Scalability of Distributed Dynamic Load Balancing Mechanisms

Alcides Calsavara and Luiz A. P. Lima Jr.
Graduate Program on Computer Science (PPGIa)
Pontifical Catholic University of Paraná - PUCPR
Curitiba, Brazil
{alcides, laplima}@ppgia.pucpr.br

***Abstract*—A load balancing mechanism for large-scale systems should be distributed and dynamic in order to accomplish scalability and high availability. Also, it should be autonomic in order to ease network management. The recent development in utility computing architectures, such as the so-called cloud computing platforms, has increased the demand for such mechanisms. This paper investigates a novel approach, based on the concept of virtual magnetic fields, by which ready-to-start tasks launched on a network middleware are "attracted" to idle nodes. The key issue on such approach is that the update of workload information amongst the cooperating nodes of a network must be a low-cost and an autonomic operation. Two different update algorithms are presented, and their complexity is assessed through simulation. The results show that both algorithms fulfill the scalability requirement.**

Keywords - load balancing; scalability; distributed algorithm; autonomic systems

I. INTRODUCTION

The recent development in utility computing architectures, such as the so-called cloud computing platforms, has increased the demand for load balancing mechanisms, which provide for scalability, high availability and ease of management, amongst other requirements [6][7]. Typically, utility computing architectures are deployed as large-scale complex systems, where changes in resource availability happen very often and in an unpredictable way because, in principle, a task launched at some client node can be assigned to any server node, at any time. Moreover, the geographically distributed nature of such systems makes centralized assignment of tasks to specific servers infeasible, as the corresponding scheduler would become a network bottleneck and a single point of failure. Therefore, a key issue in the development of utility computing architectures is the provision of a distributed load balancing mechanism which is able: firstly, to respond within an approximately constant time regardless network size and topology, i.e., it should scale well; secondly, to proceed responding in the presence of failures such as node crash and network partition, i.e., it should be highly available; and, thirdly, to manage workload information change with a minimum of, preferably none, human intervention, i.e., it should be autonomic.

Several distributed load balancing mechanisms have been previously reported in the literature, as discussed in Section II, and, to the best of our knowledge, their suitability for utility computing architectures are yet to be verified. This paper investigates a novel approach, based on the concept of virtual magnetic fields, by which ready-to-start tasks launched on a network middleware are "attracted" to idle nodes.

The paper is organized as follows. Section II discusses some related work and their weaknesses and strengths. A generic formal model for Virtual Magnetic Fields is presented in Section III, and two workload update algorithms (namely, *QuickPath* and *ShortPath*) are described in Section IV, each taking a different approach to the problem. The algorithms are then evaluated and compared through simulation, and the corresponding results are presented in Section V. Finally, Section VI draws some conclusions and discusses future work.

II. RELATED WORK

The problem of load balancing in an open environment, such as a P2P overlay network, is well discussed in [1], where many complex related issues are listed. Amongst them, the problem of resource discovery, which means to search for idle CPU cycles in this case, is considered as extremely difficult since such resource is perishable, cannot be shared, and is dynamic. Moreover, the set of participating hosts is potentially very large and highly dynamic. The authors compared several methods to solve that problem and they noticed that a hard problem to solve is that large jobs may dominate, resulting in delays for scheduling smaller jobs.

The problem of load balancing is also well studied in the context of grid computing. However, differently from P2P computing, where cycles are obtained from ordinary users in a distributed open environment, in grid computing, cycles are normally obtained from a previously known set of users who agreed to share such resource according to well-defined rules. A corresponding scheduling algorithm enforces such rules and, as well, takes care of proper load balancing. Thus, it can be simpler than a scheduling algorithm for P2P computing. As an example, a dynamic tree-based model to represent a grid architecture in order to manage workload is proposed in [2]. Its main purpose is to improve response time of user's submitted applications by ensuring maximal utilization of available resources through a hierarchical load balancing strategy and associated algorithms based on neighborhood properties. The authors claim to have achieved a reduced communication overhead induced by tasks transferring and flow information. Such solution is based on a group manager who receives, in a periodic way, workload information from

 ISBN:978-1-61208-113-7

each network element.

A discussion on various load balancing algorithms in the context of parallel computing can be found in [8]. The authors also present a list of parameters which can be useful to analyze those algorithms, including *stability*: the cost of workload information transfer versus the benefits of a better overall load balance. According to their research, static algorithms – like round-robin and randomized algorithms – are more stable than dynamic algorithms. However, dynamic algorithms show better overall results than static algorithms when all analysis parameters are considered, especially where fault tolerance is a key requirement.

A comparative study of distributed load balancing algorithms is presented in [7]. The authors discuss the importance of such algorithms in the context of cloud computing, although no particular mapping of the algorithms to real computing platforms is presented. Three distinct algorithms, namely Honeybee Foraging, Biased Random Sampling and Active Clustering, are assessed through simulation in an ideal scenario for cloud computing where nodes are typed to accomplish for node heterogeneity, by using task throughput as the main evaluation parameter. According to their experiments, Honeybee Foraging has a much better throughput than Biased Random Walk and Active Clustering when the number of node types increases in a fixed-size number of nodes; the throughput is slightly better for Active Clustering than for Biased Random Walk. On the other hand, when the number of nodes increases for a fixed-size number of node types, Biased Random Walk and Active Clustering present a much better throughput than Honeybee Foraging, with a small advantage for Biased Random Walk.

III. PROPOSED LOAD BALANCING MECHANISM

Given any two linked nodes, say node *S* and node *T*, a relationship between them can be defined in which *S* attracts messages (which contain tasks) that were initially sent to *T*. Metaphorically speaking, a node S that can attract messages from a neighbor node *T* contains a *virtual magnet* and *T* is within the *virtual magnetic field* produced by such magnet. So, *S* and *T* have a *magnetization relationship*: a source of attraction *S* magnetizes a target *T*. In other words, a magnetization relationship from S to T implies that S can help T to perform its tasks. In particular, any node that possesses a magnet will attract messages to itself. Moreover, the magnetization relationship is defined as transitive, so that, if node *x* (directly) magnetizes node *y* and node *y* (directly) magnetizes node *z*, then node *x* (indirectly) magnetizes node *z*. As a result, the set of all nodes and their magnetization relationships define an overlay network, named as *magnetization network*. For the sake of simplicity, a magnetization network is modeled as a directed graph where each vertex corresponds to a network node and each edge corresponds to a magnetization relationship. The set of nodes which are directly magnetized by a node *x*, denoted as $T(x)$, is discovered by following the edges that leave *x*. Conversely, the set of nodes that directly magnetize a node *x*, denoted as $S(x)$, is discovered by following the edges that enter *x*. By definition, each node is self-magnetized, i.e. any node *x* belongs to both $S(x)$ and $T(x)$. The set of nodes which are either directly or indirectly magnetized by a node *x*, denoted as $T^*(x)$, and, as well, the set of nodes which either directly or indirectly magnetizes a node *x*, denoted as $S^*(x)$, are determined simply by traversing magnetization relationships.

Each node *x* attracts messages according to an associated *strength* or *force*, denoted as $F(x)$, which is set according to local workload information, such as the locally available processing power. So, given a node *x*, the strength $F(x)$ will be determined according to some criteria and, in the general case, it may change over time, as node and network properties change and, as well, as messages are delivered to nodes. For any node *x*, the strongest node in $S^*(x)$ is called the *global pivot* for node *x*, denoted as $P^*(x)$. So, according to the semantics of the magnetization relationship, any message sent to a node *x* must be delivered to $P^*(x)$. Also, for any node *k* in $S(x)$, the *partial pivot for x with respect to k*, denoted as $P_k(x)$, is the global pivot for node *k*, that is, $P_k(x)=P^*(k)$. Thus, given a node *x*, $P^*(x)$ is recursively computed as the node with greatest strength between all nodes $P_k(x)$ where *k* belongs to $S(x)$. Naturally, care must be taken to prevent infinite loops in such computation, since cycles are allowed in the magnetization network. Thus, supposing that tasks are independent from each other and they do not depend on remote data, in the proposed load balancing mechanism, an application message that contains a ready-to-run task is simply attracted, i.e., routed to the node corresponding to the global pivot with respect to the node where the message is originally created. From the implementation point of view, a magnetization network is a network middleware where the life cycle of any application message *m* accomplishes the following steps:

1. *m* is created at the application level and sent to a node *x;*
2. *m* is routed from node *x* to a node *y*, where *y* belongs to $S^*(x)$, such that $F(y)$ is greater or equal to $F(k)$, for any *k* that belongs to $S^*(x)$, that is, $y=P^*(x)$;
3. *m* is delivered to the application level of node *y;*
4. *m* is properly handled at node *y* and, depending on the application semantics, it is either eventually destroyed or kept at node *y* forever.

Every time a change happens with respect to the strength of any node, the magnetic field of such node must be properly updated. Also, as a consequence, the global pivot for any node may change, thus requiring, proper update of its magnetic field, as well. In Section IV, two distinct update algorithms are described and analyzed.

IV. WORKLOAD UPDATE ALGORITHMS

As discussed in Section III, a virtual magnetic field has to be updated when the strength of the corresponding magnet (i.e. its workload) changes significantly, since such magnet strength can be relevant to nodes which are either direct or indirectly magnetized by the node that hosts such magnet. Also, any node can change its global pivot at any time, primarily as a consequence of one or more events of magnet

 ISBN:978-1-61208-113-7

strength change. A global pivot change for a node *x*, on the other hand, can be relevant to the nodes which are directly magnetized by *x*, since their own global pivot can change as a consequence. Hence, this section presents two completely distributed algorithms to keep track of magnetic field information in order to try to guarantee that any application message sent to a node *x* will be attracted to the node that either direct or indirectly magnetizes *x* with the strongest magnet. Naturally, there will always be a chance that an application message is not routed to the ideal (actual strongest) node because the latency of networks may delay the effect of update messages.

Both algorithms work on the basis that there is no centralized component, so every node behaves autonomously and, at the same time, cooperatively with its neighbors with respect to the magnetization network, thus assuming that they can trust each other. That is achieved by defining a common data structure stored by each node in order to keep track of magnetic field information, and a common behavior to internally handle update messages, possibly issuing new ones. Each node is assumed to be uniquely identified and network messages take arbitrary finite time to be delivered, though they never get lost, duplicated or corrupted. In other words, the underlying network protocol does not need to guarantee order, but it has to be reliable.

A. QuickPath

QuickPath is the distributed self-stabilizing algorithm that propagates changes in magnetic fields by updating the strength perception of each node *k* in *T*(x)* using the first notification message that arrives at *k*. The consequence of doing so is that further application messages *m* containing some task that are sent to node *x* will be routed to *P*(x)* through the fastest path (in case there are more than one) determined at magnetic field update time. Once a node *x* receives a notification of magnetic field change, this notification is propagated to *T(x)-{x}* only if either *P*(x)* or *F(P*(x))* have changed. In order to do so, *QuickPath* must avoid remagnetization of nodes. Therefore, if *P*(y)=x*, there must be only one path $k_1, k_2, \ldots, k_n$ from *y* to *x*, in which *P*(* k_i*)=x* ($1 \leq i \leq n$). All other nodes in *T*(x)* that do not belong to $\{k_1, k_2, \ldots, k_n\}$ will be updated with alternative information (typically, second greatest strength and pivot) collected in the update notification path.

This procedure produces two nice properties of *QuickPath*: (a) the algorithm always stabilizes within a finite amount of time for a finite number of nodes (in other words, the number of change notification messages of *QuickPath* is always finite regardless the network topology), and (b) the algorithm updates the perception each node *x* has of *P*(x)* generating acyclic (possibly non-deterministic) routing paths from *x* to *P*(x)*. The first property (a) is consequence of the fact that propagation of change notifications is only carried out if the perceived strength and/or pivot have changed. So, the same message arriving more than once at a node will not cause further propagations. Property (b) derives from the avoidance of remagnetization as explained above.

The core of *QuickPath* can be described as follows. Let *m*={*rm*,*ra*} be a change notification message where *rm*=<*p*,*F(p)*> and *ra*=<*a*,*F(a)*> are pairs with information about the known pivot (*p*) and alternative pivot and their respective strengths (*rm.pivot = p* and *rm.strength=F(p)*). Let *lm* and *la* be similar pairs with information about known pivot and pivot alternative before the receipt of *m*. *QuickPath*'s propagation algorithm is defined by:

```
Node x upon receiving (rm, ra):
begin
   update local info about known pivot and alternative \
      using (rm,ra)
   {lm',la'} = information known after the update
   if (lm'≠lm) or (la'≠la) then
      {pm,pa} = {lm',Max{la',ra}}
      if pm=pa then
         pa = la'      // avoid sending repeated info
      endif
      if pm.strength=pa.strength and pm.pivot>pa.pivot then
         swap(pm,pa)
      endif
      send {pm,pa} to T(x)-{x}
   endif
end
```

The algorithm above always causes propagation of the best alternative pivot known along with de actual pivot information every time the local perception of the magnetic field changes. If the strengths of the main and alternative pivots are the same, always propagate consider pivot the one with smaller *id*, so to break symmetry and avoid propagation loops. Further details of *QuickPath* can be found in [4].

B. ShortPath

In the *ShortPath* algorithm, an application message will traverse the shortest path in case there is more than one path between a node and its global pivot. A more detailed description of the algorithm can be found in [9], where the algorithm is applied for general message routing. The data structure stored by any node *x* consists of the following items:

- *F(x)* : The strength of *x*.
- *S(x)* : The set of nodes that directly magnetizes *x*. For each node *s* in *S(x)*, the following fields are stored: *(1)* The identifier for *s*. *(2)* The identifier for the global pivot for *s*, that is, *P*(s)*. *(3)* The distance from *P*(s)* to *x* with respect to the magnetization network. *(4)* A timestamp corresponding to the local time at *s* when *P*(s)* was set for the last time. Such timestamp is useful to discard related messages which arrive out of order. *(5)* A flag to indicate whether the global pivot information is either up-to-date or obsolete.
- *T(x)* : The set of nodes directly magnetized by *x*. For each node *t* in *T(x)*, the only information stored is the identifier for *t*.
- *K(x)* : The set of nodes known by *x* which either direct or indirectly magnetizes *x*. Hence, *K(x)* is a subset of *S*(x)*. Because of the distributed nature of the algorithm, a node *x* does not know *S*(x)* in advance, so it is discovered as update messages arrive at *x*. For each node *k* in *K(x)*, the following fields are stored: *(1)* The identifier for *k*. *(2)* The strength of *k* currently known by *x*, denoted as $F_x(k)$. *(3)* The timestamp corresponding to the local time at *k* when its strength changed to $F_x(k)$. *(4)* The distance from *k* to *x* with respect to the magnetization network.

 ISBN:978-1-61208-113-7

- *P*(x)* : The global pivot for *x*, which is determined according to the information available on *K(x)*.
- *M(x)* : A node *m* that belongs to *S(x)* such that the strength of the global pivot for *m* is maximum among all global pivots for nodes in *S(x)*. This field is employed to route application messages, since it ultimately leads to *P*(x)*.

All messages exchanged by the algorithm flow according to the magnetization network. So, any message is sent from a given node *x* to a node *y* that is directly magnetized by *x*. There are two types of messages, namely *strength change* and *pivot change*, explained as follows.

A message *m* of type *strength change* contains the following fields: *(1)* The identifier for a sender node *x*. *(2)* The identifier for a destination node *y*. *(3)* The identifier for a node *s* which is the source node whose strength change is being notified by *m*. *(4)* The strength of *s* being notified by *m*, denoted as *F'(s)*. *(5)* A timestamp corresponding to the local time at *s* when its strength changed to *F'(s)*. *(6)* A distance from *s* to *y* with respect to the magnetization network. Such distance is useful for two purposes: (i) to determine the shortest magnetization path between *y* and its global pivot in the case where more than one such path exist; (ii) to detect message loops that may occur due to cycles in the magnetization network. A strength change message *m* which is sent from node *x* to node *y* in order to notify about the strength of a node *s* is handled by node *y* in the following way:

```
if the strength change of s notified by m is relevant
   according to timestamps in m and K(y) then
 1. register into K(y) all data about s contained in m
 2. if the strength of s went down then
      2.1. send a strength change message to every node t
           in T(y) in order to notify about s
      2.2. if s happens to be P*(y) then
          2.2.1. update P*(y) and M(y) according to data on
               S(y)
          2.2.2. if any data regarding P*(y) has changed
               then send a pivot change message to every
               node t in T(y) to notify about P*(y)
```

A message *m* of type *pivot change* contains the following fields: *(1)* The identifier for a sender node *x* which is the source node whose global pivot has changed and is being notified by *m*. *(2)* The identifier for a destination node *y*. *(3)* The identifier for a node*(p* which is the global pivot for *x* being notified by *m*. *(4)* A timestamp corresponding to the local time at *x* when *p* became the global pivot for *x*. *(5) The strength of p* at the time when such node became the global pivot for *x*, denoted as *F'(p)*. *(6)* A timestamp corresponding to the local time at *p* when its strength changed to *F'(p)*. *(7) The distance from p* to *y*. Similarly, to strength change messages, such distance is useful to determine shortest path and to detect message loops. A pivot change message *m* which is sent from node *x* to node *y* in order to notify about the global pivot *p* for node *x* is handled by node *y* in the following way:

```
if the pivot change of x notified by m is relevant
   according to timestamps in m and S(y) then
1. register into S(y) all data about x and p contained in m
2. if either the strength of p notified by m is stale
according to timestamps in m and K(y) or the distance from p
to y notified by m is greater than such distance registered
on K(y)
   then mark x as obsolete on S(y)
   else if the strength of p notified by m is relevant
          according to timestamps in m and K(y) then
           2.1. register into K(y) all data about p
           contained in m
           2.2. if the strength of p went down then send a
           strength change message to every node t in T(y)
           to notify about p
3. update P*(y) and M(y) according to data on S(y)
4. if any data regarding P*(y) has changed then send a pivot
change message to every node t in T(y) to notify about P*(y)
```

V. SIMULATION RESULTS

A preliminary evaluation of the algorithms described in Section IV is presented in this section. The results were obtained from simulation and they show the impact of magnetization network connectivity on the number of update messages, as well as on the time to complete an update cycle and the number of bytes transmitted in each edge. The strength of a node is computed from the amount of resources consumed (memory, CPU, storage space and so on – details are out of the scope of this paper) and it is represented by a value ranging from 0% (the node is completely busy) to 100% (the node is completely idle). For the sake of simplicity, each node has just one server and all servers are capable of processing any task. Simulation was carried out using basically the parameters presented in [3], by randomly scattering a number of nodes over a fixed-sized (1,500x500 meters) rectangular area. The magnetization network was built by assigning a random radial range of influence to each node *x* from 50 to 200 meters; nodes within that range are considered neighbors of *x* (i.e., they are under the magnetic influence of *x*). Next, the magnetization network was a connected graph by consecutively (a) generating its adjacency matrix; (b) computing its transitive closure; (c) generating the set of reachable nodes from each other node; (d) joining sets whose intersection is not empty; (e) joining remaining sets by connecting the nearest nodes belonging to each pair of sets. The number of nodes scattered over the fixed-sized area were multiple of 10, ranging from 10 to 100, in such a way that the magnetization network connectivity and the number of edges per node increased accordingly, thus providing a means to assess its impact on the number of protocol messages. For each number of nodes, 1,000 tests were carried out and the results presented in the sequence correspond to a simple average. Initially, *F(x)* = 100 for all *x* and the measures were taken after producing random strength drops in all nodes simultaneously at instant zero causing a destabilization of *all* magnetic fields.

The number of messages can be large if small strength changes are considered. For that reason, a parameter (not shown in the algorithms), named *threshold* (*th*), was introduced to control when a strength change is relevant, i.e. when a strength change must be notified to the corresponding magnetized nodes. So, when the threshold is set to zero, absolutely any strength change is considered relevant, even the smallest. On the other hand, if, for instance, the threshold is set to 10 then a variation of strength is notified only if it is either higher or lower than 10 units of strength with respect to the last notified strength, otherwise such strength variation is

 ISBN:978-1-61208-113-7

considered irrelevant.

The graphs in Figures 1, 2 and 3 present the results obtained after applying *QuickPath* and *ShortPath* algorithms to a network with N=10 to 100 nodes.

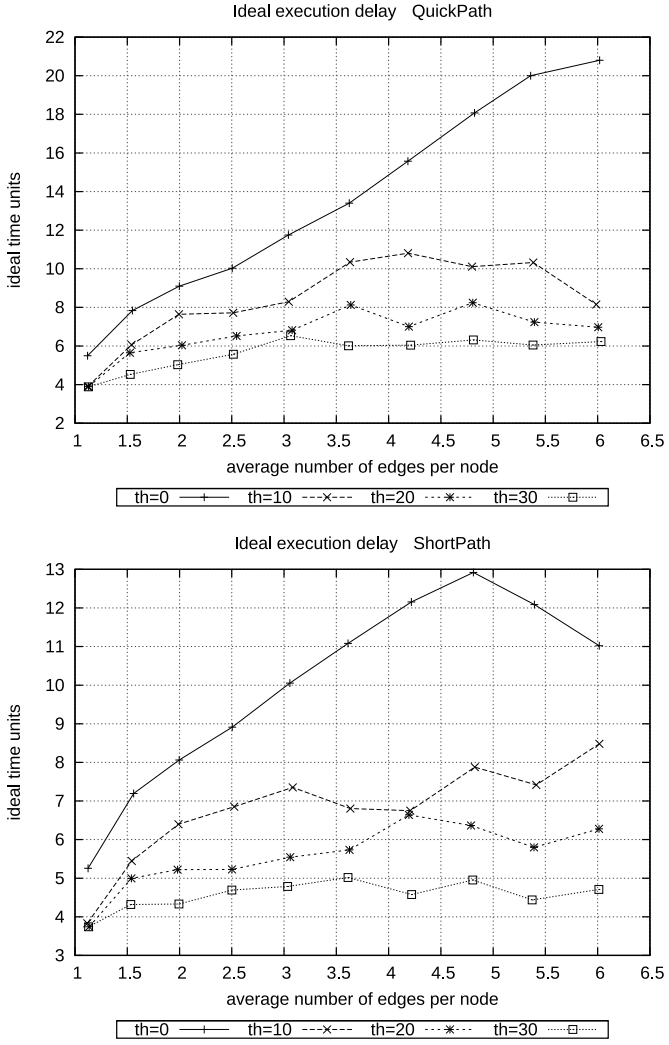

Figure 1. Ideal Execution Delay till stabilization for *QuickPath* and *ShortPath* algorithms with threshold values (th) ranging from 0 to 30.

The number of edges per node ranges from (*N-1)/N* (a tree, since the neighborhood graph is always connected) to approximately 6, as the number of nodes in the fixed-size area is increased.

Assuming that a single message takes *one time unit* to be processed and transmitted over any communication link – the "*ideal time*" [5] –, the graphs in Figure 1 permit to conclude that the total amount of ideal time units tends to stabilize as the number of edges per node grows, thus indicating excellent scalability. Moreover, this tendency is confirmed when the curve with the *threshold* value greater than zero is analyzed.

Figure 2 shows the total amount of messages exchanged by all nodes in order to update all magnetic field information in the network. Notice that the growth of the number of messages in the whole network obeys a quadratic equation as the average number of edges per node grows. However, its growth per node (not shown) can be described by a linear equation, which demonstrates again the scalability of the algorithms. Also, the number of messages issued by *QuickPath* drops faster than *ShortPath*'s as the threshold increases, since *QuickPath* decides to carry out further propagations using a single condition that is greatly influenced by the threshold value.

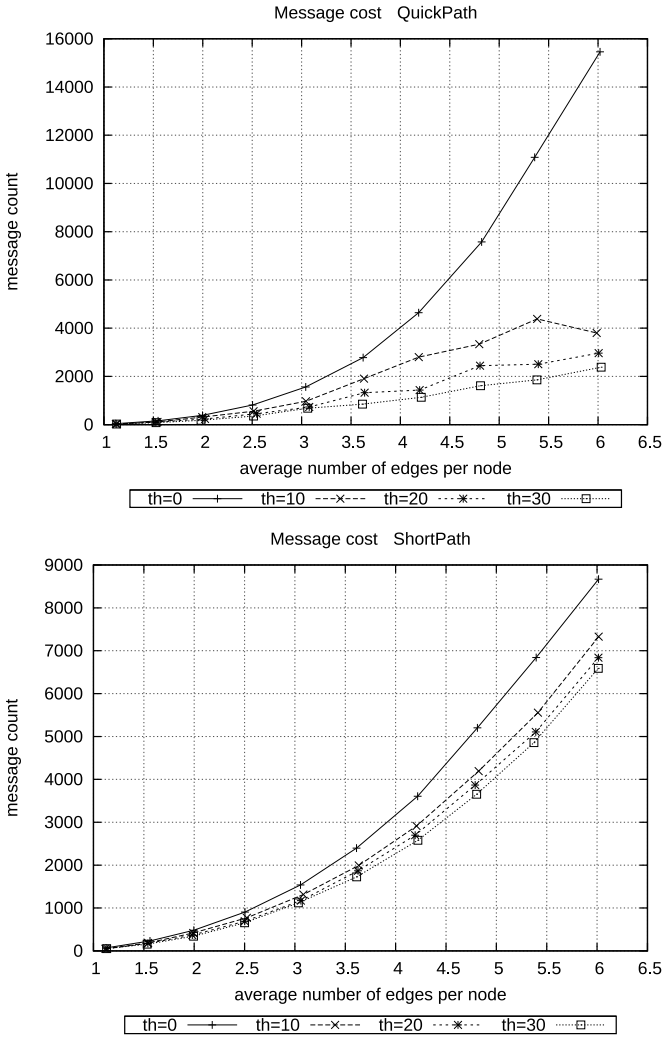

Figure 2. Comparative overall number of messages exchanged by all nodes till magnetic fields stabilization in each algorithm, considering threshold values ranging from 0 to 30.

The average number of bytes transmitted in each link is represented by the graphs in Figure 3. It can be noticed that the curves growth is close to linear, yet again indicating good scalability. Also, it should be noticed that, at instant zero, update messages are sent over *every* edge of the network while, at each time slot, up to the ideal execution delay, the number of bytes transmitted drops till it reaches zero. This happens because, at each step, only relevant information causes further propagations.

In real-world situations (those with *th*>0), the behavior of all the curves indicate good scalability. Just to have an idea of

 ISBN:978-1-61208-113-7

real values, in a network with 1.5 Mbps links, the stabilization delay is approximately 6 ms in the worst case (with a maximum initial destabilization and with *th*=0).

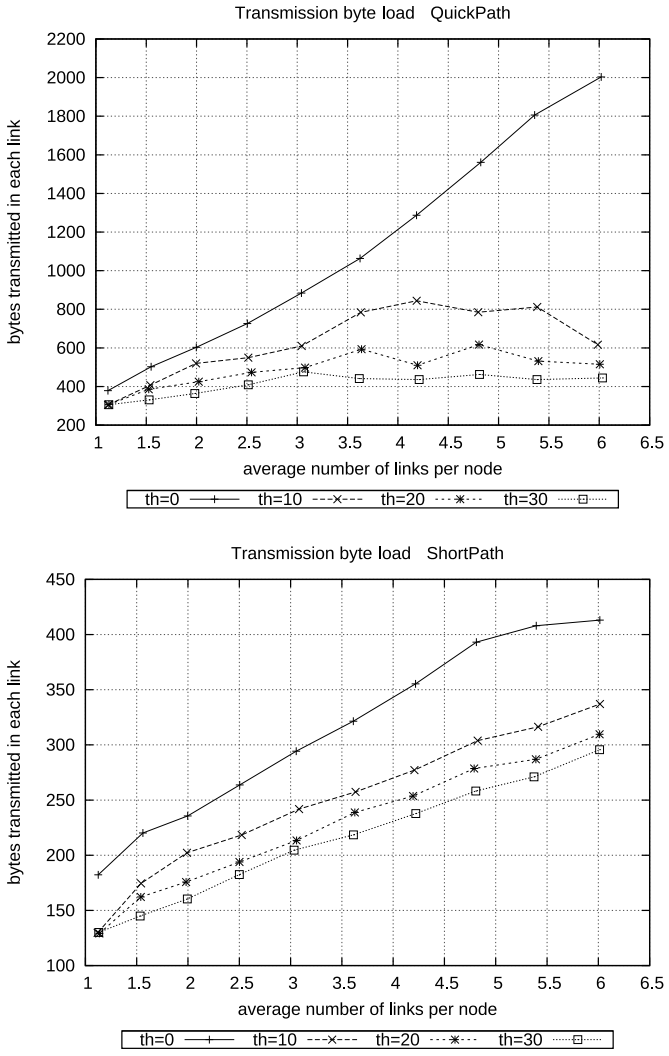

Figure 3. Average number of bytes transmitted in each link for both *QuickPath* and *ShortPath* algorithms with threshold values ranging from 0 to 30.

Other measures like local memory consumption and node workload also show similar results, i.e., linear growth, pointing to good scalability of both algorithms.

VI. CONCLUSION AND FUTURE WORK

This paper presented a novel mechanism for load balancing based on the concept of virtual magnetic fields. There is no centralized global scheduler, as tasks are simply forwarded to the idlest source node, according to the corresponding magnetic field. Also, there is no particular component to manage workload information, as every relevant change of workload in a network element is perceived by all magnetized nodes, recursively, in order to update the corresponding magnetic field. Two distinct distributed autonomic algorithms for dynamically updating magnetic fields were verified by means of simulation so that their feasibility and performance could be evaluated. The results showed that the proposed mechanism is effective, requiring acceptable cost of storage, processing and communication. In fact, the simulation results permitted to conclude that both algorithms scale very well.

The proposed mechanism can be further investigated, as follows. Firstly, the behavior of the mechanism should be verified in continuous systems operation, i.e., when tasks are started anywhere anytime, and also for the cases where nodes are heterogeneous. In such scenario, the efficiency of the proposed mechanism should be verified according to a theoretical analysis on the approximation ration to the optimum solution. Secondly, network faults should be injected to verify the correctness of the mechanism, as well as, the impact of such faults on its success rate. Thirdly, magnetic fields can be experienced in the context of cloud computing and, then, compared to standard solutions to scheduling and load balancing. The implementation of the proposed mechanism on top of a real-world platform will permit to assess its performance in terms of throughput of virtual transport connections, such as TCP and UDP. Fourthly, other important issues, such as trust, reputation and security should be considered. For example, an aspect to analyze is the effect of the transitive property of magnetization relationships on QoS and security requirements. Fifthly, the proposed mechanism can be directly compared to other load balancing techniques, such as those mentioned in Section II, and also techniques based on the swarm approach, on active networks and on mobile agents.

ACKNOWLEDGMENT

This work was partially funded by Fundação Araucária (367/2010 - Prot: 20.063) and CNPq (482593/2010-5).

REFERENCES

[1] Lo, V., Zappala, D., Zhou, D., Liu, Y., and Zhao, S. “Cluster computing on the fly: P2P scheduling of idle cycles in the Internet”. In 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004), pp. 227-236, 2004.

[2] Yagoubi, B. and Slimani, Y. “Task load balancing strategy for grid computing”, Journal of Computer Science, 3(3):186-194, 2007.

[3] Ting, Y.-W. and Chang, Y.-K. “A novel cooperative caching scheme for wireless ad hoc networks: GroupCaching”. In International Conference on Networking, Architecture and Storage (NAS 2007), pp. 62-68, 2007.

[4] Lima Jr., L.-A. and Calsavara, A. 2010. “Autonomic application-level message delivery using virtual magnetic fields”. Journal of Network and Systems Management, 18(1):97-116, March 2010.

[5] Santoro, N. “Design and Analysis of Distributed Algorithms”, Wiley, 2006, ISBN: 978-0-471-71997-7.

[6] Zhang, Q., Cheng, L., and Boutaba, R. “Cloud computing: state-of-the-art and research challenges”. Journal of Internet Services and Applications, 1(1):7-18, 2010.

[7] Randles, M., Lamb, D., and Taleb-Bendiab, A. “A comparative study into distributed load balancing algorithms for cloud computing”. In IEEE 24th International Conference on Advanced Information Networking and Applications, pp. 551-556, 2010.

[8] Sharma, S., Singh, S., and Sharma, M. “Performance analysis of load balancing algorithms”. In World Academy of Science, Engineering and Technology, 38:269-272, 2008.

[9] Calsavara, A. and Lima Jr., L.-A. “Routing based on message attraction”. In 24th Advanced Information Networking and Applications Workshops (WAINA), pp. 189-194, 2010.

 ISBN:978-1-61208-113-7