Research Paper Reading Group

Pilot Session 1
## Pilot overview

<table>
<thead>
<tr>
<th>Session 1</th>
<th>4/3/2021</th>
<th>A perspective on research papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session 2</td>
<td>5/4/2021</td>
<td>Identifying worthwhile papers</td>
</tr>
<tr>
<td>Session 3</td>
<td>6/1/2021</td>
<td>Discussing research papers</td>
</tr>
</tbody>
</table>
What is a typical research paper

For computer science and engineering, it is a documentation of novel technical contributions in...

Applications/Services  Systems
Types of research papers
Finding research papers

Google Scholar

arXiv.org

Computer Architecture and Automated Design Lab

As part of our commitment to open access to research, the workshop/conference papers, presentation slides, and videos are free and open to the public on the OSDI ’20 technical sessions page.
<table>
<thead>
<tr>
<th>Avg citations per paper</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 66.3</td>
<td>CSUR—ACM Computing Surveys</td>
</tr>
<tr>
<td>2. 53.5</td>
<td>SOSP—ACM Symposium on Operating Systems Principles</td>
</tr>
<tr>
<td>3. 52.8</td>
<td>OSDI—Operating Systems Design and Implementation</td>
</tr>
<tr>
<td>4. 43.4</td>
<td>NDSS—Network and Distributed System Security Symposium</td>
</tr>
<tr>
<td>5. 40.9</td>
<td>MobiHoc—Mobile Ad Hoc Networking and Computing</td>
</tr>
<tr>
<td>6. 40.3</td>
<td>SIGCOMM—ACM SIGCOMM Conference</td>
</tr>
<tr>
<td>7. 38.2</td>
<td>SenSys—Conference On Embedded Networked Sensor Systems</td>
</tr>
<tr>
<td>8. 36.7</td>
<td>MOBIHOC—Mobile Computing and Networking</td>
</tr>
<tr>
<td>9. 36.1</td>
<td>CIDR—Conference on Innovative Data Systems Research</td>
</tr>
<tr>
<td>10. 35.3</td>
<td>USENIX Security Symposium</td>
</tr>
<tr>
<td>11. 35.3</td>
<td>EUROCRYPT—Theory and Application of Cryptographic Techniques</td>
</tr>
<tr>
<td>12. 35.0</td>
<td>NDSS—Networked Systems Design and Implementation</td>
</tr>
<tr>
<td>13. 34.4</td>
<td>JASSS—The Journal of Artificial Societies and Social Simulation</td>
</tr>
<tr>
<td>14. 33.5</td>
<td>TOCS—ACM Transactions on Computer Systems</td>
</tr>
<tr>
<td>15. 33.5</td>
<td>S&amp;P—IEEE Symposium on Security and Privacy</td>
</tr>
<tr>
<td>16. 33.4</td>
<td>MobiSys—International Conference on Mobile Systems</td>
</tr>
<tr>
<td>17. 32.5</td>
<td>IJCV—International Journal of Computer Vision</td>
</tr>
<tr>
<td>18. 32.2</td>
<td>TOG—ACM Transactions on Graphics: SIGGRAPH</td>
</tr>
<tr>
<td>19. 31.6</td>
<td>VLDB—Very Large Databases</td>
</tr>
<tr>
<td>20. 30.9</td>
<td>BioMED—Biomedical Engineering</td>
</tr>
<tr>
<td>21. 30.9</td>
<td>IEEETRANS ROBOTICS AUTOMAT—IEEE Transactions on Robotics and Automation</td>
</tr>
<tr>
<td>22. 30.6</td>
<td>CRYPTO—International Cryptology Conference</td>
</tr>
<tr>
<td>23. 30.1</td>
<td>PAMI—IEEE Transactions on Pattern Analysis and Machine Intelligence</td>
</tr>
<tr>
<td>24. 29.6</td>
<td>PLDI—SIGPLAN Conference on Programming Language Design and Implementation</td>
</tr>
<tr>
<td>25. 29.3</td>
<td>MICRO—International Symposium on Microarchitecture</td>
</tr>
<tr>
<td>26. 29.1</td>
<td>Journal of Web Semantics</td>
</tr>
<tr>
<td>27. 28.5</td>
<td>BIB—Briefings on Bioinformatics</td>
</tr>
<tr>
<td>28. 27.4</td>
<td>JMLR—Journal of Machine Learning Research</td>
</tr>
<tr>
<td>29. 27.0</td>
<td>ISMB—Intelligent Systems in Molecular Biology</td>
</tr>
<tr>
<td>30. 26.8</td>
<td>PODS—Symposium on Principles of Database Systems</td>
</tr>
<tr>
<td>31. 26.5</td>
<td>VTC—Vehicular Technology Conference</td>
</tr>
<tr>
<td>32. 25.5</td>
<td>SIGMOD—International Conference on Management of Data</td>
</tr>
<tr>
<td>33. 24.6</td>
<td>STOC—ACM Symposium on Theory of Computing</td>
</tr>
<tr>
<td>34. 24.0</td>
<td>TOIS—ACM Transactions on Information Systems</td>
</tr>
<tr>
<td>35. 24.0</td>
<td>IEEE SAP—IEEE Transactions on Speech and Audio Processing</td>
</tr>
<tr>
<td>36. 23.9</td>
<td>SCA—Symposium on Computer Animation</td>
</tr>
<tr>
<td>37. 23.3</td>
<td>BIOINFORMATICS—Bioinformatics/computer Applications in The Biosciences</td>
</tr>
</tbody>
</table>
The peer review process

Typical paper format

Abstract
The paper's elevator pitch

Introduction
- "Apple pie"
- The specific problem(s)
- Proposed solution overview
- Specific contributions made
- Structure of the paper

Related Work
Proving that the problem exists and has not been solved yet

Method
Implementation details, design decisions and other nuances

Results
Proving that the problem was solved

Conclusion
Recap

Discussion
An analysis of the solution's impact, limitations etc.
The value of research papers
The ‘specific’ value of research papers

Technical contributions

Discussion

Introduction

Related Work

Method

Results

Potential future directions

Discussion

Introduction

Related Work

Method

Results
The ‘general’ value of research papers

Practical knowledge of the field

The scientific process
Case studies of general value: OSDI 2020

**Paper 1:** Scheduling track

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms
Pradeep Ambati, University of Massachusetts, Amherst; Iñigo Goiri, Felipe Fruejier, Microsoft Azure and Microsoft Research; Alper Gun and Ke Wang, Google; Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscbroda, Sameh Elmikety, Marcus Fontoura, and Ricardo Bianchini, Microsoft Azure and Microsoft Research
https://www.usenix.org/conference/osdi20/presentation/ambat

**Paper 2:** OS & Networking track

PANIC: A High-Performance Programmable NIC for Multi-tenant Networks
Jiaxin Lin, University of Wisconsin - Madison; Kiran Patel and Brent E. Stephens, University of Illinois at Chicago; Anirudh Sivaraman, New York University (NYU); Aditya Akella, University of Wisconsin - Madison
https://www.usenix.org/conference/osdi20/presentation/lin

**Paper 3:** Security track

Orchard: Differentially Private Analytics at Scale
Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce, University of Pennsylvania
https://www.usenix.org/conference/osdi20/presentation/roth
Paper 1: Scheduling Track

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms
Cloud providers usually rent their resources to customers as Infrastructure as a Service (IaaS) VMs. When deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years and may request more VMs over time.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Nadagha Arulale1, Diego Golez1, Ka Wang2, Pran Desai2, Brian Coroll2, Siddar Pande3, Thomas Moschitu4, Senthil Ezhilay4, Edike Bong3, Alpaatre Jayal3, Moufieda Friman4, Ricard Fustach5

Abstract

Cloud providers own the resources they allocate as exclusive virtual machines (VMs) to customers. In this paper, we show how the same resources can be made available to multiple customers through resource allocation. By doing so, we can provide SLOs for virtual machines (VMs), which is a key component of many resource allocation frameworks. We propose a new class of VM called Harvest VM, which provides for the exclusive use of resource allocation. Harvest VMs allow clients to specify their own resource allocation, ensuring that the VMs are allocated to the VMs that need them. This results in an efficient allocation of resources, as each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly deployed, each VM consumes a fixed amount of resources from the server where it lands. Customers can keep their VMs for seconds or years [16] and may request more VMs over time. Thus, providers need to provide the illusion of perfectly
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Pradip Ambate1, Bipra Garg2, Kui Wang1, Brian Dub1, Ryan Covell1, Sidney Pasquale1, Thomas Mendenhall3, Sanchit Tuli4, \textsuperscript{*}

Microsoft Azure

Abstract

Cloud providers must be transparent; they do not allocate or reserve virtual machines (VMs). As a result, in this paper, we investigate the use of scheduled instances in Microsoft Azure, and show that they are much better at supporting SLOs than the resource-based approach. In particular, we propose a new class of VM called Harvest VM, to cater and manage the supply and demand. A Harvest VM is more flexible and efficient because it can be scheduled to work in excess of demand during off-peak hours. This extends the existing elasticity to higher demand levels. Harvest VMs can be used to harvest on-demand resources in three different ways: First, to harvest the unused on-demand resources in the existing instance of a virtual machine (VM). Second, to harvest the unused on-demand resources in the existing instance of a virtual machine (VM). Third, to harvest the unused on-demand resources in the existing instance of a virtual machine (VM). To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

I Introduction

Motivation. Cloud providers wish to meet their customers' needs in a similar fashion as a service. This requires that providers can offer services that can be easily scaled up or down. However, traditional approaches to resource allocation are not well suited to meet these needs. For instance, they do not support non-functional requirements such as SLOs.

A Harvest VM is a virtual machine that is scheduled to work in excess of demand during off-peak hours. This extends the existing elasticity to higher demand levels. Harvest VMs can be used to harvest on-demand resources in three different ways:

1. To harvest the unused on-demand resources in Microsoft Azure. The specifications shown in Figure 1 compare the Harvest VMs to the traditional VMs in terms of price, availability, and performance.

2. To harvest the unused on-demand resources in the existing instance of a virtual machine (VM). Third, to harvest the unused on-demand resources in the existing instance of a virtual machine (VM). To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

3. To harvest the unused on-demand resources in the existing instance of a virtual machine (VM). Third, to harvest the unused on-demand resources in the existing instance of a virtual machine (VM). To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.

To illustrate the effectiveness, we present case studies and results from harvesting VMs and their framework in production.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Abstract

Cloud providers often use their resources to economize through automation as a service model (AaaS). When a cloud provider, such as a cloud computing service, cannot ensure that the resources are available when they are needed, it may cause a delay in service delivery. In this context, we propose a new plan of VM called Harvest VM, which is used to monitor and manage the available resources. A Harvest VM is more flexible and efficient than a regular VM because it provides a mechanism for collecting and analyzing resource usage data. The proposed framework for Harvest VM is implemented based on three main components: a microservice, a virtual machine (VM), and a cloud control plane. The microservice provides an abstraction layer for the VM, which can be used to monitor and adjust resource usage. The VM is responsible for collecting resource usage data from the underlying infrastructure. The cloud control plane manages the resources and ensures that they are available when needed. Using this framework, we can efficiently utilize the resources and achieve the desired level of performance.

I Introduction

Motivation: Cloud providers usually run their resources to economize through automation as a service model (AaaS). When a cloud provider, such as a cloud computing service, cannot ensure that the resources are available when they are needed, it may cause a delay in service delivery. In this context, we propose a new plan of VM called Harvest VM, which is used to monitor and manage the available resources. A Harvest VM is more flexible and efficient than a regular VM because it provides a mechanism for collecting and analyzing resource usage data. The proposed framework for Harvest VM is implemented based on three main components: a microservice, a virtual machine (VM), and a cloud control plane. The microservice provides an abstraction layer for the VM, which can be used to monitor and adjust resource usage. The VM is responsible for collecting resource usage data from the underlying infrastructure. The cloud control plane manages the resources and ensures that they are available when needed. Using this framework, we can efficiently utilize the resources and achieve the desired level of performance.

To measure the available capacity, providers of VMs with reserved SLOs at the cloud platform. In particular, they can use a simple model to reserve VMs that can be exploited for maximum resource efficiency. However, it is challenging to accurately estimate the available capacity of the VMs due to the dynamic nature of the cloud environment. In this paper, we propose a new plan of VM called Harvest VM, which is used to monitor and manage the available resources. A Harvest VM is more flexible and efficient than a regular VM because it provides a mechanism for collecting and analyzing resource usage data. The proposed framework for Harvest VM is implemented based on three main components: a microservice, a virtual machine (VM), and a cloud control plane. The microservice provides an abstraction layer for the VM, which can be used to monitor and adjust resource usage. The VM is responsible for collecting resource usage data from the underlying infrastructure. The cloud control plane manages the resources and ensures that they are available when needed. Using this framework, we can efficiently utilize the resources and achieve the desired level of performance.

II Technical Details

To measure the available capacity, providers of VMs with reserved SLOs at the cloud platform. In particular, they can use a simple model to reserve VMs that can be exploited for maximum resource efficiency. However, it is challenging to accurately estimate the available capacity of the VMs due to the dynamic nature of the cloud environment. In this paper, we propose a new plan of VM called Harvest VM, which is used to monitor and manage the available resources. A Harvest VM is more flexible and efficient than a regular VM because it provides a mechanism for collecting and analyzing resource usage data. The proposed framework for Harvest VM is implemented based on three main components: a microservice, a virtual machine (VM), and a cloud control plane. The microservice provides an abstraction layer for the VM, which can be used to monitor and adjust resource usage. The VM is responsible for collecting resource usage data from the underlying infrastructure. The cloud control plane manages the resources and ensures that they are available when needed. Using this framework, we can efficiently utilize the resources and achieve the desired level of performance.

For these reasons, they need to leave unallocated capacity.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

To monetize this unallocated capacity, providers offer VMs with relaxed SLOs at discounted prices. Specifically, they offer low-priority evictable VMs, often called spot VMs [1, 8, 14]. These VMs are evicted if their resources are needed by regular-priority (or simply regular) on-demand VMs.

Thus, evictable VMs are ideal for customers to run batch jobs or other workloads that can tolerate evictions, at low cost.

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.

We first characterize the unallocated resources in Microsoft Azure. The characteristics show there is potential for harvesting these resources further. Furthermore, our experimentation with Azure demonstrates that the cloud provider can harvest existing reservation resources over multiple time slots.

Next, we propose two ideas of scalable VMs, Harvest VMs, to avoid costly to maintain unallocated resources. A Harvest VM uses a traditional reservation in terms of physical resources, but dynamically reseats more of its physical resources beyond the reservation, depending on the amount of resources available for harvesting. The idea is that the Harvest VM is only used to harvest resources instead of regular VMs in the cloud, so it achieves high CPU use.

Performing experiments in open source VMs for Harvest VMs, we found that Harvest VMs are able to harvest resources for a certain period and have many resources, but may have low CPU use. However, if a customer wants to reserve 10 VMs, the SLAs may indicate that 90% of these resources are unused, which is an important consideration.

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

1 Introduction

Motivation. Cloud providers often over-provision resources to ensure that a service is available when needed. When deployed, each VM consumes a fixed amount of resources, and the provider must determine how much to allocate to ensure availability. Thus, providers often provide the full allocation of physical resources (e.g., by reserving dedicated physical nodes), while reserving the automation with software available (e.g., by incorporating hardware-level abstraction, dynamic resource management, or other workloads that can tolerate evictions, at low cost).

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.

1 Introduction

Motivation. Cloud providers often over-provision resources to ensure that a service is available when needed. When deployed, each VM consumes a fixed amount of resources, and the provider must determine how much to allocate to ensure availability. Thus, providers often provide the full allocation of physical resources (e.g., by reserving dedicated physical nodes), while reserving the automation with software available (e.g., by incorporating hardware-level abstraction, dynamic resource management, or other workloads that can tolerate evictions, at low cost).

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.

1 Introduction

Motivation. Cloud providers often over-provision resources to ensure that a service is available when needed. When deployed, each VM consumes a fixed amount of resources, and the provider must determine how much to allocate to ensure availability. Thus, providers often provide the full allocation of physical resources (e.g., by reserving dedicated physical nodes), while reserving the automation with software available (e.g., by incorporating hardware-level abstraction, dynamic resource management, or other workloads that can tolerate evictions, at low cost).

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.

1 Introduction

Motivation. Cloud providers often over-provision resources to ensure that a service is available when needed. When deployed, each VM consumes a fixed amount of resources, and the provider must determine how much to allocate to ensure availability. Thus, providers often provide the full allocation of physical resources (e.g., by reserving dedicated physical nodes), while reserving the automation with software available (e.g., by incorporating hardware-level abstraction, dynamic resource management, or other workloads that can tolerate evictions, at low cost).

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.

1 Introduction

Motivation. Cloud providers often over-provision resources to ensure that a service is available when needed. When deployed, each VM consumes a fixed amount of resources, and the provider must determine how much to allocate to ensure availability. Thus, providers often provide the full allocation of physical resources (e.g., by reserving dedicated physical nodes), while reserving the automation with software available (e.g., by incorporating hardware-level abstraction, dynamic resource management, or other workloads that can tolerate evictions, at low cost).

On these reservations of existing evictable VMs, we argue that these should be the first class of scalable VMs that are dynamically and automatically the resources of choice as new VMs are launched with the cost savings and benefits of cloud computing.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

P. Anvari, S. González, K. Wang, B. Liu, B. Cruz, S. Pasquale, T. Menezes, S. Eldersveld, E. Petriu, A. González, M. Fuentes, R. Bianchini

Abstract

Cloud providers are the entities that deliver services on demand via virtual machines (VMs). The primary motivations for cloud providers to offer SLOs for VMs are increased customer satisfaction and increased revenue. However, ensuring SLAs for VMs in cloud platforms is not straightforward. The key reasons are the dynamic nature of cloud environments and the need for efficient resource management. In this paper, we present a novel approach for providing SLOs for VMs in cloud platforms. We propose a new class of VMs, called "resource-harvesting VMs" (RH-VMs), that are designed to efficiently manage resource requests. The RH-VMs are equipped with a set of additional capabilities, such as dynamic resource allocation, real-time resource monitoring, and adaptive resource management. This allows cloud providers to efficiently manage resource requests while ensuring SLOs for VMs are met. The proposed approach is demonstrated through a case study, showing the effectiveness of the RH-VMs in meeting SLOs for VMs in cloud environments. The results demonstrate that the proposed approach can significantly improve the efficiency of resource management in cloud platforms, thereby enabling cloud providers to offer reliable and cost-effective services to their customers.

1 Introduction

Motivation: Cloud providers usually offer resources to consumers as on-demand cloud services. When deployed, each VM consumes a fixed amount of resources proportional to its size. This can lead to over-provisioning in cloud environments, resulting in inefficient resource utilization. Therefore, there is a need for efficient resource management to ensure that resources are allocated and released in a timely manner. This requires a holistic approach that takes into account the dynamic nature of cloud environments and the need for efficient resource management.

1. Use cases: clouds offering services on-demand, cloud resource management, cloud resource allocation, cloud resource monitoring, cloud resource optimization, cloud resource management in dynamic environments.
...an evictable VM cannot consume all the unallocated resources of a server unless it fits perfectly in it.

...a large evictable VM will be promptly evicted whenever even a single resource is needed by a newly arriving regular VM.

Multiple small evictable VMs can allocate the same amount of resources but will add overhead to operate more VMs.

...larger number of evictions introduce VM re-creation and application re-initialization overheads that may even cause unavailability.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Given these limitations of existing evictable VMs, we argue that there should be a new class of evictable VMs able to dynamically and flexibly harvest all the unallocated resources of any server on which they land.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Background and related work

- VM deployments are partitioned into geographical regions and regions are partitioned into clusters of servers. Servers in a cluster have the same hardware but each region may have different number of clusters and hardware mix.

- There is a separator for region-level and cluster-level. Examples of scheduling factors: hardware required, maintenance tasks, available mix.

- A server-level agent creates the VM and manages its lifecycle.

- Excess capacity is typically sold at discounted prices as evictable VMs. Eviction notice varies between providers – AWS gives a 2 min warning, 30s for Azure.

- Dynamically changing virtual resources of a VM and enabling scheduler support for this can be unrealistic in practice. Simplicity and maintainability is important for production deployment.

- Potential future work: harvest any allocated cores that are temporarily idle.

- Traces of AWS EC2 spot prices are publicly available and can be used to model the availability of spot instances – the challenge is the degree of accuracy and comprehensiveness.

- It is better (and possible) to quantify unallocated resources at the granularity of a server, rather than aggregate data for the entire cluster.
Providing SLOs for Resource-Harvesting VMs in Cloud Platforms

Characterizing unallocated resources: Azure 2/19 - 10/19 data
- Methodology, Temporal patterns, Cluster behaviors, Regional aggregated data, Minimum unallocated cores, Additional unallocated cores, Multiple VMs per server, High-level takeaways

Proposed VM class: Harvest VM
- Overview, Production implementation approach, Comparison to standard evictable VMs, Workload/application requirements, Privacy/Confidentiality, Pricing, Harvesting resources other than cores

Prediction for survival rate: ML based approach
- User input, ML models and features, ML training and inference, Discarded features, Applying prediction to standard evictable VM survivability

Scheduler support: Harvest Hadoop
- Architecture, Eviction management, Core reassignment management, Harvesting resources other than cores

Evaluation
- Evaluation focus, Simulator, Experiments, Analysis (benefits, accuracy, scheduler, cost)
Paper 2: OS & Networking Track

PANIC: A High-Performance Programmable NIC for Multi-tenant Networks
PANIC: A High-Performance Programmable NIC for Multi-tenant Networks

Jinlin Lin, Kiran Patel, Brent E. Stephens, Anirudh Sivarangan, Aditya Akella
University of Wisconsin-Madison, University of Illinois at Chicago, University of Wisconsin-Madison

A. Artifacts Appendix

A.1 Abstract

This artifact contains the source code and test benches for PANIC’s 100Gbps FPGA-based prototype. Our FPGA prototype is implemented in pure Verilog. Features of the prototype include: the hybrid packet/pipeline scheduler, the high-performance switching interconnect, self-contained compute units, and the lightweight KMT pipeline.

This artifact provides two test benches to reproduce the results in Figure 8c and 11a in the Verilog/Modelsim simulator.

A.2 Artifact check-list

- Hardware: This artifact does not require any specific hardware.
- Memory: This artifact measures PANIC consuming throughputs under different chaining models and traffic patterns.
- Output: The result will be printed to the console and log files.
- Experiments: This artifact includes testbenches and running scripts to replay Figures 8c and 11a.
- Public link: https://bitbucket.org/or-w-ma-lon-networking-research/panic_artifact/src/master/ README.md

A.3 Description

A.3.1 How to access

This artifact is publicly available at https://bitbucket.org/or-w-ma-lon-networking-research/panic_artifact/src/master/.

A.3.2 Software dependencies

Running this artifact requires Vivado [10]. Vivado 2019.4 WP101K version is license-free, and it has simulation capabilities to execute our results. Since installing the Vivado WP101K requires plenty of disk space (>20GB), you may choose to instantiate the FPGA Developer AEM in R; https://www.openroad.io/workbench/pcs.Device.vivado/ R; https://github.com/pci-newsroom to run this artifact. The FPGA Developer AEM has removed the required Vivado workload.

A.4 Experiment workflow

1. Check Vivado is Installed Correctly

   $ vivado --mode tcl
   // Enter the Vivado Command Prompt
   Vivado 2019.4 and 2020.1 is verified
   Vivado quit

2. Close the Reps and Make Reps

   $ git clean [Artifact_Reps]
   $ cd panic_core20_artifact
   $ make test-parallel
   $ make test-shared

   The make command first compiles the source code, then runs the simulation tasks in Vivado. The test-parallel test employs Figures 8c and the test-shared test employs Figures 11a.

A.5 Evaluation and expected result

The results will be printed to the console. The output will also be logged in Aditya_Adm_stack_output.log. For the expected output and analysis please refer to Figures 8c and 11a.

A.6 Notes

For more details about the code structure, please refer to https://bitbucket.org/or-w-ma-lon-networking-research/panic_artifact/src/master/ README.md

A.7 AE Methodology

Submission, reviewing and grading methodology:

"Apple pie"

- The gap between network line-rates and the rate at which a CPU can produce and consume data is widening rapidly.
- Emerging programmable ("smart") NICs can help overcome this problem.
- There are many different types of offloads that can be implemented on a programmable NIC.
- These offloads, which accelerate computation across all of the different layers of the network stack, can reduce load on the general purpose CPU, reduce latency, and increase throughput.
- Many different cloud and datacenter applications and use cases have been shown to benefit from offloading computation to programmable NICs.

PANIC: A High-Performance Programmable NIC for Multi-tenant Networks

Abstract

Programmable NICs have diverse uses, and there is a need for a NIC platform that can offload computation from multiple co-resident applications running different types of workload, including hardware accelerators, embedded NICs, and serverless function executions. Unfortunately, there is no existing NIC that can provide the flexibility required to offload such workloads. 1
designed to support multiple independent, high-performance, low-latency, flexible, software-based, and secure

1 This paper presents PANIC, a new programmable NIC. There are in common, the requirements for the NICs, i.e., (1) a high performance/multi-threading interface that is scalable to very wide吞吐量 ranges; (2) a high throughput interface that allows efficient communication between the CPU and the NIC; (3) a flexible and powerful software interface that allows applications to easily and efficiently offload workloads; and (4) a low-latency interface that allows for fast communication between the CPU and the NIC. Our work on PANIC, which combines several hardware and software technologies, provides a new platform for offloading computation to programmable NICs.

1 Introduction

The gap between network line rates and the rate at which a CPU can produce and consume data is widening rapidly. Today, emerging programmable NICs can help overcome this problem. They can implement various types of offloads, including hardware accelerators, embedded NICs, and serverless function executions. Unfortunately, there is no existing NIC that can provide the flexibility required to offload such workloads. This paper presents PANIC, a new programmable NIC that combines several hardware and software technologies to provide a new platform for offloading computation to programmable NICs.
Problems/Challenges

- No existing programmable NIC that supports all of the following properties:
  - Offload variety
    - Some offloads like cryptography are best suited for hardware implementations, while an offload providing a low-latency bypass for RPCs in an application is better suited for an embedded core
  - Offload chaining
    - To minimize wasted chip area on redundant functions, the NIC should facilitate composing independent hardware offload units into a chain as needed, with commonly-needed offloads shared across tenants
  - Multi-tenant isolation
    - Tenants should not be able to consume more than their allocation of a shared offload
  - Variable-performance offloads
    - There are useful offloads that are not guaranteed to run at line-rate, as well as important offloads that run with low latency and at line-rate.

PANIC: A High-Performance Programmable NIC for Multi-tenant Networks
Problems/Challenges

- Existing programmable NIC designs, categorized below, have key limitations:
  - **Pipeline-of-Offloads (ASIC + FPGA)**
    - Modifying chaining requires significant amount of time and developer effort for FPGA synthesis
    - Slow offloads cause packet loss or head-of-line (HOL) blocking
  - **Manycore NICs (CPUs)**
    - CPU cores add tens of microseconds of additional latency
    - No performant mechanisms today to isolate competing tenants
    - Performance degrades significantly if working set does not fit in non-volatile memory
  - **RMT NICs (programmable ASIC)**
    - Limited offload support
    - Each pipeline stage must be able to handle a new packet every single clock cycle
Background and related work (Sections 2 and 8)

- NICs should support both hardware and software offloads since not all offloads are best implemented on the same type of underlying engine. For example, crypto offload works better using hardware accelerators while walking a hash table resident in main memory is better suited for embedded cores.

- Applications, and even individual packets, can have different requirements. Secure remote memory access may require: crypto + congestion control + RDMA offload blocks. Key value store - that serves requests both from within data center and WAN distributed clients - can require an IPSec and/or compression offloads, but only WAN packets will likely use them.

- Some offloads may not run at line-rate. Of the compression, cryptography, authentication, and inference offloads that we ran on hardware, only inference was able to run at 100 Gbps. Compression and authentication performance depends on packet size. Slow offloads can be duplicated across multiple engines (e.g., 3 AES-256 engines) for line-rate operation.

- An offload that is used for TX and RX on a dual port NIC needs to operate at four times line-rate to prevent becoming a bottleneck.

---

### Table 2: Programmable NIC designs compared w.r.t. the requirements in Section 2.1.

<table>
<thead>
<tr>
<th>NIC Design</th>
<th>Offload Chaining</th>
<th>Multi-Tenant Isolation</th>
<th>Variable Perf</th>
<th>High Perf</th>
<th>Offload Variety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Manycore</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RMT</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

This paper proposes a design implementation called "PANIC," which is the outcome of extensive experimentation involving NIC designs. PANIC aims to operationally enable hardware-software co-design at an unprecedented scale. The PANIC design leverages these key principles:

1. Offloads should not be constrained: NICs should use non-invasive methods for selecting NIC designs. PANIC aims to be operationally enabled with only simple command-line tools.
2. Applications, and even individual packets, can have different requirements. Secure remote memory access may require: crypto + congestion control + RDMA offload blocks. Key value store - that serves requests both from within data center and WAN distributed clients - can require an IPSec and/or compression offloads, but only WAN packets will likely use them.
3. Some offloads may not run at line-rate. Of the compression, cryptography, authentication, and inference offloads that we ran on hardware, only inference was able to run at 100 Gbps. Compression and authentication performance depends on packet size. Slow offloads can be duplicated across multiple engines (e.g., 3 AES-256 engines) for line-rate operation.
4. An offload that is used for TX and RX on a dual port NIC needs to operate at four times line-rate to prevent becoming a bottleneck.
# PANIC: A High-Performance Programmable NIC for Multi-tenant Networks

## Architecture overview
- Operational overview, Offload variety support, Dynamic offload chaining support, Policies for dynamic multi-tenant isolation, Support for offloads with variable and below line-rate performance, Support for high performance

## Design of individual components
- RMT pipelines, High performance interconnect, Centralized scheduler, Compute unit

## ASIC analysis
- RMT, PIFO parser, Interconnect, Compute units

## FPGA Prototype
- RMT pipelines, FPGA-based crossbar, Central scheduler and packet buffer, Compute units

## Evaluation
- Testbed and methodology, Microbenchmarks, Comparison with the pipeline design, RISCV core performance, Hardware resource usage, End-to-end performance
Paper 3: Security Track

Orchard: Differentially Private Analytics at Scale
"Apple pie"

- When operating a large distributed system, it is often useful to collect some data from the users’ devices—e.g., to train models that will help to improve the system.
- Since this data is often sensitive, differential privacy is an attractive choice, and several deployed systems are using it today to protect the privacy of their users.
  - Google is using differential privacy to monitor the Chrome web browser.
  - Apple is using it in iOS and macOS, e.g., to train its models for predictive typing and to identify apps with high energy or memory usage.
  - Other deployments include those at Microsoft and at Snap.
- Today, this data is typically collected using local differential privacy.
  - Each user device individually adds some random noise to its own data.
  - Then each user uploads the data to a central entity.
  - The central entity then aggregates the uploads and delivers the final result.
- Local differential privacy can be done efficiently at scale.
Problems/Challenges

- The final result of local differential privacy contains an enormous amount of noise.
  - Even in a deployment with a billion users, it is easy to miss signals from a million users.
  - Reducing noise weakens privacy guarantee considerably.

- Global differential privacy can address this since noise is only added once i.e. by the aggregator.

- However, global differential privacy requires a lot more trust in the aggregator since individual users have to send raw data and trust that the aggregator will not look at it.

- Crypto techniques like Multi Party Computation and Fully Homomorphic Encryption can avoid the untrusted aggregator problem, but do not scale to millions of participants with current technology.

- Systems like Honeycrisp use additively homomorphic encryption which is much more efficient at scaling, but can only answer the count-mean sketches query.
Orchard: Differentially Private Analytics at Scale

**Background and related work (Section 2, 8)**

- **Important goals for a differential privacy system:**
  - **Privacy**
    - The amount of information that either the aggregator or other users can learn about the private data of an honest user should be bounded, according to the formulation of differential privacy.
  - **Correctness**
    - If all users are honest, the answers to queries should be drawn from a distribution that is centered on the correct answer and has a known shape.
  - **Robustness**
    - Malicious users should not be able to significantly distort the answers.
  - **Efficiency**
    - Most users should not need to contribute more than a few MB of bandwidth and a few seconds of computation time per query.

---

**Privacy**

- The amount of information that either the aggregator or other users can learn about the private data of an honest user should be bounded, according to the formulation of differential privacy.

- **Correctness**

- **Robustness**

- **Efficiency**

---

**Overview**

1. **Scenarios**

   - We consider a scenario illustrated in Figure 1, with a very large number of users (millions), each holding a few hundred or thousand numbers. Other experiments, also not shown, confirm that the privacy and query accuracy of Orchard are not substantially affected by increasing the number of users.

   - **Threat model:** We make the standard assumptions that (1) users are honest and (2) the client only sees the output of the queries, not the raw data.

   - **Results:** The privacy and query accuracy of Orchard are not substantially affected by increasing the number of users.

---

**Differential privacy**

- Differential privacy (DP) is a property of algorithms that (1) describes how output depends on an input and (2) guarantees that the output does not reveal too much information about the input.

   - In essence, the output is a function of the input, but the function is designed such that it is insensitive to any single input value.

   - **Mathematically:** Let \( D \) be the set of all possible inputs, \( O \) be the set of all possible outputs, and \( \delta \) be the maximum amount of information that can be learned from the output.

   - **Definition:** An algorithm \( A \) satisfies \( \varepsilon \)-differential privacy if for all subsets \( S \subseteq D \) and all outputs \( O \) in \( O \):

     \[
     
     \Pr[A(D) \in O] \leq e^{\varepsilon} \Pr[A(D') \in O] + \delta
     
     \]

   - **Intuition:** The privacy guarantee states that the probability of outputting any particular output is not increased or decreased by the presence or absence of any single input value.

---

**Conclusion**

- In conclusion, Orchard is a scalable and robust differentially private analytics system that achieves high privacy guarantees and low query response times.

---

**References**


---

**Figure 1:** Orchard: Differentially Private Analytics at Scale
Differential privacy is a property of randomized queries that take a database as input and return an aggregate output. Informally, a query is differentially private if changing any single row in the input database results in "almost no change" in the output.

**Background and related work (Section 2, 8)**

- **Differential privacy** is a property of randomized queries that take a database as input and return an aggregate output. Informally, a query is differentially private if changing any single row in the input database results in "almost no change" in the output.

- **A standard method for achieving differential privacy for numeric queries** is the Laplace mechanism, which involves two steps:
  - calculating the sensitivity, $s$, of the query which is how much the un-noised output can change based on a change to a single row
  - adding noise drawn from a Laplace distribution with scale parameter $s/\varepsilon$; this results in $\varepsilon$-differential privacy.

- For queries with discrete values, the standard method is exponential mechanism which is based on:
  - A "quality score" that measures how well a value ‘x’ represents a database ‘d’
  - The sensitivity of the quality score.

- Differential privacy is compositional - if we evaluate two queries which are $\varepsilon_1$ and $\varepsilon_2$ differentially private, then publishing results from both queries is at most $(\varepsilon_1 + \varepsilon_2)$ differentially private.

- We can define a privacy budget ($\varepsilon_{max}$) that corresponds to the maximum acceptable privacy loss.
  - The $\varepsilon$ for each query is deducted from this budget till it is exhausted.
Orchard: Differentially Private Analytics at Scale

Programming language selection: Fuzz
- Running example: k-means, Language features, Alternative languages

Transform centralized Fuzz queries to support distributed execution
- Program zones, The bmcs operator. Extracting dependencies, Transformation to bmcs form, Optimizations, Limitations

Distributed query execution
- Overall workflow, Security: Aggregator, Security: Malicious clients, Handling churn

Implementation
- Encryption, MPC, Secret sharing, Verifiable computation, Security parameters

Evaluation
- Coverage, Optimizations, Robustness to malicious users, Experimental setup, Cost for normal participants, Cost for the committee, Cost for the aggregator
Join us for the next session!

<table>
<thead>
<tr>
<th>Session 1</th>
<th>4/3/2021</th>
<th>A perspective on research papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session 2</td>
<td>5/4/2021</td>
<td>Identifying worthwhile papers</td>
</tr>
<tr>
<td>Session 3</td>
<td>6/1/2021</td>
<td>Discussing research papers</td>
</tr>
</tbody>
</table>
Sign-up / Comments / Suggestions / Feedback

https://forms.gle/6Y2ZBH2Bq2y5Qmie7
Thank you!