Stratus: Clouds with Microarchitectural Resource Management

Kaveh Razavi*
ETH Zürich
kaveh@ethz.ch

Animesh Trivedi*
VU Amsterdam
a.trivedi@vu.nl

Abstract
The emerging next generation of cloud services like Granular and Serverless computing are pushing the boundaries of the current cloud infrastructure. In order to meet the performance objectives, researchers are now leveraging low-level microarchitectural resources in clouds. At the same time these resources are also a major source of security problems that can compromise the confidentiality and integrity of sensitive data in multi-tenant shared cloud infrastructures. The core of the problem is the lack of isolation due to the unsupervised sharing of microarchitectural resources across different performance and security boundaries. In this paper, we introduce Stratus clouds that treat the isolation on microarchitectural elements as the key design principle when allocating cloud resources. This isolation improves both performance and security, but at the cost of reducing resource utilization. Stratus captures this trade-off using a novel abstraction that we call isolation credit, and show how it can help both providers and tenants when allocating microarchitectural resources using Stratus’s declarative interface. We conclude by discussing the challenges of realizing Stratus clouds today.

1 Introduction
We are in the midst of a fundamental shift in cloud computing as researchers are pursuing the next-generation of services and infrastructure projects [12,81,97]. For example, cloud functions (FaaS, Serverless) enable developers to build bursty, highly-parallel and scalable applications [23,24,39]. Granular computing [51] proposes a new computing fabric consisting of large numbers (1k-1M) of small tasks at scale for a short burst of activity (1-10ms). Traditional monopolistic services are now being broken down into hundreds of microservices [27]. Overall, these next-generation services aim to push the performance of current clouds by another order of magnitude.

To meet such performance demands, our computing platforms have also evolved, taking advantage of emerging hardware in commodity computing. Devices such as GPUs [73], FPGAs [71,82], SmartNICs [22,41], and programmable storage [17,74] are now being used in cloud services and applications. Traditionally, these devices are managed by an operating system and a cloud-scale resource manager at the exposed architectural interfaces (e.g., number of cores, GPUs, DRAM, etc.). However, as the demand for high performance increases, the attention has gradually been shifting to reason about and even manage (directly or indirectly) microarchitectural resources as well [21]. As an example, contention on the last level cache (LLC), a microarchitectural resource, can lead to sub-optimal performance in high-speed networks when building distributed applications [20,94]. Similarly, mismanagement of various on-chip microarchitectural resources inside SmartNICs [18,40], or unintentional cross-talk inside Non-Volatile Memories (NVM) devices [11,45], can lead to significant performance degradations.

Performance, however, is not the only issue. Recent high-profile security attacks show that these microarchitectural resources can also be exploited by attackers to leak sensitive information [16,46,53,58,87,88,98] or inject faults in the data [72,93]. Broadly speaking, these attacks break hardware isolation boundaries in shared microarchitectural resources like DRAM, caches, and instruction execution units to compromise systems. Beyond the CPU, researchers have also demonstrated significant attacks on/using storage [49], FPGA [48], and GPU [25,103]. More worryingly, as seen recently, these attacks are also possible remotely over the network [50,85,86]. With the emergence of faster and diverse hardware, these issues will only worsen.

The core of the problem is the lack of isolation due to the unsupervised sharing of microarchitectural resources across different performance and security boundaries. The response from the community has been reactive: microarchitectural resources are currently managed in an ad hoc manner using a mix of techniques to improve either performance [13,40] or security [31,47,55], but never both. For example, Apache Crail,

*Equal contributions.
Table 1: Architectural and associated microarchitectural resources.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Microarchitectural</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>Caches, TLBs, Hyperthreads, ALUs</td>
</tr>
<tr>
<td>SmartNICs</td>
<td>Caches (memory, requests, connection), TLBs, RMT pipelines, DMA engines</td>
</tr>
<tr>
<td>NVM Storage</td>
<td>Blocks, pages, internal I/O ports, programmable cores, SRAM</td>
</tr>
<tr>
<td>CPUs</td>
<td>Memories, caches, execution units</td>
</tr>
<tr>
<td>Switches</td>
<td>SRAM and TCAM memories, Match-action Unit processors, ALUs</td>
</tr>
</tbody>
</table>

Threat model. We assume a generic threat model against microarchitectural attacks that do not require physical access to the target. Examples include cache attacks [66,88], speculative execution attacks [87,88] and Rowhammer [14,26,72,93]. These attacks often assume co-residency with a victim process or VM, but recent advances have also made them practical against servers over the network [50,79,85]. Stratus aims to address these attacks by providing abstractions that enable isolation on microarchitectural resources.

2 The Case for Stratus

There are three primary trends that are pushing for a more principled approach towards fine-grained microarchitectural resource reasoning. The first trend is increasing diversity. With the push for heterogeneous computing, a diverse set of ISAs, accelerators, switches, programmable storage and smart networking devices have entered into mainstream computing [32]. Naturally, these devices also bring associated diverse microarchitectural resources (see Table 1) into the shared cloud computing paradigm. The second trend is the push for multi-tenancy on modern devices. After having developed single-tenant applications, now modern devices (RDMA, NVMs, FPGAs) are being deployed in a shared, multi-tenant cloud setting [36,43,44,62,70]. Consequently, they require careful attention towards resource sharing, which can have unintended performance implications [13,20,102]. The last trend is the evolving security threats. As many of recent high-profile security attacks have demonstrated that an unsupervised or misguided sharing of microarchitectural resources can lead to information leaks, and full system compromises. Such attacks become possible because (a) there is a misplaced trust in the hardware to deliver safe sharing through isolation [33]; (b) a system software lacks any direct visibility to reason about the sharing and isolation at the microarchitectural level.

One could argue that it should be the operating system (OS) on each node that is tasked for the management of microarchitectural resources. While this argument holds for applications that run on a single node, cloud services such as replication [43,84], storage [96], machine learning [37], and even OSes [80] run on a number of different nodes with a variety of accelerator devices. Furthermore, in-networking resources such as SRAM and processing elements from programmable switches are now also being used for building datacenter services [38,70]. Hence, reasoning about security and performance properties in an end-to-end manner requires looking beyond end-points, and instead taking a more holistic and distributed approach towards microarchitectural resource management.

LLC Sharing - A Motivating Example In order to sustain data rates of high-performance networks, modern Intel CPUs directly place network data in its LLC [34] (Figure 1). This design immediately raises the question of how a remote LLC (i.e., a microarchitectural resource) should be managed to avoid cross-talk and maximize the performance of multiple competing tenants. Current technologies such as Intel DDIO,
simply share the LLC slice that is dedicated to I/O traffic among all clients. Such a default policy (with low isolation) not only delivers sub-optimal performance [20, 94], but more worryingly enables side-channel attacks over the network to leak information [50]. The core of both performance and security problems, in this example, is the unsupervised sharing of the LLC. Is it possible to share the remote LLC to improve performance while preserving security?

3 Design of Stratus

Stratus is a cloud framework that aims to capture and reason about microarchitectural isolation in a principled manner. The key insight in building Stratus is that security and performance are the two sides of isolation. This isolation is expressed by constraints which are predicates attached to resources that a provider must satisfy when allocating those resources. Currently these isolation constraints are hidden underneath various resource allocation strategies for a large spectrum of computing abstractions offered by cloud providers (e.g., functions, containers, VMs IaaS). One point on this spectrum is FaaS-like clouds, where tenants only provide “functions” (without any constraints) and providers are free to optimize their utilization objectives when allocating resources for the function execution [91]. Another extreme can be imagined as a cloud which exposes all of its resources and their status to its tenants, offering them full control over allocations. In the middle, there are IaaS clouds where tenants provide constraints such as the number of cores or the amount of memory encoded in the VM types that the cloud provider offers. Without limiting the current resource allocation strategies, Stratus aims to complement them with microarchitectural constraints to capture performance and security properties. In Stratus, microarchitectural constraints (e.g., isolated LLC cache) are in the majority of the cases decoupled from the more coarse-grained architectural resources (e.g., a core). This allows Stratus to support all existing resource allocation schemes while satisfying tenant-provided microarchitectural constraints.

We envision that tenants provide a set of microarchitectural-specific constraints to Stratus. Stratus then finds a set of suitable servers that can satisfy resource allocation constraints by querying a database that captures available microarchitectural resources. This database is, in essence, similar to the database of architectural resources in popular cloud infrastructures such as OpenNebula [60]. After choosing servers, Stratus uses available mechanisms on each machine to isolate microarchitectural resources for a given tenant. In exchange for these services, the provider can charge the tenant. The amount depends upon the balance between the effort required from the provider and the perceived value of satisfying the given constraints for the tenant. We capture this balance using a new abstraction called Isolation Credit.

In the following sections we show how Stratus captures isolation constraints (§3.1), introduce isolation credit (§3.2), discuss how Stratus evaluates a tenant’s constraints using our proposal of a cloud knowledge base (CKB) (§3.3), and uses existing mechanisms for enforcing isolation (§3.4). Figure 2 shows the overall interaction among these components in Stratus for principled microarchitectural resource allocation.

3.1 Capturing Tenants Requirements

Stratus allows tenants to express their isolation requirements in a declarative manner. An internal resource allocator uses constraint-logic programming (CLP) to analyze and satisfy the constraints. Constraints enable Stratus to reason in a principled manner if any isolation requirements are violated. Tenants specify isolation constraints using the following syntax:

Listing 1: Syntax of defining a constraint using ISOLATE

```
handle = ISOLATE(resource, scale, quantity);
```

A resource is a microarchitectural resource such as LLC, a NIC packet processor, etc. There are two types of microarchitectural resources, hard and soft. Hard resources are the ones that can be partitioned in space and used exclusively by a single tenant such as the LLC. Soft resources are contended in time, such as the DRAM bandwidth. The resources are modeled as they appear in the system topology where the top-level represents top-level architectural resources such as CPU, DRAM, or NIC. scale is a scalar quantity between \[0, 1\] capturing the extent of the isolation requested. A zero value, which is the default for all resources, indicates no isolation constraints from a tenant and the provider is free to optimize for the maximum utilization. Hard microarchitectural resources can only take discrete values of 0 or 1, whereas soft resources can take any value in between. quantity is the minimum number of requested resources for which the constraints must be satisfied. For example, a tenant might only be interested in the first 64 requested cache sets of LLC for network traffic, and not beyond that.

Isolating microarchitectural resources is alone not enough to provide end-to-end isolation. Thus, Stratus allows attaching isolated resources to each other using the ATTACH operator:

Listing 2: Combining (AND) constraints using ATTACH

```
ATTACH(handle1, handle2, ...);
```

This allows tenants of Stratus to properly isolate network clients of a DDIO-enabled server shown in Figure 1 using the following constraints:
when it comes to providing isolation. Strong isolation leads to better performance (e.g., 99.9th percentiles) and security, which are desired properties by tenants. In contrast, providers typically aim for high utilization by co-hosting tenants on shared infrastructure (minimum isolation) to maximize their profits. To capture this tension, we introduce the isolation credit, a currency that represents the amount of effort required from a provider to satisfy a tenant’s isolation constraints as well as the value derived by the tenant for their workloads.

The cost of providing isolation is not the same across different resources. For example, isolating an entire LLC cache may require other cores on the same processor socket not to be utilized by other tenants. A provider, hence, asks for a certain amount of isolation credits for satisfying an isolation constraint. Tenants can buy credits from their cloud provider and spend them on their isolation requests. The abstraction of isolation credit quantifies and monetizes the effort required for isolation. It forces a tenant to make sensible isolation requests (the ones that generate the maximum value), and pushes a provider to innovate in low-overhead isolation mechanisms.

### Spending isolation credits

Isolation credit can further be used as an abstraction for simplifying the low-level constraint interface of Stratus. For some tenants, microarchitectural constraints might be too low-level and detailed to enumerate. Instead, a tenant can simply provide Stratus with a credit budget that they are willing to spend, and Stratus will explore a strategy to simultaneously optimize performance, security, and utilization properties for the given budget. This strategy incentivizes cloud providers to find solutions with efficient isolation mechanisms to offer differentiating services. If another cloud provider offers a better way to charge less isolation credits, the tenant is tempted to run on the second provider.

#### 3.3 Evaluating Constraints

The overall goal of constraints evaluation is to allocate resources on machines that can satisfy all microarchitectural constraints specified by a tenant, while maximizing the utilization (or any other metric) for the cloud provider. To model information and solve constraint allocation, we take inspiration from the System Knowledge Base (SKB) component of the Barrelish operating system [7, 77] and the Barrelfish operating system [7, 77]. SKB is a service inside Barrelfish for storing and querying hardware knowledge to solve resource allocation constraints. We propose building a distributed version of SKB, called Cloud Knowledge Base (CKB) where we will gather data, model cloud resources, and query for allocations. CKB will manage data gathered from two primary sources. First, factual information from literature and manuals such as the number of TLB entries on a CPU, the number of DRAM banks, or the number of parallel processing units in a SmartNIC. The number of such resources defines how many fully isolated discrete allocations Stratus can do. Second, online measurements to monitor utilization, occupancy, latencies and bandwidths of interconnects, etc. This information is used for soft resource allocation.

Naturally, one key concern is the performance of the constraints evaluation as the system scales. For a VM, a few seconds for allocation time could be fine, but it is not acceptable to launch FaaS functions where allocations must be done in 10s of milliseconds. General constraint solving (SAT solving) to find a solution is a NP-hard problem. However, Stratus has to check satisfiability for a given set of possible allocation solutions. Satisfiability checks of Stratus’s constraints, which are in the CNF form, scale linearly with the number of constraints. For a given set of solutions (i.e. the list of servers), which can satisfy given constraints, a cloud provider can choose the one that maximizes its utilization or any other objective using existing mechanisms. Looking beyond server
resources, in-network resources on all involved switches must also be evaluated, if defined. For example, given a set of in-switch constraints, connections between servers need to be routed differently, or it may be necessary to migrate or refresh previous allocations. In all cases, the space exploration is bounded by the isolation credits provided by a tenant.

3.4 Building on Available Mechanisms

A research question that Stratus addresses is to what extent microarchitectural resources can be isolated between different tenants. Microarchitectural resources by definition are not directly exposed to tenants and there is no explicit API for managing them. However, there are mechanisms that can be built to indirectly ensure that microarchitectural resources are allocated and used under given constraints.

As for the CPU resources, there are mechanisms for isolating microarchitectural resources in the memory hierarchy. For example, LLC allocation and sharing is a well-studied problem [20] and there are mechanisms such as page coloring [99] or explicit partitioning [55] that can be used to satisfy Stratus’s isolation commands. Partitioning computational resources such as ALU ports inside a core is much more challenging and would require an entire core allocation for satisfying their isolation when needed [3, 9, 30].

Considering off-CPU devices, DRAM resources can be partitioned by the careful selection of memory pages [85]. SmartNICs (e.g., RDMA NICs) contain various on-NIC packet processing units (PUs), co-processors, connection/queue pairs (QPs) states, caches for work queue and memory translation entries, and DMA engines [40, 69]. A careful management of these resources is necessary to ensure high performance [40]. We expect that SmartNICs can either support resource isolation via state/session tracking (like in RDMA QPs), or hardware virtualization (PCI-e SRIOV RNICs), or software virtualization [44, 68]. Sharing of in-network computing resources is an active area of research, where there are very limited mechanisms for ensuring isolation between tenants [8].

For storage, Open-Channel SSDs can be used that expose the microarchitectural resources behind the block abstraction to a host for management [10, 11, 57] In such a design, a host becomes responsible for data placement (th, implicitly controlling the mapping of a location to die, plane, and parallel I/O ports), sharing (write buffers among tenants), and error handling. We believe Stratus can use these mechanisms to enforce isolation among multiple tenants [29].

4 Open Challenges

Realizing a Stratus cloud requires addressing a number of challenges, three of which we discuss here:

Picking the Right Isolation Constraints. Stratus tenants can either identify microarchitectural resources directly or use isolation credits as the mechanism for achieving isolation. In the former case, a tenant needs to know which microarchitectural resources require isolation, and in the latter case, this task is given to Stratus. For achieving security, tenants or Stratus can provide different isolation policies that mitigate different attacks (e.g., avoiding the execution of other tenants on sibling hardware threads mitigates certain speculative execution attacks [87, 88]). These policies can also be provided by third parties as a collection of open policy libraries that tenants can use. Building these security policies against known attacks will be the first such attempt and it remains to be seen whether the current interface of Stratus is expressive enough for such a task. For achieving improved performance, tenants can again directly ask for isolated microarchitectural resources or provide Stratus the freedom to use isolation credits for improving performance. We envision novel distributed profile-guided tools that enable the tenants or Stratus to reason about the benefits of isolating certain microarchitectural resources versus the accrued cost via isolation credits.

Scalable Allocation. Resource allocation/selection lies in the critical path for fast booting, scheduling, and execution of components that make cloud-scale services. For example, reducing the booting time (including resource acquisitions) of FaaS functions is an active research area [65]. Not just limiting to the latency, operating new computing frameworks like Granular Computing require starting 10s of thousands of small tasks in a few milliseconds [51]. Can Stratus evaluate a tenant’s constraints for all of these instances in a reasonable time, at scale? Furthermore, cloud providers may prefer to satisfy these isolation constraints next to other desired constraints such as increasing per-server utilizations. It remains to be seen whether these constraints from both providers and tenants can be solved in an efficient and scalable manner.

Enforcing Isolation. Stratus requires the possibility of isolating microarchitectural elements of any given architectural device that is shared between tenants. While this has proven to be possible for certain microarchitectural elements in CPU and DRAM, the rest – network, storage, in-network computing – is subject to research exploration and development of new hardware interfaces that allow the management of their microarchitectural resources when necessary. The mitigation of speculative execution attacks via the network may require the isolation of speculation effects which is currently a subject of active research [42, 78, 95]. Another challenge is developing novel abstractions that simplify the deployment of microarchitectural constraints. We envision Stratus to introduce microarchitectural resource containers akin to resource containers [6] that can be applied to a given tenant’s execution context. Building efficient support for such abstractions and verifying their execution (e.g., using attestation) at the operating system-level are other challenges that need to be addressed in Stratus.
5 Contributions to Workshop Discussion

Expected feedback and discussion points

- What are we missing from an operational point of view? Running cloud-scale services is a complex operation and allocating resources is one of the many steps taken in a long process. What are the implications of Stratus decision making on end-to-end operational properties such as fault tolerance, load balancing, etc.?
- Is isolation credit with a declarative interface the right abstraction for reasoning about microarchitectural resources? A declarative interface is a powerful and simple interface which has been used to manage resources [54, 100], explore configurations [63], manage heterogeneity [64, 92] and in networking [56, 76]. Furthermore, a previous study has shown that one-dimensional scalar quantities (similar to isolation credits) can be effective in communicating a tenant’s intention to its cloud provider [19]. Put together, we believe the abstractions we choose are powerful. We are, however, eager to hear counterarguments.
- What is the cost of building an efficient and scalable CKB? Constraint solving at scale, in a bounded time budget is a challenging problem. A recent work from Google shows that it is possible to build an efficient distributed system for solving graph reachability and membership evaluation problems for ACL management [67]. We take inspiration from such designs, but it remains to be seen what performance and scale CKB can deliver.
- In general, we are aiming to spark a discussion of how best to manage microarchitectural resources. Should we invest more in developing better policies and abstractions for the tenants to choose from or should we instead focus on building more expressive and fine-grained mechanisms?

Controversial questions

- Is microarchitectural resource management really worth it? In this paper we made a case for microarchitectural resource management in shared clouds. However, we understand that beyond technology, operational costs and complexities might put limits to the realization of this idea.
- Are hardware manufacturers willing to change hardware to provide better microarchitectural interfaces? CPU vendors already offer a limited form of mechanisms (Intel CAT, and cache invalidation instructions) to control microarchitectural resources. However, often policies are entangled with mechanisms [61]. Is there an opportunity here to identify the right interface for a variety of devices to expose their microarchitectural resources in a principled manner?
- What is a principled approach for a new ISA to include microarchitectural resource management? Moving forward, with a new ISA there is an opportunity to provide proper abstractions for microarchitectural resource management. There are many new trade-offs here: the interface can allocate resources on the fly with added hardware complexity, or the allocations can be reserved.

Acknowledgments

We thank our shepherd, Jon Howell, and the anonymous reviewers for their constructive comments. This work has been supported by NWO 016.Veni.192.262 and by Intel Corporation through the Side Channel Vulnerability ISRA.

References


[39] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-chie Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion...


