Parallel and Distributed Simulation: from Many Cores to the Public Cloud

Gabriele D’Angelo
<gda@cs.unibo.it>
http://www.cs.unibo.it/gdangelo/

Istanbul, Turkey

International Conference on High Performance Computing and Simulation (HPCS), 2011
Tutorial material and extra information

- These slides, the tutorial paper and some extra information can be found in my homepage:
  - http://www.cs.unibo.it/gdangelo
  - “Gabriele D'Angelo” → Google
Tutorial outline

- A little **background**
- **Parallel And Distributed Simulation** (**PADS**)
- New **challenges** of today and tomorrow
- **Functionality** and **limitations** of current PADS approaches
- In the search of **adaptivity**: the ARTÌS/GAIA approach
- **Conclusions**
Tutorial outline

- A little background
  - simulation and its motivations
  - simulation paradigms
  - Discrete Event Simulation (DES)
  - DES: a simple example
  - implementation of DES
  - DES on a single CPU: sequential simulation
  - going parallel: Parallel Discrete Event Simulation (PDES)
Starting from scratch: **simulation**

- “A computer simulation is a computation that models the behavior of some real or imagined system over time”
  (R.M. Fujimoto)

- **Motivations:**
  - performance evaluation
  - study of new solutions
  - creation of virtual worlds such as online games and digital virtual environments
  - …
Simulation: more motivations

- The system that needs to be evaluated can not be built (e.g. for cost reasons)

- **Testing** on an existing system can be very dangerous

- Some **stress testing** is actually impossible to perform

- Often many different solutions have to be **investigated** in order to choose the best one

- It can be used to support the **decision making** (e.g. real-time what-if analysis)
Simulation paradigms

- There is a strong demand for more and more complex systems

- A huge number of simulation tools following different paradigms

- A lot of issues on the performance of such software tools

- In the years, many different simulation paradigms have been proposed, each one with specific benefits and drawbacks

- There is not the "correct way" of doing simulations, there are many different ways. It is really a case-by-case evaluation
**Discrete Event Simulation (DES)**

- The **state** of the simulated system is represented through a set of variables.

- The key concept is the **"event"**.

- An event is a **change in the system state** and it **occurs at an instant in time**.

- Therefore, the evolution of a modeled system is given by a **chronological sequence of events**.

- All is done through the **creation, delivery and computation** of events.

- The computation of an event can **modify some part of the state** and lead to the creation of new events.
DES: a simple example

1) A set of mobile wireless hosts
DES: a simple example

2) At time $t$ the red node starts transmitting
DES: a simple example

2) At time $t$ the red node starts transmitting
DES: a simple example

2) At time $t$ the red node starts transmitting
2) \textit{At time $t$ the red node starts transmitting}
DES: a simple example

3) *At time* $t + \alpha$ *the green node starts receiving*
4) **At time** $t+\beta$ **the dark violet node starts receiving**
DES: a simple example

5) At time $t + \gamma$ the red node stops transmitting
Implementation of DES

Data structures:

- a set of **state variables** *(to describe the modeled system)*
- an **event list** *(pending events that will be processed in future)*
- a **global clock** *(the current simulation time)*

Simulator:

- the simulator is mostly made by a set of “**handlers**”, each one managing a different event type

Notes:

- events are not produced in (simulated) time order but **have to be executed in non-decreasing time order**
- in fact, the **event list** is a **priority queue**
- the list based implementation is very inefficient
- heap-based solutions are widely used
DES on a single CPU: sequential simulation

- All such tasks are accomplished by a **single execution unit** (that is a CPU and some RAM)

- **PROS**: it is a **very simple approach**

- **CONS**: there are a few **significant limitations**
  - the **time** required to complete the simulation run
    - *how fast is a single CPU?*
    - *in some cases results have to be in real time or even faster!*
  - if the model is quite large and detailed the RAM is not sufficient: it is **not possible to model some systems**

- **This approach does not scale!**
Parallel Discrete Event Simulation (PDES)

- **Multiple interconnected** execution units (CPUs or hosts)
- Each unit manages a part of the simulation model
- Very large and complex models can be represented using the resources **aggregated** from many execution units
- Each execution unit has to manage a local event list
- **Locally generated events** may have to be delivered to remote execution units
- All of this needs to be carefully **synchronized**
- “Concurrent events” can be executed in parallel, this can lead to a significant **speedup of the execution**
Parallel Discrete Event Simulation (PDES)

- Multiple interconnected execution units (CPUs or hosts)
- Each unit manages a part of the simulation model
- Very large and complex models can be represented using the resources aggregated from many execution units
- Each execution unit has to manage a local event list
- Locally generated events may have to be delivered to remote execution units
- All of this needs to be carefully synchronized
- “Concurrent events” can be executed in parallel, this can lead to a significant speedup of the execution

It means that the model has to be partitioned and each part allocated on a different CPU. Is that easy?
Tutorial outline

- A little background
- Parallel And Distributed Simulation (PADS)
  - what is a PADS?
  - parallel, distributed or... mixed?
  - partitioning
  - synchronization
  - data distribution
  - in depth: synchronization approaches
  - software tools
Parallel And Distributed Simulation (PADS)

- “Any simulation in which more than one processor is employed” (K.S. Perumalla)
- This is a very simple and general definition, there are many different “flavors” of PADS
- A lot of good reasons for going PADS:
  - scalability
  - performance (obtaining the results faster)
  - to model larger and more complex scenarios
  - interoperability, to integrate commercial off-the-shelf simulators
  - composability of different simulation models
  - to integrate simulators that are geographically distributed
  - Intellectual Property (IP) protection
  - ...
There is **no global state**: this is the key aspect of **PADS**

A **PADS** is the **interconnection** of a set of **model components**, usually called **Logical Processes (LPs)**

Each **LP** is responsible to manage the evolution of only a **part of the simulation**

Each **LP** has to interact with other **LPs** for **synchronization** and **data distribution**

In practice, each **LP** is usually executed by a **processor** (or a **core** in modern multi-core architectures)

The **communication** among **LP** (and the **type of network** that interconnect the processors) is of main importance

It strongly affects simulator characteristics and **performance**
Parallel, distributed or... mixed?

- What is **parallel** simulation and what **distributed**?

- The difference is quite elusive but with some importance

- We choose a very simple definition from the many that are available

- **Parallel**: the processors have access to some **shared memory** or a **tightly coupled** interconnection network

- **Distributed**: **loosely coupled** architectures (e.g. **distributed memory**)

- Real world execution architectures are more **heterogeneous**

- **For example**: a) **LAN-based clusters of multi-CPU (and multi-core) hosts**, b) **a mix of local and remote resources (e.g. Cloud Computing)**
On the (lack of) **global state** and its consequences

- In a sequential simulation there is a global state that represents the **simulated system at a given time**
- In a **PADS**, such a global state is **missing**
- There are some very interesting consequences
- The model has to be **partitioned** in **components** (the LPs)
- In a parallel/distributed architecture **synchronization** mechanisms have to be implemented
- **Data is produced locally** (within the LP) but can be of interest to other parts of the simulator (other LPs): **data distribution** mechanisms
- All these are **main problems of PADS**: we need to introduce them a little more in detail
Partitioning: creating and allocating parts

- Each LP is responsible for the management of a part of the simulated model

- In some cases the partitioning follows the structure and the semantics of the simulated system

- In other cases is much harder, for example if the system is monolithic and hard to split in parts

- Many different aspects have to be considered in the partitioning process

- For example:
  - minimization of network communication
  - load balancing of both computation and communication in the execution architecture
Synchronization: on the correct order of events

- Some kind of **network** interconnects the **LPs** running the simulation

- Each **LP** is executed by a different **CPU** (or **core**), **possibly at a different speed**

- The **network** can **introduce delays** but we assume that the communication is reliable (e.g. TCP-based communications)

- The results of a **PADS** are **correct** only if its outcome is **identical** to the one obtained from the corresponding **sequential** simulation

- **Synchronization mechanisms** are used to **coordinate** the **LPs**: different approaches are **possible**

- This task usually has a **very relevant cost**
Data distribution: on the dissemination of information

- Each component of the simulator will produce state updates that are possibly relevant for other components.
- The distribution of such updates in the execution architecture is called data distribution.
- For overhead reasons broadcast cannot be used.
- The goal is to match data production and consuming based on interest criteria.
- Only the necessary data has to be delivered to the interested components.
- There are both communication and computation aspects to consider.
- Data distribution also has to be properly synchronized.
In-depth: **synchronization, causal ordering**

- Implementing a **PDES** in a **PADS** architecture requires that all generated events have to be **timestamped** and delivered following a **message-passing** approach.

- Two events are said to be in **causal order** if one of them can have some consequences on the other.

- The execution of events in **non causal order** leads to **causality errors**.

- In a sequential simulation it is easy avoid causality errors given that there is a single ordered pending event list.

- But in a **PADS** this is **much harder**!

- In this case the goal is to:
  - execute events **in parallel**, as much as possible
  - do not introduce **causality errors**
In-depth: synchronization, approaches

- The most studied aspect in PADS because of its importance

- Many different approaches and variants have been proposed, with some simplification **three main methods:**

  - **time-stepped**: the simulated time is divided in **fixed-size timesteps**

  - **conservative**: causality errors are prevented, the simulator is built to avoid them

  - **optimistic**: the causality constraint can be violated and errors introduced. In case of causality violations the simulator will fix them

- In the following we will see more in deep each of them
In-depth: synchronization, time-stepped

- The **simulated-time** is divided in **fixed-size timesteps**
- Each **LP** can proceed to the **next timestep** only when all other **LPS** have **completed the current one**
- It is a **discretization** of time, that is clearly continuous
**In-depth: synchronization, time-stepped**

- The **simulated-time** is divided in **fixed-size timesteps**

- Each LP can proceed to the **next timestep** only when all other LPs have **completed the current one**

- It is a **discretization** of time, that is clearly continuous

- The **timestep size** can be chosen by the model developer but **strongly affects performances** (*smaller steps equals to more synchronization points*)

- The main **advantage** is its **simplicity**: simple to implement and quite easy to understand for the simulation developer

- **Drawbacks**: **unnatural paradigm** for some systems to model, in some cases the step needs to be very small (*e.g. in the simulation of media access control protocols*)
In-depth: **synchronization, conservative**

- The **goal** of this approach is to **prevent causality errors**

- **Before processing each event** (e.g. with timestamp $t$), the **LP** has to decide if the **event** is “safe” or “not”

- It is **safe** if, **in future**, there will be no **events with timestamp less than** $t$

- Remember that, in this case, there cannot be **causality errors** to be fixed, the simulator has to **avoid them a priori**

- If all **LPs** process event in **timestamp order** then the **PADS results** will be **correct**

- A **mechanism** to determine **if and when an event is safe** is needed
In-depth: synchronization, CMB

- The Chandy-Misra-Bryant (CMB) is a widely used algorithm for conservative synchronization.

- LPs can only process events that are “safe”.

- In many cases the LP will have to stop, waiting to get enough information to decide if an event is “safe” or not.

- The deadlock is avoided using NULL messages.

- A NULL message is an event with no semantic content.

- It is necessary only for spreading information on synchronization.

- Every LP, each time a new event is processed, has to send as many NULL messages as the number of LPs at which it is connected to.
In-depth: synchronization, optimistic

- The LPs are free to violate the causality constraint.
- They can process events in receiving order (vs. timestamp order).
- There is no a priori attempt to detect “safe” events and to avoid causality violations.
- In case of violation this will be detected and appropriate mechanisms will be used to go back to a prior state that was correct.
- The main mechanism is the roll back of internal state variables of the LP in which happened the violation.
- If the error has propagated to other LPs then also the roll back has to be propagated to all the affected LPs.
In-depth: \textit{synchronization, Time-warp}

- The Jefferson's \textbf{Time Warp} mechanisms implements optimistic synchronization
- Each LP process all events that it has received up to now
- An event is "late" if it has a timestamp that is smaller than the current clock value of the LP (that is the timestamp of the last processed event)
- The violation of \textit{local causality} is fixed with the roll-back of all the \textit{internal state variables} of the simulated model
- Likely the violation has propagated to other LPs
- The goal of "anti-messages" is to annihilate the corresponding unprocessed events in LPs pending event list or to cause a \textit{cascade of roll-backs} up to a \textit{globally correct state}
In-depth: **synchronization, what is best?**

- All these approaches have been deeply investigated and many *variants / tunings* have been proposed.

- **What is the best synchronization approach for PADS?**

- Very hard question, the **performance** of such methods heavily depends on many factors:
  - *simulation model*
  - *the execution environment*
  - *the specific scenario*

- **Forecasting** the **performance** of PADS is **very hard**, it depends on **too many factors**, some *static* and some *dynamic*, some *known* and many *unknown* in advance (*e.g.* *the runtime conditions of the execution architecture*)
**PADS: software tools**

- There are many software tools for the implementation of PADS

- Some of them are compliant with the **High Level Architecture (HLA) IEEE 1516 IEEE standard**: RTI NG Pro, Georgia Tech FDK, MÄK RTI, Pitch RTI, CERTI Free HLA, OpenSkies Cybernet, Chronos and the Portico Project

- Many others are more focuses on **performance** or other aspects such as **extensibility** or testing of **new features**. For example: _μsik, SPEEDES_ and _PRIME_

- In the next part of the tutorial we will discuss if current PADS technologies are ready for the **new challenges** of today and tomorrow
Tutorial outline

- A little background
- Parallel And Distributed Simulation (PADS)
- New challenges of today and tomorrow
  - what's next?
  - the many cores architectures
  - simulation as a service: simulation in the public cloud
New challenges: what's next?

- Evolution in computing technology is fast and often confusing.
- But it is possible to identify some **characteristics** and **trends**.
- Frequent **updates in hardware** but **software is slow in supporting them**.
- On the other hand, **software** is **limited** by **hardware characteristics**.
- For many years, 32 bits processors have limited the max amount of memory of sequential simulators.
- Now with 64 bits CPUs memory remains an issue only with huge simulations.
New challenges: some existing and new trends

- The so called “MHz race” in CPUs has slowed down
- **Multi-core CPUs** are now available at bargain prices
- Only few users have access to **High Performance Computing** facilities (*i.e. supercomputers and dedicated clusters*)
- Many are willing to use **Commercial Off-The-Shelf (COTS)** hardware that is also **shared with other tasks** (*e.g. desktop PCs or underloaded servers*)
- **Outsourcing** the **execution of simulations** is the next big step in this direction
New challenges: cloud computing

- **Cloud computing** is a model for providing on-demand network access to a shared pool of computing resources.

- Such resources can be provisioned and released quickly and with minimal management effort.

- For many reasons cloud computing is becoming mainstream.

- Implements the “pay-as-you-go” approach: virtual computing environments in which you pay only for capacity that you actually use.

- The resources are obtained from a shared pool and provided by commercial service providers.
Available simulators are unable to cope with such changes in the execution environment.

- Often they do not exploit all the available resources.
- That means that they are too slow in obtaining the results.
- The effect is that users are more and more encouraged to oversimplify the simulation models.
- That's a very risky move...
- In the next slides we'll discuss more in detail a couple of these challenges.
New challenges: many cores

- **Entry level CPUs** provide 2 or 4 cores but processors with **16** cores are already available on the market.

- **CPUs** with **100** cores are announced for the end of this year.

- This is a **big change in the execution architecture** and will **not be transparent** to simulation users.

- **Sequential simulators** are, for the most part, **unable to exploit more than one core**.

- This means that **PADS** techniques will be **necessary** even to run simulations on a **desktop PC**.
**New challenges: many cores**

- Even if assuming that all **cores** are **homogeneous** *(and that is not always true)*, the simulation model has to be **partitioned** in **more and more LPs**.

- The **partitioning** is a **complex task** and increasing the number of cores it becomes **harder and harder**.

- The **load of each core** has to be **balanced** and the **communication** among cores has to be **minimized**.

- Who is in charge of the partitioning has to predict **a priori**:
  - **the behavior of the simulated model**
  - **the load of the execution architecture**
New challenges: many cores

- All **static approaches** are **suboptimal**: the **runtime conditions** are **variable**

- Who is in charge of **partitioning**?
  - *currently, the software is unsuitable to perform this task*
  - *it is still in charge of the simulator user!*

- It is clear that **this approach does not scale**!

- Most simulation users are not willing to become experts of **PADS** or computing architectures

- Their goal is to **obtain results as fast as possible** and **with the least effort**

- It is clear that it should be a **software task**!
New challenges: the public cloud

- Everything is going “on the cloud”. Why simulation is not?

- Please do not confuse the private cloud and the public cloud infrastructures, they are very different!

- The big goal is to follow the “everything as a service” paradigm and to rent the resources for running simulations.

- On the market there are many providers of cloud services (e.g. Google, Amazon, Microsoft…)

- You pay only for the rented resources and you can increase or decrease them dynamically.

- This is great for small or medium size firms: no more investments in hardware!
New challenges: the public cloud

- A public cloud environment can be very dynamic, variable and heterogeneous.

- For example, the virtual instances providing the services can be located in different data centers, with different Service Level Agreements and from different providers.

- Under the PADS viewpoint, also in this case it is a matter of partitioning.

- This is an even more complex version of the partitioning problem.

- But we have already seen that current software tools are unable to cope with this problem.
New challenges: the public cloud on steroids

- Let's go on with our vision of “simulation-as-a-service”

- The price of cloud computing services is highly dependent on aspects such as reliability and guaranteed performance

- It is a pricing model based on the assumption that all customers have the same requirements

- PADS tools could (automatically) rent very inexpensive (and low reliability) cloud services

- The middleware running the PADS will be in charge of coping with faults

- This can be “easily” done adding some degree of replication

- This is a further extension of the partitioning problem
Tutorial outline

- A little background
- Parallel And Distributed Simulation (PADS)
- New challenges of today and tomorrow
- **Functionality** and **limitations** of current PADS approaches
  - *is PADS ready for primetime?*
  - *usability (lack of)*
  - *cost assessments: the need for new metrics*
  - *in search of performance*
Is **PADS** ready for **primetime**?

- The **complexity** of studied systems is **increasing**
- Many would expect a **broad application** of **PADS** techniques
- **Is that happening? No, it is not!**
- Many users are **unwilling** to dismiss the “old” (**sequential**) tools and switch to more modern ones
- Even if there is a **strong demand** for **scalability** and **faster execution speed**
- **What is missing?**
- There is obviously a problem that should be more clearly defined and investigated
Historical perspective on PADS

High Performance Computing Community

Chandy/Misra/Bryant
algorithm
Time Warp algorithm
second generation algorithms
making it fast and
easy to use

1975 1980 1985 1990 1995 2000

SIMulator NETworking (SIMNET)
(1983-1990)
High Level Architecture
(1996 - today)

Distributed Interactive Simulation (DIS)
Aggregate Level Simulation Protocol (ALSP)
(1990 - 1997ish)

Defense Community
Dungeons and Dragons
Board Games
Adventure
(Xerox PARC)

Internet & Gaming Community
Multi-User Dungeon (MUD)
Games
Multi-User Video Games

Richard M. Fujimoto, tutorial, 2000
Historical perspective on PADS

High Performance Computing Community

Chandy/Misra/Bryant algorithm
Time Warp algorithm
early experimental data
second generation algorithms
making it fast and easy to use

1975 1980 1985 1995 2000

High Level Architecture (1996 - today)

SIMulator NETworking (SIMNET) (1983-1990)
Distributed Interactive Simulation (DIS)
Aggregate Level Simulation Protocol (ALSP) (1990 - 1997ish)

Defense Community

Dungeons and Dragons
Board Games
Adventure (Xerox PARC)

Internet & Gaming Community

Multi-User Dungeon (MUD) Games
Multi-User Video Games

Richard M. Fujimoto, tutorial, 2000
PADS: what happened in the last two decades?

- Two **main research goals**:
  - *make it fast*
  - *make it easy to use*

- A lot of work in synchronization and data dissemination management has been done
  
  → **in some conditions** PADS is very fast

  ... *properly partitioned model, appropriate synchronization algorithm, homogeneous execution architecture …*

- What about **usability**?

  **PADS does not work straight out of the box**

- The level of **knowledge** modelers are required is still too high, some aspects are **hard to manage** and **understand**
**PADS: what is the better choice?**

- In some cases **PADS** techniques are necessary, in other are not *(e.g. when **PADS** is slower than sequential)*

- Each time there is something to simulate, the main question should be: **what is the better choice?**
  
  ... **sequential, parallel, distributed, conservative, optimistic** ...

- The execution environments are becoming much more heterogeneous:
  - *multi-core CPUs* (... *many core CPUs*)
  - *clusters, private and public clouds*

- Up to now, the whole problem is left to the simulation model developer

- **It feels like PADS tools are for initiates**
Why is it so difficult to decide what is best?

- Even the **sequential** or **PADS** choice is **hard to make**!

- It depends on **dynamic parameters** in **all the logical layers** of the architecture (**e.g., hardware, software**)

- All those parameters need a **case-by-case evaluation**

- Furthermore, they can change within the simulation runs:
  - **semantic** of the simulation model
  - **variable background load** in the execution architecture

- In many cases all such **aspects** and **parameters** are **not known a priori**
PADS: usability (lack of)

- The **user** of simulation tools should be able to **focus on modeling** and **analysis** of results.

- Very often the modeler uses a **different tool** if he wants to build a **sequential** simulation or a **PADS** one.

- What happens if after implementing a sequential one he discovers that it is **too slow**?

- Now many **key aspects** are left to the simulation developer and that's **clearly wrong**!

- In 2000 it has been approved the **IEEE 1516 standard** for distributed simulation called **High Level Architecture (HLA)**.

- **HLA** supports optimistic synchronization but a **significant part** of the support mechanisms is **left to the simulation developer**.
PADS: cost assessments, the need for new metrics

- The **amount of time** needed for completing a simulation run is called **Wall-Clock-Time (WCT)**

- The **WCT** has always been the **main metric** to evaluate the **efficiency of simulators**

- This can be right in classic execution architectures but it is **not** when the resources are obtained following the “**pay for what you use**” scheme (e.g. public cloud)

- A **more complex evaluation** has to be done:
  - *how much time the user can **wait for the results**?*
  - *how much he **wants to pay** for running the simulation?*

- Are the current **PADS** algorithms and mechanisms suitable for this new evaluation metric?
PADS: cost assessments, the need for new metrics

- As seen previously the Chandy-Misra-Bryant (CMB) algorithm is often used for implementing conservative synchronization.

- To avoid deadlocks it introduces artificial events (i.e. without any semantic content).

- The number of such events can be very high.

- Despite of many optimizations the amount of extra communications is often prohibitive.

- In a distributed execution environment such as the cloud in which the available bandwidth is limited (and costly) this approach is not very promising.

- What about optimistic synchronization? Is it better?
**PADS: cost assessments, the need for new metrics**

- **Computation** is much **faster** (and **cheaper**) than **communication**

- This **assumption** is at the **basis** of **optimistic** synchronization

- This means that, in a **PADS**, the **CPU** will be often **idle waiting** form some data from the network

- Therefore it is better to proceed with the computation and roll-back if something has gone wrong *(e.g. a causal violation)*

- Also this approach is **not well suited** for the “**pay for what you use**” model

- In optimistic simulations a large part of the computation can be thrown away due to roll-backs
PADS: in search of performance

- Let's assume that **costs are not a problem** and that the goal is to **obtain the results as fast as possible**

- Continuing to focus on **synchronization**, the traditional algorithms are **fast** when run in a **public cloud**?

- What level of **performance** we can expect?

- The answer is quite simple: using the traditional approaches the **obtained results can be poor**

- What is the **problem**?
Both in **timestepped** and **conservative** approach a **slow LP** would become the **bottleneck** of the whole simulation.

The real problem is the **lack of adaptivity**: the **static partitioning** of the simulated model has big drawbacks.

With **optimistic** it is **even worse**.

E.g. **Jefferson's timewarp** is well-known to have good performance if all **LPs** have the **same execution speed**.

This assumption is very **unrealistic** in public environments.
**Tutorial outline**

- A little **background**
- **Parallel And Distributed Simulation (PADS)**
- New **challenges** of today and tomorrow
- **Functionality** and **limitations** of current PADS approaches
- In the search of **adaptivity**: the ARTÌS/GAIA approach
  - *model decomposition*
  - *dynamic partitioning*
  - *finding and removing bottlenecks*
  - **ARTÌS and GAIA+**
  - **Reliable GAIA+**
**How: on the adaptive approaches**

- **Warning**: the “silver bullet” does not exist, even in simulation

- In our vision, all starts with the **partitioning problem**: decomposing the simulation model into a number of components and properly allocating them among the execution units

- **Constraints**: the computation load has to be kept balanced while the communication overhead has to be minimized

- Given that the runtime conditions are largely **unpredictable** and the environment is **dynamic** and very **heterogeneous**, all static approaches are not adequate
Migration-based **adaptive partitioning**

- The simulated model is divided into very small parts (called **Simulated Entities, SEs**)
- Each **SE** is a tiny piece of the simulated model and interacts with other **SEs** to implement the model behavior
- It is some sort of **Multi Agent System (MAS)**
- Each **node** (called **Logical Process, LP**) in the execution architecture is the container of a dynamic set of **SEs**
- The **SEs** are **not statically allocated** on a specific **LP**, they can be **migrated** to:
  - *reduce the communication overhead*
  - *enhance the load balancing*
Adaptive clustering: migration of entities

- In a parallel/distributed simulation the communication overhead is usually quite high.
- Each SE will have (possibly) different interaction patterns.
- In the simulation, it is possible to find “interaction sets” composed of SEs interacting with high frequency.
- The main strategy is to cluster the SEs interacting with high frequency within the same LP.
- All of this can be done analyzing the communication pattern of each SE and migrating some of them.
- The load balancing has to be considered!
Adaptive clustering: migration of entities

In dashed lines, the interactions of SE₃ with other simulated entities
Adaptive clustering: migration of entities

In solid lines, the migrations that should be done to enhance the partitioning
The **ARTÌS/GAIA** simulator

- **simulation model**
  - model behavior:
    - state variables
    - event handlers
  - high level communication APIs
  - migration support
  - clustering heuristics

- **GAIA**

- **ARTÌS**
  - runtime services:
    - synchronization
    - communication
    - simulation management

- **operating system**
**ARTÌS and GAIA, some details**

- Multi-year effort for building an **efficient simulation middleware**: Advanced RTI System (**ARTÌS**)

- Used as a testbed for many research works

- The **Generic Adaptive Interaction Architecture** (**GAIA**) framework implements the **adaptive features**:
  - *adaptive clustering* for overhead reduction
  - *dynamic load balancing* of communication and computation
  - *support for* heterogeneous execution platforms and **shared computing resources**
  - **Reliable-GAIA**: support for *fault-tolerance* (work in progress)

For details and software download: http://pads.cs.unibo.it
ARTÌS and GAIA, some details

- Multi-year effort for building an efficient simulation middleware:
  - Advanced RTI System (ARTÌS)

- Used as a testbed for many research works

- The Generic Adaptive Interaction Architecture (GAIA) framework implements the adaptive features:
  - adaptive clustering for overhead reduction
  - dynamic load balancing of communication and computation
  - support for heterogeneous execution platforms and shared computing resources
  - Reliable-GAIA: support for fault-tolerance (work in progress)

It is free for education and research purpose!

Many parts are provided with source code (e.g. all the simulation models)

Our goal is to Open Source as soon as possible the whole software stack

For details and software download: http://pads.cs.unibo.it

It is free for education and research purpose!

Many parts are provided with source code (e.g. all the simulation models)

Our goal is to Open Source as soon as possible the whole software stack

For details and software download: http://pads.cs.unibo.it
Tutorial **outline**

- A little **background**

- **Parallel And Distributed Simulation (PADS)**

- New **challenges** of today and tomorrow

- **Functionality** and **limitations** of current PADS approaches

- In the search of **adaptivity**: the ARTÌS/GAIA approach

- **Conclusions**
Conclusions

- There is a strong demand for **scalable simulators**

- **Parallel And Distributed Simulation (PADS)** is the natural choice for enhancing the performance of simulations

- The diffusion of **multi-core CPUs** and **cloud computing** will deeply change the execution environment of simulations

- Current **PADS** technologies are unable to cope with such changes

- The simulation modeler is in charge of too many details

- We really need **smarter software**: **adaptive PADS**
Further information

Gabriele D'Angelo

Parallel and Distributed Simulation: from Many Cores to the Public Cloud

Proceedings of the International Conference on High Performance Computing and Simulation (HPCS 2011). Istanbul, Turkey, July 2011

An extended version of this tutorial paper is freely available at the following link:

- http://arxiv.org/abs/1105.2301

The ARTÌS middleware and the GAIA framework can be downloaded from:

- http://pads.cs.unibo.it

Gabriele D'Angelo

- E-mail: <g.dangelo@unibo.it>
- http://www.cs.unibo.it/gdangelo/
Parallel and Distributed Simulation: from Many Cores to the Public Cloud

Gabriele D’Angelo  
<gda@cs.unibo.it>  
http://www.cs.unibo.it/gdangelo/

Istanbul, Turkey

International Conference on High Performance Computing and Simulation (HPCS), 2011