Scalable Performance Predictions of Distributed Peer-to-Peer Applications

Bogdan Florin Cornea*, Julien Bourgeois*, The Tung Nguyen†, Didier El-Baz†

*DISC, FEMTO-ST Institute, University of Franche-Comté, 1 cours Leprince Ringuet; 25201 Montbéliard; France.
{bogdan.cornea, julien.bourgeois}@univ-fcomte.fr

†CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France.
Université de Toulouse, LAAS, F-31400 Toulouse, France.
{ttnguyen, elbaz}@laas.fr

***Abstract*—Recently, a new environment for high performance peer-to-peer distributed computing was proposed. This environment, named P2PDC, addresses stable or volatile systems communicating in a decentralized manner using the self-adaptive protocol P2PSAP. P2PDC is devoted to task parallel applications like numerical simulation problems or optimization problems solved via parallel or distributed iterative algorithms. For distributed applications meant to run with P2PDC, a performance prediction tool named dPerf was proposed. dPerf combines static and dynamic analysis with trace-based simulation to provide scientist with information about the execution of their large scale numerical simulation applications. dPerf addresses real parallel and distributed numerical simulation and optimisation applications written in C, C++ or Fortran for P2PDC.**

This paper introduces an enhancement of the dPerf tool which provides scalable performance prediction results. Scaling is done with respect to (i) network configuration and (ii) number of peers. Scaling predictions based on network configuration is achieved through trace-based simulation, where various architectures can be studied. Scaling predictions based on the number of peers implies analyzing the communication topology and modifying trace files prior to simulation.

We present experimental results obtained for the obstacle problem, a C/P2PDC implementation of the code used in mechanics and finance. Prediction for this application is computed under real conditions, with a reduced slowdown and by providing user with scalable results.

***Keywords* - Scalability, performance prediction, static analysis, dynamic analysis, peer-to-peer computing, high performance computing, task parallel model, numerical simulation.**

I. INTRODUCTION

High Performance Computing (HPC) architectures are undergoing a continuous evolution. The increase in number of computing nodes, or the choice of a new network topology which provides better performance usually represent a non-negligible investment. The development of parallel and distributed applications for emerging computing architecture requires adaptation of existing applications, otherwise the computing resources risk being inefficiently used. Peer-to-peer (P2P) architectures come as an alternative type of computing platform to stable HPC systems. With the recent development of P2PDC [1], a decentralized peer-to-peer environment devoted to task parallel applications necessitating frequent data exchanges between peers, like the solution of numerical simulation problems via parallel iterative methods, scientists may port their HPC applications to P2P testbeds for exploiting computing power at a more convenient cost.

For best usage of the computing potential of HPC systems, a key role is held by performance prediction tools. For this reason, dPerf [2] was recently proposed to the HPC community as a tool for predicting performance of distributed applications which communicate either via the P2PSAP protocol [3], employed by P2PDC, or using MPI. In this way, scientists willing to port their HPC codes to P2P testbeds have access to the computing environment as well as to the prediction tool giving them estimates on their application behavior.

This paper introduces the scalability of predictions calculated with dPerf.

From a prediction viewpoint, scientists expect specialized methods to evaluate HPC applications under conditions close to reality. For most of proposed tools, this implies executing the compiled application which leads to an increased slowdown of the prediction method. Moreover, execution of the analyzed code imposes constraints regarding the scalability of the prediction. dPerf was first introduced in this context but only some aspects have previously been addressed. This paper introduces new features of dPerf which extend its functionality from HPC benchmarks to a large number of real numerical simulation and optimisation applications considered under real conditions. One such application is the obstacle problem which occurs in finance and mechanics. The real conditions are taken into account by considering compiler optimizations when computing the prediction. The novelty in dPerf features is the scalability of results which extends previous work on dPerf [2, 4]. The development of dPerf is a result of a thorough study of the state of art in performance prediction tools which shows that several aspects are hardly addressed by existing methods. These aspects are related to (i) the slowdown of a method, (ii) the support for decentralized high performance P2P computing, (iii) the support for multiple programming languages by

978-0-7695-4749-7/12 $26.00 © 2012 IEEE
DOI 10.1109/HPCC.2012.34

the same tool, and (iv) the analysis of applications using communication paradigms other than MPI.

Relevant related work is presented in section II, showing the main characteristics of existing methods for P2P computing and for HPC performance prediction in general. Section III of this paper presents the peer-to-peer decentralized computing environment P2PDC, then the scalability aspect of dPerf is introduced in section IV. A case study on the obstacle problem is presented in section V. We conclude this contribution with section VI, where we also present our perspectives on future improvements to P2PDC and dPerf.

II. Related work

Performance prediction methods have always accompanied the evolution in computing systems and applications. Most prediction tools are lagged behind by today's computing architectures.

The usability of the numerous existing performance prediction tools varies from providing developers with an insights on their application behavior, to assisting scientists in better choosing future computing system configuration. We can classify, performance prediction tools as: *analytical* [5–7], *profile-based* (based on compilers and instrumentation tools) [8, 9], and *hybrid* [10–14]. The hybrid methods are a combination of analytical- and profile- based. Our tool, dPerf, belongs to the latter category.

Most research works for predicting application performance address single-processor systems, are developed for specific applications, or are executed in centralized environments. Previous performance prediction tools do not address decentralized peer-to-peer environments. For this reason, we developed dPerf based on existing analysis and simulation tools such as ROSE and Simgrid, which we adapted and extended to the peer-to-peer decentralized computing environment P2PDC. To the best of our knowledge, P2PDC is the first decentralized computing environment designed for peer-to-peer HPC applications with frequent direct communications between peers and dPerf is the only prediction tool that can analyze the performance of parallel or distributed applications written for the P2PDC environment. Performance prediction tools developed prior to dPerf and to the existence of P2PDC environment did not take into account the decentralization in the logical topology of computing systems.

III. P2PDC

In this section, we recall briefly some features of the decentralized version of the P2PDC environment. The reader is referred to [4] for more details on P2PDC. We recall that P2PDC relies on a reduced set of communication operations (P2Psend, P2Preceive and P2Pwait) and that the programmer cares only about the choice of distributed iterative scheme of computation (synchronous or asynchronous) he wants to be implemented via P2PDC and does not care about the communication mode between any two peers. The programmer has also the possibility to select a hybrid iterative scheme of computation whereby computations are synchronous locally and asynchronous at the global level. P2PSAP chooses dynamically the most appropriate communication mode between any two peers according to decisions made at application level like schemes of computation and elements of context like network topology at transport level. The decentralized version of P2PDC is based on a hybrid topology manager and a hierarchical task allocation mechanism which make P2PDC more scalable. In the sequel, a *task* is a computation submitted to P2PDC and a *subtask* is a part of the computation assigned to a given peer.

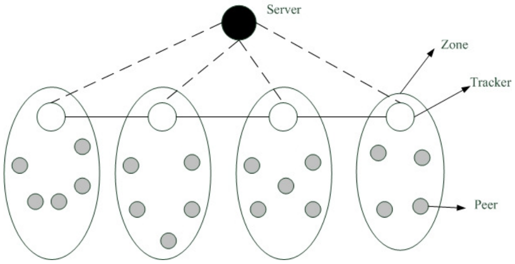


Figure 1: General topology architecture.

A. Hybrid topology manager

The topology manager of the decentralized version of P2PDC is based on a hybrid architecture. This hybrid architecture is simple and ensures scalability and efficient peer collection for computation.

1) General topology architecture: Figure 1 illustrates the general topology architecture. It consists of a Server, Trackers and Peers.

- Server manages informations regarding trackers connection / disconnection. It is the contact point of new nodes joining overlay network for the first time. When trackers or peers have no contact to join overlay network, they contact the server in order to receive a list of closest connected trackers, then they connect to trackers in the received list. The server can also store statistic information regarding connection/disconnection time, resources donated/consumed of all nodes in the overlay network.
- A tracker manages informations regarding a set of peers, called a zone. It collects statistic information regarding connection/disconnection time, resources donated/consumed in his zone and periodically sends these data to server.
- Peers are donors of computational resources. Peers are grouped in zones and managed by the tracker of the zone.

Trackers topology is a line (see Figure 2). Each tracker Ti maintains a set of closest trackers Ni in order to avoid

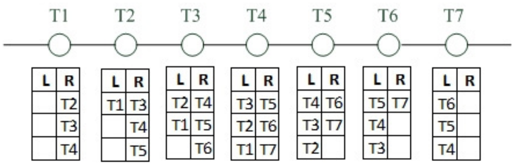


Figure 2: Trackers topology.

the case where some trackers are isolated. There are in the set Ni, $|Ni|/2$ closest trackers having IP address greater than IP address of owner tracker and $|Ni|/2$ closest trackers having IP address smaller than IP address of owner tracker. Moreover, each tracker maintains connection with the closest tracker on the right side and the closest tracker on the left side.

2) IP-based proximity metric: In the literature, there are several proximity metrics that can be used in order to calculate the proximity between peers in the network such as IP path length, AS path length, geographic distance, and measures related to RTT, etc (see [15]). Each metric has its own advantages and drawbacks. We have chosen IP-based proximity metric since it makes use of local information (IP address) to calculate the proximity, hence it does not consume network resource and it is faster than other metrics.

3) Initial topology: We assume that the system has initially a server and some trackers managed by system administrator. These nodes are the core of the system and are on-line permanently. When the number of peers increases, the system administrator chooses some reliable volunteers (peers) to become trackers. The choice of trackers relies on on-line time, i.e. volunteers peers with largest on-line time are chosen. Moreover, trackers are chosen in order to ensure that the number of peers in the different zones is well balanced. When the P2PDC environment is downloaded and installed at a node, the IP address of the server and a list of trackers are set and stored in the local memory. This tracker list will be updated when node joins the overlay network.

B. Hierarchical task allocation

When the submitter has collected enough peers, it divides peers into groups based on proximity; in each group, a peer is chosen by submitter to become a coordinator that will manage others peers in the group. The number of peers in a group cannot exceed *Cmax* in order to ensure efficient management of coordinator. We have chosen *Cmax = 32*. The submitter sends peers list of a group to the coordinator. Then, the coordinator connects to all peers in its group and sends a ”reverse” message to peers. When a peer is reserved for a computation, it sends a message to its tracker to inform that it is not free anymore. Figure 3 illustrates the allocation graph.

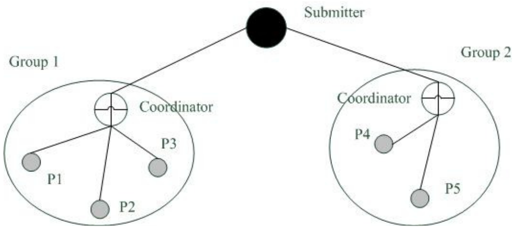


Figure 3: Allocation graph.

The submitter decomposes task into subtasks and sends subtasks to groups coordinators. Subtasks are then sent by coordinators to peers. Subtasks results are sent in reverse direction, i.e. peers send their subtask result to coordinator, then coordinator transfers them to the submitter.

IV. Scalable performance predictions

We propose a novel method for obtaining scalable predictions. Scaling the performances of a computing architecture has direct impact on the application execution. Studying how an applications scales with respect to the architecture is one of the main concerns of application developers in the HPC domain. Our method gives results which can be scaled to any number of processes and to any network configuration. The scalability of predictions made by dPerf relies on the usage of execution traces and of the trace-based parametrized network simulator SimGrid [16].

A. Prerequisites

Several libraries and frameworks are required by dPerf. These prerequisites are necessary for obtaining accurate results which take into account the compiler optimization levels, as well as for reducing dPerf slowdown and supplying users with scalable results.

Our contribution to the performance prediction category of tools uses a static approach for analysis and instrumentation of an input source code. For this we use Rose [17], a compiler framework which supports multiple programming languages, such as C, C++ or Fortran, and provides a front-end, mid-end and back-end to develop custom static analysers. We use the front-end for parsing source codes and for obtaining intermediate representations such as the Abstract Syntax Tree or the System Dependence Graph. Rose mid-end is useful for traversal and transformation of the intermediate representations, while the back-end is essentially designed for unparsing an AST into a transformed source code, correct from syntax point of view.

We propose performance predictions that give accurate results and consider a real context. For this, our contribution makes use of hardware counters. These registers are now the main tools for measurement since they are present in most processing units. Performance registers are very accurate and the stored information is accessible via interfaces such as perfmon [18], perfctr [19] or PAPI [20, 21]. Our focus is on the use of PAPI because it provides low and high level interfaces which introduce a reduced noise in the measured system. Details about dPerf usage of hardware counters were presented previously in [2] (section III) and [4] (section III.D.2).

Scaling of performance predictions with dPerf is achieved (i) through simulation and (ii) by identifying the communication topology. dPerf makes use of a trace-replay module available from SimGrid, a framework for building custom simulators. By relying on SimGrid for replying traces, we can (i) use any communication protocol, (ii) vary the system parameters, (iii) study various network configurations, and (iv) scale the number of computing nodes. Trace-based simulations output the result extremely fast (see Figure 10 from [2]), the cost ($t_{simulation}$) being ten to one hundred times faster than actual application execution ($t_{real\,execution}$) or the total time for obtaining a prediction result ($t_{prediction}$).

Another important role in scaling the prediction results is identifying the logical topology used by the analyzed application. This is done with methods available from Rose compiler. dPerf is able to identify the communication patterns of certain application types in a static manner. For this, dPerf uses the AST and SDG representation and it calculates the neighbors of each process according to our rules. In this paper we address the following logical topologies: master-worker, 2D mesh and 3D torus.

B. Prediction under real conditions

For accurate predictions we consider the different levels for optimizing a code at compilation time. This is achieved by automatic instrumentation, compilation and execution.

The automatic instrumentation is implemented in dPerf based on Rose Compiler Framework. We chose to implement our contribution using Rose due to its support for multiple languages, its front-end parser, the intermediate representations and the back-end unparser. This powerful framework allows creating intermediate representations among which we mention the Abstract Syntax Tree (AST) and the System Dependence Graph (SDG). dPerf searches for relevant instructions in the AST and proceeds to their instrumentation. The SDG is useful for solving some data dependencies such as the propagation of variables throughout the program.

During the static analysis, dPerf applies a technique of benchmarking by blocks of instructions, first introduced in [2]. This technique separates the code into computation and communication phases and yields accurate results with a slowdown of approximately one. The slowdown is one of the factors characterizing prediction tools. It is defined as

$$slowdown_{per\,process} = \frac{t_{prediction}}{t_{real\,execution} \times No.of\,processes} \quad (1)$$

with $slowdown_{per\,process}$ the slowdown expressed per simulated process, $t_{prediction}$ the time necessary for obtaining a prediction, $t_{real\,execution}$ the analyzed application execution time. A value equal to one means that the prediction takes as much time as the real execution of the analyzed application (see $t_{normal\,execution}$ and $t_{prediction}$ curves in Fig.4). The slowdown shows by how much a prediction tools is slower than the actual execution time of the analyzed application. For this reason, a prediction should tend to be lower or equal to one. Otherwise, the efficiency of the prediction tool decreases by spending more time computing the prediction than to actually run the analyzed code. Most performance prediction tools have a slowdown greater than one, i.e.[22, 23]. This places our implementation in the group of rapid prediction tools, such as [14, 24] .

C. Reducing the slowdown

Knowing that an important metric for classifying performance prediction tools is the slowdown, we previously proposed an optimized block benchmarking technique presented in [2]. This method computes a threshold value which, together with information identified in SDG (see section IV-B) helps reducing the number of iterations for out-most loops. By reducing the number of iterations in relevant loops, the application behavior remains unchanged while the run time is shortened. The $t_{threshold\,prediction}$ curve in Fig. 4 depicts the time for obtaining a prediction with dPerf using optimized block benchmarking ($t_{threshold\,prediction}$) with respect to real execution ($t_{real\,execution}$) and prediction using simple block benchmarking ($t_{prediction}$). It can be noticed that $t_{threshold\,prediction}$ is much smaller than $t_{real\,execution}$, hence the slowdown is less than 1. This places dPerf in the group of tools such as [9], which are characterized by a gain rather than by a slowdown.

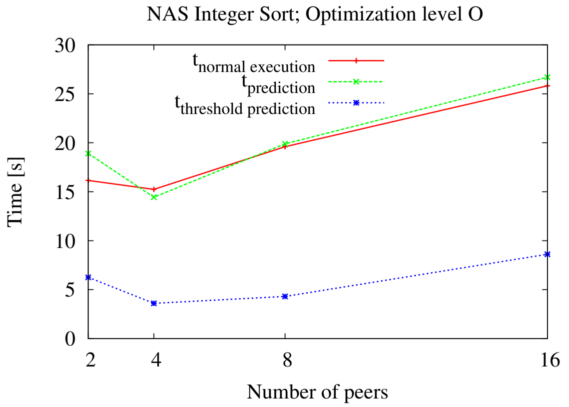


Figure 4: Measured time for the NAS IS benchmark (Class A); duration of the prediction process with simple block benchmarking; duration of the prediction process with optimized block benchmarking technique.

D. Scaling the prediction

We aim at predicting performance for a number of nodes different from the one of the host architecture. This allows the study of the impact of the variation of the number of nodes on application performance. Scaling the prediction is done with respect to system (i) network configuration and (ii) number of computing nodes. It is achieved through simulation and identification of communication topology. Scalability from network point of view was presented in [4], but the current paper proposes an improved performance prediction method with respect to system network configuration and node computing power.

A first way of expressing application performance is with respect to network configuration. For this, dPerf relies on simulations done using SimGrid MSG module. This allows studying various network configurations for homogeneous or heterogeneous systems. The different network types are defined in terms of computing power of each machine, as well as network fabric characteristics, i.e. links, bandwidth, latency. The computing power of participant nodes can be customized so that it can match the trace files. In our implementation, machine power is taken from trace files and it is expressed in nanoseconds. The cost for studying different network configurations is very low due to the simulation based on trace files. These traces are obtained upon execution of the instrumented source code. SimGrid allows users to choose the desired format for the traces, permitting the definition of handlers for this custom format.

Another way for expressing application performance is with respect to the number of computing nodes. We distinguish three phases in dPerf: (i) identifying the communication topology, (ii) modifying traces, and (iii) simulating performances based on trace files. For simplicity, we denote by $dPerf_{topolog}$the communication topology identified with dPerf.

Identifying the communication topology used by an application requires source code analysis. Depending on complexity and resolution of data dependency, applications may require static or dynamic analysis. The current approach addresses only those applications having statically identifiable parameters. This includes constants and variables which are independent of the runtime data. The three virtual topologies identified by our method are (i) master-worker, (ii) 2d mesh and (iii) 3d torus. The *master-worker* communication topology is identified according to formula (1) from [25]. We recall that a program P follows a master-worker paradigm if and only if:

$$\forall k \in [1; ntasks], k \in \mathbb{N},\, \nexists\, v_j / v_i v_j \in E(G_i)\; with\; j \neq 0 \quad (2)$$

with $G_k(V, E)$ are k directed graphs with $V(G_k)$ representing the tasks and $E(G_k)$ the communications of program P. $G_k(V, E)$ represents therefore the communication scheme of task number k. For this, the application must use a *star* communication pattern having only one element at the center of the star, the master process. By deriving the above-presented formula, we identify 2d mesh and 3d torus topologies using the position of a current node with respect to its communication neighbors. Knowledge about application logical topology is necessary for changing the scale of the computing architecture, hence this step must be successfully completed before proceeding to trace modification and simulation.

Modifying traces is the second step towards the scaling of predictions with respect to the number of computing nodes. The scaling of an application's performances is tightly connected to its logical topology. Let N be the number of nodes on the host architecture and N' the number of nodes of the scaled system, with $N' < N$. If $dPerf_{topolog}$is master-worker, we modify traces from N peer such that

$$N' = \begin{cases} N + i & , N = j \\ 2^{k+p} & , N = 2^k \\ N + 2 \times k \times p & , N = 2 \times k \end{cases} \quad (3)$$

$\forall i, j, k, p > 0,\ i, j, k \in \mathbb{N}$, with i the number of extra peers when any number j of peers can be used; 2^{k+p} or $2 \times k \times p$ the number of additional peers when the application requires a multiple of 2^k or $2 \times k$ peers respectively. If $dPerf_{topolog}$is 2d mesh, scaling from N to N' is done according to the following formulae:

$$N' = N + c \times i,\ \forall i > 0,\ i \in \mathbb{N} \quad (4)$$

for adding i rows to the mesh, with c the number of peers in each row;

$$N' = N + l \times i,\ \forall i > 0,\ i \in \mathbb{N} \quad (5)$$

for adding j columns to the mesh, with l the number of peers in each column. When dPerf identifies a 3d torus topology, we extend the formulae presented for 2d mesh such that:

$$\forall i > 0,\ i \in \mathbb{N}, N' = \begin{cases} N + c \times i, to\, add\, i\, rows \\ N + l \times i, to\, add\, i\, columns \\ N + p \times i, to\, add\, i\, planes \end{cases} \quad (6)$$

where c, l and p are the number of columns, rows and planes respectively.

Simulating performances using trace files is the last step in scaling performance predictions with dPerf. This requires the use of MSG, module available in SimGrid, the trace replay process having been previously presented in this section (see the performance scaling with respect to network configuration).

V. Experiments

In this section, we present (i) scalable performance prediction results made by dPerf for a real application, i.e. obstacle problem, ported to the P2PDC environment, as well as (ii) the performance of P2PDC.

A. The obstacle problem

The application we consider, i.e. the obstacle problem, belongs to a large class of numerical simulation problems (see [26]). The obstacle problem occurs in many domains like mechanics and financial mathematics, e.g. Black-Scholes problem for options pricing. We measured the time spent by this code in computation and communication, the ratio being shown in Fig. 5. The ratio varies with the number of peers. However, the test code remain balanced from communication-computation viewpoint, hence allowing us to analyze its performance with respect to network configuration and number of peers.

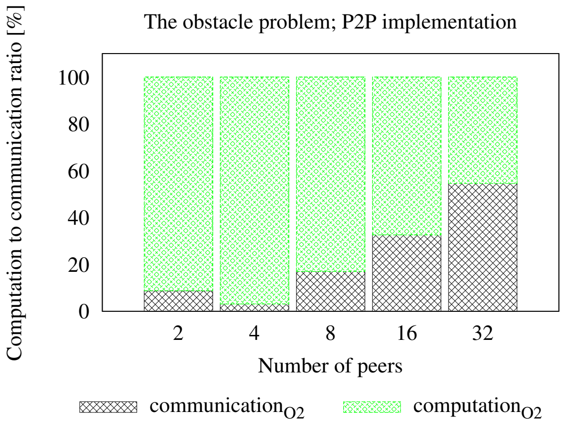


Figure 5: Network and CPU time for the obstacle problem compiled with GCC, optimization level 2.

1) Problem formulation: We briefly present the problem. In the stationary case, the obstacle problem can be formulated as follows:

$$\begin{cases} Find\ u^*\ such\ that \\ A.u^* - f \geq 0, u^* \geq \phi\ everywhere\ in\ \Omega, \\ (A.u^* - f)(\phi - u^*) = 0\ everywhere\ in\ \Omega, \\ B.C., \end{cases}$$

where $\phi \in \mathbb{R}^2 (or\ \mathbb{R}^3)$ is an open set, A is an elliptic operator, ϕ a given function and $B.C.$ denotes the boundary conditions on $\partial\Omega$.

There are many equivalent formulations of the obstacle problem in the literature like complementary problem, variational inequality and constrained optimization problem. Reference is made to [26] for more details. We concentrate here on the following variational inequality formulation:

$$\begin{cases} Find\ u^* \in K such\ that \\ \forall v \in K, \langle A.u^*, v - u^* \rangle \geq \langle f, v - u^* \rangle, \end{cases}$$

where K is a closed convex set defined by

$$K = v | v \geq \phi\ everywhere\ in\ \Omega,$$

and $\langle ., . \rangle$ denotes the dot product $\langle u, v \rangle = \int uvdx$

2) Fixed point problem and projected Richardson method: The discretization of the obstacle problem leads to the following large scale fixed point problem whose solution via distributed iterative algorithms (i.e. successive approximation methods) presents many interests.

$$\begin{cases} Find\ u^* \in V\ such\ that \\ u^* = F(u^*), \end{cases} \tag{7}$$

where V is an Hilbert space and the mapping $F : v \rightarrow F(v)$ is a fixed point mapping from V into V. Let α be a positive integer, for all $v \in V$, we consider the following block-decomposition of v and the associated block-decomposition of the mapping F for distributed implementation purpose:

$$\begin{aligned} v &= (v_1, \ldots, v_\alpha) \\ F(v) &= (F_1(v), \ldots, F_\alpha(v)). \end{aligned}$$

We have $V = \Pi_{i=1}^{a} V_i$, where V_i are Hilbert spaces; we denote by $\langle ., . \rangle_i$ the scalar product on V_i and $|.|_i$ the associated norm, $i \in \{1, \ldots, \alpha\}$; for all $u, v \in V$, we denote by $\langle u, v \rangle = \sum_{i=1}^{\alpha} \langle u_i, v_i \rangle_i$, the scalar product on V and $|.|$ the associated norm on V. In the sequel, we shall denote by A a linear continuous operator from V onto V, such that $A.v = (A_1.v, \ldots, A_\alpha.v)$ and which satisfies:

$$\forall i \in \{1, \ldots, \alpha\}, \forall v \in V, \langle A_i.v, v_i \rangle \geq \sum_{j=1}^{\alpha} n_{i,j} |v_i|_i |v_j|_j, \tag{8}$$

where

$$N = (n_{i,j})_{i \leq i, j \leq \alpha}\ is\ an\ M - matrix\ of\ size\ \alpha \times \alpha \tag{9}$$

The reader is referred to [27] for the definition of $M - matrix$. Similarly, we denote by K_i, a closed convex set such that $K_i \subset V_i, \forall i \in \{1, \ldots, \alpha\}$, we denote by K, the closed convex set such that $K = \Pi_{i=1}^{a} K_i$ and b, a vector of V that can be written as: $b = (b_1, \ldots, b_\alpha)$. For all $v \in V$, let $P_K(v)$ be the projection of v on K such that $P_K(v) = (P_{K_1}(v_1), \ldots, P_{K_\alpha}(v_\alpha))$, where P_{K_i}denotes the mapping that projects elements of V_i onto $K_i, \forall i \in \{1, \ldots, \alpha\}$. For any $\delta \in \mathbb{R}, \delta > 0$, we define the fixed point mapping F_δ as follows (see [26]).

$$\forall v \in V, F_\delta(v) = P_K(v - \delta(A.v - b)). \tag{10}$$

3) Parallel projected Richardson method: We consider the distributed solution of fixed point problem (7) via projected Richardson method combined with several schemes of computation. In this paper, we study essentially synchronous iterative schemes of computation. Nevertheless, we present and briefly analyze a first series of computational results for asynchronous and hybrid schemes of computation at the end of this section [28, 29].

B. Platform

Experiments are carried out on Grid'5000 testbed [30], the French grid platform, that is composed of 2970 processors with a total of 6906 cores distributed over 9 sites in France. All of them have at least a Gigabyte Ethernet network for local machines. Nodes between the different sites range from 2.5 Gflops up to 10 Gflops. Sites of Grid 5000 have several clusters with different performances.

For experimenting with our performance prediction implementation, one part of the available resources of Grid'5000 are used, i.e. 2^n peers of the *Bordeplage* cluster [31], with $n \in \{1, 2, 3, 4, 5\}$. On each working node, only one core is employed, regardless of the total number of available cores per node. The nodes are Intel Xeon EM64T 3GHz, 1 MB L2 cache, 2 GB Memory.

C. Computational experiments

Using the platform described in section V-B, we run experiments with dPerf and the C/P2PDC implementation of the obstacle code, by choosing a 3D problem with size $64 \times 64 \times 64$. In Stage-1 of our experiment, we measure the execution time of the test code (denoted $t_{normal\,execution}$). Then, dPerf applies a static analysis and code transformation technique. It is based on the abstract syntax tree created with ROSE and it instruments the source code as to separate sequential instruction blocks from the communication calls. By applying the block benchmarking technique briefly described in sections IV-B and IV-C, an instrumented code is obtained and executed. The outcome is a set of trace files which are passed to the network simulator SimGrid and we obtained a prediction with dPerf ($t_{predicted}$). The prediction is compared to the measured time to see the precision of our tool with respect to the reference time. The measurement, the prediction and the error percentage are shown in Fig. 6. We notice that for our study on up to 32 peers, the error is under 15 percent.

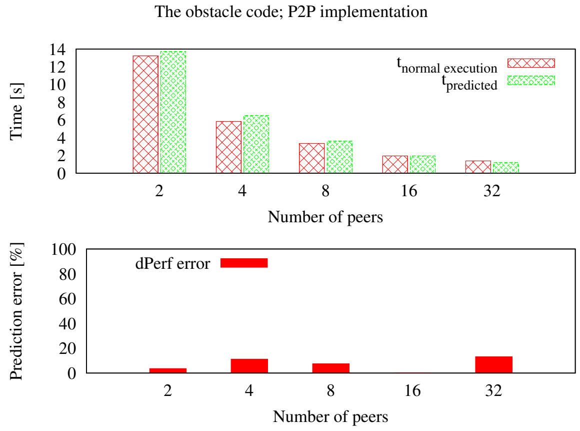


Figure 6: Measured time compared to predicted time (top), and dPerf error percentage (bottom).

In Stage-2, after having analyzed the accuracy of our prediction method implemented in dPerf, we study the network scaling capabilities of our tool. The curves in Fig.7 depict the performance of the tested application if we decide to modify the network configuration. First, we get the reference time $t_{normal\,execution}$ by measuring the real execution of the test code. The same reference time is reused in figures 6, 7, and 8. Second, we do a precision test by predicting performance for the same type of computing system as the one used for obtaining the reference measurements ($t_{predicted\,Grid5000}$)). The predicted values for $t_{predicted\,Grid5000}$ are the same as those in green in Fig.6. Then, we predict application performance for two different network configurations: (i) LAN type ($t_{predicted\,LAN}$), and DSL type ($t_{predicted\,daisy(xDSL)}$). Not only we can study performance over various potential networks, but we can also find alternatives to a given network infrastructure e.g. the obstacle code has the same performance with 8 peers over LAN as with 4 peers from Grid'5000. Similarly, 4 peers in DSL provide the same computing power as 2 peers over LAN.

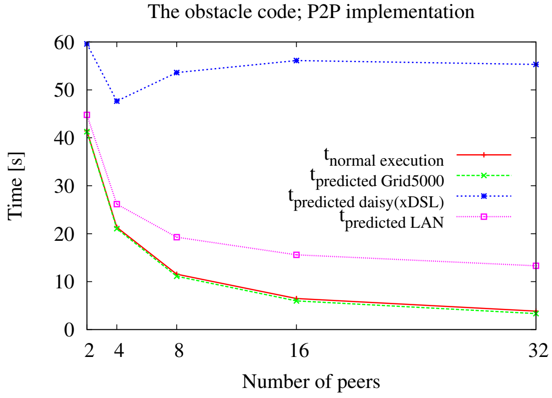


Figure 7: Predicting performance of scaled network.

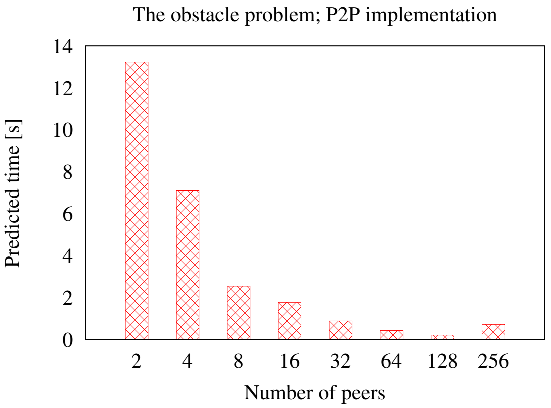


Figure 8: Predicting performance for a greater number of peers.

In Stage-3, we study the node scaling capability of dPerf. During static analysis, dPerf identifies the 2d mesh communication pattern used by the test code. Afterwards, the traces obtained for two peers are modified to create additional traces corresponding in turn to 2^n, where $n \in \{1, 2, 3, 4, 5\}$. In Fig.8, from the measurements taken on two nodes, dPerf scales the prediction for 4, 8, 16, 32, 64, 128, 256 peers. The predicted time in Fig.8 is a scaled performance prediction, while $t_{predicted}$ and $t_{predicted\,Grid5000}$ in Fig.7 are predictions obtained without scaling, from unmodified trace files.

We conclude this Section with the presentation and analysis of computational results obtained for several distributed iterative schemes carried out on Grid 5000 with up to 256 peers on five sites (see Table 1). For this, we consider the solution of a 3D obstacle problem with size $256 \times 256 \times 256$.

Regarding the decentralized peer-to-peer computing environment, we performed experiments on 8 clusters of 5 sites on the Grid'5000 testbed. Machine characteristics on each cluster and corresponding sequential computational time are presented in table I, i.e. in an heterogeneous context.

The topology server is placed on the site Toulouse. On each site, a tracker is launched in order to manage peers of the site. The submitter is a machine of the cluster Sagittaire at Lyon.

Figure 9 displays the speedup and efficiency of the different parallel iterative schemes of computation, i.e.

Table I: Machine specification and sequential computational time.

Site	Cluster	Processor	Memory	Seq time
Lyon	Sagittaire	AMD 2.4 GHz	2 Gb	32166 s
	Capricorne	AMD 2.0 GHz	2 Gb	33942 s
Sophia	Helios	AMD 2.2 GHz	4 Gb	33178 s
	Sol	AMD 2.6 GHz	4 Gb	29400 s
Toulouse	Pastel	AMD 2.6 GHz	8 Gb	27843 s
Nancy	Grelon	Intel Xeon 1.6 GHz	2 Gb	32476 s
Orsay	Gdx	AMD 2.0/2.4 GHz	2 Gb	34636 s
	Netgdx	AMD 2.0	2 Gb	34711 s

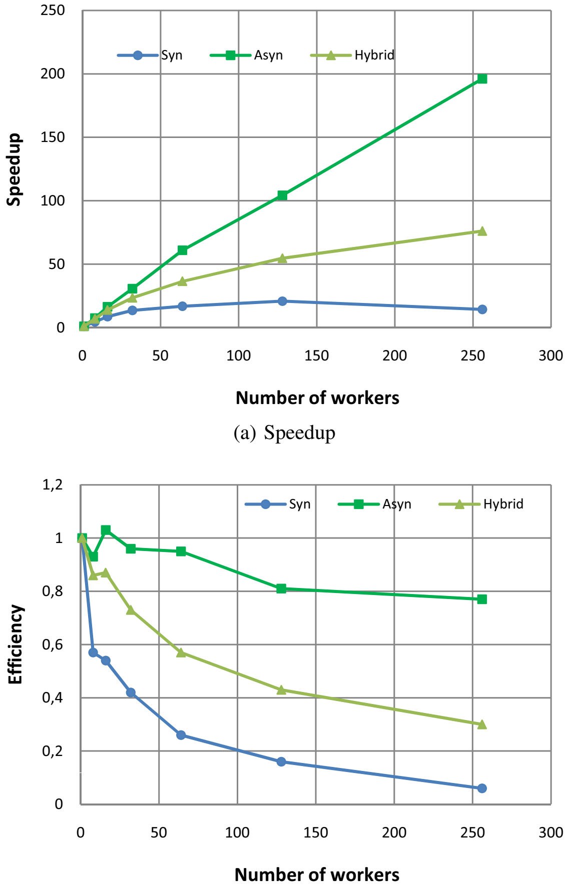
synchronous, asynchronous and hybrid schemes. Hybrid schemes of computation are a combination of synchronous and asynchronous schemes; in particular, in a multi-cluster context, computations can be carried out in a synchronous way inside clusters and asynchronously at the global level. We note that in the cases where the number of nodes is less than 256 machines, computations are carried out on 4 clusters at 4 locations: cluster Pastel at Toulouse, cluster Sagittaire at Lyon, cluster Grelon at Nancy and cluster Gdx at Orsay. For each experiment, an equal number of nodes is used at each site; for example, in experiment with 8 nodes, 2 nodes at Toulouse, 2 nodes at Orsay, 2 nodes at Nancy and 2 nodes at Lyon, respectively. In the case where the number of nodes is 256, nodes of others clusters are used. Speedup and efficiency are computed by using sequential computational time on the most performant cluster, i.e cluster Pastel at Toulouse.

Experimental results show that synchronous schemes of computation carried out with P2PDC do not scale well up on heterogeneous testbeds. Nevertheless, we note that the combination of asynchronous schemes of computation with P2PDC is very efficient. The lack of synchronization overhead and idle time due to synchronization permit one to obtain very good performance.

VI. Conclusion and perspectives

We observed the scalability of performance prediction done by dPerf, a tool that can analyze C, C++ or Fortran applications executed with the P2PDC decentralized environment. dPerf is suited for predicting performances of numerous HPC applications in numerical simulation and optimisation. We introduced notions about scalable performance predictions made with dPerf. The static-analysis approach of our tool, combined with execution and with trace-based simulation allows studying application performance on a scaled computing system.

For the application and topologies considered, we note that asynchronous schemes of computation perform better than the synchronous one. The efficiency of asynchronous schemes of computation decreases slowly with the number of processors; while the efficiency of synchronous schemes of computation deteriorates greatly when the number of processors increases; this is mainly due to synchronization overhead and waiting time. The efficiency of hybrid schemes of computation is situated in between efficiencies of synchronous and asynchronous schemes.



(a) Speedup

(b) Efficiency

Figure 9: Computational results.

We are currently working on eliminating the dependency of the instrumented code upon the host architecture. We aim at providing performance prediction relatively to a reference computing node. We plan on passing to the latest MSG module from Simgrid. This will eliminate the constraint related to the message size and thus improve the scalability of the number of peers.

We also plan to extend dPerf so as to take into account non determinism induced by distributed asynchronous iterative schemes of computation.

Finally, we shall extend the P2PSAP protocol in order to take into account Infiniband networks and carry out experiments on more peers.

Acknowledgment

This work was funded by the French National Agency for Research (contract ANR-07-CIS7-011 [32]). Experiments were carried out using the Grid5000 experimental testbed. SimGrid is developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies [30].

References

[1] T. T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and M. Chau, "High performance peer-to-peer distributed computing with application to obstacle problem," in *IPDPSW'10: IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, Atlanta,USA, 2010.

[2] B. F. Cornea and J. Bourgeois, "Performance prediction of distributed applications using block benchmarking methods," in *PDP'11: Proc. of the 19th Euromicro Conf. on Parallel, Distributed and Network-Based Processing*. IEEE Computer Society, 2011.

[3] D. El Baz and T. T. Nguyen, "A self-adaptive communication protocol with application to high performance peer to peer distributed computing," in *PDP'10: Proc. of the 18th Euromicro Conf. on Parallel, Distributed and Network-based Processing*. IEEE CPS, 2010, pp. 327–333.

[4] B. F. Cornea, J. Bourgeois, T. T. Nguyen, and D. El-Baz, "Performance prediction in a decentralized environment for peer-to-peer computing," in *IPDPS Workshops - HotP2P'11: International Workshop on Hot Topics in Peer-to-Peer Systems*. IEEE Computer Society Press, 2011, pp. 1613–1621.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: towards a realistic model of parallel computation," *Proc. of the 4th ACM SIGPLAN symposium on Principles and practice of parallel programming*, vol. 28, no. 7, pp. 1–12, 1993.

[6] D. Sundaram-Stukel and M. K. Vernon, "Predictive analysis of a wavefront application using LogGP," *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, vol. 34, no. 8, pp. 141–150, 1999.

[7] A. J. C. van Gemund, "Symbolic performance modeling of parallel systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 2, pp. 154–165, 2003.

[8] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Transactions on Computer Systems*, vol. 14, no. 4, pp. 344–384, 1996.

[9] J. Bourgeois and F. Spies, "Performance prediction of an NAS benchmark program with ChronosMix environment," in *Euro-Par'00: Proc. from the 6th International Euro-Par Conf. on Parallel Processing*. London, UK: Springer-Verlag, 2000, pp. 208–216.

[10] S. Prakash and R. L. Bagrodia, "MPI-SIM: using parallel simulation to evaluate mpi programs," in *WSC'98: Proc. of the 30th conference on Winter simulation*. IEEE Computer Society, 1998, pp. 467–474.

[11] V. S. Adve, R. Bagrodia, J. C. Browne, E. Deelman, A. Dube, E. N. Houstis, J. R. Rice, R. Sakellariou, D. J. Sundaram-Stukel, P. J. Teller, and M. K. Vernon, "POEMS: End-to-end performance design of large parallel adaptive computational systems," *IEEE Transactions on Software Engineering*, vol. 26, pp. 1027–1048, 2000.

[12] A. Snavely, N. Wolter, and L. Carrington, "Modeling application performance by convolving machine signatures with application profiles," in *WWC'01: IEEE International Workshop on Workload Characterization*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 149–156.

[13] R. M. Badia, F. Escalé, E. Gabriel, J. Gimenez, R. Keller, J. Labarta, and M. S. Müller, "Performance prediction in a grid environment," in *Grid Computing*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 2970, pp. 257–264.

[14] J. Zhai, W. Chen, and W. Zheng, "Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node," in *PPoPP'10: Proc. of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2010, pp. 305–314.

[15] B. Huffaker, M. Fomenkov, D. Plummer, D. Moore, and k. claffy, "Distance Metrics in the Internet," in *IEEE International Telecommunications Symposium (ITS)*, Sep 2002, pp. 200–202.

[16] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A generic framework for large-scale distributed experiments," in *UKSIM'08: Proc. of the 10th International Conf. on Computer Modeling and Simulation*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 126–131.

[17] M. Schordan and D. Quinlan, "A source-to-source architecture for user-defined optimizations," in *Modular Programming Languages*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2789, pp. 214–223.

[18] Perfmon project webpage. [Online]. Available: http://perfmon2.sourceforge.net/

[19] M. Pettersson. Perfctr project webpage. [Online]. Available: http://user.it.uu.se/~mikpe/linux/perfctr/

[20] PAPI project website. [Online]. Available: http://icl.cs.utk.edu/papi/

[21] PAPI sc2008 handout (papi-2008.pdf). [Online]. Available: http://icl.cs.utk.edu/graphics/posters/files/

[22] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature," *Cluster'09: IEEE International Conference on Cluster Computing and Workshops, 2009*, pp. 1–4, September 2009, poster.

[23] C. Stratan and V. Cristea, "A framework for performance prediction in distributed systems," *U.P.B. Scientific Bulletin, Series C*, vol. 71, no. 3, 2009.

[24] A. J. C. van Gemund, "Performance prediction of parallel processing systems: The pamela methodology," in *SC'93: International Conference on Supercomputing*, 1993, pp. 318–327.

[25] J. Bourgeois, V. Sunderam, J. Slawinski, and B. Cornea, "Extending executability of applications on varied target platforms," in *HPCC 2011, 13-th IEEE Int. Conf. on High Performance Computing and Communications*, Banff, Canada, Sep. 2011, pp. 1–8.

[26] P. Spiteri and M. Chau, "Parallel asynchronous richardson method for the solution of obstacle problem," in *Proc. of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, 2002, pp. 133–138.

[27] R. S. Varga, "Matrix iterative analysis." Prentice Hall, 1962.

[28] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (republished in 1997 by Athena Scientific), 1989.

[29] D. El Baz, "Contribution à l'algorithmique parallèle. le concept d'asynchronisme : étude théorique, mise en œuvre et application," in *Habilitation à diriger des recherches*. INP Toulouse, 1998.

[30] Grid5000 platform. http://www.grid5000.fr.

[31] Bordeaux hardware resources. [Online]. Available: https://www.grid5000.fr/mediawiki/index.php/Bordeaux:Hardware

[32] ANR-07-CIS7-011. ANR CIP web site. [Online]. Available: http://spiderman-2.laas.fr/CIS-CIP