Journal of Applied Computing Research, 1(2):104-110
July-December 2011

Energy Consumption in Peer-to-Peer Protocols for Ubiquitous Devices

Lucas D. Fonseca, Cicero A. S. Camargo, Mauricio L. Pilla, Gerson Geraldo H. Cavalheiro
Universidade Federal de Pelotas, Campus Universitário, s/n, Cx. P. 354, 96010-900, Pelotas, RS, Brasil.
ldfonseca@gmail.com, cadscamargo@inf.ufpel.edu.br, pilla@inf.ufpel.edu.br, gerson.cavalheiro@inf.ufpel.edu.br

Abstract: This paper studies energy consumption in peer-to-peer protocols in the context of file distribution among border devices and wireless sensors, which are limited in processing power and battery duration. Gnutella, Chord, CAN, Pastry, and Tapestry protocols were compared to the client-server approach in simulations using SimGrid and Triva, analyzing the energy cost of message exchanging. The peer-to-peer protocols presented less consumption to distribute a file among 10 devices, and the best results were achieved by the Chord protocol. The client-server architecture presented the worst results, as expected, due to the high concentration of load in a single server. Processing costs of the Gnutella protocol were compared to the client-server's, with very similar results.

Keywords: Peer-to-peer, Ubiquitous computing, Green computing.

Introduction

The base of pervasive computing is the integration among sensors or embedded systems and the user's environment. These devices suffer from severe resource restraints in terms of processing power, storage space, and battery capacity. Even though there is a natural evolution of technology, it is believed that mobile devices and sensors will remain restrained in comparison to other devices connected to fixed network and power structures (Satyanarayanan, 2001). Therefore, the use of these resources must be optimized to keep functionality and maximize performance while reducing power consumption. One way to help with these objectives is to better distribute the network load among the nodes.

In pervasive environments, the client-server approach may produce unsatisfactory results due to the centralization and, hence, overcharge server nodes, as well as underuse client nodes. This can become an even larger issue when the so-called server nodes are not directly attached to the power grid. The peer-to-peer approach is an alternative where nodes may both generate and fulfill requests.

In this paper, we evaluated the client-server and peer-to-peer approaches in the application level for the distribution of files in a scenario with uniform nodes, with the main metric being energy consumption. Gnutella, Chord, CAN, Pastry, and Tapestry protocols were compared using SimGrid, Triva, and Avrora tools. The TinyOS operating system was used to evaluate computational costs of each one of the protocols.

For the simulated scenarios, Chord presented the smallest energy consumption, due to its advantage when forwarding file requisitions. CAN and Chord presented similar results, but Chord presented slightly better results due to storing more routing information, which may reduce its application for some embedded devices. Tapestry presented the most uniform energy consumption among nodes. The client-server presented the worst results as expected, due to the concentration of load in a single server node. However, in environments where servers are not severely restricted in terms of power, this disadvantage is not important.

This paper is divided as follows. In Section 2, the main differences between the chosen protocols are presented. Then, tools are introduced in Section 3. The simulation workflow is detailed in Section 4. After that, Section 5 discusses simulation results. Finally, Section 6 presents the final remarks and future work.

Related Work

Gnutella (Ripeanu, 2001) defines an architecture where each node sends a message to all neighbors when searching for a resource. The neighbors also forward the message to all its own neighbors, except the one that originally sent it the message. The search for a resource (and the message forwarding) is finished when the resource is found or when the request is forwarded through a given number of nodes, thus avoiding an infinite search. This flooding protocol requires a large processing power and bandwidth, and it does not guarantee access to a resource. To solve this limitation, protocols of structured architecture have been developed, where the network organization is built using a deterministic procedure. The most widely used way to address these issues is to organize resources in distributed hash tables.

In systems using Distributed Hash Tables (DHTs), nodes and resources are assigned a key, usually calculated using a consistent hash function (Silva *et al.*, 2005). The most widely known protocols based in DHT are Chord, CAN, Pastry, and Tapestry (now Chimera). For all the cited protocols, the discovery process of a new node is external to the protocol.

Chord (Stoica *et al.*, 2003) maps each node using an m bit key, hence allowing a maximum number of 2^m nodes in the network. These nodes are ordered by increasing the number of identifier in a ring. Each node knows its succeeding neighbor. Resources are also identified and stored in nodes, accordingly.

Figure 1 presents a Chord ring with five nodes (circles) and three resources (rectangles). As the resources are allocated to the nodes with succeeding identifiers in the ring, resource #1 is kept by node #3, resource #4 is stored in node #5, and so on. Besides its own succeeding node, a given node may know the location of $log\ n$ other nodes to reduce communication costs.

The CAN protocol (Ratsanamy *et al.*, 2001) defines identifiers as points in a virtual Cartesian space with d dimensions. Each node is responsible for an area defined accordingly to its location. Figure 2 presents a [0,2]x[0,2] 2D space with seven nodes and five resources. Nodes are considered neighbors if they have common sides and routing is implemented through a greedy strategy where a message is sent to the node nearest to destiny in the virtual space.

The Pastry protocol (Rowstron and Dunschel, 2001) organizes nodes and resources in a virtual circular space ordered by identifiers, based on the algorithm presented by (Plaxton *et al.*, 1997). Identifiers have 128 bits. Each node has three routing tables. The first one keeps a set of leaves with the nearest nodes. The second table has lines and 2^b columns, N being the number of nodes in the network. Each line corresponds to the node whose prefix has n bits equal to the current node. The last table keeps the nearest nodes according to some metric, such as latency. Figure 3 shows an example of a table for a Pastry node.

The Tapestry protocol (Zhao *et al.*, 2004), now renamed to Chimera is similar to the

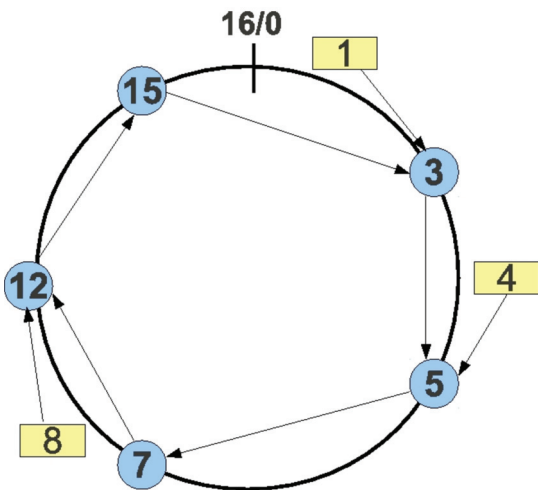

Figure 1. A Chord ring with five nodes and three resources. Source : (Coulouris *et al.* 2007).

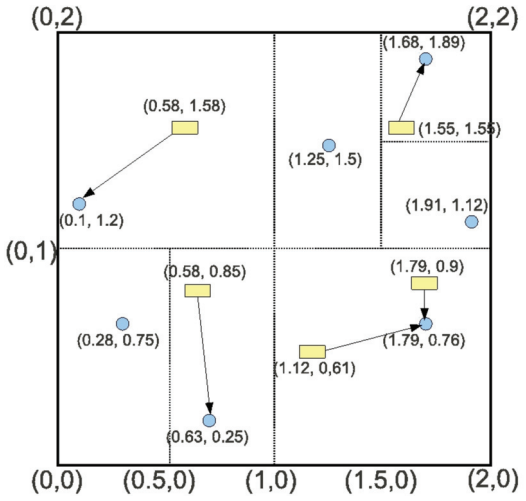

Figure 2. A CAN Virtual Space with dimensions [0,2]x[0,2]. Source: (Coulouris *et al.*, 2007).

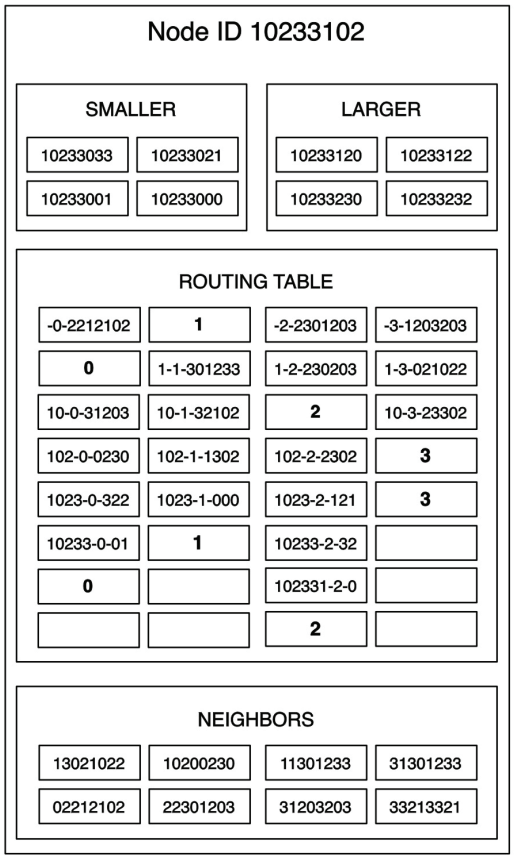

Figure 3. Tables for node 10233102 in a Pastry network.

Pastry protocol. Each node holds a table with information about some nodes. This table is divided in levels, where the *i*-level has the location of nodes with the same *i-1* bit prefix. Hence, searching a key is done in steps in the levels at each hop.

During publishing, the node with the identifier that is nearest to the resource key will be responsible for keeping its content. A publishing message is sent to all the nodes in the first level of the current node. These will forward the message to their second level and so on, until the node that holds the resource is found. To find a resource, it is enough to find a node that is part of the publishing tree to discover its location. Figure 4 shows the publishing and discovery of a resource with id 4378 in a Tapestry network.

Simulation Tools

TinyOS (Levis *et al.*, 2004) is a minimalistic operating system aimed for networks of wireless sensors. It has been developed for systems with severe power constraints. TinyOS offers abstractions for services such as sensing, communication, and storage. Communication among components occurs through interfaces, establishing a hierarchy of components. Execution of applications is done through interactions among user components and operating system.

Avrora (Titzer *et al.*, 2004) is a set tools for simulation and program analysis developed for AVR microcontrollers such as the ones found in ATMel and Mica2 sensors. Simulation is executed in instruction level, with great precision. Networks of sensors can be simulated, with information about energy consumption in Joules, number of cycles per instruction, as well as debugging and profiling functions, memory usage and content, among other possibilities. Simulations generate reports in plain text with all the information requested by the user.

Triva (Schnorr *et al.*, 2010) analyses *paje* trace files from the execution of parallel applications. Together with the GraphViz library (Ellson *et al.*, 2001), Triva presents graphical information about the behavior of monitored applications. It is possible to generate the graph of the simulated network to expose its logical topology.

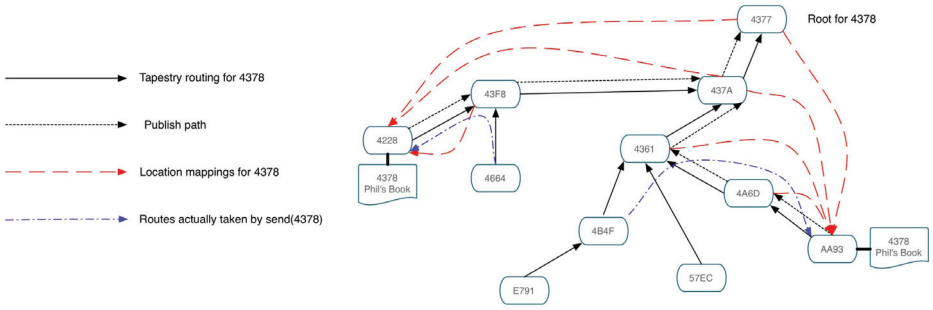

Figure 4. Tapestry routing. Source: (Zhao *et al.*, 2004)

SimGrid (Casanova *et al.*, 2008) is a set of services implemented in C for simulation. The MSG module aims to ease prototyping distributed applications, abstracting details such as communication primitives. Traces are executed and evaluated, presenting the messages exchanged among network nodes.

The first step in the SimGrid environment is to generate the platform for the simulation. This platform is defined in a XML file, generated with the help of the Simulacrum tool. Then, the application to be simulated is defined, and the processes are described in C source files. In the next step, a XML file with the description of the distribution of processes among nodes is defined. After all these configuration files have been defined, it is possible to execute the simulation.

The simulation generates a trace file, which is given as input for the Triva tool. Trace files are analyzed and a graph is generated presenting specific characteristics about the simulation. Figure 5 shows this process.

In the TinyOS environment, first a nesC source file with the application to be simulated is developed. This file is compiled and generates an object file. It is then processed by the *avr-objdump* tool to build an executable file for the Avrora architecture. The simulation outputs information about energy consumption.

Results

To compare the chosen protocols, a test scenario where a network of ten nodes and a file to be distributed was developed. For the client-server approach, ten clients and one server were used, while in the peer-to-peer approaches ten peers were simulated.

Four message types were used:

- Requisitions, sent by nodes that want to receive an information or resource;
- Answers, sent by nodes that received requisitions;
- Messages encapsulating files; and
- Acknowledgments to received file messages.

Requisition messages are very short, hence they are not counted. Answer messages have a short, constant size, empirically chosen as 5 KB. Acknowledgments and file messages were defined as 1 MB. This size was chosen with the objective of generating a significant load in the network, but without congestion. Processing a message has a cost proportional to its size, and each processor is capable of 100 KFLOPS. Transmission rates are only limited by network bandwidth, which was fixed in 125 Kbps. The latency simulated was fixed in 1 μs, simulating a local network environment.

Experiments were divided in measuring message costs, and measuring computational cost of the algorithms. Network measurements were simulated in SimGrid, while the other measurements were executed in TinyOS and Avrora.

Six different file transfer protocols were simulated in SimGrid. One of the simulations implemented a client-server approach. The remaining simulations implemented the peer-to-peer protocols discussed in Section 2: Gnutella, Chord, CAN, Pastry, and Tapestry. Both approaches were developed with the Simulacrum tool (Quinson *et al.*, 2010), with the only difference that the client-server approach has an extra node for the server.

SimGrid's simulations are deterministic, thus there is no point in repeating the experiments multiple times. In Avrora, each experiment was executed 30 times and the average calculated.

Network Simulations

The simulation of the network costs of the protocols for the simulated scenarios produces

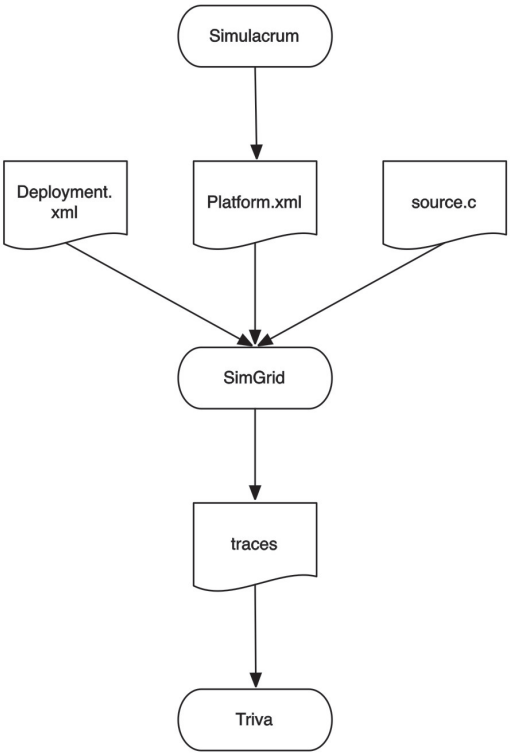

Figure 5. Workflow for the network simulation.

results measured in *energy units* (e.u.), which are not absolute values as they are dependent on specific details of the network equipment used, but are proportional to real energy units such as Joules.

For the client-server approach, the average consumption was 16528 e.u., while the standard deviation is 24700 e.u. As it can be seen, the average is much smaller than the standard deviation, because the amount of energy spent in the server is much larger (90909 e.u. for the server and 9090 e.u. for each client) and thus the individual values do not follow a Poisson distribution.

Figure 6 shows the average consumption and standard deviation for the peer-to-peer protocols. Standard deviation is shown as the error signal on top of each column. The average energy consumption for the Chord protocol is the smallest one, with 627 e.u. CAN is the second best, with an average consumption of 678 e.u. Gnutella presents the largest average consumption, with an average of 1577 e.u., more than two times the energy spent by Chord and CAN, but still less than the client-server. Its worse results are mainly due to not having a distributed hash table and relying on flooding for searches. The standard deviation for Tapestry was 319 e.u., the best result. Gnutella again presents the worst results, with a standard deviation of 981 e.u.

Figure 7 shows the distribution of energy consumption among peers in the Chord protocol. Each column corresponds to a peer, the uppermost part being related to the messages, and the bottom part related to the energy consumption during the file transfers. Notice that it is not necessary to execute an organization step for each file transfer. The large variation on the distribution of energy consumption is due to the fact that each node distributes the file to a variable number of nodes, unlike the client-server approach where the load is concentrated in a server node.

The best protocol may be chosen based on the network characteristics. The protocol that presents the lowest energy consumption during the organization step is the CAN protocol, due to its more simpler construction of an organized space of identifiers. The best protocol in the routing step was Chord. Its advantage is due to being the only one having a routing table with a fixed number of contacts defined from the size of the space of identifiers.

Processing Costs

Two applications implementing the client-server protocol and one of the peer-to-peer alternatives were simulated in the TinyOS environment. Gnutella was chosen in order to get the worst-case scenario for the peer-to-peer protocols and compare it to the client-server approach.

The Avrora tool was used to simulate the nodes and analyze the energy spent. Each simulation was repeated 30 times, and the following results are the average of all executions. Avrora outputs energy measurements in Joules.

Table 1 presents the average energy spent in each node during the distribution of a file using the client-server approach and the Gnutella protocol. For both cases the difference among clients (or peers, in the Gnutella case) are very small. However, the difference of energy spent in peers and clients is small, with 7.2% less energy spent by clients. Even if the energy spent in the server is accounted, the client-server is still more economic in average, requiring 6.5% less energy than Gnutella. But as we simulated an extra server, the total amount of energy spent was larger, with 3.5332 J in client server and 3.4350 J in Gnutella.

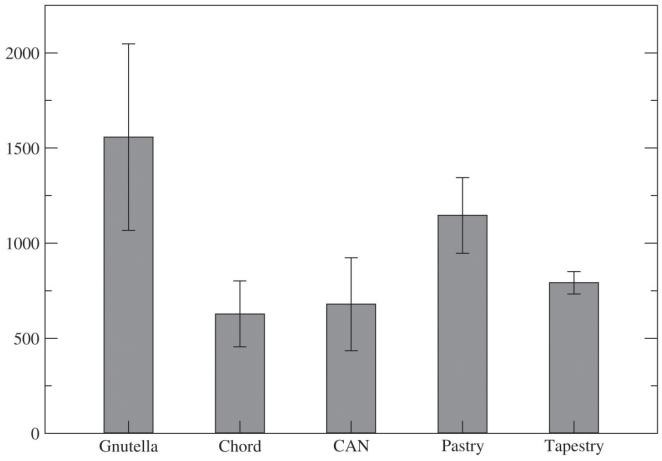

Figure 6. Average energy consumption and standard deviation for peer-to-peer protocols in energy units (e.u.)

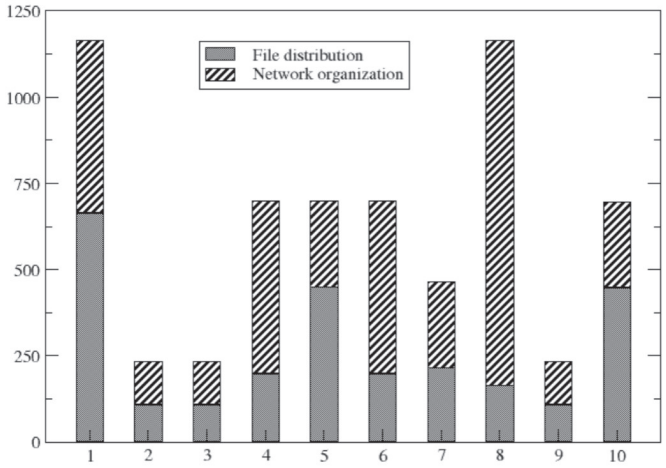

Figure 7. Energy consumption distribution among peers in the Chord protocol.

Final Remarks

In this work, two different scenarios for the distribution of a file to border devices were evaluated. The first scenario evaluated client-server architecture, while the second scenario evaluated five different protocols. Simulations in SimGrid and TinyOS showed that the client-server approach did not distribute nor minimized the energy spent; however, it may be an interesting option when the node that keeps the resource is directly connected to the power grid.

Among the peer-to-peer protocols, Chord was the most economic. On the other hand, the Tapestry protocol presented the most uniform distribution of energy consumption among peers, followed by Chord. During the organization step, the smallest consumption was observed for the CAN protocol. In the routing and file distribution steps, the Chord protocol showed the best results again.

The worst results for the peer-to-peer protocols were those from the Gnutella protocol, mainly due to it not being a structured protocol and lacking distributed hash tables. Hence, for scenarios with a small number of nodes, Chord is a good choice.

In future works, we expect to simulate and evaluate different environments, with emphasis in networks with a larger number of devices. We also intend to implement DHT-based protocols for simulation in the TinyOS to get more precise results in terms of energetic efficiency for these protocols.

Acknowledgments

This work was partially supported by a CNPq Universal grant and FAPERGS/CNPq PRONEX Green Grid project.

References

CASANOVA, H.; LEGRAND, A.; QUINSON, M. 2008. Simgrid: a generic framework for large-scale distributed experiments. *In:* IEEE INTL. CONF. ON COMPUTER MODELING AND SIMULATION, Cambridge, 2008. *Proceedings…*. Cambridge, p. 126-131. http://dx.doi.org/10.1109/UKSIM.2008.28

COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. 2007. *Sistemas Distribuídos: Conceitos e Projeto*. Porto Alegre, Bookman, 792 p.

LEVIS, P. ; MADDEN, S.; POLASTRE, J.; SZEWCZYK, R.; WHITEHOUSE, K.; WOO, A.; GAY, D.; HILL, J.; WELSH, M.; BREWER, E.; CULLER, D. 2004. TinyOS: An operating system for sensor networks. *In:* M. WEBER; J.M. RABAEY; E.H.L. AARTS. *Ambient Intelligence*. Berlin, Springer Heidelberg, p. 115-148. http://dx.doi.org/10.1007/3-540-27139-2_7

ELLSON, E.R.; GASNER, E.; KOUTSOFIOS, E.; NORTH, S.C.; WOODHULL, G. 2001. Graphviz - Open Source Graph Drawing Tools. *In:* SYMPOSIUM ON GRAPH DRAWING (GD), Vienna. *Proceedings...* Vienna, p. 483-484. http://dx.doi.org/10.1007/3-540-45848-4_57

PLAXTON, C.G.; RAJARAMAN, R.; RICHA, A.W. 1997. Accessing nearby copies of replicated objects in a distributed environment. *In:* ANNUAL ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES (SPAA), New York, 1997. *Proceedings…* New York, p. 311-320. http://dx.doi.org/10.1007/s002240000118

QUINSON, M.; BOBELIN, L.; SUTER, F. 2010. Synthesizing Generic Experimental Environments for Simulation. *In:* INTL. CONFERENCE ON P2P, PARALLEL, GRID, CLOUD AND INTERNET COMPUTING, Fukuoka, 2010. *Proceedings…* Fukuoka, p. 222-229. http://dx.doi.org/10.1109/3PGCIC.2010.37

RATNASAMY, S.; FRANCIS, P.; HANDLEY, M.; KARP, R.; SHENKER, S. 2001. A scalable content-addressable network. *In:* CONF. ON APPLICATIONS, TECHNOLOGIES, ARCHITECTURES, AND PROTOCOLS FOR COMPUTER COMMUNICATIONS (SIGCOMM), San Diego. *Proceedings…* New York, p. 161-172. http://dx.doi.org/10.1145/383059.383072

RIPEANU, M. 2001. Peer-to-peer architecture case study: Gnutella network. *In:* INTL. CONFERENCE ON PEER-TO-PEER COMPUTING (P2P), Linkoping, 2001. *Proceedings…* Linkoping, p.99-100. http://dx.doi.org/10.1109/P2P.2001.990433

ROWSTRON, A.; DRUSCHEL, P. 2001. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In:* IFIP/ACM INTL. CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS (MIDDLEWARE), Heidelberg, 2001. *Proceedings…* Heidelberg, p. 329-350. http://dx.doi.org/10.1007/3-540-45518-3_18

SATYANARAYANAN, M. 2001. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, **8**(4):10-17. http://dx.doi.org/10.1109/98.943998

SCHNORR, L.M.; HUARD, G.; NAVAUX, P.O.A. 2010. Triva: Interactive 3D visualization for performance analysis of parallel applications. *Future Generation Computer Systems*, **26**(3):348-358. http://dx.doi.org/10.1016/j.future.2009.10.006

SILVA, A.R.; ALMEIDA, H.M.P.; MACAMBIRA, T.; GUEDES, D.O.; MEIRA, W.; FERREIRA, R.A.C. 2005. Hash consistente como uma ferramenta para distribuição de tarefas em sistemas distribuídos reconfiguráveis. *In:* WORKSHOP EM SISTEMAS COMPUTACIONAIS (WSCAD), Rio de Janeiro. *Proceedings…* Rio de Janeiro, p. 169-176.

STOICA, I; MORRIS, R; LIBEN-NOWELL, D.; KARGER, D; KAASHOEK, M.F.; DABEK, F.; BALAKRISHNAN, H. 2003. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, **11**(1):17-32. http://dx.doi.org/10.1109/TNET.2002.808407

TITZER, B.; LEE, D.K.; PALSBERG, J. 2004. Avrora: Scalable sensor network simulation with precise timing. *In:* INTL. SYMPOSIUM ON INFORMATION PROCESSING IN SENSORS NETWORKS (IPSN), Los Angeles. *Proceedings…* Los Angeles, p. 477-482. http://dx.doi.org/10.1109/IPSN.2005.1440978

ZHAO, B.Y.; HUANG, L.; STRIBLING, J.; RHEA, S.C.; JOSEPH, A.D.; KUBIATOWICZ, J.D. 2004. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, **22**(1): 41-53. http://dx.doi.org/10.1109/JSAC.2003.818784

Submitted on October 05, 2011.
Accepted on December 14, 2011.