Automatic discovery of the characteristics and capacities of a distributed computational platform
**Introduction to the Grid**

**Metacomputing**: aggregating distributed computers and storage units

the resulting platform is usually called the **Grid**

- Very high potential (in power and ease of use)

- The Grid hardware is already there
  
  Share of local resources between several organizations ⇒ WAN constellation of LAN

- The Grid software infrastructure only emerging.

  Difficulties come from (amongst others):
  
  - Heterogeneity
  
  - Resource sharing (⇒ availability variations)
  
  - Multiple organizations (trust issue)
Random scheduling:
- Tasks list; existing hosts list

Simple scheduling:
- About tasks: theoretical complexity (like $O(n)$)
- About hosts: peak performance or on a given benchmark
- About links: maximal capacities

Current Grid scheduling:
- About hosts: up/down, CPU and memory load
- About links: current capacities matrix
Random scheduling:

- Tasks list; existing hosts list

Simple scheduling:

- About tasks: theoretical complexity (like $O(n)$)
- About hosts: peak performance or on a given benchmark
- About links: maximal capacities

Current Grid scheduling:

- About hosts: up/down, CPU and memory load
- About links: current capacities matrix

Information quality is crucial to scheduling quality
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)
   NWS + FAST

II. Qualitative knowledge of network topology
    ENV → ALNeM
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)
   NWS + FAST

II. Qualitative knowledge of network topology
   ENV → ALNeM
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of **needs** (tasks) and **availabilities** (servers and network)
   NWS + FAST

II. Qualitative knowledge of network **topology**
   ENV → ALNeM
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)

   NWS + FAST

II. Qualitative knowledge of network topology

   ENV → ALNeM

NWS [RSH99] forecasts:

- bandwidth, latency, memory, disk space, . . .

- host load as percentage

![Diagram of network topology]
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of **needs** (tasks) and **availabilities** (servers and network)
   
   **NWS + FAST**

II. Qualitative knowledge of network **topology**
   
   ENV $\rightarrow$ ALNeM

**NWS** [RSH99] forecasts:

- bandwidth, latency, memory, disk space, ...
- host load as percentage

**FAST** [Qui02b] provides:

- Task needs benchmarking
time and memory size (fitting to the host)

$\Rightarrow$ Duration of the task on each server
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)  
   **NWS + FAST**

II. Qualitative knowledge of network **topology**
   ENV → ALNeM

Motivating example: how to configure NWS?

- Simplest: measure everything
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network) 
   **NWS + FAST**

II. Qualitative knowledge of network **topology**
   **ENV → ALNeM**

Motivating example: how to configure NWS?

- Simplest: measure everything
- Better: hierarchical
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)
   NWS + FAST

II. Qualitative knowledge of network topology
   ENV → ALNeM

Motivating example: how to configure NWS?

- Simplest: measure everything
- Better: hierarchical

Target:
- logical topology (end-host)
- interferences
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of **needs** (tasks) and **availabilities** (servers and network)
   NWS + FAST

II. Qualitative knowledge of network **topology**
   ENV $\rightarrow$ ALNeM

ENV [SBW99]:
- 🌟 maps the network without root access
- 😞 only hierarchical (tree)
Overview of this work

Our goal: provide the information needed by the scheduler.

I. Quantitative knowledge of needs (tasks) and availabilities (servers and network)  
   NWS + FAST

II. Qualitative knowledge of network topology  
   ENV → ALNeM

ENV [SBW99]:
- ☺ maps the network without root access
- ☹ only hierarchical (tree)

ALNeM [LQ04]
- Same approach than ENV, generalized
- Stronger theoretical basements
Overview

- Introduction
- NWS: Network Weather Service
- FAST: Fast’s Agent System Timer
- ALNeM: Application-Level Network Mapper
- Conclusion
Goal: (Grid) system availabilities measurement and forecasting

Leaded by Prof. Wolski (UCSB), used by AppLeS, Globus, NetSolve, Ninf, DIET, ... 

Architecture: Distributed system

Sensor: conducts the measurements
Memory: stores the results
Forecaster: forecasts statistically the tendencies
Name server: directory service like LDAP
The Network Weather Service: presentation

**Goal:** (Grid) system availabilities measurement and forecasting
   Leaded by Prof. Wolski (UCSB), used by AppLeS, Globus, NetSolve, Ninf, DIET, ...

**Architecture:** Distributed system

- **Sensor:** conducts the measurements
- **Memory:** stores the results
- **Forecaster:** forecasts statistically the tendencies
- **Name server:** directory service like LDAP

Steady state: regular tests
The Network Weather Service: presentation

**Goal:** (Grid) system availabilities measurement and forecasting

  Leaded by Prof. Wolski (UCSB), used by AppLeS, Globus, NetSolve, Ninf, DIET, ... 

**Architecture:** Distributed system

- **Sensor:** conducts the measurements
- **Memory:** stores the results
- **Forecaster:** forecasts statistically the tendencies
- **Name server:** directory service like LDAP

Handling of a request
The Network Weather Service: presentation

Goal: (Grid) system availabilities measurement and forecasting

Leaded by Prof. Wolski (UCSB), used by AppLeS, Globus, NetSolve, Ninf, DIET, ... 

Architecture: Distributed system

Sensor: conducts the measurements
Memory: stores the results
Forecaster: forecasts statistically the tendencies
Name server: directory service like LDAP

Handling of a request

Client

Nameserver

Sensor

Sensor

Memory

Forecaster
The Network Weather Service: presentation

**Goal:** (Grid) system availabilities measurement and forecasting

- Led by Prof. Wolski (UCSB), used by AppLeS, Globus, NetSolve, Ninf, DIET, ...

**Architecture:** Distributed system

- **Sensor:** conducts the measurements
- **Memory:** stores the results
- **Forecaster:** forecasts statistically the tendencies
- **Name server:** directory service like LDAP

Handling of a request

---

Martin QUINSON  10 mai 2004
Measurements and Forecasting

- Provided metrics:
  - `availableCpu` (for an incoming process), `currentCpu` (for existing processes), `bandwidthTcp`, `latencyTcp` (Default: 64Kb in 16Kb messages; buffer=32Kb), `connectTimeTcp`, `freeDisk`, `freeMemory`, ...

- Forecasting using statistics

  Data = serie: \( D_1, D_2, \ldots, D_{n-1}, D_n \). We want \( D_{n+1} \).
  
  Methods are applied on \( D_1, D_2, \ldots, D_{n-1} \). each one predict \( D_n \).
  
  Selection of the best on \( D_n \) to predict \( D_{n+1} \).

  Used statistical methods
  - `mean`: running, (adapting) sliding window ;
  - `median`: idem ;
  - `gradian`: \( GRAD(t, g) = (1-g) \times GRAD(t-1, g) + g \times value(t) \);
  - last value.
Conclusion about NWS

😊 Complete environment
😊 Designed for scheduling
😊 Statistical forecasting
😊 Widely used

😊 Uneasy to extend
😊 Sometimes difficult to deploy
😊 TCP only (myrinet-based?)

Related work

NetPerf: HP project to sort network components, no interactivity
GloPerf: Globus moves to NWS
PingER: Regular pings between 600 hosts in 72 countries
Iperf: Finds out the bandwidth by saturating the link for 30 seconds
RPS: Forecasting limited to the CPU load

Performance Co-Pilot (SGI):
  • Same kind of architecture
  • Low level data (/proc) ⇒ not easily usable by a scheduler
  • No forecasting
Overview

- Introduction
- NWS: Network Weather Service
- FAST: Fast’s Agent System Timer
- ALNeM: Application-Level Network Mapper
- Conclusion
Goals:

- gather routine’s performance on a given host at a given time
- interactivity, ease of use

Architecture:
Fast Agent’s System Timer: presentation

Goals:
- gather routine's performance on a given host at a given time
- interactivity, ease of use

Architecture:
Goals:

- gather routine's performance on a given host at a given time
- interactivity, ease of use

Architecture:
Related Work

- Elementary operation count: the myth of the constant Mflop/s
- Analytical model, micro-benchmarking: complex ⇓ interactive, task description?
- Probability, Markov: how to instanciate it at a given time?
Routines needs modeling

Related Work

- Elementary operation count: the myth of the constant Mflop/s
- Analytical model, micro-benchmarking: complex ≠ interactive, task description?
- Probability, Markov: how to instanciate it at a given time?

FAST’s approach

- Simple (sequential) routines like BLAS
  - macro-benchmarking: benchmark \{task; host\} as a whole at installation
  - Getting the time: utime + stime to avoid background load
  - Getting the space: step by step execution (like gdb) to track changes and search peak
    ⇒ rather long, but only once
Routines needs modeling

Related Work

- Elementary operation count: the myth of the constant Mflop/s
- Analytical model, micro-benchmarking: complex \( \not\rightarrow \) interactive, task description?
- Probability, Markov: how to instanciate it at a given time?

FAST’s approach

- Simple (sequential) routines like BLAS
  - **macro-benchmarking**: benchmark \{task; host\} as a whole at installation
  - Getting the time: utime + stime to avoid background load
  - Getting the space: step by step execution (like gdb) to track changes and search peak
    \( \Rightarrow \) rather long, but only once

- Complex routines (ScaLAPACK)
  - Structural decomposition by source analysis
Routines needs modeling

Related Work
- Elementary operation count: the myth of the constant Mflop/s
- Analytical model, micro-benchmarking: complex \( \not\Rightarrow \) interactive, task description?
- Probability, Markov: how to instantiate it at a given time?

FAST’s approach
- Simple (sequential) routines like BLAS
  macro-benchmarking: benchmark \{task; host\} as a whole at installation
  - Getting the time: utime + stime to avoid background load
  - Getting the space: step by step execution (like gdb) to track changes and search peak
    \( \Rightarrow \) rather long, but only once

- Complex routines (ScaLAPACK)
  Structural decomposition by source analysis

Freddy [CDQF03], integration underway
Routines needs modeling

Related Work

- Elementary operation count: the myth of the constant Mflop/s
- Analytical model, micro-benchmarking: complex ⇛ interactive, task description?
- Probability, Markov: how to instanciate it at a given time?

FAST’s approach

- Simple (sequential) routines like BLAS
  
  macro-benchmarking: benchmark {task; host} as a whole at installation
  - Getting the time: utime + stime to avoid background load
  - Getting the space: step by step execution (like gdb) to track changes and search peak
    ⇒ rather long, but only once

- Complex routines (ScaLAPACK)
  Structural decomposition by source analysis

- Irregular routines (sparse algebra)
  No forecasting ⇒ selection of the fastest host
  Decomposition to extract simple parts
  Input of estimators from the application

Freddy [CDQF03], integration underway
### Quality of the modeling

#### Time modeling

<table>
<thead>
<tr>
<th></th>
<th>dgeadd</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>icluster</td>
<td>paraski</td>
<td>icluster</td>
<td>paraski</td>
</tr>
<tr>
<td>Maximal error</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.21s</td>
<td>5.8s</td>
</tr>
<tr>
<td></td>
<td>6%</td>
<td>35%</td>
<td>0.3%</td>
<td>4%</td>
</tr>
<tr>
<td>Average error</td>
<td>0.006s</td>
<td>0.007s</td>
<td>0.025s</td>
<td>0.03s</td>
</tr>
<tr>
<td></td>
<td>4%</td>
<td>6.5%</td>
<td>0.1%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

**dgeadd:** Matrix addition

**dgemm:** Matrix multiplication

**dtrsm:** Triangular resolution

**icluster:** bi-Pentium II, 256Mb, Linux, IMAG (Grenoble).

**paraski:** Pentium III, 256Mb, Linux, IRISA (Rennes).

**network:** Intra: LAN, 100Mb/s; Inter: VTHD network, 2.5Gb/s.

### Space modeling

**Almost perfect:** Maximal error < 1% ; Average error ≈ 0.1%

**Code size** + **Matrix size**

(constant)   (polynomial)
Forecasting with background load
dgemm with background load (CPU-intensive process in background).

Maximal error: 22%
Average error < 10%
Forecasting of sequence with background load

\[ C = \begin{cases} 
C_r = A_r \times B_r - A_i \times B_i \\
C_i = A_r \times B_i + A_i \times B_r 
\end{cases} \]

client/servers over LAN

Maximal error: 25%; Average error: 13%

Martin QUNSON
10 mai 2004
Comparison with NetSolve’s forecaster

Computation time of dgemm.

Communication time of dgemm.
Latency reduction

NWS
(99569 µs)

FAST (cache miss)
(100685 µs)

FAST (cache hit)
(24 µs)
Responsiveness improvement

Scheduler / NWS collaboration

Forecasting

NWS: out of the box
FAST: {sensors restart + forecaster reset} when the task starts or ends

Theoretical value
Virtual booking: How does it work?

FAST asks NWS to update
NWS sensor
Scheduled task
correction

Scheduling decision
Task started
NWS updated
Task ended
NWS updated

0 1 0 1 0

Time

0 1 0 1 0
Benefits of virtual booking

Measurements

NWS: ADAPT_CPU

FAST: ADAPT_CPU + virtual booking + sensors restart + forecaster reset

Theoretical value

(Result of 4 different runs)

Forecasting
Contributions of FAST

Forecasting with load

Responsiveness

Summary

- Generic benchmarking solution
- Simple interface to quantitative data
- Parallel routines handling currently integrated
- Integration: DIET, NetSolve, Grid-TLSE, cichlid
- 15 000 lines of C code, Linux, Solaris, True64
- 2 journals and 3 conferences/workshops
Overview

- Introduction
- NWS: Network Weather Service
- FAST: Fast’s Agent System Timer
- ALNeM: Application-Level Network Mapper
- Conclusion
Application-Level Network Mapper

Goal: Mapping the network topology

Authors: Arnaud Legrand, Martin Quinson

Motivation: Server hosting, Simulation, Collective Communication Forecasting

Target application: NWS hosting

Problem: Network experiments must not collide (Clique concept)
**Goal:** Mapping the network topology

**Authors:** Arnaud Legrand, Martin Quinson

**Motivation:** Server hosting, Simulation, Collective Communication Forecasting

**Target application:** NWS hosting

**Problem:** Network experiments must not collide (Clique concept)

**Simplest:** One big clique
**Goal:** Mapping the network topology

**Authors:** Arnaud Legrand, Martin Quinson

**Motivation:** Server hosting, Simulation, Collective Communication Forecasting

**Target application:** NWS hosting

**Problem:** Network experiments must not collide (Clique concept)

**Simplest:** One big clique; **Better:** Hierarchical
**Application-Level Network Mapper**

**Goal:** Mapping the network topology

**Authors:** Arnaud Legrand, Martin Quinson

**Motivation:** Server hosting, Simulation, Collective Communication Forecasting

**Focus:** Discover interferences (limiting common links), not really packet paths
Application-Level Network Mapper

**Goal:** Mapping the network topology

**Authors:** Arnaud Legrand, Martin Quinson

**Motivation:** Server hosting, Simulation, Collective Communication Forecasting

**Focus:** Discover interferences (limiting common links), not really packet paths

### Related work

<table>
<thead>
<tr>
<th>Method</th>
<th>Restricted</th>
<th>Focus</th>
<th>Routers</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP</td>
<td>authorized</td>
<td>path</td>
<td>all</td>
<td>passive, dumb routers, LAN</td>
</tr>
<tr>
<td>traceroute</td>
<td>ICMP</td>
<td>path</td>
<td>all</td>
<td>level 3 of OSI</td>
</tr>
<tr>
<td>pathchar</td>
<td>root</td>
<td>path</td>
<td>all</td>
<td>link bandwidth, slow</td>
</tr>
<tr>
<td>Other tomography</td>
<td>no</td>
<td>path</td>
<td>$d_{in} \neq d_{out}$</td>
<td>tree bipartite [Rabbat03]</td>
</tr>
<tr>
<td>ENV</td>
<td>no</td>
<td>interference</td>
<td>some</td>
<td>tree only</td>
</tr>
</tbody>
</table>
ALNeM: Notations

Def (non-interference): \[(ab) \parallel_{rl} (cd) \iff \frac{bw_{\parallel cd}(ab)}{bw(ab)} \approx 1\]

Def (interference): \[(ab) \chi_{rl} (cd) \iff \frac{bw_{\parallel cd}(ab)}{bw(ab)} \approx 0.5\]

Def: Interference matrix \(I(V, \chi_{rl})\)

\[
I(V, \chi_{rl})(a, b, c, d) = \begin{cases} 
1 & \text{if } (ab) \chi_{rl} (cd) \\
0 & \text{if not} 
\end{cases}
\]

INTERFERENCEGRAPH: Given \(\mathcal{H}\) and \(I(\mathcal{H}, \chi_{rl})\),

Find a graph \(G(V, E)\) and the associated routing satisfying:

\[
\begin{cases} 
\mathcal{H} \subset V \\
I(\mathcal{H}, \chi_{G}) = I(\mathcal{H}, \chi_{rl}) \\
|V| \text{ is minimal.}
\end{cases}
\]
**ALNeM: Notations**

**Def (non-interference):**
\[(ab) \parallel_{rl} (cd) \iff \frac{bw_{\parallel} cd(ab)}{bw(ab)} \approx 1\]

**Def (interference):**
\[(ab) \checkmark_{rl} (cd) \iff \frac{bw_{\parallel} cd(ab)}{bw(ab)} \approx 0.5\]

**Def:** Interference matrix \(I(V, \checkmark_{rl})\)

\[
I(V, \checkmark_{rl})(a, b, c, d) = \begin{cases} 
1 & \text{if } (ab) \checkmark_{rl} (cd) \\
0 & \text{if not}
\end{cases}
\]

**INTERFERENCE GRAPH:** Given \(\mathcal{H}\) and \(I(\mathcal{H}, \checkmark_{rl})\),

Find a graph \(G(V, E)\) and the associated routing satisfying:

\[
\begin{cases} 
\mathcal{H} \subset V \\
I(\mathcal{H}, \checkmark_G) = I(\mathcal{H}, \checkmark_{rl}) \\
|V| \text{ is minimal.}
\end{cases}
\]
**ALNeM: Notations**

**Def** (non-interference): \( (ab) \parallel_{rl} (cd) \iff \frac{bw_{\parallel} cd(ab)}{bw(ab)} \approx 1 \)

**Def** (interference): \( (ab) \not{\parallel}_{rl} (cd) \iff \frac{bw_{\parallel} cd(ab)}{bw(ab)} \approx 0.5 \)

**Def:** Interference matrix \( I(V, \not{\parallel}_{rl}) \)

\[
I(V, \not{\parallel}_{rl})(a, b, c, d) = \begin{cases} 
1 & \text{if } (ab) \not{\parallel}_{rl} (cd) \\
0 & \text{if not}
\end{cases}
\]

**INTERFERENCE GRAPH:** Given \( \mathcal{H} \) and \( I(\mathcal{H}, \not{\parallel}_{rl}) \),

Find a graph \( G(V, E) \) and the associated routing satisfying:

\[
\begin{cases} 
\mathcal{H} \subset V \\
I(\mathcal{H}, \not{\parallel}_{G}) = I(\mathcal{H}, \not{\parallel}_{rl}) \\
|V| \text{ is minimal.}
\end{cases}
\]
**Mathematical tools**

**Def.** (total interference): \( a \perp b \iff \forall (u, v) \in \mathcal{H}, (au) \mathcal{X}_{rl} (bv) \)

**Lemma** (separator): \( \forall a, b \in \mathcal{H}, a \perp b \iff \exists \rho \in \tilde{V} \exists \forall z \in \mathcal{H} : \rho \in (a \to z) \cap (b \to z). \)  
(\( \perp \iff \exists \rho \) separator)

**Theorem:** \( \perp \) is an equivalence relation (under some assumptions)

**Theorem** (representativity): \( C \) equivalence class under \( \perp \) (under some assumptions)

\[ \forall \rho, \sigma \in \mathcal{C}, \forall b, u, v \in \mathcal{H}, (\rho, u) \mathcal{X}_{rl} (b, v) \Leftrightarrow (\sigma, u) \mathcal{X}_{rl} (b, v) \]

(you can interchange any member of the class by any other in the matrix)
Def. (total interference): $a \perp b \iff \forall (u, v) \in \mathcal{H}, (au) \updownarrow (bv)$

Lemma (separator): $\forall a, b \in \mathcal{H}, a \perp b \iff \exists \rho \in \tilde{\mathcal{V}} \setminus \forall z \in \mathcal{H}: \rho \in (a \rightarrow z) \cap (b \rightarrow z)$.

(\perp \iff \exists \rho \text{ separator})

Theorem: $\perp$ is an equivalence relation (under some assumptions)

Theorem (representativity): $C$ equivalence class under $\perp$ (under some assumptions)

$\forall \rho, \sigma \in C, \forall b, u, v \in \mathcal{H}, (\rho, u) \updownarrow (b, v) \iff (\sigma, u) \updownarrow (b, v)$

(you can interchange any member of the class by any other in the matrix)
**Def. (total interference):** \( a \perp b \iff \forall (u, v) \in \mathcal{H}, (au) \overset{\text{rl}}{\sim} (bv) \)

**Lemma (separator):** \( \forall a, b \in \mathcal{H}, a \perp b \iff \exists \rho \in \tilde{V} / \forall z \in \mathcal{H} : \rho \in (a \rightarrow z) \cap (b \rightarrow z) \).

\((\perp \iff \exists \rho \text{ separator})\)

**Theorem:** \( \perp \) is an equivalence relation (under some assumptions)

**Theorem (representativity):** \( C \) equivalence class under \( \perp \) (under some assumptions)

\[ \forall \rho, \sigma \in C, \forall b, u, v \in \mathcal{H}, (\rho, u) \overset{\text{rl}}{\sim} (b, v) \iff (\sigma, u) \overset{\text{rl}}{\sim} (b, v) \]

(you can interchange any member of the class by any other in the matrix)
Mathematical tools

Def. (total interference): \( a \perp b \iff \forall (u, v) \in \mathcal{H}, (au) \sim_{rl} (bv) \)

Lemma (separator): \( \forall a, b \in \mathcal{H}, a \perp b \iff \exists \rho \in \widetilde{V} \forall z \in \mathcal{H} : \rho \in (a \rightarrow z) \cap (b \rightarrow z) \).
\( (\perp \iff \exists \rho \text{ separator}) \)

Theorem: \( \perp \) is an equivalence relation (under some assumptions)

Theorem (representativity): \( C \) equivalence class under \( \perp \) (under some assumptions)

\( \forall \rho, \sigma \in C, \forall b, u, v \in \mathcal{H}, (\rho, u) \sim_{rl} (b, v) \iff (\sigma, u) \sim_{rl} (b, v) \)

(you can interchange any member of the class by any other in the matrix)
Algorithm for cliques of trees

Equivalence class $\Rightarrow$ greedy algorithm *eating* the leaves
Algorithm for cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves

Theorem: When $|C_{inf}| = 1$, the graph built is a solution.

Theorem: If a tree being a solution exists, $|C_{inf}| = 1$.

Remark: The graph built is optimal (wrt $|V|$ since $V = H$).

Theorem: When $I$ contains no interferences, the clique of $C_i$ is a valid solution.

Remark: It is also optimal.
Algorithm for cliques of trees

Equivalence class $\Rightarrow$ greedy algorithm *eating* the leaves

Theorem: When $|C_{inf}| = 1$, the graph built is a solution.

Theorem: If a tree being a solution exists, $|C_{inf}| = 1$.

Remark: The graph built is optimal (wrt $|V|$ since $V = H$)

Theorem: When $I$ contains no interferences, the clique of $C_i$ is a valid solution.

Remark: It is also optimal.

Martin QUINSON
10 mai 2004
Equivalence class ⇒ greedy algorithm *eating* the leaves

**Theorem:** When \(|C_{inf}| = 1\), the graph built is a solution.

**Theorem:** If a tree being a solution exists, \(|C_{inf}| = 1\).

**Remark:** The graph built is optimal (wrt \(|V|\) since \(V = H\)).

**Theorem:** When \(I\) contains no interferences, the clique of \(C_i\) is a valid solution.

**Remark:** It is also optimal.
Algorithm for cliques of trees

Equivalence class ⇒ greedy algorithm *eating* the leaves

**Theorem:** When $|C_{\text{inf}}| = 1$, the graph built is a solution.

**Theorem:** If a tree being a solution exists, $|C_{\text{inf}}| = 1$.

**Remark:** The graph built is optimal (wrt $|V|$ since $V = H$)
Algorithm for cliques of trees

Equivalence class $\Rightarrow$ greedy algorithm *eating* the leaves

**Theorem:** When $|C_{\text{inf}}| = 1$, the graph built is a solution.

**Theorem:** If a tree being a solution exists, $|C_{\text{inf}}| = 1$.

**Remark:** The graph built is optimal (wrt $|V|$ since $V = \mathcal{H}$)

**Theorem:** When $I$ contains no interferences, the clique of $C_i$ is a valid solution.

**Remark:** It is also optimal
Let $a, b$ be the elements of $C_i$ with the more interferences.  

**Lemma**: no solution with $\exists z \in \mathcal{H}$ so that $z \in (a \rightarrow b)$  

$\Rightarrow$ Cut between $a$ and $b$!
Let $a, b$ be the elements of $C_i$ with the more interferences.

**Lemma:** no solution with $\exists z \in H$ so that $z \in (a \rightarrow b)$

$\Rightarrow$ Cut between $a$ and $b$!

Finding out how to cut
Let \( a, b \) be the elements of \( C_i \) with the more interferences. **Lemma:** no solution with \( \exists z \in H \) so that \( z \in (a \to b) \)

\[ \Rightarrow \text{Cut between } a \text{ and } b! \]

**Finding out how to cut**

\[
I_1 = \{ u \in C_i : a \in (b \to u) \text{ and } b \not\in (a \to u) \} \\
I_2 = \{ u \in C_i : a \not\in (b \to u) \text{ and } b \not\in (a \to u) \} \\
I_3 = \{ u \in C_i : a \not\in (b \to u) \text{ and } b \in (a \to u) \} \\
I_4 = \{ a; b \}
\]

the contrary would imply \( \circ \rightarrow u \rightarrow \bullet \).
Let $a, b$ be the elements of $C_i$ with the more interferences.  

**Lemma:** no solution with $\exists z \in H$ so that $z \in (a \rightarrow b)$  
$\Rightarrow$ Cut between $a$ and $b$!

Finding out how to cut

\[
\begin{align*}
I_1 &= \{ u \in C_i : a \in (b \rightarrow u) \text{ and } b \notin (a \rightarrow u) \} \\
I_2 &= \{ u \in C_i : a \notin (b \rightarrow u) \text{ and } b \notin (a \rightarrow u) \} \\
I_3 &= \{ u \in C_i : a \notin (b \rightarrow u) \text{ and } b \in (a \rightarrow u) \} \\
I_4 &= \{ u \in C_i : a \in (b \rightarrow u) \text{ and } b \in (a \rightarrow u) \}
\end{align*}
\]
Let $a, b$ be the elements of $C_i$ with the more interferences.  
**Lemma:** no solution with $\exists z \in \mathcal{H}$ so that $z \in (a \rightarrow b)$  
$\implies$ Cut between $a$ and $b$! 

Finding out how to cut 

\[
\begin{align*}
I_1 &= \{ u \in C_i : a \in (b \rightarrow u) \text{ and } b \notin (a \rightarrow u) \} \\
I_2 &= \{ u \in C_i : a \notin (b \rightarrow u) \text{ and } b \notin (a \rightarrow u) \} \\
I_3 &= \{ u \in C_i : a \notin (b \rightarrow u) \text{ and } b \in (a \rightarrow u) \} \\
I_4 &= \{ u \in C_i : a \in (b \rightarrow u) \text{ and } b \in (a \rightarrow u) \}
\end{align*}
\]

Topological sort on the graph associated to the matrix slice gives $I_1, I_2, I_3$
Let \( a, b \) be the elements of \( C_i \) with the more interferences. **Lemma:** no solution with \( \exists z \in \mathcal{H} \) so that \( z \in (a \rightarrow b) \)  
\( \Rightarrow \) Cut between \( a \) and \( b \)!

Finding out how to cut

How to connect parts afterward

First step on \( I_1 \rightarrow \) Finds 2 classes \( I_{1a} \) and \( I_{1a}^\alpha ; a \in I_{1a} \).
First step on \( I_3 \rightarrow \) Finds 2 classes \( I_{1b} \) and \( I_{1b}^\beta ; b \in I_{1b} \).
Let \( a, b \) be the elements of \( C_i \) with the more interferences. **Lemma:** no solution with \( \exists z \in \mathcal{H} \) so that \( z \in (a \rightarrow b) \) \( \implies \) Cut between \( a \) and \( b \)!

Finding out how to cut

How to connect parts afterward

First step on \( I_1 \) \( \rightarrow \) Finds 2 classes \( I_{1a} \) and \( I_{1\alpha} ; a \in I_{1a} \).
First step on \( I_3 \) \( \rightarrow \) Finds 2 classes \( I_{1b} \) and \( I_{1\beta} ; b \in I_{1b} \).
Reconnect \( I_{1a} \) and \( I_{1b} \) ; Reconnect \( I_{1\alpha} \) and \( I_{1\beta} \).
Let $a, b$ be the elements of $C_i$ with the more interferences. **Lemma:** no solution with $\exists z \in H$ so that $z \in (a \to b) \Rightarrow$ Cut between $a$ and $b$!

Finding out how to cut

How to connect parts afterward

First step on $I_1 \to$ Finds 2 classes $I_{1a}$ and $I_{1\alpha}$; $a \in I_{1a}$.
First step on $I_3 \to$ Finds 2 classes $I_{1b}$ and $I_{1\beta}$; $b \in I_{1b}$.
Reconnect $I_{1a}$ and $I_{1b}$; Reconnect $I_{1\alpha}$ and $I_{1\beta}$.

No demonstration of this...
Data collection

Interference measurement between each pair of hosts.

- Naïve algorithm:
  - \( N^4 \), 30s. per step \( \Rightarrow \) 50 days for 20 hosts.

- Speedups thanks to traceroute or other tomography
  - Independent tests in parallel
  - Validation of information sets

- Refinement of existing graph?

Deserve more investigation
Contributions of ALNeM

- Retrieve the interference-based topology from direct measurements
- Strong mathemathical basements (optimal for cliques of trees)
- More generic than ENV (algorithm for cycles)
- 2000 lines of C code; one research report
- Based on GRAS [Quinson03]
Contributions of ALNeM

- Retrieve the interference-based topology from direct measurements
- Strong mathematical basements (optimal for cliques of trees)
- More generic than ENV (algorithm for cycles)
- 2 000 lines of C code; one research report
- Based on GRAS [Quinson03]
  - development on simulator (SimGrid [CLM03]) and immediate deployment
  - target: distributed event-based applications, C language
  - 10 000 lines of C code, Linux, Solaris
  - Submitted to one workshop
Overview

- Introduction
- NWS: Network Weather Service
- FAST: Fast’s Agent System Timer
- ALNeM: Application-Level Network Mapper
- Conclusion
Conclusion

• Major issue on the Grid: collecting data (before scheduling)
Conclusion

- Major issue on the Grid: collecting data (before scheduling)

- Gathering quantitative data: NWS + FAST

  **NWS**: System availability

  **FAST**: Routine needs
Conclusion

- Major issue on the Grid: collecting data (before scheduling)
- Gathering quantitative data: **NWS + FAST**

**NWS**: System availability

- Lower latency
- Better responsiveness
- Process management

**FAST**: Routine needs

Future work:
- Automatic deployment
Conclusion

- Major issue on the Grid: collecting data (before scheduling)
- Gathering quantitative data: **NWS + FAST**

**NWS**: System availability
- Contributions:
  - Lower latency
  - Better responsiveness
  - Process management

**FAST**: Routine needs
- Contributions:
  - Generic benchmarking framework
  - Unified interface to quantitative data
  - Virtual booking
  - Integration: DIET, NetSolve, Grid-TLSE
  - 2 journals; 3 conferences/workshops

**Future work**:
- Automatic deployment
- Integration of Freddy
- Irregular routines (sparse algebra)
- New metrics (like I/O)?
- Yet better integration within NWS
Conclusion

- Major issue on the Grid: collecting data (before scheduling)
- Gathering quantitative data: **NWS + FAST**
- Gathering qualitative data: **ALNeM**

**ALNeM:** Network topology to know about interferences

**Contributions:**
- Strong mathematical basements
- Optimal in size for cliques of trees
- Partial cycle handling
- GRAS: application development tool
- Submitted to one workshop

**Future work:**
- Proof of NP-hardness …
- … or exact algorithm
- Experimentation on real platform
- Optimization of the measurements
- Iterative algo. (modification detection)
- Integration within NWS
- Hosting of DIET
Selected publications

Book chapter: 1 national


Journals: 2 internationals (+ 1 submitted), 1 national


Conferences/workshops: 4 internationals (+ 2 submitted), 2 nationals.

Appendix
GRAS overview

- development on simulator (SimGrid) and deployment without modification
- target: distributed event-based applications
- light virtual machine for the study and development of NWS, ALNeM, ...
- 10 000 lines of code, Linux, Solaris
- Futur: (even higher) performance and portability, interoperability
Sensor in the middle

\[ \text{bp}(AC') = \min (\text{bp}(AB); \text{bp}(BC')) \]
\[ \text{lat}(AC') = \text{lat}(AB) + \text{lat}(BC') \]

It’s a must to reassemble measurements in hierarchical monitoring
A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- Good and simple paradigm to implement the Grid
- Some of the functionalities needed:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- 5 fundamental components:
  - Client
  - Server
  - Agent
  - Monitor
  - Database
RPC and grid computing: GridRPC

A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- Good and simple paradigm to implement the Grid
- Some of the functionalities needed:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- 5 fundamental components:
  
  **Client**  Several user interfaces which submit the requests to servers
  
  Server
  
  Agent
  
  Monitor
  
  Database
A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- **Good and simple paradigm to implement the Grid**
- **Some of the functionalities needed:**
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- **5 fundamental components:**
  
  **Client**  Several user interfaces which submit the requests to servers
  
  **Server**  Runs software modules to solve client’s requests
  
  **Agent**
  
  **Monitor**
  
  **Database**
RPC and grid computing: GridRPC

A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- Good and simple paradigm to implement the Grid
- Some of the functionalities needed:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- 5 fundamental components:

  **Client** Several user interfaces which submit the requests to servers
  **Server** Runs software modules to solve client’s requests
  **Agent** Gets client’s requests and schedules them onto the servers
  **Monitor**
  **Database**
A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- Good and simple paradigm to implement the Grid
- Some of the functionalities needed:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- 5 fundamental components:
  
  **Client**: Several user interfaces which submit the requests to servers
  
  **Server**: Runs software modules to solve client’s requests
  
  **Agent**: Gets client’s requests and schedules them onto the servers
  
  **Monitor**: Monitors the current state of the resources
  
  **Database**
A simple idea: Implement the RPC model over the Grid

- Remote Procedure Call: run a computation remotely
- Good and simple paradigm to implement the Grid
- Some of the functionalities needed:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- 5 fundamental components:
  - **Client**  Several user interfaces which submit the requests to servers
  - **Server**  Runs software modules to solve client’s requests
  - **Agent**  Gets client’s requests and schedules them onto the servers
  - **Monitor**  Monitors the current state of the resources
  - **Database**  Contains static and dynamic knowledges about resources
RPC and grid computing: GridRPC

A simple idea: Implement the RPC model over the Grid

- **Remote Procedure Call**: run a computation remotely
- **Good and simple paradigm to implement the Grid**
- **Some of the functionalities needed**:
  - Computation scheduling, data migration
  - Security, fault-tolerance, interoperability, . . .

- **5 fundamental components**:
  - **Client**: Several user interfaces which submit the requests to servers
  - **Server**: Runs software modules to solve client’s requests
  - **Agent**: Gets client’s requests and schedules them onto the servers
  - **Monitor**: Monitors the current state of the resources
  - **Database**: Contains static and dynamic knowledges about resources

Knowing the platform is crucial for the agent
\[
\text{Temps pdgemm}(M, N, K) = \left\lceil \frac{K}{R} \right\rceil \times \text{temps}_d\text{gemm} + (M \times K) \tau_p^q + (K \times N) \tau_p^q + (\lambda_p^q + \lambda_q^p) \left\lceil \frac{K}{R} \right\rceil.
\]
Hypothesis 1: Routing consistent

- 1-to-N: no merge after branch
- N-to-1: no split after join

Hypothesis 2: Routing symmetric
Algorithm for cliques of trees

1. Initialization: \( i \leftarrow 0; \ C_i \leftarrow \mathcal{H}; \ E_i \leftarrow \emptyset ; \ V_i \leftarrow \emptyset \)

2. Classes lookup: \( h_1, \ldots , h_p \): classes of \( \perp \) over \( C_i \); \( \forall i, l_i \in h_i \)
\[ C_{i+1} \leftarrow \{l_1, \ldots , l_p\} \]

3. Graph update: \( V_{i+1} \leftarrow V_i \); \( E_{i+1} \leftarrow E_i \)
\( \forall h_j \in C_i, \forall v \in h_j \), do \( E_{i+1} \leftarrow E_{i+1} \cup \{(v, l_j)\} \) and \( V_{i+1} \leftarrow V_{i+1} \cup \{v\} \)

4. Interference matrix update

Let \( l_\alpha, l_\beta, l_\gamma, l_\delta \in C_{i+1} \) represent respectively \( h_\alpha, h_\beta, h_\gamma, h_\delta \).
For each \( m_\alpha, m_\beta, m_\gamma, m_\delta \in C_i \) so that \( m_\alpha \in h_\alpha, m_\beta \in h_\beta, m_\gamma \in h_\gamma, m_\delta \in h_\delta \).
\( I(C_{i+1}, \bar{\chi})(l_\alpha, l_\beta, l_\gamma, l_\delta) = I(C_i, \bar{\chi})(m_\alpha, m_\beta, m_\gamma, m_\delta) \)

5. Iterate 2–3 until \( C_i = C_{i+1} \).
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
ALNeM: example of execution
Goal: Metacomputing platform (GridRPC model)

- Complete and ready to use for users
- Extensible by researchers

Main functionalities:

- Distributed and hierarchical scheduling;
- Resources localization;
- Data persistence;
- Platform monitoring;


Targeted applications: Grid-ASP

- Digital elevation model (Geology – LST ENS-Lyon);
- Molecular dynamics (Physique – Lyon-I et al.);
- HSEP (chemical – SRSMC Nancy);
- Circuit simulation (electronic – Ircom);
- ACI TLSE (sparse matrix expertise – Toulouse);
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation: FAST (NWS)
4. Back to MA: distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation : FAST (NWS)
4. Back to MA : distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation : FAST (NWS)
4. Back to MA : distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation: FAST (NWS)
4. Back to MA: distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation : FAST (NWS)
4. Back to MA : distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation: FAST (NWS)
4. Back to MA: distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation: FAST (NWS)
4. Back to MA: distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation: FAST (NWS)
4. Back to MA: distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
DIET : Handling of a request

1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation : FAST (NWS)
4. Back to MA : distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection
1. Clients connect to the MA
2. Request transmission to servers
3. Performance evaluation : FAST (NWS)
4. Back to MA : distributed scheduling
5. (Broadcast if impossible in local tree)
6. Result sent back to the client
7. Direct client-server connection