Model-Driven Development of Performance Sensitive Cloud Native Streaming Applications

José Ángel Bañares
Engineering & Architecture School - Zaragoza University, Spain
banares@unizar.es
- Tout le monde savait que c'était impossible. Puis vint un imbécile qui ne le savait pas et qui l'a fait.
- Everyone knew it was impossible. Then along came a fool who did not know this, and who did it.
- Todo el mundo sabía que era imposible. Entonces vino un imbecil que no lo sabía, y lo hizo.

M. Pagnol
Transition to a Smart Grid

- Smart meters
  - Bi-directional, real-time communication between utility & consumer

Yogesh Simmhan (Indian Institute of Science)
A first approach to Managing Smart Grid Information in the Cloud

Cloud Domain
- Cloud Providers
- Service Providers
- End Users

Smart Grid Domain
- Bulk Generation
- Transmission
- Distribution
- Customers
- Markets
- Operations
- Service Providers

End Users

Xi Fang New Mexico State Univ., Guoliang Xe, Dejun Yang (Arizona State Univ.)
“Managing Smart Grid Information in the Cloud: Opportunities, Model and Applications” IEEE Network 2012
Complexity of Continuous Data Stream Real Time Processing Applications

- Preserving QoS applications on Shared Distributed infrastructures
- Expressing **scalable** solutions on heterogeneous infrastructures
- Processing huge volume of **data online**
- Parallel processing of data
- Scaling data storage, network and computing resources

**Data transmission/injection**

Regulate the data injection rate into the processing units

**Resource sharing**

- **HPC: Supercomputers**
- **HTC: Grid, Clusters**
- **Cloud:** elastic provisioning

Buffers

Sensor Net

Camera Nets
Cloud Native Applications

- Guidelines for cloud-native applications (CNA):
  1. **Functional View**: Decomposition of functionality in chunks of distributed functionality

---

**Layers & Tiers**

- Client: browser/Web
- Architecture N-tiers
- Web Server
- Presentation layer
- Filtro XML/HTML
- Application logic layer
- Middleware
- Resource management Layer

**Business Processes**

**Scientific Workflows**

**Pipelined Execution Semantics**

1. Best Effort
2. Blocking
3. Buffered
4. Superscalar
5. Streaming

---

**Layers**


**Processes**


**Pipelines: Filter & Pipes**

Cloud Native Applications (CNA)

2. Design Space/Operational Model:
   - Analysis of the application workloads
   - How application handle state

Types of cloud workload:
- Static Workload
- Periodic Workload
- Once-in-a-lifetime Workload
- Unpredictable Workload
- Continuously Changing Workload

Cloud Applications Architecture Patterns:
- Loose Coupling
- Distributed Application
- Stateful Component
- Stateless Component
- User Interface Component
- Processing Component


http://www.cloudcomputingpatterns.org
Cloud Native Applications

3. **Resilience & Elasticity** *(cloud application management patterns)*
   - Analysis of the **application workloads**
   - How application handle state
Different approaches

The complexity of developing cost-effective and performance sensitive cloud native streaming applications has been addressed from different approaches:

1. **Cloud based frameworks** that lift the level of abstraction **reducing complexity** and **hiding resource management**
Is it possible to reason about performance by hiding resources?

- In a cloud computing environment, dynamic resource allocation and reallocation are keys for accommodating unpredictable demands.

- A Resource Allocation System (RAS) is a discrete event system in which a finite set of concurrent processes shares in a competitive way a finite set of resources.

Let us consider a manufacturing cell ...

... where two types of parts must be produced

- Ce qui ne peut être, il ne peut pas être, et il est impossible
- What can not be, it can not be, and it is also impossible
- Lo que no puede ser, no puede ser, y además es imposible

-- Charles Maurice de Talleyrand

Image Courtesy José Manuel Colom
Different approaches

2. Collecting high-quality solutions, which are presented as patterns to recurring problems in parallel applications and cloud platforms.
Different approaches

3. Developing ad-hoc performance models to predict the behaviour of particular patterns on specific platforms

However,

- Ad-hoc models for specific patterns or platforms are not reusable
- Generic tools based on formal models are likely to encounter problems in the context of cloud applications: Involved complexity makes the models either mathematically or computationally intractable.
Bridging the gap between formal methods and real problems

- **A Petri net based Methodology** for the development of Continuous Data Stream Processing Applications
  - Descriptive power for the concurrency
  - Model driven, i.e., use of the models to reason/infer properties
  - Executable specification language across all software architecture levels
- **Model Analysis**
  - The model can be used for detecting problems and performance and economical boundaries prediction
  - When a property is not satisfied, can be used for
    - Detecting the causes of the problem
    - Modify the model.
Synoptical View of the Methodology

**Functional Level**
- Control & data flow
- Resource sharing
- Process: Identify functional Entities
  - Identify Dataflow & dependencies
  - Resource sharing

**Qualitative Analysis**
- Verification of properties
- Finding Concurrency
- Structural Analysis
- Model Checking
- Simulation

**Operational Level**
- Resource Profiling
- Functional Patterns
- Map to resources
- Performance variability of resources
- Identify supporting architecture
- Identify Functional Patterns
- Mapping Patterns to resources
- Integration functional & operational
- Computing Performance bounds
- Simulation
- Markov chain

**Quantitative Analysis**
- Stochastics PNs
- Simulation & bounds
- Reward functions
- Translation
- Model partition
- Deployment and Orchestration
- Markov chain analysis

**Implementation**
- Translation & model partition
- Node Orchestration & deployment
- Performance Monitoring
- QoS Analysis
- Autonomic PaaS

**Monitoring**
- Monitoring
- Support for Autonomic PaaS

**Deliverables**
- Functional Requirements \( \Phi \)
- Functional model
- Non-Funct. Requirements
- Correct Funct. model
- Operational model
- Target platform $\$
- Reward functions $\$
- Integrated model
- Flat model of processes
- Orchestration & Deployment Specification
- Application Infrastructure Mapping Catalogue
Methodology

- Identify different views of the model
  - **Functional Petri net model**
    - Derived from a specific algorithm that processes a number of given data streams.
    - Composition of computational task and the data dependencies among them.
    - Minimal number of constraints coming from the final execution environment can be taken into account.
  - **Operational Petri net model**
    - Explicitly consideration of characteristic of the final execution environment
Specification Language

- Specification language to support the methodology
  - Different views at different abstraction levels
    - Behavioural specification of concurrent processes
    - Data operations/transformations over data flow
    - Structural description of components
  - Component based development
    - Reuse of components
    - Prediction of system properties using the properties of components
Basic Building Blocks

**CP**: a computational process
- Sequence of operations
- **CT**: Computational Threads are instances of CP that are executed concurrently
- Idle place represents threads in the inactive state
- Incorporation of resources (processor, buffer, server, etc.) with a given capacity.
  - Resources are conservative, i.e., there is no resource leakage.
Basic Building Blocks

**CP**: a computational process
- Sequence of operations
- **CT**: Computational Threads are instances of CP that are executed concurrently
- Idle place represents threads in the inactive state
- Incorporation of resources (processor, buffer, server, etc.) with a given capacity.
  - Resources are conservative, i.e., there is no resource leakage.
Data transmission Processes (timed model)

**DTP (Data transmission processes)**
- Transmissions between CP
  - Data streams sent by physical or virtual network
- **Incorporation of resources** (FIFO queue implemented in memory, communication channel in a network, etc.) with a given capacity.
  - **Resources are conservative**, i.e., there is no resource leakage.
Operational Model
Wavefront case study

Wavefront processing

\[ Z^{(i)} = Y^{(i)} + A X^{(i)} \]

Cell Operation

\[ y \rightarrow a \rightarrow y + ax \]
Functional Model Specification

Wavefront cell component definition

{Inj_wavefront is a Functional_wavefront

Interface
InjY1, InjY2, InjY3, InjX1, InjX2, InjX3, OZ1, OZ2, OZ3

Behaviour
Components: Inj1_Y, Inj2_Y, Inj3_Y,
            Inj1_X, Inj2_X, Inj3_X: Inj

Links
Inj1_Y.End_inj+C11.Begin_op+ Inj1_X.End_inj,
Inj2_Y.End_inj+C21.Begin_op,
Inj3_Y.End_inj+C31.Begin_op,
Inj2_X.End_inj+C12.Begin_op,
Inj3_X.End_inj+C13.Begin_op

Rename
Inj1_Y:Inj1_Y.Begin_inj, Inj2_Y:Inj2_Y.Begin_inj,
Inj3_Y:Inj3_Y.Begin_inj, InjX1:Inj1_X.Begin_inj,
InjX2:Inj2_X.Begin_inj, InjX3:Inj3_X.Begin_inj,
Untimed functional model
Synoptical View of the Methodology

**Functional Level**
- Control & data flow
- Resource sharing

- Process
- Identify functional Entities
- Identify Dataflow & dependencies
- Resource sharing

**Qualitative Analysis**
- Verification of properties
- Finding Concurrency

- Structural Analysis
- Model Checking
- Simulation

**Operational Level**
- Resource Profiling
- Functional Patterns
- Map to resources

- Performanc variability of resources
- Identify supporting architecture
- Identify Functional Patterns
- Mapping Patterns to resources

**Quantitative Analysis**
- Stochastics PNs
- Simulation & bounds
- Reward functions

- Integration functional & operational
- Computing Performanc bounds
- Simulation
- Markov chain

**Implementation**
- Translation & model partition
- Node Orchestration & deployment

- Translation
- Model partition
- Deployment and Orchestration

**Monitoring**
- Monitoring
- Support for Autonomic PaaS

**Deliverables**
- Functional Requirements
  - Functional model
- Non-Functional Requirements
  - Correct Funkt. model
- Operational model
  - Target platform
- Reward functions
- Integrated model
- Flat model of processes
- Orchestration & Deployment Specification
- Application Infrastructure Mapping Catalogue
PN Analysis techniques

- **Qualitative analysis.** Desirable good properties.
  - Construction of **State space** (Reachability analysis)
  - **Structural Techniques** coming from Mathematical Programming and Graph theory. Reason of properties without the construction of the state space – which may be prohibitive
Structural Analysis (Intuitive)

- Strongly Connected marked graph
  - Each place has only one input and one output transition

- All net circuits contain at least one token

1. Any transition of the net is fireable from any reachable state (the net is live, thus deadlock free)

2. The wavefront propagate in an orderly manner without colliding one into another.

3. Two adjacent diagonals cannot fully operate concurrently

M. Silva, J.M. Colom
Petri Nets applied to the modelling and analysis of computer architecture problems. Microprocessing & Programming 38. 1-11. 1993
Structural analysis

1. Circuit with a token between adjacent transitions
   \( \Rightarrow \) Firing in mutual exclusion
1. Circuit with a token between adjacent transitions
   > Firing in mutual exclusion
2. Circuit with two tokens
   > Only two transitions can be fired concurrently.
Modify the Functional Model

Solution
- Introduce a DTP between two consecutive CPS (in an arrow, or column)
- It decouples mutual exclusion between adjacent diagonals.
Simulation
Synoptical View of the Methodology

Functional Level
- Control & data flow
- Resource sharing

Process
- Identify functional Entities
- Identify Dataflow & dependencies
- Resource sharing

Qualitative Analysis
- Verification of properties
- Finding Concurrency

Operational Level
- Resource Profiling
- Functional Patterns
- Map to resources

Quantitative Analysis
- Stochastics PNs
- Simulation & bounds
- Reward functions

Implementation
- Translation & model partition
- Node Orchestration & deployment

Monitoring
- Monitoring
- Support for Autonomic PaaS

Deliverables
- Functional Requirements
- Functional model
- Non-Functional Requirements
- Correct Functional model
- Operational model
- Target platform
- Reward functions
- Integrated model
- Flat model of processes
- Orchestration & Deployment Specification
- Application Infrastructure Mapping Catalogue
Operational model
Operational model #1

Operational wavefront array executed over nine independent VMs.
Operational model #2

Operational streaming 3 x 3 wavefront array executed in a Pipeline.
Synoptical View of the Methodology

**Functional Level**
- Control & data flow
- Resource sharing
- Process
  - Identify functional Entities
  - Identify Dataflow & dependencies
  - Resource sharing

**Qualitative Analysis**
- Verification of properties
- Finding Concurrency
- Structural Analysis
- Model Checking
- Simulation

**Operational Level**
- Resource Profiling
- Functional Patterns
- Map to resources
- Performance variability of resources
- Identify supporting architecture
- Identify Functional Patterns
- Mapping Patterns to resources

**Quantitative Analysis**
- Stochastics PNs
- Simulation & bounds
- Reward functions
- Integration functional & operational
- Computing Performance bounds
- Simulation
- Markov chain

**Implementation**
- Translation & model partition
- Node Orchestration & deployment
- Translation
- Model partition
- Deployment and Orchestration

**Monitoring**
- Monitoring
- Support for Autonomic PaaS
- Performance Monitoring
- QoS Analysis
- Autonomic PaaS

**Deliverables**
- Functional Requirements
- Functional model
- Non-Funct. Requirements
- Correct Funct. model
- Operational model
- Target platform
- Reward functions
- Integrated model
- Flat model of processes
- Orchestration & Deployment Specification
- Application Infrastructure Mapping Catalogue
PN Analysis techniques

- **Quantitative Analysis**, performance oriented interpretations (throughput/utilization rates/queue lengths).
  - *Executable* model. The model can be simulated and enriched with time and cost information.
- **Stochastic PNs**. Derivation of performance model from the reachability graph which is identified with a Markov chain.
Quantitative analysis based on Structural Analysis

- Computation of performance bounds is possible structurally.

- Lower bound for the mean cycle time is obtained by solving the linear programming problem:

\[
\Gamma_{min} = \text{maximum } Y^T \cdot Pre.\theta \\
\text{subject to } Y^T \cdot C = 0, Y^T \cdot M_0 = 1, Y \geq 0
\]

This means that the mean cycle times can be computed by the summation of all time delays involved in a circuit (P-semiflow) divided by the tokens in the circuit. In our 3x3 model is \( (1/\lambda)_{min} = 100\text{ms} \) (\( \lambda_{max} \) rate =10 data/sec).

Campos, J., Chiola, G., Colom, J. M., Silva, M. 
Properties and performance bounds for timed marked graphs. 
Quantitative analysis based on Structural Analysis

- Upper bound for the mean cycle time is obtained

\[ \Gamma_{\text{max}} = \sum_{j=1}^{m} \frac{\theta_j}{LB(t_j)} \]

Maximun mean cycle time

Liveness/ maximum degree of concurrency

In our example mean cycle time can be computed by the summation of all time delays involved in the longer circuit. In our 3x3 model is \((1/\lambda)_{\text{max}} = 404\text{ms} \quad (\lambda_{\text{min \ rate}} = 2.47 \text{ data/sec})\)
Quantitative analysis

- Translation of operational specification (timed transitions) to GreatSPN2.0.2
  - GreatSPN assumes service delivery time, data injection and transmission follow an exponential distribution.
  - Average processor service delivery time $1/\lambda = 100\text{ms}$ ($\lambda$ rate =10 data/sec)
  - Average Injection timed transitions $1/\gamma = 100\text{ms}$ ($\gamma$ rate =10 data/sec)
  - Average Transmission time $1/\beta = 1\text{ms}$.
- Reachability state with 1,392,640 states
- **Throughput for all transitions (processors)** $3,992,580$ data/sec
Simulation

- Distribution response time for a cloud modelled as a queue when service time is not exponential is complex.
  - The election of a exponential distribution is not adequate for task service times.
  - It depends on the Coefficient of Variation (CoV).
  - Khazaei et al. proposes CoV between 0.5 and 1.4 to give a reasonable insight into the behaviour of cloud centres (using a gamma distribution).

\[ CoV = \frac{\sqrt{Var[X]}}{E[x]} \]
Simulation results

Simulations with different CoV of Service Time and distributions
Petri net simulator (Renew)
Simulation results

Simulations with different CoV of Service Time using a gamma distribution Petri net simulator (Renew)
Synoptical View of the Methodology

**Functional Level**
- Control & data flow
- Resource sharing

**Qualitative Analysis**
- Verification of properties
- Finding Concurrency

**Operational Level**
- Resource Profiling
- Functional Patterns
- Map to resources

**Quantitative Analysis**
- Stochastics PNs
- Simulation & bounds
- Reward functions

**Implementation**
- Translation & model partition
- Node Orchestration & deployment

**Monitoring**
- Monitoring
- Support for Autonomic PaaS

**Deliverables**
- Functional Requirements $\Phi$
- Functional model
- Non-Functional Requirements
- Correct Functional model
- Operational model
- Target platform $\$\$
- Reward functions $\$
- Integrated model
- Flat model of processes
- Orchestration & Deployment Specification
- Application Infrastructure Mapping Catalogue

**Structural Analysis**
- Identify functional Entities
- Identify Dataflow & dependencies
- Resource sharing

**Model Checking**
- Structural Analysis
- Model Checking
- Simulation

**Performance analysis**
- Performance variability of resources
- Identify supporting architecture
- Identify Functional Patterns
- Mapping Patterns to resources

**Markov chain analysis**
- Integration functional & operational
- Computing Performance bounds
- Simulation
- Markov chain

**PN Tool box**
- PN Tool box analysis
- PN Tool box verification
Reference Nets + Renew + DVega

- “Nets-within-nets”
  - Systems net and an object net
  - Net can express creation of new net instances (“creational inscriptions”) – enabling dynamic self-modification of structure
  - Interaction via “synchronous channels”
  - Channel can contain variables whose binding is based on unification
  - Timed reference nets

- Renew
  - Java-based interpreter of Reference nets (an executable formalism)
  - Use tuples and Java expression as the inscription language
  - Objects nets can be Java objects

- DVega
  - Workflow engine based on Reference nets
  - Utilises a Linda-tuple space model for interaction between tasks in a workflow (referred to as a “message space”)
DVega

- Executable formal models
- Proposal of executable architectural model and patterns

Workflow container
Message router linda

Adapters +Meta-Scheduler
- Web Services
- Cluster
- Grid
- Cloud

Cluster Hermes - 819 CPUs
Piregrid/ Omnidata
Comet Cloud
Web Services

Tolosana-Calasanz, R., Bañares J. A., Álvarez, P., Ezpeleta, J.
Vega: A Service-Oriented Grid Workflow Management System
Failure! Exception raised.
Leaf nodes
Intermediate nodes

:send(id,["retrieve","args"])  :receive(id,args)

r1  id  r2

guard nw!=null;

:exception(ex)  :receive(id,nw)

e4 [null,ex]  e3 [nw,ex]

:w:new SubWfModel;
:w:begin(args);

t1

:w:send(id,msg);
:w:send(id,msg)

w:receive(id,result);
:w:receive(id,result)

:send(id,["checkpoint",result])

result

:t2

:w:end(result);

pattern output

output

:end(output)

w:exception(ex)

e1

nw:begin(args)

e6

e5 nw

w

E

ch1

w

:begin(input)

input

pattern input

args

ch2

w
Autonomic Streaming Pipeline

- **Streaming pipeline**
  - No “blocking” semantics
  - Continuous data transmission as a stream
  - Data processing order: arrival order (implicit) or time stamp (explicit)
  - After processing – result elements form the stream

- **Autonomic streaming**
  - Data stream “reacts” to changes in (operating) environment and producer/consumer data generation/consumption rate mismatch
  - Network congestion → alter transmission data rate
  - Alternative modes of analysis: in-transit, at-source, at-sink, etc

---

Tolosana-Calasanz, R., Bañares, J. A., Rana, O. F.
Autonomic streaming pipeline for scientific workflows
Autonomic Streaming Pipeline

Abstract workflow

Number of cores

Buffer Size

Pu to process in stage

Linda Spaces

Pu for In transit processing
Elasticity
System Architecture

Data injection rate

SLA

Tolosana-Calasanz, R., Bañares, Pham, C., J.A., Rana, O. F.
Resource Management for Bursty Streams on Multi-tenancy Cloud Environments.
Token Bucket (shaping traffic)

**Traffic shaping** component allows to control the traffic going out this component in order to match its flow to the processing speed of available resources and to ensure that the traffic conforms to policies contracted for it.

- A **policer** typically drops excess traffic.

- A **shaper** typically delays excess traffic using a buffer to hold data and shape the flow when the data rate of the source is higher than expected.
Token Bucket (shaping traffic)

Two key parameters of interest:

- **R**: Also called the **committed information rate** (CIR), it specifies how much data can be sent or forwarded per unit time on average.
- **B**: It specifies **for each burst how much data can be sent** within a given time without creating scheduling concerns.

**A(t)**: Amount of data arriving up to time *t*
Token Bucket (shaping traffic)
PETRI NET MODELS
Self adaptation with different performance constraints
Processing rate change

Processing rate change of data of ds1

Self-control of professing rate changes
Autonomic Platform as a Service

Tolosana-Calasanz, R., Bañares, Colom, J.M. 
On Autonomic Platform-as-a-Service: Characterisation and Conceptual Model  
KES-AMSTA 2015: 217-226
Conclusions

- **Model-Driven Methodology based on Petri net models**
  - Modelling Functional and operational level
  - Qualitative and quantitative Analysis
- **Executable Formal model**
  - Synergic use of analysis & Simulation
- **Proposal of Architectural models (SOA) & Patterns (Elasticity, Resilience)**
  - Exception handling patterns: Resilience
  - In-transit behaviour: Resilience/Elasticity
  - Token Bucket based control: Resilience / Elasticity
Model-Driven Development of Performance Sensitive Cloud Native Streaming Applications

PNSE’17