Distribution of a Stochastic Control Algorithm Applied to Gas Storage Valuation

Constantinos Makassikis, Stéphane Vialle, Xavier Warin

***Abstract*—This paper introduces a research project that aims to speed-up and size-up some gas storage valuations, based on a *Stochastic Dynamic Programming algorithm*. Such valuations are typically needed by investment projects and yield prices of gas storage spaces and facilities. However, they involve computations which require great amounts of CPU power or memory. As a result, their parallelization on PC clusters or supercomputers becomes highly attractive and sometimes unavoidable despite its complexity.**

Our parallelization strategy is based on a message passing paradigm, and distributes both computations and data on a cluster, in order to achieve speed-up and size-up. It includes some complex and optimized data exchanges which are dynamically computed, planned and achieved at each computation step. This optimized data distribution and memory management allows to process large problems on a high number of processors. Moreover, our parallel implementation is able to support different price models, and our first experiments on a standard 32 PC cluster show very good performances particularly for complex price models.

I. Introduction and objectives

Gas prices exhibit fluctuations which are mainly due to the modification of demand. Because of the inelasticity of production and demand, prices are, for example, higher during winter than in summer. A gas storage facility allows its owner to take advantage of the price dynamic to do some arbitrage between periods where prices are high and periods where prices are low. Recently, a lot of research has been achieved in the field of gas storage valuation (see [1], [2], [3] for example). As a result, many different price models can be used to carry out this valorization. In our study, we use three different models which are based on the dynamic of the forward curve that is given by the gas market prices for a future delivery of energy:

- the first one is a one-factor model based on an Ornstein-Uhlenbeck process described in [4] and in section II,
- the second one is based on a two-factor Ornstein-Uhlenbeck process, hence a two-factor model,
- the third one is a one-factor model similar to the first model except that the Brownian motion used is replaced by a normalized Normal Inverse Gaussian process [5].

All valuations are achieved by a stochastic programming approach described in [4] and in section II. If the time needed to compute the solution with the first model is not too long (typically less than 10 minutes), the time needed by the other models makes them virtually unusable. Hence the need of parallelization.

From a computer science point of view this is a *Stochastic Dynamic Programming algorithm* which is complex to parallelize. Indeed, despite some natural parallelism (see Stochastic control algorithm of section II), computations at any given step depend on previous results and the range of data to be computed changes regularly. As a result, computations and data need to be redistributed at each step. This requirement leads to compute and execute a complete routing plan at each step on each processor. Moreover, the data required by each processor for the next step needs to be finely identified, in order to route and store the minimal amount of data on each processor. This strategy is necessary to process large scale problems on large numbers of processors. A systematic broadcast and storage of all previous results on each processor would be easy to implement but would require too much memory on all processors.

A lot of research in financial computations focus on the parallelization of option pricing, considering independent computations [6] as well as computations requiring many communications between processors [7]. However, gas storage valuation equations need to take into account *gas stock levels* and lead to different computations and distributions.

Section II introduces the mathematical problem and the initial algorithm. Section III lists the devel-

SUPELEC, IMS research group, 2 rue Edouard Belin, 57070 Metz, France, and LORIA, ALGORILLE project team, BP 239, 54506 Vandoeuvre-lès-Nancy, France, *Constantinos.Makassikis@supelec.fr*

SUPELEC, IMS research group, 2 rue Edouard Belin, 57070 Metz, France, *Stephane.Vialle@supelec.fr*

EDF - R&D, OSIRIS group, 1 Avenue Charles de Gaulle, 92141 Clamart, France, *Xavier.Warin@edf.fr*

opment tools used by the sequential version and kept in the distributed version. The distributed and optimized algorithm for clusters is described in section IV, and the first experimental results are introduced in section V. Finally, section VI summarizes this research and our current results.

II. Stochastic control application

A. Description of the problem

A gas storage facility presents three regimes: injecting gas, withdrawing gas, and just storing the gas. The gas storage is a cavity characterized by:

- its size given in giga British Thermal Units (BTU) or MWh (a standard conversion rate is used to convert BTU to MWh prefered by electric utility),
- the daily injection/withdrawal capacity a_{in} / a_{out} which depends on the stock level of the cavity I_t,
- the standard operating and managing cost per day which depends on the operating regime: $K_{in}(I_t)$, $K_s(I_t)$, $K_{out}(I_t)$.

The storage size can be variable in time because we may want for example to hire a portion of the cavity.

Most of the time, the gas storage manager uses its facility according to a *bang bang strategy*. In this case, the instantaneous gain (or cost) at a date t depends on the gas price S_t and the management regime. Here are the characteristics of the three regimes:

$$\begin{cases} \text{Injection} & a_{in,s}(I_t), \text{ with cost:} \\ & \phi_{-1}(S_t, I_t) = -S_t a_{in}(I_t) - K_{in}(I_t) \\ \text{Storage} & \text{with cost:} \\ & \phi_0(S_t, I_t) = -K_s(I_t) \\ \text{Withdrawal} & a_{out,e}(I_t), \text{ with gain:} \\ & \phi_1(S_t, I_t) = S_t a_{out}(I_t) - K_{out}(I_t) \end{cases}$$

The instantaneous evolution of the stock I_t depends on the regime of the facility:

$$\begin{cases} dI_t = a_{in,s}(I_t)dt & \text{in injection} \\ dI_t = 0 & \text{in storing} \\ dI_t = -a_{out,e}(I_t)dt & \text{in withdrawal} \end{cases}$$

If we suppose that a strategy u_t describing the regime taken at date t can take three values: 1 in withdrawal regime, 0 in storing regime,-1 in injection regime, and if we suppose that the regime switching can occur at any date, the gain obtained by managing the facility from a date t until to a date T with a strategy u is given by:

$$J(t,s,c,i,u) = \mathbb{E}(\int_t^T \phi_{u_r}(r, S_r, I_r)dr + J(T, S_T, I_T, i_T, u_T)|S_t = s, I_t = c, u_t = i)$$

where:

- $J(T, S_T, I_T, i_T, u_T)$ a given final value function,
- s the gas price in t given by a markovian process,
- c the stock level in t,
- i the regime in t.

The goal of the manager is to find an optimal admissible adapted strategy in a given set $\mathcal{U}_t$, in order to maximize its income and therefore to solve:

$$J^*(t,s,c,i) = \sup_{u \in \mathcal{U}_t} J(t,s,c,i,u)$$

B. Stochastic control algorithm

In our models, the price of electricity is given by a markovian process. We use stochastic dynamic programming in order to optimize the management of the facility. The stock is discretized with equally spaced levels. Furthermore, the regime switching occurs only at given dates (once a day): so we discretize the time with an equally space step Δt. From the final value of J^*, we evaluate the value J^* for all the previous dates and all the levels of the stock with the algorithm of figure 1.

For $t := (N-1)\Delta t$ *to* 0
  For $c \in$ admissible stock levels
    For $s \in$ all possible price levels
      $\tilde{J}^*(s,c) := \max(-(a_{in}s + K_{in})\Delta t + \mathbb{E}(J^*(S_{t+\Delta t}, c + a_{in}\Delta t)|S_t = s), (a_{out}s - K_{out})\Delta t + \mathbb{E}(J^*(S_{t+\Delta t}, c - a_{out}\Delta t)|S_t = s), -K_s\Delta t + \mathbb{E}(J^*(S_{t+\Delta t}, c)|S_t = s))$
  $J^* := \tilde{J}^*$ *//Set J^* for the next time step*

Fig. 1. Theoretical stochastic control algorithm

This algorithm is a generic one: the price model only appears in the conditional expectation. The time loop (t variable) is inherently sequential, unlike the stock level loop (c variable) which can be efficiently parallelized. However, some complex data exchange will be mandatory at the end of each time step (see section IV).

III. Development tools

The original and sequential application was composed of a Python main program calling some efficient C++ computing functions. This programming strategy has some advantages for researchers and engineers in financial computing. C++ is used to implement the computing library very efficiently,

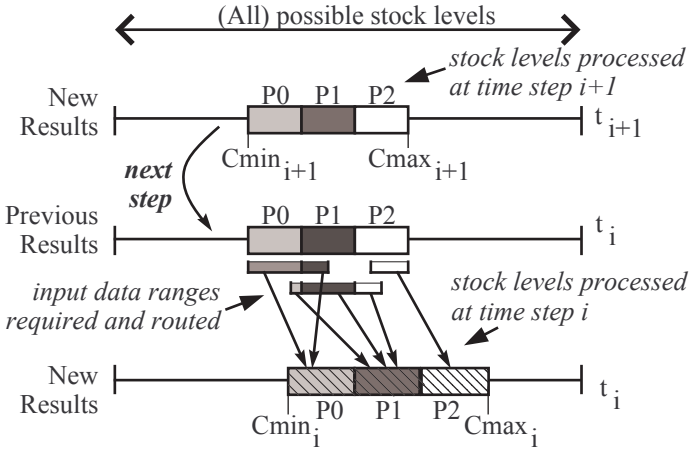

Fig. 2. Example of optimized data distribution on three processors

while Python is used to implement non intensive computing parts of the application, and allows to develop very quickly various main programs. In order to develop a distributed application adapted to *users*, we have decided to maintain this strategy. Therefore, we have designed a distributed application composed of sequential computing routines implemented in C++, a C-MPI library, and a toplevel Python program to deploy processes on the different processors, call the sequential computation routines and manage data exchange between processors. We have chosen the *Pypar-1.9.2* MPI interface module for Python [8], which is an interface to well-known and efficient C-MPI libraries (such as *mpich-1.2.7*), instead of full MPI re-implementations in Python. In the process, we have improved this Python/C-MPI interface in order to use a *buffered* message send routine (`MPI_Bsend`) instead of a basic send routine (`MPI_Send`), and to support modern *NumPy* arrays in MPI routines.

The implementation of our stochastic algorithms involves many large *arrays*. They are implemented in the Python code with arrays of the *Numpy-1.0.1* package, and in the C++ code with arrays of the *blitz-0.9* library. The latter allows to easily allocate, resize and manipulate large arrays. Finally, the interface between Python and C++ is achieved by the convenient *Boost.Python-1.33.1* library.

IV. Optimized distribution

A. Parallelization strategy

To achieve large speed-up and size-up we have decided to parallelize the stochastic control algorithm of figure 1 on scalable distributed architectures, such as PC clusters and distributed memory supercomputers. Temporal steps of the external loop have to be run sequentially, but computations of the second loop on stock levels can be run concurrently. So, we have split the stock level loop on a set of processors communicating by message passing. However, the range of stock levels to process changes at each temporal step, leading to redistribute computations and data at each step.

As illustrated on figure 2, stock level loop (from $Cmin_{i+1}$ to $Cmax_{i+1}$) is load balanced on processors at step $i+1$, and each processor stores its results. At step i (the next step), the new stock level loop (from $Cmin_i$ to $Cmax_i$) is load balanced on processors, and each processor requires a specific range of the previous results as input data. Hence, each processor computes the input data range required to process each stock level and deduces the input data range required by each processor to process its new range of stock level. Then, each processor establishes its routing plan: it points out the range of its previous results to send to each other processor, and the range of previous results to receive from each other processor. Finally, each processor executes its routing plan according to a predetermined message passing scheme. In order not to require too large temporary message buffers on processors, and not to overload the cluster interconnection network, the routing plan is split in $(P-1)/2$ steps. Processors are considered on a virtual ring, and at step i each processor exchanges data only with its two neighbors at distance i (see figure 3). This routing scheme could need improvement for the custom interconnection network of a supercomputer, but its low memory and controlled bandwidth consumption makes it suitable for a basic PC cluster with classic gigabit Ethernet links and switches.

In order to store some new input data tables with different sizes at each step, some data tables are allocated and freed at each step on each processor. This memory management strategy introduces some small overhead, but in exchange minimizes the amount of memory used and therefore allows larger problems, requiring more processors, to be treated.

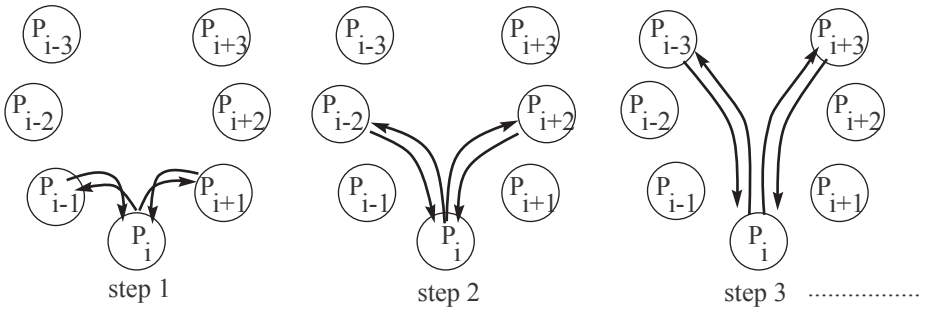

Fig. 3. Routing scheme adopted on PC cluster

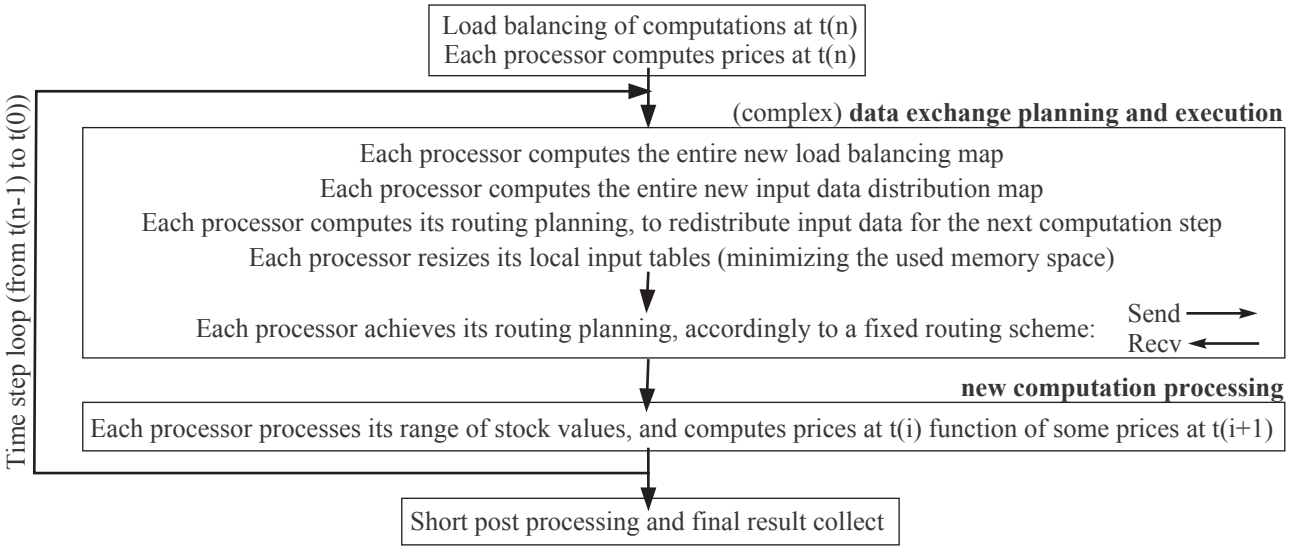

Fig. 4. Distributed algorithm for stochastic control

B. Distributed algorithm design and implementation

According to the strategy introduced in the previous subsection, we have designed the distributed algorithm depicted in figure 4. The first step consists in load balancing the computations of the prices at t_n by calling specific initialization routines. Then the algorithm enters a loop of n steps (from t_{n-1} to t_0) which encompasses two main sub-steps: *data exchange planning and execution*, and *new computation processing*.

The *data exchange planning and execution* sub-step starts computing the new load balancing map and the new input data distribution map on each processor. These computations are simple, and are faster to compute entirely on each processor than to distribute and parallelize. Then each processor builds its routing plan and resizes its local data tables, according to the new input data distribution map. The data exchanges are achieved just after these preliminary operations, and are based on point-to-point communications. Our current Python-MPI (*Pypar*) development tool does not support asynchronous communications, and is limited to the basic and implementation dependent `MPI_Send` routine. We have improved *Pypar* by adding support for the `MPI_Bsend` routine. This solution takes time to copy data into buffers, but allows to control the size and the allocation of the communication buffers. This allows us to control the total amount of memory required by our implementation and ensure that memory will not hinder the scalability of the application. The execution of the routing plan ends the *data exchange planning and execution* sub-step. The different `send` and `receive` operations are scheduled to use small and limited communication buffers and not to overload standard cluster commu-

nication networks (see previous subsection on parallelization strategy).

The *new computation processing* sub-step is illustrated at the bottom of figure 4. It consists in pure local and efficient computation on each processor, according to the stochastic control algorithm of figure 1, and to a fixed price model (see section I).

V. Performance measurements

A. Test applications

A gas storage owner wants to valuate his utility which has a capacity of 100,000 MWh for a use during two years. The injection and withdrawal rates are highly dependent on the stock level and have values ranging between 100 and 1,000 MWh per day:

- When the stock level in the cavity is high, the pressure is high too and makes injections more difficult than withdrawals.
- Conversely, when the stock level is low, injections are easier than withdrawals.

The storage is valuated for a use beginning in one year and finishing two years later. The initial stock level is 20,000 MWh and the final value of the gas storage in three years is set to 0 for simplicity. The annual interest rate of 8% that we used for the valuation is the one provided by Zeebruge at the beginning of year 2006 (the Stock Exchange of Zeebruge fixes gas prices). The discretization step of the gas storage is set to 500 MWh.

The three stochastic price models are characterized by a daily short-term volatility equal to 0.014 associated to a daily short-term mean-reverting value of 0.0022 that totally define the first Gaussian model. The two-factor model needs two additional parameters to be defined: the daily long-term volatility set to 0.004, and the daily long-term mean-reverting set to 0.01. As for the Normal Inverse Gaussian model it also needs two more parameters: the first one α is set to 0.5 and is related to the kurtosis of the distribution while the second one β is set to 0 and is associated to the asymmetry of the distribution.

The time step used for the three methods is 0.125 day. Such a refined time step is not necessary for the valuation itself, but it is to calculate the optimal command that could be used by a Monte Carlo simulator in order to get for example the cash distribution generated each month during the two years. The Normal Inverse Gaussian model also needs a step for the *space* discretization. This step was set to 0.0125. The renting price of our fictive gas storage space is calculated in euros for a two-year period. Our three models yield respectively 1,355,010 and 1,358,930 and 1,354,630.

In this gas storage it can be noticed that the prices obtained by the three models are nearly identical. Nevertheless, other tests have to be carried out in order to determine whether the sophisticated model can outperform the one factor Gaussian model. As it is shown by the performance results in the following section, such an investigation is made possible by our distributed implementation.

TABLE I
Execution times (in seconds) of test applications

#P	1	2	4	8	16	32
G-1fac.	911	475	262	164	107	90
G-2fac.	-	-	-	-	50057	25914
NIG	49403	24762	12428	6381	3475	1919

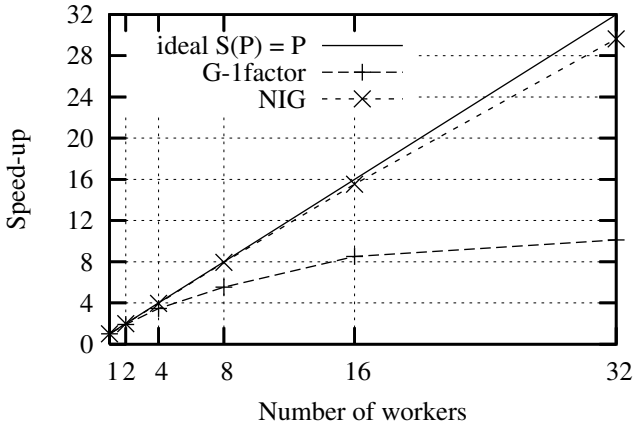

Fig. 5. Speed-ups achieved on a 32 PC cluster

B. Experimental performances

The experiment testbed was a classic 32 PC cluster with Pentium-IV processors at 3 GHz, 2 GB of main memory and a Gigabit Ethernet interconnection network. No other application was running during the benchmarks.

Table I and figure 5 show the execution times and speed-ups achieved for the three benchmarks with our implementation based on Python-C++ and a subset of MPI functionalities included in *Pypar*. The first benchmark ("G-1fac.") is associated to the one-factor Gaussian price model and is classically used for gas storage valuation. It includes frequent communications and small amount of computations during each temporal step. Thus, it is not surprising that its speed-up is limited to 10 on 32 processors. However, this model is used as a reference model to evaluate re-

sults of more complex models, and is frequently run for long-term simulations. Thus, this speed-up of 10 remains very attractive.

The third benchmark ("NIG"), which is associated to the normal inverse Gaussian model, involves long computations between communication steps. It achieves a speed-up close to 30 on 32 processors, with an efficiency greater than 92%, and an execution time decreased from 13h43 on a single processor to 0h28 on 32 processors. Obviously, such improvement will facilitate the study of this complex model.

The second benchmark ("G-2fac.") is associated to the two-factor Gaussian model and is an experiment of another complex model requiring both a lot of CPU and a lot of memory. The latter requirement is such that the memory of more than eight nodes of our cluster is necessary to run the benchmark without swapping. Thanks to our data distribution the computation has been successfully achieved. Execution times obtained when using 16 and 32 machines have been reported in table I. Despite the excessive long duration which is displayed, the execution time is divided by a factor of 1.93 when switching from 16 to 32 processors. This benchmark illustrates both size-up and speed-up abilities of our implementation.

VI. Conclusion and perspective

We have designed and implemented a stochastic dynamic programming algorithm for gas storage valuation. This distributed algorithm supports different price models of gas storage spaces and facilities, and includes an optimized dynamic data distribution and memory management in order to achieve both speed-up and size-up. The distributed implementation introduced in this paper is based on a MPI-Python interface (*Pypar*), and on C++ computing routines. It achieves very good speed-ups and high efficiency (greater than 92%) on a complex valuation model run on our standard 32 PC cluster. Moreover, our implementation allows to run and speed up another complex model requiring the memory of more than 8 PCs to install all its datastructures.

Finally, the speed-up curve of the Normal Inverse Gaussian model on figure 5 is close to the ideal speed-up, and can continue to increase on larger clusters. This parallelization allows fast investigations of complex and original valuation models for gas storage spaces and facilities, and will help to design better models of gas storage valuation.

In order to track better performances, a pure MPI-C++ version is under development. The current implementation uses the non-blocking `MPI_Issend` communication routine which does not require any additional buffers, and has already achieved a slight improvement on 32 processors with our least favorable benchmark ("G-1fact."). Memory consumption has been reduced while speed-up has increased from 10 to 11.5. This optimized MPI-C++ version will be tested using complex price models on larger clusters and on an IBM BlueGene/L supercomputer (up to 8000 processors) in a near future. The next step will consist in improving the software architecture in order to be able to *plug* and experiment new price models very easily.

Acknowledgment

This research is part of the ANR-CICG GCPMF project, and is supported both by ANR (French National Research Agency) and by Region Lorraine.

References

[1] Gobet E. Barrera-Esteve C., Bergeret F. and al, "Numerical methods for the pricing of Swing options: a stochastic approach," *Methodology and Computing in Applied Probability*, 2006.

[2] Carmona R. Ludkovski M., "Gas storage and supply: An optimal switching approach," 2005, submited, `http://www-personal.umich.edu/~mludkov/papers.html`.

[3] Forsyth P.A. Chen Z., "A semi-lagrangian approach for natural gas storage valuation and optimal operation," Tech. Rep., 2006.

[4] Tompaidis S. Jaillet P., Ronn E., "Valuation of commodity-based swing options," *Management science*, vol. 50, 2004.

[5] Barndorff-Nielsen O. E., "Processes of Normal Inverse Gaussian Type," *Finance and stochastics*, vol. 2, 1998.

[6] L. Henrio, V. Dung Doan, M. Bossy, F. Baude, S. Vialle, V. Galtier, and S. Bezzine, "A fault tolerant and multi-paradigm grid architecture for time constrained problems. application to option pricing in finance.," in *e-Science 2006*, Amsterdam, Netherlands, dec 2006, IEEE CS Press.

[7] A. V. Gerbessiotis, "Trinomial-tree based parallel option price valuations," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 18, no. 4, 2003.

[8] O. Nielsen, *Building a parallel program step-by-step*, School of Mathematical Sciences, Australian National University, Canberra, Australia, April 2002, `http://datamining.anu.edu.au/~ole/publications/parallel_python.pdf`.