. Building in increments increased visibility and buy-in across Azure which facilitated far-reaching changes to VM scheduling and offline server workflows. Overall, Hyrax demonstrates the feasibility of overcoming ossification in large software stacks.

Reduced benefits due to non-optimized software paths. The server state diagram in Figure 6 and the control plane overview in Figure 10 are vastly simplified. In principle, a degradable server should be able to return online within half an hour (after a reboot). However, before Hyrax, repairs took multiple days and sometimes even weeks, e.g., due to supply chain issues. Thus, the duration of offline states and transitions did not matter. Under Hyrax, returning to online has to wait for these states, which takes multiple hours in production.

An early variant shortcuts the offline state and returned servers to online within minutes. Unfortunately, the deployment scale of this variant is limited as few faults can be recognized as degradable without deep diagnosis. The limited scale of the shortcut variant and the slowness of Hyrax's offline implementation currently limits Hyrax's ability to improve cluster capacity. This is reflected in our simulations (§7).

The usefulness of simulations and quantitative data. We tested significant parts of the production code for inventory and state management in a mocked-up environment driven by simulated failures. Our large-scale simulations also helped convince engineering teams to help with large-scale changes. For example, we initially faced significant skepticism towards mini-batching. This was partly due to multiple past efforts that had tried and failed to implement mini-batching. These past efforts had cemented the idea that multiple components failing at once is a very rare occurrence. Simulations showed that Hyrax led to a high occurrence of mini-batched tickets.

Tailoring automated diagnostics for FIP. While our work shows that FIP can work with existing diagnostics systems, there is still room for improvement, including fine-grained diagnostics to find individual faulty cores and to improve locating other component paths (§6.4). We also find subtle shortcomings in diagnostics systems due to their focus on technician repairs. For example, current diagnostics systems prefer not to issue a ticket when they cannot reproduce a failure and pinpoint a specific repair action. This is required due to the high cost of false positives, i.e., calling a technician and replacing a component when the underlying component was not actually faulty. The flip side is a higher rate of false negatives, which we observe as repeated failures on the same server. Hyrax's automation may open up a path towards improving cloud reliability and availability. Specifically, a FIP system could tolerate a higher rate of false positives (as they lead to a negligible capacity impact), and in exchange achieve lower false negative rates.

Interaction with class failures. An early practical concern at Azure was how Hyrax interacts with the occurrence of class failures, which is a recall of a large set of components of similar types from the same manufacturing period. Over three years, we found class failures affecting multiple PSU, DIMM, and CPU models, and one SSD model. Class failures often lead to an expectation of increased failure rates which may affect availability. Thus, associated components are typically proactively swapped out for new components. While class failures cause only about 5% of repair tickets, they often affect a large percentage of servers in the same cluster at once. If the number of affected components in a server is below Hyrax's thresholds, degraded mode can be an effective mitigation. However, deactivating many components at once may negatively affect VM scheduling. Thus, when we look back at three years of class failures, Hyrax would have only been effective in mitigating one out of about a dozen of class failures.

Implications for new datacenter environments. Our findings affect how one might design a future datacenter. In short, Hyrax reduces repair needs but does not obviate the need for repairs entirely. Specifically, the capacity loss after 6 to 10 years of deployment without repairs exceeds the cost savings of most new datacenter designs. We thus expect to see continued need for individual component replacement.

Hyrax's reduction in the number of repairs may be sufficient to offset the additional repair time introduced by some designs, such as new cooling techniques [33, 72]. Specifically, we find that Hyrax enables datacenter designs that result in repairs that take about twice as long. When repair times take much longer, TCO will increase even with Hyrax. This might be the case for some server and datacenter designs including extremely dense servers [15–18, 21, 27, 28, 36], connector-less server designs with soldered-on components [45], or datacenters in hard-to-reach locations, such as sealed containers on the ocean floor [10].

Acknowledgments

We thank our shepherd, Daniel Peek, and the anonymous OSDI '23 reviewers for their great comments. We thank our many partner teams within Microsoft including Dirk Hofmann, Saptadeep Chanda, and Tom Harpel for their continued support on understanding technician training, workflows, and staffing; Rama Bhimanadhuni for his help on firmware; and Manish Dalal for early feedback on interpreting diagnostics.

References

[1] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing slos for resource-harvesting vms in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 735–751, 2020. 2

[2] Nadav Amit, Muli Ben-Yehuda, IBM Research, Dan Tsafrir, and Assaf Schuster. viommu: Efficient iommu emulation. In *2011 USENIX Annual Technical Conference (USENIX ATC 11)*. 2

[3] Backblaze. Hard drive data and stats. https://www.backblaze.com/b2/hard-drive-test-data.html accessed 6/26/2022, June 2022. 1

[4] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013. 1, 1, 2, 2

[5] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003. 1

[6] John W Bennett, Glynn J Atkinson, Barrie C Mecrow, and David J Atkinson. Fault-tolerant design considerations and control strategies for aerospace drives. *IEEE Transactions on Industrial Electronics*, 59(5):2049–2058, 2011. 8

[7] Daniel S. Berger, Fiodar Kazhamiaka, Esha Choukse, Íñigo Goiri, Celine Irvene, Pulkit A. Misra, Alok Kumbhare, Rodrigo Fonseca, and Ricardo Bianchini. Research avenues towards net-zero cloud platforms. Workshop on NetZero Carbon Computing, 2 2023. 1, 2, 2

[8] Stuart Allen Berke and Vadhiraj Sankaranarayanan. System and method for post-package repair across dram banks and bank groups, August 2019. US Patent 10,395,750. 2, 5.1, 8

[9] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57, 2016. 2

[10] Ben Cutler, Spencer Fowers, Eric Peterson, and Mike Shepperd. Project natick. OpenCompute OCPREG19 track on Rack & Power / Advanced Cooling https://natick.research.microsoft.com/ accessed 6/26/2022, October 2020. 8, 9

[11] Dell. Memory population rules for 3rd generation intel xeon scalable processors on poweredge servers. https://www.delltechnologies.com/asset/en-us/products/servers/industry-market/whitepaper-memory-population-rules-for-3rd-generation-intel-xeon-scalable-processors-on-poweredge-servers.pdf accessed 11/26/2022, 2022. 5.2

[12] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 610–621, 2014. 1

[13] Xiaoming Du and Cong Li. Combining error statistics with failure prediction in memory page offlining. In *International Symposium on Memory Systems*, pages 127–132, 2019. 5.1

[14] Elena Dubrova. *Fault-tolerant design*. Springer, 2013. 8

[15] E3NV. Ots immersion servers. https://www.e3nv.com/immersion-servers accessed 6/26/2022, 2022. 9

[16] Wesley M Felter, Tom W Keller, Michael D Kistler, Charles Lefurgy, Karthick Rajamani, Ramakrishnan Rajamony, Freeman L Rawson, Bruce A Smith, and Eric Van Hensbergen. On the performance and use of dense servers. *IBM Journal of Research and Development*, 47(5.6):671–688, 2003. 9

[17] Gigabyte. Coolit liquid-cooled ready servers. https://www.gigabyte.com/Industry-Solutions/coolit-liquid-cooled-ready-servers accessed 6/26/2022, 2022. 9

[18] GRC. Servers designed for immersion (sdi). https://www.grcooling.com/servers-for-immersion-cooling/ accessed 6/26/2022, 2022. 9

[19] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks, 2008. 2

[20] Albert Greenberg and Dave Maltz. What goes into a data center. SIGMETRICS 2009 Tutorial, 2009. 2

[21] Anthony Gutierrez, Michael Cieslak, Bharan Giridhar, Ronald G Dreslinski, Luis Ceze, and Trevor Mudge. Integrated 3d-stacked server designs for increasing physical density of key-value stores. In *ACM ASPLOS*, pages 485–498, 2014. 9

[22] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, David Dion, Esaias E Greeff, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean:vm allocation service at scale. In *USENIX OSDI*, pages 845–861, 2020. 2

[23] James R. Hamilton. An architecture for modular data centers. In *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 306–313. www.cidrdb.org, 2007. 8

[24] Peter H Hochschild, Paul Turner, Jeffrey C Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E Culler, and Amin Vahdat. Cores that don't count. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 9–16, 2021. 1, 2

[25] Duwon Hong, Myungsuk Kim, Geonhee Cho, Dusol Lee, and Jihong Kim. Guardederase: Extending ssd lifetimes by protecting weak wordlines. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 133–146, 2022. 2, 8

[26] Amy Hood. Microsoft earnings release fy22 q4. https://www.microsoft.com/en-us/Investor/earnings/FY-2022-Q4/press-release-webcast accessed 11/26/2022, 2022. 2

[27] Hypertec. Trident immersion servers. https://hypertec.com/ciara/immersion-servers/ accessed 6/26/2022, 2022. 9

[28] AVNET Integrated. Integrated rack with immersed, liquid-cooled it. https://www.avnet.com/wps/portal/integrated/resources/liquid-cooling/ accessed 6/26/2022, 2022. 9

[29] Intel. Memory error injection mei test card and utility. https://designintools.intel.com/MEI_Test_Card_and_Utility_p/stlgrn61.htm accessed 6/26/2022, 2017. 7.1.1

[30] Intel. Memory latency checker v3.9a. https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html accessed 6/26/2022, 2022. 5.2, 7.1.1

[31] Intel. Supported memory and memory population rules for the intel server board family. https://www.intel.com/content/www/us/en/support/articles/000055509/server-products/server-boards.html accessed 11/26/2022, 2022. 5.2

[32] Michael Isard. Autopilot: automatic data center management. *ACM SIGOPS Operating Systems Review*, 41(2):60–67, 2007. 1, 2, 2

[33] Majid Jalili, Ioannis Manousakis, Íñigo Goiri, Pulkit A Misra, Ashish Raniwala, Husam Alissa, Bharath Ramakrishnan, Phillip Tuma, Christian Belady, Marcus Fontoura, et al. Cost-efficient overclocking in immersion-cooled datacenters. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 623–636, 2021. 1, 9

[34] Dae-Hyun Kim and Linda S Milor. Ecc-aspirin: An ecc-assisted post-package repair scheme for aging errors in drams. In *IEEE VLSI Test Symposium*, pages 1–6, 2016. 2, 8

[35] Andi Kleen. Mcelog bad page offlining. http://www.mcelog.org/badpageofflining.html, 2021. 5.1

[36] Ravi Kollipara, Ming Li, Chuck Yuan, Hideki Kusamitsu, and Toshiyasu Ito. Evaluation of high density liquid crystal polymer based flex interconnect for supporting greater than 1 tb/s of memory bandwidth. In *2008 58th Electronic Components and Technology Conference*, pages 1132–1138, 2008. 9

[37] Alok Kumbhare, Reza Azimi, Ioannis Manousakis, Anand Bonde, Felipe Vieira Frujeri, Nithish Mahalingam, Pulkit Misra, Seyyed Ahmad Javadi, Bianca Schroeder, Marcus Fontoura, and Ricardo Bianchini. Prediction-based power oversubscription in cloud platforms. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 473–487, 2021. 2

[38] Lenovo. Balanced memory configurations with second-generation intel xeon scalable processors. https://lenovopress.lenovo.com/lp1089.pdf accessed 11/26/2022, 2022. 5.2

[39] Ilya Lesokhin, Haggai Eran, Shachar Raindel, Guy Shapiro, Sagi Grimberg, Liran Liss, Muli Ben-Yehuda, Nadav Amit, and Dan Tsafrir. Page fault support for network controllers. In *ASPLOS*, pages 449–466, 2017. 2

[40] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 574–587, 2023. 7.4

[41] Fan Lin, Matt Beadon, Harish Dattatraya Dixit, Gautham Vunnam, Amol Desai, and Sriram Sankar. Hardware remediation at scale. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 14–17. IEEE, 2018. 1, 2, 2

[42] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a google cluster. In *International*

Conference on Parallel Processing Workshops, pages 397–403, 2012. 1, 2

[43] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. A study of SSD reliability in large scale enterprise storage deployments. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 137–149, Santa Clara, CA, February 2020. USENIX Association. 1

[44] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. Reliability of ssds in enterprise storage systems: A large-scale field study. *ACM Transactions on Storage (TOS)*, 17(1):1–27, 2021. 1, 2, 8

[45] Ioannis Manousakis, Sriram Sankar, Gregg McKnight, Thu D Nguyen, and Ricardo Bianchini. Environmental conditions and disk reliability in free-cooled datacenters. In *14th USENIX conference on file and storage technologies (FAST 16)*, pages 53–65, 2016. 9

[46] Pascale Minet, Eric Renault, Ines Khoufi, and Selma Boumerdassi. Analyzing traces from a google data center. In *International Wireless Communications & Mobile Computing Conference*, pages 1167–1172, 2018. 1, 2

[47] Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *Computer*, 23(7):19–25, 1990. 8

[48] OpenCompute. Server/projectolympus. https://www.opencompute.org/wiki/Server/ProjectOlympus accessed 6/26/2022, November 2017. 1, 2

[49] Jehan-François Paris, Ahmed Amer, Darrell D. E. Long, and Thomas J. E. Schwarz. Self-repairing disk arrays. arXiv cs.DC 1501.00513, 2015. 8

[50] Jehan-François Paris, Darrell D.E. Long, and S.J. Thomas Schwarz. Zero-maintenance disk arrays. In *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, pages 140–141, 2013. 8

[51] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, page 109–116, New York, NY, USA, 1988. Association for Computing Machinery. 8

[52] Borja Peleato, Haleh Tabrizi, Rajiv Agarwal, and Jeffrey Ferreira. Ber-based wear leveling and bad block management for nand flash. In *2015 IEEE International Conference on Communications (ICC)*, pages 295–300, 2015. 2, 8

[53] Sundar Pichai and Ruth Porati. Alphabet announces fourth quarter and fiscal year 2022 results. https://abc.xyz/investor/static/pdf/2022Q4_alphabet_earnings_release.pdf?cache=9de1a6b accessed 2/15/23, 2 2023. 2

[54] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *USENIX FAST*, 2007. 1

[55] Eric L Pope and Scott P Faasse. Post package repair for mapping to a memory failure pattern, January 2020. US Patent 10,546,649. 2, 5.1, 8

[56] Meghan Rimol. Gartner predicts half of cloud data centers will deploy robots with ai capabilities by 2025. https://www.gartner.com/en/newsroom/press-releases/2021-11-01-gartner-predicts-half-of-cloud-data-centers-will-deploy-robots-with-ai-capabilties-by-2025 accessed 2/15/23, 2021. 8

[57] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: A large-scale field study. *Commun. ACM*, 54(2):100–107, feb 2011. 1

[58] Fumiyoshi Shoji, Shuji Matsui, Mitsuo Okamoto, Fumichika Sueyasu, Toshiyuki Tsukamoto, Atsuya Uno, and Keiji Yamamoto. Long term failure analysis of 10 peta-scale supercomputer. *HPC in Asia Poster, ISC*, 2015. 1

[59] Vilas Sridharan and Dean Liberty. A study of dram failures in the field. In *IEEE SC*, pages 1–11, 2012. 1

[60] Kun Tian, Yu Zhang, Luwei Kang, Yan Zhao, and Yaozu Dong. coiommu: A virtual iommu with cooperative dma buffer tracking for efficient memory management in direct i/o. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 479–492, 2020. 2

[61] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *Proceedings of the fifteenth European conference on computer systems*, pages 1–14, 2020. 1, 2

[62] Kushagra Vaid. Datacenter power efficiency: Separating fact from fiction. In *Invited talk at the 2010 Workshop on Power Aware Computing and Systems*, volume 1, 2010. 2

[63] Remco Van Erp, Reza Soleimanzadeh, Luca Nela, Georgios Kampitsis, and Elison Matioli. Co-designing electronics with microfluidics for more sustainable cooling. *Nature*, 585(7824):211–216, 2020. 1

[64] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *ACM EuroSys*, pages 1–17, 2015. 1, 2

[65] Kashi Venkatesh Vishwanath, Albert Greenberg, and Daniel A. Reed. Modular data centers: How to design them? In *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance*, LSAP '09, page 3–10, New York, NY, USA, 2009. Association for Computing Machinery. 8

[66] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204, 2010. 1, 2

[67] Osamu Wada, Toshimasa Namekawa, Hiroshi Ito, Atsushi Nakayama, and Shuso Fujii. Post-packaging auto repair techniques for fast row cycle embedded dram. In *2004 International Conferce on Test*, pages 1016–1023. IEEE, 2004. 2, 5.1, 8

[68] Chundong Wang and Weng-Fai Wong. Extending the lifetime of nand flash memory by salvaging bad blocks. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 260–263, 2012. 2, 8

[69] Guosai Wang, Lifei Zhang, and Wei Xu. What can we learn from four years of data center hardware failures? In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 25–36, 2017. 1, 2, 2

[70] Paul Willmann, Scott Rixner, and Alan L Cox. Protection strategies for direct access to virtualized i/o devices. In *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008. 2

[71] Ben-Ami Yassour, Muli Ben-Yehuda, and Orit Wasserman. On the dma mapping problem in direct device assignment. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, pages 1–12, 2010. 2

[72] Yangfan Zhong. Experiences with immersion cooling in alibaba datacenter. OpenCompute 2019 track on Rack & Power / Advanced Cooling https://www.youtube.com/watch?v=GMSLjr7Wlis&t=1067s accessed 6/26/2022, October 2019. 1, 9

[73] Ali Zolghadri. A redundancy-based strategy for safety management in a modern civil aircraft. *Control Engineering Practice*, 8(5):545–554, 2000. 8