Impact of nodes distribution on the performance of a P2P computing middleware based on virtual rings

Luiz Angelo Steffenel, Christophe Jaillet, Olivier Flauzac, Michaël Krajecki

Impact of nodes distribution on the performance of a P2P computing middleware based on virtual rings

Luiz-Angelo Steffenel, Christophe Jaillet, Olivier Flauzac and Michael Krajecki
CReSTIC-SysCOM, Université de Reims Champagne-Ardenne,
BP 1039, F-51687 Reims Cedex 2, France
Email: firstname.lastname@univ-reims.fr

***Abstract*—P2P computing middlewares are interesting options for grid computing applications that require scalability and resiliency. Nevertheless, most P2P computation systems rely on partially centralized or hybrid decentralized architectures to distribute tasks and collect the results, raising fault tolerance and bottleneck issues. CONFIIT (Computation Over Network with Finite number of Independent and Irregular Tasks) is a purely decentralized middleware for grid computing, relying on a virtual ring for topology management and for task scheduling. This paper evaluates the impact of node placement and task granularity on the performance of CONFIIT. We analyze how CONFIIT handles different distribution scenarios in a grid environment while solving the well-known Langford permutation problem.**

I. Introduction

During many years, computation and storage were done in a local way. A few years ago, a new approach emerged: the distribution of computation and storage over a network. This new approach tends to the use of all available computational resources from a local network, and to a larger scale, from the Internet. This new paradigm is called grid computing. In [1], authors discussed arising issues related to this new paradigm, and explained how to design efficient grid based solutions.

CONFIIT, introduced in [2], is a purely decentralized peer-to-peer middleware. It was designed to cope with joining the computation means to be used on a local area network or on the Internet, all while providing fault tolerance and autonomy to the peers. This paper focuses on the impact of nodes placement on the communication performance when using the **distributed mode** of CONFIIT. Because CONFIIT relies on a virtual ring, the token passing mechanism is intrinsically bound to the network speed and the ring circumference. Using the well-known Langford's combinatorial problem as a test subject, we try to verify the impact of nodes placement in the overall system performance when deploying CONFIIT over a country-wide grid environment.

The rest of this paper is organized as follows: firstly, we present some issues in grid computing. Section III describes the CONFIIT framework. We detail how the topology management is used to improve robustness, and how tasks are locally and globally scheduled on processors. Section IV illustrates how the Langford's combinatorial problem can be implemented in this computing model. After its description, experimental results in a grid environment are presented in order to observe the impact of nodes placement in the overall performance. Finally, Section VI presents our conclusions.

II. Related works

The grid computing concept appeared in the 90's, as an extension on the Internet of the *cycle stealing* principle. Former applications aimed to crack by exhaustive search encryption keys based on RC5 or DES algorithms. They showed the opportunity of aggregating hundreds of PCs to solve a problem. Web-based Computing projects arose at the end of the 90's, as Charlotte [3], Bayanihan [4] or SETI@home [5], [6].

Indeed, the SETI@home project [5], [6] popularized the *Global Computing* concept. It was designed for Searching ExtraTerrestrial Intelligence, distributing tasks on thousands of computers on Internet. The current version of the project is based on the BOINC platform [7], and gathers more than 30 TeraFlops simultaneously. It has already produced more than the equivalent of 500,000 years of computation on a PC. After SETI@home, dozens of academic or industrial projects were developed. In the former world, projects are often dedicated to a single application (as Distributed.net) or propose complete environments: XtremWeb[1] or OurGrid[2].

Grid applications are currently designed for two main purposes: data sharing and distributed computation. Whereas data sharing applications aim to share and recover data through a network, distributed computation applications aim to share a computation over a network and build a global solution. In the remaining of this paper, we will focus almost exclusively on distributed computation grid applications. Hence, the grid related issues can be classified in three main categories.

(i) *applications* specially designed to compute a particular problem. The most known application of that category is the SETI project, designed to Search for Extra Terrestrial Intelligence.
(ii) *protocols* and *libraries*, put together to help in writing a dedicated grid application. JXTA project is an example of this category [8]. It proposes a peer-to-peer communication protocol, platform and language independent.
(iii) *middlewares* offering different services (grid construction and management, task sharing management

[1]http://www.xtremweb.net
[2]http://www.ourgrid.org

and results gathering). In this last group we can find Globus [9], [10], XtremWeb [11], DIET [12] or ProActive [13].

Whatever the application is, and whatever the means are, a grid based application has to verify some properties to be efficient [14], [15]: (1) sharing hardware and software capacities and capabilities, (2) scheduling the use of these means and the capability of peer-to-peer computation over Internet. F. Capello presented the topical issues of peer-to-peer globalized computation system in [16].

Notice that authors propose partially centralized, or hybrid decentralized architecture, in most cases [17]. For example, DIET is a hierarchically centralized system: global information is maintained in replicated servers, called *Master Agents*, and a given computer organization is managed by *Local Agents*. Also, the BOINC [7] framework consists of two layers that operate under the client-server architecture.

III. CONFIIT MIDDLEWARE

A. FIIT applications

The notion of FIIT applications was defined in [18], and they are composed by a Finite number of Independent and Irregular Tasks (FIIT). We assume that each task satisfies the following features:

- A task cannot make any hypothesis on the execution of any other one. Hence, there is no communication between tasks.
- Execution time of each task is unpredictable. In other words, we cannot have a good estimation of the execution time before a task ends.
- A same algorithm is applied to compute all tasks. Hence, two tasks are distinguished by the set of data they have to process.

The well-known Mandelbrot's set [19] can be seen as a FIIT application: each pixel (or set of pixels) can be computed independently from the others, and its computation time is unpredictable since it depends on the corresponding region. In such a framework, the programmer needs to decide on how to divide the problem into a finite number of independent tasks, and how to compute each individual task.

The choices carried out are significant: decomposition granularity (number of tasks) influences load-balancing quality and impacts the use of the interconnection network [20]. If the problem is divided into too small tasks, scheduling will induce too much communication. Conversely, if the number of tasks is too small in comparison with the number of nodes, resolution will not able to exploit available parallelism. Thus, decomposition has to be adapted to the underlying computing environment (nodes architecture and interconnection network).

B. CONFIIT outline

CONFIIT (Computation Over Network for FIIT), introduced in [2], is a middleware for peer-to-peer computing. It aims to:

- distribute over the network all the tasks obtained by decomposition of a FIIT problem,
- solve each distributed task,
- spread computed results of each task over the network.

Each computer collaborating in a CONFIIT computation is called a *node*. A node is set up with three main basic thread components: a topology and communication manager, a tasks manager and one or several task solvers.

The nodes are connected according to a logical oriented ring, set up and maintained by the topological layer of the system. Basically, each node knows its predecessor and successor. Communication between nodes are achieved using a token, which carries the state of computation around the ring and ensures load balancing between nodes. Task status are exchanged to broadcast local knowledge on all nodes, and thus, to compute an accurate global view of the calculus. At the end of the computation, the ring spreads termination on nodes.

Typically, one *tasks solver* should be launched for each processor on the node. A *task manager* can cope with several *tasks solvers*. Over and above the main thread elements, each node owns a set of data representing the local known state of the global computation.

All CONFIIT parts are written in Java. A node runs a *daemon*, performing topology, communication and tasks managements, and one or several *solvers* in separate Java Virtual Machine. Each solver is dedicated to a specific running computation, and can be duplicated if the application is to be executed by more than one processor.

A node owns the different parameters of the current computations (a list of tasks and associated results). It is able to set up its local parameters of tasks to be computed. At the beginning of a computation, all tasks are marked as *uncomputed*. When a task is completed (locally), its result is stored and will be propagated to the whole ring with the token. To prevent loss of time, the task manager schedules a new task without waiting for the token. But the task could have already been scheduled by another node, and will be *replicated* in the ring. So, the local list of tasks owned by a node is randomly rearranged. Thus, when the task manager picks up the n^{th} task, it *probably* will be different from the n^{th} task on another node [2], [18]. This strategy also guarantees the termination of the computation regardless node faults.

C. Managing the CONFIIT community

To achieve robustness according to computer crashdowns and dynamic evolution, CONFIIT maintains a central structure representing the ring topology. Each node participating in the computation is known by the others. Information is stored in a local structure. The mechanism involves a k-*tolerance* to recover a functional ring when k successive nodes simultaneously fail in a n-sized ring ($k \leq n$).

As data are locally stored, a distributed mechanism must be performed to maintain a global view of the ring in each node. If we assume the token for computational state follows the logical ring clockwise, changes in the ring topology will be carried by a special token (a *service token*) that circulates counter-clockwise. Thus, when a node enters or exits the ring,

nodes that are the most interested by the information (before it clockwise) are informed first.

1) Entering the ring: When a computer wants to enter in the ring, it contacts a running node. The running node communicates it the partial topology of the ring, according to the k factor of tolerance, and the parameters of the problem being solved. With this information, the new node is able to start working on the problem. The server-side of the procedure implies on the change of the local structure of the ring and on the propagation of this new structure to all nodes in the community. For the matter of topology, the requester is inserted after the local node, thus a new node will receive and update its local ring structure directly from its predecessor. Conversely, other nodes are informed of the modification by the *service* token running counter-clockwise. This later one is propagated until it reaches the k^{th} node.

2) Exiting the ring: A node can exit a CONFIIT computation ring for two reasons: *(1)* the user decides to stop computing collaboration; *(2)* a network problem occurred and the node becomes unreachable, or the system crashes. In the former case, a procedure is initiated by the *voluntary* exiting node. In the later one, problem is detected by its predecessor. We mainly focus here in *crashdown* exits, since a voluntary exit follows the same propagation procedure.

In case of computer crashdown, the trouble is detected only when a node attempts to send information toward its successor. Then, it removes the crashed node from its own view of the logical ring, and propagates the deletion to its predecessor. Propagation is stopped by the node that initiates the deletion. If a node crashes after a token reception, but before sending it to its successor, a timeout mechanism ensures the token retransmission.

Same problem can occur for back-propagation tokens: a topology alteration involves a token to circulate back to the ring, but it can be destroyed by a computer crash. This problem is solved by a similar procedure: when a node initiates a propagation of a topology alteration, it launches a timeout delay, and re-send the information if it doesn't receive its own information before the timeout ends.

D. Programming models

Since constraints of a given application could be different and sometimes in contradiction (fault tolerance, efficiency, ...), CONFIIT offers two main programming models: distributed and centralized mode.

The **distributed mode** allows an accurate fault tolerance in the computation since task results are locally stored on each node in the community. Thus, a broken computation can be re-launched using already computed tasks. Fig. 1 shows information exchanges in the community for a distributed application. At first, the launcher sends the computing request to a node. The request is propagated along the community by the token (dotted arrows). During computation, results of individual tasks are propagated across the community (thick dashed arrows) such that each node could locally store all individual results (data blocks). Concurrently to the computations, information on the global computation is exchanged according to the thin arrows.

Another interesting point from this mode is that the launcher only needs to be connected during the initiation phase. At the end of the computation, the global result can be retrieved from any node in the community.

While this mode allows greater flexibility and fault tolerance, it may induce the need of a large storage space on each node. Similarly, as results must be spread to each node, it may overload slow network connections.

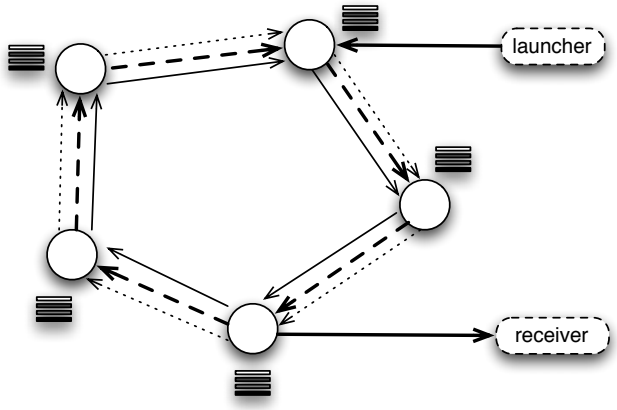

Fig. 1. Distributed mode

The **centralized mode** reduces the global load of storage space and network communication, with the drawback of reducing fault tolerance.

Fig. 2 shows information exchanges in the community for a centralized application. At first, the launcher sends the computing request to a node. The request is propagated along the community by a specific token (dotted arrows) as in the distributed mode, but the launcher must remain connected.

During computation, results of individual tasks are sent to the initial launcher (thick dashed arrows), which has the storage in charge (data blocks). As in the distributed mode, information on the global computation evolution is updated through the token (thin arrows). A crash on a computing node is not a problem for the computation since the main community ring is fault tolerant, but a crash on the launcher will stop the computation because the gathering of information cannot be achieved.

E. Placement of nodes and the ring topology

In this paper, we are especially interested on the impact of nodes placement on the communication performance when using the **distributed mode** of CONFIIT. Because CONFIIT relies on a virtual ring, the token passing mechanism is intrinsically bound to the network speed and the ring circumference. As consequence, it is important to verify the impact of nodes placement in the overall system performance.

Furthermore, in the case of the distributed mode, this virtual ring not only distributes tasks and prevents nodes about their

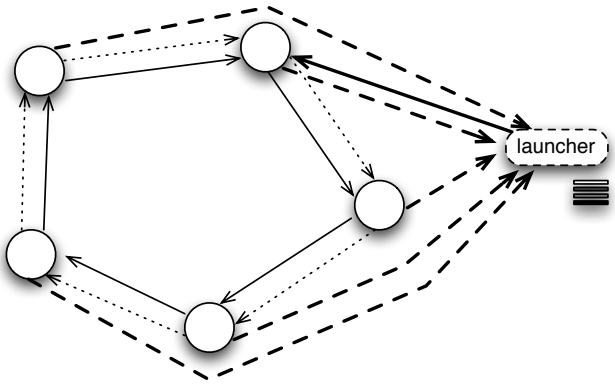

Fig. 2. Centralized mode

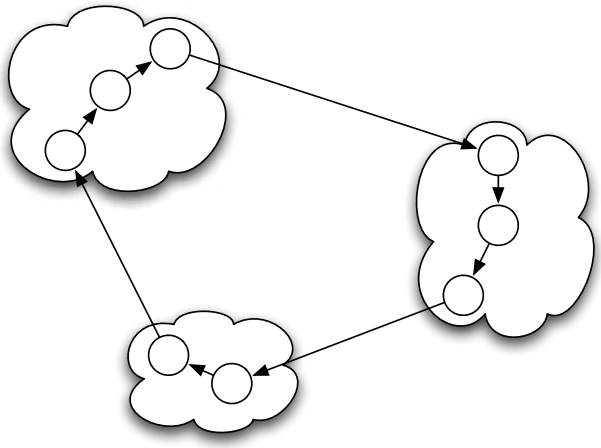

Fig. 3. Geographical distribution

completion, but also spreads the results among the nodes. In this case, the token must also ensure that all nodes receive the results from different tasks. According to the amount of data exchanged, the underlying network performance and the ring round-trip speed may impact the computing performance.

From these aspects, the placement of nodes becomes an important factor to be evaluated. Contrarily to traditional grid applications, which follow a structured hierarchy known as "cluster of clusters" [21], dynamic P2P platforms are subjected to node volatility that may reorder the topology in a small amount of time. In the aspects that concern a ring based topology, this means that nodes can be structured according to a geographic distribution (Fig. 3) but also may face worst scenarios, like when the token hops from cluster to cluster (Fig. 4). This diversity of scenarios makes the token round trip time vary by several orders of magnitude, compromising the performance of a platform that is not prepared to face such variations.

Therefore, in the next section we evaluate how the performance of CONFIIT is affected by nodes placement (and task sizes) in a grid environment.

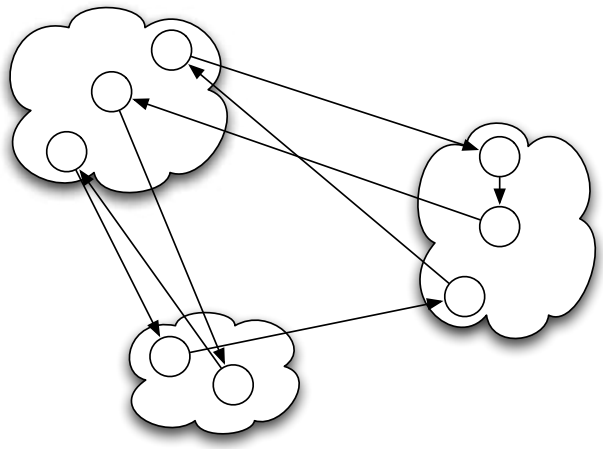

Fig. 4. Unordered distribution

IV. Case Study: the Langford's problem

C. Dudley Langford gave his name to a classic problem of permutation [22], [23]. While observing his son manipulating blocks of different colors, he noticed that it was possible to arrange three pairs of blocks of different colors (yellow, red, blue) in such a way that only one block separates the red pair, two blocks separate the blue pair and finally three blocks separate the yellow one (see Fig. 5).

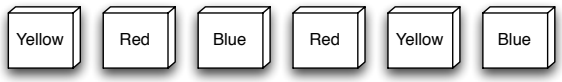

Fig. 5. L(2,3): arrangement for 6 blocks of 3 colors: yellow, red and blue.

The problem has been generalized to any number of colors n and any number of blocks having the same color s. $L(s, n)$ consists in searching for the number of solutions to the Langford problem. In November 1967, Martin Gardner presented $L(2, 4)$ (two cubes and four colors) as being part of a collection of small mathematical games and stated that $L(2, n)$ has solutions for all n such that $n = 4k$ or $n = 4k-1$ for $k \in \mathbb{N}^*$.

Recently, Toby Walsh and Barbara Smith formulated this problem as a Constraint Satisfaction Problem [24], [25]. The Langford Problem has been approached in different ways (discrete mathematics results, specific algorithms, specific encoding, ...).

A. A tree search approach

The Langford problem can be modelized as a tree search problem. In order to solve $L(2, n)$, we consider the tree of height n and width $2n - 2$ (see Fig. 6):

- every node of the tree corresponds to the place in the sequence of the cubes of a determined color;
- to the depth p, the first node corresponds to the place of the first cube of color p in first position and it ith node

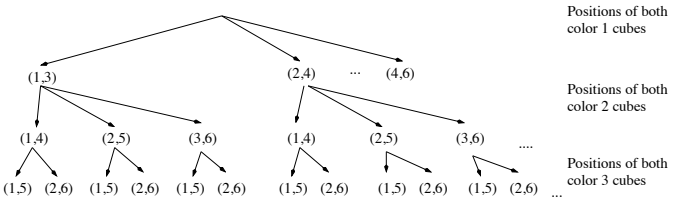

Fig. 6. Search tree for $L(2,3)$.

corresponds to the investment of the first cube of color p in position i, $i \in [1,\ 2n-1-p]$;
- every leaf of the tree symbolizes the positions of all cubes;
- a leaf is a solution if it respects the color constraint defined by the Langford problem.

To be efficient, this algorithm should avoid the recursive tree traversal. Hence, in 2002, an algebraic representation of the Langford problem has been proposed by M. Godfrey. Consider $L(2,3)$ and $X = (X_1, X_2, X_3, X_4, X_5, X_6)$. It proposes to model $L(2,3)$ by $F(X,3) = (X_1X_3 + X_2X_4 + X_3X_5 + X_4X_6) \times (X_1X_4 + X_2X_5 + X_3X_6) \times (X_1X_5 + X_2X_6)$. In this approach, each term represents a position for both cubes of a given color and a solution to the problem is equal to the polynomial coefficient of $X_1X_2X_3X_4X_5X_6$ in the development. More generally, a solution to $L(2,n)$ can be deduced from $X_1X_2X_3X_4X_5...X_{2n}$.

If $G(X,n) = X_1 \dots X_{2n}F(X,n)$ then it has been shown that:

$$\sum_{(x_1,\dots,x_{2n})\in\{-1,1\}^{2n}} G(X,n)_{(x_1,\dots x_{2n})} = 2^{2n+1}L(2,n)$$

So:

$$\sum_{(x_1,\dots,x_{2n})\in\{-1,1\}^{2n}} (\prod_{i=1}^{2n} x_i)\prod_{i=1}^{n}\sum_{k=1}^{2n-i-1} x_k x_{k+i+1} = 2^{2n+1}L(2,n) \quad (1)$$

The computation of $L(2,n)$ is in $O(4^n \times n^2)$ and an efficient long integer arithmetic is needed. This principle can be optimized by taking into account the symmetry of the problem and using the Gray code[26]. By using this approach, M. Godfrey has solved $L(2,20)$ in one week on three PCs in 2002.

It is quite obvious that a parallel version can be derived from Eq.(1). By choosing a value in $\{-1,1\}$ for one or more of the x_i in $\sum_{(x_1,\dots,x_{2n})\in\{-1,1\}^{2n}}$, a set of independent tasks is introduced. Again, a depth level of the parallelization can be defined. At depth level k, the values of x_1, $x_2, \dots, x_k$ are fixed (either 1 or -1). Indeed, at depth level k, a set of 2^k tasks is generated.

B. Current status

At the moment, the instances solved in practice, in a merely combinatorial manner, limit themselves to a small number of colors. In this case, one mentions the instance $L(2,19)$ that was solved in 2 years and a half on a DEC Alpha 300MHz in 1999. In 2002, $L(2,20)$ was solved with the help of a new algorithm and the intensive use of a cluster of 3 PCs during one week. Using Godfrey's algorithm over CONFIIT, we were initially able to solve the problem $L(2,23)$ in about 4 days (april 2004), using a local network with 63 nodes and 85 processors, which represents about 320 days of sequential computation. Later, in 2005, the $L(2,24)$ problem was solved in 94 days in a non-dedicated network with variable size. Using this variable network, which counted at most with 12 nodes (20 processors) at a given moment, we calculated the 46,845,158,056,515,936 possible combinations of the problem, for an equivalent of 850 days of sequential computation [27]. Up today, this is the largest solved instance of Langford's problem[3].

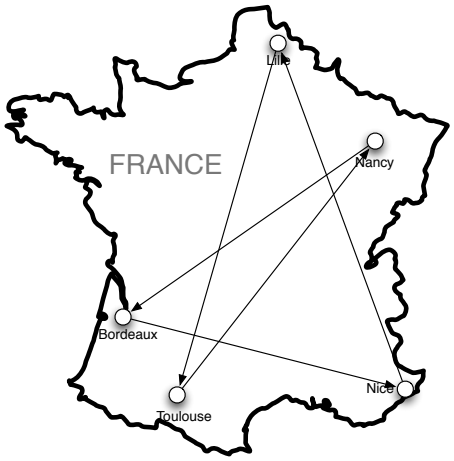

Fig. 7. Location of clusters used in the experiments

V. Experiments

A. Platform Description

In order to conduct our experiments, we used up to 5 clusters from the Grid'5000 experimental platform[4], located in Bordeaux, Nancy, Nice (Sophia Antipolis), Lille and Toulouse (see Fig.7). For a matter of uniformity, all clusters are composed of bi-processor, dual-core machines (AMD Opteron 2218 2.66 GHz, AMD Opteron 285 2.66 GHz or Intel Xeon 5110 1.6 GHz). This way, each machine runs 4 CONFIIT threads, one by core. Machines inside the same cluster are interconnected by a Gigabit Ethernet network while the clusters are connected by a private backbone of 10 Gbps. All nodes run Debian Linux, with kernel version 2.6.26.

B. Experiment scenarios

In order to evaluate the impact of nodes placement on the performance of CONFIIT, we designed a progressive scenario where we scale-up the number of clusters from 1 to 5, while keeping the same number of computing threads.

[3]See http://www.lclark.edu/~miller/langford.html
[4]https://www.grid5000.fr

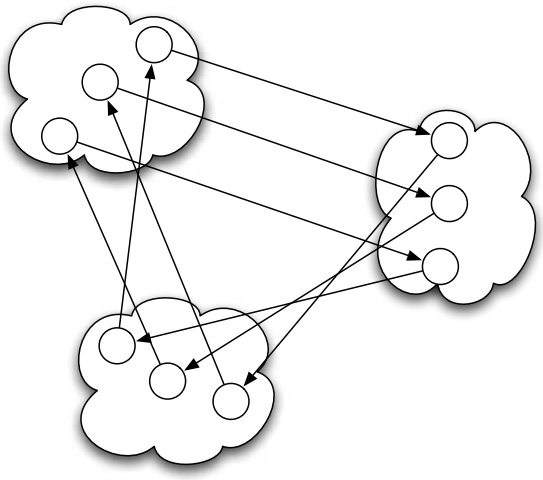

Fig. 8. Worst case distribution

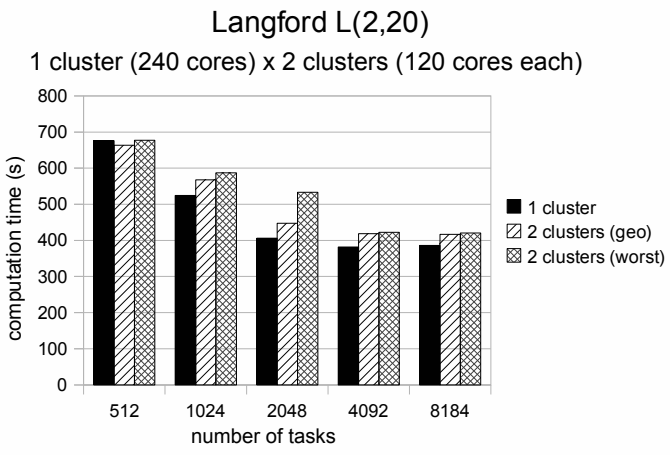

Fig. 9. Comparison between Langford on 1 and 2 clusters

Our objective is to evaluate how CONFIIT performance behaves with the increment on the ring circumference, on both best case (geographically distributed or **geo**, as in Fig. 3) and **worst** case (Fig.8). Indeed, if we consider only the network latency (extracted from the *ping* parameters from Table I), a ring with 60 machines in 5 clusters would take around 46.805ms to do a roundtrip in the best case (**geo**: Nancy→Bordeaux→Nice→Lille→Toulouse) and 535.788ms to do the same roundtrip in the **worst** case. If communication and computation do not overlap efficiently, this roundtrip time may represent a serious performance concern.

In our experiments, we also evaluate the performance of CONFIIT with different task sizes for the Langford's problem. Hence, the parallel computation of Langford's problem can be expressed by a depth level *k*, which generates 2^k independent tasks. A reduced number of large tasks induce less communication among the nodes but increases the size of result blocks to be spread among the nodes. On the other hand, a large number of small tasks may improve the speedup as no task will delay too much the termination of the computation (similar to a *pipeline* effect).

C. Analysis

Table II represents the average of 3 runs for each combination of clusters, topology and number of tasks. The ring topology is constructed in the order Nancy→Bordeaux→Nice→Lille→Toulouse for both **geo** and **worst** scenarios (the worst scenario alternates machines from each cluster, in the same order). For the matter of comparison, we also indicate the execution time in a **single machine**.

Therefore, Fig. 9, 10, 11 and 12 plot the computation time of the same instance of the Langford's problem when comparing 1 cluster and 2, 3, 4 and 5 clusters respectively. From the analysis of these data, we obtain the following insights:

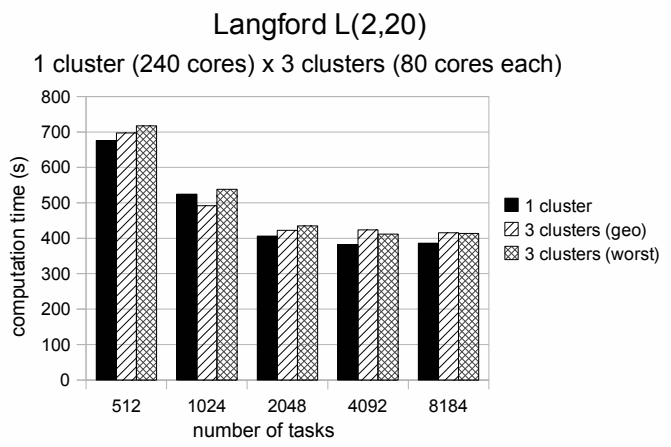

Fig. 10. Langford on 3 clusters

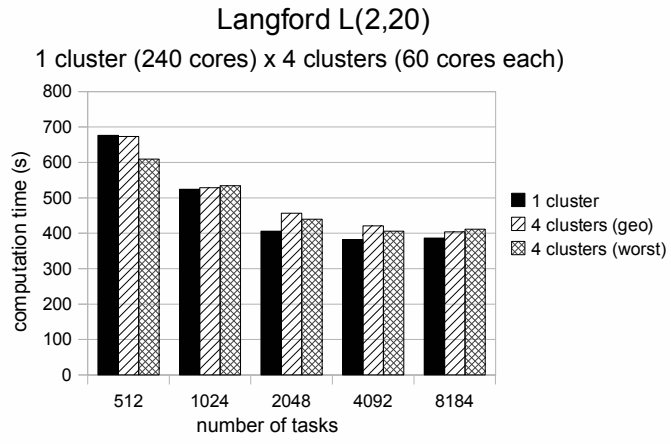

Fig. 11. Langford on 4 clusters

TABLE I
ROUND-TRIP TIME (PING) AMONG NODES FROM DIFFERENT CLUSTERS

	Bordeaux	**Nancy**	**Nice**	**Lille**	**Toulouse**
Bordeaux	0.090ms	18ms	14.9ms	17.2ms	18.6ms
Nancy	18ms	0.100ms	16.8ms	8.82ms	20.6ms
Nice	14.9ms	16.8ms	0.046ms	16.1ms	17.4ms
Lille	17.2ms	8.82ms	16.1ms	0.092ms	19.7ms
Toulouse	18.6ms	20.6ms	17.4ms	19.7ms	0.064ms

TABLE II
COMPUTATION TIMES WHEN VARYING THE NUMBER OF CLUSTERS AND THE NUMBER OF TASKS

	512 tasks (k=9)	**1024 tasks (k=10)**	**2048 tasks (k=11)**	**4092 tasks (k=12)**	**8184 tasks (k=13)**
Single machine	16425.32s	17242.13s	16675.48s	17545.15s	17361.09s
1 cluster	676.263s	524.347s	405.81s	381.576s	386,414s
2 clusters (geo)	663.646s	567.54s	447.152s	418,517s	417,009s
2 clusters (worst)	677.29s	587.003s	533.274s	422.769s	420.739s
3 clusters (geo)	697.262s	492.119s	422.628s	423.568s	415.886s
3 clusters (worst)	717.504s	538.026s	434.973s	412.001s	413.61s
4 clusters (geo)	673.082s	528.628s	456.385s	421.053s	404.126s
4 clusters (worst)	609.69s	534.328s	439.901s	405.943s	411.397s
5 clusters (geo)	654.827s	478.253s	412.802s	389.592s	413.817s
5 clusters (worst)	656.894s	499.64s	408.056s	402.154s	397.587s

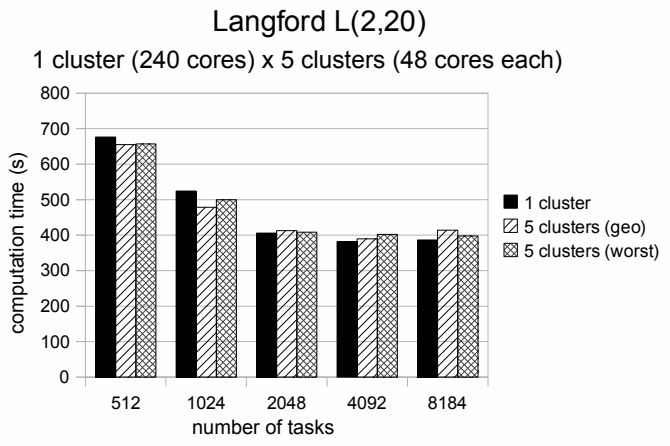

Fig. 12. Langford on 5 clusters

1) Impact of nodes placement: In spite of the difference between the placement policies **geo** and **worst**, the computation times are barely affected by the ring circumference. While the **geo** strategy generally benefits from a small advantage in comparison with the **worst** strategy, this difference seems to vanish (or event to invert) when small tasks are used (e.g.: $k = 12$ or $k = 13$).

This can be explained by the task selection mechanism from CONFIIT. Because CONFIIT nodes randomly select unsolved tasks, nodes are allowed to start new tasks even if the token has not yet returned. With more small tasks, computation of tasks can continue with a small probability of "double work", while waiting for the token update. Also at the end of the computations, the remaining unsolved tasks will be quickly deployed and represent a small computation load that does not delay the termination.

Nevertheless, these results do not exclude performance problems if the inter-cluster networks become too slow in comparison with the local-area network (like the traversing of a PSTN or ADSL link). In Grid'5000, the impact of the wide-area latency is partially compensated by its bandwidth (a 10Gbit/s dark-fiber interconnection).

2) Number of clusters: In spite of the increment in the number of clusters, no slowdown could be observed in the experiments. Also, the ratios between the performances of Langford's algorithm in one or several clusters remain stable, which is an interesting fact. For instance, it seems that the cost associated with the communication, no matter if in a single cluster or through several networks, is sufficiently reduced to be compensated by the computation/communication overlap.

Although these results are encouraging, especially when considering grid environments, we believe that further scalability tests should be necessary to determine the limits of the platform if we plan to deploy experiments in open networks.

3) Impact of task size: We finish the analysis of the experiments by looking on the impact of task sizes on the overall computation time. The Fig. 9, 10, 11 and 12 clearly show that a small number of tasks does not parallelizes as well as when a larger number of tasks is deployed. Actually it is not easy to determine if a task will have a long or small duration, as FIIT tasks are independent and their computation time depends on the evaluation algorithms. Nevertheless, we can estimate that in a combinatorial tree search the more we subdivide the tree, the shorter will be the average computational time of each branch.

This does not mean however that we can subdivide the problem in an infinite number of tasks. Because the token circulation represents a non-negligible barrier, it is important to find a trade-off between the computational time for a task and the communication cost. If we observe the average time

for task in the 5-cluster/geo scenario, for example, we obtain 76.74s for $k = 9$, 28.02s for $k = 10$, 12.09s for $k = 11$, 5.71s for $k = 12$ and finally 3.03s for $k = 13$. Hence, increment (doubling) in the number of tasks gives a speed-up of at least a factor 2 up to $k = 12$. With $k = 13$, however, the average time for task does not decreases as expected, indicating that computation/communication overlap reaches its limits. This time can even increase if we insist to multiply the tasks, as indicated by previous experiences outside this work.

VI. Conclusion

The main purpose of this paper was to evaluate how the geographical disposition of nodes in a grid environment impact on the performance of a ring-based distributed computing platform. Using a distributed solver for the Langford's problem as the evaluation subject, we were able to observe the main factors that influence the computation performance.

We observed therefore that the load balancing scheme used in the CONFIIT distributed framework is little sensitive to the performances of the network. Indeed, tasks computation is almost independent of the token mechanism, as the nodes keep selecting uncomputed tasks even without new updates. Nonetheless, the computation performance in the distributed mode (*i.e.*, the timespan) still depends on the number (and by extension, the size) of the tasks as the token propagates the results of the computed tasks. The experiments show a trade-off between computation and communication overlap that limits the speed-up in some cases. Hence, we can expect a larger timespan in slower networks as the token must propagate all updates to all nodes before termination is detected.

Further experiments will explore CONFIIT scalability behavior in both grid environments and open networks (P2P over Internet), an essential step towards larger problems such as Langford $L(2, 27)$ and $L(2, 28)$.

Acknowledgement

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see *https://www.grid5000.fr*).

This work is partially supported by ANR (*Agence National de Recherche*), project reference ANR 08 SEGI 022.

References

[1] I. Foster and C. Kesselman, *Computational Grids*. Morgan-Kaufman, 1998, ch. 2.

[2] O. Flauzac, M. Krajecki, and J. Fugère, "CONFIIT: a middleware for peer to peer computing," in *The 2003 International Conference on Computational Science and its Applications (ICCSA 2003)*, ser. Lecture Notes in Computer Science, M. Gravilova, C. Tan, and P. L'Ecuyer, Eds. Montréal, Québec: Springer-Verlag, Jun. 2003, vol. 2669 (III), pp. 69–78.

[3] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff, "Charlotte: Metacomputing on the web," in *Proceedings of the 9th Intl. Conf. on Parallel and Distributed Computing Systems*, 1996.

[4] L. F. G. Sarmenta, S. Hirano, and S. A. Ward, "Towards bayanihan: builing an extensible framework for volonteer computing using java," in *Proc. ACM Workshop on Java for High-Performance Network Computing*, 1998.

[5] D. Anderson, S. Bowyer, J. Cobb, D. Gedye, W. T. Sullivan, and D. Werthimer, "A new major seti project based on project serendip data and 100,000 personal computers," in *In Astronomical and Biochemical Origins and the Search for Life in the Universe, Proc. of the Fifth Intl. Conf. on Bioastronomy*, 1997.

[6] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An experiment in public-resource computing," in *Communications of the ACM*, Nov. 2002, vol. 45, N. 11, pp. 56–61.

[7] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *5th IEEE/ACM International Workshop on Grid Computing*, Nov. 2004.

[8] S. Li, "Jxta 2: A high-performance, massively scalable p2p network," IBM developerWorks, Tech. Rep., Nov. 2003.

[9] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[10] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, and M. Wilde, "Globus toolkit support for distributed data-intensive science," in *Computing in High Energy Physics (CHEP '01)*, Sep. 2001.

[11] G. Fedak, C. Germain, V. Néri, and F. Cappello, "XtremWeb: A generic global computing system," in *CCGRID2001, workshop on Global Computing on Personal Devices*. IEEE Press, 2001.

[12] 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. Lecture Notes in Computer Science, B. Monien and R. Feldmann, Eds., vol. 2400. Paderborn, Germany: Springer-Verlag, Aug. 2002, pp. 907–910.

[13] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici, *Grid Computing: Software Environments and Tools*. Springer-Verlag, January 2006, ch. Programming, Deploying, Composing, for the Grid. [Online]. Available: http://www-sop.inria.fr/oasis/proactive/doc/ProgrammingComposingDeploying.pdf

[14] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, 2002.

[16] F. Cappello, "Calcul global pair à pair : extension des systèmes pair à pair au calcul," *La Lettre de l'IDRIS*, vol. 4, pp. 14–25, Apr. 2002.

[17] S. Androutsellis-Theotokis, "A survey of peer-to-peer file sharing technologies," Athens University of Economics and Business, Tech. Rep., 2002.

[18] M. Krajecki, "An object oriented environment to manage the parallelism of the FIIT applications," in *Parallel Computing Technologies, 5th International Conference, PaCT-99*, ser. Lecture Notes in Computer Science, V. Malyshkin, Ed. St. Petersburg, Russia: Springer-Verlag, Sep. 1999, vol. 1662, pp. 229–234.

[19] A. Dewdney, "Récréation informatiques : un microscope informatique pour observer l'objet le plus complexe jamais défini par les mathématiciens," *Pour la Science*, vol. 26, pp. 87–93, Oct. 1985.

[20] M. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on hightly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, pp. 979–993, 1993.

[21] L. A. Steffenel and G. Mounié, "A framework for adaptive collective communications for heterogeneous hierarchical computing systems," *Journal of Computer & System Sciences, special issue on Performance Analysis and Evaluation of Parallel, Cluster, and Grid Computer Systems*, vol. 74, no. 6, pp. 1082–1093, 2008.

[22] M. Gardner, *Mathematics, Magic and Mystery*, 1956.

[23] J. E. Simpson, "Langford sequences: perfect and hooked," *Discrete Math*, vol. 44, no. 1, pp. 97–104, 1983.

[24] T. Walsh, "Permutation problems and channelling constraints," APES Research Group, Tech. Rep. APES-26-2001, January 2001. [Online]. Available: http://www.dcs.st-and.ac.uk/ apes/reports/apes-26-2001.ps.gz

[25] B. Smith, "Modelling a Permutation Problem," in *Proceedings of ECAI'2000, Workshop on Modelling and Solving Problems with Constraints, RR 2000.18*, Berlin, 2000. [Online]. Available: http://www.dcs.st-and.ac.uk/ apes/2000.html

[26] S. Ranka and S. Sahni, *Hypercube Algorithms with Applications to Image Processing and Pattern Recognation*. Springer-Verlag, 1990.

[27] C. Jaillet, M. Krajecki, and A. Bui, "Parallel tree search for combinatorial problems: a comparative study between openmp and mpi," *Studia Informatica Universalis*, vol. 4, no. 2, pp. 151–190, December 2005.