Parallel Discrete-Event Simulation of Wind-Energy Electricity Generation Systems

Mauricio Marín Eduardo Peña
Computing Department, University of Magallanes, Chile
E-mail: {mmarin,edopena}@ona.fi.umag.cl

Abstract

Wind can be an effective energy source to generate electricity at various places of the planet. The complexity associated with the analysis of control strategies for medium and large size generation systems which combine wind and diesel electricity generators along with batteries, requires one to consider computational techniques. In this paper we describe an unusual computational technique for the analysis of these systems: Parallel discrete-event simulation which to the best of our knowledge has not been so far utilized in this kind of studies. Despite of the discretisation introduced by this technique, we have observed that it can be a quite powerful analytical tool, yet the reward is to allow the simulation of many hours of simulated time at reasonable running-times.

1 Introduction

Electricity generation by means of wind energy has been shown to be a useful technology for farms or villages located at remote places. Their electricity needs can be served by combinations of wind and diesel generators which supply energy to batteries. Electrical loads take their energy from the batteries which are continuously reloaded by the generators. Wind generators are the first choice for reloading batteries whereas diesel generators are only used when the wind is not strong enough. Sudden variations in the wind/load intensity can lead the system operation to extreme cases such as peaks in the energy generation/consumption. Proper tuning, then, is achieved by using control strategies whose correctness and effectiveness can be tested using simulation models. We assume that memory and sequential running time requirements of simulation programs are large enough to consider the use of parallelism.

In this paper we describe how to apply conservative and optimistic synchronisation protocols to these kind of simulation studies. These protocols are the core of parallel simulators as they synchronise the chronological occurrence of events over the processors. A unique feature of the proposed protocols is that they are devised to run on top of an architecture independent model of parallel computing called BSP [9]. BSP has the advantage of being general purpose, portable, efficient and scalable. We took advantage of this fact by implementing parallel simulators which can run efficiently and without changes to the codes on diverse parallel computers.

2 Simulation model

Each entity of the system is simulated by a *logical process* (LP) which causes the occurrence of events associated with it. These include wind and diesel generators, batteries, converters and inverters, and several types of electrical loads. Figure 1 presents an overview of these devises. Also, over the scheme, there are control algorithms driving the operation of the system under diverse scenarios. The actual implementation of the LPs is by means of C++ objects. The LPs schedule events in other LPs by sending messages to them and the evolution of the system dictates the communication patterns among the LPs.

Wind behaviour is modeled as an stochastic input to wind generators, i.e., rate of change in its velocity and other mechanical parameters (air density, rotor area) with variations having arrival times exponentially distributed. Events are used to indicate these variations in the wind intensity. Upon each event, the involved LP sends a vector to the respective LP (wind-generator) which indicate the "shape" of the curve describing the wind intensity until the next event of this kind. This allows the LP simulating the wind-generator to include finer details about its operation.

Depending on the amount of electrical energy available in certain periods of a day, the control algorithms must decide on a set of actions to be effected. All the electrical loads take their energy from a set of batteries. Generators are in charge of keeping enough energy stored in the batteries. In case of very low intensity of wind, the control algorithms must switch on diesel generators to feed up batteries. In case of fully loaded batteries or sudden peaks in the wind intensity the controller must switch off wind/diesel generators.

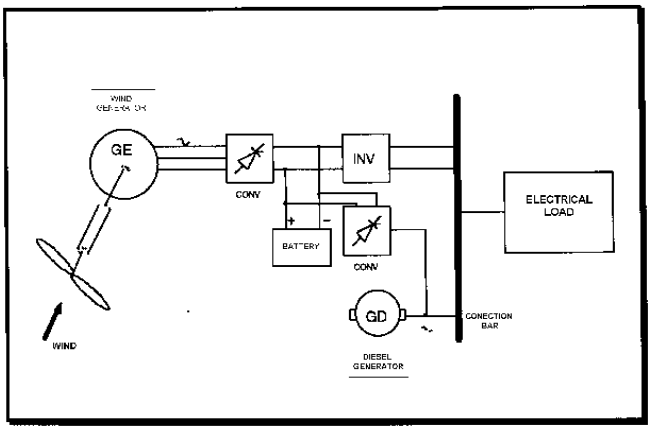

Figure 1: A very small system example.

The strategy adopted by the simulator is to signal a given situation with the occurrence of a particular event in the involved LP and whose result is a message being sent to the LP simulating the controller. Then a sequence of event-messages are sent to the devices in order to let them know of the action triggered by the controller. These events have the feature of having time increment equal to zero or close to zero if some delay is included in the system model.

Each house has different types of electrical loads. Lamps are considered as devices with the first priority for receiving the available energy. Other usual house loads are the first candidates to be disconnected from the system in case of low levels of energy coming from the batteries. Also there are special purpose loads which are connected on demand in case of peaks of energy from the batteries. Thus each house has its own local controller. The energy requirements for the electrical loads are modelled as stochastic processes whose behaviour depends on the time of the day in which the simulation is in or the type of scenario being simulated (e.g., faults and energy peaks).

For each generators-batteries group there is a controller which coordinates the operation of these devices considering both its interaction with its own set of loads and its interaction with other groups. Batteries receive energy from generators. Events (messages) are used to communicate the start/end of energy transfers. The load/unload process of the batteries is performed in accordance with time-curves for the particular devices. Events are also used to communicate variations in the energy requirements from the electrical loads connected to the batteries. Since the approach employed is event-driven there exists a necessary discretisation of the continuous processes. To improve approximations, particular events are scheduled to indicate the instant in which threshold values are under/overcome. The time for these events is predicted in advance following the expected behaviour of the device since previous events (extrapolation).

Note that at the start of the simulation each device builds-up vectors which describe approximately its transient behaviour upon events such as "switch on". These vectors are packed into event-messages so that message traffic between LPs can be reduced significantly. The whole simulation is driven by global time barriers so that the occurrence of erroneous events is minimised (optimistic simulation) or avoided completely (conservative simulation).

Overall, the strategy used to simulate this kind of system consists of decomposing it into a set of concurrent objects (LPs) which receive and send messages. The messages represent events that must take place at particular instants of the simulation time. There exists a simulation kernel which is responsible for administering the events so that they take place in chronological simulation time order in each object. Parallelism arises when the objects are placed on different processors.

There are several types of objects. Those which represent dependent entities such as diesel generators or controlled loads are mainly guided by their respective controllers. These objects normally will receive a message from the controller telling them what operation and at what simulated time it must be performed. Objects such as wind-generators will receive messages from objects simulating wind behaviour. Upon reception of one of these messages, the object will have to modify its output energy and at the same simulated time to forward messages to the objects acting as controller and battery respectively. This implies a sequence of messages containing events with time increments equal to or close to zero. This sequence of events with identical (similar) times has important effects in the synchronisation of parallel events.

Batteries are also dependent objects which receive messages from objects representing generators, houses and controllers. Batteries send messages to their controllers and houses in order to keep them with updated information about their (un)loading process. Many external messages are followed by a sequence of messages with identical or similar event times. Instead, objects modelling the behaviour of different instances of wind and lamps are independent as they send messages to other objects and do not receive messages from other objects (except cases such as critical conditions for power delivery to lamps).

Some of the results obtained with these discrete-event simulations are the following. For example, the wind behaviour is presented in figure 2. However, not all this wind energy is utilised as this depends on the level of energy in the batteries. Figure 3 shows the peaks of power delivery provided by a small wind generator used in the model.

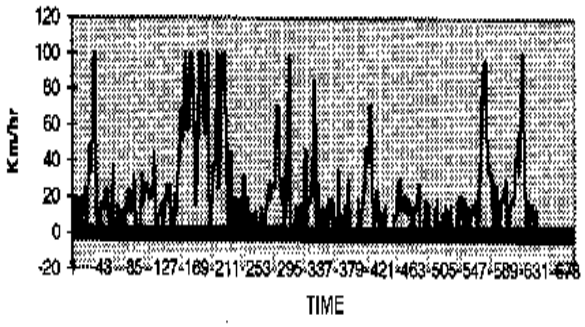

Figure 2: Wind behaviour.

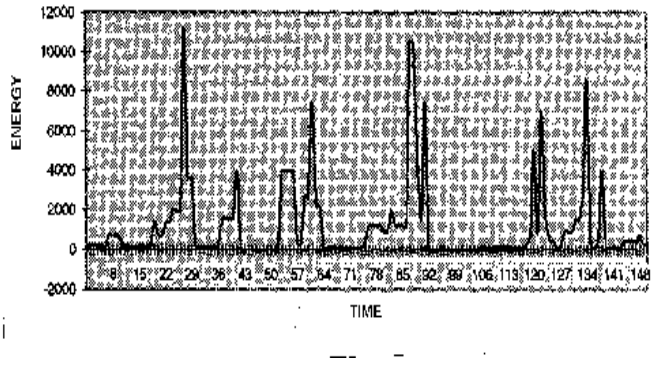

Figure 3: Wind generator.

3 Computational model and parallel simulation

We implemented our simulations over a model of parallel computing called BSP [9]. In BSP, portability is achieved by organising the computation as a sequence of *supersteps*. In each superstep the processors perform computations on local data. At the end of a superstep all the processors are barrier synchronised and all messages are sent to their destinations. That is, messages are ensured to be available for processing at the start of the following superstep. Each parallel computer is characterised by its suitability to support computation, communication and synchronisation efficiently. The practical realisation of the model comes as a library called BSPlib [7, 8]. BSP programs are written in C and C++ languages.

We employed the conservative and optimistic approaches to parallel discrete-event simulation. In the conservative view of the synchronisation problem the events are allowed to take place in their respective LPs only when there is certainty that no other earlier events will take place in the same LPs. We devised a BSP realisation of the Chandy-Misra protocol with null messages [1]. As BSP computations have a well defined structure, we introduced a number of optimizations such as reduction of the null-message traffic and use of local time windows associated with each LP. Experimental studies reported in [3] show that this method outperforms all others. However, these methods fail to achieve good performance under certain instances of wind systems (high rate of events with identical time occurring in different processors and no use of "pre-sending" as explained below).

We also resorted to optimistic simulation which adopts the view of letting events take place when they are available for processing in the LPs and executing a correction procedure in case of missing the simulation of earlier events. Such correction consists of performing a "roll-back" into the simulation past and re-starting the simulation of the involved LPs from that point onwards. In addition, erroneous event messages sent to other LPs must be cancelled which can cause the roll-back of these LPs as well. This implies the periodic saving of object states in order to selectively bring back to the "past" one or more objects during the simulation. For this case we use a protocol described in [4, 3].

3.1 Implementation details

Following the BSP approach to parallel computing, the simulation is organised in cycles (supersteps) composed of three basic operations: (i) receiving new messages at the beginning of the superstep, (ii) processing available messages (events) which satisfy certain causality restrictions, and (iii) buffering messages to be sent to other processors at the end of the superstep. The parallel simulation kernel is in charge of routing the events messages among LPs (objects). As this kind of simulation has the feature of being composed by sequences of events with identical time, the LPs can refuse to accept events when they "know" that new events with identical time are expected to arrive (possiblely from a controller located in another processor) by the next superstep. This takes the form of a locking operation which is kept until the next superstep.

Another tool to improve performance is the so-called "lookahead". This concept can be explained through an example. Let us consider an object simulating the wind behaviour for a particular wind-generator. Every time that an event takes place in this object, a new event is scheduled with a random time increment. At this point, the object could calculate in advance the new value for the wind intensity (in a sequential simulator this value would be calculated when the new event actually takes place). As we know that the new event cannot be cancelled, we can

send the message (along with the new wind intensity) to the respective wind-generator and controller right in the current superstep. This tends to increase parallelism since these events can take place in different processors during the same superstep (recall that messages take one superstep to travel from one processor to another). This is called "pre-sending" [6]. Another way to improve lookahead is to let the objects use specific knowledge about the operation of the system. For example, if it is known that a controller (or some other device) takes itself some time to answer to some event sent by the object, then this object can safely assume that it will not receive a message from the other object until such delay has occurred.

3.1.1 Conservative simulation

In the Chandy-Misra-Bryant (CMB) protocol [5] each LP has a set of input message channels and a set of output message channels wherein messages are received/sent from/to other LPs. These channels are defined in accordance with the communication topology of the LPs. If an LPs wants to process the occurrence of an event, it selects the one with the least time from its input channels. However, if there exists one or more empty input channels, the LP must block itself until new messages are available in these channels. On the other hand, each LPs must send messages through its output channels in strict chronological event time order. Blocking frequency can be minimised if LPs are able to send "null" messages through their output channels indicating lower bounds for the time of the next message to be put in the respective channels.

In BSP, each processor can be seen as a sequential simulator and thereby the burden of performing message passing between co-resident LPs can be avoided. In this case, the input message queues associated with the input channels can be merged into a single event-list implemented as a sorted linked list. Output queues are omitted since LPs can insert events directly into the event-lists of their co-resident LPs. Event-messages between LPs located in different processors are sent in bulk at the end of the current superstep, and the insertion of these events in their respective event-lists is effected by the start of the next superstep. On the other hand, actual null messages are only sent between LPs which are located in different processors. It suffices to send just one null-message between these LPs at the end of each superstep; the one with the largest time (per LP pair). In addition, the sending of a null message can be avoided if its timestamp is not greater than the timestamp of the previous null-message sent through the same channel. The combination of both schemes reduces significantly the null message traffic among processors.

For each input channel we maintain a variable which registers the time of the last *null* message received in the input channel. Let us call them *input variables*. The minimum of these (four) input variable values defines a local time barrier for the LP during the current superstep. Co-resident LPs directly update the respective input variables, and the message reception routine performs similar operations as it receives null messages at the start of each superstep. Null-messages are removed from the system once they are processed by the message reception routine.

In every superstep, all the co-resident LPs are scheduled for event processing in an arbitrary manner (i.e., with no particular timestamp order). Then for each LP, the algorithm (i) calculates its local time barrier, (ii) simulates events with times less than the barrier, and (iii) "sends" one null message through each output channel (i.e., direct updating of input variables of co-resident LPs or buffering of null messages provided that their timestamps are greater than the previous ones). The algorithm performs this operation *once* for each LP in each superstep.

3.1.2 Optimistic simulation

Our BSP Time Warp system [2] works as follows. Like any BSP computation, the simulation is carried out by executing supersteps. In each superstep, each processor simulates the occurrence of events in accordance with messages received at the start of the current and/or previous superstep(s). This simulation is effected sequentially by using a processor event-list. In addition, for each processor, there is an (adaptive) upper limit to the number of events allowed to take place in each superstep. Moreover, newly-generated and rolled-back events are treated identically: they are all kept in their respective processor event-lists so that they are selected for processing in chronological timestamp order. This means that (i) upon rolling back a LP, all the events that become "unprocessed" are re-inserted in the processor event-list, and (ii) these events might not take place in the same superstep of the roll-back.

In a superstep, straggler events arriving from other processors may cause the insertion of rolled-back events into the event-list of a given processor. However, the upper limit to events per superstep remains the same regardless of the number of rolled-back events in the processor. The net effect of these stragglers is thus the actual reduction of the speed of simulation time advance of the processor since it now has to cause the chronological occurrence of a comparatively larger number of rolled-back events per superstep. This in turn leads to its relative synchronisation in simulation time: any processor executing events too far in time is expected to receive a significant amount of stragglers in the following supersteps which will bring this processor back to synchronisation with the other processors. This is so because our scheme tends

to emulate a global event-list which gives priority to the processing of the least timestamped events in the entire system. Rollback thrashing is avoided in this way.

The scheme is completed with an adaptive method for determining the values of these upper limits. These limits can be also expressed in terms of simulation time barriers. The method is automatic, and it has the unique feature of using information from the sequential simulation of an optimal BSP synchronisation protocol to adaptively tune the operation of the Time Warp engine. This sequential simulation is performed by the Time Warp system itself during execution.

3.2 Experimental results

Table 1 shows speed-up values for the conservative and optimistic BSP protocols decribed above. Three sizes of systems where investigated while the number of processors was kept constant. In addition, we investigated situations with small lookahead (pre-sending and zero delay for answers from the controllers) and large lookahead (pre-sending and relatively large delay for answers from controller and other devices).

The results shows reasonable speed-ups, and confirm the well-known fact that the efficiency of conservative protocols is highly dependent on the amount of lookahead.

Small lookahead

Protocol	System size: Small	Medium	Large
Conservative	1.2	1.3	1.5
Optimistic	1.6	1.8	2.3

Large lookahead

Protocol	System size: Small	Medium	Large
Conservative	2.5	3.2	3.7
Optimistic	2.1	2.4	2.8

Table 1: Results for a 4-processors SGI PowerChallenge computer.

4 Final comments

In this paper we have described the main ideas behind the efficient realisation of parallel discrete-event simulation of wind-energy electricity generation systems. A few experiments were conducted and the results showed that (i) this is a feasible technique for analyses such as correctness of control mechanisms, (ii) this technique, as apposed to time-driven approaches, allows one to perform long term simulations at reasonable running-times, and (iii) it is possible to achieve reasonable speed-ups on parallel computers. The object oriented approach used to model the system simplified noticeably the software development and synchronisation of parallel events as it maps naturally to the concept of synchronisation based on logical processes.

More research remains to be done in order to explore the feasibility of combining the approach proposed in this paper with numerical simulation (time-driven based on differential equations). In this case, the numerical model would evolve until the point marked by the time barriers for the current supersteps. This would allow one to model various continuous processes within devices such as generators. In terms of speed-ups, this combination would improve performance as the grain of the computation performed in each superstep would be increased which in turn would better amortise the costs of communication and barrier synchronisation of processors.

Acknowledgement

This work has been partially funded by Fondecyt project 1990627.

References

[1] R.M. Fujimoto. "Parallel discrete event simulation". *Comm. ACM*, 33(10):30–53, Oct. 1990.

[2] D.R. Jefferson. "Virtual Time". *ACM Trans. Prog. Lang. and Syst.*, 7(3):404–425, July 1985.

[3] M. Marín. "Discrete-event simulation on the bulk-synchronous parallel model". PhD Thesis, Programming Research Group, Computing Laboratory, Oxford University, 1998.

[4] M. Marín. "Time Warp On BSP Computers". In *12th SCS European Simulation Multiconference*, June 1998.

[5] J. Misra. "Distributed discrete-event simulation". *Computing Surveys*, 18(1):39–65, March 1986.

[6] D.M. Nicol. "The cost of conservative synchronization in parallel discrete event simulations". *Journal of the ACM*, 40(2):304–333, April 1993.

[7] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. "Questions and answers about BSP". Technical Report PRG-TR-15-96, Computing Laboratory, Oxford University, 1996. Also in *Journal of Scientific Programming*, V.6 N.3, 1997.

[8] BSP Worldwide Standard. http://www.bsp-worldwide.org/.

[9] L.G. Valiant. "A bridging model for parallel computation". *Comm. ACM*, 33:103–111, Aug. 1990.