Studying the RJMS, applications and File System triptych:
a first step toward experimental approach

Joseph Emeras Yiannis Georgiou Olivier Richard Philippe Deniel

November 22-24, 2010

Outline

Outline

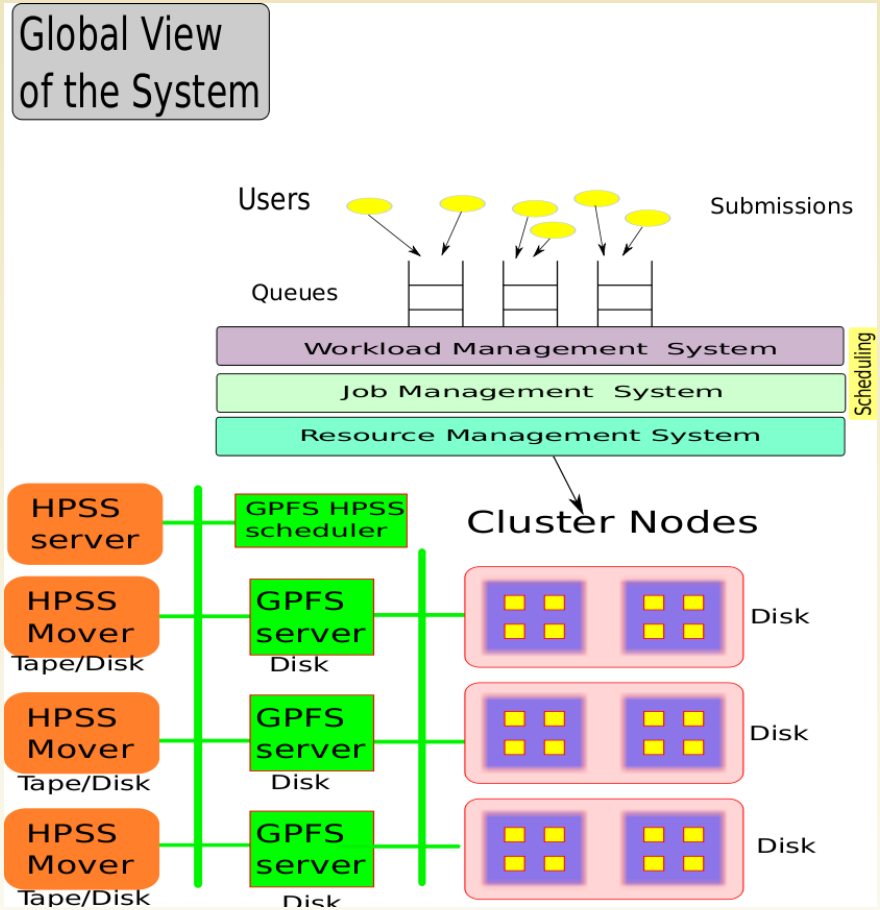
PhD Thesis Context

Study the large computing infrastructures

- study their performance
- understand their behaviours
- comprehend their evolution

Consider a global level

- Resource and Job Management System (RJMS)
- I/O
  - Distributed File System
  - Storage Backup
  - Network
  - ...
- Applications



PhD Thesis Context

Proposal

- experimental approach
- traces (real or synthetic) used to evaluate the system's performance
- performance evaluation dedicated framework

PhD Thesis Context

Why considering the IOs?

- Generally, one study the RJMS and IOs separately
- In HPC, the File System / Network is a nerve center ▸ NERSC's article
- Incomplete view
  - jobs in RJMS do IO operations
  - depending on the job's IO pattern, alter performance
    - job itself
    - other jobs
    - overall system
- So, a global view in performance evaluation will consider both RJMS and IOs

PhD Thesis Context: Questions

How to evaluate a large computing infrastructure?

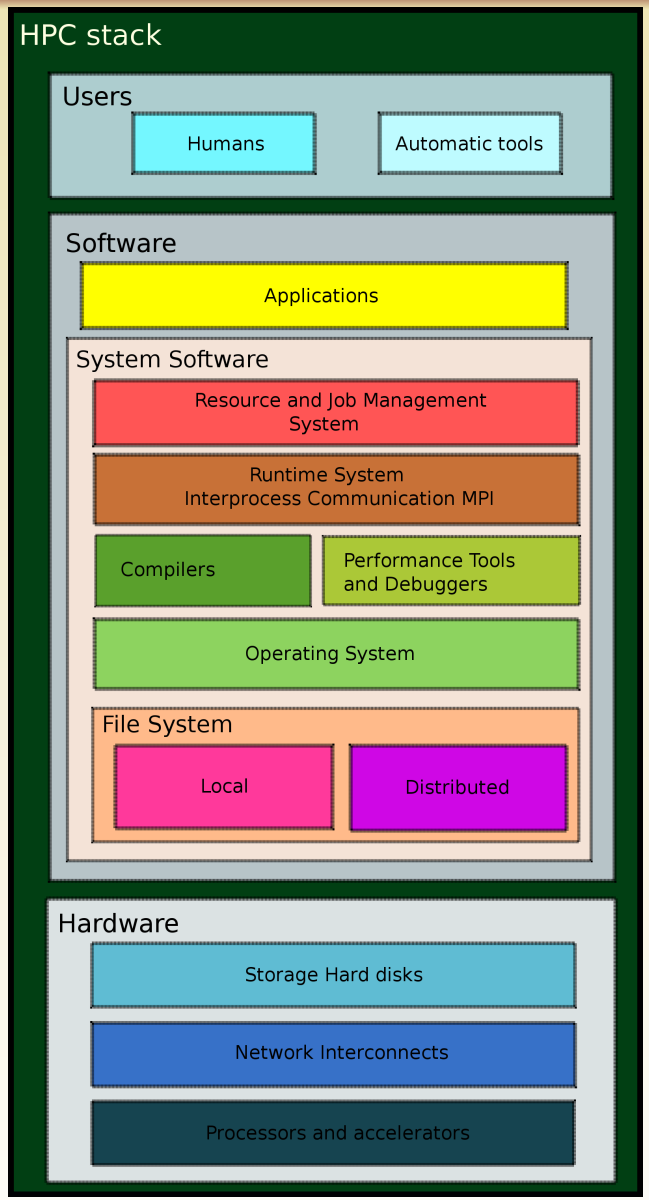
- My organization have such an infrastructure
  - Nice, but can I get all the platform for my experiments?
  - If so, how frequently?
  - Cannot spend too much time monopolizing the platform
- My organization doesn't have enough resources for that
  - How to do???

Petascale Context

Large scale computing resources

Goals

- Study the architecture as a whole
  - physical infrastructure
  - software
  - users
- Understand the global behavior to diagnose problems



Petascale Context

Constraints

- thousands of compute nodes to manage
- high occupation rate of the resources
- failures
- IO congestion: FS / network

Global study of the platform will consider

- Resource and Job Management System
- File System (Distributed)
- user's applications

Need

- **having some knowledge of the applications requirements**
- how they react to IO variation (in terms of performance)

Outline

Performance Evaluation

Challenges when evaluating a platform's performance

- The study of HPC systems depends on a large number of parameters and conditions
- Need: ease experimentation ▸ Kameleon
  - reproducibility of the results
  - recreate experiment's environment
- How to do the experiment?
- How to evaluate per se?
- Analysis: log results, experiment's condition, environment setup and parameters

Preliminary results

Experiment conduct platform

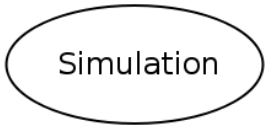
Kameleon tool: recipes to recreate a software environment ▸ Kameleon

- RJMS: OAR, SLURM
- File System: Lustre, NFS
- Evaluation tools: esp, xionee
- Export to several output formats: kvm, g5k

How to do the experiment?

Different Experimental Methodologies

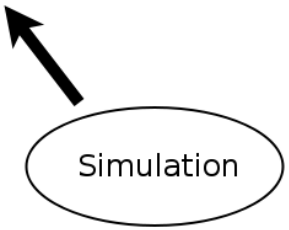
How to do the experiment?



Different Experimental Methodologies

How to do the experiment?

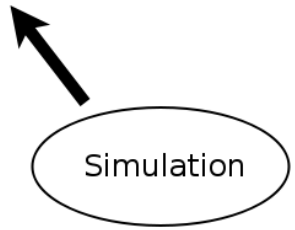
pros	cons
- Control of experiments and its conditions - Ease of reproduction	- Not real life complexity - Loss of capturing the unforeseen



Different Experimental Methodologies

How to do the experiment?

pros	cons
- Control of experiments and its conditions - Ease of reproduction	- Not real life complexity - Loss of capturing the unforeseen

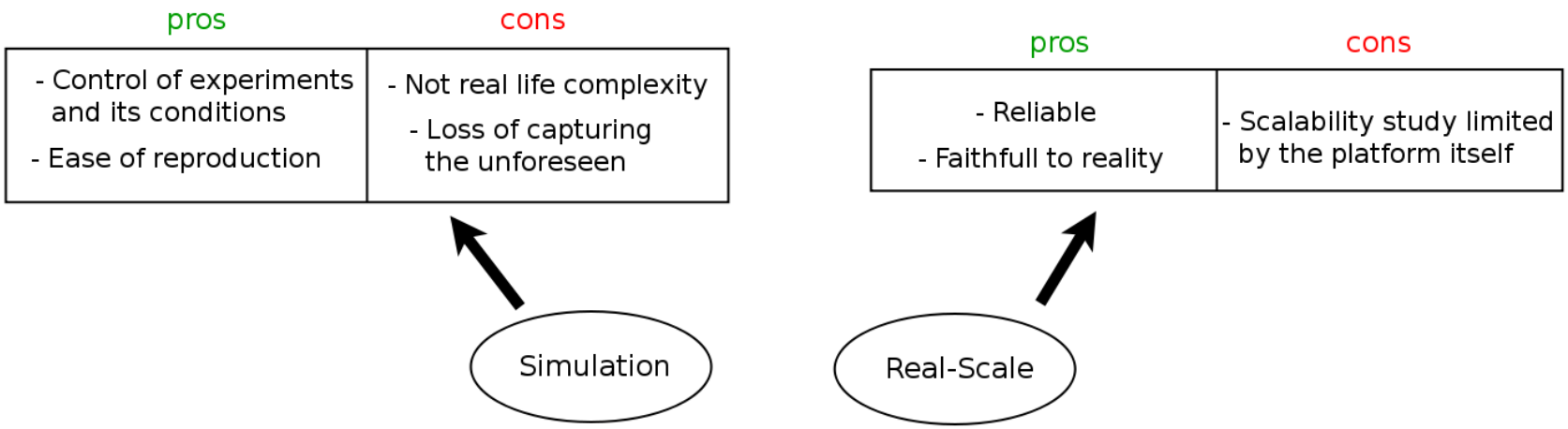


Simulation

Real-Scale

Different Experimental Methodologies

How to do the experiment?

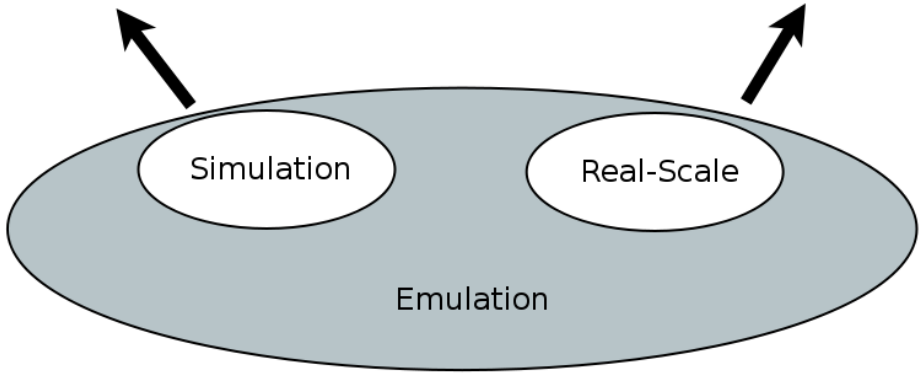


Different Experimental Methodologies

How to do the experiment?

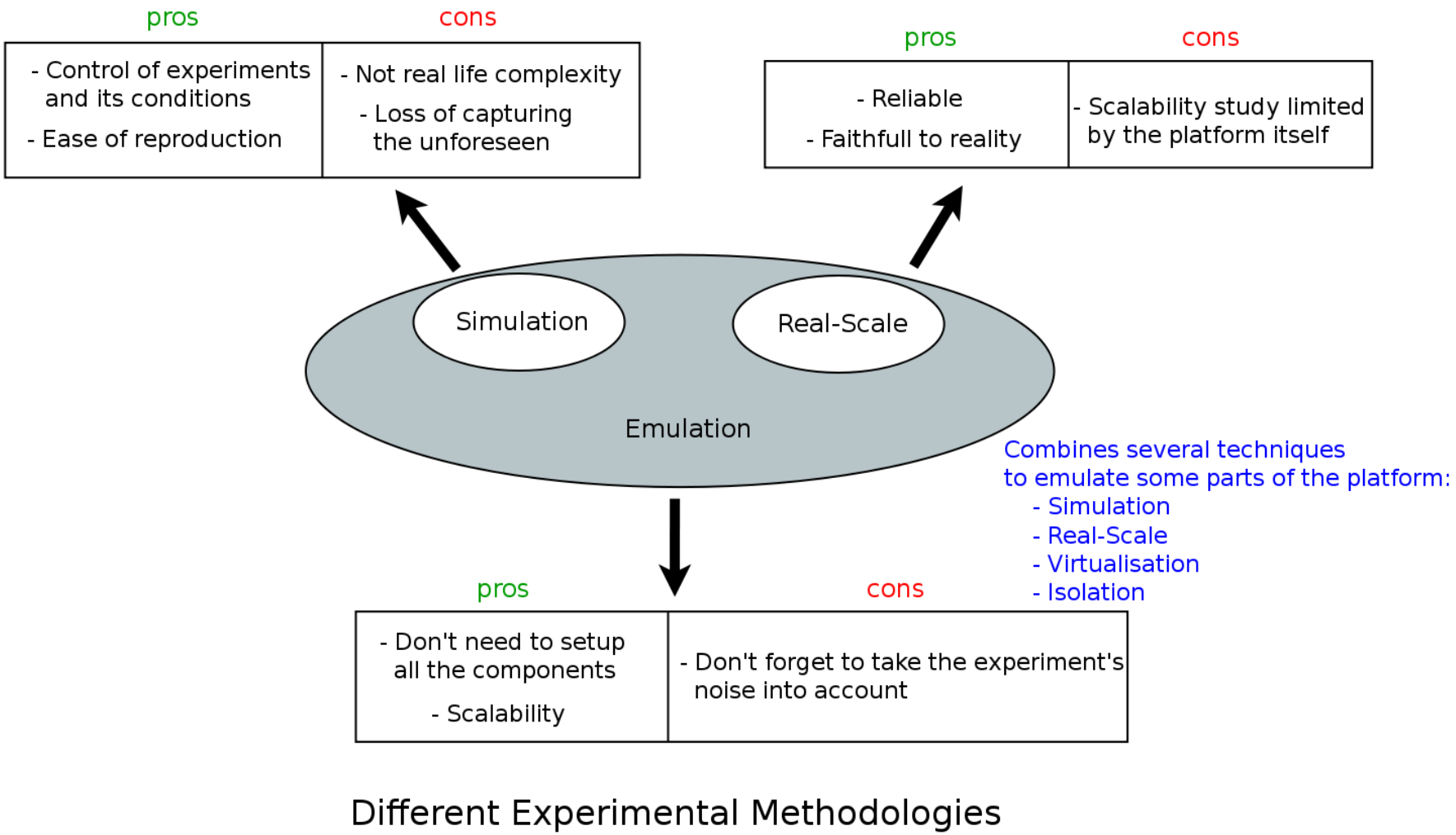
pros	cons
- Control of experiments and its conditions - Ease of reproduction	- Not real life complexity - Loss of capturing the unforeseen

pros	cons
- Reliable - Faithfull to reality	- Scalability study limited by the platform itself



Different Experimental Methodologies

How to do the experiment?



Different Experimental Methodologies

Evaluating scalability

How to evaluate large scale performance?

- Not enough physical resources for that
- Use of emulation to virtually enlarge our infrastructure
- Keep in mind: valuing the noise generated by the experiments
- But also by the emulation itself: performance loss

Proposed Experimental Methodology for Performance Evaluation

- From a physical infrastructure
  - Optionnal: emulate to make it bigger and evaluate scalability
- Inject load on the RJMS and File System
  - from real-life experiments - traces
  - from benchmark patterns - synthetic workloads
- Evaluate the system's load and responsiveness
- Log the experiment's condition and results in a "notebook" (like physicists do)
- Analysis, comparison with other experiments

Modelling the Workload

Jobs Workload modelling

- Performance evaluation by executing a sequence of jobs
- Two common ways to use a workload for system evaluation
  - either a *workload log* (like the ▸ SWF Workload Format)
  - or a *workload model* (synthetic workload like ▸ ESP benchmark [Kra08])

File System activity modelling

- Performance evaluation by executing an IO pattern
- Same possibilities
  - either an *IO log*
  - or an *IO model* (patterns in IOs benchmarks like IOR [SAS08])

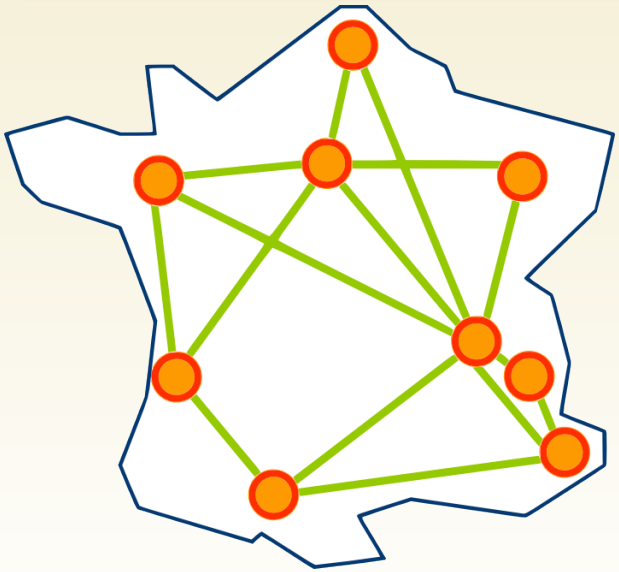
Modelling both the Jobs and FS Workload: Difficulties

- correlating jobs workload logs with IO logs
- smartly mixing patterns
- stage-in mechanism, application checkpoint

Tools for the experiments

Grid'5000

- INRIA-CNRS
- 10 sites: 9 in France, 1 in Brazil
- more than 5000 cores
- highly reconfigurable experimental platform



TGCC

- CEA
- equivalent to CEA's Tera100 (military classified)
- 60MW
- petaflop
- 7774 square yds building

Experiments: first step and later

First Step

- experiment separatly for reference
  - benchmark several RJMS with esp2
  - benchmark several Distributed FS with IOR, GoFilebench, IOZone
- mix esp2 jobs and FS benchmarks IO patterns
- compare

What next

- use real RJMS and IO traces: correlation?
- from jobs patterns to jobs workload patterns
- use emulation for scalability testing
- compare results emulated vs normal
- evaluating the experiment's "noise" on the different cases:
  - emulated or not
  - RJMS - IO

Roadmap

Goal

Identify the problems one can encounter when doing an experimental study in this domain

- Experiment with synthetic traces (esp - FS benchmarks) ≈
- Tool to help reconstructing experiments environments ✓
- Correlate FS traces with jobs traces ≈
- Getting some complete real traces x
- Playing with emulation ≈
- Performance evaluation framework x
- Follow the idea of [UAUS10] to partition the Bandwith and reserve dynamically the resources x
- ...

Thank you for your attention

Questions / Comments ?

Bibliography


William TC Kramer.
PERCU: A Holistic Method for Evaluating High Performance Computing Systems.
PhD thesis, EECS Department, University of California, Berkeley, Nov 2008.


Hongzhang Shan, Katie Antypas, and John Shalf.
Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark.
In *SC*, page 42, 2008.


Andrew Uselton, Katie Antypas, Daniela Ushizima, and Jeffrey Sukharev.
File system monitoring as a window into user i/o requirements.
In *CUG*, 2010.

Outline

Experimenting

How to ensure our experiments are valuable?

▸ Performance ▸ Results

Reproducibility in experiments

Reproducibility

- Of the results
  - need to reproduce the experiments

Experiment

- **One** definition: *"A test under* **controlled conditions** *that is made to demonstrate a known truth, examine the validity of a hypothesis, or determine the efficacy of something previously untried."*
  (http://www.thefreedictionary.com/experiment)

Reproducibility in experiments

Experiment

- Observe the experiment conditions:
  - the Object (experiment plan: parameters, configuration...)
  - and its environment: software and hardware
- Need of a **complete** environment
- Experiments conditions determine experiment itself
- Reproducibility: need to recreate the environment (at least software)

$$\Downarrow$$

Need an environment model!

Experiments reconstructability

Experiment depends on the environment.

Reconstructability

- Replay the experiments in the same conditions
- Recreate the experiment's environment
  - Hardware: hard
  - Software: ok

Software environment

Softwares evolve going forward

- Software versions
- configurations

A Tool to Generate Software Appliances

Kameleon: a tool to generate software appliances

A tool that generates software appliances from an environment description:

- Recipe (high level)
- Steps (mid and low-level)

One can combine this to:

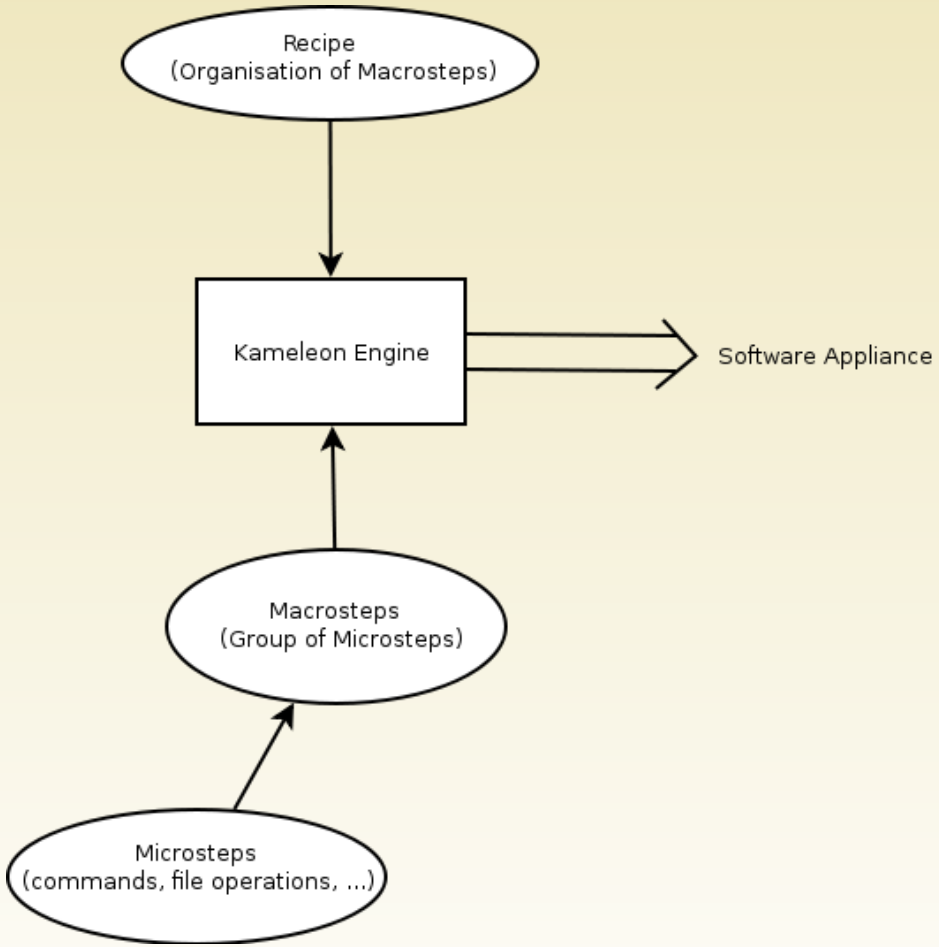
- Model the software environment: recreate it ad libitum
- Everything described: keep log of what has been done

Goals:

- Simple
- Easy to handle

Kameleon: a tool to generate software appliances

Basics



Recipes and Steps

Debian_recipe.yaml

```
global:
  distrib: debian
  debian_version_name: squeeze
  distrib_repository: http://ftp.fr.debian.org/debian/
  arch: amd64
  kernel_arch: "amd64"
  #checkpoint_file: /var/tmp/checkpoint.tgz

steps:
  - debian_check_deps
  - check_deps:
      - rsync
      - building_appliance
      - building_kvm_images
      - bootstrap
  - system_config
  - root_passwd
  - mount_proc
  - software_install:
      - extra_packages
  - kernel_install
  - strip
  - umount_proc
  - build_appliance:
      - clean_udev
      - create_raw_image
      - create_nbd_device
      - mkfs
      - mount_image
      - copy_system_tree
      - install_grub
      - umount_image
      - save_as_raw
  - clean
```

Environment Variables

Checkpointing

bootstrap.yaml

```
bootstrap:
  - debootstrap:
      - exec_appliance:  debootstrap --arch=$$arch
                         $$debian_version_name
                         $$chroot/ $$distrib_repository
```

build_appliance.yaml

```
build_appliance:
  - clean_udev:
      - exec_appliance:     rm -f etc/udev/rules.d/*persistent-net.rules*
  - save_as_tgz:
      - exec_current:       cd $$workdir; tar -czf $$distrib.tar.gz -C chroot .
  - create_raw_image:
      - check_cmd:          /sbin/sfdisk
      - exec_appliance:     dd if=/dev/zero of=$$workdir/image.raw bs=1M count=1 seek=5000
      - exec_appliance:     echo -e "1,574,83,*\n575,,82" | /sbin/sfdisk $$workdir/image.raw
```

Kameleon history

- Initially: create appliances for OAR testing
- 2^{nd} step: make it more generic
- 3^{rd} step: provide basic steps
- 4^{rd} step: provide specific steps (G5K, OAR, SLURM, ESP2, Xionee...) *In progress*
- 5^{th} step: world contamination...

Kameleon cool features

- Checkpoint
- Embedded Shell
- Idempotent (depends of the command itself)
- Stand alone
- Several abstraction levels (Recipes, Macrosteps, Microsteps)
- Simple but powerfull

Kameleon more technical features: commands

- File operations: append, create
- Macrosteps dependencies
- Breakpoints
- Check of the presence of a command
- Execution of a command

Kameleon Bonus

Friendliness: embedded shell

- history
- colored
- manual step execution
- retry on error
- environment variables

Hooks

- customizable clean

Kameleon Bonus

YAML powered

- indentation
- code readability

Provided ingredients

- appliance with kameleon preinstalled
- default recipes/steps to use/play with
  - Debian
  - Redhat
  - Grid5000

Back

Appendix

Standard Workload Format

Definition of SWF format to describe the execution of a sequence of jobs.

```
; Computer: Linux cluster (Atlas)
; Installation: High Performance Computing - Lawrence Livermore National Laboratory
; MaxJobs: 60332
; MaxRecords: 60332
; Preemption: No
; UnixStartTime: 1163199901
; StartTime: Fri Nov 10 15:05:01 PST 2006
; EndTime:   Fri Jun 29 14:02:41 PDT 2007
; MaxNodes: 1152
; MaxProcs: 9216
; Note: Scheduler is Slurm (https://computing.llnl.gov/linux/slurm/)
; MaxPartitions: 4
; Partition:  1 pbatch
; Partition:  2 pdebug
; Partition:  3 pbroke
; Partition:  4 moody20
;     j|       s|      w|      r|   p|   c|    m|   p|      u| m| s|  u|  g|  e| q| p| p| t
;     o|       u|      a|      u|   r|   p|    e|   r|      s| e| t|  i|  i|  x|  | a| r| h
;     b|       b|      i|      n|   o|   u|    m|   o|      e| m| a|  d|  d|  e| n| r| e| i
;      |       m|      t|      t|   c|    |     |   c|      r|  | t|   |   |   | u| t| v| n
;      |       i|       |      i|    |   u|    u|    |       | r| u|   |   |  n| m| i|  | k
;      |       t|       |      m|   a|   s|    s|   r|      e| e| s|   |   |  u|  | t| j|
;      |        |       |      e|   l|   e|    e|   e|      s| q|  |   |   |  m|  | i| o| t
;      |        |       |       |   l|   d|    d|   q|      t|  |  |   |   |   |  | o| b| i
;      |        |       |       |   o|    |     |    |       |  |  |   |   |   |  | n|  | m
;      |        |       |       |   c| sec|   Kb|    |       |  |  |   |   |   |  |  |  | e
   2488  4444343       0    8714 1024   -1    -1 1024      -1 -1  0  17  -1 307 -1  1 -1 -1
   2489  4444343       0  103897 1024   -1    -1 1024      -1 -1  1  17  -1 309 -1  1 -1 -1
   2490  4447935       0     634 2336   -1    -1 2336   10800 -1  1   3  -1   5 -1  1 -1 -1
   2491  4448583       0     792 2336   -1    -1 2336   10800 -1  1   3  -1   5 -1  1 -1 -1
   2492  4449388       0     284 2336   -1    -1 2336   10800 -1  0   3  -1   5 -1  1 -1 -1
```

→ But lack of information about IOs: FS, network

Back

ESP2 Benchmark

- provide a **quantitative evaluation** of launching and scheduling via a single metric: time
- Complete **independence** from the hardware performance
- Ability for **scalability** evaluation of the RJMS

$$ESPEfficiency = \frac{TheoreticDuration}{MeasuredDuration} \quad (1)$$

Job Type	Fraction of Job Size relative to total system size	Job size for a 512cores cluster (in cores)	Count of the number of job instance	Target Run Time (Seconds)
A	0.03125	16	75	267
B	0.06250	32	9	322
...	...	...	...	...
L	0.12500	64	36	366
M	0.25000	128	15	187
Z	1.00000	512	2	100
Total			230	

Table: ESP2 benchmark [Kra08] characteristics for a 512 cores cluster

Courtesy of Yiannis Georgiou

Back

ESP2 Benchmark

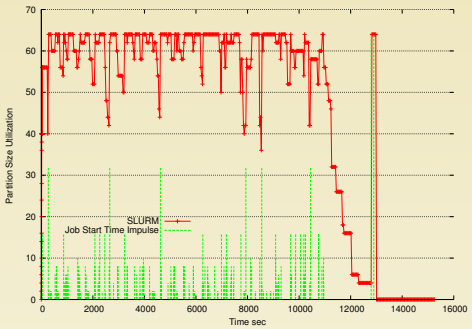


Figure: ESP2 benchmark for SLURM RJMS with 64 resources - no Z jobs

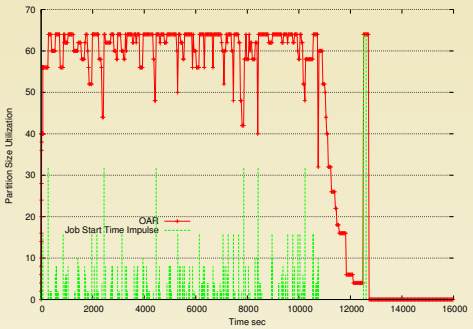


Figure: ESP2 benchmark for OAR RJMS with 64 resources - no Z jobs

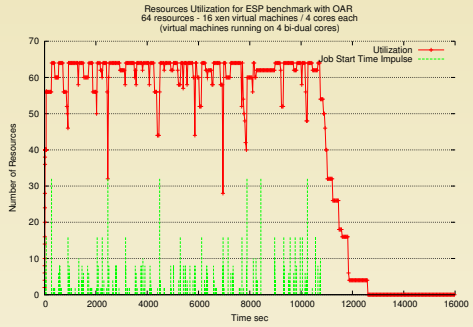


Figure: ESP2 benchmark for OAR RJMS with 64 virtual (xen) resources - no Z jobs

Comparison between Slurm, OAR and OAR in a virtual cluster - 64 resources

Back

ESP2 Benchmark

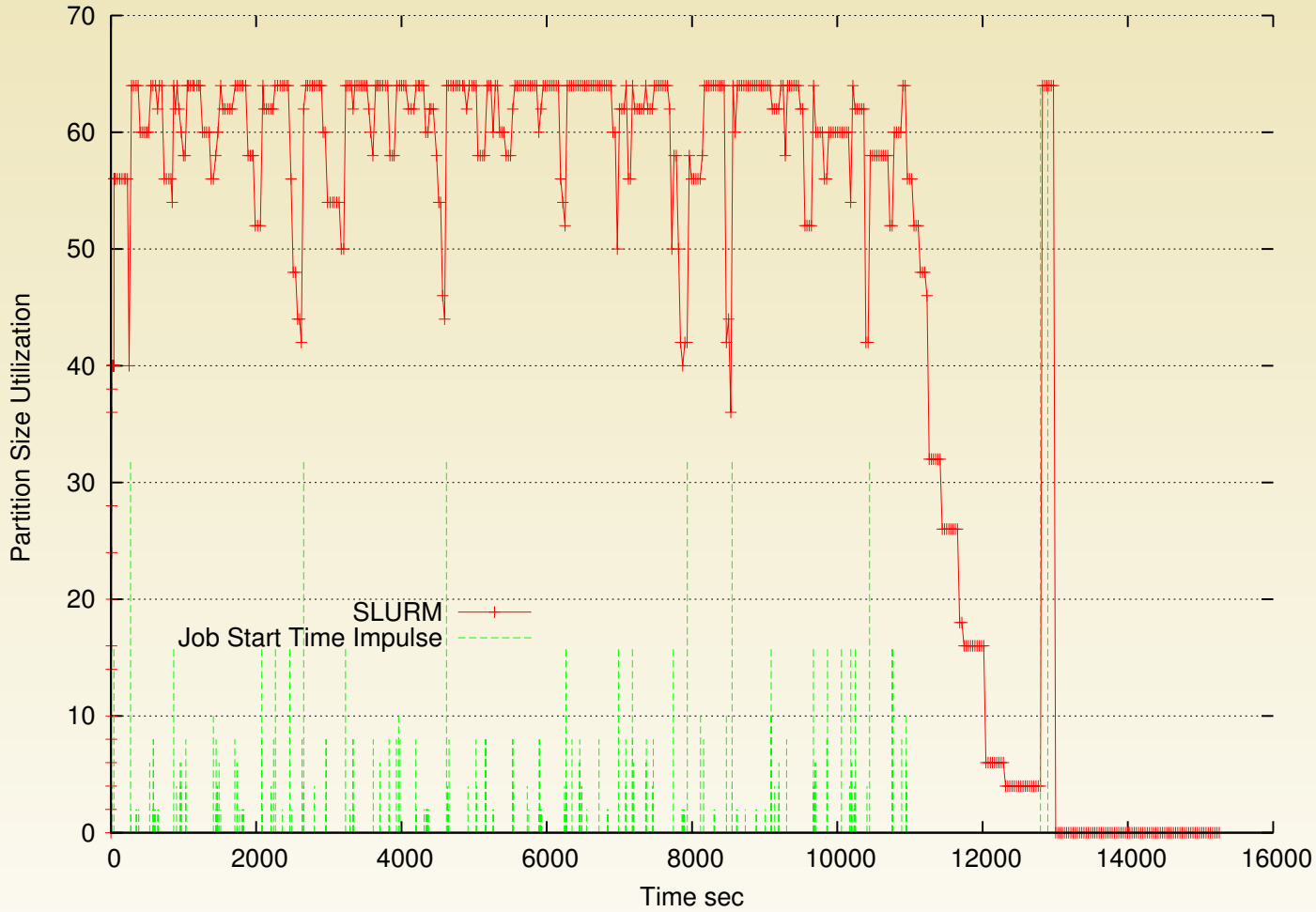


Figure: ESP2 benchmark for SLURM RJMS with 64 resources - no Z jobs

Back

ESP2 Benchmark

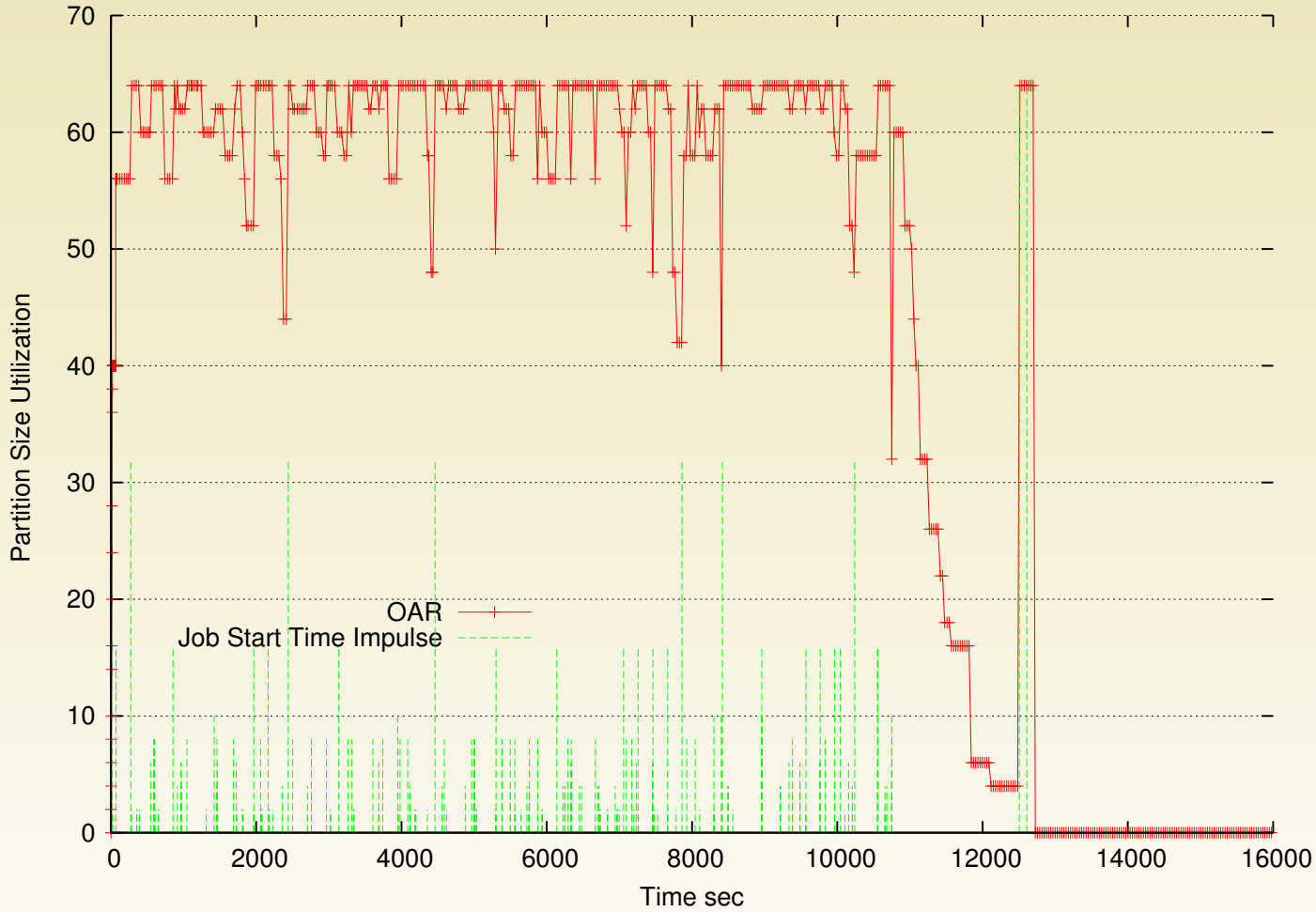


Figure: ESP2 benchmark for OAR RJMS with 64 resources - no Z jobs

Back

ESP2 Benchmark

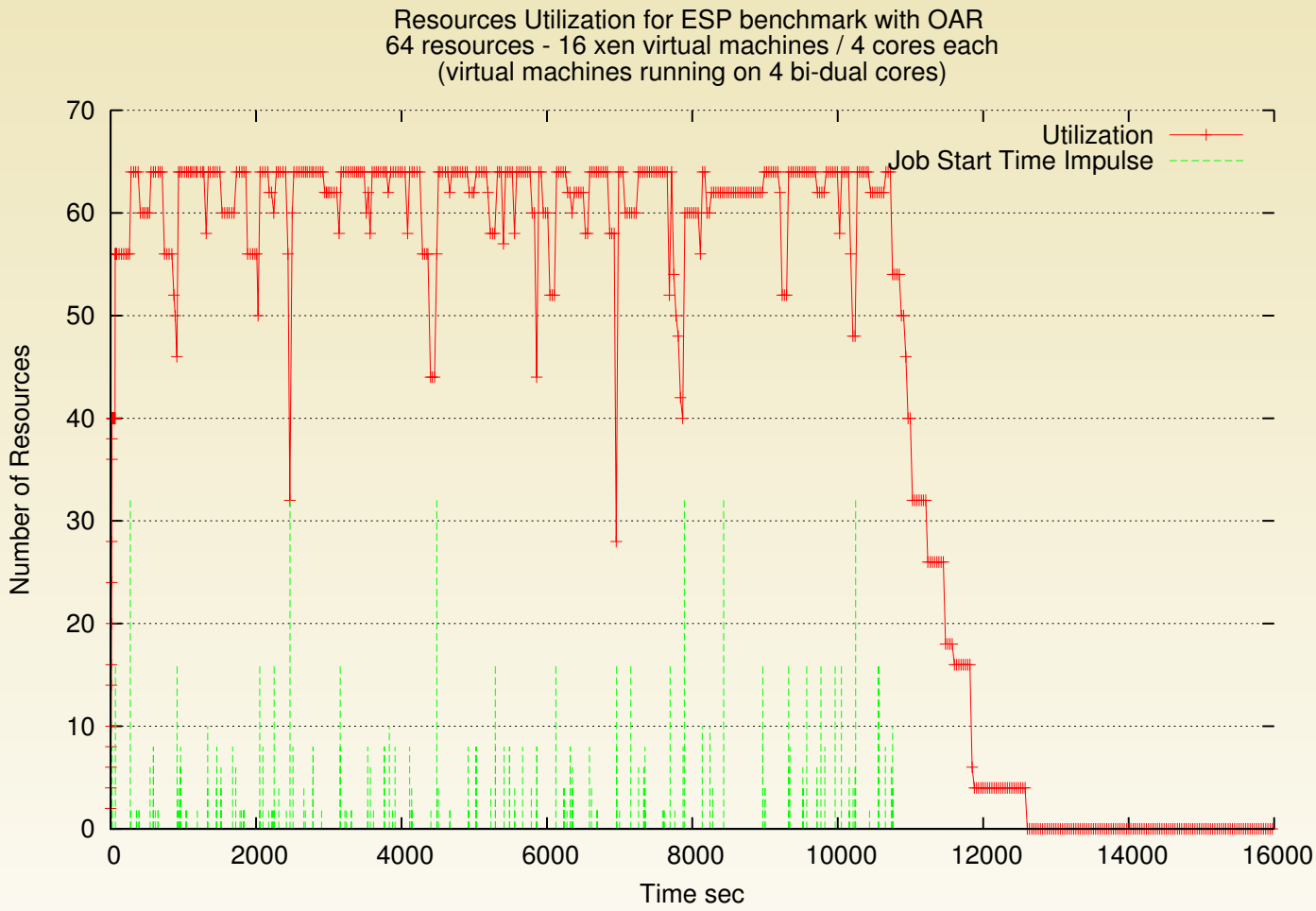


Figure: ESP2 benchmark for OAR RJMS with 64 virtual (xen) resources - no Z jobs

Back

ESP2 Benchmark

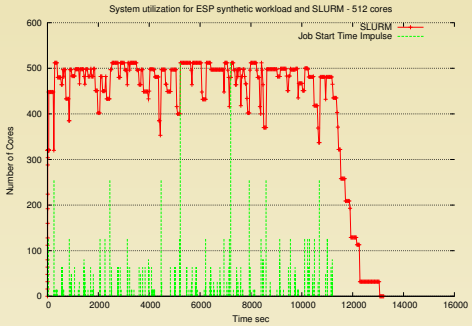


Figure: ESP2 benchmark for SLURM RJMS with 512 resources

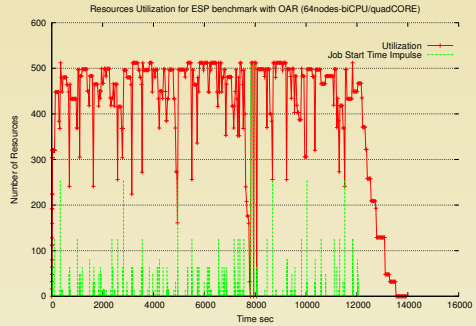


Figure: ESP2 benchmark for OAR RJMS with 512 resources

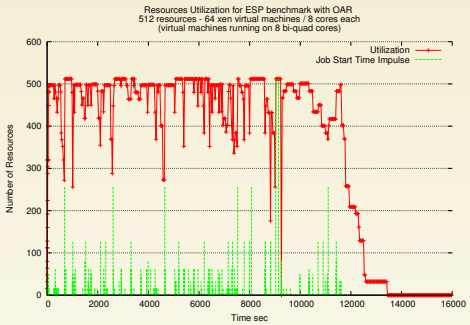


Figure: ESP2 benchmark for OAR RJMS with 512 virtual (xen) resources

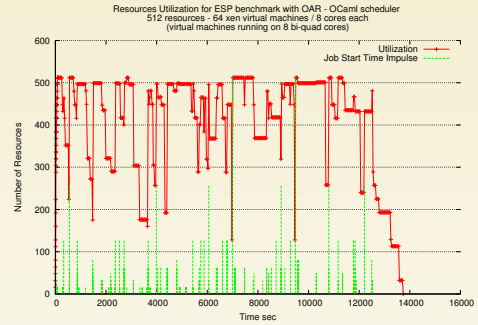


Figure: ESP2 benchmark for OAR RJMS with 512 virtual (xen) resources - OCaml scheduler

Comparison between Slurm and OAR - 512 resources

Back

ESP2 Benchmark

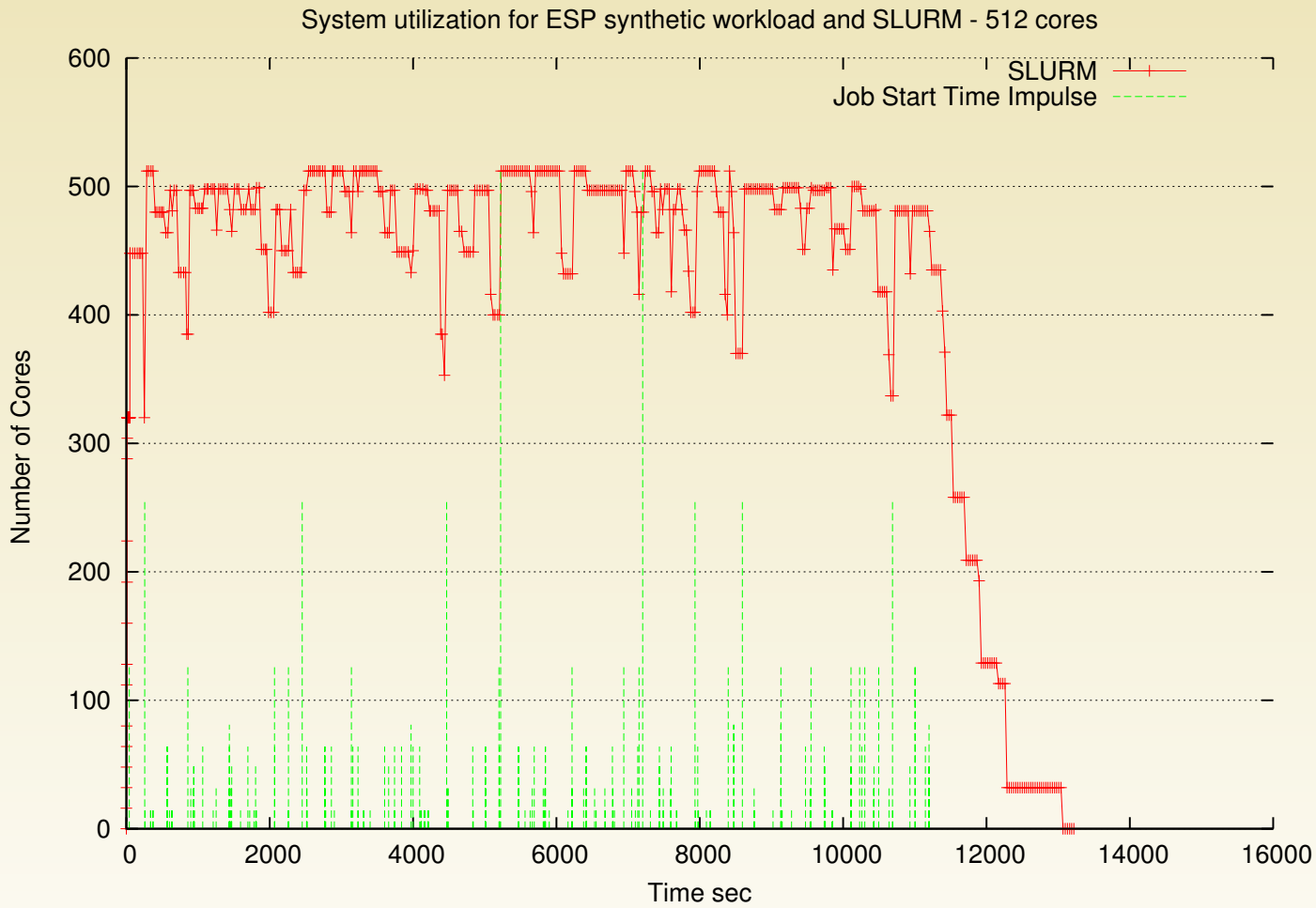


Figure: ESP2 benchmark for SLURM RJMS with 512 resources

Back

ESP2 Benchmark

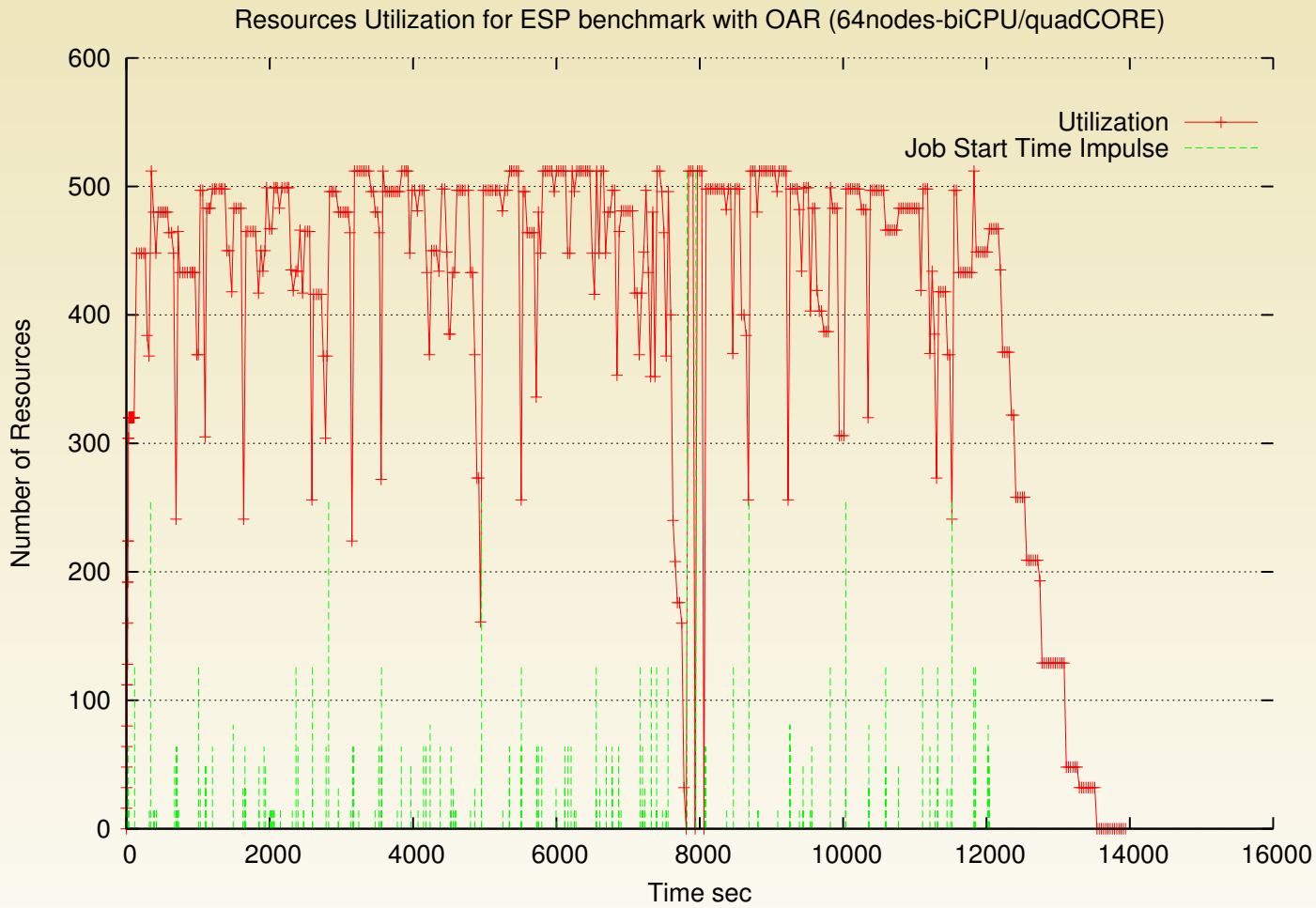


Figure: ESP2 benchmark for OAR RJMS with 512 resources

Back

ESP2 Benchmark

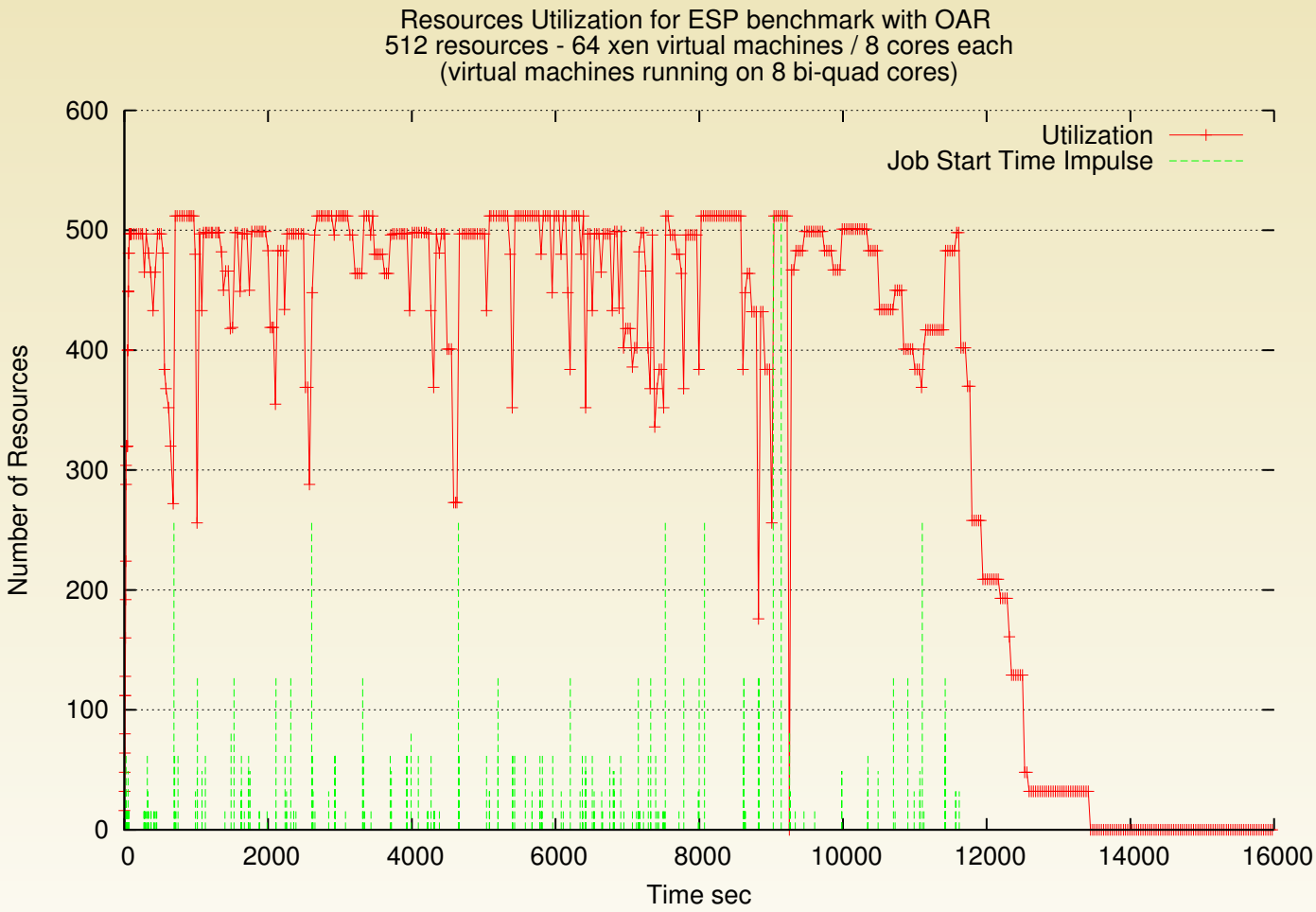


Figure: ESP2 benchmark for OAR RJMS with 512 virtual (xen) resources

Back

ESP2 Benchmark

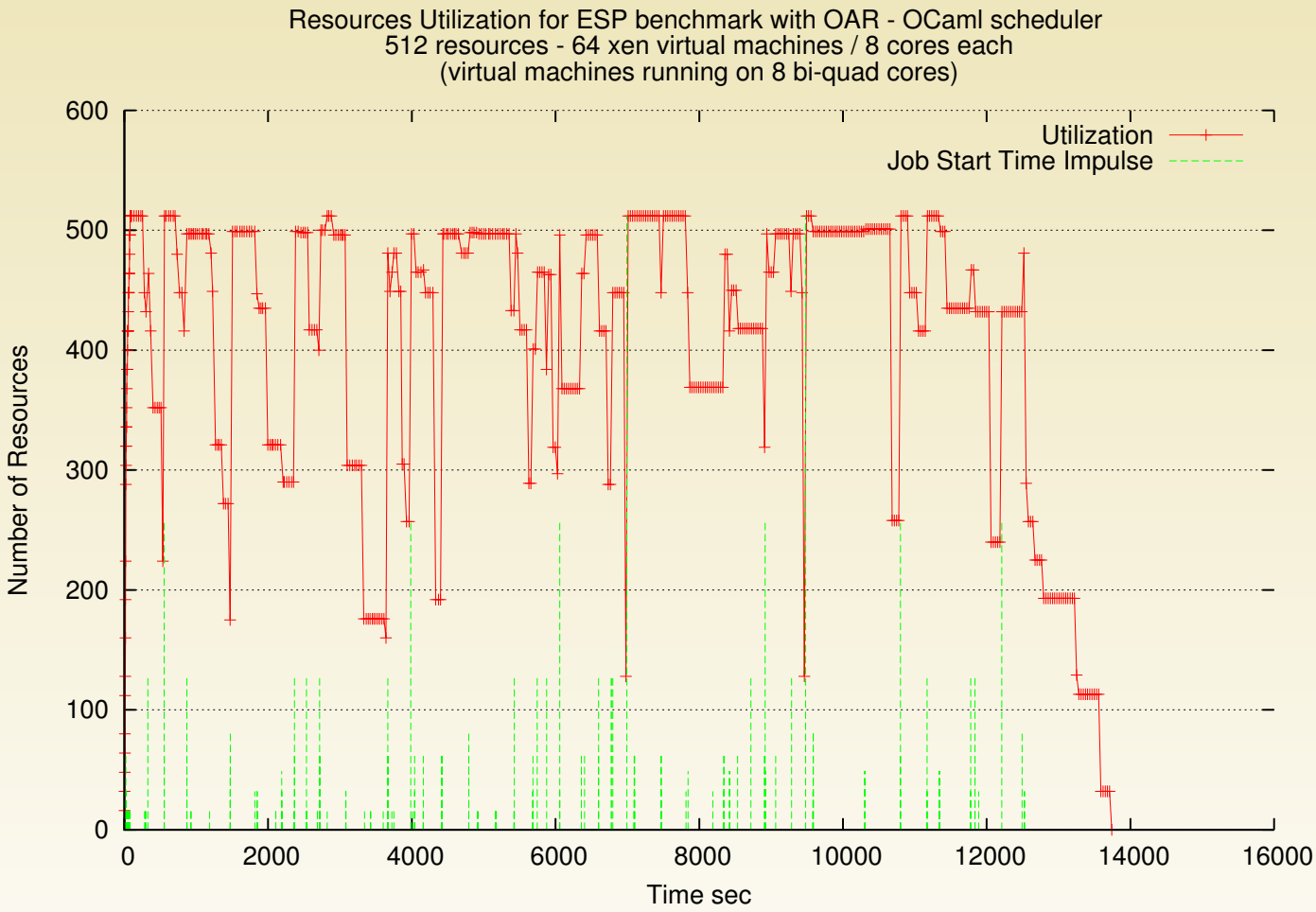


Figure: ESP2 benchmark for OAR RJMS with 512 virtual (xen) resources - OCaml scheduler

Back

[UAUS10]

In [UAUS10], NERSC experimented on the Cray XT4 that, statically partitioning their File System (Bandwith) in two parts:

- one dedicated to "big IOs" users
- one for the "regular" users

lead to improving the overall performance and decreasing the variability in IO performance.

Back