A novel routing algorithm for mobile pervasive computing

Romano Fantacci, Daniele Tarchi, and Andrea Tassi
Department of Electronics and Telecommunications
University of Florence
Firenze, Italy

***Abstract*—The interest towards real-time computing has lead an even more interest in grid computing. While in the past the implementation of grid computing has been done on high performance computers, in the recent years there is an increasing interest in the pervasive grid scenarios, where multiple devices can be used for a distributed computing. The most challenging idea is to use mobile devices connected among them with wireless connections for setting up pervasive grid environments. In this context, it is a crucial problem the optimization of the routing algorithms among the processing nodes, in order to satisfy the performance requirements of a distributed computing. Aim of this paper is the design of specific routing algorithms for different pervasive grid applications with a particular attention to time sensitive scenarios.**

I. Introduction

In the last years computing and communication fields has been deeply developed in both research and industrial sides. However, at this time, they are not truly integrated. A joint integration of the two aspects allows a more efficient use of computing power in distributed environment where a multitude of small-medium sized devices are present, thus allowing intensive processing also without dedicated infrastructures. On one side the mobile computing offers the opportunity to have a huge number of devices disseminated all over an area. On the other side the Grid paradigm for cooperative and high-performance enabling platforms is evolving to more dynamic and pervasive applications. Historically these approaches have been used for fixed computers connected by high speed networks. The introduction of broadband wireless technologies and the improvement in computational capabilities of portable devices allows now to interconnect also mobile devices with high speed structures and to use them for processing parts of a distributed application [1].

An efficient integrated model, based on an innovative approach integrating a Pervasive Grid platform and an efficient wireless communication network [2], is of crucial importance. The goal of this integration is to allow a distributed computing platform to operate on an integrated communication network characterized by high heterogeneity, mobility and dynamism.

The reason of such integration is even more crucial for all those applications that require a quick response to elaborate complex models: this is the case of emergency forecasting where complex mathematical model require high-performance computing to provide clients with prompt and best-effort services.

The classical solution is to map forecasting models to central processing centers, which support high-performance architectures (e.g. cluster) and which are geographically far from the emergency area. This classical solution is not always feasible or the best one (in terms of performance) because of hardware or software failures in communication and computation infrastructures and/or high communication overheads between far sites. A solution is to map high-performance complex forecasting models to all available processing and communication resources, which are even heterogeneous and with limited computational capabilities, but geographically near the disaster area.

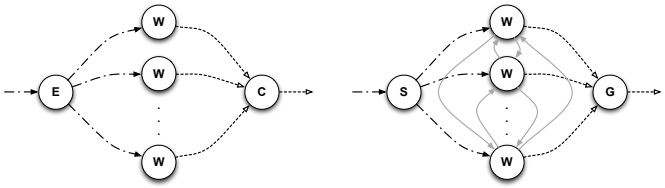
The integration of an efficient pervasive computing infrastructure with telecommunication aspects can be seen from different point of views. Among several aspects, one of the most important issues to be solved in such environments is to correctly find and route the processing structure on the *best node*: aim of the routing algorithm is to find that best node and the best path to it. Aim of this paper is propose novel routing strategies with a specific focus on the distributed environment we are facing.

II. Pervasive Grid Computing

Grid computing consists of a number of resources interconnected through a network, to be shared amongst its users. Large computing endeavors can be distributed over this network to these resources, and scheduled to fulfill requirements with the highest possible efficiency. Grids, as opposed to cluster computing efforts, consist of geographically distributed heterogeneous resources, interconnected through a variety of local area/wide area networks.

The Grid paradigm aims to enable the access, selection and aggregation of a variety of distributed and heterogeneous resources and services. However, though notable advancements in recent years have been achieved, Grid technology is not yet able to supply the needed software technology with the application requirements. A complex mapping problem must be solved dynamically each time the state of the Pervasive Grid changes in such a way that it affects the application performance.

This work was supported in part by MIUR-FIRB Integrated System for Emergency (InSyEme) under Grant RBIP063BPH.



(a) Task Farm (b) Data Parallel

Fig. 1. Considered grid schemes.

In this paper we assume that Pervasive Grid applications are developed in the ASSISTANT programming model [3], which features the following characteristics:

- application components can be provided in multiple versions, based on different sequential algorithms and parallelization techniques;
- versions are dynamically selected to face with context events. The programmer can define context events to be sensed and the adaptivity actions to be performed whenever such events are dynamically verified.

Here, we consider the case in which we have provided two operations: the first one is based on a task parallel structure, while the second one on a data parallel scheme. Broadly, these parallelism schemes feature different performance characteristics:

- **Task Farm**: the input tasks are scheduled (task emitter **E**) w.r.t. several replicated workers (**W**) according to a load balancing strategy, each worker executing the sequential algorithm. An output stream of results is produced, as a collection to the collector (**C**) of worker results (see Fig. 1a).
- **Data Parallel**: each input task is scattered by the component **S** onto several replicated workers (**W**), each one performing the sequential algorithm for its respective partition. Depending on the model, workers may cooperate during each step according to a proper communication stencil. The whole result is reassembled by the gather **G** (see Fig. 1b).

The adaptivity is based on dynamically selecting the best possible version of the same distributed application, depending on the actual situation in which it is executed. In this parallelism paradigm we could also have some functional dependencies between the computations performed by the workers (the gray lines in Fig. 1b), i.e.: a worker in order to complete its own task could require to know the output data produced during the computations performed by the other workers.

III. Routing algorithms for Pervasive Grid Computing

Our aim is the definition of a wireless routing protocol enabling resource selection based on both link (i.e., transmission latencies) and computing node performance. For computing nodes, we can keep updated system load information, e.g. related to current computing load, on each node running the routing protocol. This information, along with communication latency information, will be used to solve the mapping problem of ASSISTANT components to available resources.

During the last years, several routing algorithms for ad-hoc networks has been developed, each one with different characteristics and behavior. Among others we have resorted to the Optimize Link State Protocol (OLSR) [4] that allow a more efficient carrying of information throughout the network.

The resource discovery and the QoS assurance are independent from the routing algorithm; our aim is to propose a new routing policy able to guarantee target performance for distributed computing applications with low overhead signaling based on a clustering architecture. In that sense we have resorted to an evolution of the OLSR called QoS-OLSR (QOLSR) [5], where the HELLO messages are used also for estimating the delay and the bandwidth among neighbor nodes. However, the two QoS values are only estimated but they are not added to the routing table.

Our proposal is toward an extension of the routing table by including delay and bandwidth values along with other parameters more specific for the distributed processing under evaluation. In particular we will consider the following parameters on each node:

- *the CPU type*: expressed in terms of elaboration power or architecture type (e.g. x86, MIPS, XScale, etc.);
- *the occupied CPU percentage*: as a terms for the useful availability of the CPU;
- *allocated memory*: expressed in terms of MB of the used RAM within the node;
- *battery charge*: expressed in terms of remaining time battery charge.

Moreover, for each path is defined the residual energy as the minimum value of the batteries charge value along the path. The occupied CPU and available memory values are expressed as an Exponential Moving Average defined as:

$$a_i = \alpha x_i + (1 - \alpha) a_{i-1} \tag{1}$$

where a_i is the actually estimated mean value, x_i is the current measured value and a_{i-1} is the previous estimated value; the smoothing factor α has been set to, respectively, 0.8 and 0.9.

The routing table filling mechanism needs to take into account that we aim to build clustering set for the specific application we are considering. We consider the notable case of a set of mobile nodes interconnected by a wireless network. We can think to logically organize mobile computing nodes in clusters characterized by increasing network ranges (e.g. from 1-hop to N-hops clusters). On one hand, lower range clusters provide higher communication latency performance, but they usually aggregate a lower number of computing nodes, which influences the computing performance. On the other hand, larger range clusters provide higher parallelism degrees, but at the cost of a higher communication latency. The HELLO messages needs to be modified in this way:

- they must carry information about available bandwidth, delay and residual energy of the path;

- they must carry the node characteristics in terms of CPU type, occupied CPU percentage, available memory, battery charge.

The Topology Control (TC) messages are also extended in order to carry:

- the same QoS metrics used in the HELLO messages;
- the information needed to build the cluster table. In every node, the TC message sender will be a cluster head (CH) in the cluster tables, and its own n-hops neighbors will form the V_i set of that entry.

A node can fill a neighbor table with the list of 1-hop nodes with the IP address of the HELLO messages that store the address of the node itself. However our aim is to form a cluster structure able to support the distributed computing; for this reason, following the routing table, each node needs to build a cluster table where each possible cluster within the network is defined.

In order to reduce the complexity we will consider that the number of entry of the cluster table corresponds to the nodes within the network, by considering that the cluster is defined as the set on of node at 1-hop distance. Each entry of the cluster table has the following fields:

- **CH**: It is the cluster head identifier, including its IP number, and characteristics in terms of bandwidth, delay, battery charge, CPU, and available memory;
- $\boldsymbol{V_i}$: It represent the i-th node identifier (with the same QoS metrics of above regarding the i-th node or the link between CH and V_i) belonging tho the cluster having as cluster head the node identified in the field CH. There could be a number of entry equal to the number of nodes belonging to the cluster;
- **timestamp**: It represent the time when the cluster table has been updated.

A. Task Farm

We have now considered the use of the above described routing algorithm to the case of a task farm scheme. In this case we have one emitter node, a certain number of worker nodes and one collector node. By exploiting the routing algorithm we can select the nodes involved in the task farm computation.

Let us consider to have a set $\mathbf{V}$ containing all the nodes in the ad-hoc network. For each node i we can have:

- BW_i, the maximum occupied bandwidth among all links composing a generic path form a network node to i;
- d_i, the delay for reaching i;
- Pow_i, the maximum amount of energy consumed among the network (battery powered) nodes members of a generic path to i, including i itself;
- Mem_i, the memory allocated in the node i;
- CPU_i, the occupied CPU at node i.

Thus, we can define two cost functions, one related to variable costs (A_i), dependent on the new work to be allocated, and one related to the fixed costs (B_i), dependent on the actual status of the node:

$$A_i = \Delta \widetilde{Mem}_i + \Gamma \widetilde{CPU}_i \tag{2}$$

$$B_i = \alpha BW_i + \beta d_i + \gamma Pow_i + \delta Mem_i + \epsilon CPU_i \tag{3}$$

where $\widetilde{Mem}_i$ and $\widetilde{CPU}_i$ corresponds, respectively to the memory and CPU needed for precessing the work in the *i*-th node, and Δ, Γ, α, β, γ, δ, and ϵ are weights. We can thus derive the objective function, that is:

$$\min \left(\sum_{i \in \mathbf{V}} M_i A_i + B_i \right) \tag{4}$$

where M_i corresponds to the number of computations allocated on node i. The optimization problem has only a constraint, only one computation has to be mapped in a node so:

$$\sum_{i \in \mathbf{V}} M_i = 1 \qquad \text{where } M_i \in \{0,1\}, \forall i \in V.$$

B. Data Parallel

In this case we consider a data parallel scheme for distributed computing. We have now three node types: one scatter, a certain number of workers and one gather.

Differently from the Task farm we have now to solve two optimization problems: we need first to identify the cluster to be used for the computing, and then select which nodes should be used within the cluster.

The cluster selection is based on the following cost function to be minimized:

$$\min \left(\sum_{i \in CT} \sigma_i \sum_{j \in CT_i} \left(\alpha BW_{i,j} + \beta d_{i,j} + \gamma Pow_{i,j} + \delta Mem_{i,j} + \epsilon CPU_{i,j} \right) + \phi(MaxW - W_i) \right) \tag{5}$$

where the double index (i, j) stands for the i-th cluster head and the j-th node at one-hop distance for that cluster head, W_i is the maximum number of workers in the cluster i, $MaxW$ is the maximum number of worker required by the application, ad ϕ is a weight. The value of σ_i is equal to 1 if the cluster is the optimum and 0 otherwise.

After selecting the optimum cluster we want to select the nodes within the cluster where the computations will be performed. In a node can be executed, according with the CPU type, one or more working process (**W**) in parallel. The function to be minimized for selecting the nodes is:

$$\min \left(H \sum_{j \in CT_i} (M_j A_j + B_j) + K \left(MaxS - S \right) \right) \tag{6}$$

where $M_j \in N$ is the number of tasks dispatched to the *i*-th node of the optimal cluster CT_i, H and K are the two not negative weights. In this case the fixed cost of the *j*-th node is computed using the cluster table entry.

We have now introduced the values $MaxS$ and S that corresponds, respectively, to the maximum number of iterations needed by the data parallel application and the number

Procedure 1 The greedy heuristic

$S \leftarrow MaxS$
while $S \geq SminVal$ **do**
  $W \leftarrow f(S)$
  if $W \geq W_{CT_i}$ **then**
    $B \leftarrow sort(B)$
    $j \leftarrow 1$
    while $W > 0$ **do**
      if $maxWorker(B_i) \leq W$ **then**
        $M_{B_j} \leftarrow MaxNodeW_{B_j}$
      else
        $M_{B_j} \leftarrow W$
      end if
      $W \leftarrow W - M_{B_j}$
      $j \leftarrow j + 1$
    end while
    return M
  else
    $S \leftarrow S - 1$
  end if
end while
return *the computations can't be mapped*

of iterations. The workers' number is function ($f(\cdot)$) of the variable S such that: if S decreases, also W become smaller.

It is desirable to perform the maximum number of iterations (but not less than $SminVal$) and map the computations in the nodes subject to the following constraint: $0 \leq M_j \leq MaxNodeW_j$. $MaxNodeW_j$ is the maximum number of computations that can be executed in parallel by the *j*-th node.

The first two optimization problems proposed can be easily solved by an exhaustive search in the admissible solutions set: usually the number of nodes participating to the same wireless network is not so big. Differently the last one is a multi-objective optimization problem that is convenient to solve through the greedy heuristic reported in Procedure 1.

The greedy heuristic does not takes into account the weights H and K, that in the following will not be used. The vector of fixed costs $\mathbf{B}$ is sorted in ascending order by the *sort()* routine and B_i stores the IP address of the node in the *i*-th vector position. The scatter functionalities are mapped on then node with the minimum fixed cost.

IV. Numerical Results

The effectiveness of the proposed routing algorithms has been proven with numerical results obtained by computer simulations, resorting to the OMNeT++ framework [6].

The considered scenario is constituted by a variable number of nodes from 5 to 30, randomly distributed within a square area of $1\ \mathrm{km}^2$. The nodes are connected among them by using IEEE 802.11g links with $54\ \mathrm{Mbit/s}$, and each node can move to a random arrival point within the area with a speed uniformly distributed between $3\ \mathrm{km/h}$ and $5\ \mathrm{km/h}$; each node remains still for a time interval uniformly distributed between $3\ \mathrm{s}$ and $30\ \mathrm{s}$ [7].

TABLE I
Weight values for each considered policy.

Weights	Policy A	Policy B	Policy C
α	2	2	2
β	6	6	6
γ	0	0	0
Δ, δ	0	1500	1500
Γ, ϵ	0	1	0
ϕ	100	100	100

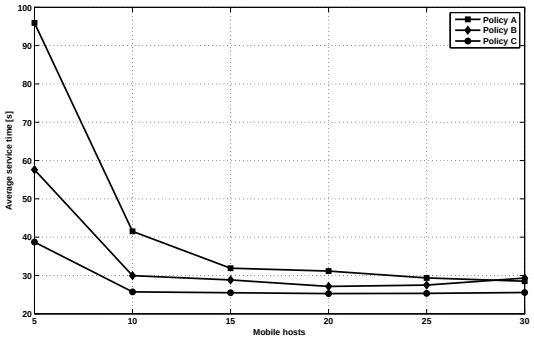
In Figs. 2a and 2b, the average service time has been considered. The service time is defined as the time needed for completing the processing of a certain job, including the processing time by the working nodes and the transmission time among the nodes. We have considered the performance with a variable number of nodes within the area of interest. The tasks to be executed are generated with a rate of $5\ \mathrm{s}$. The comparison is made among the three following policies.

Policy A refers to the simple QOLSR algorithm where no information about CPU, memory and battery energy is taken into account. Policies B and C take into account the supplementary information proposed in this paper. The difference between B and C is that in C we do not take into account the CPU status. We can state that the proposed policies outperform the standard QOLSR policy. Moreover it is possible to note that the influence of the CPU status is very small for the data parallel paradigm; this is due to fact that the processing queue is most affected by memory leakage than CPU status. In Tab. I, the weights for each policy are reported.

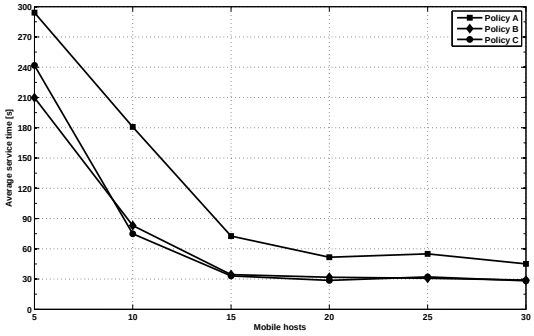
In order to carry the QoS metrics needed by the policy B or C, the QOLSR HELLO and TC messages header and payload dimensions are increased respectively of 16 and 8 bits. The policy A does not require any expansions of the HELLO and TC messages.

In Figs. 3a and 3b, the outage probability is represented. We have considered the same scenario and policies as before. The outage probability is defined as the percentage of tasks not completed within the simulation time. The outage probability is mainly affected by two effects. When the number of nodes is low, they are not in a sufficient number to execute all the tasks, so that their processing queues remain full. On the other end, when the number of nodes is high, their density is high, so that the interference at the communication layer generate an increasing packet loss, and this increases the transmission queues. However, it is possible to note that the proposed policies still outperform the QOLSR performance; for the selected scenario an optimum value for the number of nodes can also be defined.

The effectiveness of the proposed node selection mechanism can be noted by focusing our attention on Fig. 4, where we have plotted the amount of mapped computations on each node for the data parallel scheme. On the x-axis, the ID of each node is considered (we consider here the scenario with 30 nodes), while the y-axis has a double meaning: it corresponds to the percentage of memory allocated in for each node at the beginning of simulations and not freed during the experiments

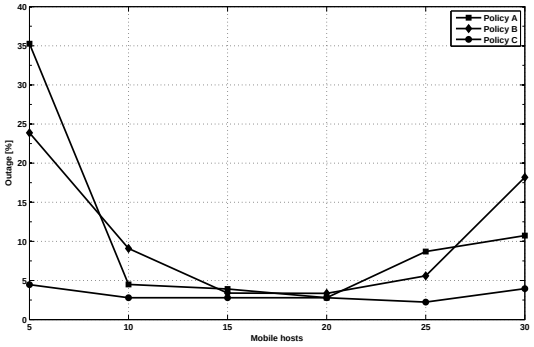


(a) Task Farm

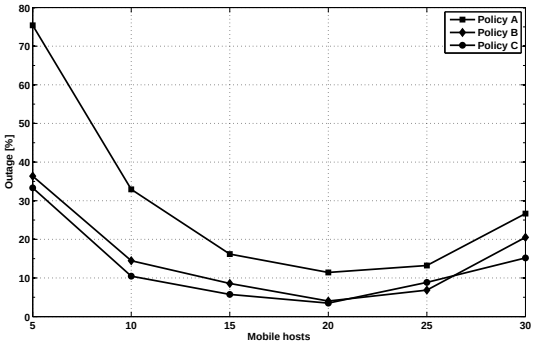


(b) Data Parallel

Fig. 2. Performance in terms of average service time.



(a) Task Farm



(b) Data Parallel

Fig. 3. Performance in terms of outage probability.

(excluding the memory overhead produced by the distributed application) and the amount of allocated computations to each node. It is possible to note that the selection mechanism allocates more computations to those nodes with lower allocated memory; moreover, nodes with the same amount of occupied memory has quite similar number of allocated jobs. This is what we expect for reducing the processing delay. Finally we can state that the policy C should be adopted with the task farm and data parallel paradigms considering the average service time and the outage probability.

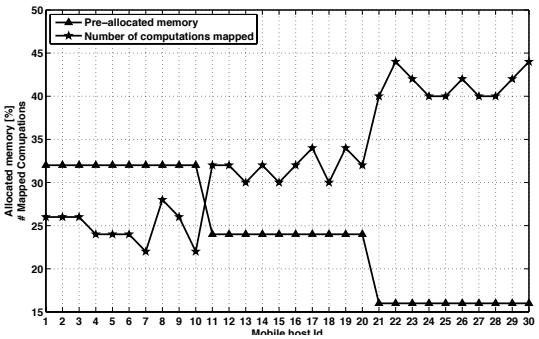


Fig. 4. Computation mapping on the nodes.

V. Conclusion

The demand for even more complex processing in several scenarios has increased the interest toward grid computing; on the other end, the recent advances in the communications field have allowed fast interconnection among nodes with wireless links. Aim of this paper is to propose a routing technique for grid computing in pervasive scenarios where multiple nodes are connected among them for setting up a pervasive grid network. The proposed routing scheme takes into account the characteristics of the considered distributed application allowing better performance in terms of fairness and service time.

References

[1] G. Li, H. Sun, H. Gao, H. Yu, and Y. Cai, "A survey on wireless grids and clouds," in *Proc. of IEEE GCC2009*, Lanzhou, Gansu, China, Aug. 2009, pp. 261–267.

[2] R. Fantacci, M. Vanneschi, C. Bertolli, G. Mencagli, and D. Tarchi, "Next generation grids and wireless communication networks: towards a novel integrated approach," *Wireless Communications and Mobile Computing*, vol. 9, no. 4, pp. 445–467, Apr. 2009.

[3] C. Bertolli, D. Buono, G. Mencagli, and M. Vanneschi, "Expressing adaptivity and context awareness in the ASSISTANT programming model," in *Autonomic Computing and Communications Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2010, vol. 23, pp. 32–47.

[4] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," RFC3626, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[5] H. Badis and K. Al Agha, "QoS routing for ad hoc wireless networks using OLSR," *European Transactions on Telecommunications*, vol. 16, no. 5, pp. 427–442, Sep./Oct. 2005.

[6] Omnet++ community site. [Online]. Available: http://www.omnetpp.org/

[7] C. Sommer, I. Dietrich, and F. Dressler, "Simulation of ad hoc routing protocols using OMNeT++," Mobile Networks and Applications, Jun. 2009, published online.