Towards Petri net-based Economical Analysis for Streaming Applications Executed over Cloud infrastructures

Rafael Tolosana-Calasanz[1], José Ángel Bañares[1], and José-Manuel Colom[1]

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain
{rafaelt,banares,jm}@unizar.es

Abstract. Streaming Applications are complex systems where the existence of concurrency, transmission of data and sharing of resources are essential characteristics. When these applications are run over Cloud infrastructures, the execution may incur an economical cost, and it can be therefore important to conduct an analysis prior to any execution. Such an analysis can explore how economic cost is interrelated to performance and functionality. In this paper, a methodology for the construction of this kind of applications is proposed based on the intensive use of formal models. Petri Nets are the formalism considered here for capturing the active entities of the system (processes), the flow of data between the processes and the shared resources for which they are competing. For the construction of a model aimed at studying different aspects of the system and for decision-taking design, an abstraction process of the system at different levels of detail is needed. This leads to several system models representing facets from the functional level to the operational level. Petri Net models are used to obtain qualitative information of the streaming application, but their enrichment with time and cost information provides with analysis on performance and economic behaviours under different scenarios.

Keywords: Economical Cost of Clouds, Petri Nets, Streaming Application.

1 Introduction

Over the last years, with the advances and development in sensor networks & technologies, there has been a growing number of sensors that are monitoring physical or environmental conditions, such as temperature, humidity, wind, etc. These sensors transmit their measurements continuously, forming a sequence of data elements known as a *data stream*. A number of applications exploit these data streams with different purposes –ranging from military applications monitoring battlefield to industrial and consumer applications, such as applications that help create Smart Cities and Smart Buildings, or applications for health care monitoring, and natural disaster prevention. Moreover, these applications

often have to deal with significants amounts of data that need to be processed *continuously* in (near) real-time, leading to the need for distributed computational infrastructures.

Hence, in such a context, the complexity of streaming applications arises from the confluence of concurrency, transmission of data and the use of distributed resources. From a conceptual design perspective, a streaming application can be seen a set of *Computational Processes (CPs)* that receive data continuously, and the data dependencies among them. Besides, a CP may require the synchronization of several input streams to conduct the computation. In order to execute such applications, CPs and their data transmissions involved require resources, including computational machines, network and storage. In the activity of mapping, resource management is involved, and its effectiveness can be measured from various system properties [15], namely economical cost, performance, and functionality. Nevertheless, depending on the target infrastructure the policies and the mechanisms involved for such an activity may be different. For instance, unlike in traditional distributed systems, load balancing in a Cloud infrastructure is not about evenly distributed the load among servers, but in minimizing the cost of providing the service. Hence, the goal is to adapt the computational power to the actual workload. It would be therefore important to conduct an analysis prior to any execution of a given workflow, analysis that must explore minimal and maximal boundaries of the economical cost of the execution of a streaming specification, in relationship with its performance and its workload.

In this paper, we propose a Petri net-based, model-driven and stepwise refinement methodology for streaming applications over Clouds. The central role in this methodology is assigned to a set of Petri Net models describing the behavior of the system including timing and cost information. The goal is to use these models in an intensive way before the deployment of the application in order to understand the system, and to obtain properties of the different solutions adopted. In some cases, the observations may induce or recommend changes into the application with the purpose of modifying parts of the design and assure agreed specifications. The consideration of Petri Nets is based on the natural descriptive power for the concurrency, but also for the availability of analytic tools coming from the domain of Mathematical Programming and Graph Theory. These tools are based on a structural analysis that support the reasoning on properties without the construction of the state space –which for such a class of systems is prohibitive.

Additionally, Petri Nets support the use of formal verification techniques such as standard Model Checking Techniques for the verification of qualitative properties. These techniques can be based on the construction of the state space of a streaming application (the complete set of reachable markings of the net by the occurrence of transitions). Then, any property to be verified can be expressed in logic terms. In case the property is satisfied, the answer of the Model Checker is just a confirmation of this, but if the property is false, then the model checker gives counterexamples that proof that the property does not hold. The main advantage is that usual properties like deadlock-freeness, liveness, home space,

maximal sets of concurrently fireable transitions, mutual exclusions, etc. can be decided. In practice, the applicability of the approach is limited to *bounded* systems with a finite state space and with a moderate size. Conclusions hold *only for the initial marking* being considered. Our models for streaming application can be exploited to understand the most appropriate approach for solving the problem: strategy(ies) for decomposing the problem into processes, communication needs and resources required for satisfying functional and non-functional requirements. Taking into account that the model is a Petri Net, we can consider it as an executable specification. Therefore, it can be simulated in order to reach this understanding or simply to obtain performance or economic boundaries, which can be obtained by enriching it with time and cost information. In other words, simulation is an analysis technique. It may be useful to discover some (un)desirable behaviors, but in general it does not allow to proof the (in)existence of some properties.

The other way to exploit the models is through the extraction of structural information from the net. Structural information allows to obtain properties that are guaranteed mainly for the net structure, i.e. the places, the transitions and the token flow relation represented by means of the arcs. On this regard, a classical example is the information obtained from the marking invariants guaranteed by the structure of net (weighted sums of the contents of tokens of some places of the net that remain constant for any reachable state and, in particular, for the initial marking of the net). Invariants or other structural objects of the net give valuable information on the net behavior that can be used to detect, for example, a sequential execution of activities that reduces the net concurrency –i.e. because there are not enough tokens to fire transitions simultaneously. Let us suppose, for example, that we have a marking invariant that says that the sum of the contents of tokens of a set of places is equal to 1 for all reachable states. This means that in the set of places contained in the invariant one and only one of the places can contain exactly one token. Therefore, all the places are in mutual exclusion, moreover, all output transitions of these places are in mutual exclusion (they cannot be never concurrently fired). With this information a designer can decide to increase the contents of tokens of these invariants to increase the degree of concurrency, or he/she can modify the structure of the net in order to remove all these limiting invariants for the concurrency. From the above comments, we can say that a first phase for exploiting the models is to determine *the correction of the adopted solution*, verifying all the good properties expected are satisfied. As a second step, when a property is not satisfied, the model can be used for *detecting the causes of the problems* or anomalies in it that prevent it to behave as expected. Then, the understanding of the causes can lead to a *modification of the model*.

In the case of this paper, Petri Nets must capture the active entities of the system CPs, the flow of data between the processes (Data Transmission Processes) and the shared resources. The final Petri Net models can be exploited as a universal specification to be used with multiple platforms and languages. Taking into account the complexity of the involved elements in the system, the

proposed approach decomposes the construction of the model in several stages. This leads to several system models representing facets from the functional level to the operational level. Petri Net models are used to obtain qualitative information of the streaming application, but their enrichment with time and cost information provides with analysis on performance and economic behaviours under different scenarios.

We validate our methodology through the wavefront algorithm, a Matrix-Vector multiplication in streaming fashion [13, 12]. First, we create a Funcional model describing the behavior of computations and data transmissions. After that we apply structural analysis techniques to verify that the functional behavior of our model was the expected: the wavefronts are propagated through the array in an orderly way. In order to verify some quantitative properties (the throughput of transitions) we add a timing interpretation to some transitions of the net, from which we can derive the economic cost. The reminding of this paper is structured as follows. In Section 2, related work is briefly discussed. In Section 3, our methodology is proposed. The Matrix-Vector multiplication example is studied in Section 4, and finally the conclusions are given in Section 5.

2 Related Work

Due to importance of processing data generated in a stream fashion, well-known workflow systems such as Kepler [19, 10], Triana [9], or JOpera [5] have already incorporated data streaming workflow patterns to be used by users in their workflow specifications. The idea is that a sequence of more than one task is applied sequentially to a data stream [20]. In general terms in pipelined workflows, performance is typically measured in terms of throughput, and therefore the throughput is conditioned by the slowest task. For such a reason, it is important that all the tasks execute in the same time, which is challenging due to the variability and heterogeneity of computational resources as well as the programs that execute the tasks. Thus, task merging and workflow transformation is essential prior to mapping the tasks onto distributed resources in order to achieve the minimal variation in execution time of tasks. We believe that our proposal in this paper can help analyze the influence of the different design decisions in the task-mapping process on workflow throughput and other properties such as economical cost. Other proposals like [23] utilised Petri Nets for predicting the execution time (makespan) of Taverna workflows at a functional level.

Regarding the problem of workflow mapping of tasks and task clustering has been studied deeply in the Pegasus workflow system [14] for workflow DAGs. In particular, in [8], the authors highlight the problems that arise with current task clustering techniques as they are based on over simplified workflow models. They investigate the causes and propose a number of task balancing methods to address these imbalance problems. To the best of our knowledge, there is no analogous work for streaming workflows. Our model can be useful for system analysis, but at this stage is intended for human assistance rather than for autonomic system

On the other hand, stream processing frameworks such as Yahoo's S4 [18], or IBM InfoSphere Streams [4] provide streaming programming abstractions to build and deploy tasks as distributed applications at scale for commodity clusters and clouds. Nevertheless, even that these systems support high input data rates, they do not consider variable input rates, which is our focus in this paper.

Additionally, the work in [21, 22, 1] is focused on designing a system architecture that processes data streams and exploits autonomic computing principles for resource management in an elastic infrastructure. It consists of a sequence of nodes, where each node has multiple data buffers and computational resources – whose numbers can be adjusted in an elastic way. They utilize the token bucket model for regulating, on a per stream basis.

In the Performance Engineering community, traditionally, three methods have been proposed, sometimes in complementary ways, to reveal how a system performs: direct measurement, simulation and analytical techniques. Although all of them allow system engineers to undertake testing before development, and at a lower cost, both simulation and analytical methods share the goal of creating a performance model of the system / device that accurately describes its load and duration of the activities involved. Performance models are often described in some formalisms including queuing network models [16], stochastic process algebra [3] or Petri nets [17] that provide the required analysis and simulation capabilities. A great number of these studies try to derive Petri net performance models from UML diagrams [2] and to compute performance metrics such as response time. Our emphasis in here is to exploit the inherent nature of Petri Nets for modelling concurrency, which serves

3 Model Construction for the Economical Analysis

In the next subsections, we introduce the basic principles for the modular construction of the Petri Net models for our proposal. After that, we introduce different interpretations in the models in order to analyze properties related to the qualitative behavior of the system, the performance of the application or the economical costs of the deployment and execution.

3.1 Modular construction of the functional Petri Net model of the streaming application

The construction of the functional model is based on the identification of the basic modules that compose an application of this class. These modules are the Computational Processes (CPs) and the Data Transmission Processes (DTPs). CPs accomplish functional operations and transformations on data, and DTPs allow data dependencies to be conducted among CPs. Both of CPs and DTPs need resources to accomplish the corresponding operation, and these resources also appear in the model, but at a conceptual, and generic way. Later in subsequent model refinements, specific resource constraints of different computational infrastructures will be added, such as limitations in parallelism, capacity, economic cost, etc.

- ***Characterization of a CP***. A CP can be viewed as a sequence of operations to be applied to a set of data elements coming from different input data streams, a type of elementary computational task of the application. We assume a CP consists of multiple instances, Computational Threads (CT), that are executed concurrently. A CP is modeled as a Petri Net, $\mathcal{N}$, and a CT as a token that moves through $\mathcal{N}$. The places (partial states) of $\mathcal{N}$ are related to the different operations (either transformations, handling or assembly/disassembly operations) to be carried out by the thread. There exists a special place named *Idle* representing the inactive state of the threads and its initial marking is the maximum number of supported threads executing simultaneously this CP. The transitions of $\mathcal{N}$ allow a CT to progress towards its final state representing the end of the computation for the input data elements, the production of the output data elements and the restarting the thread for the processing of the next data elements on the stream. A CP has distinguished input points (output transitions of the Idle place) of data elements from the input streams and output points (input transitions of the Idle place) of data elements of the output streams. The execution of a CP is achieved by the execution of a computation path, and several of them can exist in the same CP. A computation path is a sequence of transitions fireable in $\mathcal{N}$, whose occurrence represents the obtention of a computed data record.
  In Figure 1.a, a CP with two sequential states is represented. A token in the place *Operation1* or *Operation2* represents a thread executing the code corresponding to the operation 1 or 2, respectively, required by the computational task modeled by means of this CP. A thread executes these operations sequentially following the firing sequence: (1) *Input Data Stream* representing the acquisition of the data records from the input stream to realize the computational task; (2) *Change* representing the end of the operation 1 and the beginning of the operation 2; (3) *Output Data Stream* representing the delivery of the data records obtained after the computation to the output stream. The model presented in Figure 1.a is untimed. The addition of timed information to a CP is introduced by the addition of a sequence place-transition-place in parallel with a process place representing an operation of the computational task that consumes time. The transition added is labeled with time information representing the duration of the computational operation. In Figure 1.b, the CP from Figure 1.a is represented by assigning *time1* and *time2* units of time to the execution of the operations 1 and 2, respectively, according to the previously announced construction for the introduction of timing in the model. Observe that all transitions of Figure 1.a are immediate that is, do not consume time.
- ***Characterization of the Data Transmission Processes***. CP can transmit data elements to other CP in the form of a stream sent by means of a physical / virtual device such as a FIFO queue implemented in memory or a communication channel in a communication network. That transmission behavior is captured by a *Data Transmission Process* DTP. A data element to be transmitted is modeled as token that moves through a Petri Net, $\mathcal{N}$,

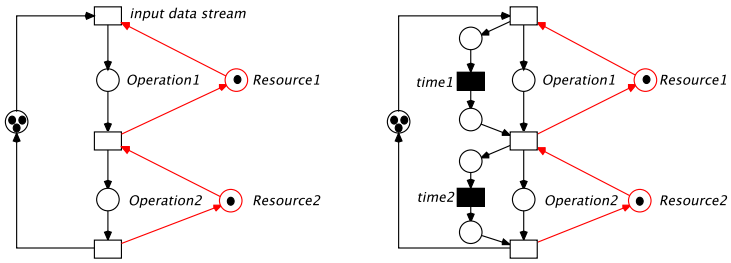

Fig. 1. A Computational Process with Resources composed by 2 sequential states each one requiring a different type of resoource. a) Untimed model; b) Timed model assigning $time_i$ units of time to the execution of the $operation_i$.

representing an Elementary DTP with capacity for a single data record. The places of $\mathcal{N}$ are related to the states in which a data element can be in the transmission device. The transitions of $\mathcal{N}$ allow a data record to progress from the source to the destination. The construction of a transmission device for a stream with a capacity for k data elements sequentially ordered requires of the concatenation of k of these elementary DTPs. The model in Figure 2.a is untimed and firing sequence of transitions *Begin Transmission* and *End Transmission* represents the movement of a single data element of a stream from the source (the final transition of any kind of Process) to a destination (the initial transition of any kind of Process). In Figure 2.b, the DTP of Figure 2.a is represented by assigning $time_1$ units of time to the transmission of a data element (according to the previously announced construction for the introduction of timing in the model).

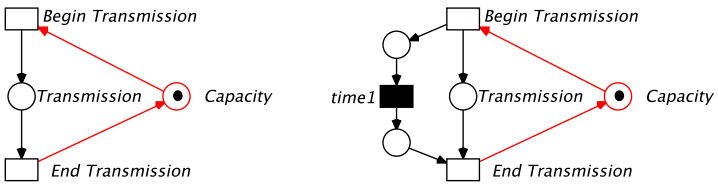

Fig. 2. A Data Transmission Process with capacity for a single data record. a) Untimed model; b) Timed model

– ***Incorporation of Resources to each Computational and Data Transmission Process***. We consider any hardware/software element part of the execution environment (i.e. a processor, a buffer, a server, a communication channel, etc.) as a resource with a given capacity. In the case of a buffer, its capacity can be the number of positions to allocate elements, a proces-

sor may have a number of cores that can be considered as its capacity, etc. Moreover, in the execution environment, there exist several resource types and for each of them, a number of identical instances can be available, representing either the number of available copies of the resource to be used (or its capacity). In all cases, the considered resources are conservative, i.e. there is no resource leakage. On the other hand, each state of a CP, for its corresponding processing step, requires a (multi-)set of resources (including the buffering capacity to hold the thread itself). In our model, a resource type is represented by means of a place whose initial marking represents either the number of available copies of the resource or its capacity. A resource place has input (output) arcs to (from) those transitions of a CP that moves a Thread to (from) a state that requires (was using) a number of copies of this resource type. In the case of DTPs, resource places represent the capacity of the storage device for transmission. The CP of Figure 1.a requires two different types of resources that are modeled by means of the places *Resource1* and *Resource2*. Observe that the CP requires a copy of *Resource1* to realize the operation 1 and a copy of *Resource2* to realize the operation 2. In the DTP of the Figure 2.a the resource place is the place named *Capacity* that represents the size of the storage in the transmission device measured in number of data records, in the figure is equal to one (the initial marking of the place *Capacity*).

– ***Construction of the global model by composition of the Modules with resources***. In order to obtain the global model of the Streaming application, a number of CPs with their corresponding DTPs (accomplishing the data dependencies among them) must be composed. Besides, the resources needed must also be considered at this step. The composition is based on the fusion of the resource places representing the same resource type in the different Modules. The initial marking of the resources, after the fusion, normally is computed as the maximum of the initial markings of the instances that have been merged. The other composition operation is the fusion of a transition representing the production of data records of an output stream in a module with the transition representing the consumption of data records of an input stream belonging to a different module. Observe that it is possible to connect directly two CP without intermediate DTPs, one of the processes acts as producer of data records and the other as consumer of data records but without any intermediate buffer.

3.2 Modular construction of the operational Petri Net model of the streaming application

The functional Petri Net model is derived from a specific algorithm that actually processes a number of given data streams. As already seen, it consists of a composition of computational tasks and the data dependencies among them. In consequence, a *minimal* number of constraints coming from the final execution environment can be taken into account and, in many cases, the functional model

is constructed under a number of hypothesis that may not hold when targeting a specific infrastructure – i.e. the resources required in order to reach the maximum degree of parallelism inherent in the model will not be available, or in case there are resources available, but the economic cost of its usage exceeds the budget, etc. Therefore, refining the functional model with the operational submodel aims at introducing specific resource constraints that may alter either economic cost, performance or even functionality. The alteration of the expected and observed behavior at the functional model may even induce changes into the functional model in order to better target a particular execution environment. In other words, the reason for the operational submodel is to consider explicitly those actual characteristics of a final execution environment, or to compare the response of the application under different deployment scenarios. In this section, we refine the Functional Petri Net model according to the characteristics of the execution environment. Nevertheless, the list of possible constraints imposed by the operational level is very large for the space in this paper. Hence, the following is an illustration on how the operational Petri Net model can be constructed from the Functional Petri Net model in two cases of relevance.

The first one corresponds to the case in which the functional Petri Net model has several DTPs that they were initially independent, but finally in the operational model they must be merged together within the same low-level DTP. The actual refinement procedure is depicted in Figure 3. There, three independent DTPs, P_1, P_2 and P_3, are displayed that were already present in the Functional Petri Net Model. Nevertheless, the design decision to be taken is that the three Processes must share the same Low-Level Data Transmission Process of capacity 2. The refinement of the model requires the splitting of each place s_i of a DTP P_i in 2 places: (1) s_{i1} represents the request of transmission to the low level; (2) s_{i2} represents the end of the transmission. These two places are connected with a low-level DTP of capacity two, as depicted in the figure. Observe that in order to recognize the process requesting the transmission, in the low-level DTP a Polling Algorithm to serve the requests has been implemented that is equal to the Polling Algorithm to send the acknowledgements to High-Level DTPs. The other aspect to take into account in the refinement activity is that in case of having timing information for the processes P_1, P_2 or P_3, this information must be removed before the actual refinement; since, after the refinement the original information, it has no significance. The reason for this is that in the refined model the consumption of time is in the Low-Level DTP.

Another case arises when several CPs of the Functional Petri Net Model, which use resources types in isolation, must share the resources between all CPs. This provokes the rise of competitive relationships. A typical scenario for this transformation appears when the number of CPs is higher than the number of processors and the actual parallelism is limited.

3.3 Exploiting Petri Net models for the economical analysis

A first phase for exploiting the models is to determine *the correction of the adopted solution*, verifying all the good properties expected are satisfied. As a

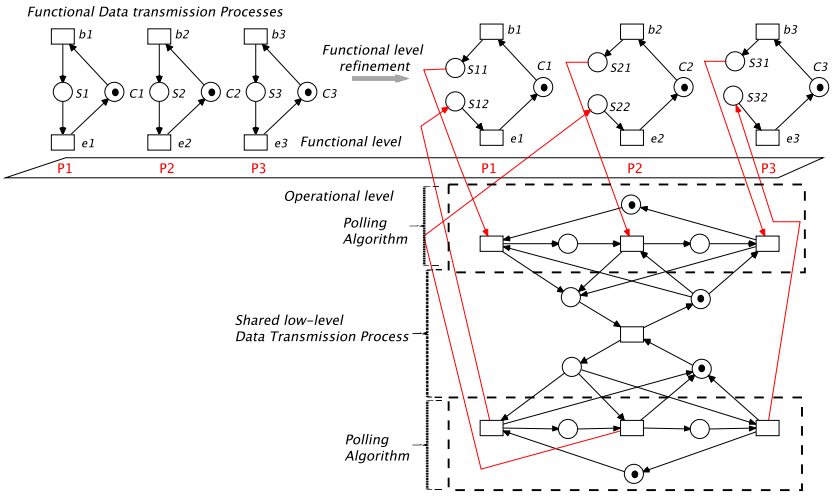

Fig. 3. Refinement of the Functional Petri Net model to take into account the operational data transmission process that must be shared for three functional data transmission processes

second step, when a property is not satisfied, the model can be used for *detecting the causes of the problems* or anomalies in it that prevent it to behave as expected. Then, the understanding of the causes can lead to a *modification of the model*.

Finally, the addition of *performance*-oriented interpretations to the model allows us to compute measurements such as throughput, utilization rates, queue lengths, etc., from which it is possible to determine the time consumed in a particular computation or the resources that have been involved. Both the time consumed and the number of resources involved can be utilised to evaluate the economic cost. The kind of performance measurements that are required to be computed and how they must be exploited for an economic analysis depends on the semantics of the elements in the model and its connection to the real system (streaming application). In the next section we discuss an example of this.

4 Case Study: Wavefront Algorithm for Matrix-Vector Multiplication in streaming

In order to illustrate our methodology, we are making use of the Matrix-Vector Multiplication problem in streaming fashion, in particular, the Wavefront Algorithm [12], which represents a simple solution for large arrays. Due to space limitations, we are focused on Functional analysis, and no particular target infrastructure for the operational analysis is considered here. Let us examine how an algorithm from Linear Algebra can be executed on a square, orthogonal 3×3 wavefront array (Figure 4).

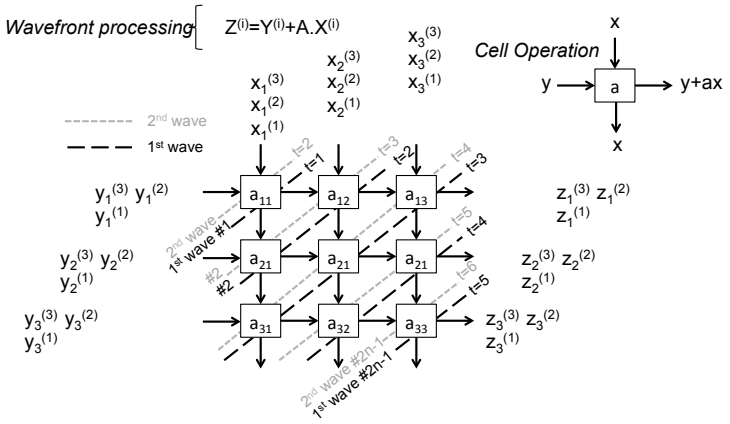

Fig. 4. Wavefront processing of $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)}$, $k = 1, 2, ...$

Let $A = (a_{ij})$ be a 3×3 matrix, and let $X^{(k)} = (x_i^{(k)})$, $Y^{(k)} = (y_i^{(k)})$ and $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)} = (z_i^{(k)})$ be 3×1 matrices for $k = 1, 2, 3$. Initially, the elements of A are stored in the array of processors (a_{ij} in processor P_{ij}). The elements $x_i^{(k)}$, for $k = 1, 2, 3$ are stored from data streams on the top of the i-th column of processors. The elements $y_i^{(k)}$, for $k = 1, 2, 3$ are stored from data streams to the left of the i-th row of processors. The computational process starts with processor P_{11}, where $y_1^{(k)} + a_{11}x_1^{(k)}$ is computed. The appropriate data is then propagated to the neighbour processors P_{12} (the result of P_{11}) and P_{21} (the input data on the top of P_{11}, $x_1^{(k)}$), which execute their respective (similar) operations. The next front of activity will be at processors P_{31}, P_{22} and P_{13}. At the end of this step, P_{13} outputs $z_1^{(1)}$. A computation wavefront that travels down the processor array appears. Once the wavefront sweeps through all the cells, the first computation for $k = 1$ is finished. Similar computations can be executed concurrently with the first one by pipelining more wavefronts in succession immediately after the first one. The wavefronts of two successive computations never intersect, since once a processor performs its share of operations for a given computation, the next set of data that it will receive can only be from the next computation.

4.1 A functional Petri Net model of the Wavefront algorithm for $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)}$, $k = 1, 2, ...$

The functional Petri Net model of the wavefront algorithm sketched in Figure 4 is constructed in a modular fashion. The basic models we need in this case are: (1) A module to describe the Computational Process carried out in a node of the wavefront array; (2) A module to describe de Data Transmission Processes of the input and output data streams to/from the wavefront array. These modules

are depicted in Figure 5. To construct the global model nine instances of the CP of Figure 5.a are needed. The modules of this type belonging to the same row are composed via the fusion of the transition End_i1 with the transition $Sync\_i2$; and the transition End_i2 with the transition $Sync\_i3$. These transitions fusions represent the transmission of the result elaborated by the column 1 or 2, as input to the columns 2 or 3, respectively, without intermediate buffering. Each one of the CPs of the first column is composed with a DTP representing the input stream of the corresponding i-th component of the vector $Y^{(k)}$ via the fusion of the transitions $Sync\_i1$ and *end*. Each one of the CPs of the last column is composed with a DTP representing the output stream of the corresponding i-th component of the vector $Z^{(k)}$ via the fusion of the transitions End_i3 and *begin*. Each one of the CPs of the first row is composed with a DTP representing the input stream of the corresponding i-th component of the vector $X^{(k)}$ via the fusion of the transitions $Sync\_1i$ and *end*. Finally, we connect two CPs belonging to the same column but located in rows 1 and 2, or in rows 2 and 3, by means of a Internal DTP describing the flow of the corresponding component of the vector $X^{(k)}$ through the rows of the array. This connection is done by the fusion of the transitions $Sync\_1j$ and one transition *begin* and the corresponding transition *end* with the transition $Sync\_2j$ (similarly for the case of rows 2 and 3).

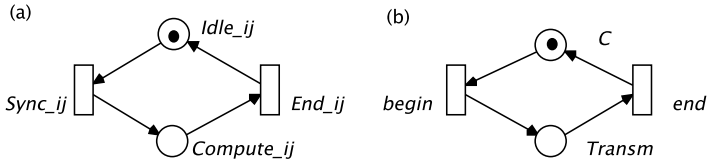

Fig. 5. Basic modules for the construction of the functional Petri Net model of the wavefront algorithm for $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)}$, $k = 1, 2, ...$ (a) Computational Process associated to a node; (b) Data Transmission Process for the external/internal data streams.

Figure 6 depicts an untimed functional Petri Net model of the wavefront algorithm for for $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)}$, $k = 1, 2, ...$ This net model is isomorphous to the flow model in Figure 4.

The addition of time to the the model of Figure 6 will be done in the way described in the previous section: adding a sequence place-transition-place in parallel with the place representing the activity that consumes time. The new added transition will be labeled with the time information. In the example, a timed sequence will be added in parallel with each place $Compute\_ij$ representing the duration of the computation realized by the CP located at row i-th, column j-th. Moreover, a timed sequence will be added in parallel to each place $Transm$ representing the consumption of time in the transmission of a data element in the corresponding DTP.

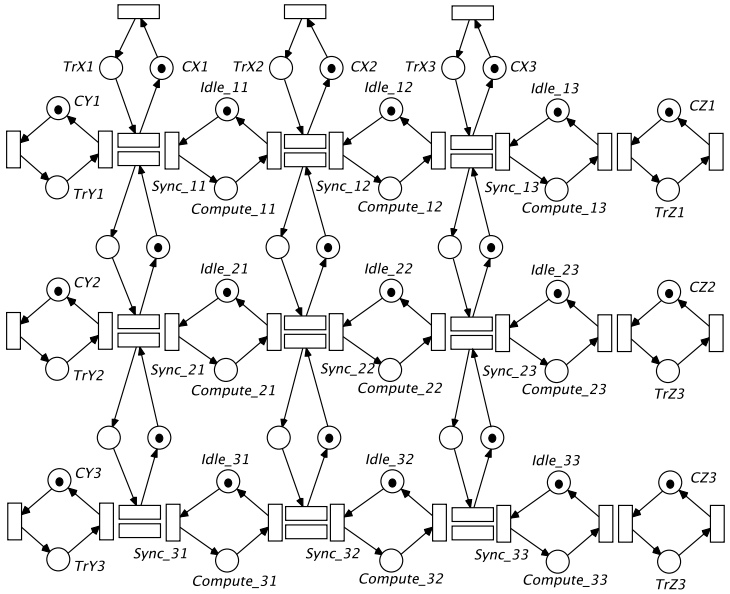

Fig. 6. Untimed functional Petri Net model of the wavefront algorithm for $Z^{(k)} = Y^{(k)} + A \cdot X^{(k)}$, $k = 1, 2, ...$

4.2 Analysis of the model

According to the proposed methodology in this paper, we proceed now to exploit the model we have obtained in Figure 6.

4.3 Structural Analysis

First, we try to determine the correction of design obtained. This is realized exploiting the structural properties of the net. This net is a *strongly connected marked graph* (a subclass of Petri nets in which each place has only one input and one output transition, being strongly connected in the sense of graph theory). Moreover, in this net all circuits contain at least one token. From these characteristics, we obtain the following functional properties of the model:

1) Any transition of the net is fireable from any reachable state of the net (the net is *live*, thus deadlock-free). The minimal repetitive sequence of transition firings contains all transitions of the net exactly once (it is guaranteed from the existence of only one T-invariant: right annuller of the incidence matrix of the net). This is a necessary condition for the execution of a wavefront in the array.
2) The wavefronts propagate in an orderly manner without colliding one into another. This can be concluded when taking into account the maximal difference between firings (in any firing sequence) of a transition $Sync\_ij$ with

respect to its: (1) right neighbor transition $Sync\_i(j+1)$ is equal to 1. To see this, observe that both transitions are covered by a circuit containing only one token. This circuit enforces a strict alternation in the firing of both transitions starting with the firing of $Sync\_ij$; (2) left neighbor transition $Sync\_i(j-1)$ is equal to 0. The reason is the same that in the previous case: the existence of a circuit with a token that enforces the alternation of both transitions starting with the transition $Sync\_i(j-1)$; (3) bottom neighbor transition $Sync\_(i+1)j$ is equal to 1. Once again, this can be proven by means of the vertical circuit covering both transitions and containning only one token; and (4) top neighbor transition $Sync\_(i-1)j$ is equal to 0. All these values can be obtained from the so called marking invariants of the net (that in the case of marked graphs are the the elementary circuits of the net) and can be computed in a structural way.

3) CP in a right to left diagonal can operate concurrently, but two right to left adjacent diagonals cannot fully operate concurrently. This can be easily proven observing that the 4 transitions $Sync\_ij$, $Sync\_i(j+1)$, $Sync\_(i+1)j$ and $Sync\_(i+1)(j+1)$ are covered by an elementary circuit containing only two tokens. This means that only two transitions of these 4 transitions can be concurrently fired. But taking into account the firing relations enumerated in the previous point, only two scenarios are possible: (1) concurrent firing of the transitions $Sync\_ij$ and $Sync\_(i+1)(j+1)$; (2) concurrent firing of the transitions $Sync\_i(j+1)$ and $Sync\_(i+1)j$. This points out the initial statement about the mutual exclusion in the execution of right to left neighbor diagonals.
4) The maximal concurrency can be obtained putting to work concurrently all CPs in odd right to left diagonals or all CPs in even right to left diagonals. This property can be obtained from the previous property extended to the full array.

The previous analysis, without the need for an exhaustive simulation or construction of the state space, points out that it is not possible to have the nine CPs working/executing concurrently. This anomaly or bottleneck limiting concurrency is due to the existence of the circuits covering four transitions, but containing only two tokens as those described in the previous point 3. In order to reach a fully concurrent operation of the nine CPs, we can modify the initial design to enforce circuits with four tokens, covering the four transitions as in the previous point 3. Thus, we can introduce a DTP between two consecutive CPs in a row, decoupling the two CPs by the introduction of a buffer of capacity 1. An alternative solution to enforce the circuits with four tokens to avoid the property of point 3, is to introduce and additional DTP of capacity 1 connected to the already existing between two rows in the same column. This additional buffering enables a design with the maximal possible concurrency in the steady state of the processing of the streams: nine simultaneous computational processes.

Economical Analysis For an economic analysis, the pricing models in [11] could be considered, including pricing cost associated to different data transfers,

CPU time or storage usage. For the sake of simplicity in here, we are just considering an economic cost per CPU usage through time, information that can be added into the model in the following way:

1) Let us assume, under a deterministic timing interpretation, a time α for the timed transitions added to represent the duration of the input of data elmements from the streams corresponding to the vectors $Y^{(k)}$ and $X^{(k)}$ (timed transitions in parallel with the places $TrYi$ and $TrXi$, a time β for transitions corresponding to the execution of the code of the CPs (timed transitions added in parallel with the places $Comp\_ij$), and a time γ for transitions corresponding to the internal transmissions in the wavefront array. A reachable (exact) bound of the throughput of the system can be computed through structural techniques as in [6], obtaining a value equal to the inverse of $max\{\alpha, \beta, \gamma\}$. In general, under a stochastic timing interpretation, the computed bounds following [6] can be eventually improved (in [7] a search for embedded queueing networks was considered).
2) A bound of the mean cycle time for this net (the elapsed time between two consecutive firings of a transition) is the inverse of the throughput, from this value, we can compute the economic cost for the processing of streams of length $k = n$, assuming the cost of the time unit per CPU, p: $Cost = max\{\alpha, \beta, \gamma\} * n * p * 9$. It allows the designer to have an accurate estimation of the cost taking into account the pricing applied to CPU time consumed. Note that this analysis corresponds to the functional level and no information on the operational level has been considered. In this case, the model is designed with nine CPs having each nine computational resources in isolation.
3) Taking into account that the model in Figure 6 is a marked graph an optimal scheduling policy is just the earliest-firing-time policy (i.e. fire the transitions as soon as possible).

The previous analysis has be done on the functional Petri Net model. In case we wanted to introduce a particular operational level, we would have to proceed to the refinement of the previous functional Petri Net model and proceed to a similar analysis to the previously carried out: the timing and pricing information that are currently in the Functional level would be removed, and then they would be appearing in the new refined submodel. Besides, in such a refined model, the relationship between economic cost and performance can be studied: i.e. how the performance behaves when the economic cost is reduced.

5 Conclusions and Future work

In this paper, we have shown that Petri Nets (eventually with an associated interpretation) are useful formal models to describe and analyze functional, performance and economical properties (of interest for the designer) of streaming applications. The separation between the graph-based structure of the model and dynamic properties such as the marking, enables the use of many structure-based

analysis techniques. Through a simple Matrix-Vector multiplication in streaming fashion, the wavefront algorithm, we have shown the underlying methodology in the use of Petri Nets. First, the modeling task has been realized in a modular fashion using the subnet describing the behavior of computations and data transmissions. After that, we have applied some structural analysis techniques (due to their good tradeoff between decision power and computational complexity) to verify that the functional behavior of our model was the expected: the wavefronts are propagated through the array in an orderly way. In order to verify some quantitative properties, namely throughput and economic cost, have added a timing interpretation to some transitions of the net, and have assumed an associated CPU pay-per-use cost.

6 Acknowledgements

This work was supported by the Spanish Ministry of Economy under the program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad", project id TIN2013-40809-R.

References

1. Bañares, J.Á., Rana, O.F., Tolosana-Calasanz, R., Pham, C.: Revenue creation for rate adaptive stream management in multi-tenancy environments. In: Altmann, J., Vanmechelen, K., Rana, O.F. (eds.) Economics of Grids, Clouds, Systems, and Services - 10th International Conference, GECON 2013, Zaragoza, Spain, September 18-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8193, pp. 122–137. Springer (2013)
2. Bernardi, S., Merseguer, J.: Performance evaluation of UML design with stochastic well-formed nets. J. Syst. Softw. 80(11), 1843–1865 (2007)
3. Bernardo, M., Ciancarini, P., Donatiello, L.: Aempa: a process algebraic description language for the performance analysis of software architectures. In: Workshop on Software and Performance. pp. 1–11 (2000)
4. Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H., Moran, C.: Ibm infosphere streams for scalable, real-time, intelligent transportation services. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. pp. 1093–1104. SIGMOD '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1807167.1807291
5. Bioernstad, B.: A Workflow Approach to Stream Processing. Phd thesis, ETH Zurich, Computer Science Department (2008), http://e-collection.ethbib.ethz.ch/view/eth:30739
6. Campos, J., Chiola, G., Colom, J.M., Silva, M.: Properties and performance bounds for timed marked graphs. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on 39(5), 386–401 (1992)
7. Campos, J., Silva, M.: Embedded product-form queueing networks and the improvement of performance bounds for petri net systems. Performance Evaluation 18(1), 3–19 (1993)
8. Chen, W., Ferreira da Silva, R., Deelman, E., Sakellariou, R.: Balanced task clustering in scientific workflows. In: 2013 IEEE 9th International Conference on eScience (eScience). pp. 188–195 (2013)

9. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming scientific and distributed workflow with Triana services: Research articles. Concurr. Comput. : Pract. Exper. 18(10), 1021–1037 (2006)
10. Dou, L., Zinn, D., McPhillips, T., Kohler, S., Riddle, S., Bowers, S., Ludascher, B.: Scientific workflow design 2.0: Demonstrating streaming data collections in kepler. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on. pp. 1296–1299 (April 2011)
11. Gohad, A., Narendra, N.C., Ramachandran, P.: Cloud pricing models: A survey and position paper. In: Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on. pp. 1–8. IEEE (2013)
12. Kung, S.Y., Arun, K.S., Gal-Ezer, R.J., Rao, D.V.B.: Wavefront array processor: Language, architecture, and applications. IEEE Trans. Computers 31(11), 1054–1066 (1982), http://dblp.uni-trier.de/db/journals/tc/tc31.html#KungAGR82
13. Kungt, H., Leisersont, C.E.: Systolic arrays (for vlsi). Society for Industrial\ & Applied p. 256 (1979)
14. Lee, K., Paton, N., Sakellariou, R., Deelman, E., Fernandes, A., Mehta, G.: Adaptive workflow processing and execution in Pegasus. In: Third International Workshop on Workflow Management and Applications in Grid Environments (WaGe08), May 25-28, 2008, Kunming, China. pp. 99–106 (May 2008)
15. Marinescu, D.C.: Cloud Computing: Theory and Practice. Morgan Kaufmann (2013)
16. Menascé, D.A., Gomaa, H.: A method for design and performance modeling of client/server systems. IEEE Trans. Software Eng. pp. 1066–1085 (2000)
17. Murata, T.: Petri nets: Properties, analysis and applications. In: Proceedings of IEEE. vol. 77, pp. 541–580 (Apr 1989)
18. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: Distributed stream computing platform. In: Proceedings of the 2010 IEEE International Conference on Data Mining Workshops. pp. 170–177. ICDMW '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/ICDMW.2010.172
19. Ngu, A., Bowers, S., Haasch, N., McPhillips, T., Critchlow, T.: Flexible scientific workflow modeling using frames, templates, and dynamic embedding. Scientific and Statistical Database Management pp. 566–572 (2008)
20. Pautasso, C., Alonso, G.: Parallel computing patterns for Grid workflows. In: Proceedings of the HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06) June 19-23, Paris, France (2006)
21. Tolosana-Calasanz, R., Bañares, J.A., Rana, O.F.: Autonomic streaming pipeline for scientific workflows. Concurr. Comput. : Pract. Exper. 23(16), 1868–1892 (2011)
22. Tolosana-Calasanz, R., Bañares, J.Á., Pham, C., Rana, O.F.: Enforcing qos in scientific workflow systems enacted over cloud infrastructures. Journal of Computer and System Sciences 78(5), 1300–1315 (2012)
23. Tolosana-Calasanz, R., Rana, O.F., Bañares, J.A.: Automating performance analysis from Taverna workflows. In: Chaudron, M.R.V., Szyperski, C.A., Reussner, R. (eds.) Component-Based Software Engineering, 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5282, pp. 1–15. Springer (2008)