SORD: Self-Organizing Resource Discovery Protocol for ALAN

Ioannis Liabotis, Ognjen Prnjat, and Lionel Sacks

Department of Electronic & EE, University College London,
Torrington Place WC1E 7JE, UK
{iliaboti | oprnjat | lsacks}@ee.ucl.ac.uk

Abstract. We present the design and implementation of the protocol for efficient resource discovery and job allocation in Application Level Active Network environments. The protocol is fully distributed and is based on the neighbour lists and small world topologies. Protocol is in XML for interoperability and extensibility; and was prototyped and first tested in the context of the ANDROID project - thus, this paper also describes the ANDROID management architecture. We present the protocol validation in the ANDROID trial, and also give detailed simulation results.

1 Introduction

Current trend in communications and distributed computing is that of peer-to-peer models, independent of centralised control and management. Examples are active and programmable networks, cluster computing and the Grid. Requirement on the decentralised management of these environments is that of autonomy and self-organisation of components. The points of control in these environments will be multiple, where no actor having the global knowledge. Many large network operators are facing these operational issues.

A key requirement in these environments is the ability to allocate the incoming workload (processes) placed on a cluster of (active) nodes in such a way as to optimally use the distributed resources. A node can be considered to be optimal for a process in terms of availability of resources: CPU; memory; encoding hardware; hard resource allocation; availability of particular software. In active networks, work has to be executed in close to real time as the components tend to be part of interactive applications. Thus, in our developments, the work is not queued: success of workload allocation is considered to be almost immediate execution. The resources in these environments are heterogeneous, dynamically changing, highly distributed, and within different domains. Resource availability is dynamic and conditional on local constraints. In such large dynamic environments, centralised resource information repository or resource broker based solutions for resource discovery and information storage could suffer from scaling and efficiency problems. Thus, a light-weight, adaptive resource discovery algorithm, available at distributed nodes, is a preferred solution.

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 127-136, 2004.

Resource management and allocation in active networks is still an under-researched issue. Most systems providing the middleware support for the Grid, such as Globus [1] and Sun Grid Engine [2], rely on the centralised or hierarchical solutions for information repositories and job allocation points. Here we present a decentralised, scalable, adaptive resource discovery protocol initially developed for Application Level Active Network (ALAN) environments, but also applicable to cluster computing and Grid scenarios. Using this protocol, network administrators could effectively distribute the workload so as to maximise resource utilisation and availability, while not relying on a centralised mechanism. Thus, resource location transparency could be achieved and hidden from applications programmer/end user.
Protocol presented here was initially prototyped, and first demonstrated, in the context of the European project ANDROID [3]. The protocol was further developed for cluster computing and the Grid (Self-Organizing Grid Resource Management project [4]), with basic principles and performance assessment presented in [5] [6]. Here we give the details of the protocol, deployment of its prototype in the final ANDROID trial, and new simulation results.

In section 2, we introduce basic principles of ALAN and ANDROID. In section 3, we present the details of our approach. Section 4 gives the protocol specification. Section 5 presents the demonstration of implementation in the context of ANDROID; while section 6 gives simulation results. Section 7 discusses the related work and 8 gives the conclusion.

2 ALAN and ANDROID

ANDROID project (Active Network DistRibuted Open Infrastructure Development) [3][5][7] focused on policy-based management of ALAN-enabled networks. ALAN [8] is the active networks approach where active services (proxylets), engineered by developers/users, are deployed on a group of servers in the existing Internet. Proxylets introduce new service functionality or enhance the level of existing service provided to the user. The active servers provide the execution environment for proxylets, and as such are envisaged to be deployed in large active server farms where various user-specified processes (proxylets) can be dynamically loaded and run. Thus, there is an analogy to cluster computing and Grid paradigms.

The ANDROID management architecture [3] deploys policies as means of control, while events (observations or changes of system state) are used for monitoring and as policy triggers. Policies allow specification of the target behaviour of autonomous components: when an event is received, policies (after conditions based on local information are evaluated) trigger a response. Policies and events are specified in XML and are communicated through the system via the Management Information Distribution system (MID). Persistent storage of policies is provided through the Directory Service (DS), which features a dynamically programmable interface to a range of underlying storage systems.

The core management functionality is policy-controlled security and resource management [3][7]. Security Manager controls the proxylet admission, in terms of

deployer and process authentication, and process access control. Resource Manager [9] deals with proxylet admission control from the resource perspective; resource/process monitoring; and process/resource (re)allocation.

The ANDROID architecture has been deployed in 3 trials. In the final trial, the integrated active networking scenario was conducted, where the architecture was deployed for managing VPNs, multicast services and active server resources and security. The resource discovery protocol has been prototyped in the 5-machine active server farm, and the details of this aspect of the trial are given in section 5.

3 Resource Discovery Protocol: Basic Principles

Our resource discovery protocol is fully distributed, and is based on the ideas of peer-to-peer computing [10]. Each node acts autonomously to identify the best possible node that can serve a request. Each node is connected to a number of neighbours. Initially the neighbours are the nearest topological neighbours (based on geographical location and bandwidth availability between nodes), and a few "*far*" ones. We use a 2-level control mechanism: first is query-based; second is advertisement-based. The query-based mechanism is invoked when an incoming request cannot be resolved by the node that received it. Query time to live (QTTL) determines the number of times a query can be forwarded. The advertisement-based mechanism is activated when a new node appears and advertises its resources. It is also initiated when a dynamic resource changes state, *e.g.* when a node becomes underutilised it can advertise resource availability to its neighbours. Advertisement time to live (ATTL) determines the number of hops that the advertisement can to be forwarded.

Information about resource availability in the network is distributed using the queries-replies and advertisements. This information is cached in the nodes that receive replies and advertisements. Nodes use this information to gain experience on resource availability. Using a different cache for each type of resource, different virtual network topologies are created for each resource. Initially the nodes are connected to their nearest neighbours and some random far neighbours creating a small world topology [11]. As requests arrive and query/advertisement protocols are activated the resource availability caches are populated with state information. The list of neighbours is changing giving separate lists for different resources; generating different virtual topologies.

Fig. 1 depicts the resource management and discovery components on a node. Self-organising resource discovery (SORD) implements the distributed discovery algorithm. It retrieves and stores information on the various caches of the system. There are 3 different caches. "*Known Resources Cache*" contains information about the resources known to the node. This includes the known resources available in the neighbour nodes, which in turn are stored in the "*Neighbour List Cache*". "*Server Meta Data Cache"* contains information on resource availability on the node. Caches can be implemented in different ways: Globus Management Distribution System or ANDROID Directory Service [5]. Our implementation uses ANDROID DS.

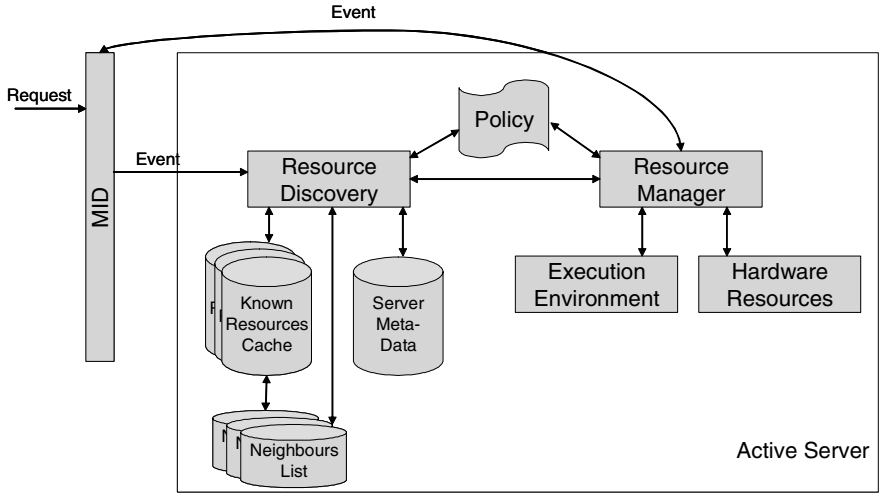


Fig. 1. Resource management and discovery architecture [5]

Typically, user requests the start of the service by sending the relevant events to SORD, which finds the optimal server to run the proxylet, and forwards the event to the resource manager of the hosting machine. Resource manager is responsible for executing a specific request on the local execution environment using the hardware resources available. After performing the resource checks based on policies, the resource manager loads the proxylet(s). Runtime management involves the resource manager monitoring the behaviour of the proxylets, and, in case of unexpected behaviour, applying the relevant policies to preserve the resource integrity and security. Policies can also control modification of service parameters, (initiated by the user or the operator), relating to proxylet reallocation or increase/decrease of resources dedicated to the proxylet. Service termination of is also policy-controlled.

4 SORD Protocol Specification

The protocol allows distributed nodes to communicate, exchange information and locate resources requested by users. The aim is to provide a decentralised mechanism for resource discovery among heterogeneous interconnected resources. Protocol defines the way in which nodes communicate over the network. It consists of a set of messages (Table 1) for communicating information about resources and a set of rules governing the exchange of messages between nodes.

A new node connects itself to the network by keeping in an internal cache the address of at least one other node currently in the network. Using the query/reply and advertisement messages the new node can obtain information about other nodes that participate in the network, and their resources.

Table 1. SORD message description

Message	Description
StartRD	Initiates resource discovery procedure. A node receiving this message is expected to send a "Reply" message to the node that sent the resource discovery request.
Query	The primary mechanism for searching for distributed resources. A node that receives a "Query" message will reply with a "Reply" message with the match found based on the query.
Reply	Response to the "Query". This message provides the recipient with information about resource availability on a node.
Advertisement	The advertisement of resource availability on a specific node.

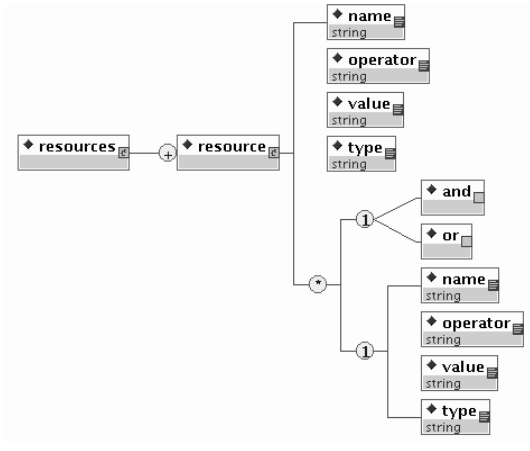


Fig. 2. The resources schema

The protocol uses XML to communicate information between nodes. Messages are exchanged as events sent by the originator to the recipient. Top level event specification follows the ANDROID event specification [3]. The data element of the event contains a *sordmessage*. Every *sordmessage* contains 4 elements: *eventtype* defines the type of the event. (possible values are *StartRD*, *Query*, *Reply*, *Advertisement)*; *ttl* specifies the time to live that is associated with each *sordmessage*; *resources* describes a number of resources; *hosturi* is optional and, if used, gives the URI of the host that has those resources. *Resources* element provides description for resources available at a host, or for resources required by user/application. Resources are de-

scribed using the Globus Resource Specification Language (RSL v1.0) [12]. Each name-value in the description serves as a parameter to control the behaviour of one or more components in the resource management system. Each resource is identified by its *name*, which is an ID known to the management system. The *value* provides either the current or the expected value of the resource. The *operator* determines what values of resource are expected by users/applications. Fig. 2 shows the resources XML schema.

5 Implementation and Demonstration

In the final ANDROID trial, the protocol was used to find an active server suitable for running the proxylet with respect to resource availability and security profile. Our query based mechanism was used to search for the active server that has the required resources. The mechanism is invoked when an incoming request cannot be resolved by the incoming node: a query is sent to the neighbours, which reply about resource availability. The "best" reply reaches the node that initiated the query and the request to run the proxylet is forwarded to the "best" node. Information on resource availability is gathered directly from the active server (*e.g*. Native Library availability). SORD is initiated by the Resource or Security Managers based on the management policies: when a new "run proxylet" event arrives at the Resource Manager and there are not enough CPU resources to run it, the Resource Manager initiates SORD by sending it "eStartRD" event.

Table 2. SORD protocol evaluation

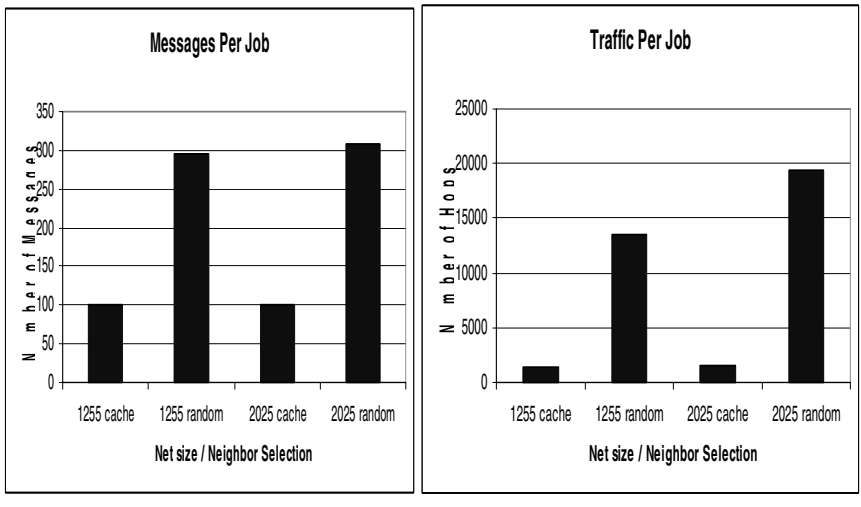
Message Type	Message Sending (ms)	Message parsing, info gathering & evaluation (ms)	Message Size (bytes)
StartRD	40	300	1002
eQuery	36	280	713
eReply	34	210	583

For the trial, we assumed that a user requests to run a proxylet that requires the least loaded active server that also has a CPU load of less than 10%, and the software library. The protocol was evaluated measuring the time that a message needs to be transferred from a machine to another, the time that is needed for parsing, information gathering and evaluation, as well as the size of the messages in bytes. Table 2. shows the values of the evaluated properties for the different types of messages exchanged by SORD. Total time needed for discovery in the scenario that included resource discovery among 5 machines was 1634ms or 1.6 sec. Total traffic generated by was 4890 bytes. Traffic and amount of time required by the protocol depends on the query time to live, the number of neighbours that each node contacts, the available bandwidth and the available processing power. The nodes where connected on a 100Mbit Ethernet LAN, and were powered by Pentium III processors.

6 Simulation

The protocol has been simulated in order to evaluate its performance and fine-tune its properties. Simulation topology was a lattice network of different sizes (number of nodes). The first part of the simulation includes the discovery of one type of resource, (CPU). The request arrival is a random process, and the request duration follows the Gaussian or the Pareto distribution. Metrics used for performance evaluation are: *total network load* (the mean load of all the nodes); *success rate* (number of times SORD returns the best available resource based on the request requirements divided by the total number of requests); and the *number of messages* sent per request. The messages include queries and advertisements. Furthermore we measure the *actual traffic* generated by the messages, defined as the number of hops that a message has to traverse in order to move from source to destination based on the network topology. Note that since we do not consider queuing of jobs/proxylets (since we require immediate execution in case of active network real-time scenarios); the actual job/proxylet loading can fail if there is no resource found.

Each node queries a specified number of neighbours in order to discover the requested resource. We evaluate 2 different types of neighbour selection: random; and selection based on previous replies. Random selection is based on querying random nodes; previous replies selection caches node IDs that previously gave "good replies".



(a) (b)

Fig. 3. Messages and traffic per job request for different network sizes and different neighbour selection policies

Results shown in Fig. 3. (a)(b) were obtained using Gaussian job duration, while the network load was 94%. The success rate of the protocol was 98%. Fig. 3. (a) presents the number of messages generated by the protocol for two different network sizes (1225 and 2025 nodes) and the two different neighbour selection policies (cache

and random). The figure shows that the neighbour selection based on cached information generates 3 times less messages than the random selection. Fig. 3. (a) also shows that the number of messages generated by SORD does not increase as the network size increases from 1255 to 2025 nodes. Fig. 3. (b), presents the actual traffic generated by SORD for 2 different network sizes (1225 and 2025 nodes) and 2 different neighbour selection policies (cache and random). The traffic generated using the neighbours cache is significantly less than the random neighbour selection.

Second set of simulations was conducted assuming that the request duration follows the Pareto distribution. Pareto was chosen since UNIX process lifetimes were shown [13] to follow it. We anticipate that processes running on an ALAN enabled network will also follow Pareto. Fig. 4shows the success rate of SORD for two networks (1225/2025 nodes). While the utilisation of the network is low the success rate is 100%. For utilisation higher than 95% the success rate drops. For 98% and higher, the success rate drops to 92% giving less guarantees that the optimal node will be found. This is not considered as a drawback since we anticipate that active network operators will run their networks with a target utilisation of 75% to 85%.

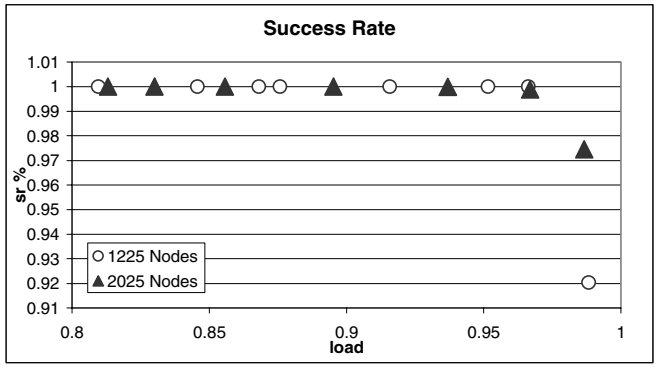


Fig. 4. Success rate of SORD for Pareto request duration

7 Related Work

The details of our work specific to ALAN management in the ANDROID context are presented [3][5][9]: Resource management for the Grid is an active area of research. The Globus management architecture [14] [15] tackles the problem through deploying distinct resource management components: local manager, resource broker and resource co-allocator. The exchange of resource requirements is facilitated via the resource specification language. Resource discovery has been addressed in several application contexts (see [5]). In the context of Grid, different projects have proposed various centralised, hierarchical or distributed algorithms. Globus [1] Nimrod-G [16] and AppLeS [17] use local resource managers (GRAM) to update a directory service with information about availability and capabilities of managed resources. The direc-

tory service is based on LDAP. Condor [18] deploys a centralised *collector* that provides a resource information store which tracks the resource availability. The *Data Grid* project [19] uses a decentralised discovery mechanism and hierarchically organised scheduler based on LDAP. Peer-to-peer resource discovery protocols have been applied to file-sharing systems: Gnutella [20] uses aggressive flooding; Freenet [21] request forwarding and file replication. PEERS project uses iterative deepening directed BFS and local indexes [22]. Chord [23] is a protocol for data discovery in a distributed system that uses key association to data items and key mapping to nodes according to hashing algorithm. [24] proposes a fully decentralised resource discovery mechanism for Grid environments, based on a peer-to-peer architecture using request forwarding algorithms.

8 Conclusion

In this paper, we describe a self-organising mechanism that aids the optimal resource discovery in a farm of Application Level Active Network nodes. The protocol is fully distributed and policy controlled. The protocol is defined using XML events for interoperability and extensibility. We also identify a set of rules that construct virtual overlay topologies over the existing IP connectivity. The protocol has been effectively deployed in ANDROID project trial where it was used to find an optimal ALAN active server to run the user-specified proxylet. Moreover, the protocol was also validated through a set of simulations which demonstrate its efficiency.

References

1. The Globus Toolkit. www.globus.org
2. http://wwws.sun.com/software/Gridware/
3. O. Prnjat, I. Liabotis, T. Olukemi, L. Sacks, M. Fisher, P. McKee, K. Carlberg, G. Martinez, "Policy-based Management for ALAN-Enabled Networks"; IEEE 3rd International Workshop on Policies - Policy 2002.
4. The SO-GRM project: Grant GR/S21939 under the UK eScience program, though EPSRC and sponsored by BTExacT
5. L. Sacks, et. al., "Active Robust Resource Management in Cluster Computing Using Policies", JNSM, Special Issue on Policy Based Management of Networks and Services, Vol. 11., No. 3, September 2003.
6. I. Liabotis, O. Prnjat, T. Olukemi, A. L. M. Ching, A. Lazarevic, L. Sacks, M. Fisher, P. McKee, "Self-organising management of Grid environments", International Symposium on Telecommunications IST'2003.
7. T. Olukemi, I. Liabotis, O. Prnjat, L. Sacks, "Security and Resource Policy-based Management Architecture for ALAN Servers", Net-Con'2002 - IFIP and IEEE Conference on Network Control and Engineering for QoS, Security and Mobility, 2002.
8. M. Fry, A. Ghosh, “Application Level Active Networking”, Computer Networks, 31 (7) (1999) pp. 655-667.

9. I. Liabotis, O. Prnjat, L. Sacks, "Policy-Based Resource Management for Application Level Active Networks", Second IEEE Latin American Network Operations and Management Symposium LANOMS 2001.
10. http://www.openp2p.com/
11. D. J. Watts, “Small Worlds”, Princeton Univ Pr; ISBN: 0691005419; 1999.
12. The Globus Resource Specification Language RSLv1.0, http://www-fp.globus.org/gram/rsl_spec1.html
13. W. E. Leland and T. J. Ott., “Load-balancing heuristics and process behaviour”, in Proc. Joint International Conference on Measurement and Modelling of Computer Systems (ACM SIGMETRICS '86), pages 54--69, Raleigh, NC, May 1986.
14. I. Foster, C. Kesselman, “Globus: A metacomputing infrastructure toolkit”, The International Journal of Supercomputer Applications and High Performance Computing, 11(2): 115-128, Summer 1997.
15. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, " A Resource Management Architecture for Metacomputing Systems", Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, 1998.
16. R. B. David. “Nimrod/g: An architecture for resource management and scheduling system in a global computational Grid”, 2000.
17. F. Berman and R. Wolski, “The AppLeS Project: A status report”, Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997.
18. M. Litzkow, M. Livny, M. Mutka: “Condor – a hunter of idle workstations”, 8th International conference of distributed computing systems (ICDCS 1988)
19. W. Hoschek, F. J. Janez, A. Samar, H. Stockinger, K. Stockinger: “Data management in an international data Grid project.” In “Grid”, pages 77-90, 2000.
20. Gnutella protocol specification, http://www.clip2.com/
21. I. Clarke, O. Sandberg, B. Wiley, T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system”, in “Designing Privacy Enhancing Technologies”, Springer LNCS, New York, 2001.
22. G. Molina, H. Yang, “Efficient search in peer-to-peer networks”, technical report http://dbpubs.stanford.edu:8090/pub/2001-47, October 2001.
23. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. “A scalable peer-to-peer lookup service of internet applications”, Proceedings of the 2001 ACM SIGCOMM Conference.
24. A. Iamnitchi, I. Foster: “On fully decentralized resource discovery in Grid environments” IEEE International workshop on Grid computing, Denver, 2001