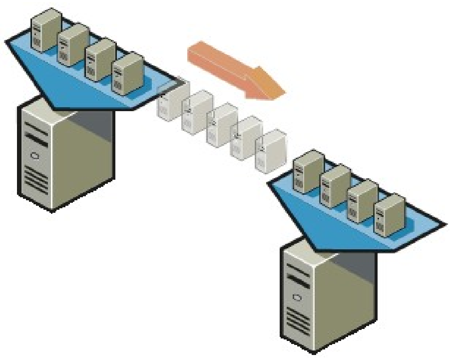
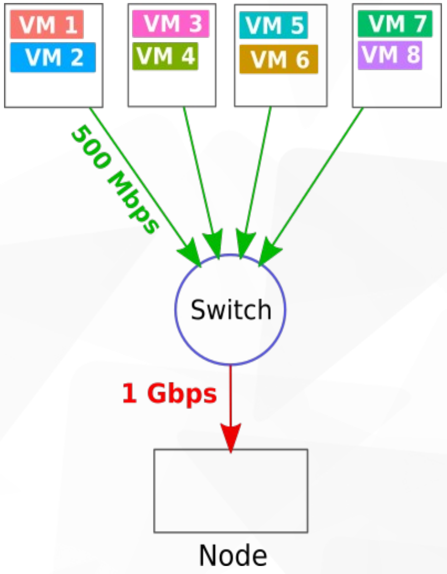
Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations

Vincent Kherbache, *Fabien Hermenier*,
Eric Madelaine

Dynamic VMs management

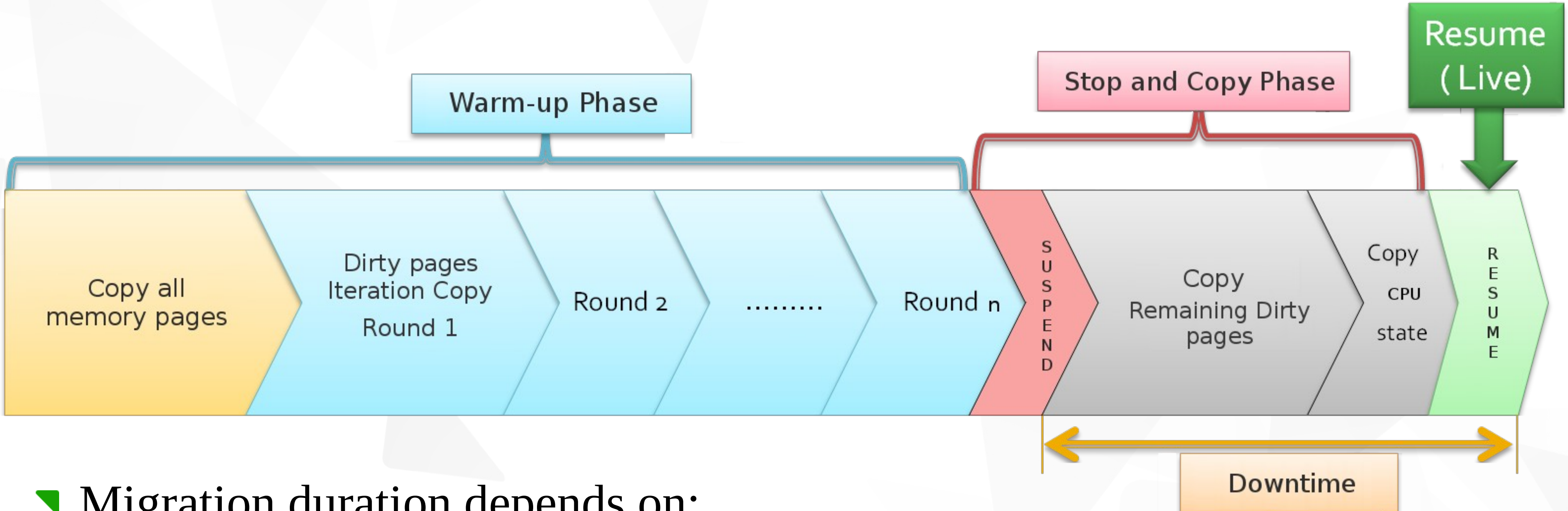


- Use-cases
  - Load-balancing
  - Maintenance tasks on production servers
  - Reducing energy consumption
- Mechanism
  - Live migration
- In practice
  - Schedule multiple migrations to terminate ASAP



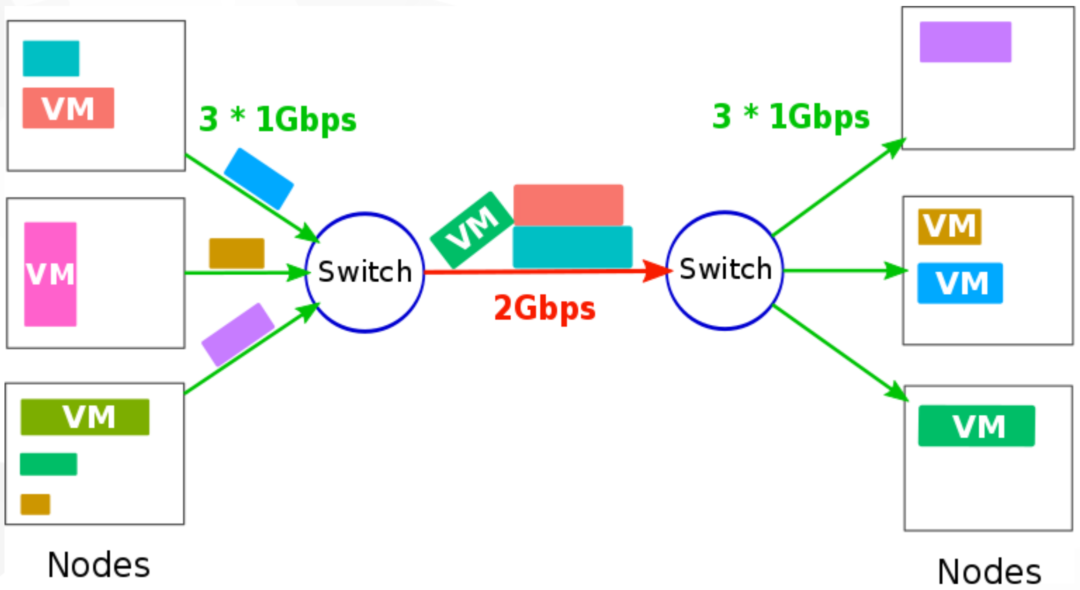
Live-migration: pre-copy algorithm

- Migration in 2 phases:
  1. Warm-up phase
  2. Stop-and-copy phase



- Migration duration depends on:
  - the available bandwidth
  - the VM memory activity

Migrations scheduling in theory



Intuitions:

- Allocate as much bandwidth as possible per migration
- Parallelize without reducing maximal migration bandwidth

State of the art

- Proposed solutions: [Entropy, BtrPlace, CloudSim, Memory Buddies, ...]

Theoretical simplifications:

- Non-blocking network
- Memory workload ignored
- Abusive or inappropriate parallelism

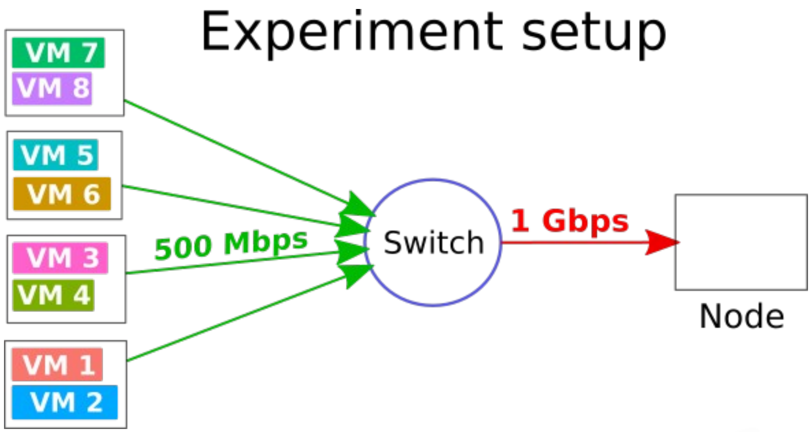
Practical consequences:

- Unanticipated long migrations
- Reduced VMs performance
- Limited fine-grained control capabilities

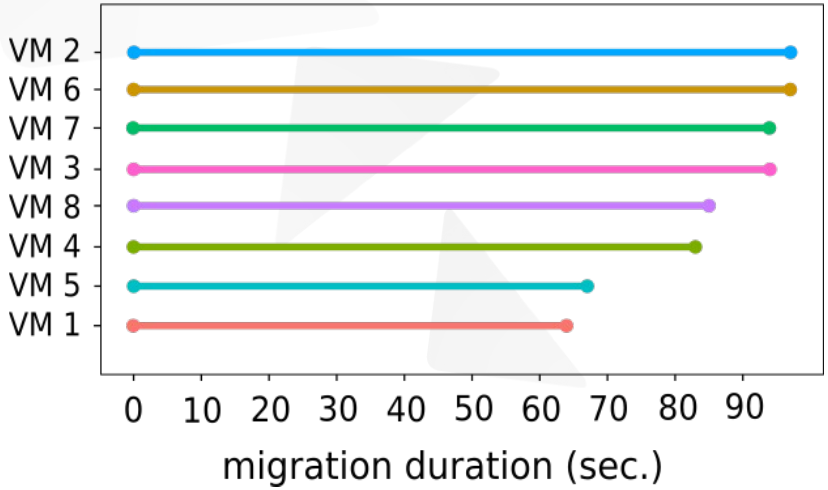
Migrations scheduling in practice

Compute for each migration:

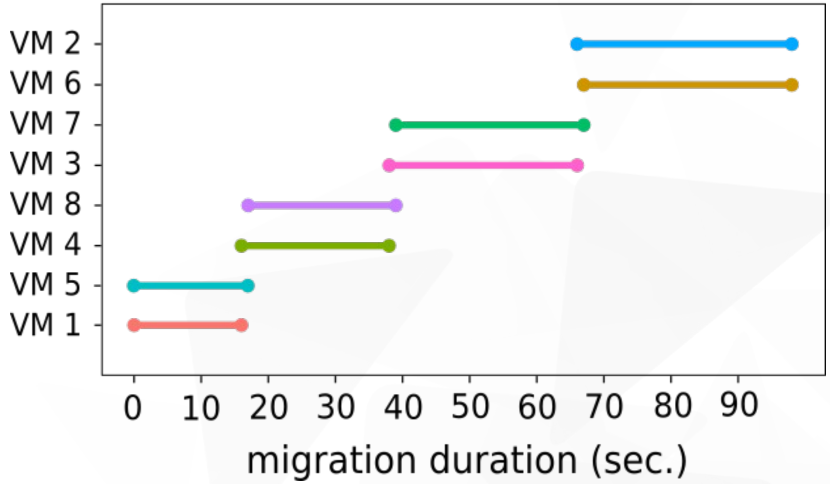
- The bandwidth to allocate
- Its theoretical duration
- The moment to start



[Entropy, BtrPlace, CloudSim]



Optimal schedule

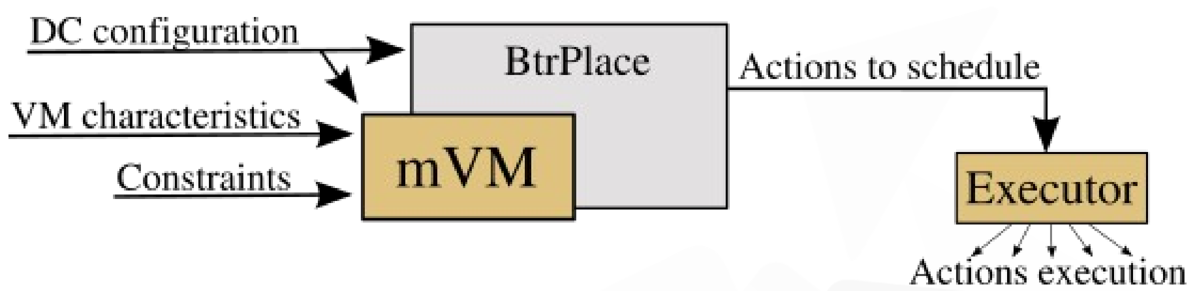


- All migrations in parallel
- Long migrations duration

- Topology-aware parallelism
- Group migrations by duration

Solution: **mVM**, a migrations scheduler

- Replace the migrations scheduler of BtrPlace
- Propose a new scheduling model
  - Network model
  - Migration model



~ 1600 lines of Java code

- Include an energy model
  - Derived from *[Liu et al., Cluster'13]*
- Domain specific constraints
  - Temporal scheduling constraints / energy constraint

Migration model: estimate the duration of a migration

- Minimal duration (without workload)
  - *Memory used / Bandwidth* ([Entropy, BtrPlace, CloudSim])

- Real duration
  - Memory dirty pages transfer

    2 phases: Hot pages → Cold pages

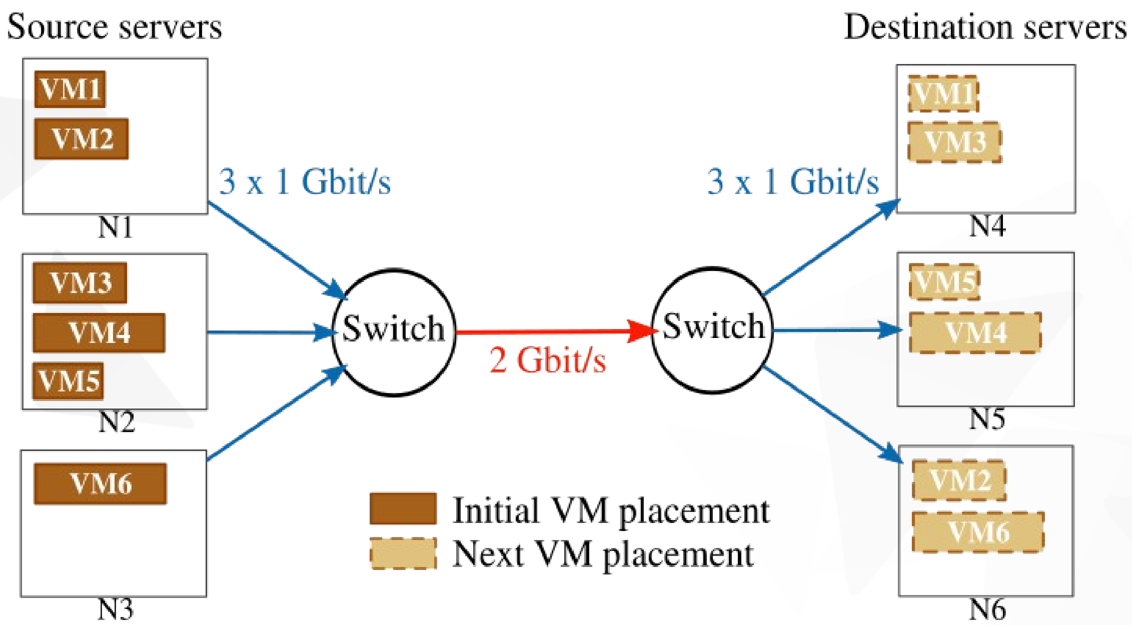
  - Dirty rate equivalent to a bandwidth reduction

$$Duration = \frac{MemoryUsed}{Bandwidth - \mathrm{DirtyPageRates}}$$

- Maximal bandwidth already known
  - Pre-compute migrations duration

Networking model: concepts

- Sharing bandwidth over time
  - Full-duplex links
  - Heterogeneous topologies
  - Blocking network elements



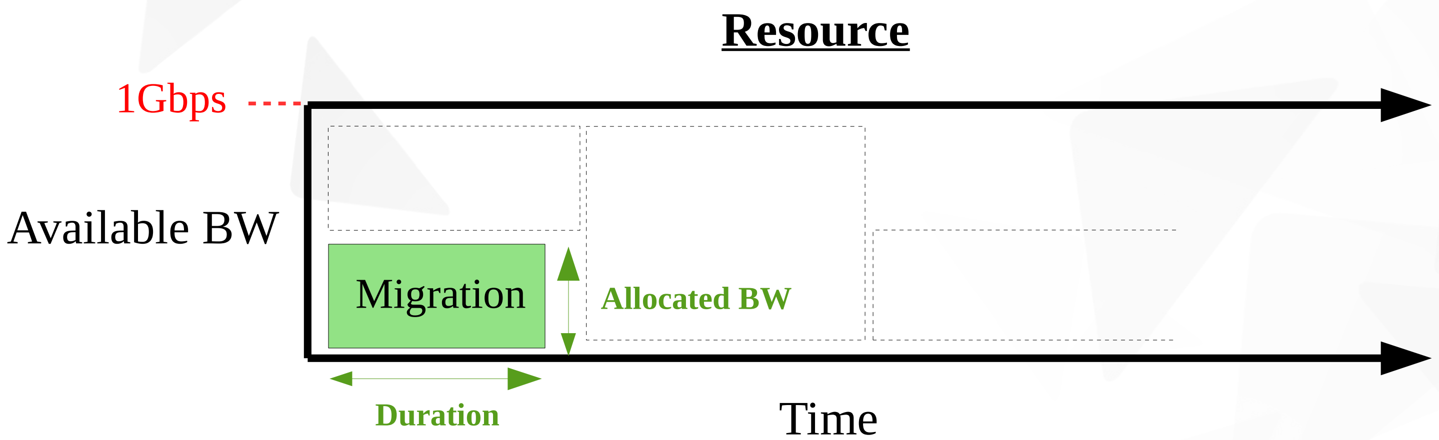
Networking model : implementation

- Mainly implemented using standard « `cumulative` » constraints
  - **Place tasks with varying heights and lengths on limited resources:**

    1 task <=> 1 migration

    resources <=> network elements

  - **2** resources per link: <u>uplink</u> and <u>downlink</u> bandwidth (full-duplex)
  - **1** resource per blocking switch (limited switch capacity)



Temporal control of the scheduling

- Temporal **constraints**:
  - `sync (vm[1-4]);`
  - `seq (vm[5,8]);` } Control the parallelism
  - `before (vm-1,vm-7);` → Control the priority

- **Objective**: MinMTTR
  - Migrate each VM as soon as possible
  - Ensure a low completion time

Energy control of the scheduling

- Absolute and relative power capping **constraints**:
  - `powerBudget (1000 Watts);`
  - `powerBudget (500 Watts, [22:00-06:30]);`

DC4Cities

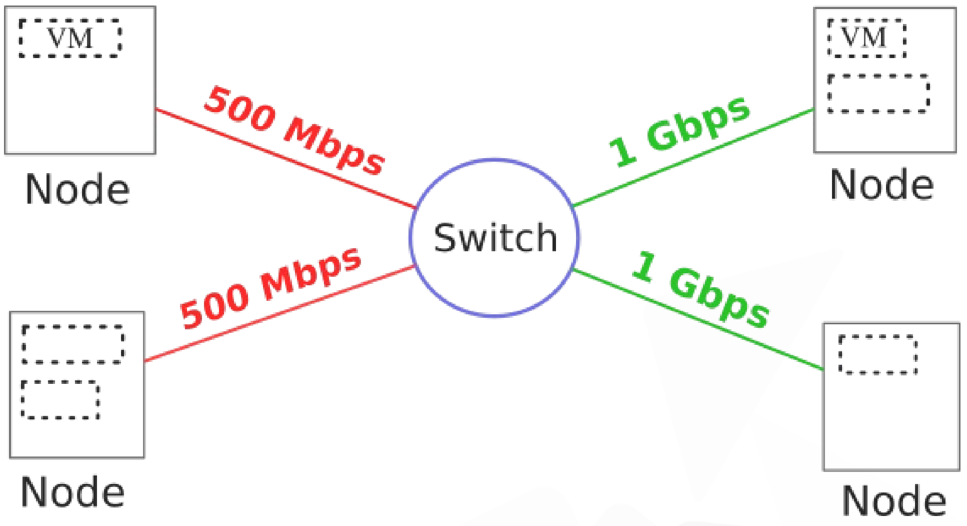
  - Allows to adapt the power consumption according to:
    - Availability of renewable energies → maximize the use of renewable
    - Energy cost variability → reduce energy costs

- **Objective**: Minimizing the energy consumed
  - Ensure a low completion time
  - Shutdown idle nodes at the earliest

Evaluation: parallelism decisions

50 random scenarios

- Configuration:
  - 4 servers
  - 10 VMs – 2 templates
  - Heterogeneous network
  - Random placements



- Experimental setup:
  - Hypervisor: KVM
  - Shared storage (NFS)
  - Trafic shaping via « tc » command
  - Workloads via « stress » command

Evaluation: migration duration

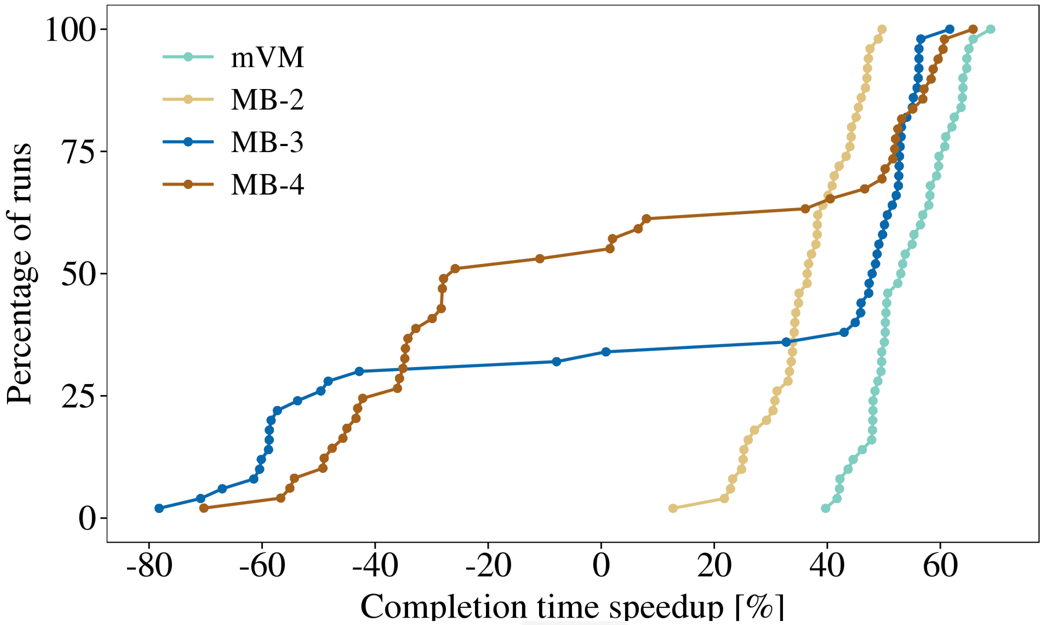
- Against a **sequential schedule** (lowest migration durations)
  - Only 7% slowdown for mVM
  - 30% for the best MB configuration
- Appropriate parallelism
  - vary from 2 to 6 in mVM
  - *constant* in MB ([2-4])

Scheduler	Mean migration time (sec.)	Average slowdown (%)
mVM	**45.55**	**7.35 %**
MB-2	57.22	29.69 %
MB-3	113.2	141.3 %
MB-4	168.6	259.2 %

Evaluation: completion time

- Against a **sequential schedule**

CDF



- mVM provides the best speedups
- MB not always reliable
  - Blind parallelization!

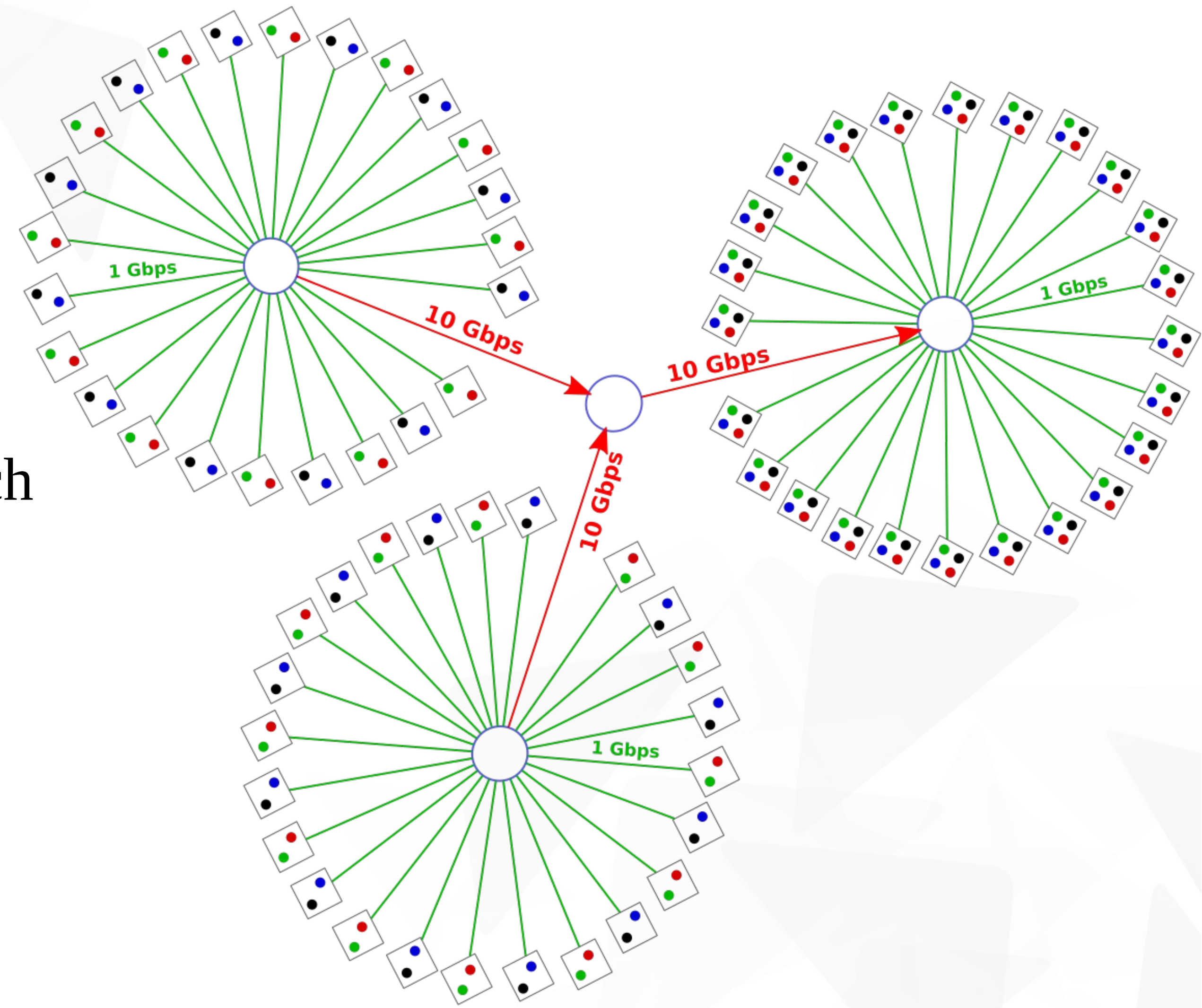
- 54% speedup against 36% for MB
- mVM is 30% faster than MB

Scheduler	mVM	MB-2	MB-3	MB-4
Mean migration time (sec.)	**212,8**	295,9	394,6	479,4
Average speedup (%)	**54,18 %**	36,42 %	15,94 %	-2,64 %

Evaluation: minimizing the energy consumed

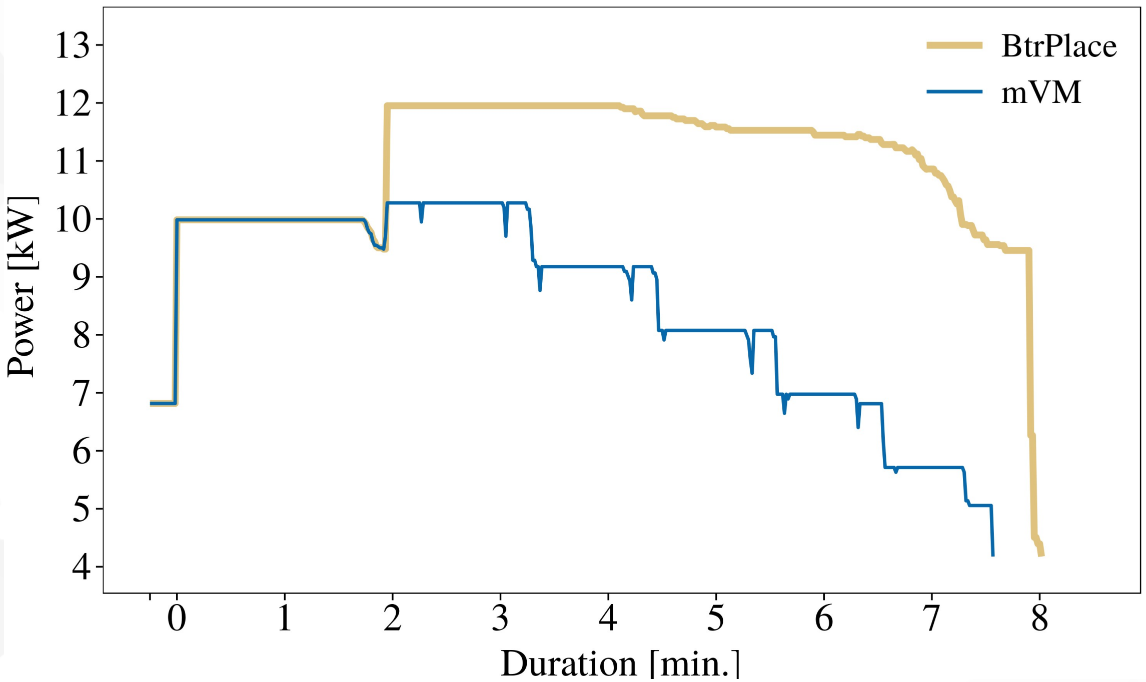
Decommissioning scenario:

- 3 * 24 => 72 servers
- 2 source → 1 destination
- 2 VMs per source server
  => 96 migrations
- 10 Gbit/s aggregation switch



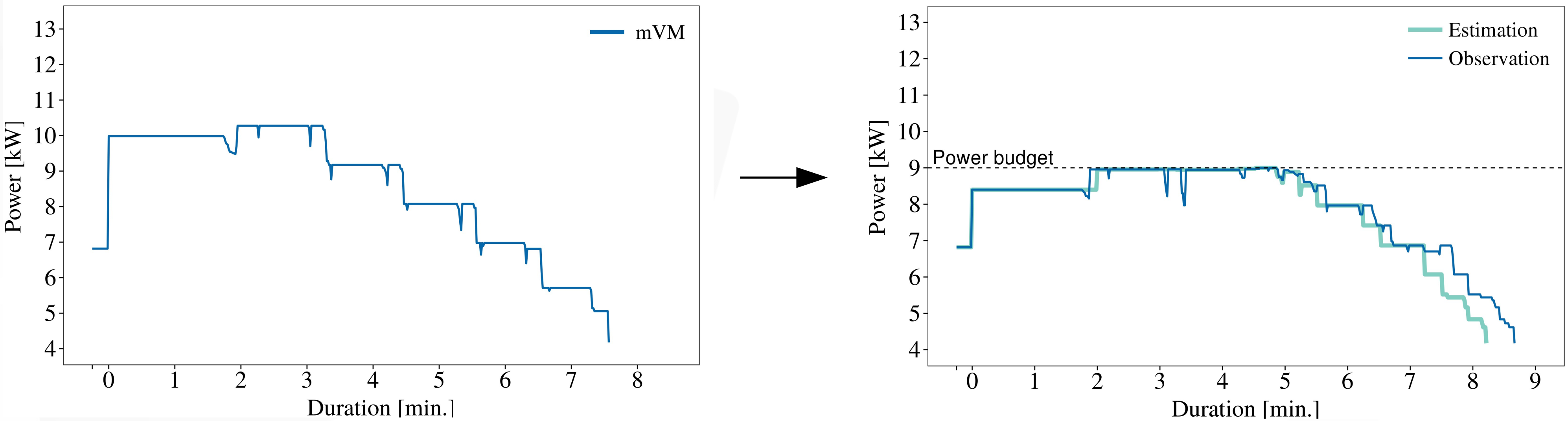
DC4Cities

Evaluation: minimizing the energy consumed



- mVM behavior:
  - Migrate VMs 10 by 10, an optimal parallelism
  - Release nodes at the earliest to save energy
  - 21.55% of energy saved compared to BtrPlace

Evaluation: 9 kW power capping PoC



- Cap the instantaneous power usage to fit renewable energy availability

- mVM deductions:
  - Boot actions postponed
  - Restricted parallelism

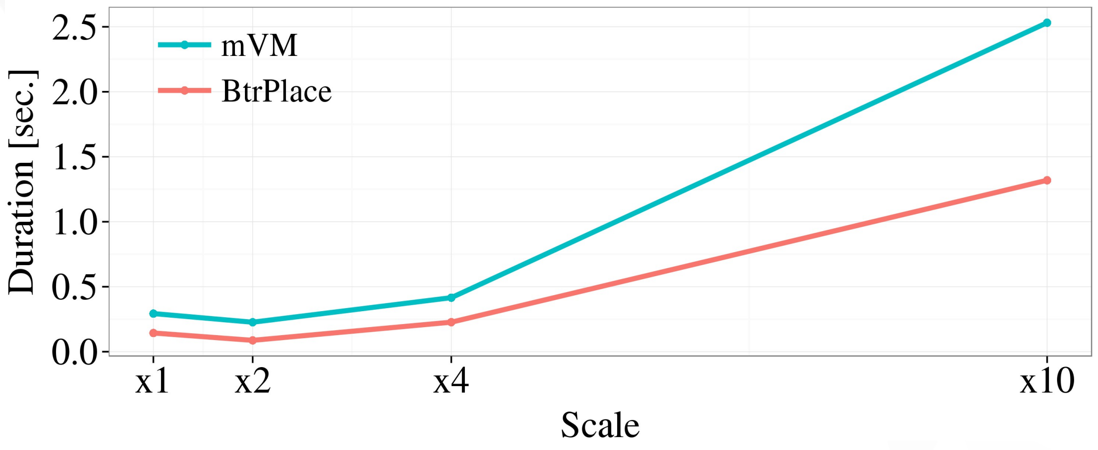
- Theoretical VS Real:
  - 93 % accuracy for migration durations
  - 6 % longer (32 sec.) than the estimation

Evaluation : scalability

Scheduling problem: NP-Hard

x1
- 96 VMs
- 72 Servers
- Network: 10 Gbps

x10
- 960 VMs
- 720 Servers
- Network: 100 Gbps



1,5 additionnal seconds

Larger scale: Partitioning migrations. ***e.g.*** per cluster / rack / blade / ...

Conclusion

Migrations scheduling

- mVM considers network and memory loads
  - Accurate migrations scheduler (> 90 %)
  - Migrations complete 20.4 % faster than Memory Buddies
- Controlable via high level constraints
  - Synchronization, sequentialization / parallelization
  - Energy aware management
    - « power capping » constraints
    - 21 % energy saved during a decommissioning scenario

Future works

- Joint placement and scheduling decisions.
- Manage user-defined downtime as a constant into the model.
- Consider a shared management & production network.

- **Contact**
  - **Vincent KHERBACHE**
  - **vincent.kherbache@inria.fr**
  - **http://vincent.kherbache.fr**

- mVM is open source
  - shipped within BtrPlace
    - http://www.btrplace.org

- Reproducibility
  - https://github.com/btrplace/migrations-UCC-15

- Tutorials
  - https://github.com/btrplace/scheduler/wiki