Resource Use Pattern Analysis for Opportunistic Grids

Marcelo Finger Germano C. Bezerra Danilo R. Conde

Department of Computer Science (IME-USP)
University of São Paulo

Supported by CNPq/Brazil project 550895/2007-8.

Topics

Topics

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments
- Opportunistic Grid Computing
  - Idle time of machines

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments
- Opportunistic Grid Computing
  - Idle time of machines
  - High-performance computation

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments
- Opportunistic Grid Computing
  - Idle time of machines
  - High-performance computation
  - Resource-owners have to give permission

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments
- Opportunistic Grid Computing
  - Idle time of machines
  - High-performance computation
  - Resource-owners have to give permission
  - QoS must remain high

Grids and Opportunism

- Applications for Grid Computing
  - computationally intensive
  - distributed
  - heterogeneous environments
- Opportunistic Grid Computing
  - Idle time of machines
  - High-performance computation
  - Resource-owners have to give permission
  - QoS must remain high
- InteGrade: opportunistic grid infrastructure

Effective Opportunistic Computing

- Desirable: prediction of resource availability
- Prediction available for grid scheduler
- The better the prediction, the lower impact on QoS

Effective Opportunistic Computing

- Desirable: prediction of resource availability
- Prediction available for grid scheduler
- The better the prediction, the lower impact on QoS
- Proposed Solution:

Resource Use Pattern Analysis

UPA: (Resource) Use Pattern Analysis

- Consists of:

  Detecting the local **use pattern** of each **resource** at each **machine** in the grid

UPA: (Resource) Use Pattern Analysis

- Consists of:

  Detecting the local **use pattern** of each **resource** at each **machine** in the grid

- Basic Hypothesis:

  Each **resource** has a (temporal) **"pattern"** of use

Goal

- To develop a method

Goal

- To develop a method
- that automatically performs resource use pattern analysis

Goal

- To develop a method
- that automatically performs resource use pattern analysis
- for machines belonging to opportunistic grids

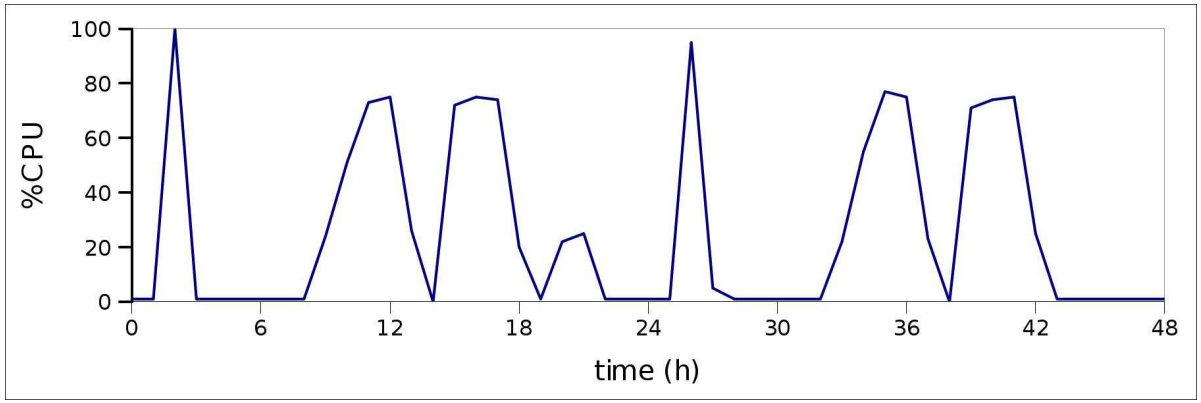
Strategy

- Discover the prototypical patterns of use (off-line)
  - Unsupervised machine learning
  - Clustering analysis
- Runtime prediction
  - Comparing prototypical with "current" pattern of use
- Development method:
  - Simulation: parameter setting
  - Implementation: LUPA module for InteGrade grid

Topics

Resource Use Objects

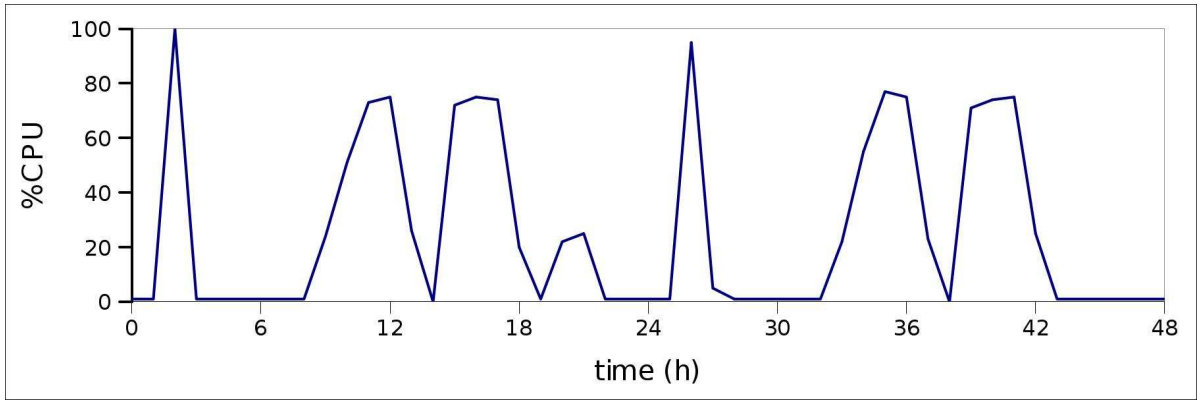
- A resource use object:



- Sampled at every 5 min
- Span of 48h used for prediction
- Resources: **CPU use**, **available RAM**, disk space, swap space, network and disk I/O

Resource Use Objects

- A resource use object:



- Sampled at every 5 min
- Span of 48h used for prediction
- Resources: **CPU use**, **available RAM**, disk space, swap space, network and disk I/O
- Objects represent availability (not only use)

The UPA Method

- Resource Use Pattern Analysis
  - Unsupervised machine learning
  - Obtain fixed number of use classes

The UPA Method

- Resource Use Pattern Analysis
  - Unsupervised machine learning
  - Obtain fixed number of use classes
    - A class represents a frequent use pattern
    - E.g. busy work day, light work day, holiday, etc
  - Each class represented by a prototypical object

The UPA Method

- Resource Use Pattern Analysis
  - Unsupervised machine learning
  - Obtain fixed number of use classes
    - A class represents a frequent use pattern
    - E.g. busy work day, light work day, holiday, etc
  - Each class represented by a prototypical object
- Two phases:
  - training/learning phase: off-line
  - execution/prediction phase: runtime

Off-line Learning

- Inputs a large amount of objects
- Collected from regular operation
- Clustering is applied to training data
- Reliability depends on amount of data
  - At least 60 objects, or 2 months of data.

Off-line Learning

- Inputs a large amount of objects
- Collected from regular operation
- Clustering is applied to training data
- Reliability depends on amount of data
  - At least 60 objects, or 2 months of data.
- Several parameters have to be set

Learning Parameters

- Number of clusters: 5, 10

Learning Parameters

- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational

Learning Parameters

- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational
- Computation of prototypical element: centroid, centre

Learning Parameters

- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational
- Computation of prototypical element: centroid, centre
- Similarity measurement: Euclidean distance between points.
  - Between clusters: single linkage, complete linkage, centroid method, Ward's method

Learning Parameters

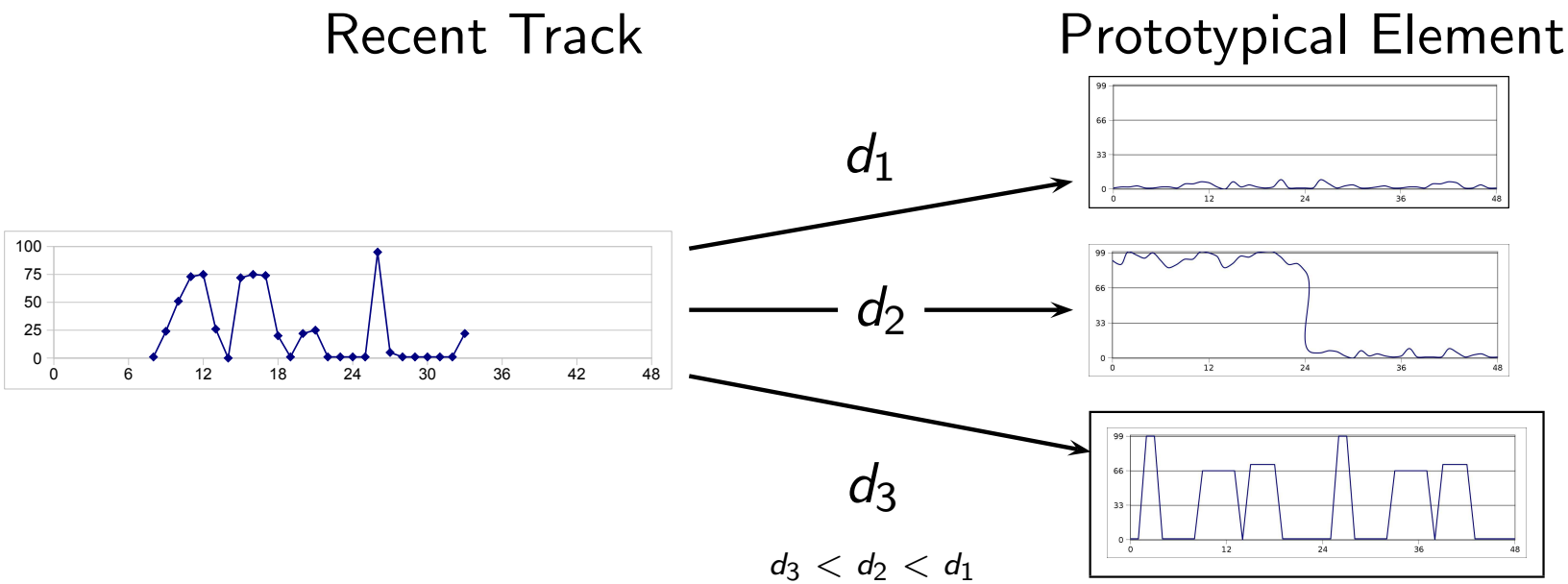
- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational
- Computation of prototypical element: centroid, centre
- Similarity measurement: Euclidean distance between points.
  - Between clusters: single linkage, complete linkage, centroid method, Ward's method
- Clustering algorithms: hierarchical, sequential, k-means, etc.

Learning Parameters

- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational
- Computation of prototypical element: centroid, centre
- Similarity measurement: Euclidean distance between points.
  - Between clusters: single linkage, complete linkage, centroid method, Ward's method
- Clustering algorithms: hierarchical, sequential, k-means, etc.

Simulation was performed to choose parameters

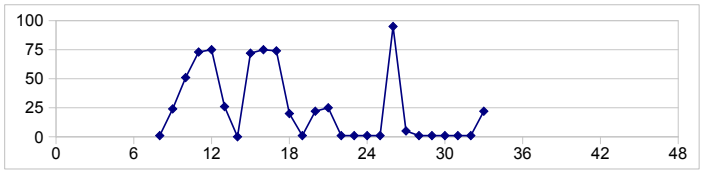
Runtime Prediction

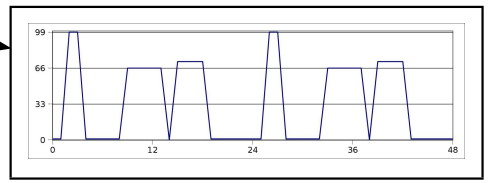


Runtime Prediction

Recent Track

Prototypical Element





Predictions

- Recent 24h record is compared against 48h use classes
- The closest class is the **current use class**.
- Predicted future in current use class: **[now–48h]**
- Request at 6am can predict 18 hours
- Request at 6pm can predict 6 hours (only)

Predictions

- Recent 24h record is compared against 48h use classes
- The closest class is the **current use class**.
- Predicted future in current use class: **[now–48h]**
- Request at 6am can predict 18 hours
- Request at 6pm can predict 6 hours (only)
- Options for longer predictions:
  - Use less than 24 of current record.

Predictions

- Recent 24h record is compared against 48h use classes
- The closest class is the **current use class**.
- Predicted future in current use class: **[now–48h]**
- Request at 6am can predict 18 hours
- Request at 6pm can predict 6 hours (only)
- Options for longer predictions:
  - Use less than 24 of current record. Predicts $<$ 48h
  - Chain of predictions (not implemented yet)

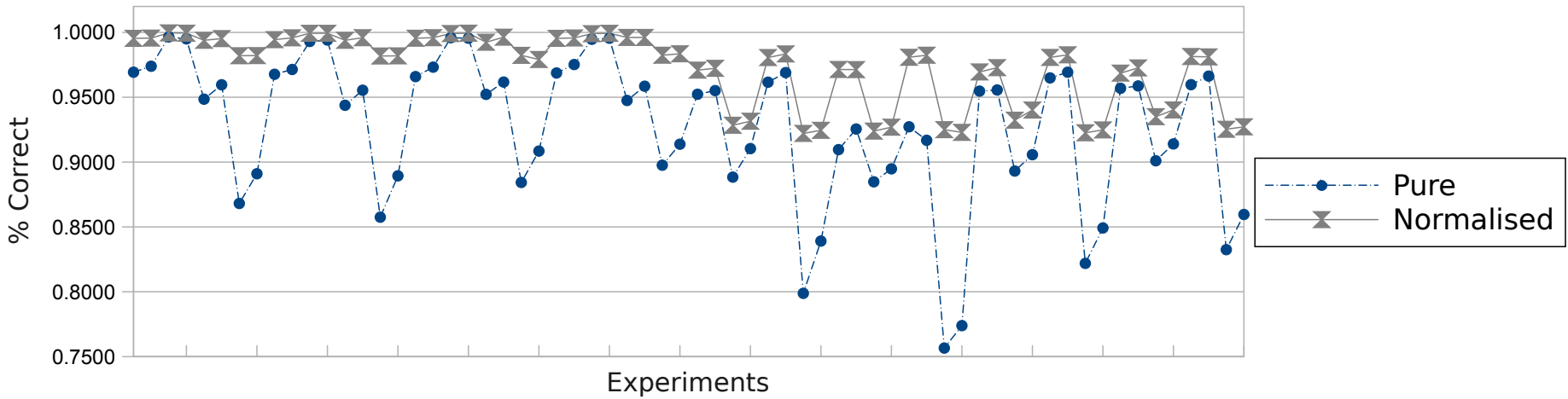
Topics

Trace-driven Simulation

- Data collected for CPU and RAM use, 120 days
- Linux machines with very different types of users:
  1. a general purpose machine with more than 30 users;
  2. a single user machine;
  3. a general purpose machine with 6 users;
  4. a multi-user machine employed for testing heavy computational linguistics programs.
- Several parameters compared

Simulation Results

Correct predictions above 75% in all parameter combinations



Parameters Learned

- Number of clusters: 5, 10
- Data normalisation: no normalisation, variational
- Computation of prototypical element: centroid, centre
- Similarity measurement: Distance between clusters: points.
  - single linkage, complete linkage, centroid method, Ward's method
- Clustering algorithms: hierarchical, sequential, k-means, etc.

Parameters Learned

- Number of clusters: **5**, 10
- Data normalisation: no normalisation, **variational**
- Computation of prototypical element: **centroid**, centre
- Similarity measurement: Distance between clusters: points.
  - single linkage, complete linkage, **centroid method**, Ward's method
- Clustering algorithms: **hierarchical**, sequential, *k*-means, etc.

Parameters Learned

- Number of clusters: **5**, 10
- Data normalisation: no normalisation, **variational**
- Computation of prototypical element: **centroid**, centre
- Similarity measurement: Distance between clusters: points.
  - single linkage, complete linkage, **centroid method**, Ward's method
- Clustering algorithms: **hierarchical**, sequential, k-means, etc.

Simulation validated UPA hypotheses on the use of past patterns to predict future behaviours

Topics

Implementation Objectives

- Explore ways the scheduling of grid applications using the UPA method

Implementation Objectives

- Explore ways the scheduling of grid applications using the UPA method
- Compare UPA scheduling with other methods

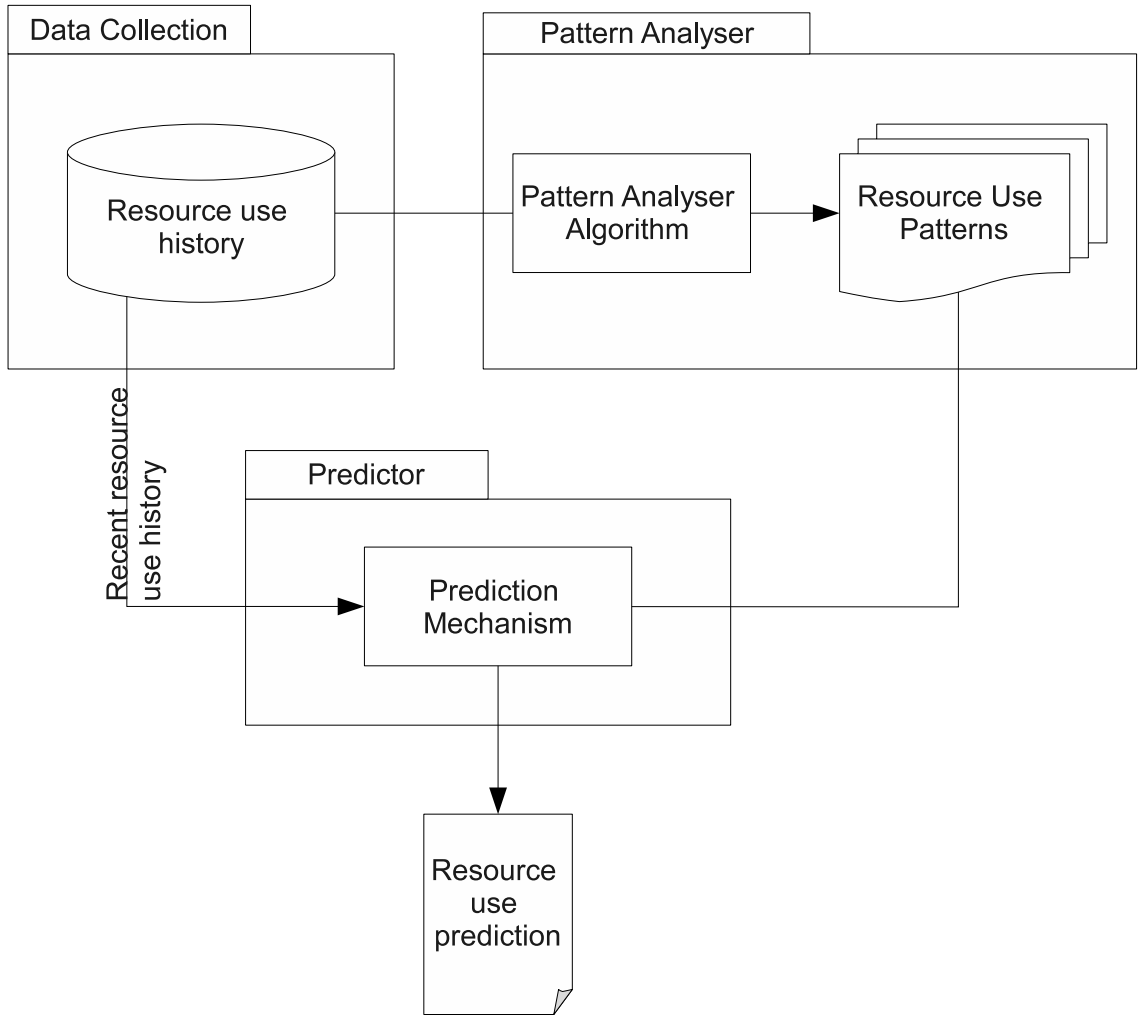
Implementation Objectives

- Explore ways the scheduling of grid applications using the UPA method
- Compare UPA scheduling with other methods
- Identify initialisation strategies
  - because UPA needs a few months of resource use data collection to be reliable

Design Decisions

- UPA data analysed locally, not centrally
  - Advantages: privacy, no network traffic due to UPA processing
- Local resource Use Pattern Analyser (LUPA) module installed in all grid machines
- LUPA module is part of InteGrade middleware
- C++, on several Linuxes

LUPA Architecture



LUPA Submodules

- **Data Collection**: resource use sampling.
- **Pattern Analyser**: off-line clustering.
  Patterns are recomputed once a day.
- **Predictor**: runtime predictions. Interface

```
double[] getPrediction(resource r, int hours);
```

  returns a vector of values representing the `r`-use prediction for the next `hours`, in 5-minute intervals.

LUPA is not yet integrated with the scheduler.

Topics

Experiment Goals

Compare UPA scheduling with other methods

Experiment Goals

Compare UPA scheduling with other methods

Scheduling methods for choosing n machines for h hours:

Experiment Goals

Compare UPA scheduling with other methods

Scheduling methods for choosing n machines for h hours:

- **RR**: round robin. No prediction

Experiment Goals

Compare UPA scheduling with other methods

Scheduling methods for choosing n machines for h hours:

- **RR**: round robin. No prediction
- **last4**: Predicts to the future average of last 4h.

Experiment Goals

Compare UPA scheduling with other methods

Scheduling methods for choosing n machines for h hours:

- **RR**: round robin. No prediction
- **last4**: Predicts to the future average of last 4h.
- **UPA**

Experiment Goals

Compare UPA scheduling with other methods

Scheduling methods for choosing n machines for h hours:

- **RR**: round robin. No prediction
- **last4**: Predicts to the future average of last 4h.
- **UPA**

Predicting methods choose n machines with highest prediction of CPU availability for the next h hours

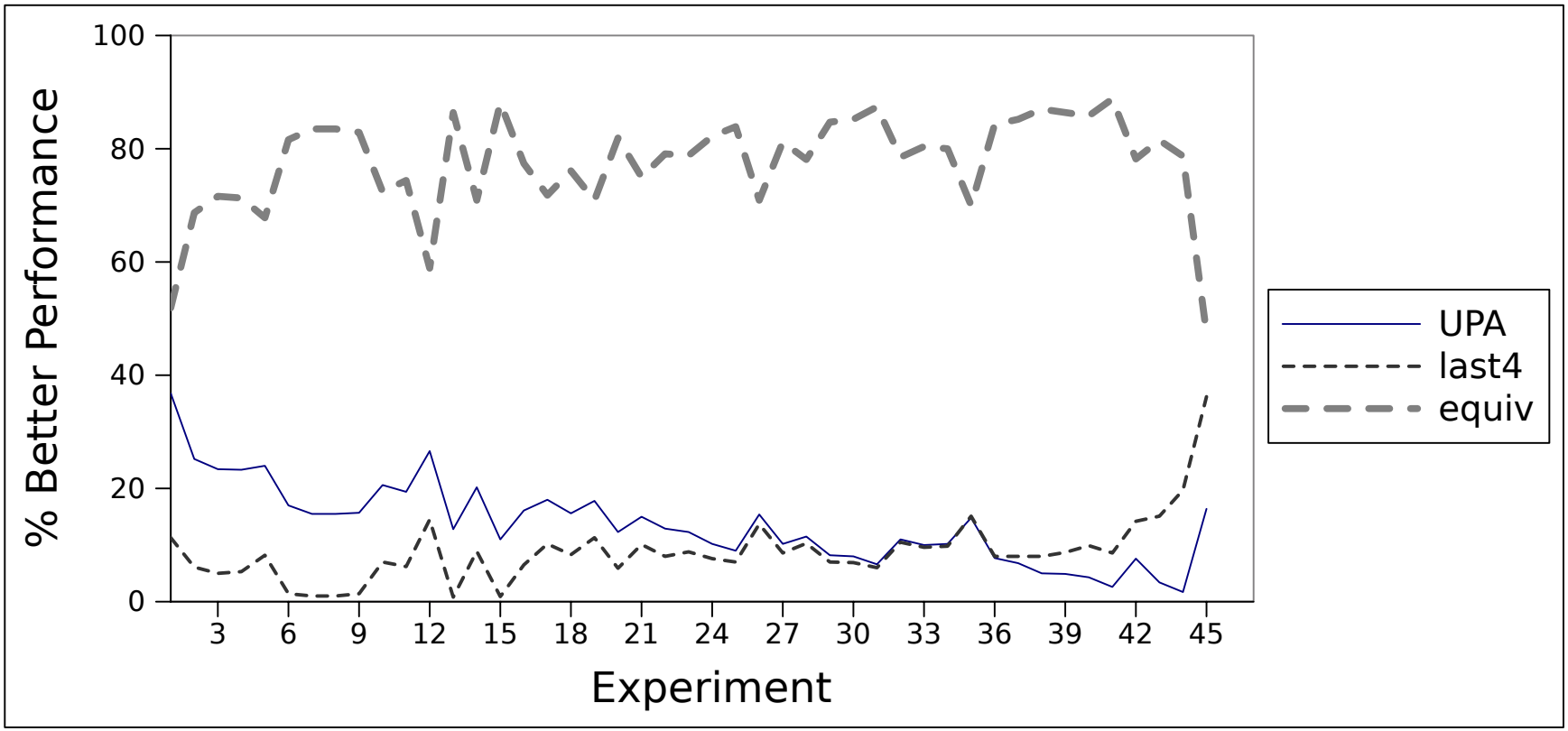
Experiment Description

- 15 machines, logs of 41–120 days
- Sequence of tests for a fixed set of parameters: n, h
- Instant tests (1 experiment $\approx$ 527 tests)
  - Choose a day with m valid machines
  - 24 tests are executed for 3 methods, each for an hour of the day

$$performance_s(t) = 1 - \frac{\sum_{i=1}^{n} use(m_i, t, h)}{n}$$

  - s: scheduling algorithm
  - n: number of chosen machines
  - m_i: machine chose to run for h hours starting at t
  - $use(m_i, t, h)$: average CPU at m_i for $[t, t + h]$

UPA *versus* last4



Comparison of Scheduling Methods

	UPA	**last4**
Performance	Similar to **last4**, but better	Similar to **UPA**, but worse
Impact of $\uparrow m/n$	$\uparrow$ performance	no effect
Impact of $\uparrow h$	no effect	no effect

Performance of **RR** consistently below both **last4** and **UPA**

Adaptative UPA Method

- Use **UPA** if more than 21 days of data collection is available; else
- Use **last4** if more than 4 hours of data collection is available; else
- Predict that resource use at request time persists.

Overhead Analysis

- Running time for pattern analysis was always below 1s
- Running time for prediction calls was always below 3ms

Overhead Analysis

- Running time for pattern analysis was always below 1s
- Running time for prediction calls was always below 3ms
- Measurements made on notebook [AMD Turion 64 1.8GHz CPU, 1GB RAM running Kubuntu 7.10 (32 bits) Linux]
- Running pattern analysis once a day poses negligible overhead

Topics

Correlated Availability

- BOINC: Volunteer computing
- Correlated Availability in Internet-Distributed Systems (Kondo, Andrzejak & Anderson, 2008)
- Computes patterns of simultaneous **CPU availability** via clustering

Correlated Availability

- BOINC: Volunteer computing
- Correlated Availability in Internet-Distributed Systems (Kondo, Andrzejak & Anderson, 2008)
- Computes patterns of simultaneous **CPU availability** via clustering

- Monitors 112,268 hosts
- Global monitoring and pattern discovery
- Several interesting clusters of machines discovered, with potential applications.
- Not originally intended for scheduling

Preemptive Resume Scheduling
[Roy and Livny, 2003]

- Condor:
  - Combination of dedicated and opportunistic scheduling
  - Opportunistic scheduling based on checkpointing
  - Preemptive resume scheduling
  - No prediction

Preemptive Resume Scheduling
[Roy and Livny, 2003]

- Condor:
  - Combination of dedicated and opportunistic scheduling
  - Opportunistic scheduling based on checkpointing
  - Preemptive resume scheduling
  - No prediction
  - **No need to predict job duration**

Predictive Scheduling
[Yang, Schopf and Foster, 2003]

- One-step-ahead CPU load prediction
  - Load Tendency Prediction
  - Increase/decrease adaptation processes
- Interval Load Prediction
  - Average over aggregated CPU load time series
- Load Variance Prediction
  - Standard deviation time series
  - Combined with one-step-ahead prediction
- Conservative scheduling: combines interval and variance load prediction

Topics

Conclusions

- Some form of prediction is always preferable to no prediction
- UPA method compares favourably w.r.t. other methods
- Small overheads involved
- Method can be used for practical grid scheduling

Conclusions

- Some form of prediction is always preferable to no prediction
- UPA method compares favourably w.r.t. other methods
- Small overheads involved
- Method can be used for practical grid scheduling
  - but needs task duration prediction

Ongoing and Future Works

- Scheduler using LUPA information
- Automated learning of task duration prediction
- Long term predictions
- Preemptive task migration using UPA
- Automated "booking" of machine resources for future executions using UPA

www.integrade.org.br

mfinger@ime.usp.br