Hotspot Mitigation for Virtual Machine Migration

Dr. Rajiv Misra

Associate Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss ***hot spot mitigation techniques*** to automate the task of monitoring and detecting hotspots by determining a new mapping of physical to virtual resources and initiating the necessary migrations.

- We will discuss a ***black-box approach*** that is fully OS- and application-agnostic and a ***gray-box approach*** that exploits OS- and application-level statistics.

Enterprise Data Centers

- **Data Centers are composed of:**
  - Large clusters of servers
  - Network attached storage devices
- **Multiple applications per server**
  - Shared hosting environment
  - Multi-tier, may span multiple servers
- **Allocates resources to meet Service Level Agreements (SLAs)**
- **Virtualization increasingly common**

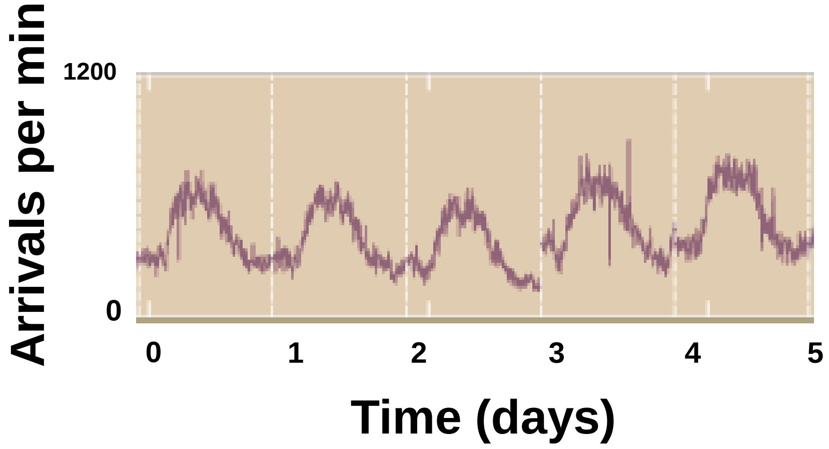
Benefits of Virtualization

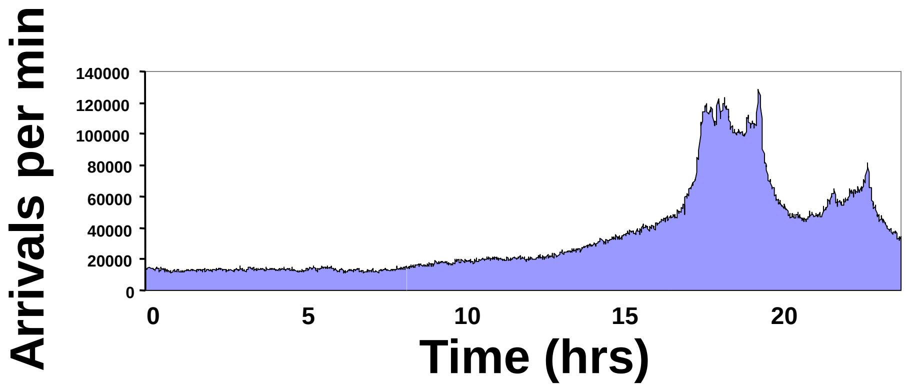
- **Run multiple applications on one server**
  - Each application runs in its own virtual machine
- **Maintains isolation**
  - Provides security
- **Rapidly adjust resource allocations**
  - CPU priority, memory allocation
- **VM migration**
  - "Transparent" to application
  - No downtime, but incurs overhead

How can we use virtualization to more efficiently utilize data center resources?

Data Center Workloads

- Web applications, enterprise systems, e-commerce sites etc see dynamic workloads
- dynamic workload fluctuations: caused by incremental growth, time-of-day effects, and flash crowds etc





How can we provision resources to meet these changing demands while meeting SLAs is a complex task**?**

Provisioning Methods

- **Hotspots** form if resource demand exceeds provisioned capacity

- **Over-provisioning**
  - Allocate for peak load
    - Wastes resources
    - Not suitable for dynamic workloads
    - Difficult to predict peak resource requirements

- **Dynamic provisioning**
  - Adjust based on workload
    - Often done manually
    - Becoming easier with virtualization

Problem Statement

How can we automatically (i) monitoring for resource usage,(ii) hotspot detection, and (iii) mitigation ie determining a new mapping and initiating the necessary migrations (ie **detect and mitigate Hotspots) in virtualized data centers?**

Hotspot Mitigation Problem

- Once a hotspot has been detected and new allocations have been determined for overloaded VMs, the migration manager invokes its hotspot mitigation algorithm.
- This algorithm determines which virtual servers to migrate and where in order to dissipate the hotspot.
- **Determining a new mapping of VMs to physical servers that avoids threshold violations is NP-hard—the multidimensional bin packing problem can be reduced to this problem, where each physical server is a bin with dimensions corresponding to its resource constraints and each VM is an object that needs to be packed with size equal to its resource requirements.**
- Even the problem of determining if a valid packing exists is NP-hard.

Research Challenges

- **Hotspot Mitigation:** automatically detect and mitigate hotspots through virtual machine migration
- ***When*** to migrate?
- ***Where*** to move to?
- ***How much*** of each resource to allocate?
- **How much information needed to make decisions?**

Sandpiper

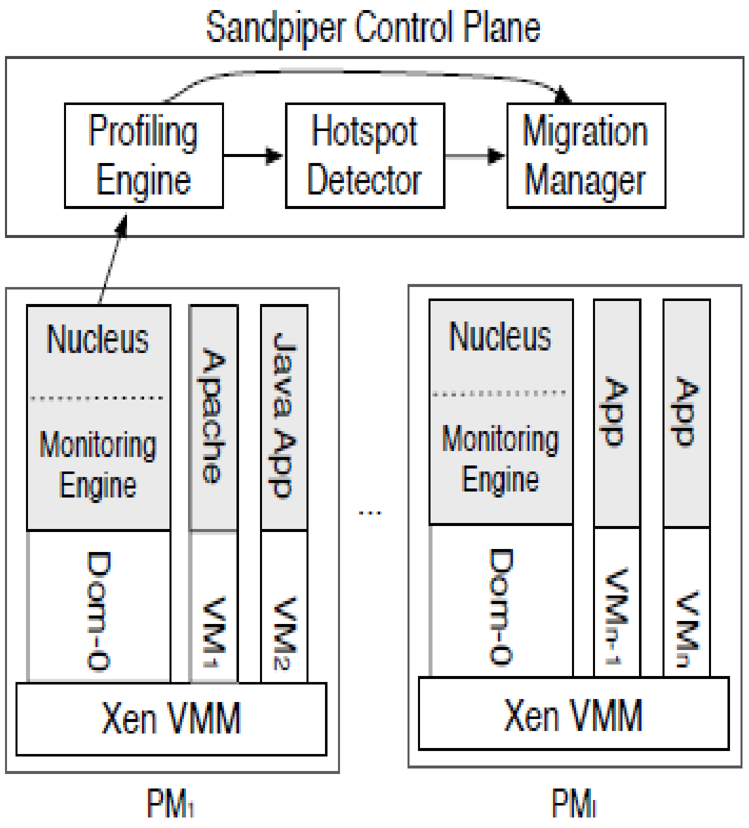
A migratory bird

Background

- **Dynamic replication**:
  - Dynamic provisioning approaches are focused on dynamic replication, where the number of servers allocated to an application is varied.
- **Dynamic slicing**:
  - In dynamic slicing, the fraction of a server allocated to an application is varied.
- **Application migration:**
  - In the virtualization, VM migration is performed for dynamic provisioning
  - Migration is transparent to applications executing within virtual machines

Sandipiper Architecture

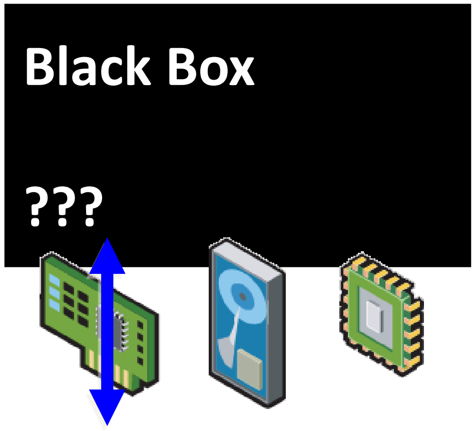
- **Nucleus**
  - *Monitor* resources
  - Report to control plane
  - One per server
  - Runs inside a special virtual server (domain 0 in Xen)
- **Control Plane**
  - Centralized server
- **Hotspot Detector**
  - Detect *when* a hotspot occurs
- **Profiling Engine**
  - Decide *how much* to allocate
- **Migration Manager**
  - Determine *where* to migrate

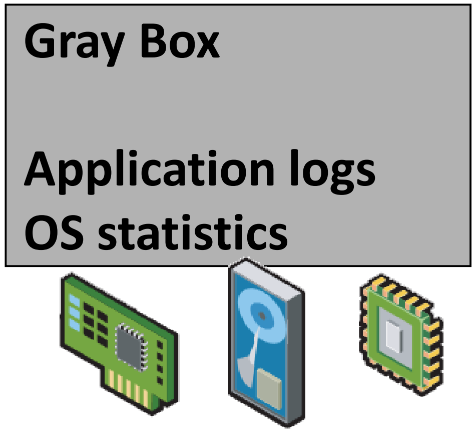


PM = Physical Machine
VM = Virtual Machine

Black-Box and Gray-Box

- **Black-box:** only data from outside the VM
  - Completely OS and application agnostic



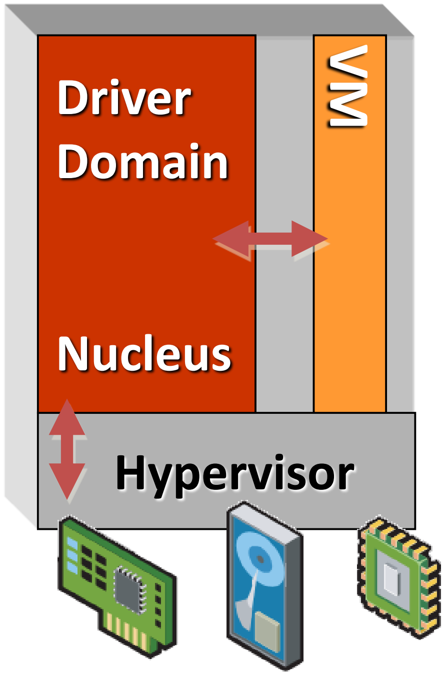


- **Gray-Box:** access to OS stats and application logs
  - Request level data can improve detection and profiling
  - Not always feasible – customer may control OS

Is black-box sufficient?
What do we gain from gray-box data?

Black-box Monitoring

- **Xen uses a "Driver Domain"**
  - Special VM with network and disk drivers
  - Nucleus runs here
- **CPU**
  - Scheduler statistics
- **Network**
  - Linux device information
- **Memory**
  - Detect swapping from disk I/O
  - Only know when performance is poor



Black-box Monitoring

- **CPU Monitoring:** By incrementing the Xen hypervisor, it is possible to provide domain-0 with access to CPU scheduling events which indicate when a VM is scheduled and when it relinquishes the CPU. These events are tracked to determine the duration for which each virtual machine is scheduled within each measurement interval *I*.

- **Network Monitoring:** Domain-0 in Xen implements the network interface driver and all other domains access the driver via clean device abstractions. Xen uses a **virtual firewall-router (VFR) interface**; each domain attaches one or more virtual interfaces to the VFR. Doing so enables Xen to multiplex all its virtual interfaces onto the underlying physical network interface.

Black-box Monitoring

- **Memory Monitoring:** Black-box monitoring of memory is challenging since Xen allocates a user specified amount of memory to each VM and requires the OS within the VM to manage that memory; as a result, the memory utilization is only known to the OS within each VM.

- It is possible to instrument Xen to observe memory accesses within each VM through the use of shadow page tables, which is used by Xen's migration mechanism to determine which pages are dirtied during migration.

- However, trapping each memory access results in a significant application slowdown and is only enabled during migrations. Thus, memory usage statistics are not directly available and must be inferred.

Black-box Monitoring

- **Black-box monitoring is useful in scenarios where it is not feasible to "peek inside" a VM to gather usage statistics. Hosting environments, for instance, run third-party applications, and in some cases, third-party installed OS distributions.**
- Amazon's Elastic Computing Cloud (EC2) service, for instance, provides a **"barebone"** virtual server where customers can load their own OS images.
- While OS instrumentation is not feasible in such environments, there are environments such as corporate data centers where both the hardware infrastructure and the applications are owned by the same entity.
- In such scenarios, it is feasible to gather OS-level statistics as well as application logs, which can potentially enhance the quality of decision making.

Gray-box Monitoring

- **Gray-box monitoring** can be supported, when feasible, using a light-weight monitoring daemon that is installed inside each virtual server.
- In Linux, the monitoring daemon uses the **/proc** interface to gather OS level statistics of CPU, network, and memory usage. The memory usage monitoring, in particular, enables proactive detection and mitigation of memory hotspots.
- The monitoring daemon also can process logs of applications such as web and database servers to derive statistics such as request rate, request drops and service times.
- Direct monitoring of such application-level statistics enables explicit detection of SLA violations, in contrast to the black-box approach that uses resource utilizations as a proxy metric for SLA monitoring.

What is a Hotspot ?

- **A hotspot indicates a resource deficit on the underlying physical server to service the collective workloads of resident VMs.**

- Before the hotspot can be resolved through migrations, The system must first estimate how much additional resources are needed by the overloaded VMs to fulfill their SLAs; these estimates are then used to locate servers that have sufficient idle resources.

Hotspot Detection

- **The hotspot detection algorithm** is responsible for signalling a need for VM migration whenever SLA violations are detected implicitly by the black-box approach or explicitly by the gray-box approach.
- Hotspot detection is performed on a per-physical server basis in the black-box approach—a hot-spot is flagged if the aggregate CPU or network utilizations on the physical server exceed a threshold or if the total swap activity exceeds a threshold.
- In contrast, explicit SLA violations must be detected on a per-virtual server basis in the gray-box approach—a hotspot is flagged if the memory utilization of the VM exceeds a threshold or if the response time or the request drop rate exceed the SLA-specified values.

Hotspot Detection

- To ensure that a small transient spike does not trigger needless migrations, a hotspot is flagged only if thresholds or SLAs are exceeded for a sustained time. Given a time-series profile, a hotspot is flagged if at least k out the n most recent observations as well as the next predicted value exceed a threshold. With this constraint, we can filter out transient spikes and avoid needless migrations.
- The values of k and n can be chosen to make hotspot detection aggressive or conservative. For a given n, small values of k cause aggressive hotspot detection, while large values of k imply a need for more sustained threshold violations and thus a more conservative approach.
- **In the extreme, n = k = 1** is the most aggressive approach that flags a hostpot as soon as the threshold is exceeded. Finally, the threshold itself also determines how aggressively hotspots are flagged; lower thresholds imply more aggressive migrations at the expense of lower server utilizations, while higher thresholds imply higher utilizations with the risk of potentially higher SLA violations.

Hotspot Detection

- In addition to requiring **k out of n violations**, we also require that the next predicted value exceed the threshold.

- The additional requirement ensures that the hotspot is likely to persist in the future based on current observed trends. Also, predictions capture rising trends, while preventing declining ones from triggering a migration.

Hotspot Detection

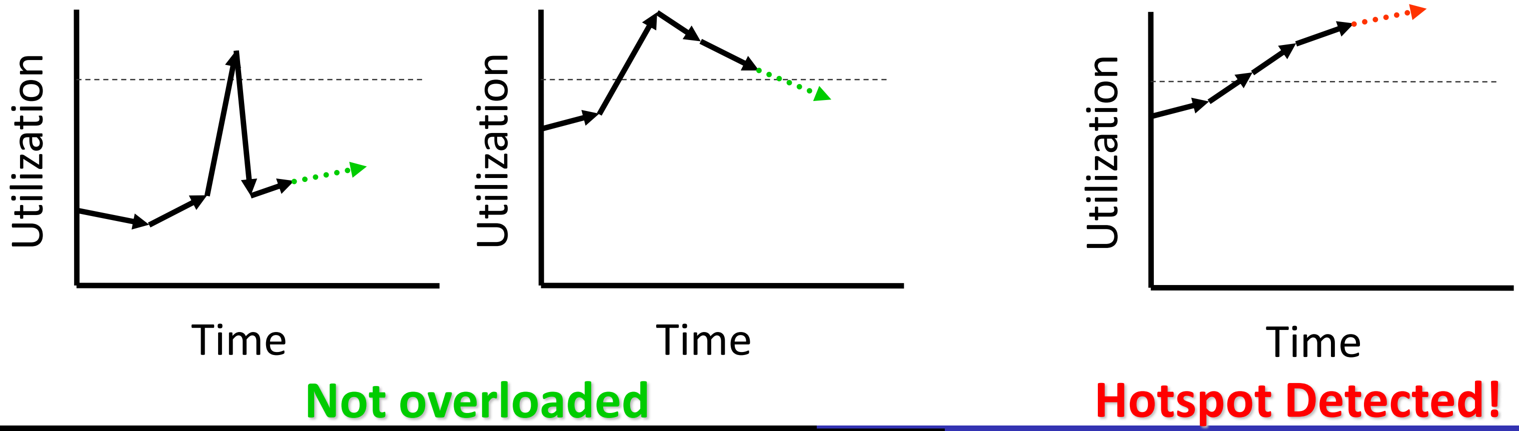
- It employs **time-series prediction techniques to predict future values.** Specifically, the system relies on the auto-regressive family of predictors, where the n-th order predictor AR(n) uses n prior observations in conjunction with other statistics of the time series to make a prediction. To illustrate the first-order AR(1) predictor, consider a sequence of observations: **u1, u2, ..., uk**. Given this time series, we wish to predict the demand in the **(k + 1)th** interval. Then the first-order **AR(1)** predictor makes a prediction using the previous value **uk**, the mean of the time series values μ, and the parameter which captures the variations in the time series. The prediction ˆuk+1 is given by:

$$\boxed{\hat{u}_{k+1} = \mu + \phi(u_k - \mu)}$$

- As new observations arrive from the nuclei, the hot spot detector updates its predictions and performs the above checks to flag new hotspots in the system.

Hotspot Detection – When?

- **Resource Thresholds**
  - Potential hotspot if utilization exceeds threshold
- **Only trigger for *sustained* overload**
  - Must be overloaded for k out of n measurements
- **Autoregressive Time Series Model**
  - Use historical data to predict future values
  - Minimize impact of transient spikes



Resource Provisioning: Black-box Provisioning

- The provisioning component needs to estimate the peak CPU, network and memory requirement of each overloaded VM; doing so ensures that the SLAs are not violated even in the presence of peak workloads.

- **Estimating peak CPU and network bandwidth needs:** Distribution profiles are used to estimate the peak CPU and network bandwidth needs of each VM. The tail of the usage distribution represents the peak usage over the recent past and is used as an estimate of future peak needs.

- This is achieved by computing a high percentile **(e.g., the 95th percentile)** of the CPU and network bandwidth distribution as an initial estimate of the peak needs.

Limitation of the black-box approach

- **Example:** Consider two virtual machines that are assigned CPU weights of 1:1 resulting in a fair share of 50% each. Assume that VM1 is overloaded and requires 70% of the CPU to meet its peak needs. If VM2 is underloaded and only using 20% of the CPU, then the work conserving Xen scheduler will allocate 70% to VM1.

- In this case, the tail of the observed distribution is a good indicator of VM1's peak need. In contrast, if VM2 is using its entire fair share of 50%, then VM1 will be allocated exactly its fair share. In this case, the peak observed usage will be 50%, an underestimate of the actual peak need. Since the system can detect that the CPU is fully utilized, it will estimate the peak to be 50 +Δ .

Resource Provisioning: (i) Black-box Provisioning

- **Estimating peak memory needs:** Xen allows a fixed amount of physical memory to be assigned to each resident VM; this allocation represents a hard upper-bound that can not be exceeded regardless of memory demand and regardless of the memory usage in other VMs.

- Consequently, the techniques for estimating the peak CPU and network usage do not apply to memory. The provisioning component uses observed swap activity to determine if the current memory allocation of the VM should be increased.

- If swap activity exceeds the threshold indicating memory pressure, then the current allocation is deemed insufficient and is increased by a constant amount Δm.

Resource Provisioning: (ii) Gray-box Provisioning

- **Since the gray-box approach has access to application level logs, information contained in the logs can be utilized to estimate the peak resource needs of the application.**
- Unlike the black-box approach, the peak needs can be estimated even when the resource is fully utilized.
- To estimate peak needs, the peak request arrival rate is first estimated. Since the number of serviced requests as well as the number of dropped requests are typically logged, the incoming request rate is the summation of these two quantities.
- Given the distribution profile of the arrival rate, the peak rate is simply a high percentile of the distribution. Let λpeak denote the estimated peak arrival rate for the application.

Estimating peak CPU needs:

- An application model is necessary to estimate the peak CPU needs. Applications such as web and database servers can be modeled as **G/G/1 queuing systems**. The behavior of such a **G/G/1 queuing system** can be captured using the following queuing theory result:

$$\boxed{\lambda_{cap} \geq \left[s + \frac{\sigma_a^2 + \sigma_b^2}{2 \cdot (d - s)} \right]^{-1}}$$

- where d is the mean response time of requests, s is the mean service time, and λ_{cap} is the request arrival rate. $\sigma^2 a$ **and** $\sigma^2 b$ are the variance of inter-arrival time and the variance of service time, respectively. Note that response time includes the full queueing delay, while service time only reflects the time spent actively processing a request.

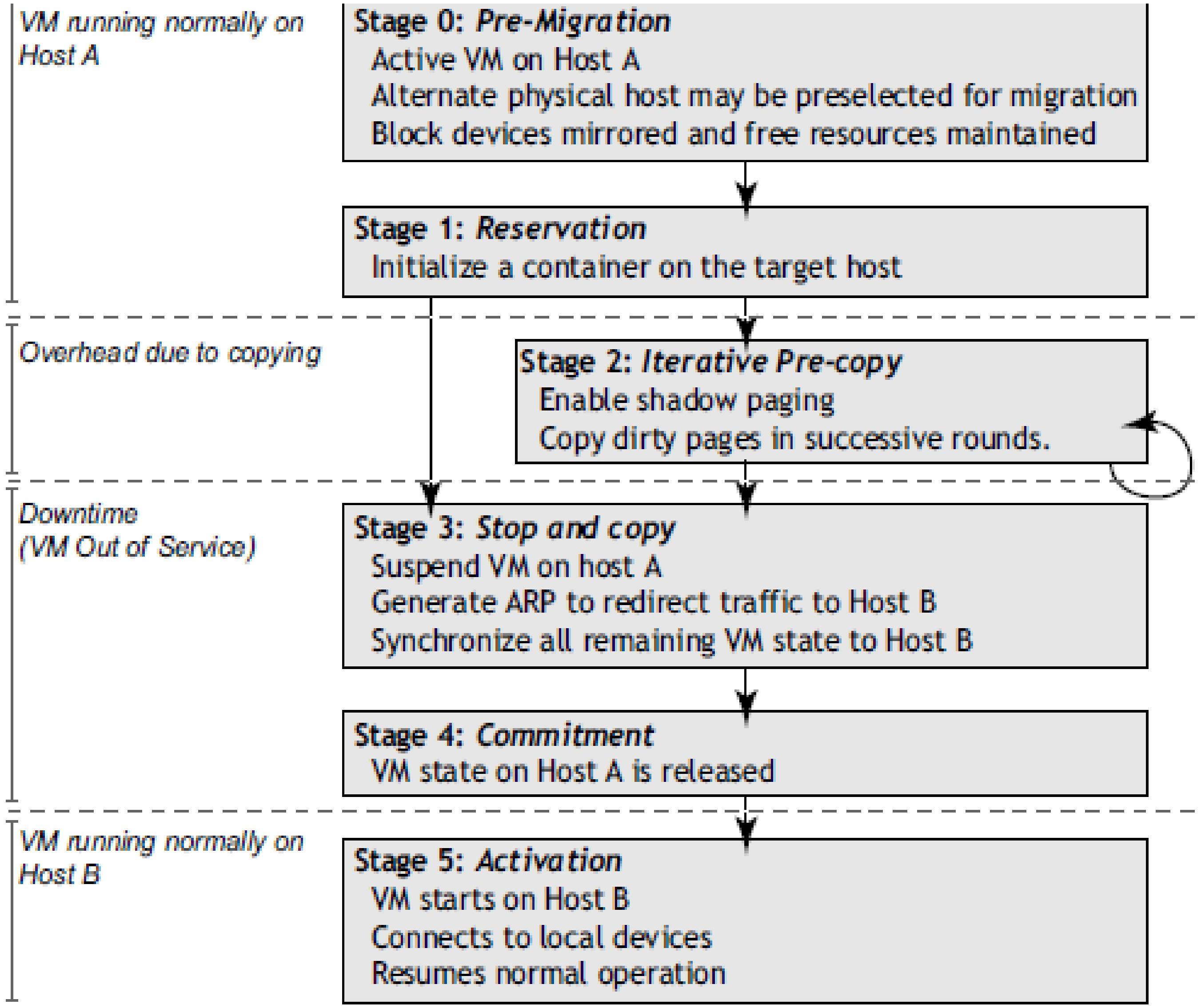
Estimating peak CPU needs:

- While the desired response time d is specified by the SLA, the service time **s** of requests as well as the variance of inter-arrival and service times **σ^2 a** and **σ^2 b** can be determined from the server logs. By substituting these values into equation, a lower bound on request rate **λcap** that can be serviced by the virtual server is obtained.
- Thus, **λcap** represents the current capacity of the VM. To service the estimated peak workload **λpeak** , the current CPU capacity needs to be scaled by the factor **λpeak / λcap**
- Observe that this factor will be greater than 1 if the peak arrival rate exceeds the currently provisioned capacity. Thus, if the VM is currently assigned a CPU weight w, its allocated share needs to be scaled up by the factor **λpeak / λcap** to service the peak workload.

Estimating peak network needs:

- The peak network bandwidth usage is simply estimated as the product of the estimated peak arrival rate **λpeak** and
- The mean requested file size **b**; this is the amount of data transferred over the network to service the peak workload. The mean request size can be computed from the server logs.

Live VM Migration Stages


VM running normally on Host A
Stage 0: Pre-Migration
Active VM on Host A
Alternate physical host may be preselected for migration
Block devices mirrored and free resources maintained
Stage 1: Reservation
Initialize a container on the target host
Overhead due to copying
Stage 2: Iterative Pre-copy
Enable shadow paging
Copy dirty pages in successive rounds.
Downtime (VM Out of Service)
Stage 3: Stop and copy
Suspend VM on host A
Generate ARP to redirect traffic to Host B
Synchronize all remaining VM state to Host B
Stage 4: Commitment
VM state on Host A is released
VM running normally on Host B
Stage 5: Activation
VM starts on Host B
Connects to local devices
Resumes normal operation

Live VM Migration Stages

- **Stage 0: Pre-Migration** We begin with an active VM on physical host A. To speed any future migration, a target host may be preselected where the resources required to receive migration will be guaranteed.

- **Stage 1: Reservation** A request is issued to migrate an OS from host A to host B. We initially confirm that the necessary resources are available on B and reserve a VM container of that size. Failure to secure resources here means that the VM simply continues to run on A unaffected.

- **Stage 2: Iterative Pre-Copy** During the first iteration, all pages are transferred from A to B. Subsequent iterations copy only those pages dirtied during the previous transfer phase.

Live VM Migration Stages

- **Stage 3: Stop-and-Copy** We suspend the running OS instance at A and redirect its network traffic to B. As described earlier, CPU state and any remaining inconsistent memory pages are then transferred. At the end of this stage there is a consistent suspended copy of the VM at both A and B. The copy at A is still considered to be primary and is resumed in case of failure.

- **Stage 4: Commitment** Host B indicates to A that it has successfully received a consistent OS image. Host A acknowledges this message as commitment of the migration transaction: host A may now discard the original VM, and host B becomes the primary host.

- **Stage 5: Activation** The migrated VM on B is now activated. Post-migration code runs to reattach device drivers to the new machine and advertise moved IP addresses.

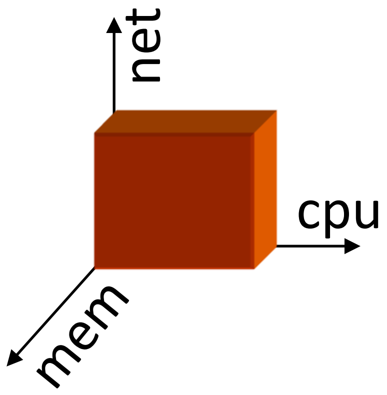
Hotspot Mitigation

- **The hotspot mitigation algorithm resorts to a heuristic to determine which overloaded VMs to migrate and where such that migration overhead is minimized.**
- Reducing the migration overhead (i.e., the amount of data transferred) is important, since Xen's live migration mechanism works by iteratively copying the memory image of the VM to the destination while keeping track of which pages are being dirtied and need to be resent. This requires Xen to intercept all memory accesses for the migrating domain, which significantly impacts the performance of the application inside the VM.
- **By reducing the amount of data copied over the network, It can minimize the total migration time, and thus, the performance impact on applications.**

Determining Placement – Where to?

- **Migrate VMs from overloaded to underloaded servers**

$$\boxed{\textit{Volume} = \frac{1}{1-cpu} * \frac{1}{1-net} * \frac{1}{1-mem}}$$



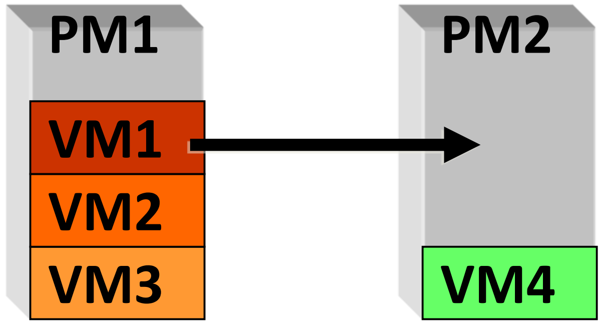
- **Use *Volume* to find most loaded servers**
  - Captures load on multiple resource dimensions
  - Highly loaded servers are targeted first

- **Migrations incur overhead**
  - Migration cost determined by RAM
  - Migrate the VM with highest Volume/RAM ratio

Maximize the amount of load transferred while minimizing the overhead of migrations

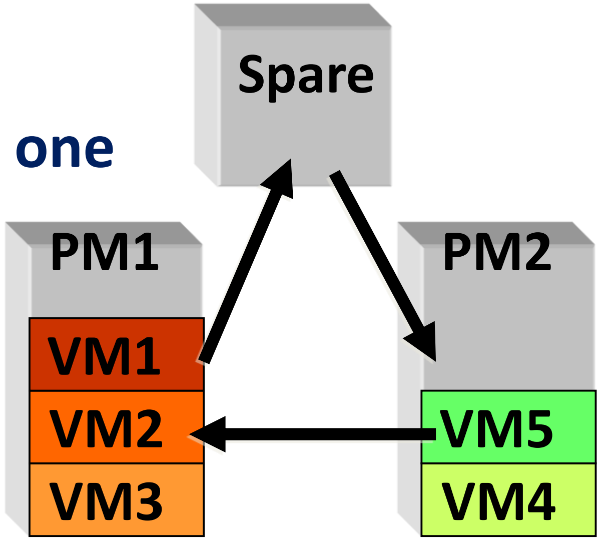
Placement Algorithm

- First try **migrations**
  - Displace VMs from high *Volume* servers
  - Use *Volume/RAM* to minimize overhead
- **Don't create new hotspots!**
  - **What if high average load in system?**
- **Swap** if necessary
  - Swap a **high *Volume* VM** for **a low *Volume* one**
  - Requires 3 migrations
    - Can't support both at once



Migration

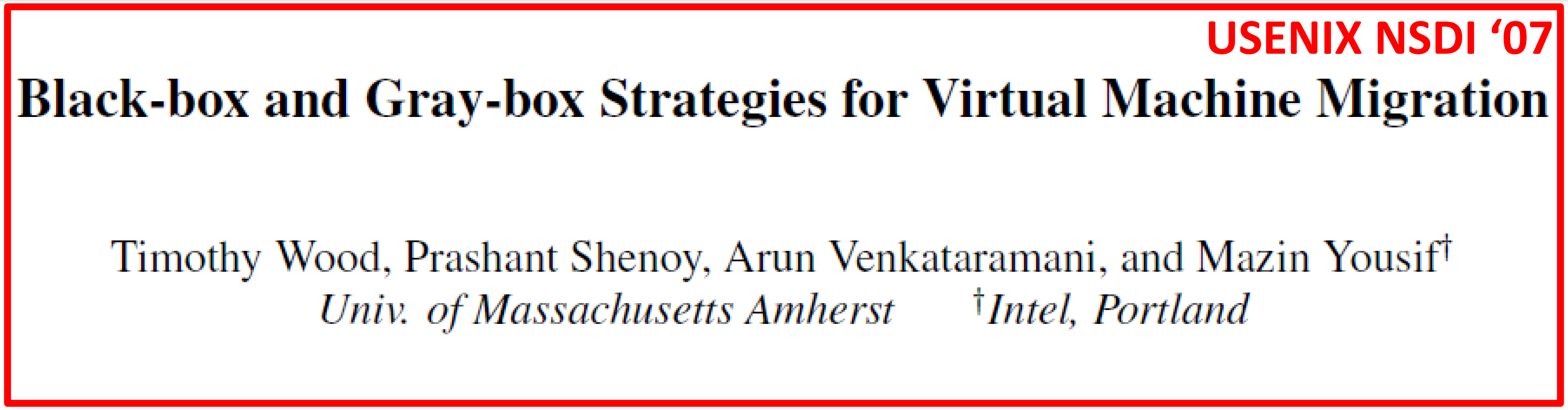
Swaps increase the number of hotspots we can resolve



Swap

Reading

USENIX NSDI ‘07

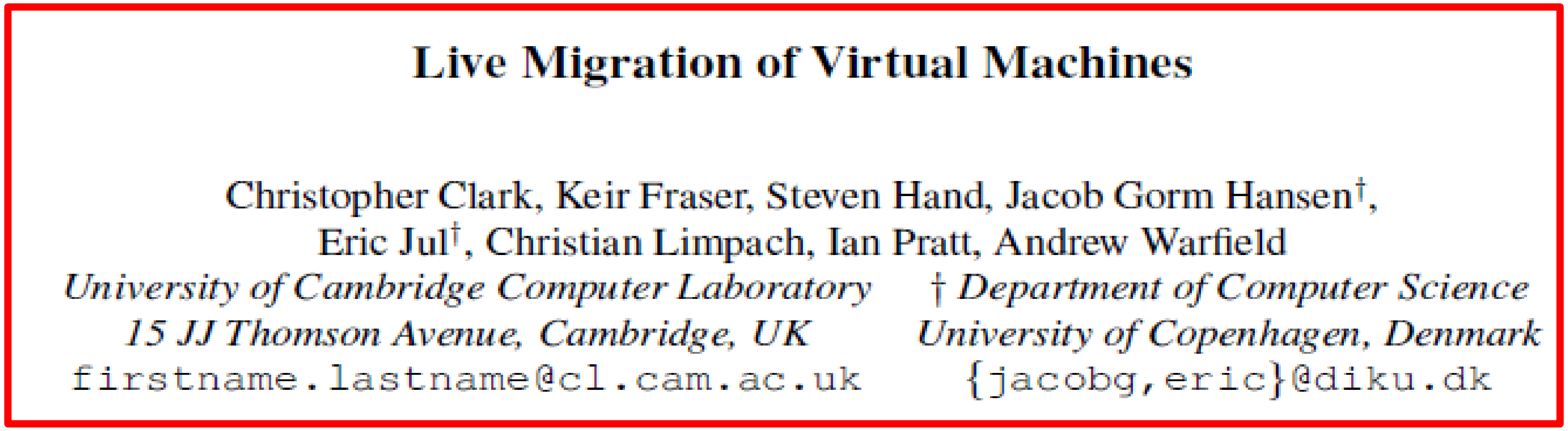


Black-box and Gray-box Strategies for Virtual Machine Migration

Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif[†]
Univ. of Massachusetts Amherst [†]*Intel, Portland*

Source: https://www.usenix.org/legacy/event/nsdi07/tech/full_papers/wood/wood.pdf

USENIX NSDI ‘05



Live Migration of Virtual Machines

Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen[†],
Eric Jul[†], Christian Limpach, Ian Pratt, Andrew Warfield
University of Cambridge Computer Laboratory [†] *Department of Computer Science*
15 JJ Thomson Avenue, Cambridge, UK *University of Copenhagen, Denmark*
`firstname.lastname@cl.cam.ac.uk` `{jacobg,eric}@diku.dk`

Source: https://www.usenix.org/legacy/event/nsdi05/tech/full_papers/clark/clark.pdf

Conclusion

- **Virtual Machine migration is a viable tool for dynamic data center provisioning.**
- In this lecture, we have discussed an approach to rapidly detect and eliminate hotspots while treating each VM as a **black-box.**
- **Gray-Box** information can improve performance in some scenarios
  - Proactive memory allocations
- **Future work**
  - Improved black-box memory monitoring
  - Support for replicated services