A self-adaptive communication protocol with application to high performance peer to peer distributed computing#

Didier EL BAZ, The Tung NGUYEN
CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse France
{elbaz, ttnguyen}@laas.fr

***Abstract*— A self adaptive communication protocol is proposed for peer to peer distributed computing. This protocol can configure itself automatically in function of application requirements and topology changes by choosing the most appropriate communication mode between peers. The protocol was designed in order to be used in conjunction with a decentralized environment for high performance distributed computing. A first set of computational experiments is also presented and analyzed for an optimization application, i.e. nonlinear network flow problems.**

Keywords— communication protocol, self-adaptive protocol, micro-protocols, high performance computing, peer to peer computing, nonlinear optimization, network flow problems.

I. INTRODUCTION

The peer to peer concept has known great developments with file sharing applications like Gnutella [1] or FreeNet [2]. Recent advances in microprocessors architectures and high bandwidth networks permit one to consider high performance peer to peer distributed computing as an economic and attractive solution. Nevertheless, the implementation of high performance computing methods on peer to peer networks gives rise to numerous challenges like scalability issues, machine heterogeneity and peers volatility. Existing protocols like TCP and UDP are designed to answer to application requirements like fault tolerance and exchanged messages order. Generally, they are not well suited to peer to peer networks.

In this paper, we propose a self adaptive communication protocol for high performance peer to peer distributed computing. The configurable protocol chooses the most appropriate communication mode between peers according to choices made at the application level, like the computation scheme, i.e. synchronous or asynchronous iterative method, or events that occur at the network level like topology changes. This protocol will be further used in a decentralized environment for high performance peer to peer computing [3]. Section 2 deals with micro-protocols and protocol composition frameworks. In section 3, we propose some modifications to the Cactus framework and our self adaptive communication protocol. Section 4 deals with a first version of the decentralized environment for high performance peer to peer distributed computing. Finally, a first set of computational experiments for optimization problems, i.e. nonlinear network flow problems is presented and analyzed in Section 5.

II. MICRO-PROTOCOLS

Micro-protocols are an interesting approach to design self adaptive communication protocols. Micro-protocols were first introduced in x-kernel [4]. They have been widely used since in several systems. A micro-protocol implements merely a functionality of a given protocol. A protocol results from the composition of a given set of micro-protocols. This approach permits one to reuse the code, facilitate the design of new protocols and give the possibility to configure the protocol dynamically.

Several protocol composition frameworks have been proposed in order to deploy communication architectures. One can divide these frameworks according to 3 models: the hierarchical, nonhierarchical and hybrid models. In the hierarchical model, a stack of micro protocols composes a given protocol. This model can be found in the x-kernel [4] and APPIA [5] frameworks. In the nonhierarchical model, there is no particular order between micro-protocols; the Coyote [6] and ADAPTIVE [7] frameworks correspond to this model. The hybrid model is a combination of the two previous models; micro-protocols are composed here hierarchically and non-hierarchically. One can find this last model in the XQoS [8] and Cactus [9] frameworks.

We have concentrated on the Cactus framework since this approach is flexible and efficient. We present the Cactus framework in the next subsection.

A. Cactus framework

Cactus is a system for constructing highly-configurable protocols for networked and distributed system [9]. It has two grain levels. Individual protocols, termed *composite protocols*, are constructed from micro-protocols. Composite protocols are then layered on top of each other to create a protocol stack using an interface similar to the standard x-kernel API.

Cactus is an event-based framework. Events are used to signify state changes, such as arrival of messages from the network. Each micro-protocol is structured as a collection of event handlers, which are procedure-like segments of code and are bound to events. When an event occurs, all handlers bound to that event are executed. Events can be raised in different ways, explicitly by micro-protocols or implicitly by the runtime

#The present work was funded by ANR-07-CIS7-011 grant

system, with either blocking or non-blocking semantics, with a specific delay and a priority execution number. Arguments can be passed to handlers in two ways, statically when a handler is bound to an event and dynamically when an event is raised. The runtime system also provides operations for unbinding handlers, creating and deleting events, halting event execution, and canceling a delayed event. Handler execution is atomic with respect to concurrency, i.e., a handler is executed till completion before another handler is started unless it voluntarily yields the CPU.

The Cactus framework provides a message abstraction named dynamic messages, which is a generalization of traditional message headers. A dynamic message consists of a message body and an arbitrary set of named message attributes. Micro-protocols can add, read, and delete message attributes. When a message is passed to a lower-level protocol, a pack routine combines message attributes with the message body; while an analogous unpack routine extracts message attributes when a message is passed to a higher-level protocol. Cactus also supports shared data that can be accessed by all micro-protocols configured in a composite protocol.

The CTP Configurable Transport Protocol is designed and implemented using the Cactus framework [10]. Fig. 1 shows the CTP implementation with events on the right side and micro-protocols on the left side. An arrow from a micro-protocol to a given event indicates that the micro-protocol binds a handler to this event.

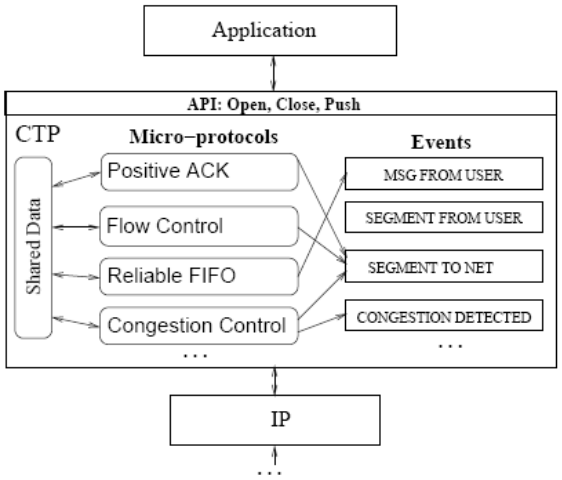

Fig. 1: CTP composite protocol

The CTP protocol includes a wide range of micro-protocols including a small set of basic micro-protocols like Transport Driver, Fixed Size or Resize and Checksum that are needed in every configuration and a set of micro-protocols implementing various transport properties like acknowledgements, i.e. PositiveAck, NegativeAck and DuplicateAck, retransmissions, i.e. Retransmit, forward error correction, i.e. ForwardErrorCorrection, and congestion control, i.e. WindowedCongestionControl and TCPCongestionAvoidance.

III. SELF-ADAPTIVE COMMUNICATION PROTOCOL

In this Section, we propose the so-called P2PSAP Peer To Peer Self Adaptive communication Protocol for high performance peer to peer distributed computing. Fig. 2 shows the architecture of the protocol; this protocol has a Socket interface and two channels: a control channel and a data channel. We present now in detail those components.

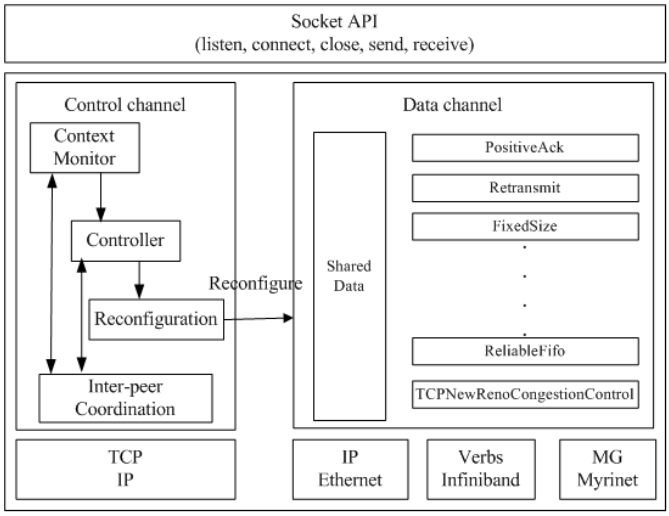

Fig. 2: P2PSAP Protocol architecture

A. Socket API

A main lack of Cactus CTP is that it has no application programming interface. Application has to use an interface as though it was just another composite protocol. So, we placed on the top of our protocol a socket-like API. Application can open and close connection, send and receive data. Furthermore, application can get session state and change session behavior or architecture through socket options, which are not available in Cactus. Session management commands like listen, open, close, setsockoption and getsockoption are directed to Control channel; while data exchange commands, i.e. send and receive commands are directed to Data channel.

B. Data channel

The data channel transfers data packets between peers; it is built by using the Cactus framework. To improve protocol performance and facilitate the reconfiguration, we have made some modifications to Cactus framework.

Firstly, Cactus doesn't allow concurrency handler execution. It means that a handler must wait for current executed handler completion before being executed. But nowadays, almost PCs have more than one core and concurrent handler execution is necessary to improve performance. So, we have modified Cactus to allow concurrency handler execution. Each thread has its own resources and its handler execution is independent of others.

Secondly, we have eliminated unnecessary messages copies between layers. In the Cactus framework, when a message is passed to upper or lower layers, Cactus runtime creates a new message and pass it to upper or lower layer. Hence, a significant number of CPU cycles and memories are consumed

in multiple-layers systems. In our protocol, message copies occur between Socket API layer and Data channel, and within the Data channel. In order to eliminate message copies, we have modified the pack and unpack functions so that only a pointer to message is passed between layers. Therefore, no message copy is made within the stack.

Finally, Cactus provides operations for unbinding handler but it has no explicit operation for removing a micro-protocol. In order to facilitate protocol reconfiguration, we have added to Cactus API an operation for micro-protocol removing. In addition to the micro-protocol initializing function, each micro-protocol must have a remove function, which unbinds all its handlers and releases its own resources. This function will be executed when the micro-protocol is removed.

After presenting our modifications to Cactus framework, we now consider in detail the data channel. The data channel has two levels: the physical layer and the transport layer; each layer corresponds to a Cactus composite protocol. We encompass the physical layer to support communications on different networks, i.e. Ethernet, InfiniBand and Myrinet. Each communication type is carried out by a composite protocol. The data channel can be triggered between the different types of networks; one composite protocol is then substituted to another. The transport layer is constituted by a composite protocol made of several micro-protocols, which is based on CTP. We have added to the existing micro-protocols a remove function corresponding to the modifications we have introduced to Cactus framework. In addition, we have designed some new micro-protocols that enable CTP to be used for sending and receiving messages in parallel computing applications.

Synchronization micro-protocols. CTP supports only asynchronous communication. But distributed application may have plural communication synchronization semantics. Hence, we have implemented 2 micro-protocols corresponding to 2 communication modes: synchronous and asynchronous. These micro-protocols introduce some new events, *UserSend* and *UserReceive*, which will be raised when send and receive socket commands are called by an application. In response to message sent from application, these micro-protocols may return the control to application immediately after message sent (asynchronous send) or wait for an acknowledgement indicating that message was received by receiver side application (synchronous send). Likely, in response to receive call from application, they may return the control to application immediately with or without message (asynchronous receive), or wait until message arrives (synchronous receive).

Buffer management micro-protocol. There are 2 buffers to manage: send buffer and receive buffer. Send buffer stores messages waiting to be sent or waiting for an acknowledgement from receiver application. Receive buffer stores messages which arrive from network and waits to be received by application. So, this micro-protocol must implement handlers for the *UserSend* and *MsgFromNet* events to catch messages from application and from network.

New congestion control micro-protocols. CTP actually has serveral micro-protocols implementing SCP congestion control and TCP-Tahoe congestion control. We have designed and used new micro-protocols implementing the TCP New-Reno congestion control [11] which is widely used and the H-TCP congestion control for high speed-latency network [12].

At this level, data channel configuration is carried out by substituting or removing and adding micro-protocols. The behavior of the data channel is triggered by the control channel.

C. Control channel

The Control channel manages session opening and closure; it captures context information and (re)configures the data channel at opening or operation time; it adapts itself to these information and their changes; it is also responsible for coordination between peers during reconfiguration process. Note that we use the TCP/IP protocol to exchange those messages since Control messages cannot be lost.

Before describing the main components of the control channel, we present first a session life cycle, like in Fig. 3. Suppose process A wants to exchange data with process B, it opens a session through socket create and connect command. Then, a TCP connection is opened between 2 processes. Process B accepting connection must send its context information to process A. Process A, based on its context information and those of B, chooses the most appropriate configuration for data channel and send configuration command to process B. After that, the two processes carry out

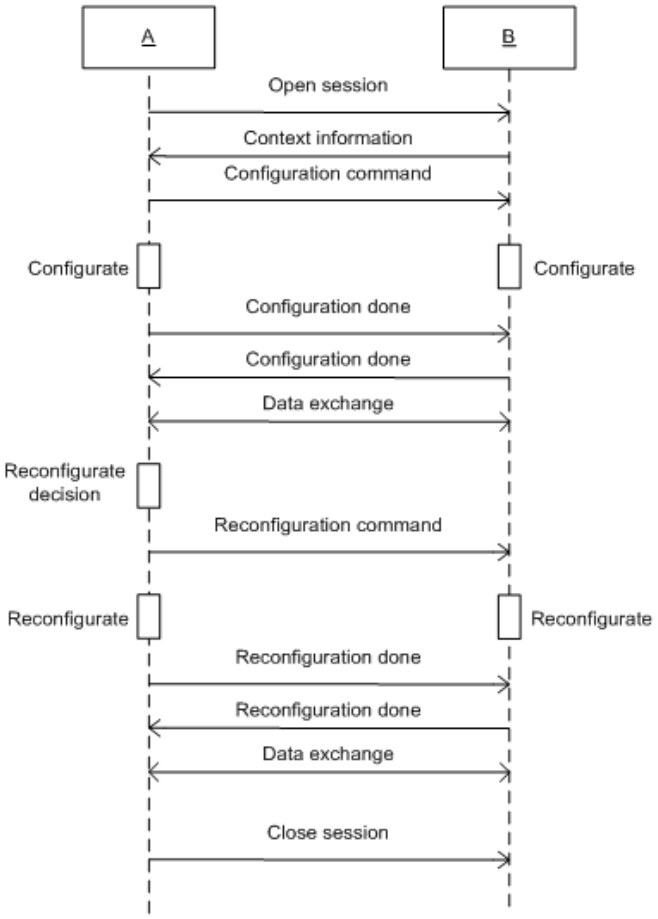

Fig. 3: Session life cycle

the configuration of data channel. When the configuration is done, each process has to inform the other process and waits for the notification of other process. Data is exchanged only when both processes have finished data channel configuration. During the communication, a process can decide to change configuration of data channel due to context changes or user requirements, like process A in Fig. 3. Then, a similar procedure will be realized. When session is closed, the data channel is closed firstly; the control channel with TCP connection is closed later on.

We describe now the main components of the control channel.

1) Context monitor: the context monitor collects context data and their changes. Protocol adaptation is based on context acquisition, data aggregation and data interpretation. Context data can be requirements imposed by the user or the algorithm at the application level, i.e. asynchronous algorithms, synchronous algorithms or hybrid methods. Context data can also be related to peers location and machine loads. Context data are collected at specific times, periodically or by means of triggers. Data collected by the context monitor can be referenced by the controller.

2) Controller: the controller is the most important component in the control channel; it manages session opening and close through TCP connection opening and close; it also combines and analyzes context information provided by the context monitor so as to choose the configuration (at session opening) or to take reconfiguration decision (during session operation) for data channel. The choice of the most appropriate configuration is determined by a set of rules that are described by a specification language such as OWL, ECA, etc. These rules specify new configuration and actions needed to realize it. The (re)configuration command along with necessary information is sent to component Reconfiguration and to other communication end point.

3) Reconfiguration: reconfiguration actions are made by the reconfiguration component via the dedicated Cactus functions. Reconfiguration is made at the physical layer by substituting a composite protocol supporting communication on a network board to a composite protocol supporting communication on another type of network board or at the transport layer by removing and adding or substituting micro-protocols.

4) Inter-peer coordination: the coordination component is responsible of context information exchanges and coordination of peers reconfiguration processes so as to insure proper working of the protocol.

D. Example of scenario

We present now a simple scenario for the P2PSAP protocol so as to illustrate its behavior.

We consider a high performance computing application, like for instance a large scale numerical simulation application or a complex optimization problem, solved on the network composed of two clusters shown in Fig. 4. The first cluster is composed of two similar machines: P1 and P2 that can be connected via Ethernet or InfiniBand. The second cluster is made of two similar machines: P3 and P4 connected only via Ethernet. The communication protocol between machines P3 and P4 is based on synchronous communication (since machines have similar characteristics and loads) via micro-protocols insuring e.g. reliability and order, i.e. ReliableFifo, and congestion control, i.e. TCPNewRenoCongestionControl. The communication protocol between machines P1 and P2 is based on synchronous communication for the same reasons via Infiniband since Infiniband is faster than Ethernet. Moreover, InfiniBand insures reliability and message order; as a consequence, the data channel needs only micro-protocols insuring buffer management (Buffer management), synchronous communication (Synchronous) and segment size management (Resize). Communications between machines of the different clusters are asynchronous; as a consequence, we need neither order, nor reliability micro-protocols in this case.

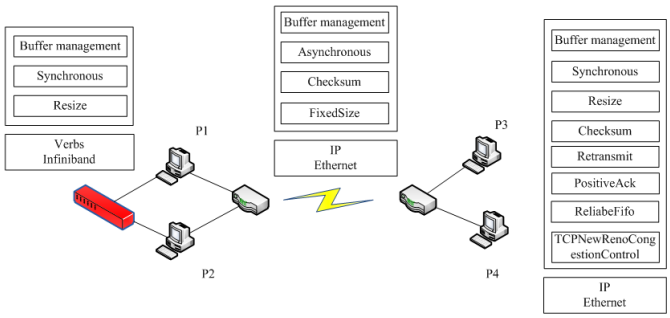

Fig. 4: Example of scenario

IV. Environment for High Performance Peer to Peer Distributed Computing

In this section, we present a first prototype of an environment for high performance peer to peer distributed computing. For previous research work, see [15]. Our environment uses the P2PSAP protocol for data exchanges between peers. Fig. 5 illustrates the architecture of the environment. In the future, we shall develop a decentralized environment based on P2PSAP.

We describe now the main components of the decentralized environment.

1) User daemon is the interaction interface between the application and the environment. It allows users to submit their tasks and retrieve final results.

2) Topology management organizes connected peers into clusters and maintains links between peers and between clusters.

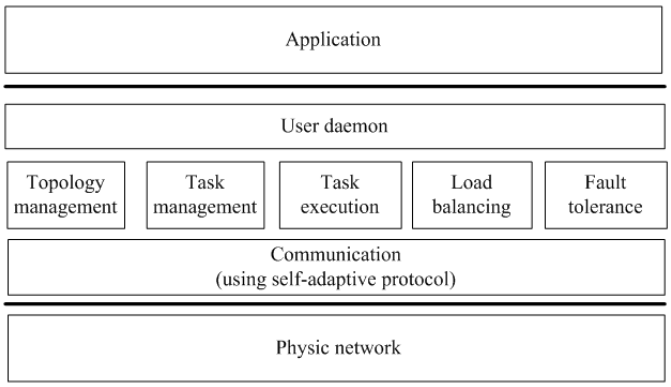

Fig. 5: Environment prototype

3) Task management is responsible for task splitting into sub-tasks, sub-tasks distribution and results collection.

4) Task execution executes sub-tasks and exchanges intermediate results.

5) Load balancing estimates peer workload and migrates a part of work from overload peer to non-loaded peer.

6) Fault tolerance ensures the integrity of the calculation in case of peer failure. We intend to use the rollback recovery mechanism with checkpoint to recover state of failure peer on another peer.

7) Communication provides support for data exchange between peers using self-adaptive protocol.

V. EXPERIMENTATION

We present now a first set of computational experiments for nonlinear network flow problems.

A. Network flow problems

Network flow problems [13] consist in computing the network flow from a source to a destination that minimizes the total traffic cost. The problems occur in many domains: electrical networks, gas or water distribution, financial models, communication and transportation networks. The solution of nonlinear network flow problems requires intensive computations, so a distributed or parallel solution of these problems is very attractive.

1) Problem formulation

Let $G=(N,A)$ be a connected directed graph. N is referred to as the set of nodes, $A \subseteq N \times N$ is referred to as the set of arcs, and the cardinal number of N is denoted by n. Let $c_{ij}: R \to (-\infty,+\infty]$ be the cost function associated with each arc $(i,j) \in A$, c_{ij} is a function of the flow of the arc (i,j) which is denoted by f_{ij}. Let b_i be the supply or demand at node $i \in N$, we have $\sum_{i \in N} b_i = 0$. The problem is to minimize total cost subject to a conservation of flow constraint at each node:

$$\min \sum_{(i,j)\in A} c_{ij}(f_{ij}) \text{, subject to } \sum_{(i,j)\in A} f_{ij} - \sum_{(m,i)\in A} f_{mi} = b_i, \forall i \in N. \quad \text{(V.1)}$$

We assume that problem (V.1) has a feasible solution. We consider the following standing assumptions on c_{ij}.

Assumption V.1. c_{ij} is strictly convex.

Assumption V.2. c_{ij} is lower semicontinuous.

Assumption V.3. The conjugate convex function of c_{ij}, defined by

$$c_{ij}^*(t_{ij}) = \sup\{t_{ij}.f_{ij} - c_{ij}(f_{ij})\},$$

is real valued, i.e., $-\infty < c_{ij}^*(t_{ij}) < +\infty, \forall t_{ij} \in R$.

2) Dual problem

A dual problem for (V.1) is given by

$$\min_{p \in R^n} q(p), \quad \text{(V.2)}$$

subject to no constraint on the vector $p = \{p_i \mid i \in N\}$, where q is the dual functional given by

$$q(p) = \sum_{(i,j)\in A} c_{ij}^*(p_i - p_j) - \sum_{i \in N} b_i.p_i$$

We refer to p as a price vector and its components as prices. The i-th price p_i, is a Lagrange multiplier associated with the i-th conservation of flow constraint. The necessary and sufficient condition for optimality of a pair (f,p) is given as follows: a feasible flow vector $f = \{f_{ij} \mid (i,j) \in A\}$ is optimal for (V.1) and a price vector $p = \{p_i \mid i \in N\}$ is optimal for (V.2) if and only if for all $(i,j) \in A$, $p_i - p_j$ is a sub-gradient of c_{ij} at f_{ij}. An equivalent condition is $f_{ij}^* = \nabla c_{ij}^*(p_i - p_j)$, $\forall (i,j) \in A$, where $\nabla c_{ij}^*(x)$ denotes the gradient of $c_{ij}^*(x)$.

3) Dual optimal solution set

The optimal solution of the dual problem is never unique since adding the same constant to all coordinates of a price vector p leaves the dual cost unaffected. We can remove this degree of freedom by constraining the price of one node, say the destination node d, to be zero. Consider the set $P = \{p \in R^n \mid p_d = 0\}$. We concentrate on the reduced dual problem:

$$\min_{p \in P} q(p). \quad \text{(V.3)}$$

The reduced optimal solution set P^* is defined by:

$$P^* = \{p' \in P \mid q(p') = \min_p q(p)\}$$

Assumption V.4. The reduced dual optimal solution set P^* is nonempty and compact.

We note that assumption V.4 is not very restrictive (see [13]). In the sequel, $g(p)$ will denote the gradient of the dual functional, the components $g_i(p)$ of $g(p)$ are given by:

$$g_i(p) = \frac{\partial q(p)}{\partial p_i} = \sum_{(i,j)\in A} \nabla c_{ij}^*(p_i - p_j) - \sum_{(m,i)\in A} \nabla c_{mi}^*(p_m - p_i) - b_i, i \in N.$$

4) Parallel iterative algorithms

One can implement several parallel iterative methods for the solution of the reduced dual problem. We present now a gradient type method. The components F_i' of the gradient mapping F' are defined by $F_i' = p_i - \frac{1}{\alpha} g_i(p)$ for all $i \in N - \{d\}$ and $p \in P$, where α is a positive constant. Clearly F' is continuous since g is continuous. We introduce the following assumption.

Assumption V.5. c_{ij} is strongly convex with modulus $\frac{1}{\beta}$.

Under Assumptions V.1 to V.5, there exists a constant $\alpha = \beta.max_{i \in N} a_i$, where a_i denotes the degree of node $i \in N$, such that for all $p, p' \in P$ satisfying $p' \le p$, we have $g(p) - g(p') \le \alpha.(p - p')$. Therefore, the gradient mapping F' is monotone on P if $\alpha = \beta.max_{i \in N} a_i$ (see [13]).

The gradient type method consists in iterating on the *i*-th component of the vector of prices as follows:

$$p_i^q = p_i^{q-1} - \frac{1}{\alpha} g_i(p_i^{q-1}, p), i = 1,...,n, q \in \{1,2,...,q'\},$$

where $p_i^0 = p_i$ and q' is the number of iterations such that $|g_i(p_i^{q'})| \le \varepsilon$ where $p_i^0 = P_i$ and ε is the research accuracy.

B. Experiment platform

Experiments have been carried out on the LAASNETEXP experimental network (see [14]). The topology of the small network used for this first set of computational experiments is displayed in Fig. 6 where peers are connected via a Gigabit Ethernet network. Machines P1, P2, P3 and P4 are similar i.e.: Dual Core Xeon 3050, 2.13GHz with Linux Debian.

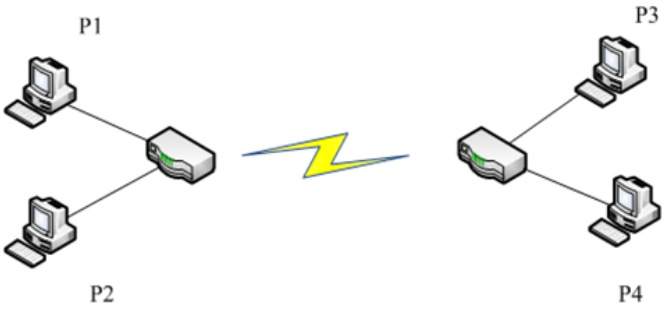

Fig. 6: Network used for computational tests

We have used the C implementation of Cactus 2.2 for micro-protocol composition over Linux UDP sockets.

C. Self-adaptation

In these experiments, the context data used for configuring the protocol are the computational schemes chosen by the user (synchronous iterative schemes, asynchronous iterative schemes or hybrid schemes, i.e. combination of synchronous and asynchronous iterative schemes) and topology parameters i.e. peer location which implies inter or intra cluster communication. Decision rules are summarized in Table 1. In the sequel, we explain those rules.

TABLE I:
COMMUNICATION MODE ADAPTATION RULES

	Synchronous	**Asynchronous**	**Hybrid**
Intra-cluster	Synchronous Reliable Com.	Asynchronous Reliable Com.	Synchronous Reliable Com.
Inter-cluster	Synchronous Reliable Com.	Asynchronous Unreliable Com.	Asynchronous Unreliable Com.

In clusters, machines are often identical; they are connected via a local network with small latency, high bandwidth and reliable data transfer. As a consequence, if computational loads are well balanced, then synchronous communication between peers can be very efficient. On the other hand, in inter-cluster data exchanges where there may be heterogeneity (processors, OS, bandwidth) and communications may have high latency and unreliability, synchronization may be an obstacle to efficiency due to waiting times and overheads. Thus, asynchronous communication seems more appropriate in this case.

Communication modes have to correspond to computational schemes which depend on algorithm nature and are generally chosen by users. In the case of synchronous schemes, the synchronous communication must be implemented in both intra-cluster and inter-cluster data exchanges. Thus, micro-protocols used for the data channel will be the Synchronous micro-protocol with some reliability and order micro-protocols. To explore the high-speed long distance networks, the data channel can use H-TCP congestion control micro-protocol for inter-cluster communication instead of TCP New-Reno congestion control micro-protocol which works well only in low latency network.

Likely, in the case of asynchronous schemes, the asynchronous communication must be implemented in both intra-cluster and inter-cluster data exchanges. We note that asynchronous schemes of computation are fault tolerant in some sense since they allow messages losses. However, messages losses may lead to some iterations increases. For this reason, in intra-cluster communication with low latency, it may be better to add some reliability micro protocols to the data channel along with the Asynchronous micro-protocol. While in inter-cluster communication with high latency and message losses recovery time may be comparable with updating time, thus those messages can become obsolete. Hence, reliability micro protocols are not needed in this case.

There are also some problems that can be solved by using all computational schemes. In this last case, the user can leave the protocol to freely choose communication mode. As a consequence, it can choose the most appropriate mode according to topology parameters, i.e., synchronous communication for intra-cluster data exchanges and asynchronous communication for inter-cluster data exchanges; this corresponds to the so-called *Hybrid* computational scheme.

D. Results

Computations have been carried out on 1, 2 and 4 machines. In the distributed case, i.e. for several machines, the original network flow problem is decomposed into several sub-networks. In the particular case of 2 machines, computations are made within the same cluster. We have carried out experiments with different computational schemes and communication scenarios i.e. synchronous, asynchronous and hybrid (synchronous / asynchronous). A first set of computational results is displayed in Table II for gradient type methods applied to gas distribution problems with 2000 nodes.

TABLE II.
COMPUTATIONAL RESULTS

Number of PCs	Method	Number of iterations				Times (s)	Speedup	Efficiency
		P1	P2	P3	P4			
1	-	399813	-	-	-	2135	-	-
2	Syn	400694	400694	-	-	1481	1,44	0,72
2	Asyn	385780	583735	-	-	1209	1,76	0,88
4	Syn	402056	402056	402056	402056	1241	1,72	0,43
4	Hybrid	449372	449372	398421	398421	935	2,28	0,57
4	Asyn	419175	389144	464128	743636	656	3,25	0,81

For the application and topology considered, we note that asynchronous schemes have performed better than the synchronous ones. Moreover, the efficiency of synchronous schemes deteriorates greatly when the number of processors increases, i.e. 0.72 with 2 machines and 0.44 with 4 machines. The efficiency of asynchronous distributed schemes decreases slowly with the number of processors, i.e. 0.88 with 2 machines and 0.81 with 4 machines. This is mainly due to waiting time due to synchronization and synchronization overhead. When using asynchronous schemes of computation, some processors may iterate faster than others; this is particularly the case when loads are unbalanced as for the application considered here. The use of hybrid schemes, i.e. synchronous communication between peers in the same cluster and asynchronous communication between peers in different clusters gave efficiency that is in between efficiencies of pure synchronous and asynchronous methods.

It follows from the computational experiments that the choice of communication mode has important consequences on the distributed method efficiency. The ability for the protocol P2PSAP to choose the best communication mode in function of network topology, context and computational scheme choices appears as a crucial property. We note also that the choice of communication mode has important consequences on the reliability of the distributed method and everlastingness of the high performance computing application. In this last context, we note that asynchronous communications are more appropriate in the case of communications between clusters.

VI. CONCLUSIONS

In this paper, we have proposed a self adaptive communication protocol for high performance peer to peer distributed computing. This protocol has been implemented on a small network for the solution of nonlinear optimization problems, i.e. network flow problems. A first set of computational experiments shows that the protocol permits one to obtain good efficiency when using asynchronous communications or a combination of synchronous and asynchronous communications.

We plan to study a specification language for controller decision rules description. We shall also concentrate on the design of a decentralized environment for high performance peer to peer distributed computing. This type of environment will permit one to use all the specificities offered by the peer to peer concept for high performance computing purpose. Self organization of peers for efficiency purpose or for insuring everlastingness of applications in hazardous situations or in the presence of faults will also be studied. Finally, we plan to consider other applications e.g. financial mathematics problems like American or European options problems. The different applications considered will permit us to validate our protocol and decentralized environment in different high performance computing contexts.

REFERENCES

[1] Gnutella Protocol Development. *http://rfc-gnutella.sourceforge.net.*

[2] The FreeNet Network Projet. *http://freenet.sourceforge.net.*

[3] D. El Baz, T. T. Nguyen et al, "CIP - Calcul intensif pair à pair", Poster, Colloque *Ter@tec2009*, SUPELEC, 30 Juin-1 Juillet 2009.

[4] N. Hutchison and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, pp. 64-76, 1991.

[5] H. Miranda, A. Pinto, and L. Rodrigues, "Appia: A flexible protocol kernel supporting multiple coordinated channels," in *Proc. 21st International conference on Distributed Computing Systems* (ICDCS-21), (Phoenix, Arizona, USA), pp. 707–710, 2001.

[6] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu, "Coyote: a system for constructing fine-grain configurable communication services," in *ACM Transactions on Computer Systems*, 16(4): pp. 321–366, 1998.

[7] D. C. Schmidt, D. F. Box, and T. Suda, "ADAPTIVE—A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment," Journal of Concurrency: Practice and Experience, 5(4): pp. 269–286, 1993.

[8] E. Exposito, P. Senac, M. Diaz, "FPTP: the XQoS aware and fully programmable transport protocol," in Networks, 2003. ICON2003. The 11th IEEE International Conference on, pp. 249-254.

[9] Matti A. Hiltunen, "The Cactus Approach to Building Configurable Middleware Services", in *DSMGC2000*, Nuremberg, Germany, 2000.

[10] G.T Wong, M.A Hiltunen, R.D Schlichting, "A configurable and extensible transport protocol," in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska (2001), pp. 319–328.

[11] S. Floyd, T. Henderson, "The New-Reno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, Apr 1999.

[12] D. Leith and R. Shorten, "H-TCP protocol for high-speed long distance networks," in *PFLDnet*, Feb. 2004.

[13] D. El Baz, P. Spiteri, J. C. Miellou, D. Gazen, "Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems," in *Journal of Parallel and Distributed Computing*, Vol. 38, pp. 1-15, 1996.

[14] P. Owezarski, P. Berthou, Y. Labit, D. Gauchard, "LaasNetExp: a generic polymorphic platform for network emulation and experiments," in *TridentCom'2008*, Innsbruck, Austria, March 18-20, 2008.

[15] D. El Baz, G. Jourjon, "Some solutions for Peer to Peer Global Computing," in *13th Euromicro conference on Parallel, Distributed and Network-Base Processing*, 2005, pp. 49-58.