Research Unit UTIC
ESSTT, University of Tunis

Laboratory LIPN , UMR 7030
CNRS, University of Paris 13

How to decentralize Desktop Grid middlewares, lessons learned and future works

Heithem Abbes, Christophe Cérin, Mohamed Jemni
christophe.cerin@lipn.univ-paris13.fr

DAAD Summer School on Current Trends in Distributed Systems
26/09/2009

Outline

- Introduction (the concept of Desktop Grids)
- Objectives of the talk
- Two visions in our research group
  - BonjourGrid
  - PastryGrid
- Other visions for the future of Destop Grids

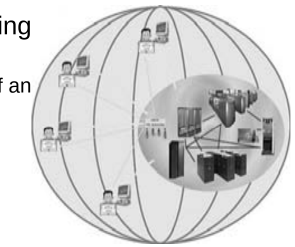
Introduction 1/3

- P2P systems have allowed large improvements in the field of file sharing over Internet.
- Gnutella, Kazaa and Freenet

→ Decentralized architecture
No **coordination** between machines

Introduction 2/3

- Grid computing : obtaining an infrastructure offering computing power for users applications.
  - **Coordination** between machines during the execution of an application.
  - Centralized or hierarchical architectures (Globus, Glite, Condor).



→
- No scalability
- Complicated procedure of installation
- Complicated configuration phase for an ordinary user

Examples : TeraGrid, EGEE and OpenScience Grid (OSG) :

Introduction 3/3

- Desktop Grid led the community to build computing systems based on voluntary machines.
- Current systems use Master/Worker model
- United Devices, BOINC, PLANETLAB, XtremWeb, Condor

Condor High Throughput Computing

SETI@HOME

BOINC

PLANETLAB

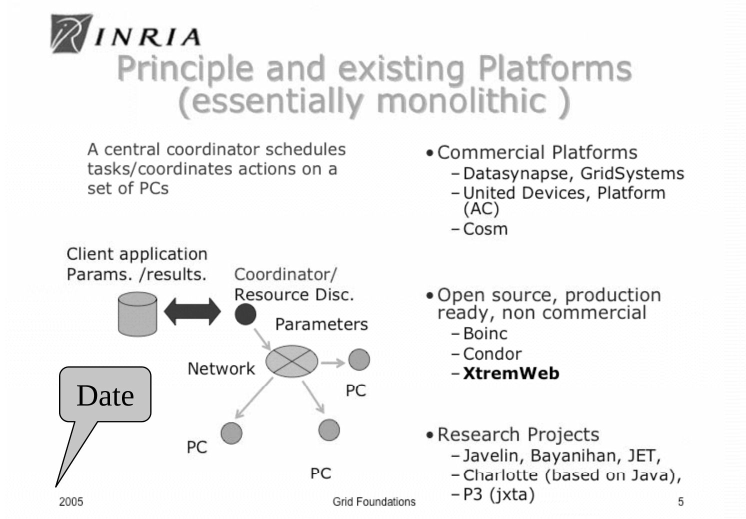
XtremWeb-CH

- Applications domain
  - Global climate prediction (BOINC)
  - Search for extraterrestrial intelligence (SETI@Home)
  - Cosmic rays study (XtremWeb).

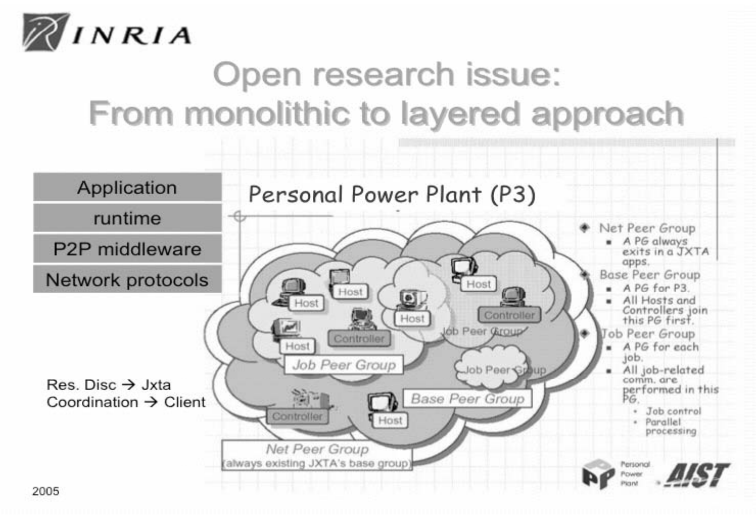
✓ Demonstrate the potential of Desktop Grid

✗ To suffer from being hardly scalable due to centralized control
✗ To rely on permanent administrative staff who guarantees the master operation

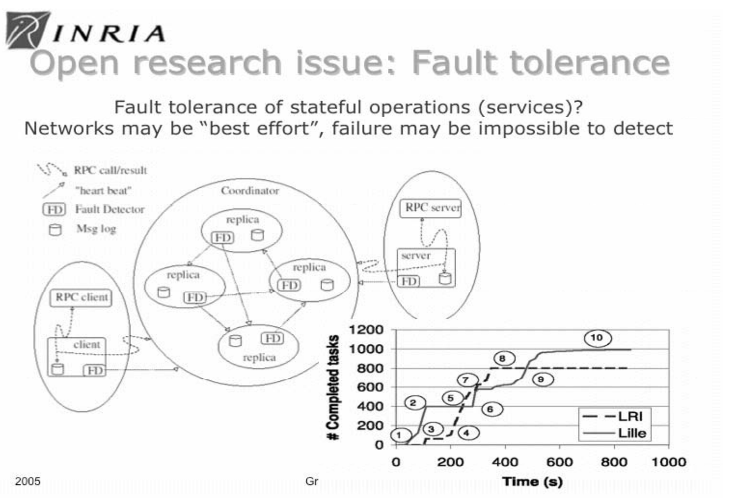
The history of Desktop Grids by Franck Cappello



The history of Desktop Grids by Franck Cappello



The history of Desktop Grids by Franck Cappello



Objectives of this talk

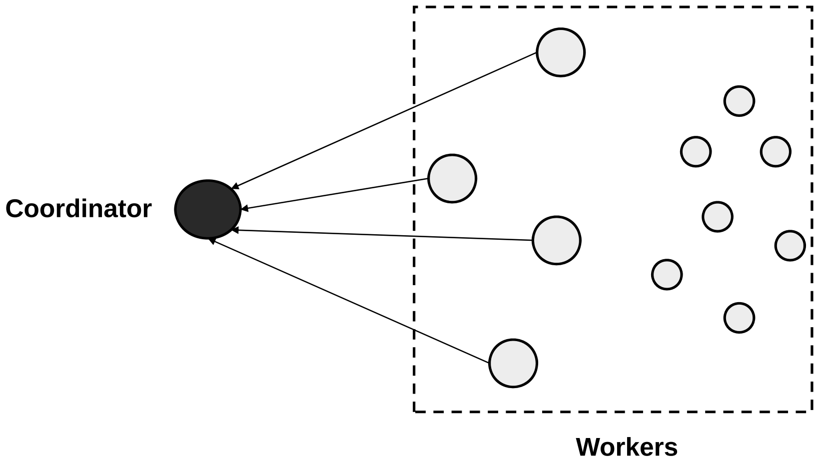
- To offer a comprehensive survey of (some) hot topics in Desktop Grids
- To illustrate with innovative middleware
- To explain views from other researchers
- To motivate people to join our community

BonjourGrid Middleware

- To offer a collaborative, decentralized and multi-coordinators platform
- To build an infrastructure which does not depend on a central element.
- No static coordinator
- To create coordinators in a dynamic, automatic way and without system administrator intervention
- Each coordinator asks and seeks, in a decentralized way, idle machines to participate to the execution of a given application
- Pluggable computing systems

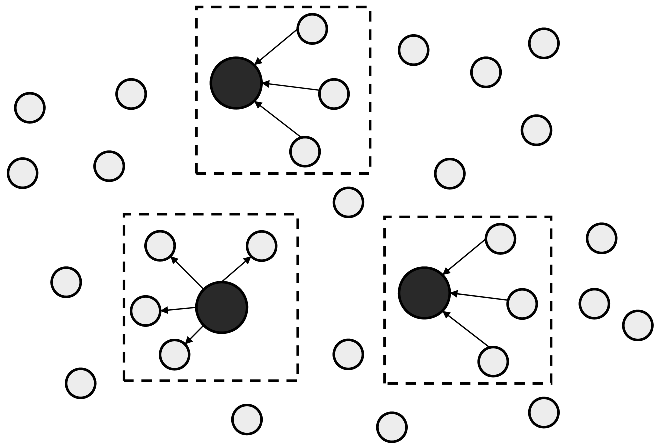
Collaborative Grid | Meta-Grid

BonjourGrid : Basic design (1/3)



Computing Element (CE) = **1** Coordinator + **N** Workers

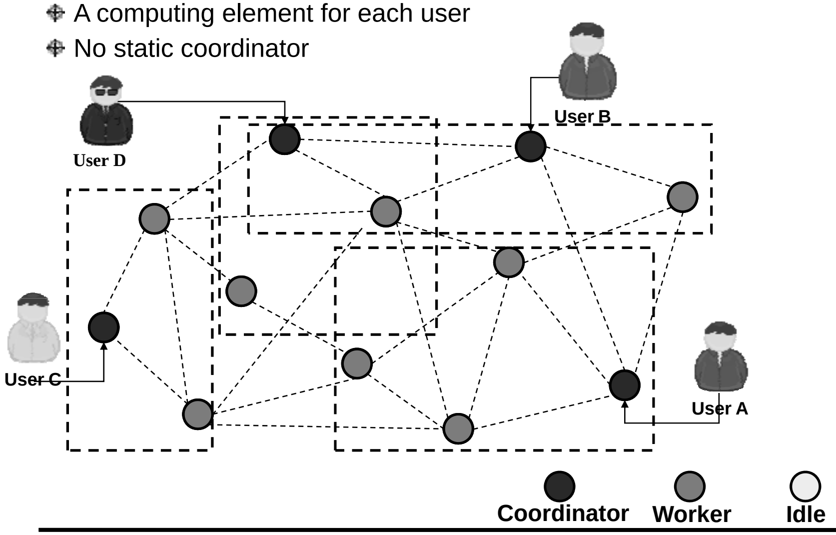
BonjourGrid : Basic design (2/3)



Control and coordinate multiple instances through Pub/Sub system

BonjourGrid : Basic design (3/3)

- A computing element for each user
- No static coordinator



Advantages in using pub/sub systems

- Only 3 primitives in the toolbox: publish, subscribe, browse
- Notions of global state and global/multicast operations
- Easy to develop applications

BonjourGrid's vision

- A user requests for a computation
- He provides tasks graph and codes implementing his distributed application
- He deploys locally a coordinator node and requests for participants
- The coordinator node selects a subset of idle nodes (CPU, RAM, Cost)
- When the coordinator node finishes application tasks it becomes free and returns to the idle status

  ➔ Its worker nodes become idle

Fundamental Parts

- BonjourGrid is composed of three fundamental parts:
  - ***A resources discovery protocol***
    - Fully decentralized protocol
  - ***A "computing elements" constructor***
    - Executes and handles the various tasks of an application (XtremWeb, Condor, Boinc, MPICH)
  - ***A global coordination protocol***
    - Manages and controls all resources, services and computing elements
    - Does not depend on any specific machine or centralized element

Discovery protocol

- Based on Bonjour protocol
- Bonjour is an implementation by Apple of the ZeroConf protocol
  - To obtain a functional IP network without DHCP or DNS servers
- Bonjour is structured around three functionalities:
  - Dynamic allocation of IP addresses without DHCP
  - Resolution of names and IP addresses without DNS
  - Services discovery without directory server

Analysis of Pub/Sub systems

- Questions ?
  - Can a system based on publish/subscribe be scalable?
  - What is the response time to publish a service?
  - What is the discovery time of a service?
- Experiments on Bonjour protocol, on the Grid5000 platform using more than 300 nodes (Orsay site)
  - Bonjour is reliable and very powerful in resources discovery.
  - It discovers all the services (100%)
  - It succeeds to discover more than 300 services published simultaneously in less than 2 seconds

A computing element (CE)

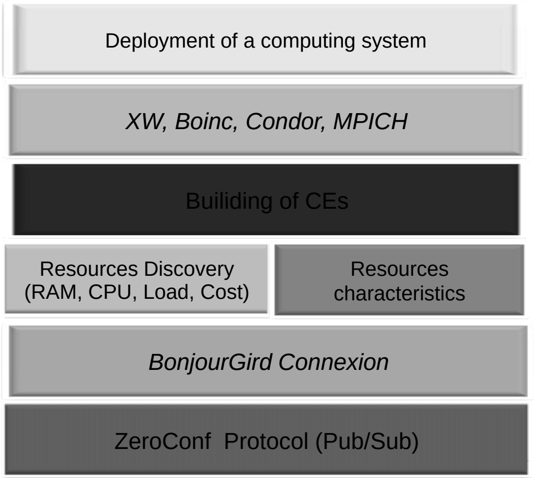
- Each user uses his machine as a coordinator
- Each coordinator creates dynamically its CE
  - ❑CE = Coordinator + set of workers
- CE functionalities
  - ❑To allocate workers
  - ❑To submit and run tasks on workers
  - ❑To schedule and get results
- Computing systems
  - *XtremWeb, Condor, Boinc, MPICH*

➡ Specific CE middleware for each user

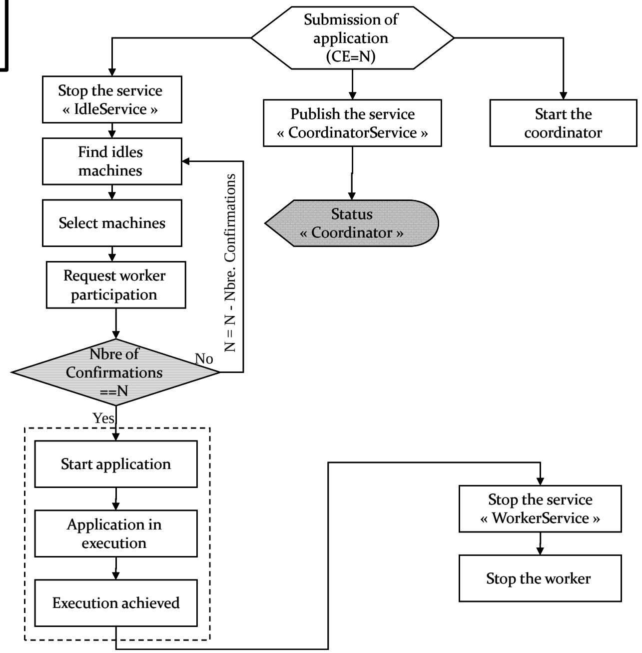
BonjourGrid node state

- Each machine can have one of the three states (Idle, Worker or Coordinator).
- A machine announces its state by publishing the <u>specific service</u> to this state
  - ***IdleService*** for Idle state
  - ***WorkerService*** for worker state
  - ***CoordinatorService*** for coordinator state.
- When a machine state changes:
  - it publishes the according service to advertise this new state,
  - after having deactivated the old one.
- Every machine can discover machines that are in a given state.

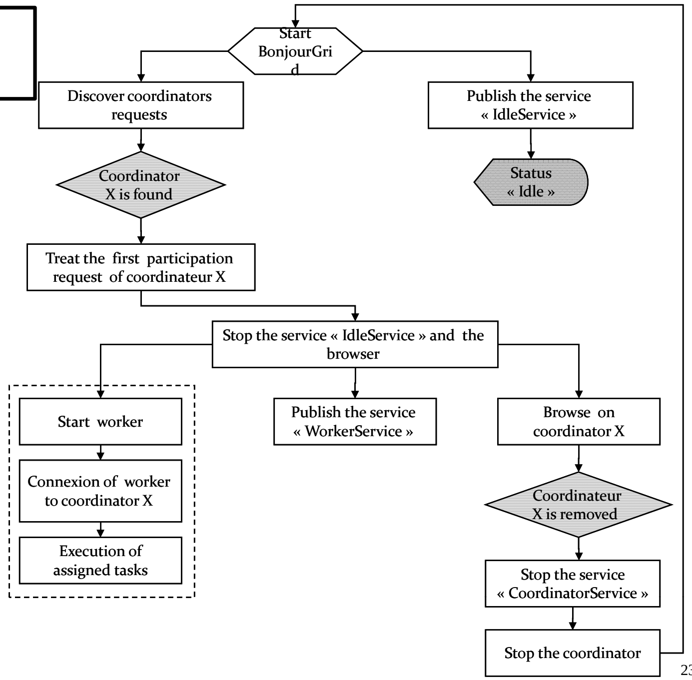
BonjourGrid's Layers



Idle into Coordinator



Idle into worker



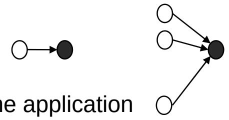
Experimentation

- Evaluate a system
  - based on a set of specific applications ?
  - based on a specific arrival pattern (Poisson's Law) ?
- Workload model very close to the reality
  - Feitelson and Lublin
  - Inputs of the workload model
    - Number of nodes (system size)
    - Arrival time of applications
    - Maximum number of parallel tasks
    - Tasks execution times

Application ID	Arrival Time	Execution Time	Nbre of parallel tasks
1	19	4	32
2	39	11	13
3	69	13	16
4	98	87	1
5	200	100	4

Experimentations

- To emulate a set of users with a set of applications
- An application is submitted by a user
- To create CEs to carry out all submitted applications
- A python generator which creates a set of applications using a workload model
- An application is created for each entry in the workload with different parameters
  - Binary and data files
  - A data flow graph (in XML format)
  - Gather : fictitious task to detect the final date of the application
- Size of an application varies from 2 to N tasks
  - N may be equal to the maximum number of available machines in the network



Experimentations

- 1 CE is created dynamically for each application
- Emulator
  - List of machines
  - List of applications
  - Workload model
- Submit an application following the arrival pattern of applications in the workload
- Look for free machine on which a coordinator will start to initiate the application tasks execution
- The CE is released when application tasks finish

Experimentations

Grid'5000 *

A very brief overview

A nation wide Experimental Grid

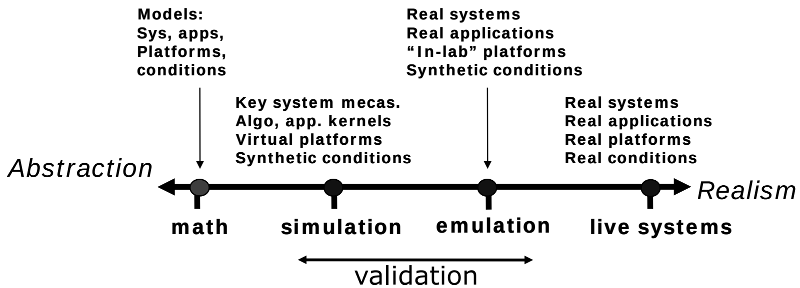
Franck Cappello
(with all project members)
INRIA
fci@lri.fr

INRIA Renater

Experimentations

Tools for Distributed System Studies

To investigate Distributed System issues, we need:
1) Tools (model, simulators, emulators, experi. Platforms)



2) Strong interaction between these research tools

Experimentations

Comparative study of BonjourGrid and XW

- ❑Execution time = Final date – Submission date
- ❑Same set of applications
- ❑Same workload model
- ❑N machines for BonjourGrid = 1 Coordinator + N-1 workers for XW

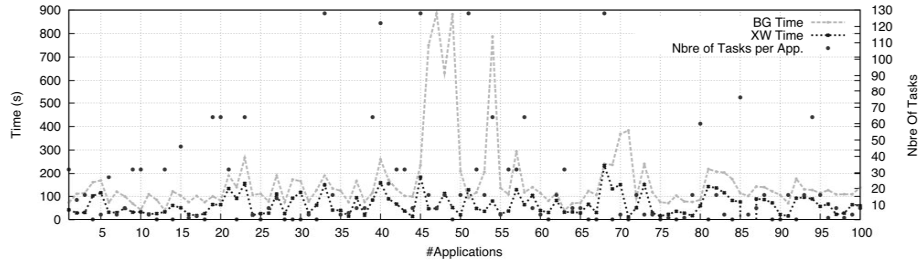
1st Setup

- ❑128 machines on Grid5000 (Orsay's node).
- ❑Creation of 100 applications with // tasks (1 à 128) (2150 tasks)
- ❑3 hours of execution (arrival dates are elapsed within this period)
- ❑Management of 100 instances of CE

Experimentations

Results

- BonjourGrid generates an overhead of about 60 sec for 90 % applications
- Waiting time to find a free machine for a coordiantor (delay the end time of an application with BonjourGrid) + CE building Time
- BonjourGrid can give, in some cases, better turnaround times than XW
  - XW-Coordinator: access to mysql DB, task assignment, workers allocation ➡ overload of the coordinator
  - Loss of workers connections (same port on the central server to maintain connections)
  - XW-Coordinator must wait for WorkRequest to submit a task (back-off effect) ➡ submission tasks may delay



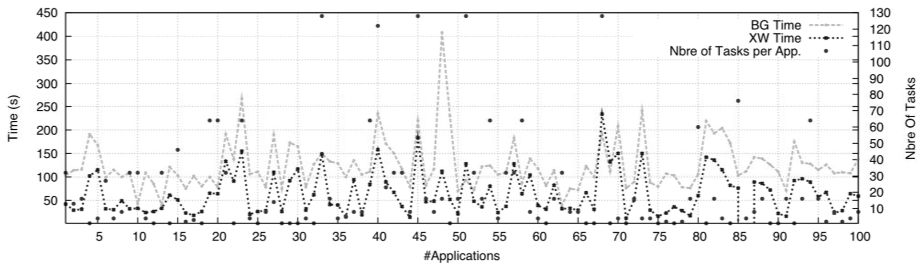
Experimentations

2nd Setup

- 1st setup with 20 (15%) machines more for BonjourGrid.
- Relaxing the previous scenario
- Give more chance to BonjourGrid to find a free machine for the coordinator.

Results

- The difference between BonjourGrid and XW turnaround times is decreased
- 1 single peak with a diminution of 140 sec (3 applications which have requested more than 120 workers)
- BonjourGrid can provide better performance with other more relaxed scenarios.



Experimentations

- To make more experiments over a large number of machines
- To check if BonjourGrid scale well? Can it orchestrate a great number of CEs?
- To use virtual machine to increase the experiment scale
- To use the virtual system **Vgrid**
- Vgrid provides a mechanism to create several virtual machines on the same host
- The different virtual machines communicate through a virtual Hub created on each physical machine (or real machine denoted by RM).
- 500 virtual machines means:
  - ❑10 VM*50 RM or 5VM*100 RM.

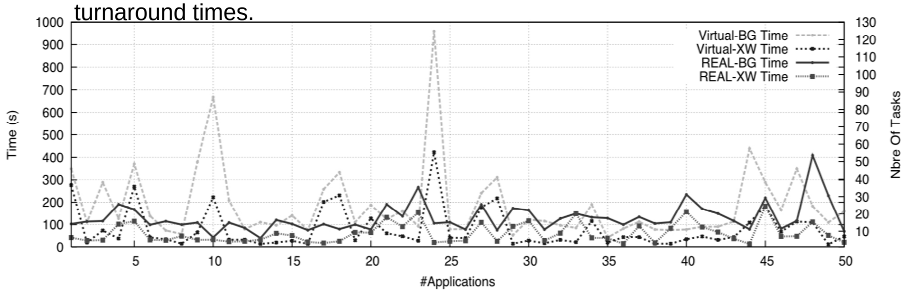
Experimentations

3rd Setup

- 500 VM = 4 VM x 125 RM
- 50 applications already executed on128 real machines (RM)
  - ❑ To observe the impact of virtual machines(VM) use.

Results

- Virtual machines lacked sufficient RAM to manage many opened sockets at the same time and to allocate necessary memory for the Java virtual machine.
- The increase in machines number from128 RM to 500 VM did not enhance turnaround times.



Experimentations

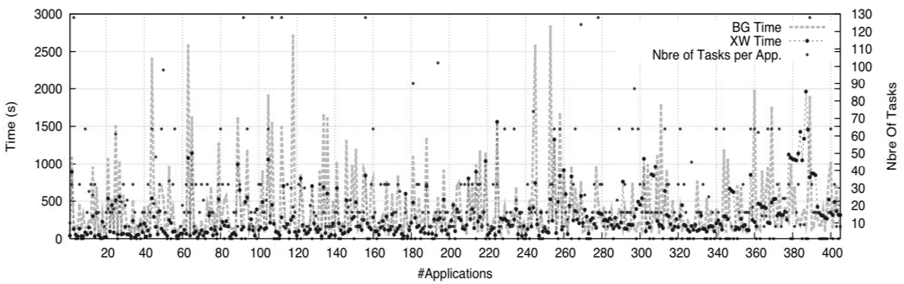
4th Setup

- 405 applications ===> 405 CE managed by BonjourGrid
  - ❑ The parallel tasks number varies from 2 to 128
- BonjourGrid : 1000 VM (4 VM x 250 RM)
  - ❑ 300 MB per each VM
- XW: 1 coordinator + 480 workers
  - ❑ special VM with 1500 MB for the coordinator to reach 480 connexions of workers
  - ❑ 300 MB per each worker

Experimentations

Results

- Bonjour can manage more than 400 CEs
- BonjourGrid performs better around the application 380
  - ❑ The overload of XW-Coordinator
  - ❑ The lose of workers connexions
  - ❑ Back-off effect



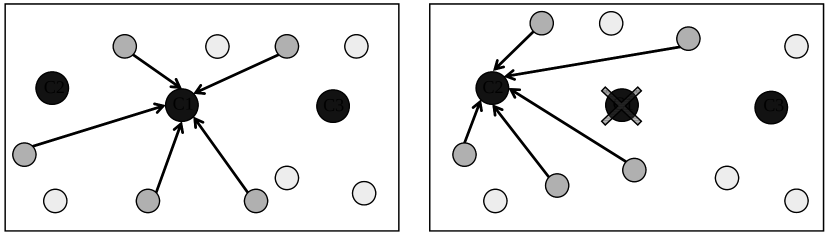
Fault tolerance in BonjourGrid

- A CE is managed by a computing system (XtremWeb, Boinc, Condor)
  - A computing system handles fault inside a CE (faults of workers)
  - BonjourGrid must tolerate the coordinator fault

➡ A replication approach of coordinator states

Fault tolerance in BonjourGrid

- Replication approach:
  - Create dynamically replicas (C2 and C3) of the principal coordinator (C1)
  - Use virtual machines (Xen) to save the state (checkpoints) of the principal coordinator
  - Send periodically checkpoints to replicas (C2 and C3)
  - Use publish/subscribe infrastructure to exchange information and detect failure
  - Redirect workers to the new coordinator (C2)



Conclusion about BonjourGrid

- BonjourGrid: A novel approach for making a collaborative and decentralized Desktop Grid systems.
- Publish/Subscribe protocol
  - Orchestrate the participants
- A computing system (e.g XW, Boinc, Condor, MPICH) for the execution level of an application
- BonjourGrid makes a distributed control over resources and does not depend on a central element.
- BonjourGrid favors collaborative execution and Meta-Grids orchestration

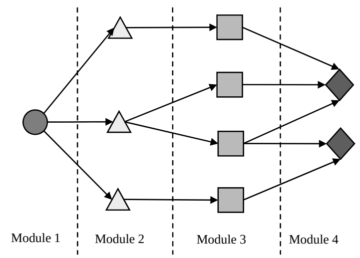
PastryGrid objectives

- Decentralize the execution of distributed applications with precedence between tasks
- New approach:
  - PastryGrid: a fully decentralized system based on DHT which decentralizes the execution of distributed applications.
  - Decentralize the resources management in the Desktop Grid (DG)

Distributed Application (1/4)

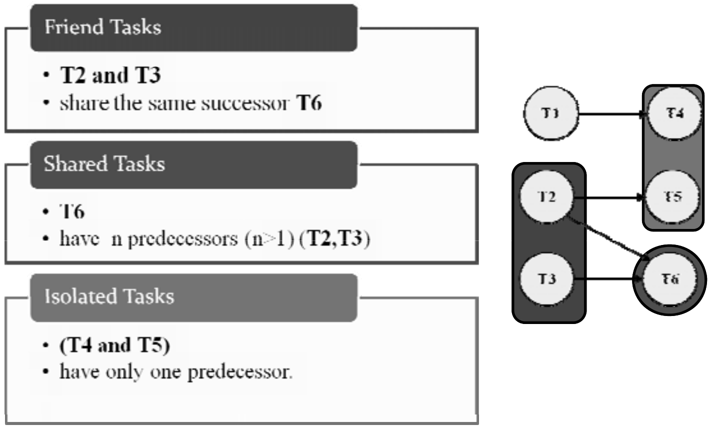
Chalenge : Distributed applications with precedence between tasks:

- DA is composed of several modules
- A module is a set of tasks, using the same binary and, generally, different input files.
- Tasks of the same module can be carried out in a parallel way



Distributed Application (2/4)

Terminology



Distributed Application (3/4)

- How a user can describe his application?
- He provides a compressed package : binary, data , data flow graph.
- Data flow graph: nodes are tasks, edges are precedence between tasks
- *Application.xml* to describe his application
- Data flow graph
- Execution requirement
- Path of binary file and data

→ NOT easy to write it by hand

Distributed Application (4/4)

- SDAD : System for Distributed Applications Description
- Help user to prepare his package
- A tool with graphical mode for simple applications and an advanced mode for complex ones
- User has just to draw the data flow graph for simple applications or follow special wizard for complex ones.
- SDAD generates the ***Application.xml***, puts data and binary files in a directory tree, and compresses all these data in a zip file.

SDAD

Design of PastryGrid

- PastryGrid is composed of four components:
  - Addressing scheme to identify machines and applications
  - RDV concept
  - Protocol of resources discovery
  - Coordination approach between machines carrying out a given application

1. Addressing scheme

- Withdraw the master/worker model ➔ Decentralized style
- Modern P2P networks: Pastry, CAN, CHORD
  - Based on DHT (Distributed Hash Table)
  - Nodes can join/leave the network
  - DHT is updated and mapped to new physical nodes
  - Fully decentralized routing algorithm
  - delivery of lookup messages to the appropriate physical node in at most O(log(N)) (N = number of alive physical nodes)
- Pastry : overlay network for the PastryGrid system.

1. Addressing scheme

- Node: is assigned with a 128-bit ***nodeId*** to each physical node when it joins the network.
- Application:
  - is assigned with a 128-bit ***ApplicationId***
  - hashed using application and user name and current date on the submission machine
  - Unique identifier for each application

2. Concept of RDV

- How to localize a node without identifier (neither IP address neither Hostname) ?
- From where can I get my data (binary + data) ?
- Where should I store my results ?

Solution

→ RDV created according to the application identifier ApplicationID

Communication with RDV via ApplicationID→

2. Concept of RDV

Coord.

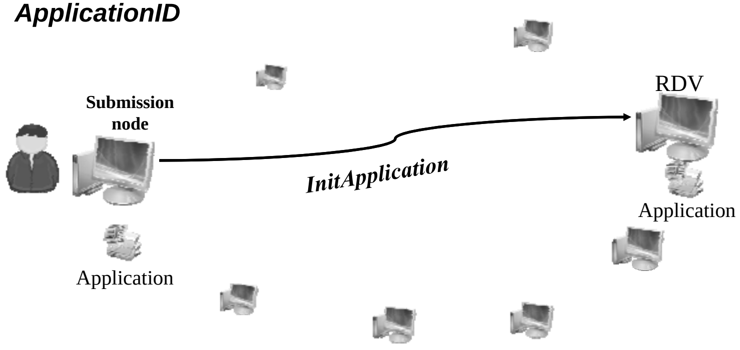
- Known and fixed
- Static central element
- In case of breakdown, the system collapses
- Centralised management of resources
- Management of all applications
- Simple
- Overload

RDV

- Unknown
- Variable
- In case of breakdown the system remain functional
- Decentralised resources management
- RDV for each application
  - no overload

RDV initialization

- RDV is the machine which ***nodeId*** is numerically closest to ***ApplicationID***



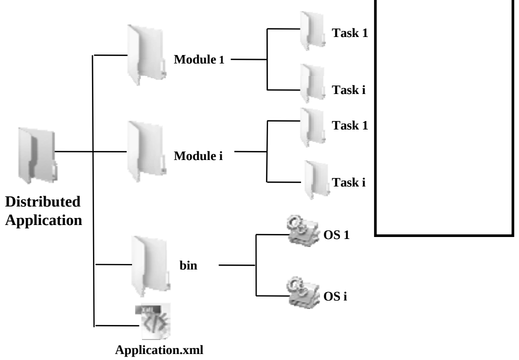
RDV initialization

Application

- Extracts the zip file to rebuild the initial tree structure of the application
- Generates ***Task.xml*** file for each task (Application.xml)

Info in ***Task.xml*** of **Ti:**

- Task name Ti
- successors of Ti (Succ(Ti))
- Requirements of Succ(Ti)
- Friends of Ti

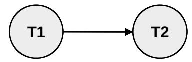


3. Resources discovery protocol

- Each node M participates in the execution of a task T, is susceptible to look for free machines for the successors of M
- No machine dedicated for research (to avoid centralisation)

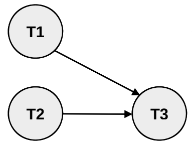
Notation: Mi execute the task Ti

<u>Case 1:</u>



M1 finishes **T1** then looks for **M2** for **T2**

<u>Case 2:</u>



Avoid the research redundancy
Only one machine, among M1 and M2, looks for M3
Optimization: the last that finishes, looks for M3

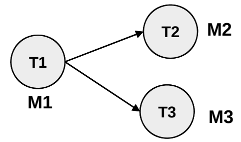
3. Resources discovery protocol

New discovery protocol

- Implementation: simplicity of grafting and controlling the discovery module
- Existing : a machine sends its local information (CPU, RAM, OS, Cost) to one or many servers ➔ Not real information (real time)
- Proposal solution: to access to the machine and to check directly its characteristics

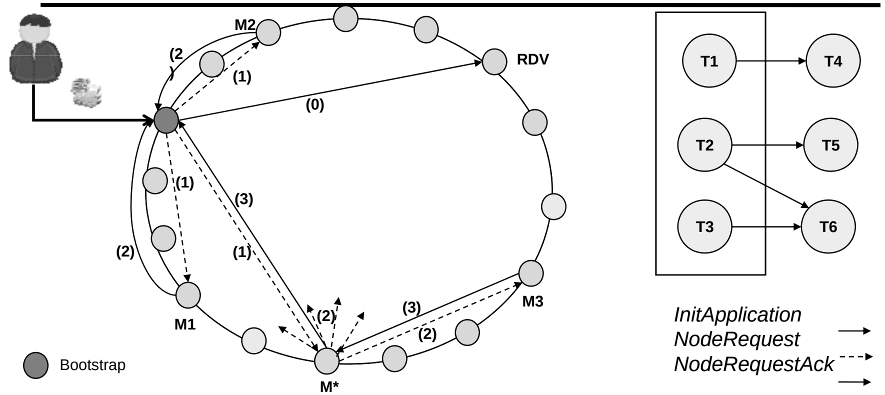
3. Resources discovery protocol

M1 executes **T1** and search available and suitable machines for **T2** and **T3**



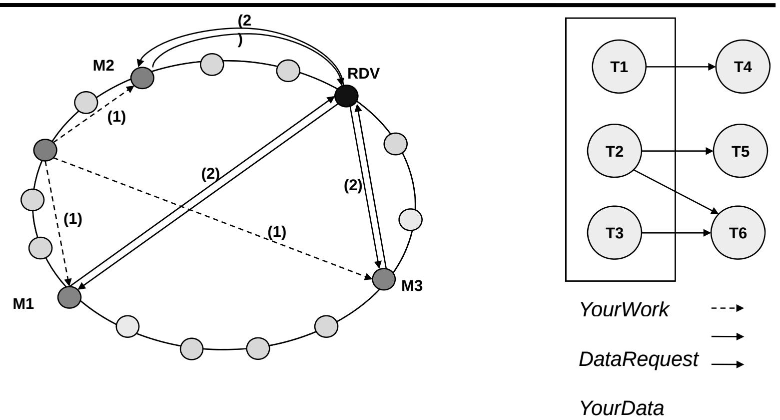
1. **M1** constructs a vector **Va (H,W,R)** :
   - **a** : application ID
   - **H** : list of nodes handles of **M1** leafset
   - **W** : list of tasks names (**T2**, **T3**)
   - **R** : execution requirement
2. **M1** sends **Va(H,W,R)** to **head(H)**
3. **M=head(H)** checks if it is free and fits **R**: if yes, **M** takes **T**=**head(W)** to execute it and deletes it from **W** (**W=W-T**).
4. If **W=∅**, then the discovery process is accomplished. Else, **M** deletes its node handle from **H** (**H=H-M**). Goto 2
5. If **H=∅**, **M** updates it with a new leaf set: **H** leaf set of **M**. **M** forwards the new **Va(H,W,R)** to the **head(H)**. Goto 2

Coordination and data transfer



- Hash (Application + User + Date) ➔ Unique Identifier: ***ApplicationId***
- Initialization of ***RDV*** ➔ machine which node is numeriquely closest to ***ApplicationId***
- Lookup for free machines for **T1**, **T2** et **T3** ➔ **M1, M2 and M3**

Coordination and data transfer



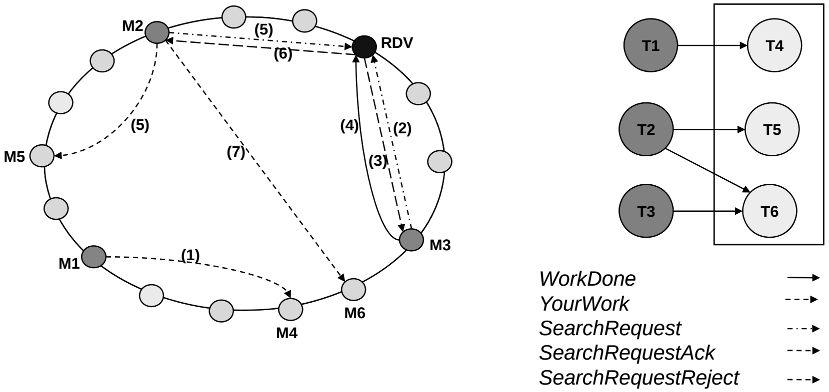
Assignment of tasks ***T1***, ***T2 and T3*** to ***M1***, ***M2*** and ***M3***

➔ ***YourWork*** message containing the task name and ***ApplicationId .***

Demand and recuperation of the data by ***M1***, ***M2*** and ***M3***

➔ ***DataRequest*** and ***YourData***

Coordination and data transfer



- **M1** assigns **T4** to **M4** which it has just found
- **M3** finishes **T3** but do not search a machine for **T6**
- **M2** searches **M5** and **M6** and affect them to **T5** and **T6**

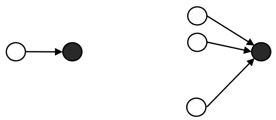
Implementation and experimentation

- PastryGrid is entirely developed in JAVA
- FreePastry API to create the overlay network and implement DHT functionalities
- Comparison study between a central model, XW-CH, and PastryGrid
- To validate and evaluate PastryGrid
- workload model very close to the reality
- Feitelson and Lublin
- Inputs of the workload model
- Number of nodes (system size)
- Arrival time of applications
- Maximum number of parallel tasks
- Tasks execution times

Application ID	Arrival Time (s)	Execution Time (s)	Nbre of parallel tasks
1	19	4	32
2	39	11	13
3	69	13	16
4	98	87	1
5	200	100	4

Implementation and experimentation

- Extension of the basic model by adding a fictitious "gather" task



- "gather" receive all results of the previous tasks.
- An application with k tasks means that k-1 friend tasks and the k^{th} is the gather one
- Stressful scenario (Big number of friend tasks)

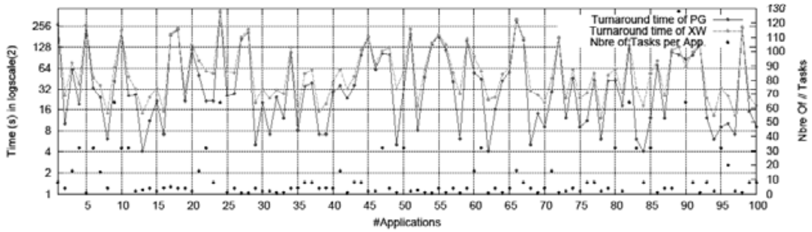
Experimentations

- Grid'5000 platform using Orsay's node
- 205 machines Amd Opterons, 2Go RAM, connected via 1GB/s network
- "Python" based applications generator
- Setup :
  - PastryGrid : 205 machines
  - XtremWeb : 1 coordinator + 204 workers
  - 100 applications (2 to 129 // tasks), 2500 tasks
  - Task turnaround time varies from 1 to 450 sec.
  - 3 hours as period for arrival pattern

Experimentations

PastryGrid gives turnaround times < XW ones :

- XW-Coordinator: access to mysql DB, task assignment, workers allocation ➔ overload of the coordinator
- Loss of workers connections (same port on the central server to maintain connections)
- XW-Coordinator must wait for WorkerRequest to submit a task ➔ submission tasks may delay



Fault tolerance in PastryGrid

RDV fault tolerance▪

Active replication of RDV▪

Maintain a fixed replication coefficient ▪

Last state (checkpoint) saved on replication machines▪

Migration on a new RDV copy, numerically closest to the ▪

old one in case of fault

PastryGrid vs Vigne

Vigne (Christine Morin, INRIA, Rennes (2007))

App. manager is also created according to the application ID ▸

App. Manager makes "ALL", controls the tasks, looks for ▸ resources, sends data…

App. Manager supervises participants nodes. ▸

PastryGrid

RDV: primordial function: a simple storage point of results and ▸ data

RDV does not make research ▸

A direct data exchange may take place between the nodes ▸ without crossing by the RDV

A different research strategy : collaborative research. ▸

Participants, permanently, share information between themselves ▸ via the RDV.

Conclusion about PastryGrid

PastryGrid

- Executes distributed applications with precedence between tasks in a decentralized way
- Collaborative resources and tasks management between participating nodes.
- No overhead caused by decentralization

Views from other researchers

- Jon Weissman, University of Minnesota, USA, (PCGrid'2009 talk in Rome)
- Promote proxy network comprised of volunteer nodes and how proxies accelerate applications spanning one or more clouds.
- Describe how several classes of cloud applications might be better suited to an on-demand cloud comprised of distributed volunteer resources

Views from other researchers

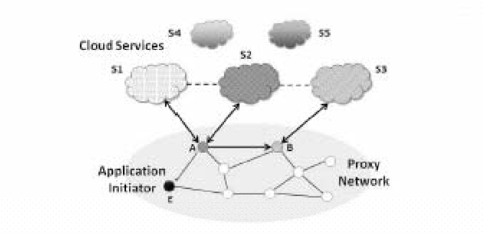


Figure 1: System Model: S_i are the cloud services, nodes A and B are proxies, and E is an initiator for an application that uses clouds S_1, S_2 and S_3. Solid arcs represent actual proxy-to-proxy and proxy-cloud interactions, and dotted lines represent logical cloud-to-cloud interactions.

Proxies may serve as:

Cloud service interaction:

Computing: A proxy may carry out computations on data via a set of data operators.

Caching:

See : http://www-users.cs.umn.edu/~jon/

Views from other researchers

- David Anderson, University of Houston, USA, Keynote Talk at GPC'2009 (Geneva):
- Exa-Scale Volunteer Computing
- Also available on http://www.researchchannel.org/prog/displayevent.aspx?fID=569&rID=27420
- Mentioned (rank=2) the problem of result certification
- Certification is not the process that flotting points operations are 'good' but refers to certify that the result is correct and not falsified

Views from other researchers

- Techniques for result certification
- Redundancy + votes (consensus in distributed systems is hard)
- Probabilistic certification (massive attacks) :
- http://moais.imag.fr/membres/jean-louis.roch/perso_html/publications.html
- Without any specific secure system is the architecture: see http://www.loria.fr/~ejeannot/aleae-doc/Canon-Collusion.pdf
- Use TPM?

Some others perspectives in our group

Short term

- Production versions of BonjourGrid and PastryGrid
- Rich user interface that helps users of BonjourGrid and PastryGrid to deploy their applications
- Extended evaluation of BonjourGrid and PastryGrid
  - Real applications deployment
  - Collaboration of other research teams (physics, numeric simulation, bio-informatics…)

Some others perspectives in our group

Mean and long term

- Fault tolerance
  - PastryGrid : checkpoints to not re-execute long tasks
  - BonjourGrid : optimization of data exchange between virtual machines
- Dynamic infrastructure of services using BonjourGrid
  - Create services on demand (storage, computing, ..)
- Reservation rules to share, optimally, resources between users (BonjourGrid)
- Resource authentication in BonjourGrid and PastryGrid
- Interaction between computing elements in BonjourGrid
  - Negotiations to exchange workers between coordinators (XW, Condor, Boinc)