High Performance Hydraulic Simulations on the Grid using Java and ProActive

Guilherme Peretti Pezzi, Denis Caromel, Evelyne Vaissié, Yann Viala, Bruno Grawitz, Frédéric Bonnadier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

High Performance Hydraulic Simulations on the Grid using Java and ProActive

Guilherme P. Pezzi — Denis Caromel — Evelyne Vaissié — Yann Viala — Bruno Grawitz — Frédéric Bonnadier

N° 7508

January 2011

Distributed Systems and Services

ISSN 0249-6399 ISRN INRIA/RR--7508--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

High Performance Hydraulic Simulations on the Grid using Java and ProActive

Guilherme P. Pezzi * † , Denis Caromel † , Evelyne Vaissié * , Yann Viala* , Bruno Grawitz* , Frédéric Bonnadier*

Theme : Distributed Systems and Services
Networks, Systems and Services, Distributed Computing
Équipe-Projet Oasis

Rapport de recherche n° 7508 — January 2011 — 19 pages

Abstract: This document presents the work of redesigning a legacy hydraulic simulation software developed by the *Société du Canal de Provence* in order to solve its performances issues using Grid Computing and also to enable interactions with newer systems used in the company.

Key-words: water distribution networks, hydraulic, hydrodynamics, simulation models, java, HPC computing, grid computing

* Société du Canal de Provence et d'aménagement de la région provençale
† INRIA Sophia-Antipolis - Université de Nice Sophia Antipolis

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Simulations hydrauliques de haute performance dans la Grille avec Java et ProActive

Résumé : Ce document présente le travail de rénovation d'un logiciel de simulation hydraulique développé par la *Société du Canal de Provence*. Le but est d'en augmenter les performances en termes de temps de calcul en utilisant des grilles informatiques et de permettre son intégration dans les outils du système d'information de la société.

Mots-clés : réséaux de distribution d'eau, hydraulique, hydrodynamique, modèles de simulation, java, calcul de haute performance, grilles informatiques

Contents

1 Introduction

The *Société du Canal de Provence* (*SCP*) is a company responsible for building and maintaining water distribution networks (*WDN*). For this purpose the company uses extensively a simulation software called IRMA. IRMA started being developed the *SCP* in 1977 and it implements a demand model (*Débit de Clément*) [8] that was developed to estimate on-demand irrigation consumption behavior.

IRMA was written first in Fortran 77, it was later converted to Fortran 90 and nowadays it is very difficult to implement new features or interactions of this legacy code with current technologies deployed at the *SCP*. IRMA had also some performance limitations due to the growth of its networks and maintaining this software became too expensive because of the lack of qualified Fortran developers.

For these reasons the project of redesigning IRMA started, with the goal of building a new simulation engine using up-to-date technologies, allowing to overcome the performance issues and to deliver new features for the users.

This paper will present the development process of this new tool written in Java, highlighting the challenges of migrating a scientific Fortran application to a modern object oriented environment as well as the performance results obtained on the Grid using the ProActive [6] Middleware.

2 Simulation model

IRMA underlying engine computes pressures, flow and head losses in piped water distribution networks. In top of that, there are other layers for modeling equipments, demands, tank state over time. Most IRMA features are very similar to EPANET [12], which is extensively used for *WDN* simulation, but some features are specifically designed to fit *SCP*'s needs and methodologies.

2.1 Overview

The mathematical model used by IRMA is based on the Kirschoff's laws (commonly applied to circuits). These laws are used to create a set of equations that are divided in 3 groups:

- First group of equations represents the flow conservation at the nodes: sum of the flows in each node must be *zero*.
- Second group represents the meshes' head loss equations: sum of head losses from the first to the last node in each loop must be *zero*.
- Third group represents the head losses in the paths between tanks: head loss in each path between two tanks must be equal to head difference between first and last tank.

The first group is filled with linear coefficients but second and third groups contain non linear elements and must be linearized in order to be solved using traditional equations solving algorithms (Sec. 2.4).

IRMA determines initial flow values for each pipe and then calculates the head in each node. The resolution is then performed iteratively by applying the Kirschoff's laws to calculate the resulting flows and then updating the heads based on these new flows. This process is repeated until the sum of all differences between input and result flows are smaller than a determined convergence value or until a maximum number of iterations is performed (without reaching a stable state).

2.2 Equipments

Most networks have equipments in order to give satisfactory performance, such as pumps, pressure regulators and flow limiters. IRMA is able to represent these equipments and also as a tool for studying the conception of new equipments in an existing network. Here are some examples:

Pumps: are usually modeled by giving a negative head loss value to the concerned pipe. Pumps can have fixed or variable speeds and can be described for example by it's equation $(a + b.Q + c.Q^2)$, imposed flow or head value.

Pressure regulators: are used to keep the pressure above/below a certain value. They can be upstream or downstream and are modeled by adjusting the head loss on the concerned pipe until the criteria is respected.

One-way valves: are modeled by closing the pipe whenever IRMA detects a flow going into the wrong direction.

IRMA can also model other structures such as singular/differential head losses, flow regulators and closed pipes. New equipments can be easily modeled

simply by implementing how to calculate its head loss coefficient or by extending an existing equipment.

2.3 Flow conservation equations

Each node is represented in the linear system by one equation that defines its connections with all other pipes: assigning $'1'$ to represent a connection and $'0'$ otherwise. These equations represent flow conservation at the nodes, therefore the system's second member will express the demand in each node.

Since each pipe is usually connected to a maximum of 3 other pipes and the system is built using mostly flow conservation equations, the resulting matrix will be sparse.

The remaining equations describes the head loss in meshes and path between tanks, as explained in next section.

2.4 Head loss equations

The head loss (J) in each pipe is calculated in function of the flow (Q) and is given by the formula:

$$J(Q) = -(a + b.Q + c.Q^2) + \alpha.Q^\beta + D.Q^2$$

where a, b, c represent pump coefficients (if it exists), α represents linear head loss coefficients and D represents singular head loss coefficient.

This head loss expression is continuous and derivable. Therefore we can write, with $Q = Q_0 + \Delta Q$ and $J'(Q_0) = (\frac{dV}{dQ})Q_0$:

$$J(Q) = J(Q_0) + J'(Q_0).\Delta Q = (J(Q_0) - Q_0.J'(Q_0)) + J'(Q_0).Q$$

Head loss in a mesh

The sum of all head losses in a mesh is *null*. In a mesh we have $J(Q) = 0$, therefore:

$$\sum J'(Q_0).Q = -\sum (J(Q_0) - Q_0.J'(Q_0))$$
First member . X = Second member

This formula will be used to write one equation for each mesh in the network.

Head loss in a path between tanks

In these path equations the sum of all head losses in each path is equal to the head difference between the two tanks. In each path we have $J(Q) = \Delta H$, therefore:

$$\sum J'(Q_0).Q = -\sum (J(Q_0) - Q_0.J'(Q_0)) + \Delta H$$

This formula will be used to write one equation for each 2 tanks in the network. If a network contains t tanks, there will be $t-1$ tank equations in the system.

2.5 Linearizing head loss values

The methods for expressing head losses usually contains non linear elements in its formulas and thus they must be linearized in order to build the system of linear equations. IRMA can use use different methods for calculating the head loss: Williams/Hazen and Lechapt/Calmon and Colebrook.

For example, given the Colebrook equation:

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\varepsilon/D}{3.7} + \frac{2.51}{\mathrm{Re}\sqrt{f}}\right)$$

where f is the Darcy friction factor, ϵ is roughness height, D is the pipe diameter and Re is the Reynolds number. Head loss values (J_1 and J_2) are calculated with this formula using two reference flow values. Then, a linear head loss coefficient α is determined by the equation:

$$\alpha = (2.\sqrt{J_1} - \sqrt{J_2})^2.(1 - \frac{\sqrt{J_1}-\sqrt{J_2}}{2\sqrt{J_1}})^2$$

After obtaining the linear coefficient α, the following equation is used for calculating the head loss:

$$J(Q) = \alpha.Q^2$$

Finally, matrix's head loss equations (meshes and path between tanks) will be filled using this formula:

$$J'(Q_0) = \frac{J(Q_0+h)-J(Q_0)}{h}$$

where h is a constant small value.

2.6 Sample network

This section presents a sample network containing the basic elements modeled by IRMA and its equation system.

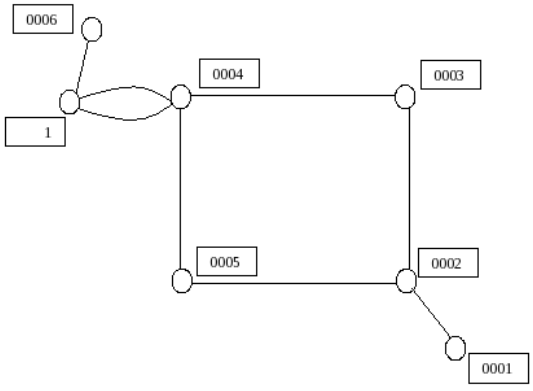

Figure 1: Sample network with 7 nodes, 8 pipes, 2 tanks and 2 meshes.

Figure 1 shows a diagram of this network, nodes 0001 and 0006 are tanks and there aren't any equipments. The meshes are formed by the nodes $\{1-0004\}$ and $\{0002-0003-0004-0005\}$.

$$\begin{bmatrix} -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & -5.82813 & -5.82813 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -612.153 & -612.153 & -58.2814 & -58.2814 & 0 \\ 5.82813 & 5.82813 & 0 & 612.1526 & 612.1526 & 0 & 0 & -58.2814 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ Q_7 \\ Q_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1.47132 \\ -53.3477 \\ -50.3808 \end{bmatrix}$$

Figure 2: Linear system that represents the sample network in a given iteration.

Figure 2 shows the equation system that represent network from Figure 1. Lines 1 to 5 represent connections among ordinary nodes and since there's no consumption second member is always equals to 0. Lines 6 and 8 represent meshes equation and line 8 represents the path equation between the 2 tanks.

3 Choosing a linear system solver

Since IRMA's resolution method relies on an iterative simulation its performance will heavily depend on the linear system solving engine. This section presents the linear solvers studied and their performance.

3.1 Java solvers

The former Fortran solver implements a direct method for solving the linear system and for this new version we tried to use more recent iterative methods that are known to be more efficient. Several solving and preconditioning methods were tried, using 3 different Java Numerical Libraries: Jama [4], Colt [3], MTJ [5]. Unfortunately none of these methods was able to solve all of our use cases, some methods partially succeeded but presented convergence problem in some cases.

After obtaining these results we decided to continue using a direct method for solving the system, more specifically the LU Decomposition. Besides the 3 libraries already mentioned, a test was performed with a Java library [2] that is a translation of the Fortran BLAS implementation, which gave the best results among the tested dense matrix solvers.

Even if these solvers give correct results, none of them implement sparse matrix storage when using direct methods. This can be acceptable for solving networks with up to 1.000 pipes but cannot scale much further. Since our networks that can have up to 15.000 pipes none of these libraries could be adopted.

3.2 Fortran solver

We finally decided to fall back to the original solution, by integrating the solver used in the Fortran to the Java code. This Fortran library [9] was written at the Yale University and it implements the LU Decomposition with sparse matrix storage. This package also allows the reuse of *pivots*, what can dramatically speed up our simulation because we need to solve several times our systems and the *pivots* rarely change among the iterations.

In order to use this Fortran library from Java we used native calls using JNI (Java Native Interface). Since JNI does not allow direct access to Fortran code from Java, a C++ interface was written to wrap the Fortran library. The main drawbacks of this approach are reducing the portability of the Java code and complicating the compilation process, but the code was successfully deployed in Windows, HP-UX Unix and Linux.

This library takes as arguments the list of system's *pivots* and the matrix stored using the Yale sparse matrix format. Therefore the last requirements for using this library are to search the *pivots* before first iteration (also after a few particular cases, e.g. when a pipe is closed between two iterations) and write the matrix respecting Yale's compressed matrix storage format.

3.3 Performance results

This section presents the performance results of the previously mentioned solvers when simulating the network with 1.763 pipes called *"TAMAGNON LA MARO-*

NNE LES PLAINES". These tests were performed in a Desktop PC running Linux except for the Fortran version that was executed in the production HP-UX Unix server.

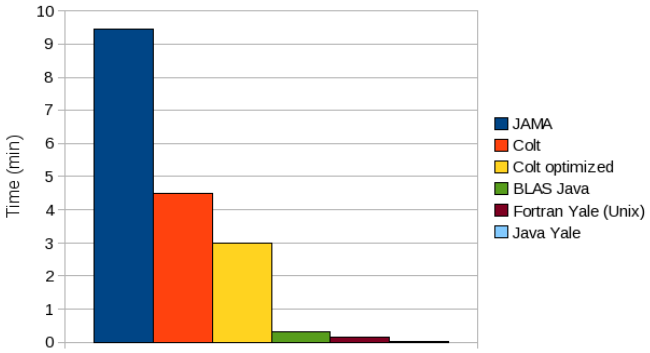

Figure 3: Execution time of a network simulation using different solvers in a Linux machine and the former IRMA Fortran version executed in a HP-UX machine.

Figure 3 shows the execution times: Jama, Colt and Blas are pure Java solutions, Fortran Yale is pure Fortran and Java Yale is a mixed Fortran/Java solution (as explained in Sec. 3.2).

In Colt optimized the results come from a modified library version where we adapted the LU Decomposition in order to save the *pivots* found on the first iteration and reuse them in the following iterations.

At this point, BLAS Java presents acceptable performance for being a 100% Java solution but lacking sparse matrix storage it does not scale for simulating networks containing more than 3.000 pipes.

4 Simulation features

This section presents some of the most important simulation features offered by IRMA. These features can be used to calculate peak demand, maximum pressures, the impact of adding new clients to existing networks, extended period simulation (tank design), to design new pumps and also other equipments.

4.1 Demand scenarios

IRMA can perform different kinds of simulation according to the user's needs and adopt different **consumption scenarios** if necessary. In order to calculate the maximum pressure that can be obtained in each node, a scenario without consumption will be simulated.

If we want to calculate the pressure obtained in each node where all the demands are known, a scenario with only continuous flows will be adopted.

SCP's customers can be domestic, rural and industrial, and their demands are not known beforehand. For these networks another type of scenario will be calculated using a probabilistic approach. A method was developed for calculating the flow taking into account *SCP*'s customers demand and is called ***Débit de Clément*** , more detail about this model can be found in [8] and [11].

Basically, this method divides all the offtakes in groups of the same class and simulates iteratively these groups using for each individual offtake considering its probability of use. This probability can be assigned directly to one customers or can be taken from its probability zone (for example rural regions with the same type of crop).

4.2 Pump profiling

IRMA can also be used to project new pumps in a network. Pump profiling is performed first by defining a fixed amount of flow (demand) objective values, varying from the minimum demand that can be observed (only continuous flows) up to a demand that is equivalent to 25% more than the estimated peak consumption.

Second step is to generate the possible offtake scenarios that will reach each objective flow value. These scenarios could be generated exhaustively in networks containing only a few offtakes, however in most networks this is not possible and for that reason they will be randomly generated. The number of scenarios will be calculated in function of the number of offtakes, with a minimum of 250 scenarios in order to guarantee a minimum sample size in case of networks with a small number of offtakes.

Finally, all these scenarios will be simulated and the resulting value for each scenario will be the required head in the node where the pump will be placed in order to provide the guaranteed pressure to all clients. All this data will then gathered to plot one graph where each line represents the required head to satisfy a % of scenarios.

Figure 4 displays a sample result of this simulation in an existing network, objective flows are represented in the x axis, required head in the y axis and each line represent a % of scenarios that are satisfied.

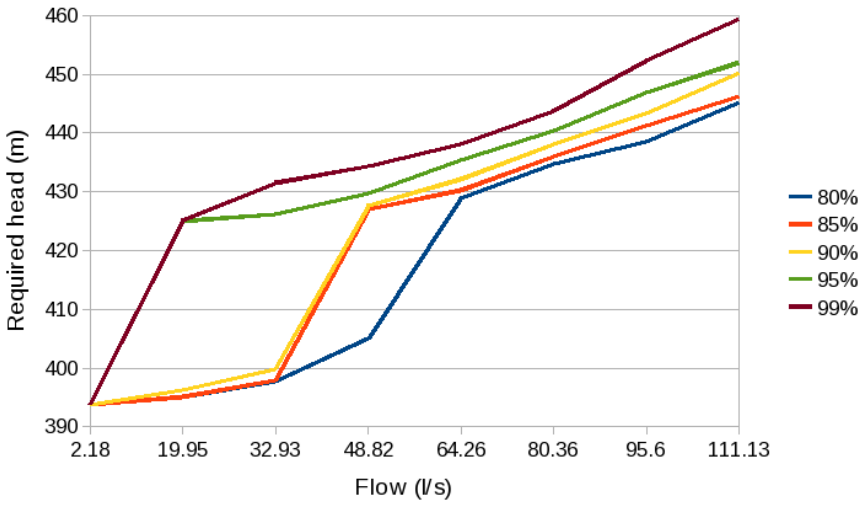

Figure 4: Pump profiling in the "Boutre" network.

4.3 Pressure driven analysis

The demand is usually fully satisfied under normal pressure conditions, but in scenarios with low pressures modeling the demand independently from the pressure is not realistic and may often lead to negative pressures in the results.

The alternative for demand driven analysis is to take into account the pressure obtained in each point in order to estimate the actual demand. There are several approaches for performing this analysis, for example [10] and [7] where it is presented an extension to EPANET to perform pressure driven analysis.

In IRMA the user can associate to each node a nominal flow value Q_n and a couple of bounding pressure values P_0 and P_1. The estimated flow Q is obtained as follows:

- If pressure P is between the bounding pressure values $[P_0, P_1]$, flow Q is taken as a fraction of Q_n equals to $[\frac{P-P_0}{P_1-P_0}]^{\frac{1}{2}}$
- If P is lower than P_0, Q is taken as 0
- If P is higher than P_1, flow Q is taken as:

  Q_n if there's a flow limiter

  $[\frac{P-P_0}{P_1-P_0}]^{\frac{1}{2}}.Q_n$ if there's no flow limiter

These coefficients will be iteratively calculated in function of the obtained pressures, until convergence is reached.

5 AGOS Project use case

Rewriting IRMA was done as part of the AGOS Project (*Architecture Grille Orientée Services*) [1], a project that provides a Service Oriented Architecture (SOA) Framework for deploying and running applications on the Grid. One of the main reason for choosing Java as new programming language for IRMA is to be able to easily integrate with the AGOS infrastructure. This section presents the work done with IRMA as use case for validating AGOS framework and performance results obtained in the Grid using the AGOS IRMA prototype.

5.1 Deploying IRMA as a service

IRMA is a standalone Java application that processes an input file, simulates the network and generates an output file to write the results.

First step for integrating IRMA in the AGOS infrastructure is to wrap the Java application as a service that processes the input file and gives as result the simulation's results. This service will then be made available to the users by deploying it on the Grid as a web service using Apache TomCat [1].

This way users are already capable of launching simulation one by one, but we also want to provide a way to launch a batch of simulations at once. For doing that we adapted one template service provided by the AGOS infrastructure for performing parameter sweeping. This new service will basically take as input a folder containing several IRMA input files and will automatically launch separately each file and then will retrieve all result files back to the user.

The other key elements behind this infrastructure are the ProActive Resource Manager, used to build and manage the Grid, and the ProActive Scheduler that is used to process and dispatch the tasks among the available resources.

5.2 Sequential benchmarks

Before benchmarking IRMA using the Grid, it's necessary benchmark sequentially both Fortran and Grid enabled Java versions in order to measure IRMA's performance evolution since the project started.

Figure 5 presents the execution times using original solution on the Unix server and the new solution on a Linux desktop machine. This set of networks is very representative regarding the network size, they contain the largest networks simulated in the *SCP* (with 1.845 up to 10.395 nodes).

Due to an inefficiency problem in the Fortran version, in these tests the Fortran version does not perform a full analysis of the network topologies, it uses cached results instead. In the new version this problem has been solved, more details in Sec. 6.2.

The purpose of this test is to guarantee there are no performance regressions when using standard user desktop machines with the new version, regarding the Fortran version that currently runs only in the dedicated HP-UX Unix server.

5.3 Grid execution with ProActive

This section presents IRMA's performance in a Grid infrastructure using the ProActive Middleware [6]. ProActive enables us to easily build a Desktop Grid

[1] http://h30423.www3.hp.com/?fr_story=c4ec832e00c43f2791a9e4f15189d1cd5a91f819

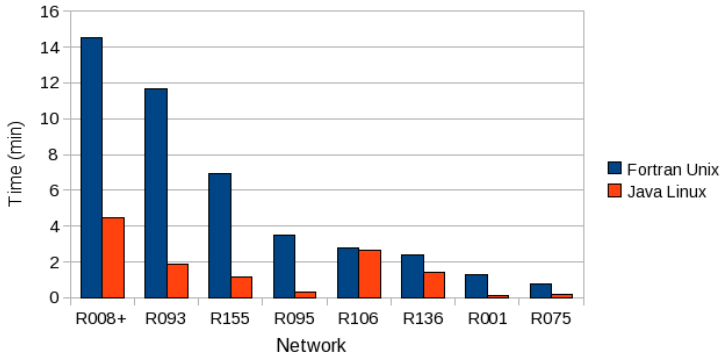

Figure 5: Execution time of networks selected for the AGOS Use Case.

using the machines available in the local network. We can then use these resources to run instances of application in any machine transparently to the user. This means ProActive will automatically transfer user files before execution and retrieve results back to the user machine after executing.

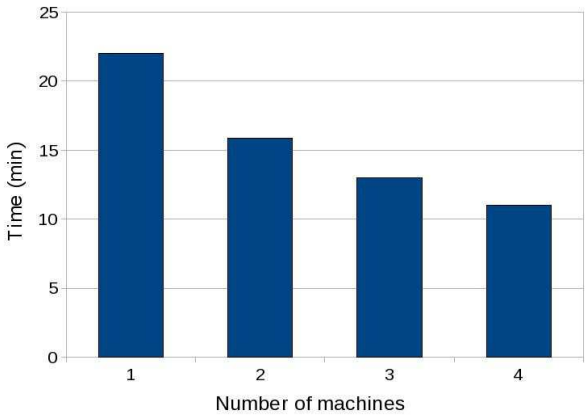

Figure 6: Execution time of 119 networks on the Grid using up to 4 machines.

Figure 6 shows the execution time when simulating a batch of 119 networks using 1 machine up to 4 machines. Total sequential execution time is around 22 minutes and with 4 machines we reach 11 minutes, which is already close to the longest sequential execution time since the largest network runs in 7 minutes.

6 Results

This sections explains how the development of IRMA was validated and then presents some of the improvements obtained regarding the original Fortran version.

6.1 Validation

The first phase when validating the Java version consisted in comparing new version's results with the Fortran version. This validation is performed automatically by a module that can (optionally) run the former IRMA version, read the Fortran results and compare with the new version the following values: heads, flows, maximum pressures and head losses. This module reports if there's any difference superior to 1% in any of these values. We have simulated all networks from the 'Maintainance' department and currently this test passes for 108 networks out of 115.

IRMA's results have also been compared with results obtained with EPANET by using a module that read EPANET's *.inp* files. This module is able to read network topology, but currently it does not treat other sections (like equipments).

Table 1: Flow differences between EPANET and IRMA in a sample network.

Pipe		Flow (l/s)		Difference	
Node 1	Node 2	EPANET	IRMA	l/s	%
6	T	-683.87	683.06	0.81	0.12
T	4	341.93	341.53	0.40	0.12
4	T	-341.93	-341.53	0.40	0.12
5	2	-95.10	94.99	0.11	0.12
4	5	95.10	94.99	0.11	0.12
3	4	-588.77	-588.07	0.70	0.12
2	3	-588.77	-588.07	0.70	0.12
1	2	-683.87	-683.06	0.81	0.12

In order to compare both simulators, a sample network was designed using EPANET, exported through *.inp* file and read by IRMA. In this test there are some differences in flow results, but head and head loss values obtained are exactly the same in both simulators. Table 1 shows the differences between EPANET and IRMA flow results, they vary in l/s but the difference in % is constant.

6.2 Topology analysis

Redesigning completely IRMA brought the possibility to improve it in many ways. The improvement that most impacted the users is the **topology analysis** phase. Execution time has been significantly reduced by simplifying Fortran data structures and by optimizing the algorithm that performs the topology analysis.

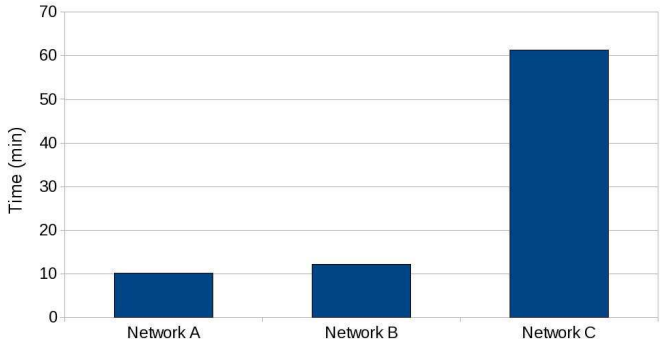

Figure 7: Time required to perform topology analysis using the IRMA Fortran.

Figure 7 represents the time required to analyze the topology of 3 networks, from 10 minutes up to 1 hour. Some cases, heavily meshed, can even require days to be analyzed. As a workaround, the analysis is cached after first execution so that the user does not need to wait for this analysis in every execution, as long as the topology is not modified.

In the new version, topology analysis has been optimized and simplified using modern Java data structures to represent the graph in a way that topology analysis now takes less than 1 minute for any *SCP* network. We no longer need to cache topology results and the user can always modify the network structure with no impact in execution time.

6.3 Convergence optimization

Most networks reach stability after a few iterations when simulated with IRMA. However, it may also happen that several iterations are required and in some cases we never reach a stable state. This can be caused by several reasons, in most cases the equipments are responsible. The way equipments are modeled or the interaction of two or more equipments may cause new adjusts in coefficients in every iteration indefinitely.

Figure 8 shows the number of iterations performed before reaching a stable state in all networks from the 'Maintainance' department, excluding pathological cases that do not converge at all. The main consequences of not converging are the loss of precision in the results and the increase of execution time because IRMA performs a maximum amount of iterations.

The former IRMA version has set as 500 the maximum number of iterations. In some simulations with convergence problems this limit is reached several times, because the simulation is performed for each class of offtake when calculating the peak consumption. In these cases where stability is not reached, simulation time is much longer and accurate precision is not guaranteed even after 500 iterations.

In order to decrease execution time in these cases we have decided to change the maximum number of iterations, like in EPANET for example, where this limit can be set by the user. Figure 9 compares the execution time of 3 different networks with convergence problems when reducing this limit from 500 to 50

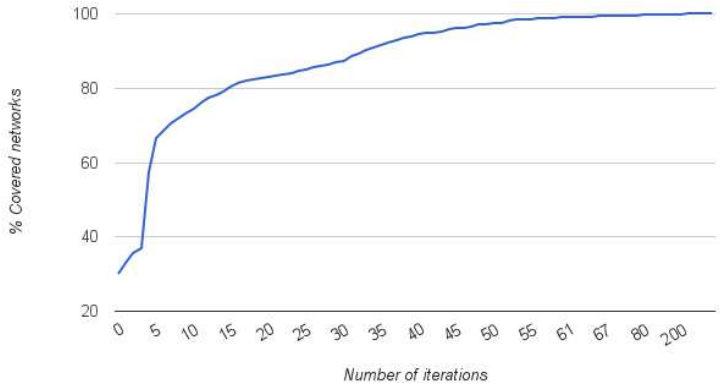

Figure 8: Number of iterations required for converging *SCP*'s networks.

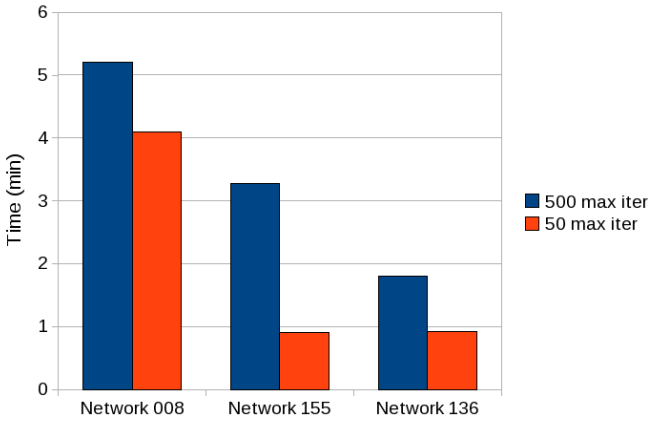

Figure 9: Execution time with 2 different iteration limits for 3 networks with convergence problems.

iterations. The given results with this change are slightly different but the difference is always lower than 1%.

7 Final considerations and future work

The new IRMA version is now able to replace the legacy Fortran code for simulating most networks, with better performance and portability. There are only a few special cases yet to treat before delivering this new tool to production.

This work explores the Java language for scientific high performance simulations, facing the challenges of not using the standard languages for solving numerical problems and how to overcome these difficulties. The final solution exploits the advantages of standard numerical tools, by using a scalable high performance Fortran library with minimal memory footprint, and the advantages of developing the new model using a modern object oriented language.

The new high level simulation model is modularized and it can be understood more easily. Equipment specification has become much simpler and independent from the linear system construction. Code reading has been enormously simplified by eliminating global variables, `goto` calls and other programming practices that were used when IRMA was coded using punch cards.

Many possibilities for improving user experience are now open, text input file editing can now be replaced by geographically-aware graphical editors and simulation data can be integrated with other tools available at the company.

There are still improvements and tests that can be done in the simulation core, such as implementing another method for solving flow continuity and head loss equations. For example, the use of the gradient method proposed by Todini [13] could allow us to replace the Fortran solver with one of the Java iterative linear solver implementation and therefore we could eliminate the C++/Fortran JNI bridge and have a pure Java solution.

References

[1] Apache tomcat website, 2010. <http://tomcat.apache.org/>.
[2] Blas java translation, 2010. <http://www1.fpl.fs.fed.us/linear_algebra.html>.
[3] Colt project website, 2010. <http://acs.lbl.gov/software/colt/>.
[4] Java matrix (jama) website, 2010. <http://math.nist.gov/javanumerics/jama/>.
[5] Matrix toolkits java (mtj) website, 2010. <http://code.google.com/p/matrix-toolkits-java/>.
[6] Proactive website, 2010. <hhttp://proactive.inria.fr/>.
[7] P. B. Cheung, J. E. Van Zyl, and L. F. R. Reis. Extension epanet for pressure driven demand modeling in water distribution system. In *Computing and Control for the Water Industry*, volume 1, pages 311–316. Center for Water Systems, University of Exeter, 2005.
[8] R. Clément. Calcul des débits dans les réseaux d'irrigation fonctionnant à la demande. In *La Houille Blanche*, volume 5, pages 553–576. Société Hydrotechnique de France, 1966.
[9] S. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package, ii. the nonsymmetric codes. Technical report, Department of Computer Science, Yale University, 1977.
[10] O. Giustolisi, D. Savic, and Z. Kapelan. Pressure-driven demand and leakage simulation for water distribution networks. *Journal of Hydraulic Engineering*, 134(5):626–635, 2008.
[11] N. Lamaddalena and J. Sagardoy. *Performance Analysis of On-demand Pressurized Irrigation Systems*. Food and agriculture organization of the United Nations, 2000.
[12] L. A. Rossman. *EPANET User's manual*. U.S. Environmental Protection Agency, Risk Reduction Engineering Laboratory, Cincinnati, OH, 2000.
[13] E. Todini and S. Pilati. *A gradient algorithm for the analysis of pipe networks*, pages 1–20. Research Studies Press Ltd., Taunton, UK, 1988.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399