Workload Intelligence: Punching Holes Through the Cloud Abstraction

Lexiang Huang[1,3], Anjaly Parayil[3], Jue Zhang[3], Xiaoting Qin[3], Chetan Bansal[3],

Jovan Stojkovic[2,3], Pantea Zardoshti[3], Pulkit Misra[3], Eli Cortez[3], Raphael Ghelman[3], Íñigo Goiri[3],

Saravan Rajmohan[3], Jim Kleewein[3], Rodrigo Fonseca[3], Timothy Zhu[1], Ricardo Bianchini[3]

[1]*The Pennsylvania State University*   [2]*University of Illinois at Urbana Champaign*   [3]*Microsoft*

Abstract

Today, cloud workloads are essentially opaque to the cloud platform. Typically, the only information the platform receives is the virtual machine (VM) type and possibly a decoration to the type (*e.g.*, the VM is evictable). Similarly, workloads receive little to no information from the platform; generally, workloads might receive telemetry from their VMs or exceptional signals (*e.g.*, shortly before a VM is evicted). The narrow interface between workloads and platforms has several drawbacks: (1) a surge in VM types and decorations in public cloud platforms complicates customer selection; (2) essential workload characteristics (*e.g.*, low availability requirements, high latency tolerance) are often unspecified, hindering platform customization for optimized resource usage and cost savings; and (3) workloads may be unaware of potential optimizations or lack sufficient time to react to platform events.

In this paper, we propose a framework, called Workload Intelligence (WI), for dynamic bi-directional communication between cloud workloads and cloud platform. Via WI, workloads can programmatically adjust their key characteristics, requirements, and even dynamically adapt behaviors like VM priorities. In the other direction, WI allows the platform to programmatically inform workloads about upcoming events, opportunities for optimization, among other scenarios. Because of WI, the cloud platform can drastically simplify its offerings, reduce its costs without fear of violating any workload requirements, and reduce prices to its customers on average by 48.8%.

1   Introduction

From its inception with a simple interface of offering virtual machines to users, today's cloud has grown incredibly complex for both cloud providers and workload owners. Cloud providers constantly seek to improve their efficiency, which gives rise to many optimizations. Some of these are exposed to workload owners via VM types (*e.g.*, Spot VMs [4, 13, 17],

Harvest VMs [6], Burstable VMs [11, 15, 55]) and dedicated interfaces (*e.g.*, auto-scaling [34]), whereas other optimizations are internal to the cloud (*e.g.*, oversubscription [26], pre-provisioning [60]). These can reduce costs, and/or improve efficiency, performance, sustainability, or reliability. Even with these interfaces, cloud workloads are essentially opaque since cloud platforms lack visibility into the desires and expectations of workload owners. Hence, the platforms are limited to inferring workload characteristics, adding/extending interfaces, or being overly conservative. This results in undesirable performance effects, increased complexity, and higher costs, negatively impacting both workload owners and providers.

On the other hand, workload owners lack visibility and control over the impact of the provider's optimizations on their workloads. Typically, they only receive telemetry from their VMs and any exceptional signals (*e.g.*, shortly before a VM is evicted). As a result, performance-conscious workload owners need to develop workarounds to cope with the variability and unknowns (*e.g.*, spawning multiple VMs to find ones without noisy neighbors [33]).

Revisiting the cloud interface. The narrow communication interface between workloads and platform has multiple negative effects: (1) the number of VM types and decorations has exploded in public cloud platforms, making it difficult for workload owners to select the ideal ones; (2) many important workload characteristics (*e.g.*, low availability requirements, high tolerance to latency) are never made explicit, so the platform is unable to customize its service to them (*e.g.*, by optimizing their resource usage and passing any dollar savings to workload owners); and (3) workloads often are unaware of optimizations that they could make or do not have enough time to react to platform events.

Our work. In this paper, we study how internal workloads use cloud platform optimizations and identify the fundamental characteristics that cloud platform optimizations require to operate. Based on this characterization, we propose a framework, called Workload Intelligence (WI), for dynamic bi-directional communication between cloud workloads and cloud platform.

---

[1]Lexiang Huang was an intern at Microsoft.
[2]Jovan Stojkovic was an intern at Microsoft.

1

Category	Workload characteristics	Core usage						
Scalability	Stateless nature	Stateless		Partially stateless		Stateful		
		45.5%		17.4%		37.1%		
	Deployment time requirements	Strict		Not strict				
		28.5%		71.5%				
Reliability	Availability	Five Nines	Four Nines	Three Nines	Two Nines	One Nine	None	
		2.4%	34.5%	58.0%	3.9%	0.5%	0.4%	
	Preemptibility	0%	0-20%	20-40%	40-60%	60-80%	80-100%	100%
		39.3%	41.1%	4.8%	6.5%	0.3%	1.8%	6.1%
Performance	Delay tolerance	Delay tolerant		Delay sensitive				
		24.5%		75.5%				
Geographical	Region independence	Region-agnostic		Partially region-agnostic		Not region-agnostic		
		47.5%		13.9%		38.6%		

Table 1: Overview of cloud workload characteristics and their core usage based on an internal survey.

WI enables workloads to programmatically communicate their key characteristics, requirements, dynamic changes, and shifting behaviors.

In the other direction, WI allows the platform to programmatically notify workloads about upcoming events, optimization opportunities, and other useful scenarios. With WI, the cloud platform can drastically simplify its offerings, reduce costs without fear of violating any workload requirements, and lower prices for workload owners.

In building WI, we address three key challenges: (1) creating a general, extensible, and incrementally adoptable interface for workloads to express their main characteristics, requirements, and dynamic behaviors; (2) developing a dynamic framework for the platform to seamlessly interact with workloads, even at a potentially high rate, while preventing potential attacks or bugs that could harm it or other workloads; and (3) enabling the platform to process received information intelligently, maximizing optimization opportunities, and maintaining quality of service.

We start by exploring the characteristics and requirements of 188 real cloud workloads at a major cloud provider (ProviderX for blind review). Next, we discuss optimizations that the platform can perform by knowing these characteristics and requirements, enabled by its bi-directional communication capability with workloads. Afterward, we introduce the design and implementation of WI, focusing mainly on how WI effectively tackles the four challenges. To evaluate WI, we explore various workload and optimization scenarios.

We conclude that it is possible to build a frameworks for bi-directional communication between workloads and the platform in a safe and effective manner. By punching holes in the cloud abstraction for this communication, WI can simplify cloud offerings and make platforms more efficient and cost-effective, while providing excellent service to workloads.

Summary. We make the following contributions:

- We identify the fundamental workload characteristics that cloud platform optimizations require to operate and show their savings potential.
- We develop Workload Intelligence (WI), a novel and extensible framework that enables bidirectional communication between workloads and the cloud provider for improving cloud efficiency.
- We evaluate the applicability and potential of WI across ten cloud optimizations at ProviderX and demonstrate that WI can on average save workload costs by 48.8%.

2 Characterizing workloads in the cloud

We answer 2 research questions: (1) What are the characteristics and requirements of cloud workloads? (2) What workload characteristics are required to enable cloud optimizations?

2.1 Workloads

Methodology. We study the characteristics of a diverse set of cloud workloads running at ProviderX. We surveyed all the 990 internal workloads and got responses from 188 of them. This represents 19% of the workloads and 1.4 million cores across over 400K VMs. These include web search, collaboration and productivity suites, and real-time communication workloads. They are deployed across 48 regions worldwide and used by hundreds of millions of users.

Results. We divide the results by workload characteristics, weighted by core usage. Table 1 shows the core usage for each characteristic. We group these into four main categories: scalability, reliability, performance, and geography sensitivity.

First, under *scalability*, we characterize workloads by their *statelessness* (*i.e.*, feasibility to scale in/out without persisting states) and *deployment time requirements* (*i.e.*, whether VMs have strict deployment latencies). According to the survey, 62.9% of the workloads are partially to fully stateless and the majority does not have strict deployment time requirements.

Cloud Optimization	Cloud Resources	User Benefit (Average)	Min Pricing	Max Pricing	Platform Benefit Model
Auto-scaling	Compute	19% less cost, ↓ Carbon	Average number of regular VMs		Compute allocation
Spot VMs	Spare compute	85% less cost	15% regular VM		Compute allocation
Harvest VMs	Spare compute	91% less cost	Spot VM	Spot VM+Harvested	Compute allocation
Overclocking	CPU frequency	11% less cost, ↑ Perf	Regular VM	Regular VM+OC time	Reliability, power/energy
Underclocking	CPU frequency	1% less cost, ↓ Carbon	99% Regular VM	Regular VM	Power, energy
Non pre-provision	Spare compute	2% less cost	98% Regular VM	Regular VM	Compute allocation
Region agnostic	Compute	22% less cost, ↓ Carbon	Region price		Efficient region
VM oversubscription	Compute	15% less cost, ↓ Carbon	85% Regular VM		Compute allocation
VM rightsizing	Compute	50% less cost, ↓ Carbon	Rightsized VM		Compute allocation
MA datacenters	CPU frequency	40% less cost	60% Regular VM		Infrastructure cost

Table 2: Benefits and incentives of the cloud platform optimizations for the users.

Second, under *reliability*, we look at *availability* (*i.e.*, cloud downtime tolerance) and *preemptibility* (*i.e.*, ability to pause and resume progress if X% of the VMs are still alive). The survey responses show that 62.8% of the cloud workloads require three nines of availability or less, which translates to 8.76 hours to 36.5 days system downtime tolerance per year [14]. Besides, 60.6% of the cloud workloads are at least partially preemptible. These provides flexibility for cloud to manage resources more efficiently. Workloads with 100% preemptibility or 0% availability requirement are generally test and dev environments.

Third, *delay tolerance* shows the workload flexibility within a specified deadline. For example, a workload serving requests may have a tail latency service level objective (SLO) of 100 ms while most requests complete within 20 ms. The specific target metric depends on the workload. Our data indicates that around a quarter of the cloud workloads are tolerant to delays and they have a less strict performance requirement for the cloud platform.

Lastly, *region independence* shows the workloads' ability to migrate among geographical regions without restrictions such as workload dependencies and security policies, and 61.4% of the workloads are partly to fully available to migrate without negative impact on their operation.

2.2 Cloud optimization mechanisms

Cloud platforms implement many optimizations to improve their efficiency. To make our discussion concrete, we now look at ten common cloud optimizations among public cloud providers. Table 2 summarizes the resources, workload owner benefits (*i.e.*, how much workload owners can save), pricing (based on public cloud pricing), and the platform benefit model (*i.e.*, how does the platform benefit) for each cloud platform optimization. The rest of this section explains the mechanism, interface, and required workload characteristics for each optimization.

Auto-scaling. To allow workload owners to not always provision VMs for the peak load, providers offer auto-scaling to dynamically adjust the number of VMs based on load [34].

This allows owners to save money by running fewer VMs when not needed and providers to monetize this free capacity.

Auto-scaling is usually offered as a separate service [9, 41]. Workload owners define their own policy defining a time schedule (*e.g.*, scale out from 1 to 4 PM) or a load threshold (*e.g.*, scale out if the CPU utilization is higher than 40%) and it is suitable for workloads that allow scaling in and out, which is characterized by their stateless and delay tolerant nature. Owners also need to specify *deployment time requirement*, if the workload requires a VM to be immediately available.

Spot VMs. To monetize unallocated capacity, providers offer VMs with relaxed SLOs. These VMs are evicted if their resources are needed by on-demand VMs. Spot VMs are offered at discounted prices which allows owners saving money to run their workloads. Providers usually offer Spot VMs [4, 13, 17] as a VM type or deployment flag and may offer dynamic pricing to decide which Spot VMs to evict first. Spot VMs are ideal for workloads that tolerate evictions [5, 12]. These are workloads that support preemptions (*i.e.*, 20% or higher).

Harvest VMs. To use unallocated resources, cloud platforms can place more Spot VMs. However, it's inefficient to create/remove VMs to use all the resources in a server. Harvest VMs build on top of Spot VMs and can dynamically grow and shrink to utilize spare CPU [6, 57], memory [21], and storage [40] in the server. This is similar to Burstable VMs [11, 15] but without the credit abstraction. Providers offer harvesting as a new VM type or a deployment flag specifying the amount of resources to harvest [6]. In addition to the characteristics from Spot VMs (*i.e.*, high preemptibility), Harvest VMs are ideal for workloads that can scale up/down and thus they need to tolerate delays.

Overclocking. To improve workload performance, cloud platforms can increase component-level (*e.g.*, CPU cores) frequency for VMs [28]. To provide the benefit, the platform needs to determine the bottleneck resource to overclock and factor the impact of overclocking on component reliability and power draw in its decision-making. The capability is provided to workload owners via dedicated VM types. Workloads can also use an interface similar to auto-scaling to define a

3

time schedule or a load threshold as signals to the platform for their overclocking requirements. This optimization targets workloads with high CPU utilization periods [28] (*i.e.*, 95th percentile of max CPU utilization greater than 40%), that can scale up/down, and are tolerant to delays. Overall, owners can provision fewer VMs to serve their peaks using overclocking.

Underclocking. Platforms can reduce their energy usage and carbon footprint by decreasing the frequency of VMs during periods of low activity. This optimization is available through certain VM types offered by providers, similar to overclocking. Workloads that are delay-tolerant, support scaling down, or have no persistent state and can handle delays are well-suited for this optimization.

VM pre-provisioning. To reduce the time to create a VM, providers may provision VMs ahead of the time being instantiated when requested by workloads, hence, reducing the time to deployment [60]. This is a good complement to auto-scaling as it allows adding VMs quickly when needed (*e.g.*, a load spike). But cloud providers currently provision VMs without considering their utility to the workload they serve. Disabling VM pre-provisioning (*i.e.*, Non pre-provisioning) for workloads without strict deployment time requirements can reduce costs with minimal impact on performance.

Region-agnostic placement. Running workloads on VMs in cheaper and greener regions (*i.e.*, regions with lower CO_2 emissions) can help reduce costs and carbon footprints, especially for workloads that do not have strict latency or data-locality requirements. Currently, cloud providers require workload owners to specify the region for VM deployment. Although there have been proposals for semi-automatic region selection [2, 48], no commercial solutions are available yet. Providers can place/migrate workloads to cheaper and greener regions (*e.g.* utilizing solar energy) when needed if the workloads are region-agnostic.

VM oversubscription. To increase server utilization, cloud platforms may oversubscribe servers by deploying more VMs on them than the available resources, relying on statistical multiplexing to manage resource allocation. However, if all VMs spike at the same time, the platform will throttle the least critical VMs to ensure stability. Currently, platforms heuristically determine which VMs can be oversubscribed and to what degree [19] or offer oversubscribed VM types explicitly [7]. Further knowledge of the workload characteristics can help identify good candidates for oversubscription. If the 95th percentile CPU utilization of a workload is less than 65% and the workload is delay-tolerant or non-user-facing, then it is suitable for oversubscription [19].

VM rightsizing. To enhance efficiency and minimize expenses for workload owners, the platform provides smart VM selection by identifying VM miss-utilization and recommend transitions to more suitable types/sizes. Automated adjustments apply to preemptible workloads with relaxed availability requirements. This is advantageous for workloads capable of scaling down less utilized components (*e.g.*, below 50%), facilitating a move to a smaller VM, typically half the original size. Conversely, if a single resource encounters high usage, the VM type can be upgraded. Overall, optimal VM selection considers factors like workload resource needs, performance requirements, and budget constraints.

Multi-availability datacenters (MA DCs). Cloud providers can reduce infrastructure redundancy (*e.g.*, power delivery and cooling) to decrease costs. However, this may lead to infrastructure failures or maintenance events that require the platform to throttle or selectively turn off servers. Traditionally, platforms have inferred which VMs are less critical and throttled them down or evicted them. MA DCs take further advantage of workloads that explicitly require low availability, providing resources and charging users accordingly.

3 Revisiting the cloud abstraction

From the descriptions of the optimizations above, we observe that each one relies on a different subsets of workload characteristics, and has particular interfaces to gather inputs from workload owners. Table 3 summarizes the required workload characteristics and interfaces for each optimization.

Problems with these interfaces are four-fold: (1) They are ad-hoc: some are specific to a service, some rely on VM type and deployment flags, some rely on inference, or are just based on recommendations. As cloud platforms introduce new optimization mechanisms, the overall cloud interface becomes complex and untenable. For example, *Spot VMs*, *Harvest VMs*, *Overclocking*, and *Underclocking* each require an additional dimension in the already-complicated VM type interface, which limits their usage. (2) These interfaces are tied to the corresponding optimizations and require expertise with the corresponding optimization. For instance, workload owners need to know Harvest VMs can shrink/expand core counts and are applicable if their workloads can scale up/down. (3) They are mostly static, as they tend to be specified at deployment time. (4), Many of these interfaces rely on inferred characteristics with questionable accuracy and lack of explicit user contracts. As a result, the provider in many cases has to be conservative, and not utilize optimizations to their fullest extent. For example, in the absence of extra information, the provider has to assume that a VM requires maximum reliability, and cannot move to other regions.

To address these challenges, we observe that ideally the cloud interface should *decouple* workload characteristics – which are known to workload owners – from the cloud optimizations they enable – best understood by the cloud provider. This creates a proper separation of concerns, can reduce interface complexity and changes as optimizations evolve, and can enable the effective utilization of cloud optimizations. In this paper we propose an extension to the cloud interface that enables this separation. Before describing our proposal in Section 4, we first discuss some challenges and requirements that such an extension should meet.

4

Cloud Optimization	Cores (%)	Existing Ad-hoc Interface	Required Workload Characteristics						
			Scale up/down	Scale out/in	Deploy time	Availability	Preemptibility	Delay tolerance	Region independence
Auto-scaling	33.1	Dedicated service		✓	✓			✓	
Spot VMs	21.6	VM type, deployment flags, dynamic pricing					✓		
Harvest VMs	6.4	VM type, deployment flags	✓				✓	✓	
Overclocking	41.3	VM type	(✓)					✓	
Underclocking	36.0	VM type	(✓)				✓	✓	
Non pre-provision	68.8	Inferred			✓				
Region-agnostic	43.0	Explicit region selection							✓
VM oversub	7.6	VM type, inferred	(✓)					✓	
VM rightsizing	2.1	Inferred, recommendation	(✓)			✓	(✓)		
MA datacenters	59.6	Inferred				✓			

Table 3: Overview of popular cloud optimizations, with percentages of applicable cores in the cloud platform, existing cloud interfaces, and required workload characteristics. To calculate the core percentages, we sum up the percentage of cores from workloads in the internal survey that has a specific workload characteristic. (✓) indicates optional characteristic.

3.1 Challenges

Generality. We need to support a wide range of workloads and cloud platform optimizations. The interface needs to be *general* for any workload to express their main characteristics and requirements. These characteristics can also be dynamic and change over time. In addition to the interface at deployment time, we need to expose an interface to allow updating the characteristics and requirements at runtime.

Based on the survey results and our internal discussions with cloud optimization teams, Table 3 summarizes the essential workload characteristics that cloud optimizations need to operate. Note that many of these characteristics benefit multiple cloud optimizations. We target these ten cloud platform optimizations but this number may grow. In addition, some workloads may introduce new characteristics. The interface needs to be *extensible*.

Critical optimizations (*e.g.*, *MA DCs*) require to push updates to the platform in real-time, while other optimizations (*e.g.*, *Spot VMs*) may pull information only when needed (*e.g.*, to create room for on-demand VMs). The same applies to workloads. For example, for *Spot VMs*, we need to push priority updates and receive future eviction events. Therefore, we need to provide both pull and push interfaces.

Incentives. Workload owners must be incentivized to use any interface extensions. It is clear from Table 2 that the optimizations have the potential to reduce cost and/or improve performance, sustainability for workloads. The interface implementation should guarantee that VMs are *no worse off* by using the interface, and possibly better. Any extension should also be *incrementally adoptable*, meaning that performance/cost/sustainability should not degrade by not adopting the interface, after its introduction.

Safety. As public cloud platforms, safety must be ensured. We need to prevent workloads from providing wrong information (*e.g.*, due to bugs) and ensure its public interfaces are resilient to attacks (*e.g.*, denial of service). The interface should not enable workloads to abuse the system.

For workload owners, we must protect against leaks of sensitive information from workloads by using interface isolation and encryption to prevent side-channel attacks. At the same time, we need to ensure the *correctness* of the information to prevent performance degradation or unnecessary cost.

Coordination. The goal of our extension is to enable the cloud platform to reason about the information it receives and optimize accordingly, while maintaining quality of service. To achieve this, any implementation must aggregate data at different levels for various optimizations. For example, *Auto-scaling* considers all the VMs for a workload, *Overclocking* considers physical domains (*i.e.*, servers and racks), and *Spot VMs* considers all the VMs that can be evicted.

We must also enable *coordination* between multiple cloud platform optimizations that may want to take actions on the same resources. This coordination is necessary to resolve conflicts that may arise when, for example, both *Overclocking* and *MA DCs* are attempting to adjust CPU frequencies simultaneously. At the same time, we need to ensure the resources are shared *fairly* among multiple workloads.

3.2 Requirements

Besides addressing the above challenges, any implementation of a cloud interface extension must address:

Scalability. It must be *scalable* and handle a high rate of dynamic bi-directional communication between many workloads and multiple cloud optimizations. Potentially, it needs to support exchanging information from/to all VMs in the cloud platform.

Availability. The use of any new interface must maintain *high availability* and *tolerate failures*. The new information provided must be *persisted* even if cloud optimizations or workloads are restarted.

Efficiency. The extension needs to be *low-overhead* and avoid imposing unnecessary burdens and overheads on the system.

Maintainability. Maintaining a new service requires effort for development, operation, bug fixing, and others. We need to build a *simple* service with minimal maintenance overhead and rely on existing infrastructure as much as possible.

4   Workload Intelligence

Considering these challenges and requirements, we propose Workload Intelligence (WI) as an extension to the cloud interface: a framework for dynamic bi-directional communication between cloud workloads and the cloud. WI allows workloads to explicitly specify their characteristics and requirements through *hints* and to dynamically change them. WI makes hints available to the cloud platform to optimize its operation.

Hints. They are *best-effort*: there is no guarantee that they will be fulfilled, but they may (only) improve some dimension of quality: price, performance, sustainability, etc. They are also *incentive-compatible*: in their absence, the provider assumes the most conservative version of a workload characteristic.

With WI in place, the cloud platform can significantly simplify its offerings, reduce costs without fear of violating workload requirements, and lower prices for workload owners. Figure 1 shows an overview of WI enabling communication between multiple workloads and optimizations.

Workload hints. We define seven hints based on the workload characteristics needed by cloud optimizations identified in Table 3: (1) scale up/down (boolean), (2) scale out/in (boolean), (3) deploy time (milliseconds), (4) availability (number of 9s), (5) preemptibility (percentage), (6) delay tolerance (milliseconds), and (7) region independence (boolean).

Hints define if that particular characteristic is relaxed (*e.g.*, the workload has low availability requirements). If unspecified, we assume the most conservative setting (*e.g.*, the workload wants fast deployment times for its VMs). Common workload targets and goals (*e.g.*, cost and CO_2) are not hints.

Platform hints. In the other direction, WI allows the platform to programmatically inform workloads about upcoming events and opportunities for optimization, among other useful scenarios. Example hints include VM evictions for Spot VMs and higher CPU frequency available for overclocking. Workloads can then react to these hints by specifying a VM with the lowest penalty upon eviction for graceful shutdown or a VM with the highest benefit for overclocking.

4.1   Architecture

Figure 2 shows the WI architecture with an example for three cloud optimizations: Spot VMs, Overclocking, and MA DCs.

Local managers. For *scalability*, each server in the cloud provider runs a local WI manager as shown in the left of Figure 2. This local manager collects the runtime hints from the workloads running in the VMs in the server and passes them to the global manager. This local manager also collects
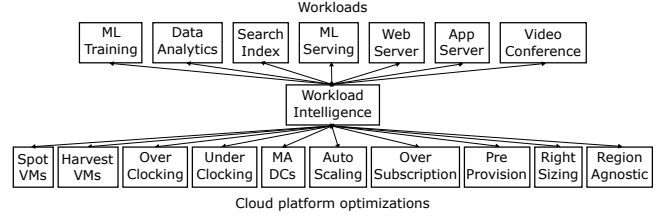

Figure 1: Workload Intelligence overview.

the notifications coming from the cloud optimizations and exposes them to the VMs running the workloads.

Global manager. For every region, we have a global WI manager that is logically centralized but physically distributed. This is shown in the in the center of Figure 2. This component stores the hints, aggregates them, and enables *coordination* across multiple cloud optimizations and multiple workloads. It acts as a broker that exchanges information and hints between the cloud optimization and the workloads running in VMs. It provides multiple interfaces to retrieve this information at scale in near real-time and aggregate it at multiple granularities (*e.g.*, per server and per rack).

Cloud optimization managers. Each optimization can leverage a basic WI optimization manager. This manager gets the hints from the global manager. It also uses the global manager to pass hints to the workloads through the local manager. The right of Figure 2 shows the example for three optimizations.

4.2   Communicating and storing hints

To provide *scalability*, *high availability*, *maintainability*, and *fault tolerance*, WI uses a combination of a PubSub and a distributed database to communicate and store the hints. For the PubSub, WI uses Kafka [56] which synchronously delivers the hints at *large scale*. For the database, it uses CloudDB[1] which provides *fault tolerance* and *durability*. These are also mature services that are *easy to maintain*.

Depending on the use case, hints need to be sent synchronously or asynchronously. For example, in the Spot VM case, workloads can specify their evictions preference asynchronously and the cloud platform gathers this information whenever it needs the capacity for regular VMs. On the other hand, when the cloud platform decides to evict VMs, it needs to immediately notify the VM.

Deployment hints. When deploying a VM (or a set of VMs), the workload owner can specify the attributes for the workload that they will run (*e.g.*, tag a set of VMs as highly preemptible). Workload owners can specify these hints through the common deployment interfaces [10, 16, 42]. The cloud platform internally uses the WI global manager REST interface to store these hints. The global manager stores the deployments hints in CloudDB and publishes them using Kafka. Cloud platform
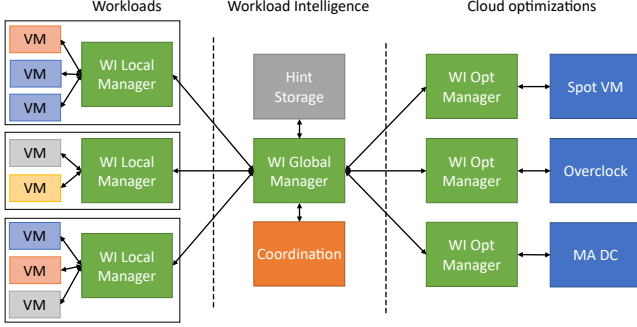
---

[1]Fictitious name for a cloud database, for anonymity.

Figure 2: Workload Intelligence architecture showing three example cloud optimizations.

Priority	Cloud optimization
0	On-demand
1	MA datacenters
2	Rightsizing
3	Oversubscription
4	Auto-scaling
5	Non pre-provision
6	Region agnostic
7	Underclocking
8	Overclocking
9	Spot VMs
10	Harvest VMs

Table 4: Priorities across our ten cloud optimizations.

optimizations can leverage the optimization manager to (1) retrieve this hints asynchronously when needed or (2) subscribe to the hints of a particular type. These two actions offer a *general* interface that can be used by a wide variety of workloads. Workloads can also update their hints after deployment while the VMs are running.

Runtime hints: workload → platform. The workloads running in the VMs can also provide hints. For example, a VM that it is currently not running critical jobs can specify it is more tolerant to evictions. These runtime hints can be specified within the VM or from a logically centralized manager.

For a workload running inside of a VM to set a runtime hint, it uses local interfaces (*e.g.*, Hyper-V KVP [37] or XenStore [59]). The local WI manager in each server polls for these runtime hints and uses Kafka to publish them. The global manager is subscribed to these events and stores them in CloudDB. The optimization manager can also subscribe to these events or asynchronously check through CloudDB.

In addition to the local hints that VMs can specify, a global workload manager can use the global manager REST interface to specify hints. For example, the Resource Manager in a Hadoop YARN [53] deployment can set preference for evictions for a set of VMs.

Multiple entities can be publishing hints for the same resource. To provide *correctness*, if the cloud platform optimiza-

tion identifies that the hints are not consistent, it can notify the workload that it is ignoring them.

Runtime hints: platform → workload. To let workloads adapt to the cloud platform actions, such as evictions of Spot VMs, WI provides a mechanism to send hints (*e.g.* early notifications) from the cloud optimizations to the workload via Kafka. The global manager is subscribed to these events and stores these hints in CloudDB. The local manager also subscribes to these events and exposes them to the VM through the local interfaces. Cloud platforms already offer interfaces for VMs to locally check their attributes. An example is the metadata service [18, 35, 45] which offers data like the VM identifier or the type of VM. Scheduled events [38, 46] is another communication channel which notifies about events that will happen soon (*e.g.*, reboot, maintenance, evictions).

The cloud platform can also inform the workload that a planned maintenance event will take place. With this info, the workload can shut down gracefully.

4.3 Providing hints safely

WI ensures safety for both workload owners and the cloud platform. To protect the interface against DoS attacks, we enforce maximum rates per optimization and workload when setting deployment and runtime hints for all interfaces separately. As hints are best-effort, DoS mitigations are simpler.

To protect the cloud platform from abusive usage of a single resource, the cloud enforces fair-share among VMs and between workload owners. Also, the cloud platform has the right to provide alternative optimizations (*e.g.*, Spot VM vs. VM Pre-provisioning) to suit workloads' need.

To protect workload owners from side-channel attacks, we encrypt the hint communication. Besides, the cloud platform does not provide details on its resource management decisions after applying (or ignoring) hints to prevent information leaks (*e.g.*, VM placement). For example, in the case of a Harvest VM expansion, the cloud platform does not give reasons on their decisions and only the target VM is directly informed.

To protect the owners from sending wrong information, the cloud platform ignores any inconsistent/incompatible hints based on history. In addition, from the incentives point of view, workloads can only hurt their own performance and cost by providing wrong information.

4.4 Coordinating optimizations

WI enables the cloud platform to reason about the information it receives to maximize its opportunities for optimization while protecting quality of service. This includes handling conflicts that may arise when multiple cloud optimizations target the same resources. For example, both Spot VMs and Pre-provisioning try to use the unallocated capacity in servers. Another example, *Overclocking* may try to increase the CPU frequency [28] and *MA* may try to reduce it [28, 61].

To address potential conflicts, we implement an algorithm based on cloud optimization priorities. Table 4 shows our pri-
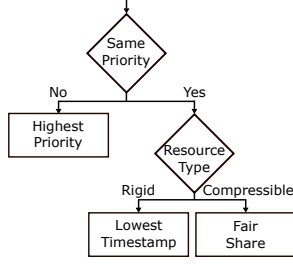
Figure 3: Conflict resolution for competing resources.

orities for the ten discussed optimizations (on-demand VMs have the highest priority). This ranking reflects their significance for the provider, with Harvest VMs being the most opportunistic. Note, most optimizations do not compete for the same resources and many of them (*e.g.* VM rightsizing) are designed to release unnecessary resource (Table 2).

Figure 3 illustrates this algorithm. In cases where the optimizations have the same priority but the resource is compressible (*e.g.*, CPU), we employ *fair* share allocation. Otherwise, we allocate the resource to the VM with the earliest request time. In the rare event of requests submitted simultaneously, we pick randomly. Moreover, the WI ensures the fair sharing of resources among multiple VMs, guaranteeing equitable distribution between workloads.

4.5 Alternative designs

We discuss trade-offs between several design choices with a focus on coordination policy and interface.

Coordination. To resolve conflicts when the cloud platform attempts to apply multiple optimizations on the same resources, one can envision three approaches.

Pricing. The cloud platform specifies a price for each of the optimizations. It can then use this pricing to resolve conflicts. However, each optimization has different magnitudes, making it challenging to unify prices into the same unit. Additionally, the price can change over time and may not always reflect the intrinsic value of the resources. (*e.g.*, special discounts).

Bidding. Workload owners can define a price they are willing to pay for their VMs. The cloud platform can then use this information to resolve conflicts and assign resources to the highest paying workload owners. This is similar to bidding pricing strategy for Spot VMs, which has been either dropped or unsupported by major cloud providers [13, 17, 20]. Fundamentally, this approach is challenging for workload owners to understand and could introduce a gaming aspect.

Our approach. When building a new cloud optimization, we define a priority for each optimization and apply a set of rules depending on the resource in conflict. This approach allows for easy unification and is simple for workload owners to understand and reason about. At the same time, it makes it easy for the platform to operate and maintain.

Interface. To offer cloud optimizations to workload owners, one can envision three major interface designs.

Reduced. To reduce the complexity of the cloud systems and increase the generality of the interface, one can simplify the interface to take no workload owner inputs. For example, instead of letting owners to specify the VM types and required resources, the platform can infer the workload characteristics from the owners' past history and provide a VM type based on its best guesses. This approach works well with predictable workloads (*e.g.*, scheduled events), however, it does not support optimizations to react to workload changes in time (*e.g.*, delayed auto-scaling, sub-optimal Spot VM evictions).

Discrete. The current design maintains discrete interfaces for each individual optimization. This preserves the most flexibility for individual systems, which could potentially lead to well-optimized cloud operations. However, the predominant drawback is its sheer complexity, since each interface is customized. Besides, this design is not extensible, as the number of interfaces explodes as new cloud optimizations emerge.

Centralized. A key observation from the discrete design is that many of the existing interfaces require similar workload characteristics from the workload owners either directly or implicitly. Instead of letting owners indirectly or repetitively submit their requirements, we can create a centralized interface that aggregates the minimum set of workload information needed and store them at one place. This design has lower complexity, but the communication overhead is potentially high, since all workloads and optimizations need to communicate via the centralized storage.

Our approach. To combine the benefits of both the discrete and the centralized design, we propose a hybrid interface. In our design, the hints are physically distributed among workloads and optimizations (*i.e.*, WI Local Managers and WI Opt Managers), to provide flexibility to update their runtime status at desired frequency. The WI Global Manager aggregates and distribute hints at one place (*i.e.*, logically centralized). This allows optimizations and workloads to exchange hints on demand. In addition, the workload characteristics utilized by WI are shared among multiple cloud optimizations and multiple workloads. To onboard future workloads/optimizations, the WI interface only needs to be updated for the delta, if any.

4.6 Other resources

While current cloud optimizations primarily focus on compute resources, WI has the potential to benefit other resources (*e.g.*, storage and networking). For storage, *delay tolerance* hints enable opportunities for co-locating storage and compute for workloads with I/O bottlenecks or using cheaper storage for lower costs in delay-tolerant workloads. *Region independence* hints can indicate data locality requirements and security concerns (*e.g.*, GDPR), which can help enforce the desired data replication configurations.

For networking, cloud Load Balancer services can benefit

8

from *scalability* and *availability* hints to make better task placement decisions. Also, *delay tolerance* hints can be used in future optimizations to adjust cloud Content Delivery Network (CDN) service levels to reduce costs for delay-tolerant workloads or improve performance for delay-sensitive workloads. It can be used in conjunction with *region-independence* hints to optimize which regions to cache the data.

5 Implementation

5.1 Extending workloads beyond VMs

Many workloads leverage frameworks and orchestrators like Kubernetes [31] and Functions-as-a-Service (FaaS) [47] for their deployment. Cloud providers offer these managed frameworks [8, 36, 43, 44] and they are usually deployed on top of VMs [3, 36, 44, 47].

These frameworks can leverage WI to orchestrate the workloads running on top or directly expose WI. This enables workloads running on these frameworks to take advantage of cloud optimizations with minor extensions. In this section, we describe how to extend three common frameworks.

Big data analytics: Hadoop. To support WI, we extend Harvest Hadoop [6, 21]. The management components (*e.g.*, Resource Manager and Name Node) run on VMs with high requirements while the workers run in a mix of VMs with low and high requirements.

We use the WI interface to retrieve notifications for evictions and change of resources (*e.g.*, more CPU or memory available). Then we pass this information to Harvest Hadoop which already handles evictions and changes in the number of resources (*e.g.*, CPU and memory).

We also add a new component to the workers that uses the local WI interface to specify the VM priority depending on: the criticality, the amount of containers running, elapsed job processing time, and whether containing master nodes. For example, a VM that is running many critical containers will have a "High" priority while an empty node will have "Low". This will make "High" priority VMs less likely to be evicted and to get more resources (*e.g.*, harvesting or overclocking).

Microservices: Kubernetes. The Kubernetes control plane components run in VMs with high requirements (*e.g.*, high-availability, low preemptibility). When provisioning the worker nodes, we leverage Karpenter [30], a node provisioning manager for Kubernetes clusters. We extend Karpenter to provide hints based on the pod requested by the applications. The worker nodes can be a mix of Regular, Spot, Harvest, Overclocking, Underclocking, and Oversubscribed VMs. These workers are grouped into different pools based on their characteristics. We also add a new component to each VM which runs next to the Kubelet and uses the WI interface to provide and receive hints. For example, if a VM is to be evicted, the WI component uses graceful shutdown to stop the pods in that VM and migrate the load to other pods.

Workloads running on Kubernetes use the node pool ab-

straction through tolerations and node affinities [32]. For example, if we want to run the Social Network from the Death-StarBench [22] on Kubernetes [39], we specify the frontend pods to run in the node pool with high-preemptibility. In addition, the logic microservices (*e.g.*, compose post, social graph, write timeline) can specify when their latency is too high and trigger optimizations like overclocking.

FaaS: OpenWhisk. We use the OpenWhisk implementation from FaaS on Harvest VMs [62] as a base. Similarly, to Hadoop and Kubernetes, we run the logically centralized control plane components (*e.g.*, Nginx, Controller, Kafka) on VMs with high requirements (*e.g.*, low fault tolerance and high-availability). For the worker components, we use VMs with heterogeneous requirements based on the workload. We add a component to track the running functions and adjust the characteristics of the VMs that are running worker nodes. For example, if we have many long running functions, we deploy more regular VMs while if the functions are mostly short, we deploy them on VMs with lower requirements.

In the worker node, we extend the *Resource Monitor* that runs next to the *Invoker* to interact with the local WI interface. Depending on the number of functions running on the worker and their duration, the Resource Monitor sets a higher or lower priority for the VM.

Other workloads. The extensions for these three frameworks can be implemented for workloads that do not leverage orchestrators. Even without the runtime extensions, some workloads can leverage deployment hints by tuning the way they are deployed. For example, the workload owner could specify that specific VMs can leverage scaling up and WI can apply corresponding optimizations such as overclocking.

5.2 Extending cloud platform optimizations

For cloud platform to onboard an optimization, we need to define (1) managed resources and the (2) priority compared to other optimizations. In addition, to track the benefits we also need to define the (3) workload owners benefit, (4) pricing, and (5) cost model. This is not difficult for new optimizations to get onboard, because all these features should have been defined by the optimizations themselves.

For existing cloud optimizations, Table 2 defines the resources and the pricing for each optimization, Table 4 specifies the priorities for each optimization, and Table 5 describes the hints that they need to consume and publish from and into WI. When applying the changes to the resources, the cloud optimizations leverage the coordination described in Section 4.4 which leverages priorities.

6 Evaluation

To evaluate WI, we first describe three use cases that demonstrate how three workloads can leverage multiple cloud platform optimizations. We choose these three workloads because they fall into the categories of big data analytics, web applications and real-time communication respectively, and together,

Optimization	Modifications for WI
Auto-scaling	Consume deployment *scale in/out* hints.
Spot VMs	Consume deployment *preemptible* hints.
	Consume runtime *preemption* priority.
	Publish runtime *preemption* notification.
Harvest VMs	Same as Spot VMs.
	Consume runtime *scale up/down* priority.
	Publish runtime *scale up/down* notification.
Overclocking	Consume deployment *scale up/down* hints.
	Consume runtime *scale up* priority.
	Publish runtime *scale up* notification.
Underclocking	Consume deployment *scale up/down* hints.
	Consume runtime *scale down* priority.
	Publish runtime *scale down* notification.
Pre-provision	Consume deployment *deployment time* hints.
RA placement	Consume deployment *locality* hints.
Oversubscription	Consume deployment *scale up/down* hints.
	Consume deployment *delay tolerance* hints.
	Consume runtime *scale down* priority.
VM rightsizing	Consume deployment *scale up/down* hints.
	Consume deployment *delay tolerance* hints.
MA DCs	Consume deployment *scale up/down* hints.
	Consume deployment *preemptible* hints.
	Publish runtime *scale down* notification.
	Publish runtime *preemption* notification.

Table 5: Extensions to existing cloud platform optimization.

these workload classes comprise 84% of the cloud cores usage at ProviderX. In addition, Table 6 shows that the workloads for our evaluation provide both good coverage and high diversity of hints. We then evaluate the potential for cost and carbon savings at a cloud provider scale.

6.1 Case study: Big data analytics

Methodology. We deploy our WI-aware Hadoop (§5.1) on a 20-node cluster composed of 5 VMs for the management components (*i.e.*, Resource Manager and NameNode) each with 4 vCPUs and 16 GB of memory and 15 VMs for the workers each with 8 vCPUs and 64 GB of memory. Table 6 summarizes the hints for deploying the worker node VMs.

We use a 5-day MapReduce workload trace from a production cluster at ProviderX in June 2020. For reproducibility, we scale the trace down to fit into a 20-node cluster by randomly down-sampling at a 2% rate. This scaled down trace comprises 100 jobs and lasts 5 hours. To emulate the characteristics of the original jobs, we assign the same job priorities to our synthetic MapReduce jobs as in the original trace.

For reproducibility, we will open-source the setup to emulate this experiment in ProviderX. This includes a "user-space" implementation of WI.

Operation. Using the WI deployment hints, the platform decides to enable Auto-Scaling, Spot VMs, and Harvest VMs for the VMs running the Hadoop workers. At runtime, each Hadoop worker posts hints to the local WI server with the runtime "preemptibility" hint for that VM every second (same
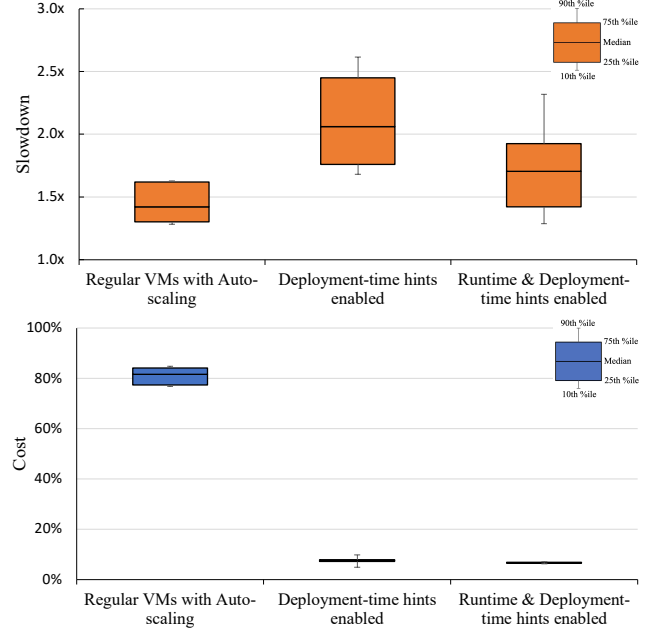

Figure 4: Slowdown and cost running the Big Data workload. The baseline (*i.e.*, 1×, 100%) is the median for Regular VMs.

as the default Apache YARN heartbeat interval [53]). To calculate this hint, the workload consider the number of containers, their uptimes, user-specified job priorities, and whether it hosts Application Masters. For example, if the worker VM has been running many critical jobs for a long time (*e.g.*, >30 seconds based on the typical cloud eviction notice time), it unmarks the runtime "preemptibility" hint to reduce the eviction probability and maximize the amount of resources assigned. WI uses these hints to determine which VMs to shrink and evict when reclaiming resources.

Results. Figure 4 compares the baseline setup, which is Regular VMs, to WI with deployment hints and WI using both deployment and runtime hints. We also include Regular VMs with auto-scaling for comparison. Regular VMs achieves the best performance because it constantly has access to all the resources. But they are also the most expensive option. We normalize our results based on the Regular VMs performance.

WI with deployment hints shows a median slowdown of 2.1× as it enables Auto-scaling and Harvest VMs. However, it significantly reduces the median cost by 92.6%. When enabling runtime hints, the median slowdown is reduced by 21.0%. Based on the pricing listed in Table 2, the cost is further reduced by 13.5% compared to only deployment hints.

The full WI setup achieves the lowest cost (93.5% cost reduction) while maintaining reasonable performance (1.7× slowdown for delay-tolerant workloads). This is because in addition to leveraging Spot and Harvest VMs, the platform communicates with the workload owner to minimize the penalty of evictions and to maximize the utility of harvested resources.

Case study	Workload characteristics of required hints						
	Scale up/down	Scale out/in	Deploy time	Availability	Preemptibility	Delay tolerance	Region independence
Big data analytics (§6.1)	✓	✓			✓	✓	
Microservices (management nodes) (§6.2)	✓			✓			
Microservices (worker nodes) (§6.2)	✓	✓	✓	✓	✓	✓	
Video conference (media-service VMs) (§6.3)	(✓)	✓	✓	✓	✓	✓	✓

Table 6: Overview of workload specified hints utilized by WI for our case studies. "✓" indicates hint required.

6.2 Case study: Microservices

Methodology. We deploy the social network workload from the DeathStarBench [22] on a Kubernetes [31, 39] cluster that runs our WI extensions (§5.1). Both control plane and worker VMs have 8 vCPUs and 32 GB of memory. We use two management VMs and a minimum of four worker VMs.

We extend the social network [22] to mark Load Balancer, Media Frontend, Memcached, MongoDB, and Redis to run in node pools with "management" requirements. The rest of the components (*i.e.*, Nginx and the logic like post composing, timeline management, etc.) run in the node pool with "worker" requirements. These worker components are load-balanced and replicated into multiple pods. Table 6 shows the hints specified for the VMs in each node pool.

We emulate the load using a scaled-down trace from a production cluster. We run this traffic generation from a separate set of VMs in the same virtual network. We will also open-source this setup.

Operation. Based on the WI hints, the platform enables oversubscription for the control VMs. Given the load, the platform oversubscribes CPU by 50% and memory by 20%.

For the workers, WI enables Auto-scaling, Harvest VMs, Overclocking, and MA DCs. At runtime, the workload sets the runtime "preemptibility" hint for all the VMs except one to reduce its probability of eviction. The workload also sets the runtime "scale up/down" hint to prefer harvesting and overclocking. As Kubernetes places more containers in a worker node, it removes the "preemptibility" hint. The pods with higher requirements (*e.g.*, Redis) are deployed in a node pool that cannot be evicted.

In addition, MA [61] tracks the first set for early throttling and the second one for eviction. In case of a power event, most components will be throttled and some workers evicted.

Results. The tail latency when running the workload in Regular VMs with autoscaling is 376 ms. In a setup with WI where we leverage overclocking and Harvest VMs, we lower the latency down to 332 ms, which is 13.3% improvement. Note that we do not observe latency spikes even during evictions.

The cost for the workload owner is reduced by 44% compared to the baseline with plain autoscaling. Most of the savings come from running with overclocking while the rest comes from running on Harvest VMs.

6.3 Case study: Video conference

Methodology. We setup a Video Conference workload on a WI-enabled cloud platform. We extend the existing deployment scripts to provide the hints in Table 6. This includes deploying VMs dedicated to media processing, responsible for voice and video handling. The load is balanced across VMs to efficiently manage calls. The client generator is deployed separately and replicates conference traces from a production environment involving approximately 50 to 100 audio calls with 4 to 50 users, along with 2 to 75 video calls that accommodate user counts ranging from 4 to 250. The load follows a daily pattern, with more calls during business hours. Additionally, there are load spikes at the beginning of the hour and the half-hour mark, aligning with the start of most meetings. For confidentiality requirements, we cannot open-source the code to run this experiment.

Operation. The cloud platform enables various optimizations for the media-service VMs: Auto-scaling, Overclocking, Pre-provisioning, VM rightsizing, and Region-agnostic.

The local WI manager monitors the hints, while each media-service VM tracks its usage and elevates its priority during high loads. The local overclocking controller determines whether to increase or decrease the CPU frequency of the VM based on this priority, while also factoring in the power budget and processor lifetime [28]. Moreover, the right-sizing manager uses utilization data to set the right VM type for the media-service VMs.

Results. We compare a default setup using regular VMs with the one with WI enabled. The WI-enabled setup is 26.3% more cost-effective by reducing the necessary VMs for off-peak periods. In addition, it reduces the carbon footprint by 51% by running VMs in a greener region.

Throughout the experiment, the workload sustains the required service level, and the conference processing rate (*i.e.*, the number of conference calls it can handle per second) is 35.4% higher with WI. This extra headroom indicates our conservatism, suggesting that we could have further reduced the number of VMs even further.

WI utilizes pre-provisioned VMs during peak time to achieve two primary objectives: firstly, it effectively reduces the latency associated with adding new media-service VMs, and secondly, it enhances overall performance. This improvement is marked by a 22% increase in conference process
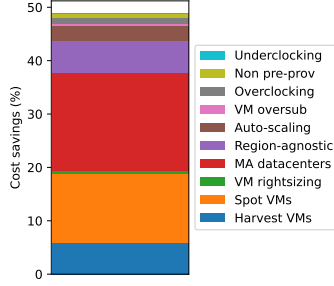
Figure 5: Cost savings breakdown by optimizations.

rates, coupled with a complete elimination of instances where conference processing encounters significant delays.

Furthermore, the WI Rightsizing Manager recommends a new VM size by monitoring utilization information from the WI Global Manager. This rightsizing reduces costs by 13.4%.

6.4 Benefits at provider scale

Based on the workload core usage from our survey (Table 1), the core percentage among optimizations (Table 3), data from the literature [4, 6, 13, 17], internal statistics for optimizations, and our case study experimental results (Table 2), we compute the savings for workload owners when using WI. To calculate the cost/carbon benefits, we need the joint probability distribution of core percentages among 10 optimizations. Given the complexity to compute these distributions precisely, we estimate it based on the joint distribution of up to 2 optimizations and significant (*i.e.*, >5%) multiple-optimization scenarios via Linear Programming [23]. We derive the cost savings for each optimization and summarize them in Table 2. We combine the results to compute the user benefit of enabling WI. As described in Section 4.4, we set priorities for the optimizations that cannot be enabled simultaneously due to operational conflicts based on discount values. Specifically, Spot VMs, Harvest VMs and Pre-provisioning do not simply provide multiplicative cost benefits when enabled at the same time due to their contention for spare compute resources. Similarly, Overclocking, Underclocking, and MA are not compatible with each other due to CPU frequency adjustments.

Cost impact. Figure 5 shows the cost savings breakdown from each optimization. We follow the decreasing order of the owner benefits which mimics the workload owners' preferences when choosing their optimizations manually.

WI reduces workload owner costs by 48.8% on average. MA DCs and Spot VMs offer substantial savings by 18.3% and 13.0% respectively. Region agnostic, Harvest VMs, Auto-scaling and Overclocking also provide considerable cost reductions by 6.0%, 5.8%, 2.8% and 1.3% individually at cloud scale. Note that actual savings vary based on owners' individual workloads, and other optimizations may play a more important role. For example, web proxy workloads benefit from Auto-scaling, Non pre-provisioning, Overclocking, Un-

derclocking and VM rightsizing predominantly due to its high availability requirement and the dynamic nature of web traffic.

Paradoxically, Figure 5 shows that a higher discount from an optimization does not necessarily translate to higher cost savings. For example, Harvest VMs have higher discounts than Spot VMs (91% vs. 85%) but contribute less to the overall savings. This is because Harvest VMs have more strict requirements for workloads and thus result in fewer applicable scenarios. Given workload characteristics, WI assist workload owners by automatically enabling the best set of optimizations that maximize their cost savings.

Carbon impact. We also account for the carbon generated. Region-agnostic sends some workloads to low-carbon regions (*e.g.*, the low 10th percentile) and reduce carbon footprint by 51% (*i.e.*, from 546 g/kWh to 267 g/kWh). Since VM rightsizing, Auto-scaling and VM oversubscription reduce the number of VMs required to run a workload, they also reduce carbon by 50%, 19% and 15% respectively. Overall, WI can reduce carbon at provider scale by 27.6%.

7 Related Work

User-provided information. Mesos [25] presents a management layer that enables fine-grained resource sharing across diverse cluster computing frameworks and lets the organizations specify their policies for resource sharing. In high-performance computing, many works propose users giving job characteristics, such as expected job run times, which are often found to be inaccurate [52]. We propose to design incentives that motivate users to provide better hints.

Attribute inference. The literature on inferring attributes relevant to cloud services broadly falls into ad-hoc resource management frameworks [24, 27, 29] and general frameworks that combine multiple resource managers [19, 49–51, 54, 58]. Borg [51, 54] estimates job characteristics (*e.g.*, resources needed, job priorities) to find suitable machines. Twine [50] is a framework that collaborates with applications to manage their lifecycle. Resource Central [19] collects VM telemetry, learns these behaviors online, and provides predictions to various resource managers. SOL [58] proposes a general on-node framework that considers fine-grained workload dynamics, resource utilization, and provides informed decisions using ML-based agents. To optimize carbon-efficiency, Ecovisor [49] virtualizes the energy system to applications.

These works infer the attributes for various resource managers via APIs or client-side libraries. However, all the workload hints may not be amenable for inference and these works also miss out on the relevant information from workload owners. There still exists a gap in safe and efficient ways to leverage the resource managers as well as the inputs from workload owners for optimizing cloud efficiency.

Other cloud interfaces. Container orchestration systems such as Kubernetes [31] and Docker Swarm [1] automate software deployment, scaling, and management, but they are unaware

of the workload characteristics running inside containers and rely on users to manually specify resource management policies. Serverless computing [47] provides more flexibility for the cloud platform to manage resources by allowing workload owners to upload code as tasks. However, the lack of explicit workload characteristics may lead to incorrect usage such as submitting tasks that are long-running or of low parallelism, which may result in even higher cost than using regular VMs.

8 Conclusion

In this paper, we have proposed Workload Intelligence (WI), a novel framework for dynamic bi-directional communication between cloud workloads and cloud platform. By punching holes through the current cloud abstraction, the platform can drastically simplify its offerings, reduce its costs without violating any workload requirements, and pass the savings to workload owners. We have evaluated the applicability and potential of WI across 10 cloud optimizations at a major cloud provider and demonstrated significant benefits for both providers and workload owners.

References

[1] Docker swarm mode overview, Aug 2023. https://docs.docker.com/engine/swarm/.

[2] Muhammad Abdullah Adnan, Ryo Sugihara, and Rajesh K Gupta. Energy efficient geographical load balancing via dynamic deferral of workload. In *CLOUD*, 2012.

[3] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *NSDI 20*, 2020.

[4] Amazon Elastic Compute Cloud. Amazon EC2 Spot Instances, 2019. https://aws.amazon.com/ec2/spot/.

[5] Amazon Elastic Compute Cloud. Running batch jobs at scale for less, 2020. https://aws.amazon.com/getting-started/hands-on/run-batch-jobs-at-scale-with-ec2-spot/.

[6] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, et al. Providing SLOs for Resource-Harvesting VMs in Cloud Platforms. In *OSDI 20*, 2020.

[7] Ari Liberman and Tyler Sanderson. Performance-driven dynamic resource management in E2 VMs, 2019. https://cloud.google.com/blog/products/compute/understanding-dynamic-resource-management-in-e2-vms.

[8] Microsoft Azure. Azure Functions. https://azure.microsoft.com/en-us/services/functions/.

[9] Microsoft Azure. Overview of autoscale in Microsoft Azure. https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-overview.

[10] Microsoft Azure. What are ARM templates? https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview.

[11] Microsoft Azure. Introducing B-Series, Our New Burstable VM Size, 2019. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/.

[12] Microsoft Azure. Use low-priority VMs with Batch, 2019. https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms.

[13] Microsoft Azure. Azure Spot Virtual Machines, 2020. https://azure.microsoft.com/en-us/pricing/spot.

[14] B. Beyer, C. Jones, J. Petoff, and N.R. Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

[15] Amazon Elastic Compute Cloud. Burstable Performance Instances, 2019. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html.

[16] Google Cloud. Google Cloud Deployment Manager documentation. https://cloud.google.com/deployment-manager/docs.

[17] Google Cloud. Preemptible VM Instances, 2020. https://cloud.google.com/compute/docs/instances/preemptible.

[18] Google Cloud. About VM metadata, 2022. https://cloud.google.com/compute/docs/metadata/overview.

[19] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *SOSP*, 2017.

[20] Sylvia Engdahl. New amazon ec2 spot pricing model: Simplified purchasing without bidding and fewer interruptions, 2008. https://aws.amazon.com/blogs/compute/new-amazon-ec2-spot-pricing/.

[21] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. Memory-harvesting VMs in Cloud Platforms. In *ASPLOS*, 2022.

[22] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *ASPLOS*, 2019.

[23] S.I. Gass. *Linear Programming: Methods and Applications*. Dover Books on Computer Science Series. Dover Publications, 2003.

[24] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, et al. Protean: Vm allocation service at scale. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 845–861, 2020.

[25] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[26] Rachel Householder, Scott Arnold, and Robert Green. On Cloud-based Oversubscription. *IJETT*, 2014.

[27] Syed M Iqbal, Haley Li, Shane Bergsma, Ivan Beschastnikh, and Alan J Hu. Cospot: a cooperative vm allocation framework for increased revenue from spot instances. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 540–556, 2022.

[28] 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 *ISCA*, 2021.

[29] Seyyed Ahmad Javadi, Amoghavarsha Suresh, Muhammad Wajahat, and Anshul Gandhi. Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments. In *Proceedings of the ACM symposium on cloud computing*, pages 272–285, 2019.

[30] Karpenter. Karpenter, 2023. https://karpenter.sh/docs/.

[31] Kubernetes. Production-Grade Container Orchestration, 2020. https://kubernetes.io/.

[32] Kubernetes. Scheduling, Preemption and Eviction, 2023. https://kubernetes.io/docs/concepts/scheduling-eviction/.

[33] Wes Lloyd, Shrideep Pallickara, Olaf David, Mazdak Arabi, and Ken Rojas. Mitigating Resource Contention and Heterogeneity in Public Clouds for Scientific Modeling Services. In *IC2E*, 2017.

[34] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *SC*. IEEE, 2011.

[35] Microsoft. Azure Instance Metadata Service, 2022. https://aka.ms/azureimds.

[36] Microsoft. Azure Kubernetes Service (AKS), 2022. https://docs.microsoft.com/en-us/azure/aks/.

[37] Microsoft. Hyper-V Integration Services, 2022. https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/integration-services.

[38] Microsoft. Monitor scheduled events for your Azure VMs, 2022. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/scheduled-event-service.

[39] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices. In *OSDI*, 2020.

[40] Benjamin Reidys, Jinghan Sun, Anirudh Badam, Shadi Noghabi, and Jian Huang. BlockFlex: Enabling Storage Harvesting with Software-Defined Flash in Modern Cloud Platforms. In *OSDI*, 2022.

[41] Amazon Web Services. Amazon EC2 Auto Scaling. https://aws.amazon.com/ec2/autoscaling/.

[42] Amazon Web Services. AWS CloudFormation Templates. https://aws.amazon.com/cloudformation/resources/templates/.

[43] Amazon Web Services. AWS Lambda. https://aws.amazon.com/lambda/.

[44] Amazon Web Services. Amazon Elastic Kubernetes Service (EKS), 2022. https://aws.amazon.com/eks/.

[45] Amazon Web Services. Instance metadata and user data, 2022. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html.

[46] Amazon Web Services. Scheduled events for your instances, 2022. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-instances-status-check_sched.html.

[47] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *ATC*, 2020.

[48] Jiuchen Shi, Kaihua Fu, Quan Chen, Changpeng Yang, Pengfei Huang, Mosong Zhou, Jieru Zhao, Chen Chen, and Minyi Guo. Characterizing and orchestrating VM reservation in geo-distributed clouds to improve the resource efficiency. In *SoCC*, 2022.

[49] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. Ecovisor: A virtual energy system for carbon-efficient applications. In *ASPLOS*, 2023.

[50] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, et al. Twine: A unified cluster management system for shared infrastructure. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 787–803, 2020.

[51] 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.

[52] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, jun 2007.

[53] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *SoCC*, 2013.

[54] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–17, 2015.

[55] Cheng Wang, Bhuvan Urgaonkar, Neda Nasiriani, and George Kesidis. Using Burstable Instances in the Public Cloud: Why, When, and How? 2017.

[56] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with Apache Kafka. VLDB, 2015.

[57] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. SmartHarvest: Harvesting idle CPUs safely and efficiently in the cloud. In *EuroSys*, 2021.

[58] Yawen Wang, Daniel Crankshaw, Neeraja J Yadwadkar, Daniel Berger, Christos Kozyrakis, and Ricardo Bianchini. SOL: Safe On-Node Learning in Cloud Platforms. In *ASPLOS*, 2022.

[59] Xen. XenStore, 2022. https://wiki.xenproject.org/wiki/XenStore.

[60] Randolph Yao, Chuan Luo, Bo Qiao, Qingwei Lin, Tri Tran, Gil Lapid Shafriri, Yingnong Dang, Raphael Ghelman, Pulak Goyal, Eli Cortez, Daud Howlader, Sushant Rewaskar, Murali Chintalapati, and Dongmei Zhang. Infusing ML into VM Provisioning in Cloud. In *CloudIntelligence*, 2021.

[61] Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, et al. Flex: High-availability datacenters with zero reserved power. In *ISCA*, 2021.

[62] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and Cheaper Serverless Computing on Harvested Resources. In *SOSP*, 2021.