Fault-Tolerance for PastryGrid Middleware

Christophe Cérin\textsuperscript{1}, Heithem Abbes\textsuperscript{1,2}, Mohamed Jemni\textsuperscript{2}, Yazid Missaoui\textsuperscript{2}

\textsuperscript{1}LIPN, Université de Paris XIII, CNRS UMR 7030, France
\textsuperscript{2}UTIC, ESSTT, Université de Tunis, Tunisia

HPGC’10 - IPDPS
Outlines

1. Introduction
2. PastryGrid
3. Fault Tolerance in PastryGrid
4. Conclusion
**Desktop Grid Architectures**

### Desktop Grid

#### First Gen Architecture

- **Centralized architecture**
- **Monolithic architecture**

- **Client application**
  - Parameters
  - Results

- **Coordinator/Resource Disc.**

- **PC**
- **Firewall/NAT**

### Key Points

- **Federation of thousand of nodes;**
- **Internet as the communication layer: no trust!**
- **Volatility; local IP; Firewall**
Desktop Grid Architectures

Future Generation (in 2006)

- **Distributed Architecture**
- Architecture with modularity: every component is “configurable”: scheduler, storage, transport protocols
- Direct communications between peers;
- Security;
- Applications coming from any sciences (e-Science applications)
In search of distributed architecture

PastryGrid

- An approach based on structured overlay network to discover (on the fly) the next node executing the next task
In search of distributed architecture

**PastryGrid**
- An approach based on structured overlay network to discover (on the fly) the next node executing the next task
- Decentralizes the execution of a distributed application with precedences between tasks
PastryGrid’s overview

Main objectives

- Fully distributed execution of task graph;
PastryGrid’s overview

Main objectives
- Fully distributed execution of task graph;
- Distributed resource management;
PastryGrid’s overview

Main objectives

- Fully distributed execution of task graph;
- Distributed resource management;
- Distributed coordination;
PastryGrid’s overview

Main objectives

- Fully distributed execution of task graph;
- Distributed resource management;
- Distributed coordination;
- Dynamically creation of an execution environment;
PastryGrid’s overview

**Main objectives**

- Fully distributed execution of task graph;
- Distributed resource management;
- Distributed coordination;
- Dynamically creation of an execution environment;
- No central element;
PastryGrid’s overview

Main objectives

- Fully distributed execution of task graph;
- Distributed resource management;
- Distributed coordination;
- Dynamically creation of an execution environment;
- No central element;
PastryGrid’s Terminology

Task terminology

- **Friend tasks**: $T_2, T_3$ share the same successor ($T_6$)
PastryGrid’s Terminology

**Task terminology**

- **Friend tasks**: $T_2, T_3$ share the same successor ($T_6$)
- **Shared tasks** $T_6$: has $n > 1$ ancestors ($T_2, T_3$)
PastryGrid’s Terminology

**Task terminology**

- **Friend tasks**: $T_2, T_3$ share the same successor ($T_6$)
- **Shared tasks** $T_6$: has $n > 1$ ancestors ($T_2, T_3$)
- **Isolated tasks** $T_4, T_5$: have a single ancestor
PastryGrid’s Terminology

Task terminology

- **Friend tasks**: $T_2, T_3$ share the same successor ($T_6$)
- **Shared tasks** $T_6$: has $n > 1$ ancestors ($T_2, T_3$)
- **Isolated tasks** $T_4, T_5$: have a single ancestor

Example
PastryGrid components

- Addressing scheme to identify applications and users (based on hashing application name + submission date + user name — DHT (Pastry))
PastryGrid components

- Addressing scheme to identify applications and users (based on hashing application name + submission date + user name — DHT (Pastry))
- Protocol of resource discovering; No dedicated nodes for the search of the next node to use → on the fly! Optimization: the machine that terminates the last starts the search.
PastryGrid components

- Addressing scheme to identify applications and users (based on hashing application name + submission date + user name — DHT (Pastry))
- Protocol of resource discovering; No dedicated nodes for the search of the next node to use → on the fly! Optimization: the machine that terminates the last starts the search.
- Rendez-vous concept (RDV); Objectives: localisation of a node without IP; task coordination; data recovery;
PastryGrid components

- Addressing scheme to identify applications and users (based on hashing application name + submission date + user name — DHT (Pastry))
- Protocol of resource discovering; No dedicated nodes for the search of the next node to use → on the fly! Optimization: the machine that terminates the last starts the search.
- Rendez-vous concept (RDV); Objectives: localisation of a node without IP; task coordination; data recovery;
- Coordination protocol between machines participating in the application.
PastryGrid components

- Addressing scheme to identify applications and users (based on hashing application name + submission date + user name — DHT (Pastry))
- Protocol of resource discovering; No dedicated nodes for the search of the next node to use → on the fly! Optimization: the machine that terminates the last starts the search.
- Rendez-vous concept (RDV); Objectives: localisation of a node without IP; task coordination; data recovery;
- coordination protocol between machines participating in the application.
RDV Concept

Coordinator

- Known at the beginning;
RDV Concept

Coordinator

- Known at the beginning;
- Central element on a dedicated place;
RDV Concept

Coordinator

- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
RDV Concept

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
RDV Concept

Coordinator

- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)
## RDV Concept

### Coordinator
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

### RDV
- Unknown;
**RDV Concept**

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

**RDV**
- Unknown;
- Variable;
**RDV Concept**

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

**RDV**
- Unknown;
- Variable;
- Failure: may still run;
RDV Concept

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

**RDV**
- Unknown;
- Variable;
- Failure: may still run;
- Distributed data management;
**RDV Concept**

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

**RDV**
- Unknown;
- Variable;
- Failure: may still run;
- Distributed data management;
- RDV for each application (limited overload)
RDV Concept

**Coordinator**
- Known at the beginning;
- Central element on a dedicated place;
- Failure: the system crashes;
- Centralized resource management;
- Management of all applications (overload)

**RDV**
- Unknown;
- Variable;
- Failure: may still run;
- Distributed data management;
- RDV for each application (limited overload)
How PastryGrid works

Hash (Application Name + User Name + Submission Date): Unique identifier

Initialization of RDV: The machine which is closest numerically to ApplicationId

Search for free machine and assignment of tasks T1, T2 and T3
How PastryGrid works

- Hash (Application Name + User Name + Submission Date): Unique identifier ApplicationId
How PastryGrid works

- Hash (Application Name + User Name + Submission Date): Unique identifier ApplicationId
- Initialization of RDV: The machine which is closest numerically to ApplicationId
How PastryGrid works

- Hash (Application Name + User Name + Submission Date): Unique identifier ApplicationId
- Initialization of RDV: The machine which is closest numerically to ApplicationId
- Search for free machine and assignment of tasks T1, T2 and T3
How PastryGrid works

- Hash (Application Name + User Name + Submission Date): Unique identifier \textit{ApplicationId}
- Initialization of RDV: The machine which is closest numerically to \textit{ApplicationId}
- Search for free machine and assignment of tasks T1, T2 and T3
How PastryGrid works

- Request and Data Recovery by M1, M2 and M3: DataRequest and YourData
How PastryGrid works

M1 assigns T4 to M4 that she had found
M3 ends T3 but does not seek a machine for T6
M2 seeks M5 and M6 and assigns T5 and T6
How PastryGrid works

- M1 assigns T4 to M4 that she had found
**How PastryGrid works**

- M1 assigns T4 to M4 that she had found
- M3 ends T3 but does not seek a machine for T6
M1 assigns T4 to M4 that she had found

M3 ends T3 but does not seek a machine for T6
M1 assigns T4 to M4 that she had found
M3 ends T3 but does not seek a machine for T6
How PastryGrid works

- M1 assigns T4 to M4 that she had found
- M3 ends T3 but does not seek a machine for T6
- M1 assigns T4 to M4 that she had found
- M3 ends T3 but does not seek a machine for T6
How PastryGrid works

- M1 assigns T4 to M4 that she had found
- M3 ends T3 but does not seek a machine for T6
- M2 seeks M5 and M6 and assigns T5 and T6
M1 assigns T4 to M4 that she had found
M3 ends T3 but does not seek a machine for T6
M2 seeks M5 and M6 and assigns T5 and T6
Fault Tolerance in PastryGrid

- Passive replication based on Past (maintaining of $k$ copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain $k$)
Fault Tolerance in PastryGrid

- Passive replication based on Past (maintaining of \( k \) copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain \( k \))
- If we adopt such approach \( \Rightarrow \) node explosion;
Passive replication based on Past (maintaining of \( k \) copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain \( k \))

If we adopt such approach \( \Rightarrow \) node explosion;

A new component has been added: FTC (Fault Tolerant Component) node

- Supervises tasks that are running;
Passive replication based on Past (maintaining of $k$ copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain $k$)

If we adopt such approach $\Rightarrow$ node explosion;

A new component has been added: FTC (Fault Tolerant Component) node

- Supervises tasks that are running;
- A FTC component for each application; It contacts the RDV to decide the tasks to supervise;
Fault Tolerance in PastryGrid

- Passive replication based on Past (maintaining of \( k \) copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain \( k \))
- If we adopt such approach \( \Rightarrow \) node explosion;
- A new component has been added: FTC (Fault Tolerant Component) node
  - Supervises tasks that are running;
  - A FTC component for each application; It contacts the RDV to decide the tasks to supervise;
  - \( k \) copies of the FTC and \( k \) copies of the RDV per application. In fact you have 3 types of nodes: computing nodes, FTC nodes and RDV nodes to manage;
 Fault Tolerance in PastryGrid

- Passive replication based on Past (maintaining of $k$ copies of the node states); update copies when a modification occurs on a source node; automatically creation of a copy (to maintain $k$)
- If we adopt such approach $\Rightarrow$ node explosion;
- A new component has been added: FTC (Fault Tolerant Component) node
  - Supervises tasks that are running;
  - A FTC component for each application; It contacts the RDV to decide the tasks to supervise;
  - $k$ copies of the FTC and $k$ copies of the RDV per application. In fact you have 3 types of nodes: computing nodes, FTC nodes and RDV nodes to manage;
M initializes the RDV and the FTC of the application
M assigns tasks T1, T2 to M1 and M2
PAST creates k (k = 2) replicas RDV1, RDV2 for RDV and FTC1, FTC2 for FTC
M initializes the RDV and the FTC of the application
M initializes the RDV and the FTC of the application
M assigns tasks T1, T2 to M1 and M2
M initializes the RDV and the FTC of the application
M assigns tasks T1, T2 to M1 and M2
PAST creates k (k = 2) replicas RDV1, RDV2 for RDV and FTC1, FTC2 for FTC
Fault Tolerance in PastryGrid

- M1 and M2 recover from RDV, the data for T1 and T2
- The RDV informed the FTC of running tasks (T1 and T2)
- The FTC supervises the execution of tasks T1 and T2 on M1 and M2
M1 and M2 recover from RDV, the data for T1 and T2
M1 and M2 recover from RDV, the data for T1 and T2
The RDV informed the FTC of running tasks (T1 and T2)
M1 and M2 recover from RDV, the data for T1 and T2
The RDV informed the FTC of running tasks (T1 and T2)
The FTC supervises the execution of tasks T1 and T2 on M1 and M2
M1 and M2 recover from RDV, the data for T1 and T2
The RDV informed the FTC of running tasks (T1 and T2)
The FTC supervises the execution of tasks T1 and T2 on M1 and M2
The FT part

- Intensive experiments have been conducted (each machine has a probability $P$ to fail for $X$ seconds): $P = 20\%, 40\%, 80\%$; 100 applications (2 to 128 tasks); on 200 nodes.
The FT part

- Intensive experiments have been conducted (each machine has a probability $P$ to fail for $X$ seconds): $P = 20\%, 40\%, 80\%$; 100 applications (2 to 128 tasks) on 200 nodes

- Main observations:
  - In all cases, PastryGrid terminates;
  - The recovery time depends on the node type;
  - The delay varies from 4:53s to 7:16:41s... but it works! The number of delayed applications varies from 44 to 98.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Prob. in %</th>
<th>Execution time (s)</th>
<th>#Failed nodes</th>
<th>#Delayed applications</th>
<th>#FTC nodes</th>
<th># RDV nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>20</td>
<td>2h, 13mn and 2secs</td>
<td>66</td>
<td>44</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>50</td>
<td>3h, 22mn and 27secs</td>
<td>198</td>
<td>58</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>80</td>
<td>9h, 24mn and 49secs</td>
<td>583</td>
<td>98</td>
<td>12</td>
<td>14</td>
</tr>
</tbody>
</table>
Conclusion and Perspectives

Conclusion

- PastryGrid: Fault-tolerant decentralized system for running distributed applications with precedence between tasks
Conclusion

- PastryGrid: Fault-tolerant decentralized system for running distributed applications with precedence between tasks
- Creation of a dynamic execution environment for each application
Conclusion

- PastryGrid: Fault-tolerant decentralized system for running distributed applications with precedence between tasks
- Creation of a dynamic execution environment for each application
- Decentralized collaboration between machines for application tasks management
Conclusion and Perspectives

Perspectives

- DG has proved to be relevant for resource sharing ⇒
  transpose this success story to the Cloud and PaaS universes ⇒
  offer a technical alternate to Google, Salesforce, Amazon
  big farm of servers
Conclusion and Perspectives

**Perspectives**

- DG has proved to be relevant for resource sharing ⇒ transpose this success story to the Cloud and PaaS universes ⇒ offer a technical alternate to Google, Salesforce, Amazon big farm of servers

- PastryGrid is based on emerging open source Cloud solution. From an economic point of view: if it is less expensive to host services locally and if it support a wide range of applications → more potential partners, then small/medium size companies will adopt PastryGrid;
Perspectives

- DG has proved to be relevant for resource sharing ⇒ transpose this success story to the Cloud and PaaS universes ⇒ offer a technical alternate to Google, Salesforce, Amazon big farm of servers
- PastryGrid is based on emerging open source Cloud solution. From an economic point of view: if it is less expensive to host services locally and if it support a wide range of applications → more potential partners, then small/medium size companies will adopt PastryGrid;
Fault-Tolerance for PastryGrid Middleware

Christophe Cérin\textsuperscript{1}, Heithem Abbes\textsuperscript{1,2}, Mohamed Jemni\textsuperscript{2}, Yazid Missaoui\textsuperscript{2}

\textsuperscript{1}LIPN, Université de Paris XIII, CNRS UMR 7030, France
\textsuperscript{2}UTIC, ESSTT, Université de Tunis, Tunisia

HPGC’10 - IPDPS