Transparent Throughput Elasticity for IaaS Cloud Storage Using Guest-Side Block-Level Caching

Bogdan Nicolae (IBM Research, Ireland)
Pierre Riteau (University of Chicago, USA)
Kate Keahey (Argonne National Laboratory, USA)

Presented by Pierre Riteau

UCC 2014

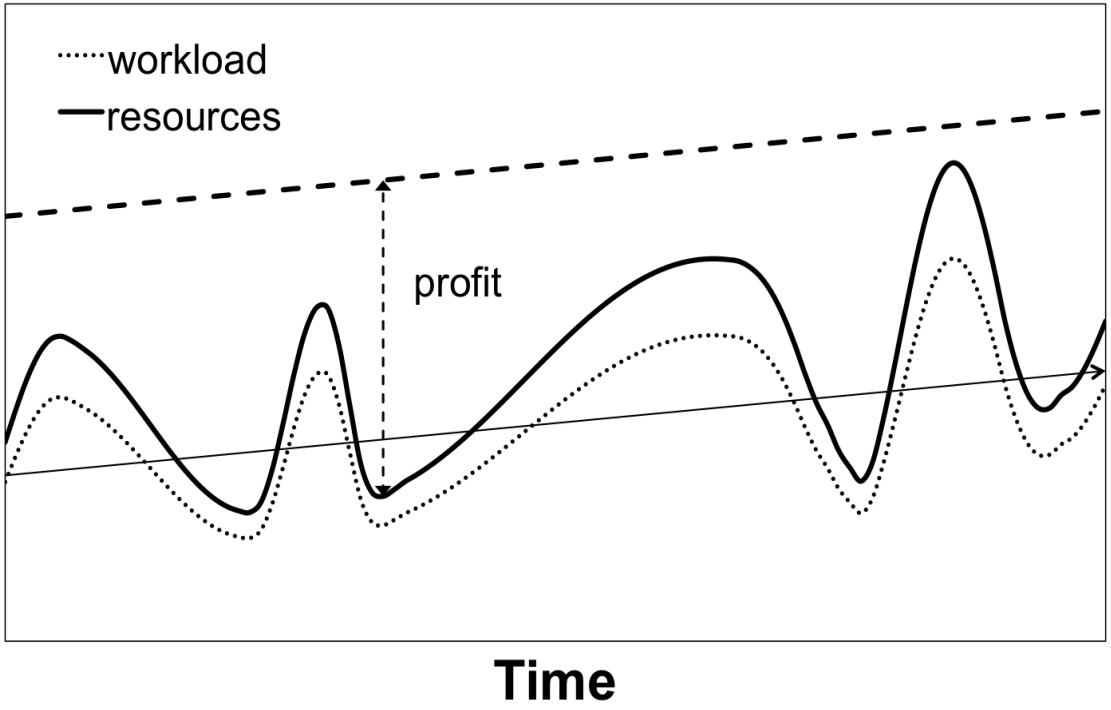
IaaS storage via virtual disks

- **How does it work?**
  - Attach/detach a virtual disk to a VM instance using REST API
  - The guest OS sees the virtual disk as a regular block device
  - A higher level abstraction (typically file system) is used to leverage the block device

- Cost model
  - Users are charged per GB and time unit a fixed cost
  - Fixed cost depends on throughput (e.g. SSD more expensive than HDD)

Elasticity: a key feature of IaaS

The ability of a system to change its **capacity** in direct response to the workload **demand**



Unawareness of elasticity typically leads to **overprovisioning** (bold dotted line) and **explosion of costs** compared to the average utilization (thin dotted line)

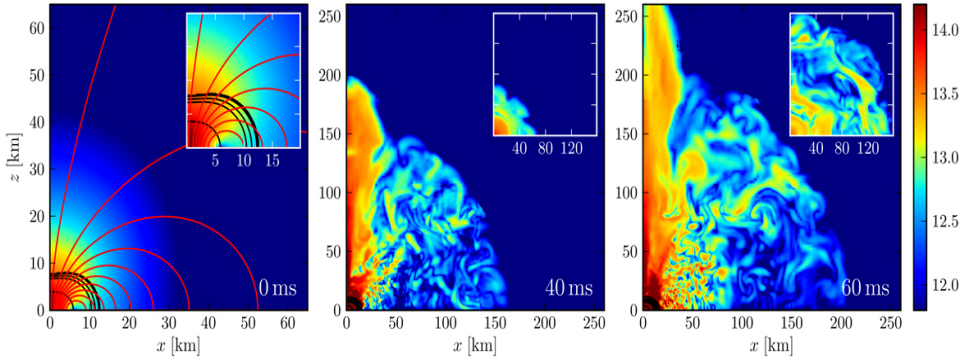
Why is storage elasticity important?

- HPC workloads moving to the cloud
- Need large number of VMs
- Store huge amounts of data

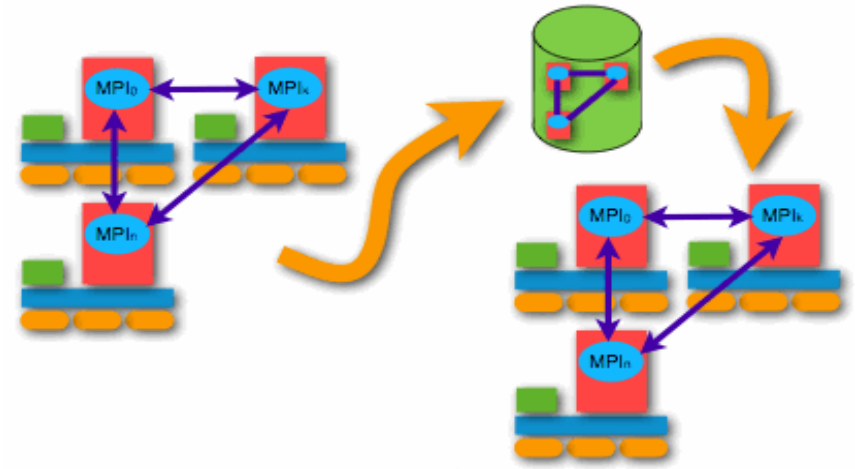
- At large scale it matters even for short runtimes
  - Overprovisioning costs accumulate and explode
  - **Rapid fluctuations** of space (*IPDPS 2014*) and **throughput** requirements

Checkpointing: a common pattern

- HPC applications interleave computationally intensive phases with I/O intensive phases
- Most common scenario: checkpointing
  - All processes dump in-memory data simultaneously to capture a globally consistent snapshot
  - Happens every N iterations
- Why checkpointing?
  - Fault tolerance: Checkpoint-Restart
  - Debugging
  - Migration



Simulation of black hole formation



Checkpoint-Restart

Problem summary

- Many apps interleave I/O intensive periods with compute-intensive periods
- Scientific applications use large number of instances and virtual disks
- Virtual disks have fixed I/O throughput
- Traditional approach:
  - **Overprovision** with faster virtual disks to get best performance during **I/O peaks**
  - **Underuse** outside peaks → **high storage costs**

Solution

- Block device with elastic disk throughput
- **High performance** during I/O peaks
- **Minimizes storage costs**
- Large, slow virtual disk for primary storage
- Boosts throughput with small, short-lived, fast virtual disks used as caching layer

Three key design principles

1. Transparent block level caching
2. Adaptive flushing of dirty blocks
3. Access-pattern-aware cache sizing

1. Transparent block level caching

- Elastic storage as single block device
- Transparent access from applications
- Transparent approach from cloud provider
- Monitor I/O throughput utilization
- When above utilization threshold (*UT*):
  - Provision new, faster *caching device*
  - I/O throughput boost until threshold below *UT*
  - Cached data flushed to *backing device*
  - Caching device removed

2. Adaptive flushing of dirty blocks

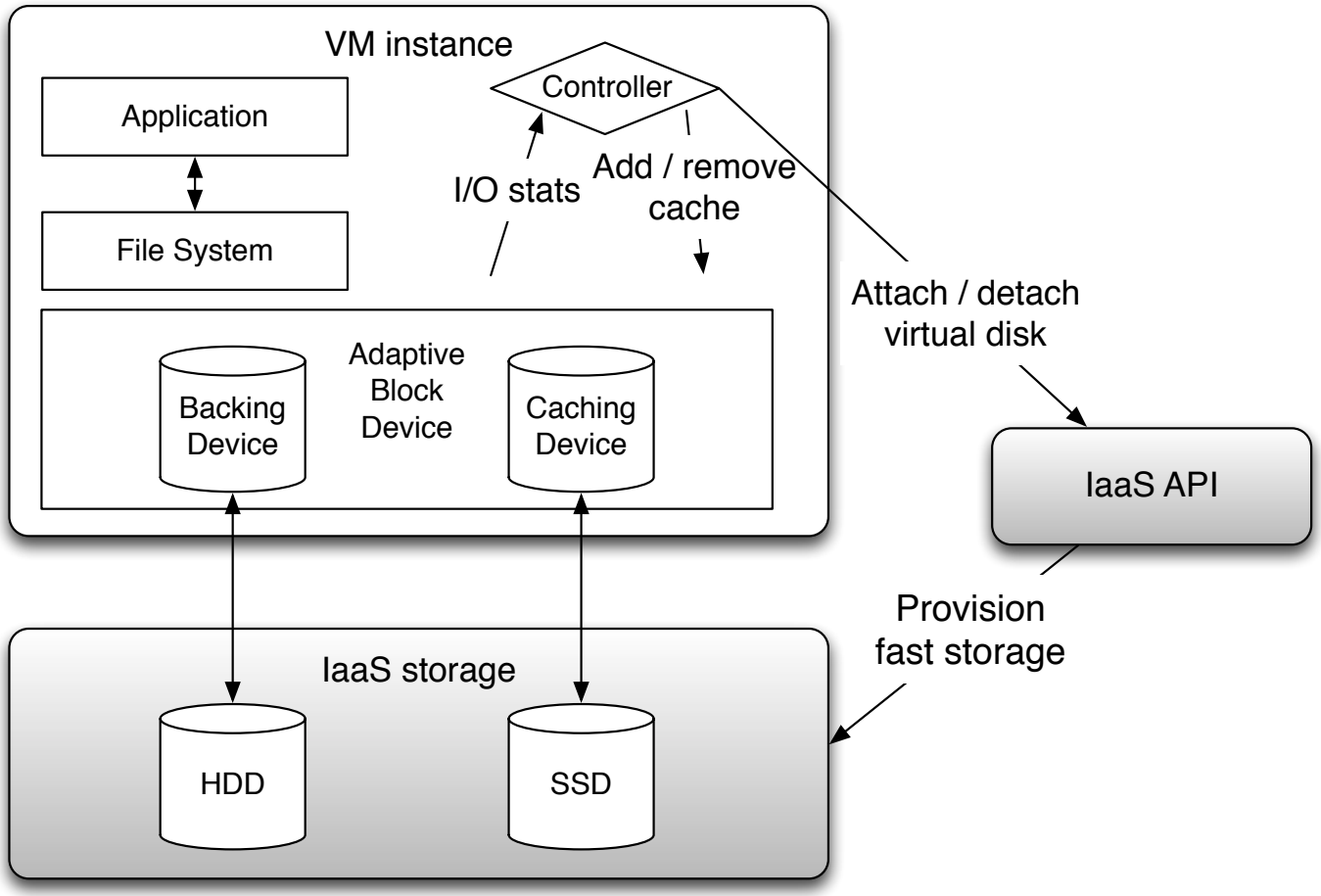
- Constant data movement from caching to backing device can hurt performance
- Read/write cache with a custom dirty block commit strategy (dynamic writeback)
- First phase: writeback, priority to application I/O
  - Async flush if bandwidth available or cache full
- Phase change after time *ID* (inactivity delay) spent under *Utilization Threshold*
- Second phase: priority to cache flush
  - Writes go straight to backing device

3. Access-pattern-aware cache sizing

- Must be careful with size of caching device
- Cache too large → long flush delay
- Cache too small → forced flush
- Optimize caching device size
  - According to access patterns
  - Relies on repetitive behavior of scientific apps
  - Start with large cache and monitor utilization
  - Decrease or increase size as required

Architecture

- Based on **Bcache**
- Controller monitors I/O throughput and selects best size for caching device



Experimental evaluation platform

- Cloud environment simulated via QEMU/KVM 1.6 as the hypervisor
- 32 physical nodes on *Shamrock* testbed
- VM guest operating system:
  - Recent Debian Sid with Linux 3.12
- VM network interfaces
  - VirtIO driver
  - Bridged to host physical interface
  - Enables direct L2 communication between VMs

Experimental evaluation

- Static pre-allocation with slow virtual disk (***static-slow***)
  - Large, fixed-size virtual disk as RAW file
  - Fixed maximum I/O bandwidth
- Static pre-allocation with fast virtual disk (***static-fast***)
  - Similar as above but using **RAM-disk**
  - Fixed (but **higher**) maximum I/O bandwidth
  - **Simulates SSD**
- Throughput elasticity with our approach (***adaptive***)
  - Backing device as in *static-slow*
  - Caching device as in *static-fast*

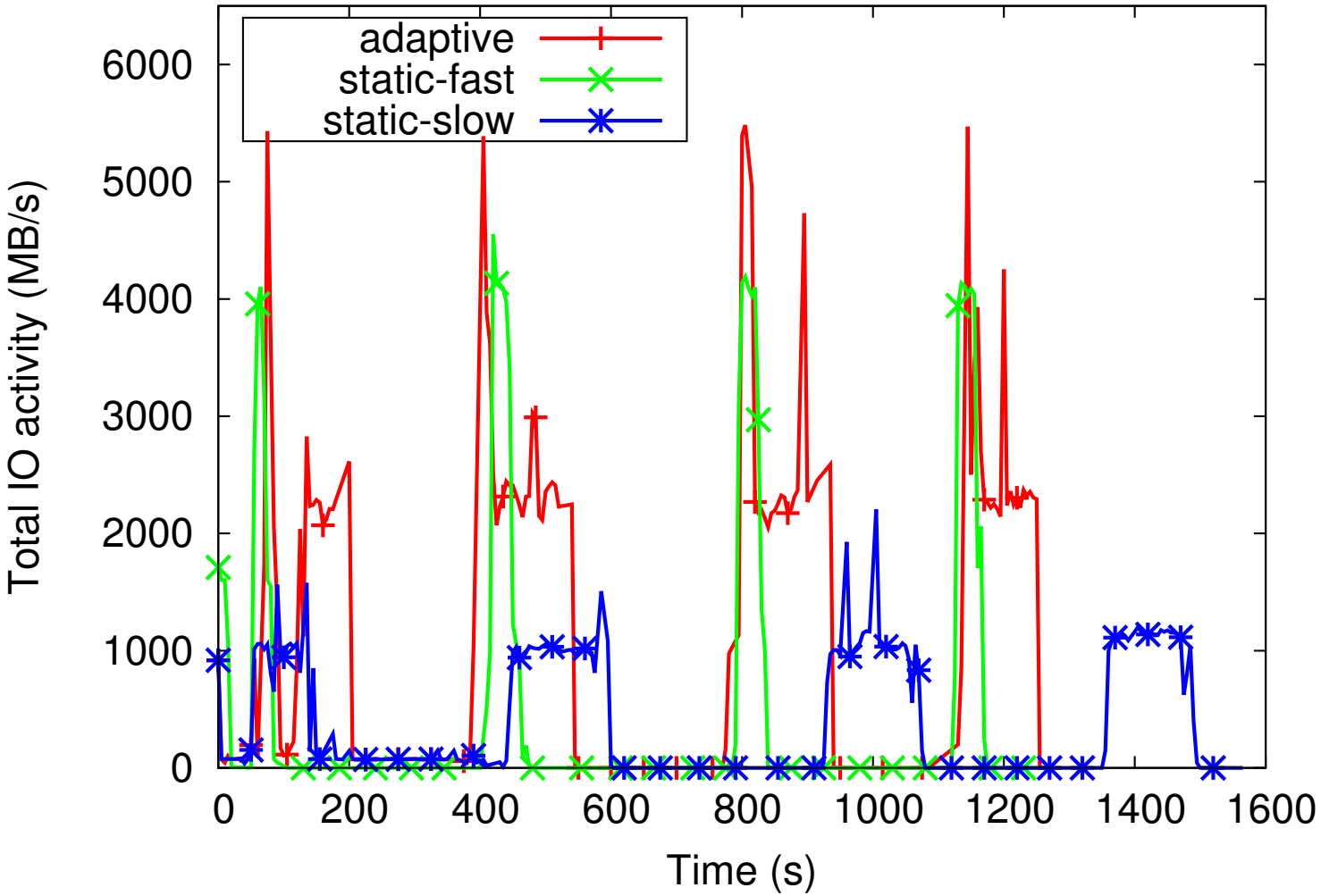
Metrics

- Performance overhead vs. *static-fast*
  - Micro benchmarks: sustained I/O throughput
  - Real-life applications: completion time
- Total adjusted storage accumulation
  - Sum of storage cost for all VMs
- Evolution of I/O activity
  - For understanding the behavior of the system

Case study: CM1

- Real-life HPC MPI application
- Alternates between compute phase and I/O-intensive phase (results and checkpoints)
- 32 VMs with one per physical node
  - 11 cores per VM (one for OS and 10 for CM1)
  - One core kept for host OS
  - 18 GB of RAM (enough for 10 MPI processes)
- Disks:
  - Backing device: 50 GB
  - Static-slow: 33 MB/s
  - Static-fast: 128 MB/s with initial size of 10 GB

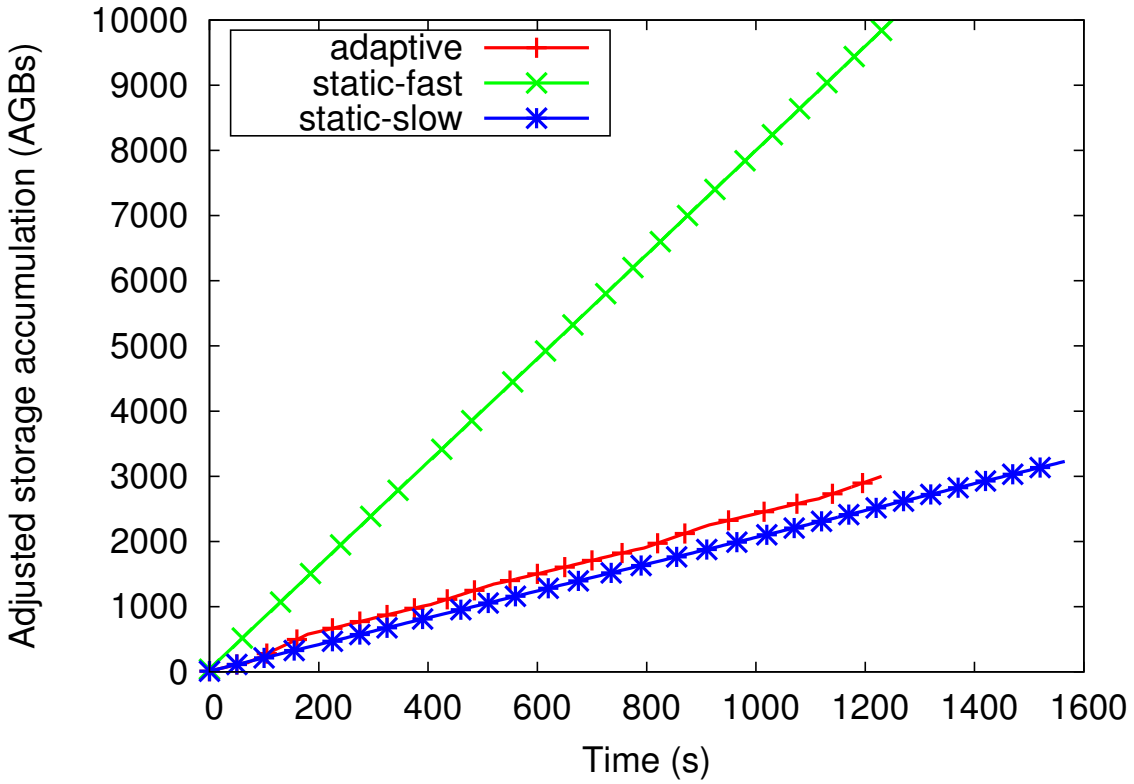
CM1 I/O throughput



CM1 performance

Approach	Completion Time	Overhead
static-slow	1471s	23%
static-fast	1190s	–
adaptive	**1231s**	**3.3%**

CM1 storage costs



- Compared with *static-fast*, our *adaptive* approach amounts to a **reduction** of **65%** in cost for a performance overhead of **3.3%**.
- Compared with *static-slow*, our *adaptive* approach amounts to a **reduction** of **7%** in cost by being **16% faster**.

Conclusions

- Our **transparent** solution achieves **great performance** with massive **cost reductions**
  - Cost reduction of **65%** to **70%** versus overprovisioning with **1% - 3.3% overhead**
- **Lower cost** than underprovisioning too
  - 16% faster → **7% less expensive**
- Caching no longer physically limited
- Can adapt to applications demands

Questions?