Orleans
Cloud Computing for Everyone

Hamid R. Bazoobandi
Vrije University of Amsterdam

March 16, 2012
Outline

1. Introduction

2. Orleans
   - Orleans overview
   - Grains
   - Promise
   - Activation

3. Performance measurement
   - Applications
   - Cluster
   - Throughput measurement

4. Conclusion
Clouds are interesting, but programming for clouds …?

Writing software for clouds is challenging because ...

- Cloud systems are inherently parallel and distributed,
- High availability is financially crucial,
- and many other historical problems like security, reliability, elasticity, and etc.
Clouds are interesting, but programming for clouds ...?

Writing software for clouds is challenging because ...

- Cloud systems are inherently parallel and distributed,
- High availability is financially crucial,
- and many other historical problems like security, reliability, elasticity, and etc.
Clouds are interesting, but programming for clouds ...?

Writing software for clouds is challenging because ...

- Cloud systems are inherently parallel and distributed,
- High availability is financially crucial,
- and many other historical problems like security, reliability, elasticity, and etc.
Clouds are interesting, but programming for clouds ...?

Writing software for clouds is challenging because ...

- Cloud systems are inherently parallel and distributed,
- High availability is financially crucial,
- and many other historical problems like security, reliability, elasticity, and etc.
Orleans in brief

Orleans = Programming Model + Distributed Runtime

Applications

Orleans

.NET + Azure
What are Grains?

- Grains are the basic programming units in Orleans.
- They don’t share memory or any other transient state with each other.
- They are single threaded.
- They process each request before handling the next one.
- They interact entirely through asynchronous message passing.
What are Grains?

- Grains are the basic programming units in Orleans.
- They don’t share memory or any other transient state with each other.
- They are single threaded.
- process each request before handling the next one.
- interact entirely through asynchronous message passing.
What are Grains?

- Grains are the basic programming units in Orleans.
- They don’t share memory or any other transient state with each other.
- They are single threaded.
  - process each request before handling the next one.
  - interact entirely through asynchronous message passing.
What are Grains?

- Grains are the basic programming units in Orleans.
- They don’t share memory or any other transient state with each other.
- They are single threaded.
- process each request before handling the next one.
- interact entirely through asynchronous message passing.
What are Grains?

- Grains are the basic programming units in Orleans.
- They don’t share memory or any other transient state with each other.
- They are single threaded.
- process each request before handling the next one.
- interact entirely through asynchronous message passing.
Grain Example

Shopping Cart Grain

<table>
<thead>
<tr>
<th>Behavior</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy(...)</td>
<td>Total Price: $1300</td>
</tr>
<tr>
<td>Checkout(...)</td>
<td>Products: 2</td>
</tr>
<tr>
<td></td>
<td>Customer: 1</td>
</tr>
</tbody>
</table>

Product X Grain

<table>
<thead>
<tr>
<th>Behavior</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumAvailable()</td>
<td>Name: &quot;Canon EOS T3i&quot;</td>
</tr>
<tr>
<td>Buy(...)</td>
<td>SKU: B0043W90Y</td>
</tr>
<tr>
<td></td>
<td>Quantity: 12345</td>
</tr>
<tr>
<td>AddReview(...)</td>
<td>Price: $800</td>
</tr>
</tbody>
</table>

Product Y Grain

<table>
<thead>
<tr>
<th>Behavior</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumAvailable()</td>
<td>Name: &quot;Sony TC412&quot;</td>
</tr>
<tr>
<td>Buy(...)</td>
<td>SKU: X032DJ12</td>
</tr>
<tr>
<td></td>
<td>Quantity: 315</td>
</tr>
<tr>
<td>AddReview(...)</td>
<td>Price: $500</td>
</tr>
</tbody>
</table>
What are Promises?

- Asynchrony primitives of Orleans
- Expectation of receipt of the result, in unspecified future.
- A delegate code associated to each promise, is executed when the promise is fulfilled.
What are Promises?

- Asynchrony primitives of Orleans
- Expectation of receipt of the result, in unspecified future.
- A delegate code associated to each promise, is executed when the promise is fulfilled.
What are Promises?

- Asynchrony primitives of Orleans
- Expectation of receipt of the result, in unspecified future.
- A delegate code associated to each promise, is executed when the promise is fulfilled.
Promise Example

Promise

Buy(...)

Reply

Closure can Execute

Vrije University of Amsterdam

Orleans 8
Promise Example

(1) AsyncCompletion p1 = grainA.MethodA();
(2) AsyncCompletion p2 = p1.ContinueWith(() => {
(3)     return grainB.MethodB();
(4) });
(5) p2.Wait();
What are activations?

*Grains* are logical programming abstractions and *Activations* are run-time execution units.

- Orleans run-time system, automatically creates multiple activations of a busy grain.
- Activations process independent requests for the grain, possibly across multiple servers.
What are activations?

Grains are logical programming abstractions and Activations are run-time execution units.

- Orleans run-time system, automatically creates multiple activations of a busy grain.
- Activations process independent requests for the grain, possibly across multiple servers.
What are activations?

**Grains** are logical programming abstractions and **Activations** are run-time execution units.

- Orleans run-time system, automatically creates multiple activations of a busy grain.
- Activations process independent requests for the grain, possibly across multiple servers.
Activation Example

State Reconciliation
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - Less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Applications

Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
Two applications are built on Orleans:

- **Chirper**
  - A Micro-Twitter.
  - Communication intensive.
  - less than 200 lines of C#.

- **Linear Algebra Library**
  - Very large Matrix Computations.
  - Communication, Computation, and IO intensive.
## Cluster Specifications

- **50 nodes**
- Each node with 2 AMD Quad-Core Opteron processor 2.10GHz
- Each core 32GB Ram
- A 64-bit Windows Server 2008 R2 as Operating System
Cluster Specifications

- 50 nodes
- Each node with 2 AMD Quad-Core Opteron processor 2.10GHz
- Each core 32GB Ram
- A 64-bit Windows Server 2008 R2 as Operating System
Cluster Specifications

- 50 nodes
- Each node with 2 AMD Quad-Core Opteron processor 2.10GHz
- Each core 32GB Ram
- A 64-bit Windows Server 2008 R2 as Operating System
Cluster Specifications

- 50 nodes
- Each node with 2 AMD Quad-Core Opteron processor 2.10GHz
- Each core 32GB Ram
- A 64-bit Windows Server 2008 R2 as Operating System
Chirper Scalability Measurement

![Graph showing throughput vs. number of servers](image)
Chirper Speedup Measurement

![Graph showing speedup measurement](image)

- **Throughput measurement**
- **Chirper Speedup Measurement**
- **Graph showing speedup and linear speedup compared to the number of activations.**
Throughput measurement

Linear Algebra Library Speedup On a single machine

![Graph showing Linear Algebra Library Speedup on a single machine. The x-axis represents the number of cores, ranging from 1 to 16, and the y-axis represents speedup, ranging from 2 to 16. The line indicates linear speedup, while the squares represent actual speedup measurements.](image-url)
Linear Algebra Library Measurement with Orleans

Throughput measurement

![Graph showing speedup and linear speedup with the number of servers]

- **Speedup**
- **Linear Speedup**

Number of servers

Speedup

12 4 8 12 16 20 24 28 32
Summery

- Orleans makes programmers split the whole task, into some small, independent, and automatically manageable units named **Grains**
- Orleans runtime system, persist, migrate, replicate, and reconcile grains without programmer intervention.
- Orleans makes it possible for **non-expert** developers to build scalable, elastic, reliable cloud services.
Orleans makes programmers split the whole task, into some small, independent, and automatically manageable units named **Grains**

Orleans runtime system, persist, migrate, replicate, and reconcile grains without programmer intervention.

Orleans makes it possible for **non-expert** developers to build scalable, elastic, reliable cloud services.
Orleans makes programmers split the whole task, into some small, independent, and automatically manageable units named **Grains**.

- Orleans runtime system, persist, migrate, replicate, and reconcile grains without programmer intervention.
- Orleans makes it possible for **non-expert** developers to build scalable, elastic, reliable cloud services.
The moral of the story

Let’s make cloud computing for everybody!