Enhancing the Efficiency of Resource Usage on Opportunistic Grids

7th International Workshop on Middleware for Grids, Clouds and e-Science – MGC 2009

Raphael de A. Gomes
Fábio M. Costa
Fouad J. Georges

November, 2009

MIDDLEWARE 2009

InteGrade

Opportunistic Grids

Use the idle capacity of non-dedicated resources

- Usually, large amounts of resources can be harvested to run even high-performance applications
  - E.g., users' desktop machines, computer labs
- Virtually at no cost
- Condor, OurGrid, InteGrade

Similarly to voluntary computing, but in a managed way

The Problem

Opportunistic grids prioritize the local applications running on shared resources

- Best effort principle: When local apps need resources, grid apps are evicted or migrated to another node and possibly resumed from the last checkpoint

Rationale:

- A significant part of high resource usage events by local applications are temporary bursts
- It might be more effective for grid apps to wait for resources to turn available again than to migrate

Usual Approach

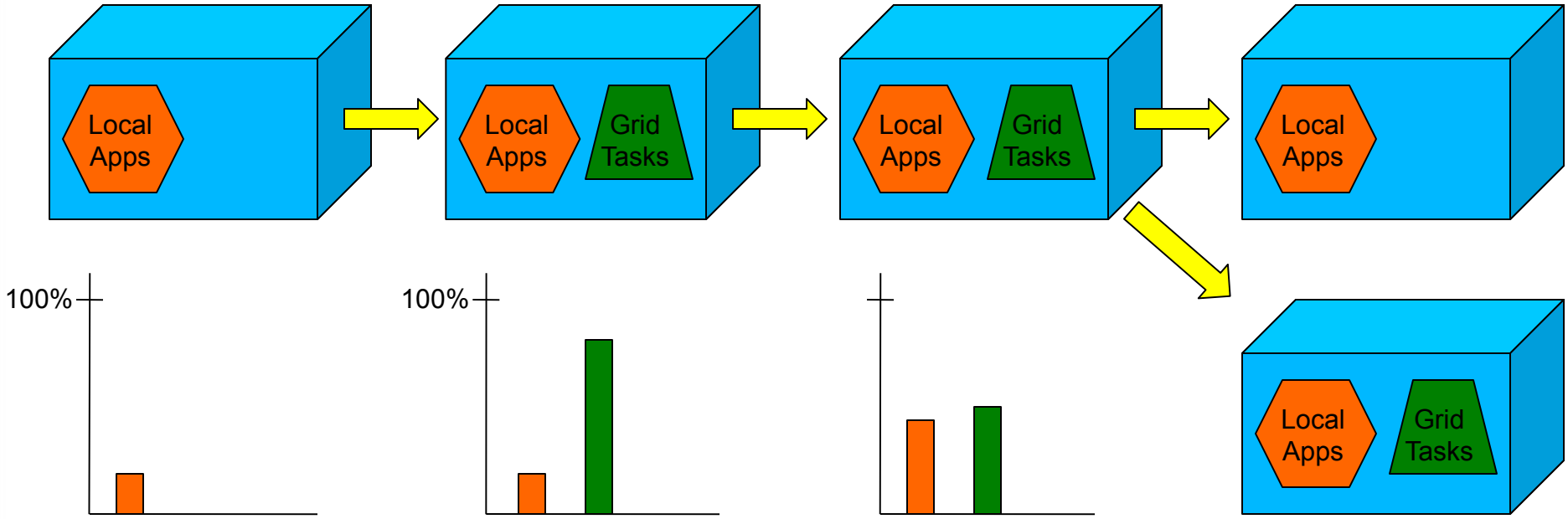
Base application schedule on resource usage profiles

- Effective when the use of grid resources strictly follows the profile

Problem:

- Profiles are based on coarse-grained statistics, such as averages
- They do not capture important, short-term behavior, such as resource usage bursts, which may be interpreted as the need to migrate grid apps

Usual Dynamics on Opportunistic Grids

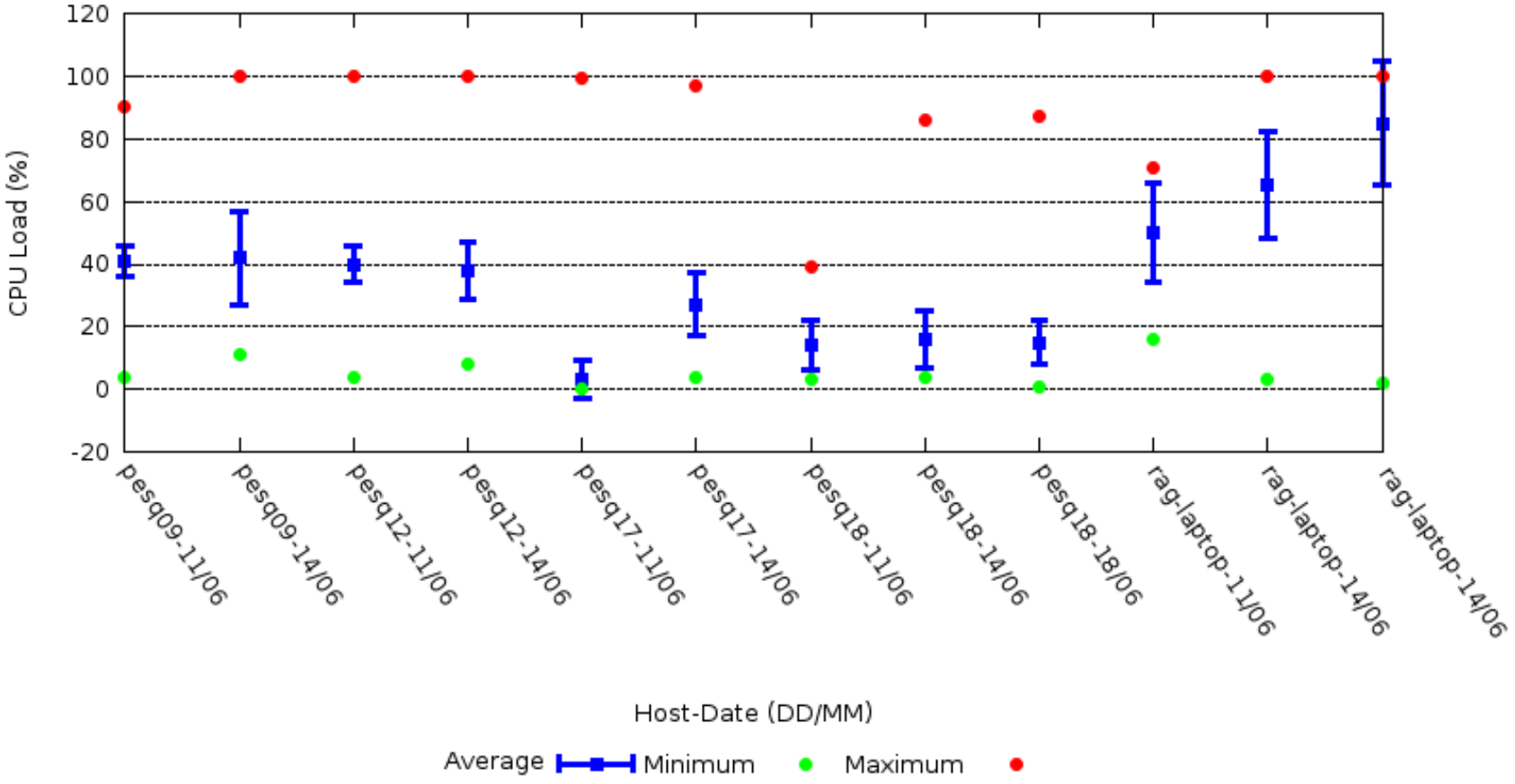


Problem with Averages

- Usage pattern analysis may predict that a machine is sufficiently idle, causing grid tasks to be scheduled for it
- However, bursts in CPU usage are very frequent and may be interpreted as sudden resource failures, causing task migration

We should be able to not only detect such bursts, but also to evaluate their duration

Problem with Averages



Proposed Approach

Resource usage burst analysis

- Predict the duration of resource usage bursts
- Determine if it's more cost-effective to wait for the resource to become available again instead of migrating a grid application's tasks
- Consider the cost of loosing all computations performed since the last checkpoint

Proposed Approach

Analysis of the execution pattern of individual local applications

- Sample the behavior of local applications on grid machines over an extended period

When a burst occurs:

- Identify which local apps are causing the burst
  - The ones that are most active at the moment
- Prediction of burst duration is based on the (possibly combined) behavior of such apps

Example

%	0 ⊢ 10	10 ⊢ 20	20 ⊢ 30	30 ⊢ 40	40 ⊢ 50	50 ⊢ 60	60 ⊢ 70	70 ⊢ 80	80 ⊢ 90
10 ⊢ 20	31.17 / 0								
20 ⊢ 30	18.78 / 2	13.78 / 2							
30 ⊢ 40	37.44 / 3	24.22 / 3	05.37 / 0						
40 ⊢ 50	31.80 / 3	20.53 / 2	09.77 / 2	03.77 / 2					
50 ⊢ 60	53.98 / 0	41.98 / 0	33.66 / 0	06.67 / 0	07.29 / 0				
60 ⊢ 70	53.27 / 12	52.66 / 11	45.75 / 11	09.03 / 11	08.08 / 8	06.71 / 0			
70 ⊢ 80	54.00 / 1	29.21 / 1	21.00 / 1	14.05 / 1	08.84 / 1	02.86 / 0	01.35 / 0		
80 ⊢ 90	31.00 / 0	29.00 / 0	25.97 / 0	20.17 / 0	18.24 / 0	09.98 / 0	02.66 / 0	01.15 / 0	
90 – 100	39.20 / 0	31.00 / 0	27.00 / 0	20.89 / 0	17.43 / 0	08.43 / 0	05.15 / 0	03.43 / 0	02.46 / 0

Average and minimum burst durations for a process running Firefox (in seconds)

The occurrence of short bursts is a common fact

Resource Usage Estimation

Estimate the duration of resource consumption peaks

This estimate is based on:

- the resource usage pattern of active local apps
- the percentage of resources required by grid applications
- system's current state with respect to
  - overall amount (%) of resource usage

Estimating Burst Duration

Two parameters:

- γ: Level of resource usage in the current burst
- δ: Target resource usage level (required by grid apps)

Determine (predict) how long it will take for the resource usage level to transition from γ to δ

Ex.: from 90%-100% to 10%-20%: 31 secs

Considering the mix of all active apps

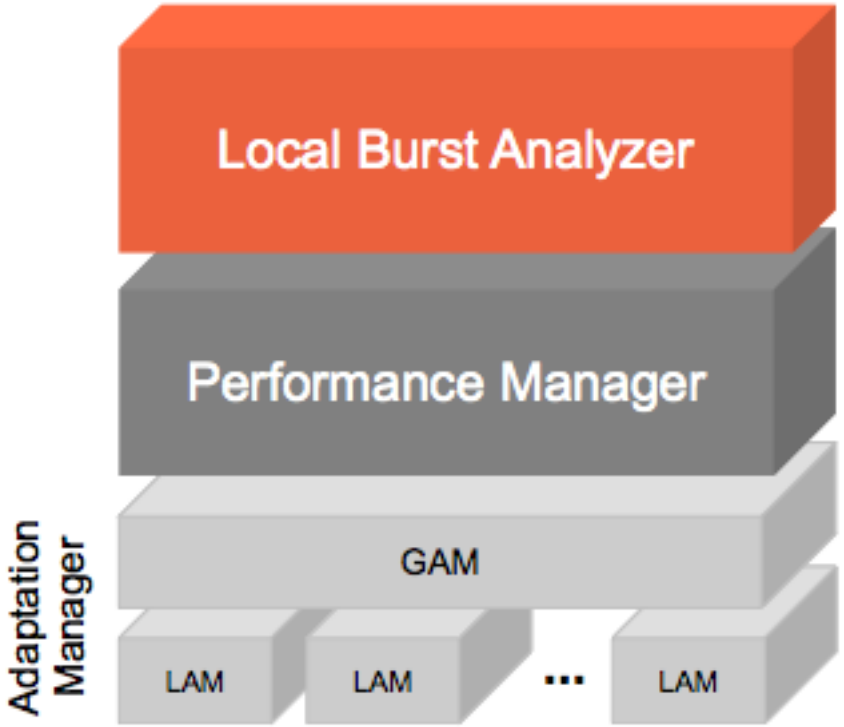
- Pessimistic algorithm: take the length of the largest burst among all analyzed apps
- Details about the algorithm in the paper (room for improvement)

Architecture

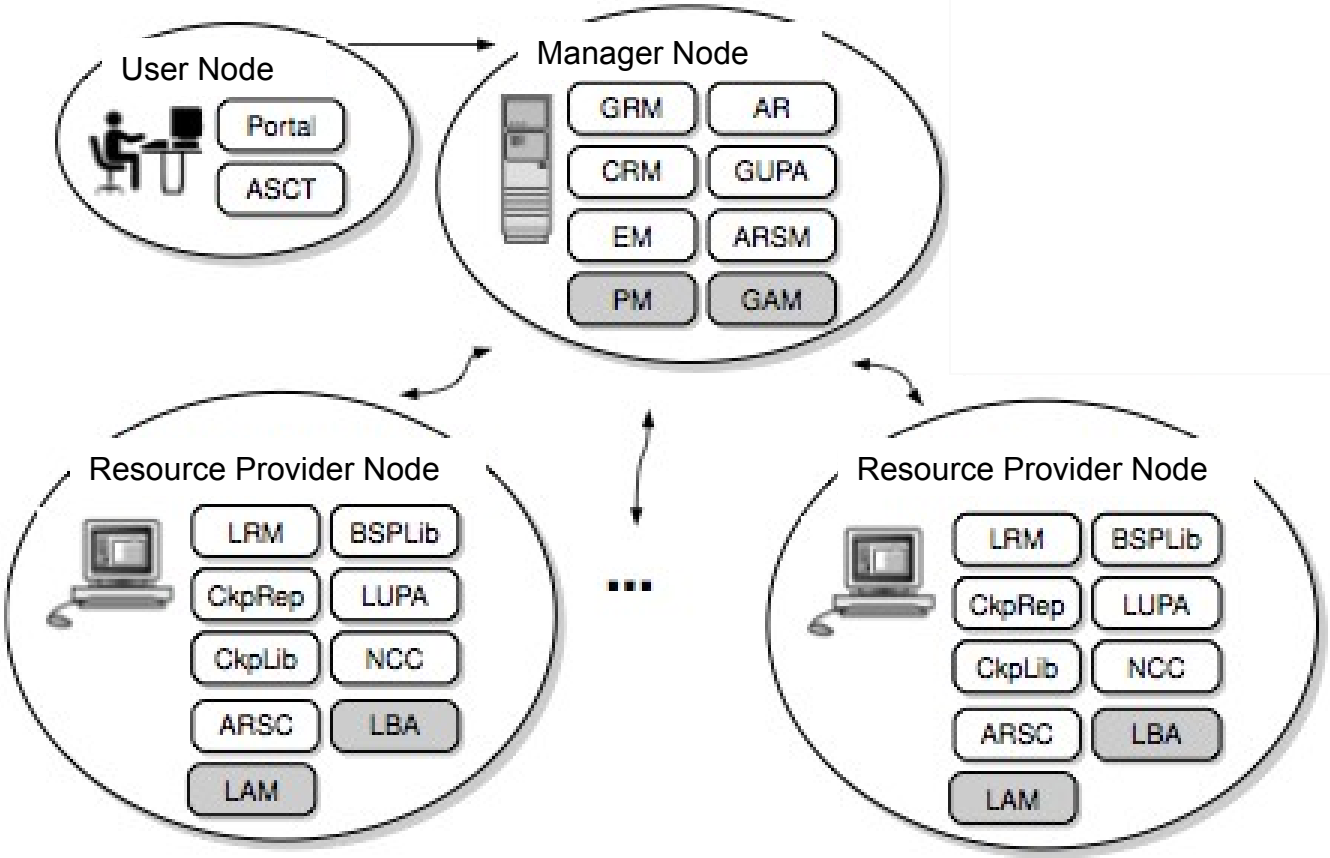
The approach requires the introduction of three new modules in the InteGrade architecture:

- Local Burst Analyzer (LBA)
- Performance Manager (PM)
- Adaptation Manager (AM)

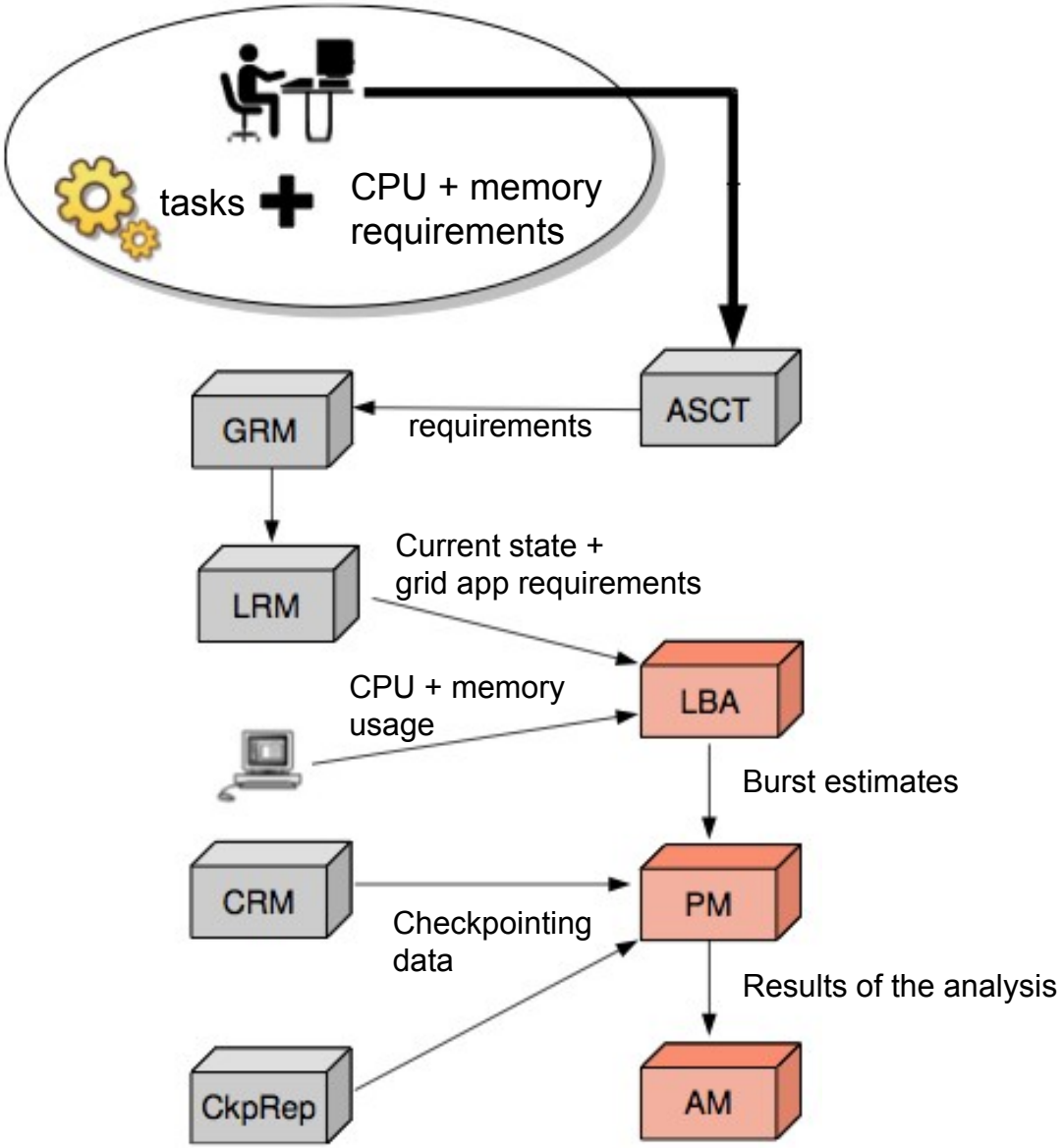
Architecture



As Part of InteGrade



As part of the InteGrade Architecture



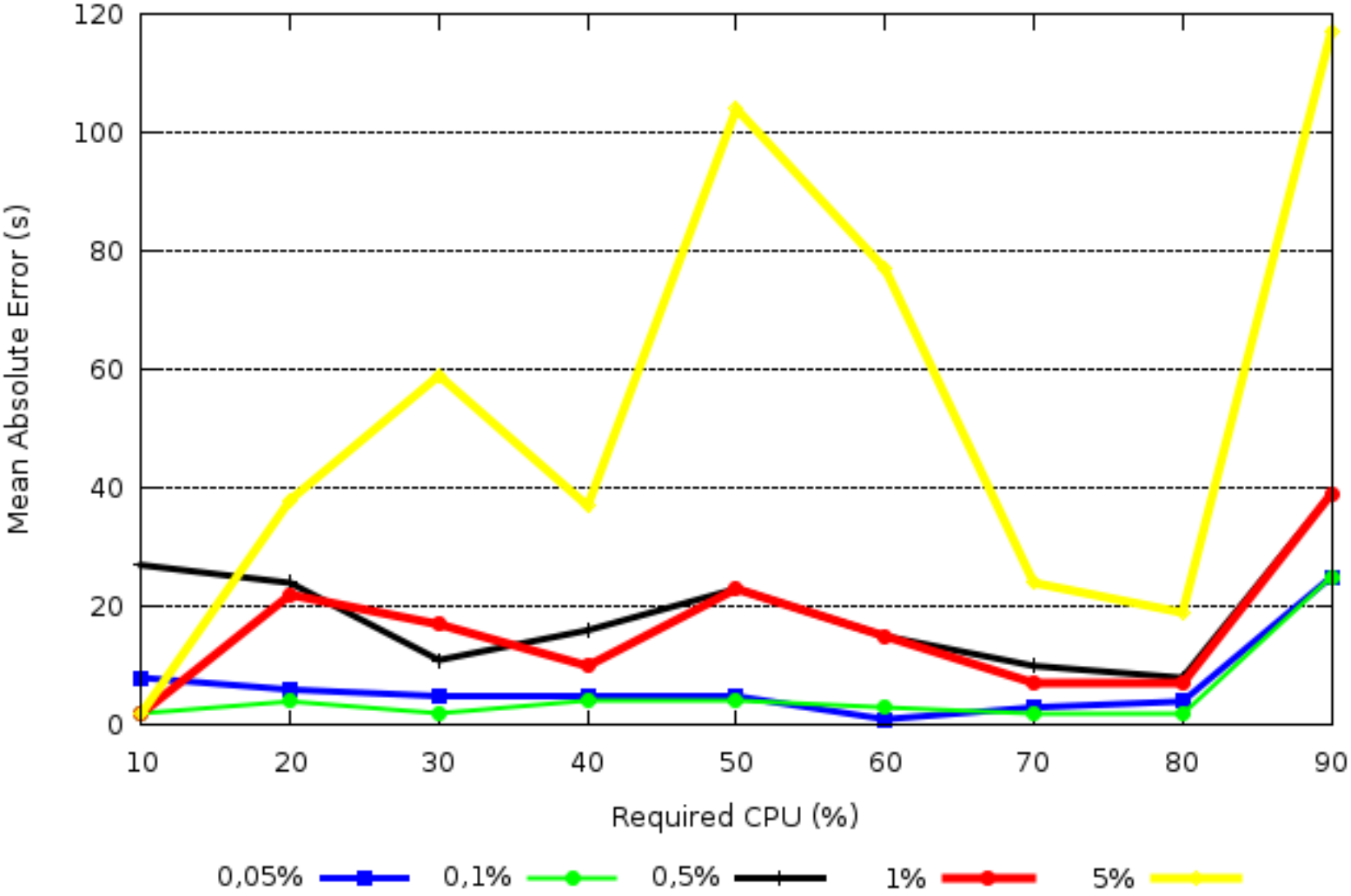
Evaluation

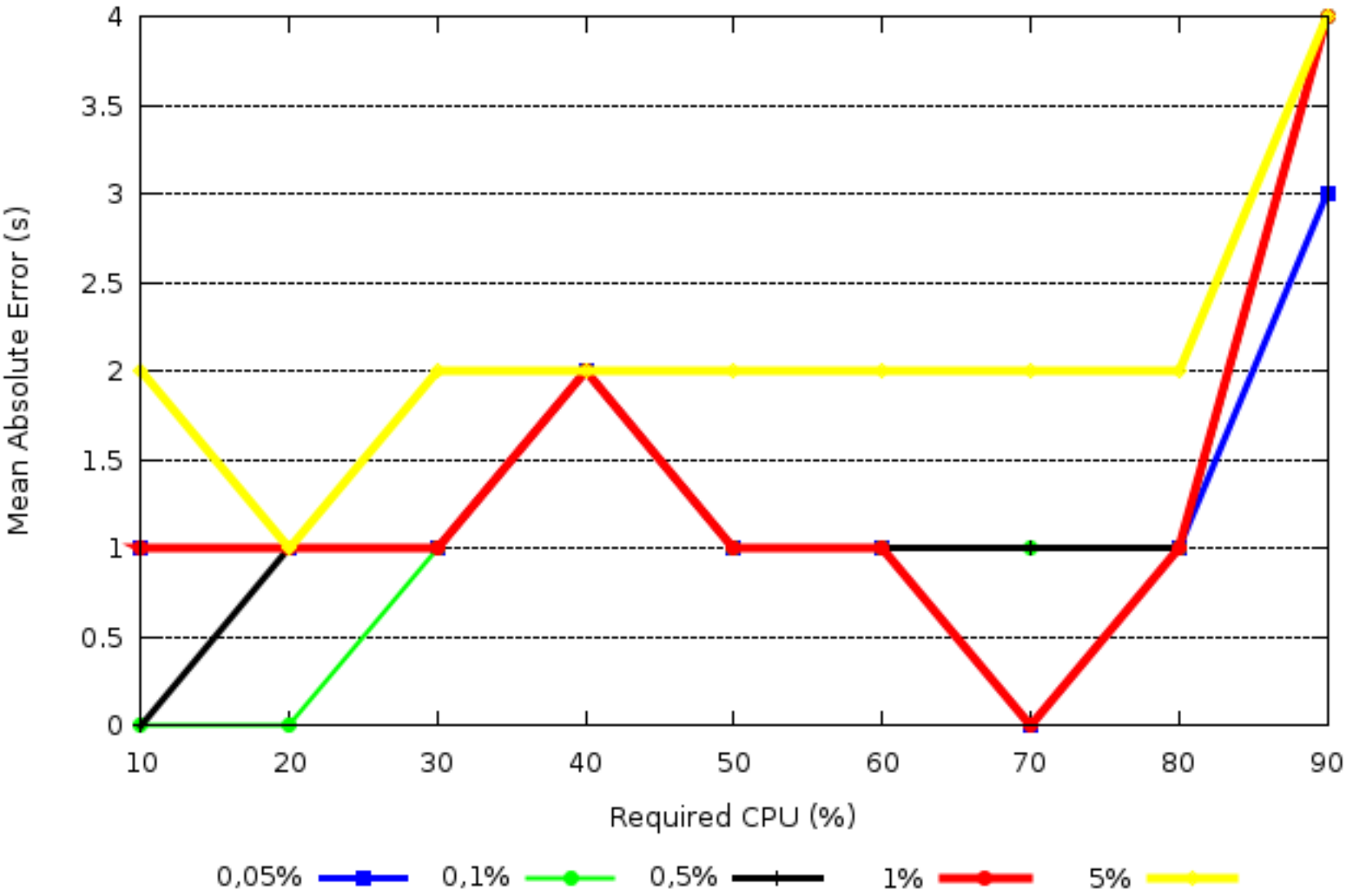
- Accuracy of burst duration prediction
- Overhead of burst analysis

Accuracy of Prediction

Methodology:

- Use resource usage data collected from real application executions to simulate realistic workload
- Use LBA to predict burst duration for a number of test cases
  - when grid apps request different amounts of CPU (10%, 20%, 30%, …, 90%)
- Compare the prediction with real burst duration
- Using different sample sizes
  - 0,05%, 0,1%, 0,5%, 1%, 5%

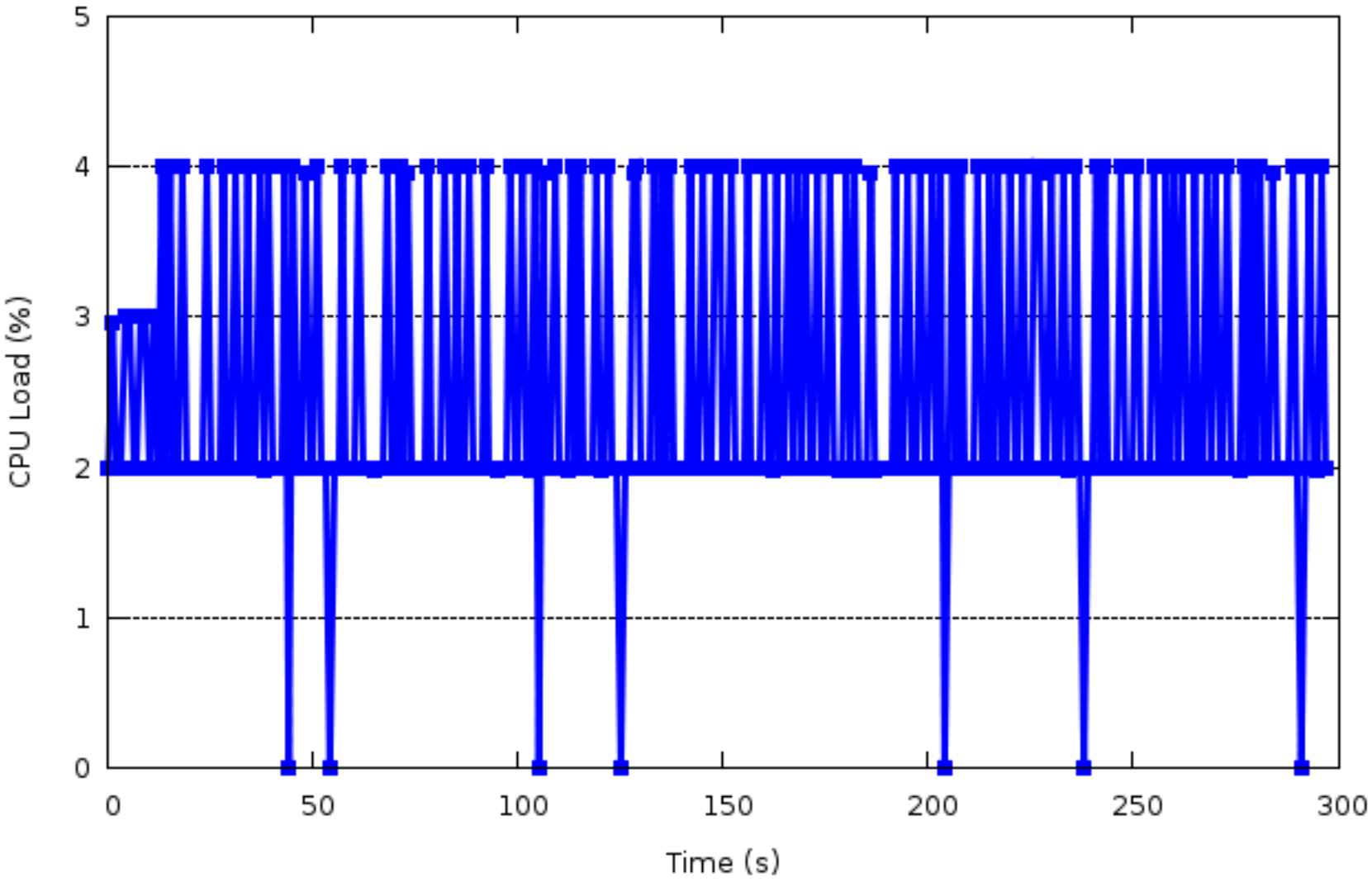

Mean Absolute Error (s)
120
100
80
60
40
20
0
10
20
30
40
50
60
70
80
90
Required CPU (%)
0,05%
0,1%
0,5%
1%
5%


Mean Absolute Error (s)
4
3.5
3
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60
70
80
90
Required CPU (%)
0,05%
0,1%
0,5%
1%
5%

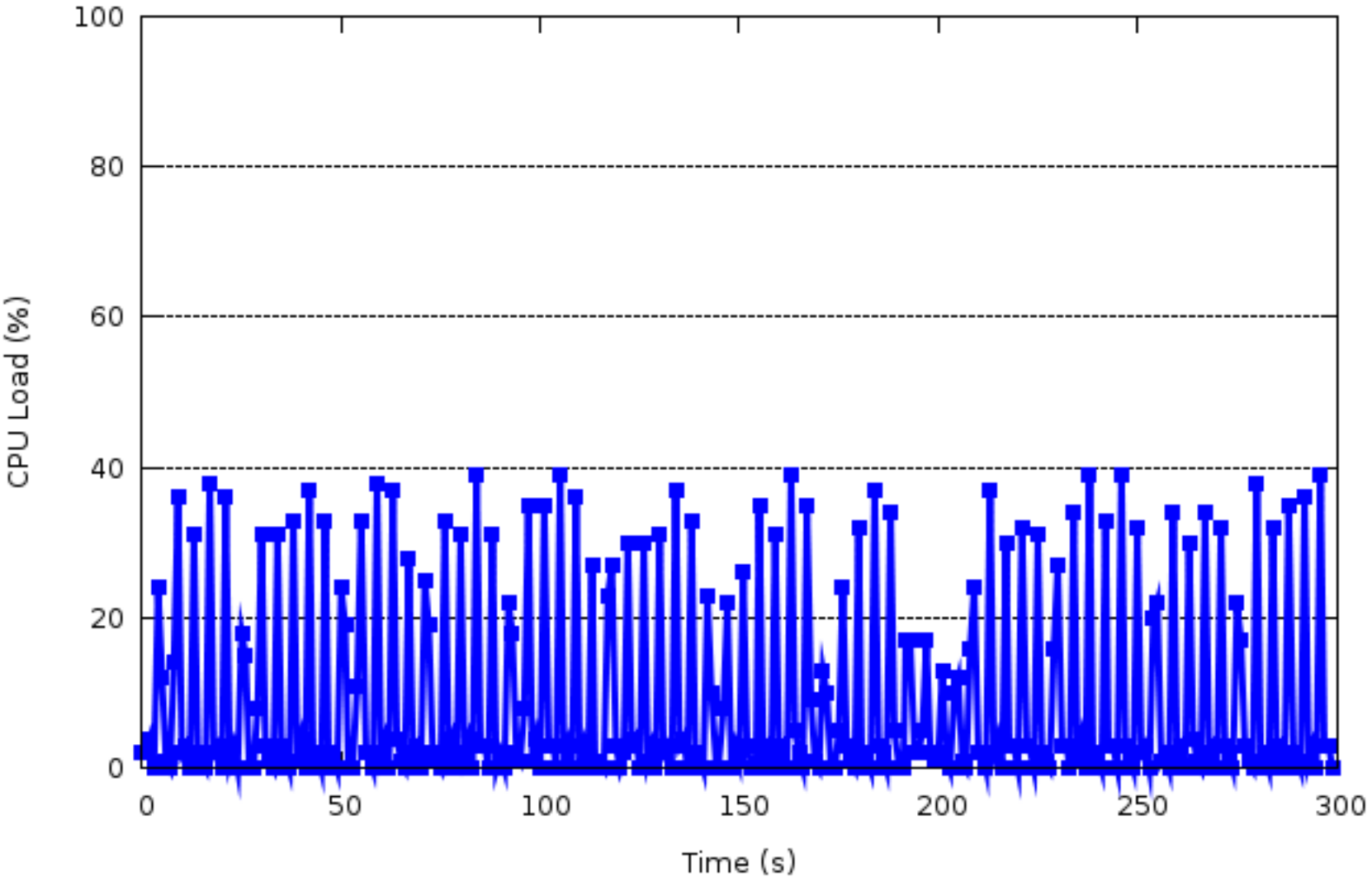
LBA Overhead

Three experiments:

- no grid apps are running – baseline overhead
  - 0% CPU
  - 6MB (shared libs + InteGrade + LBA)
- Grid apps are running, but requiring 0% CPU
  - Only the cost of monitoring resource usage: 2% - 4% cost
- Grid apps are running and requiring 100% CPU
  - LBA is constantly monitoring resource consumption and al events of resource usage are considered bursts
  - Below 5% of overhead for almost 70% of the time



LBA Overhead for 0% Req. CPU



LBA Overhead for 100% Req. CPU

Conclusion

A mechanism to limit the need to perform task migration in case of resource failures

Temporary resource usage bursts (by local) apps do not justify the cost of migration

Evaluation shows that burst prediction has enough accuracy

Overall goal: lower the makespan of grid applications in the presence of resource failures

Future Work

Implement the PM and AM components

Evaluate the overall impact of the mechanism in terms of the makespan of grid applications

- Compared to the sole use of checkpointing and task migration

Refine the algorithm used to combine the effect of different local applications in the prediction of burst duration

Questions?

THANK Y0U!

Burst Prediction Algorithm

For an individual local app

Algorithm 1: $getDurationByProcess(spd, currUsage)$

Input : actual application and total resource usage.
Output: estimated burst duration for the application.

```
1  hpd ← application's behavior history
2  if δ > hpd.φ or procCurrUsage < hpd.φ then
3  |  if procCurrUsage = δ then
4  |  |  return SUPER_DURATION
5  |  end
6  |  secs ← hpd.R[procCurrUsage, δ]
7  |  if secs ≠ SUPER_DURATION then
8  |  |  analisedValue+ = (procCurrUsage − δ)
9  |  end
10 else
11 |  if procCurrUsage = hpd.φ then
12 |  |  return SUPER_DURATION
13 |  end
14 |  secs ← hpd.R[procCurrUsage, hpd.φ]
15 |  if secs ≠ SUPER_DURATION then
16 |  |  analisedValue+ = (procCurrUsage − hpd.φ)
17 |  end
18 end
19 return secs
```