

Decentralized Resources Management for Grid

Thibault Bernard Alain Bui Olivier Flauzac
Cyril Rabat

SysCom, CReSTIC

Université de Reims Champagne-Ardenne

BP1039, F-51687 Reims Cedex 2, France

`thibault.bernard,alain.bui,olivier.flauzac,cyril.rabat}@univ-reims.fr`

Abstract

Among all components of a grid or peer-to-peer application, the resources management is unavoidable. Indeed, new resources like computational power or storage capacity must be quickly and efficiently integrated. This management can be achieved either by a fully centralized way (*BOINC*) or by a hierarchical way (*Globus*, *DIET*). In the latter case, there is a greater flexibility and a greater scalability. But the counterpart is the difficulty to design and to deploy such a solution, particularly if the resources are volatile.

In this article, we combine random walks and circulating word to derive a fully distributed solution to the resources management. Random walks have proved their efficiency in distributed computing and are well suited to dynamical networks like peer-to-peer or grid networks. There is no condition on nodes lifetime and we need only one application for each node.

1 Introduction

The aim of grid computing is to share resources for a global use. These resources can be in a local network or widely distributed over the Internet. Since few years, a lot of solutions have been proposed and each one is adapted to a specific grid application.

The authors of [1] have identified several kinds of grid applications. Among all of them, the *on-demand computing* needs a good cost-performance ratio rather than an absolute performance. At the opposite, the *distributed supercomputing* needs a lot of computing resources and generally this kind of grid gathers supercomputers. So, the resources management highly depends on the application that uses the grid and must be adapted in function.

In [2], the authors define the resources management as the localization and allocation of computational resources, process creation and resource prepara-

tion. They propose a complete architecture composed by resource brokers, resource co-allocators and resource managers. These components are integrated to the *Globus Toolkit* ([3]). So, the resources management is distributed and needs a hierarchy of components. For the middleware *DIET* ([4]), the authors choose to organize a hierarchy of servers called *agents*. Even if distributed methods are more scalable and flexible, the hierarchy of *Globus* and *DIET* can be hard to configure. How many components or agents we need and where they have to be installed?

A peer-to-peer approach has been proposed in [5] that lets the nodes in charge of the grid functions (resources management and scheduling). Another fully distributed method is proposed for the *CONFIIT* middleware ([6]): the nodes are gathered in a virtual ring and the topology management is assumed by a token circulation.

Our solution concerns wide volatile grids like desktop grids that are well suited to the on-demand computing. This kind of grid is composed of many computers that have a limited and heterogeneous computing power and a high volatility. No assumption is made on the nodes lifetime. The network has a low and variable bandwidth. So it is important to limit the number of exchanged messages. In these conditions, the random walk has already proved its efficiency. In [7], the authors use it in peer-to-peer networks for searching and building dynamical topology. Adaptation to dynamical environments is also a quality of the random walks: in [8], a solution for routing in ad-hoc networks is proposed. So, our resources management combines a random walk of a token and a circulating word to avoid the maintenance of virtual structures. It allows the detection of connection or lost of connection of resources and localization of specific resources corresponding to a request.

2 Preliminaries

A model for grid In [9], we propose a model composed of 5 layers to analyze grid applications. The three lower layers concern the network, the routing and the messages exchange protocols. Layer 4 represents the resources management for the grid and the last one the other grid components (scheduling, monitoring, ...). Even if the solution we expose here is for Layer 4, a grid is built over three other layers and we have to take care of their impacts for the resources management. We show that we can model a grid by a *directed* graph $G = (V, E)$, where V is a set of active nodes of the grid with $|V| = n$ and E is the set of directed communication links. An active node is a resource or a node that uses resources. In the following, we use the terms "resource", "node" and "active node" interchangeably.

A communication link (i, j) exists if and only if j is a neighbor of i in the grid, i.e. i can directly send a message to j . Every node i can distinguish all its links of communication and maintains a set of neighbors denoted N_i . We consider that all resources of the grid have a distinct identity (IP address, for example or a complete description with a specific language like *RSL* [2], in that

case an indexation is needed to have better performances).

As G is a communication graph, we assume it is strongly connected. Indeed, if the graph is not strongly connected at a time, there exists a sink subgraph $E(G)$: resources of $G \setminus E(G)$ cannot be reached from any node of $E(G)$. For a token circulation, it means that the token will stay in $E(G)$ and cannot reach nodes of $G \setminus E(G)$. With our method, we accept that the graph stays not strongly connected during a short time. If this transient state is too long, unreachable resources will be considered as disconnected.

Random walks A random walk is a sequence of nodes visited by a token that starts at i and visits other resources according to the following transition rule: if the token is at i at time t then at time $t + 1$, it will be at one of the neighbors of i , chosen uniformly at random among N_i ([10, 11]). Similarly to deterministic distributed algorithms, the time complexity of random walk based token circulation algorithms can be viewed as the number of "steps" it takes for the algorithm to achieve the network traversal. With only one walk at a time (which is the case we deal), it is also equal to the message complexity. The cover time C — the average time to visit all nodes in the system — and the hitting time denoted by h_{ij} — the average time to reach a node j for the first time starting from a given node i — are two important values that appear in the analysis of random walk-based distributed algorithms. Both of them are on average bounded by n^3 . There are three properties about random walks: *percussion* — an arbitrary node is visited in a finite time, *coverage* — all nodes are visited in a finite time and *meeting* — several random walks will meet each other in a finite time. We have proposed a method to compute such values in [12].

3 Resources management

We propose a fully distributed algorithm that maintains the set of grid resources and paths to reach them from any nodes of the grid. It is based on the token random moves. As it circulates perpetually, the set of grid resources and paths are maintained through a circulating word and are automatically updated.

3.1 Circulating Word Management

A *circulating word* is a token message collecting pieces of information along its moves through the grid. This concept has been introduced in [13]. The author presents the detection of all the communication graph cycles with the evaluation of the collected information of circulating words successively broadcasted by all nodes. Indeed, each node can build the path of received token thanks to the circulating word content.

We propose to use this concept in order to maintain a partial image of the grid communication graph. Such an image can be used to reach a resource corresponding to a specific request from a node. In [14], we address the problem of

routing tables computation (i.e. maintain paths between each pair of nodes) but in an undirected environment. Because we focus on directed communications according to our model, we have to propose a new circulating word information management.

We note W_i the i^{th} element of the circulating word W . Several functions are used to manage W . $size(W)$ returns the size of the word and $identities(W)$ returns the set of the distinct resources identities contained in the word. To add an element e at the first position (at left), we use the function $add(W, e)$. $remove(W, i, j)$ deletes the subword $\langle W_i, \dots, W_j \rangle$. Finally, functions $left(W, k)$ and $right(W, k)$ return respectively the subwords $\langle W_1, \dots, W_k \rangle$ and $\langle W_k, \dots, W_{size(W)} \rangle$.

In order to collect the visited resources identities, each time the token moves, the current identity is added in the word at the first position (at the left). With the word content, a node can eventually compute a path from it to others resources. Particularly, it is possible to build from a specific position in the word paths from a specific resource to all others visited resources. The position of this element is called *minimal position*.

Definition 1 *The minimal position noted pos_{min} of word W satisfies:*

$$pos_{min} = \min \{i \in [1, size(W)] \mid \forall k \in identities(W), \exists j \leq i, W_j = k\}$$

From pos_{min} , we can build a (partial) spanning tree rooted in $W_{pos_{min}}$. But it is insufficient to construct paths between each pair of nodes unless there exists a path from W_1 to $W_{pos_{min}}$. If there exists j such that $W_1 = W_j$ and $pos_{min} < j$, such a path exists. So, we have a cycle $\langle W_1, \dots, W_{pos_{min}}, \dots, W_j \rangle$.

Definition 2 *Word W contains a cycle if there exists $(i, j) \in \{1, \dots, size(W)\}^2$, $i < j$ and $W_i = W_j$. We note this cycle $\mathcal{C}(i, j)$.*

If all visited nodes identities are contained in a cycle $\mathcal{C}(i, j)$, we can construct a path between each visited node.

Definition 3 *A cycle noted $\mathcal{C}(i, j)$ in Word W is called constructive if:*

$$\forall k \in identities(W), \exists l \in \{i, \dots, j\} \mid W_l = k$$

Remark 1 *The first constructive cycle is reached when the token comes back to the first visited node.*

Getting a constructive cycle is sufficient but not necessary to construct paths between visited nodes. For instance, the word $\langle 1, 2, 3, 1, 4, 3 \rangle$ does not contain a constructive cycle, but we have paths between each pair of nodes. By combining two non-constructive cycles, we can build a constructive cycle: $\langle 3, 1, 4, 3 \rangle$ is rotated and becomes $\langle 1, 4, 3, 1 \rangle$ and with the cycle $\langle 1, 2, 3, 1 \rangle$, the word becomes $\langle 1, 2, 3, 1, 4, 3, 1 \rangle$ that contains a constructive cycle.

We decide to base our resources management on a constructive cycle maintenance inside the circulating word to reduce the complexity.

3.2 Constructive cycle maintenance

As the construction of all paths between resources does not require information beyond the constructive cycle, older information (at its right) can be deleted. But not the part at its left: it allows its evolution and thus to handle dynamicity.

Inside the constructive cycle, we can found useless information. But it is not possible to delete it easily. If we delete the subword $\langle W_i, \dots, W_j \rangle$, the link (W_{i-1}, W_{j+1}) appears in the word and the node cannot verify if it exists in the communication graph of the grid. But, if W owns a cycle $\mathcal{C}(i, j)$, we can delete the subword $\langle W_{i+1}, \dots, W_j \rangle$ because the link (W_i, W_{j+1}) really exists. So, if information in $\mathcal{C}(i, j)$ is useless, we can delete this cycle called *non-constructive*.

Definition 4 A cycle $\mathcal{C}(i, j)$ is called *non-constructive* if $\forall k, i < k < j, \exists (l > j) \vee (l < i) | W_k = W_l$

In order to reduce all non-constructive cycles in the constructive cycle, we use Algorithm 1. For each identity of the word, the algorithm searches the same identity at the left. If it exists, the word contains a cycle. Then the algorithm checks if it is non-constructive. In fact, it has to search all identities of the cycle (the set \mathcal{V}_C) outside. To speed up this search, a set \mathcal{V} is maintained that contains the identities at the right of the current identity.

Algorithm 1 Reduction of non-constructive cycles in a circulating word W

```

 $\mathcal{V} \leftarrow \emptyset$ 
 $pos \leftarrow size(W)$ 
while  $pos > 2$  do
   $\mathcal{V} \leftarrow \mathcal{V} \cup W_{pos}$ 
   $i \leftarrow pos - 1$  /* We search a cycle  $\mathcal{C}(i, pos)$  */
   $\mathcal{V}_C \leftarrow \emptyset$ 
  while  $(i > 1) \wedge (W_i \neq W_{pos})$  do
    if  $W_i \notin \mathcal{V}$  then
       $\mathcal{V}_C \leftarrow \mathcal{V}_C \cup W_i$ 
       $i \leftarrow i - 1$ 
  if  $W_i = W_{pos}$  then
     $j \leftarrow i - 1$  /* Cycle found : searching the elements of  $\mathcal{V}_C$  at the left of  $i$  */
    while  $(j \geq 1) \wedge (\mathcal{V}_C \neq \emptyset)$  do
       $\mathcal{V}_C \leftarrow \mathcal{V}_C / W_j$ 
       $j \leftarrow j - 1$ 
    if  $\mathcal{V}_C = \emptyset$  then
       $remove(W, i + 1, pos)$  /* The cycle  $\mathcal{C}(i, pos)$  is a non-constructive cycle */
       $pos \leftarrow i$ 
    else
       $pos \leftarrow pos - 1$ 
  else
     $pos \leftarrow pos - 1$ 

```

Lemma 1 The size of the constructive cycle is bounded by $\frac{n^2}{4} + n$.

Proof We note i the identity which appears the most in the constructive cycle and k the number of occurrences of i . So, there are $k - 1$ cycles that are **not** non-constructive in the constructive cycle (else these cycles would have been reduced). So there are $k - 1$ identities that appear only one time in the constructive cycle and $n - (k - 1)$ identities that appear at most k times (by

definition of k). The size of the word can be write in function of k : $T(k) = (k - 1) + (n - (k - 1)) * k = -k^2 + (n + 2)k - 1$. $T(k)$ has a maximum when $T'(k)$ equals 0: $T'(k) = 0 \Leftrightarrow k = \frac{n+2}{2}$. Then:

$$T_{max} = \begin{cases} \frac{n^2+4n}{4} & \text{if } n \text{ is even} \\ \frac{n^2+4n-1}{4} & \text{if } n \text{ is odd} \end{cases}$$

3.3 General description of the algorithm

The algorithm is divided in 3 different phases: the *initialization phase* consists in adding identities until obtaining a constructive cycle, the *maintenance phase* aims to maintain and let it evolve and the *collect phase* is an intermediary phase that consists in maintaining a maximal cycle that speeds up the construction of a constructive cycle when a new identity is discovered.

3.3.1 Initialization phase

During this phase, information is collected in the word in order to build a constructive cycle. Then, the algorithm passes in the maintenance phase. At each move of the token, we reduce non-constructive cycles thanks to Algorithm 1 in order to limit the size of the token.

The percussion property of random walks ensures that the token will return on the first visited node. So, by Remark 1, a constructive cycle will be created and the algorithm will reach the maintenance phase. In fact Algorithm 1 can just be executed at this time.

Remark 2 *The deletion of non-constructive cycles during this phase reduces the token size but some constructive information may be deleted. For instance, the word $\langle 3, 4, 1, 3, 1, 2, 1 \rangle$ contains a non-constructive cycle $\langle 1, 3, 1 \rangle$. If such a cycle is removed, it is no longer possible to build path between 3 and 4. But the set identities(W) is not modified and as described in the following, a constructive cycle finally appears.*

3.3.2 Maintenance phase

In this phase, the word contains a constructive cycle. The aim is to let it evolve in order to adapt it to topological changes (resource connection or disconnection). So, when the token comes in a node, its identity is added at the left of the constructive cycle (this part is called the *head*) until a new one is built. When i identities have been added, the word has the following structure:

$$\langle \underbrace{W_1, \dots, W_i}_{head}, \underbrace{W_{i+1}, \dots, W_{size(W)}}_{C_C(i+1, size(W))} \rangle$$

At each addition, either **the identity is a new one** so the constructive cycle is broken and the algorithm enters the collect phase or **the identity already appears in the word** and the algorithm verifies if a reduction is possible (a non-constructive cycle or a new constructive cycle).

Remark 3 *The search of non-constructive cycles must be achieved only in the head else it could induce a loss of relevant information. For example, the word $\langle 4, 2, 1, 5, 2, 4, 2, 3, 1 \rangle$ contains a constructive cycle $\langle 1, 5, 2, 4, 2, 3, 1 \rangle$ and a non-constructive cycle $\langle 2, 4, 2 \rangle$. If we remove this last one, Identity 4 disappears of the constructive cycle.*

Property 1 *If a word contains a constructive cycle $\mathcal{C}_C(i + 1, \text{size}(W))$, then all cycles $\mathcal{C}(j, k)$ with $j < k \leq i + 1$ are non-constructive.*

This property improves the search of non-constructive cycles. If we apply such a reduction, it ensures the only possible non-constructive cycle is $\mathcal{C}(1, j)$ with $j \leq i + 1$. Moreover $\text{size}(\text{left}(W, i))$ becomes bounded by $n - 2$.

If a non-constructive cycle is found, then the constructive cycle is not modified. Else, we have to search if a new one appears.

Property 2 *If a word contains a constructive cycle $\mathcal{C}_C(i + 1, \text{size}(W))$, then the minimal position is inside $\mathcal{C}_C(i + 1, \text{size}(W))$.*

Property 3 *Assume a word contains a constructive cycle $\mathcal{C}_C(i + 1, \text{size}(W))$. If a new element is added to the word and if a new constructive cycle appears, its right bound is in $[\text{pos}_{\min}, \text{size}(W)]$.*

We have only to search W_j , the right bound of the new constructive cycle, in the part $\langle W_{\text{pos}_{\min}}, \dots, W_{\text{size}(W)} \rangle$. In this subword, we cannot have any cycle and W_1 cannot appear (else a new constructive cycle is built). So, $\text{right}(W, \text{pos}_{\min})$ is bounded by $n - 2$. Finally, the maintenance phase has only to analyze the subwords $\langle W_1, \dots, W_i \rangle$ and $\langle W_{\text{pos}_{\min}}, \dots, W_{\text{size}(W)} \rangle$ then only n identities.

3.3.3 Collect phase

To improve the construction of a new constructive cycle, we maintain a maximal cycle. So, we have the following structure:

$W_1, \dots, W_i, \dots, W_k$ with $k = \text{size}(W)$ and $\mathcal{C}(i, \text{size}(W))$ the maintained cycle

So, in this phase, we only add new identities until an identity already appears in this cycle. Then, we can easily rewrite the word:

$$W_1, \dots, W_i, \dots, W_j, \dots, W_k, \dots, W_j \text{ with } W_j = W_1$$

We have a new constructive cycle and the algorithm can return to the maintenance phase after reducing non-constructive cycles.

3.3.4 Main algorithm

From the circulating word, the main algorithm can detect in which phase it is. This detection depends on the minimal position. If $\text{pos}_{\min} = \text{size}(W)$, the

algorithm is in the initialization phase. Else we search from W_1 a position i such $W_i = W_{size(W)}$. Then, $\mathcal{C}(i, size(W))$ can be a maximal cycle or a constructive cycle. We verify if W_1 exists in this cycle then the algorithm is in the maintenance phase else the algorithm is in the collect phase.

3.4 Faults management

Each fault that occurs in a grid must be managed to avoid unreachable resources. There are different kinds of faults that can be gathered in two main categories: topological changes and communication faults.

3.4.1 Topological changes

As specified in Section 2, each node i manages its neighborhood denoted by N_i and if a network change occurs like a resource disconnection, N_i is updated and no other operation has to be done. Nevertheless a global topological information is stored in the token then a network change can produce erroneous paths. So, at the reception of the token, the node has to check locally the consistency between its neighborhood and the topological information contained in the word.

For instance, if a link (W_i, W_j) exists in the word and if Node W_j does not appear in N_{W_i} , Node W_i detects an inconsistency and has to correct the word. It removes the subword $\langle W_{k+1}, \dots, W_j \rangle$ where W_k is the first neighbor of Node W_i at its left. For instance, if Node 4 has $N_4 = \{1, 2\}$ and if the word is $\langle 1, 2, 3, 4, 2, 4 \rangle$, there is an inconsistency and Node 4 reduces the word to $\langle 1, 2, 4, 2, 4 \rangle$.

Remark 4 *The correction of the word is done with local knowledge and some inconsistencies cannot be corrected. The circulation of the token will ensure that all concerned nodes apply this correction. Then all inconsistencies will be deleted.*

3.4.2 Communication faults

The first kind of communication faults concerns the *loss of a message*. To avoid the lack of token in the system, we place a timeout in each node, that is reseted on a token visit. On timeout triggering, the associated node produces a new empty token. The initial timeout value is very important. A good value is around $max_{(i,j) \in V^2} \{h(i, j)\}$ which is the average time to hit an arbitrary node.

Finally, we have to manage the *duplication of a message*. The timeout procedure described previously can produce a new token even if a token is already alive in the network. In order to reduce the bandwidth consumption the number of tokens should be reduced by keeping only one of them. To speed up the integration of new resources, the algorithm should also merge the topological information gathered by these tokens. The merge can be achieved

easily if there is at least one constructive cycle in one of the tokens. If the merge can not be achieved, the two tokens continue their walks.

When the algorithm compares the two circulating words $W1$ and $W2$, it compares the sets $identities(W1)$ and $identities(W2)$ in order to build the largest (partial) image of the communication graph. So, if $identities(W1)$ is included inside $identities(W2)$, $W1$ can be dropped (all resources are already known by $W2$). On the other hand, the algorithm merges the two words by inserting the constructive cycle of $W1$ inside $W2$. Because the token are on Node i , they share Identity i so such an insertion is always possible as soon as the element at the bound of a constructive cycle is the current node. If not, we can rotate it as described in Section 3.2.

4 Resources localization

Resources management also consists in localization of a resource corresponding to a specific request. From the topological information of the circulating word, it is possible to build spanning structures. In particular, if the word contains a constructive cycle, we can build a spanning tree from each node. This construction is achieved by Algorithm 2.

Algorithm 2 Construction of a spanning tree \mathcal{T} rooted on Node $root$ from a circulating word W

```

set_root( $\mathcal{T}$ ,  $root$ )
 $i \leftarrow 1$ 
 $pos \leftarrow 1$ 
while  $i < size(W)$  do
  while  $(i < size(W)) \wedge (W_i \notin \mathcal{T})$  do
     $i \leftarrow i + 1$  /* Node  $W_i$  is not in the tree */
  if  $W_i \in \mathcal{T}$  then
     $j \leftarrow i - 1$  /* Adding the new branch rooted on  $i$  */
    while  $j \geq pos$  do
      if  $W_j \notin \mathcal{T}$  then
        add_tree( $\mathcal{T}$ ,  $W_j$ ,  $W_{j+1}$ )
       $j \leftarrow j - 1$ 
     $i \leftarrow i + 1$ 
   $pos \leftarrow i$ 

```

If a node needs to localize a specific resource of the grid, we can use the wave with feedback mechanism like *DIET*. It means that the request is sent along a spanning tree. The corresponding resource can reply along this same tree. But with our model, the grid is modeled as a directed graph and the spanning tree is not valid for the reply. So, we need two different trees: a diffusion tree \mathcal{T}_D computed by Algorithm 2 and a feedback tree \mathcal{T}_F . Whereas \mathcal{T}_D is directed from the request node to resources, \mathcal{T}_F is directed from resources to the request node. The first identity of the circulating word is the request node one so, for construction of \mathcal{T}_F , each identity of the word is added as the son of its left identity.

If a node needs a specific resource, at the token reception, it builds \mathcal{T}_D and \mathcal{T}_F . Then, it can send its request along \mathcal{T}_D (\mathcal{T}_F must be diffused in this request).

Each node that receives this request can reply along \mathcal{T}_F . Then, the emitter can select among the set of the free corresponding resources.

5 Conclusion

We propose in this article, a resources management for wide and volatile grids. Contrary to many others resources managements, our solution is fully distributed. It is based on random walk and circulating word. No centralization on a node is needed and no hierarchy has to be deployed. So, it limits the waste of computational power. On the other hand, we take into account directed communications in order to maximize the resources localization and aggregation.

References

- [1] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufman Publishers, September 1999.
- [2] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, ser. LNCS, vol. 1459. Springer-Verlag, 1998, pp. 62–82.
- [3] I. Foster and C. Kesselman, "Globus : a metacomputing infrastructure toolkit," in *Supercomputer Applications*, I. Press, Ed., vol. 11 (2), 1997, pp. 115–128.
- [4] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter, "A Scalable Approach to Network Enabled Servers," in *Proceedings of the 8th International EuroPar Conference*, ser. LNCS, vol. 2400. Springer-Verlag, 2002, pp. 907–910.
- [5] J. Cao, O. M. K. Kwong, X. Wang, and W. Cai, "A Peer-to-Peer Approach to Task Scheduling in Computation Grid." in *GCC (1)*, ser. LNCS, vol. 3032. Springer-Verlag, 2004, pp. 316–323.
- [6] O. Flauzac, M. Krajecki, and J. Fugere, "CONFIIT : a middleware for peer to peer computing," in *ICCSA 2003*, ser. LNCS, vol. 2669. Springer-Verlag, 2003, pp. 69–78.
- [7] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," in *INFOCOM*, 2004.
- [8] W. Choi, S. Das, J. Cao, and A. Datta, "Randomized dynamic route maintenance for adaptative routing multihop mobile ad hoc networks," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 107–123, 2005.

- [9] C. Rabat, A. Bui, and O. Flauzac, “A random walk topology management solution for grid,” in *II2CS*, ser. LNCS, vol. 3908. Springer-Verlag, 2006, pp. 91–104.
- [10] L. Lovasz, “Random walks on graphs: A survey,” in *Combinatorics: Paul Erdos is Eighty (vol. 2)*. Janos Bolyai Mathematical Society, 1993, pp. 353–398.
- [11] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, and C. Rackoff, “Random walks, universal traversal sequences and the complexity of maze problems,” in *20th IEEE Annual Symposium on Foundations of Computer Science*, 1979, pp. 218–223.
- [12] T. Bernard, A. Bui, M. Bui, and D. Sohier, “A new method to automatically compute processing times for random walks based distributed algorithm,” in *ISPD 03*, vol. 2069. IEEE Computer society Press, 2003, pp. 31–36.
- [13] I. Lavallée, *Algorithmique distribuée et parallèle*, Hermes, Ed. Hermes, 1990.
- [14] O. Flauzac, “Random circulating word information management for tree construction and shortest path routing tables computation,” in *On Principle Of DIstributed Systems*. Studia Informatica Universalis, 2001, pp. 17–32.