Java-based Distributed Architectures for Intensive Computations related to Electrical Grids

M. Di Santo, N. Ranaldo, A. Vaccaro, E. Zimeo
Department of Engineering - RCOST
University of Sannio, Benevento, Italy
zimeo@unisannio.it

IPDPS 2004
Work. on Java for Parallel and Distributed Computing
Santa Fe, New Mexico, USA
26 April 2004
On-line power systems security analysis (OPSSA) is one of the most relevant assessments made to assure the optimal control and management of electrical networks.

There are many phenomena (contingencies) that can compromise power systems operation:
- an unexpected variation in the power system structure
- a sudden change of the operational conditions

OPSSA deals with the assessment of the security and reliability levels of the system under any possible contingency.
Three main steps:

1. **Screening** of the most “credible” contingences

2. **Predicting** their impact on the entire system operation
   - The contingencies analysis is performed according to the \((n-1)\) criterion
   - For each credible contingency, the simulation of the system behaviour and the verification of operational limits violations
   - The system behavior is verified finding the solution of the system state equations (power flow or load flow equations)

3. **Preventive and corrective controlling**
   - Identification of proper control actions able to reduce the risk of system malfunctioning
• **Focus: on-line prediction (step 2)**
  – computation times should be less than **few minutes** for information to be useful

• **Unfortunately OPSSA is computing and data intensive**
  – structure of modern power systems
  – **computational complexity** of algorithms
  – number of contingencies to analyze

• **New methodologies** to reduce computational times
  – parallel processing on **supercomputers** and then on **cluster and network of workstations** (to reduce costs) has been employed (i.e. PVM)
Our proposal:
- A java-based distributed architecture instead of PVM

Advantages:
- Programming is easier
- Portability is assured on each architecture implementing a JVM
- Better integration with Web technologies
- Object-Oriented programming allows for adopting architectural and design patterns

Disadvantages:
- Efficiency is reduced due to Java communication overheads
- Execution time is higher due to interpretation
Contents

- The overall distributed architecture
- Computational engine
  - Algorithms
  - Design goals
  - RMI based implementation
  - ProActive based implementation
- Deployment on a testbed
- Conclusions and future work
The overall distributed architecture

scability
accessibility
manageability

Presentation tier

Internet - HTTP interactions

Applet
HTML client
Web Browser

Applet
HTML client
Web Browser

Internet - TCP and HTTP interactions

Electrical Grid

FEM
FEM
FEM
IED
IED
IED
IED
IED
IED

Servlet
JSP
Business Logic
Application Server

Computational Engine

Middle tier

Storage tier

DBMS
A network of field power meters (FEMs)

- distributed in the most critical sections of the electrical grid
- to provide input field data for power flow equations, such as active, reactive and apparent energy
- based on ION 7330-7600™ units
- equipped with an on-board web server for their full remote control
A network of distributed Intelligent Electronic Devices (IEDs)

- distributed in the most critical sections of the electrical grid
- to monitor continuously the thermal state of critical sections of the electrical grid
- to analyse system behavior
  - to verify limits violations (such as load capability)
- remote controlled by the TCP/IP protocol
Clients

- used by the power systems operators
- to access OPSSA information for system monitoring and management
Web components

- handle the presentation at the server-side
- access data stored in a DBMS
- continuously monitor the power system, coordinating the execution of security analysis
A DataBase Management System (DBMS)

- used to permanently store output data
- remote accessed

Presentation tier

Middle tier

Storage tier

Web Browser

Applet

HTML client

Web Browser

Applet

HTML client

Web Browser

Servlet

JSP

Business Logic

Application Server

Computational Engine

Business Logic

DBMS

Internet - TCP and HTTP interactions

Electrical Grid

FEM

IED

IED

IED

IED

IED

IED

DBMS

- used to permanently store output data
- remote accessed
- a parallel and distributed processing system
- to execute power flow equations in a short time
Computational engine: algorithm

- **Sequential algorithm**
  - Acquire field data
  - Compute the state estimation
    - Select a contingency
      - Compute the power flow solution
      - Check the voltage and power constraints
        - Check the equipment constraints \( i = 1 \leq I \)
          - \( i \leq \text{No} \)
            - Generate an alarm
              - NO
              - Contingency list empty?
                - NO
              - Any constraint violated?
                - NO
                - Generate an alarm
              - YES
            - YES
        - NO
      - \( i \rightarrow \text{No} \)
    - NO
  - Any constraint violated?
    - NO
    - Generate an alarm
  - YES

- **Concurrent algorithm**
  - Acquire field data
  - Compute state estimation
  - Schedule the jobs
    - Select the contingency \( k \)
      - Compute the power flow solution
      - Check the constraints
      - Check the constraints of the equipment \( i \) in contingency \( k \)
        - \( i \leq \text{No} \)
          - \( i \rightarrow \text{No} \)
        - NO
      - NO
    - NO
  - NO
  - \( k \rightarrow \text{No} \)
  - NO

Adopting a concurrent algorithm based on the domain decomposition
Computational Engine: design goals

- The computational engine is designed as a framework, whose main goals are:
  1. **high performance**
     - The framework is to be able to compute the analysis of each contingency in parallel with the others
  2. **flexibility and scalability**
     - The framework is to be able to exploit all the available resources to minimize the computation time
  3. **hierarchy**
     - The framework is to be able to exploit clusters of workstations or networks of workstations handled by front-ends

- **Architectural design solution:** hierarchical master/slave model
The hierarchical m/s model

- hierarchical master/slave model
  - object-level parallelism
  - master and slaves are transparently created
The hierarchical m/s model

Three implementation problems:

- Task allocation in order to minimize the execution time
  - Time minimization algorithm
- Object-oriented design to support a transparent hierarchical master/slave model
  - Hierarchical master/slave pattern
- Object-oriented implementation to support a distributed and parallel implementation of the hierarchical master/slave pattern
  - RMI based implementation
  - ProActive based implementation
Task allocation: time minimization

• This phase is important and complex due to the heterogeneity of hardware architectures
  – to minimize load imbalance, the workload should be distributed dynamically and adaptively to the changing conditions of resources.
• We consider only static information
  – Defined $N$ as the number of independent sub-tasks in which the overall task is divided;
  – $n$ as the number of available resources at a certain level;
  – and $t_i$ as the elapsed time that the resource $i$ needs to complete a single sub-task;
  – $n_i$ the number of sub-tasks assigned to the resource $i$
  – the problem to minimize the execution time using assigned resources can be formulated as follows:

$$
\begin{aligned}
& n_1 \cdot t_1 = n_2 \cdot t_2 = \ldots = n_n \cdot t_n \\
& \sum_{i=1}^{n} n_i = N
\end{aligned}
$$
The hierarchical m/s pattern

- **M/S pattern:**
  - `splitWork()` is used to create slave objects, to allocate them onto the available resources, and to partition a task into sub-tasks.
  - `callSlaves()` is used to call the method `service()` on all the slave objects, which perform the same computation with different inputs.
  - `combineResults()` is used to combine the partial results in order to produce the final result.

- **Hierarchical M/S pattern:**
  - An additional class (Server) is introduced
  - `service()` is used to hide the difference between master and slave objects
HM/S pattern implementation

• To support distributed and parallel implementation of HM/S pattern
  – `service()` has to be asynchronously invoked
  – Each `service()` method has to be executed by a different computational resources

• Solutions: an object-oriented middleware based on remote method invocations
  – RMI – the well known middleware provided by SUN
  – ProActive - Java library for seamless sequential, concurrent and distributed programming
    • It is based on the active object pattern and allows for invoking a method of an active object respecting the same syntax of local invocations
    • the invocation mechanism is asynchronous, and implemented through future objects
    • It does not require manually stub generation
Each Server
- is defined as a remote object
- is allocated onto a different computational resource
- its method `service()` has to be asynchronously and remotely invoked by the client
- the asynchronous invocation is implemented by invoking `service()` within a dedicated thread of control
• Each Server
  – is defined as an active object (that can be dynamically created)
  – is allocated on a different computational resource (dynamically)
  – its method `service()` has to be asynchronously and remotely invoked by the client.
  – the asynchronous invocation is directly supported by ProActive
Software platform evaluation on a testbed 1/2

- Standard IEEE 118-nodes test network
  - electrical network description is local to each computational resource

- The experiments refer to 186 contingencies

- A COW with P1 proc.
  - P1 = Pentium II 350 MHz

- A NOW with P2 proc.
  - P2 = Pentium IV 2 GHz

\[
\begin{align*}
    n_1 &= n_2 = \ldots = n_8 = 6 & P_1 \\
    n_1 &= n_2 = \ldots = n_4 = 35 & P_2
\end{align*}
\]

- RMI - ProActive
- Tomcat 4.1
- Java SDK 1.4.1
- MATLAB 6.5
The framework shows good results

RMI vs ProActive:
- Exhibit almost the same results (ProActive is based on RMI)
- ProActive simplifies parallel and distributed programming
- ProActive enables group communication (not used in this work)
Conclusions and Future Work

• A distributed Web-based architecture for the OPSSA implemented by using RMI and ProActive middleware, has been presented
• Experimental results demonstrate the validity of the framework and its applicability to obtain results in more and more realistic and useful times

• In a parallel work we are:
  – using ProActive atop HiMM in a Grid Environment
  – adopting a broker-based architecture to automatically acquire resources and split the workload

• In the future we intend:
  – to include the solution of the power flow equations in the dynamic scenario
  – to extend our mapping algorithms in order to consider also communication tasks
  – to employ group communication in order to reduce communication overheads