Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 33 (2007) 561–571
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Service address routing: a network-embedded resource management layer for cluster computing

Isaac D. Scherson *, Daniel S. Valencia, Enrique Cauich

Department of Computer Science – Systems, The Bren School of Information and Computer Sciences, University of California, Irvine, CA 92697-3425, USA

Received 23 June 2006; received in revised form 22 November 2006; accepted 26 April 2007
Available online 10 May 2007

Abstract

Service address routing is introduced as a novel and powerful paradigm for the integration of resource management functions into the interconnection fabric of cluster computers. SAR provides a "location independent" mechanism for the distribution of computations (services) among the computational resources of the cluster. The intelligence to allocate services and, later on, invoke their instantiation is embedded into intelligent switching devices [Isaac D. Scherson, C.-K. Chien, Least common ancestor networks, VLSI Design 2(4) (1995) 353–364]. Invocation of services is effected transparently to requesting nodes by these network-embedded management functions. Thus, applications can benefit from the inherent parallelism of the cluster while being totally unaware of the specific "location" where required services are rendered. The performance of SAR's service discovery mechanism in hierarchical Least Common Ancestor Networks is studied by means of simulating two different system configurations: level-global knowledge and Level Caches. It is shown that searches using Level Caches work better than level-global knowledge for a typical scientific computing workload.

Keywords: Distributed resource discovery; Service address routing; Intelligent interconnection networks

1. Introduction

Computer clusters have traditionally been seen as a networked set of computational components (nodes) that, by sharing certain system knowledge, provide the ability to work together for solving problems as a *single system* [2,8,11]. The usual implementation of "shared knowledge" is a middleware layer that sits between the Operating System and the Applications. These implementations, while enabling concurrent problem solving, also impose both spatial and temporal overhead to the nodes. A plethora of work has been aimed at making these implementations correct and efficient [4,5,7]. Two strands of development are usually followed: proprietary networks (usually found in supercomputers) and the Internet. The latter is preferred for cluster networks that span vast distances (as seen on the GRID) and for low-budget small networks.

* Corresponding author. Tel.: +1 333 2164 8942.
E-mail address: isaac@ics.uci.edu (I.D. Scherson).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2007.04.005

Recently, there have been attempts to implement client-server architectures [12,13] into this middleware in an effort to alleviate the overhead associated with the service search mechanism, thus enhancing performance. An added advantage to application developers of the client-server model is its conceptual simplicity. SUN's Jini [13] provides a similar approach for workstation farms with a centralized service directory to allow searching nodes by functionality. Although the Jini approach has many advantages, it introduces new challenges due to the bottleneck that represents the central directory. There have been attempts to produce a distributed version, but none has been widely adopted.

Even a distributed Jini implementation resembles a middleware-based system where nodes include an intermediate layer to mask the underlying networked architecture. Nevertheless, these models provide the capability of finding nodes by functionality using distributed searches. However, overhead incurred by the presence of thick middleware layers still affects the nodes' performance.

An alternative to having the intelligence resident in the nodes is to move it down into the interconnection network itself. The architecture that provides this capability is called SAR (service address routing [10]). It allows a computer network to provide a single system image (SSI) with minimal overhead for the nodes by implementing all SSI-related intelligence in the interconnection network. One of the basic ideas behind SAR is to provide a means to identify nodes based on the functionality they provide instead of the physical node address, much like Jini already does. However, the novel idea in SAR is to implement the related intelligence within the interconnection network. In this manner, the nodes are relieved from an important amount of overhead, and the intelligence can be implemented in a topologically-dependent manner without risking the loss of generality. Clearly, services provided by the nodes need to be either registered with the network or distributed among the nodes by the network itself. Once the network acquires the knowledge of services provided by the cluster nodes, it performs the service discovery mechanism that characterizes SAR.

Table 1 shows the basic differences and similarities between CORBA [12], Jini [13] and SAR [10]. Observe that at the heart of these systems is the service discovery mechanisms. CORBA works as a repository of software components while Jini uses a central directory. Using distributed search algorithms would improve Jini's performance by avoiding the bottleneck of centralized searches. However, distributed service discovery in a cluster is a difficult and challenging task that may not be resolved without network hardware changes. Recognizing that network switches have some computational "intelligence" and that a service table directory can be distributed among the switches of the network, the SAR network architecture is proposed to solve the problem of distributed service discovery in Jini-like systems. SAR network efficiency is clearly dependent on the search algorithm implemented among network switches.

This paper focuses on the feasibility and performance analysis of the SAR networks, and their associated distributed search mechanism via simulation. The basic structure of the network is assumed to be hierarchical, and least common ancestor networks (LCANs [9]) were chosen as the basic embodiment. LCANs provide a parameterized description of hierarchical networks, and hence the generality of the simulation results obtained. There are three different search designs for implementing SAR considered herein. The first one is similar to a distributed version of Jini-based architectures. A second one consists of assuming level-global knowledge (a switch knows the information in all switches within its subnetwork). It resembles the Carmen architecture [3], but the design in this work has the advantage of added fault tolerance via redundancy.

Table 1
Feature comparison between CORBA, Jini and SAR

	CORBA	Jini	SAR
Load distribution	Yes	Yes	Yes
Dynamic service allocation	No	No	Yes
Dynamic load balancing (with task migration)	No	No	Yes
OSI layer	Application	Application	Network
Support for grid computing on the internet	Yes	Yes	Yes
Resource management performed in the switch	No	No	Yes
System state information present in the nodes	Distributed	Distributed	No
Memory consistency (single system image)	No	No	Yes
Programming language assumed	None	Java	None

The third approach calls for using fixed-size knowledge at all levels. The latter behaves as a level "cache" that stores the most recently used service information of its subnetwork. The LCANs considered are tightly-coupled interconnection networks for computing clusters, assumed to be circuit-switched optical networks. Using this kind of interconnection, it is possible to envision a tightly coupled computer cluster that spans hundreds of miles (hence applicable to GRID computing). The paper is organized as follows: Section 2 describes SAR and the three implementations considered and Section 3 describes the experimental results obtained.

2. Service address routing

The main idea behind SAR is that nodes are not aware of each other, but of the functionality available for them to use. Nodes do not explicitly send messages to each other, but instead request instances of a particular function (service) to be used. In this manner, addresses do not identify nodes directly, but by the types of services the nodes can provide [10]. The intelligence related to which node will receive a service request is in the network. Although it is assumed to be implemented as a monolithic switch, a hierarchical, multi-module implementation is proposed for scalability purposes using the LCAN as the interconnection design.

The LCAN architecture inherently provides faster simultaneous lookups and redundancy, thus increasing performance and fault tolerance within the system. Each switch has d downer ports and u upper ports, thus making the network a $d \times u$ LCAN. A figure of a three-layer 3×2 LCAN is shown in Fig. 1.

The main source of information for the in-network intelligence of SAR is the service table. In a hierarchical switch (or any distributed switch), that table must be distributed (Fig. 2). Three ways of distributing the service table are proposed, namely local knowledge, level-global knowledge and level caches.

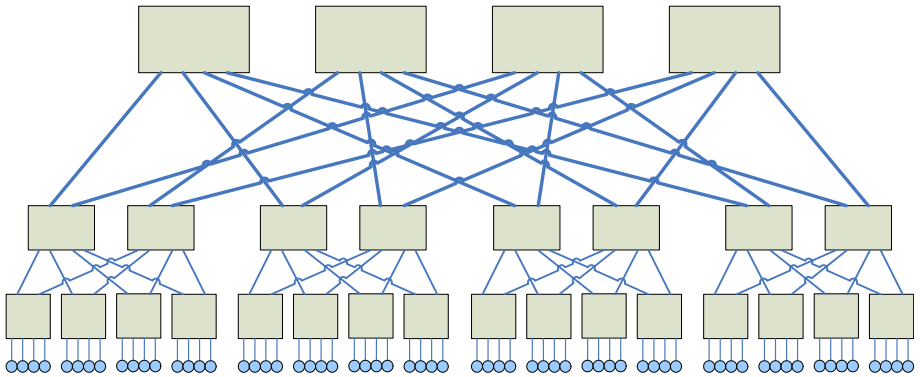

Fig. 1. An example of a hierarchical network is this 3-layered 4×3 least common ancestor network (LCAN) interconnecting 64 nodes of a cluster.

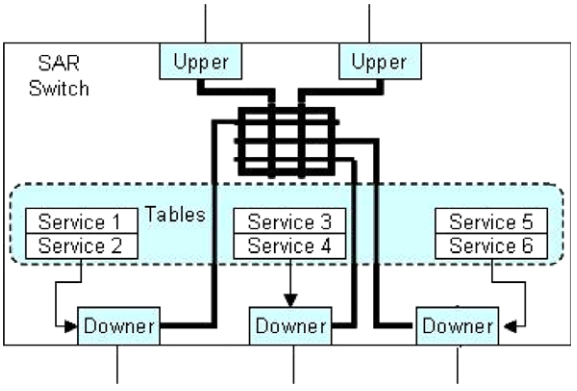

Fig. 2. Internal architecture of a LCAN switch.

Local knowledge consists of having service tables in first-level switches only, so they altogether store one copy of the service table. However, higher-level switches do not contain any information whatsoever. The advantage of this technique is the simplicity of the switches, for their only need is to keep connection states. The overhead derived from service lookup messages is considerably high due to the large number of broadcast messages involved for each service lookup. This behavior can be found in systems like DIET [1]. This model is proposed for completeness only and will not be analyzed further.

2.1. Level-global knowledge

In the originally-proposed knowledge distribution [10], each downer must store information on every specific service type that exists in its subnetwork. The number of service table entries grow exponentially for each higher level of the LCAN, but all switches have a complete view of the sub-network beneath them (Fig. 3). For a switch in level l, the table in each downer will have $T = Su^{l-1}$ entries, where S is the maximum number of different services a node can provide, and u is the number of uppers per switch.

The advantage of this model is that, when a request arrives at a switch, it can be determined if a service provider can be located in its subnetwork (Fig. 3). This implementation would be expected to yield the optimal performance. The challenge is that knowledge stored in the switches is relatively global, which means that complete information is stored, but only for the subnetwork under their umbrella.

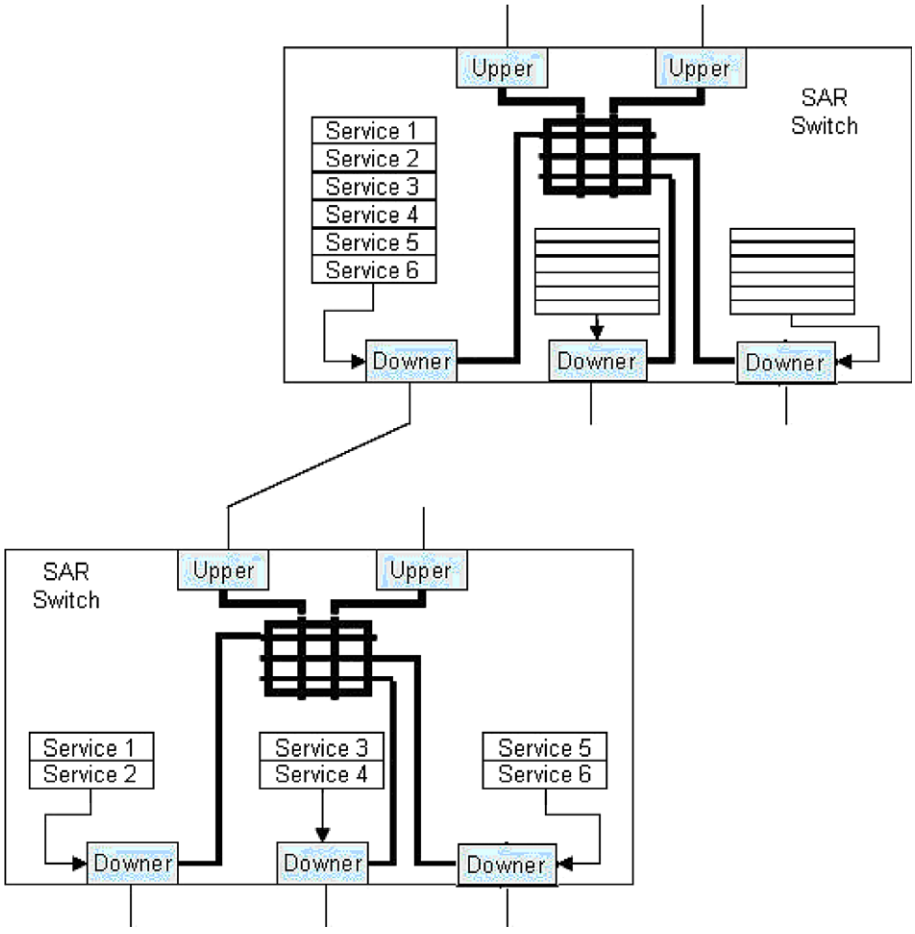

Fig. 3. Service table using level-global knowledge.

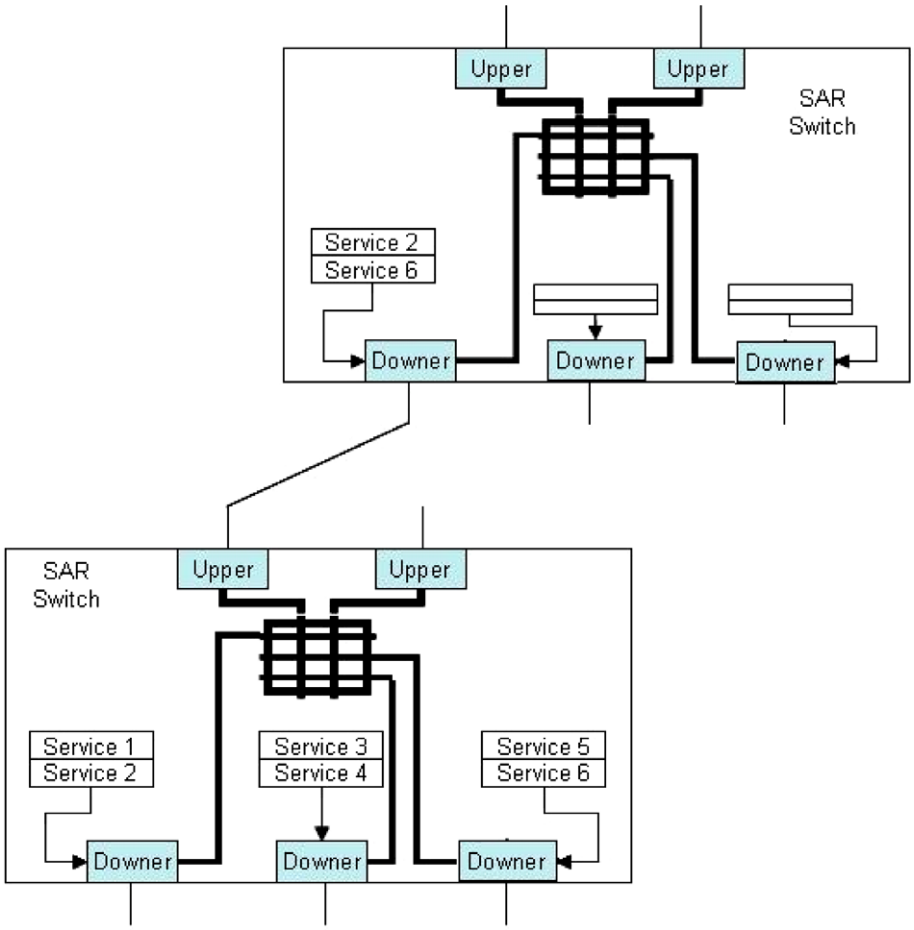

Fig. 4. Service table using caches.

2.2. Level caches

In an effort to alleviate the exponential table growth, a design is proposed where all service tables are of the same size. One advantage of this model is that all switches are identical, regardless of the level (Fig. 4). The drawback of this model is that services are more likely to not to be found in higher-level switches and the replacement of entries is more likely to happen. An algorithm's characteristic is to not be deterred by the lack of information, but manage to maintain an even load distribution by keeping only the most used and important entries by using the temporal and spatial principles in caches.

3. The experimental system

A discrete event-driven simulator was written to compare the performance of a greedy search algorithm on the proposed configurations. The objective of this work is to compare them side-by-side. For the purposes of this work, the network modeled was a 4-level LCAN where nodes were connected to the first-level switches (Fig. 1). The switches were configured with three downers and two uppers, thus making room for 81 nodes. Each node was capable of providing five instances of each service it registered. Each node was able to provide five different services of thirty possible. The type of service was distributed uniformly among the nodes. The network was circuit-switched (resembling purely optical networks) and each link could carry only one signal at a time.

3.1. Service search and allocation algorithm

Upon a request arrival through any port, the switch triggers a simultaneous search in all free downers (the ones that are not participating in any communication interchange at the moment) of the same switch. All ports will reply with at most one hit each, where the ports are prioritized by their index. The number of instances provided inside a subnetwork is fixed and not enough to fulfill the number of requests. The reason is that requests are expected to acquire a large number of instances at once. Thus, the way a switch's subnetworks get fully loaded due to particular downer choices is irrelevant. For this reason, the responding downer with the lower index is chosen. If no hits are to be reported, no servers are assumed to exist and the request is forwarded through the first available upper. Uppers are ranked by index in the same manner because all parents of a switch have the same view of the subnetworks they share. In the event that no uppers are available, the request is sent back to the requesting node with a message indicating the node to reschedule the request for a later time. The request carries the physical address of the requesting node, so that when the provider finished rendering the service, it could request a physical route to the requestor to let it know the outcome.

For the network with level caches, the algorithm was modified so that when the service request message has to use an upper for whatever reason, a Question message is sent through all available downers asking their children switches if they can provide a path for that service. When such a message arrives at a switch and any instance of the service is found in the tables, an update is generated and broadcasted up the LCAN. Otherwise the Question messages are being broadcasted down the subnetworks. A request will not wait for the update to arrive, but instead will keep propagating upwards until a suitable server is found or the request has to be rescheduled. Therefore, the update will only benefit later requests.

Fig. 5 describes in pseudocode the search algorithm's implementation inside a switch, and Fig. 6 includes modifications aimed at making the algorithm work with level caches.

3.2. Workloads

The experiments were run on a modeled system with synthetic and real workloads. Real workloads were obtained from [6], they correspond to the 128-node iPSC/860 located in the Numerical Aerodynamic Simulation Systems Division at NASA Ames Research Center, and the 256-node Cray T3D located at the Lawrence Livermore National Lab. For each request, the traces identify the kind of services to be requested, time of the request, processing time for every specific service required, and the user who produced the request. Other information is also provided in the traces, but for these experiments, just those values are meaningful. It can be seen in the traces that a user is allowed to ask for a set of instances of the same service concurrently, up to 128, with the same processing time (Workload 1 and Workload 3 in the figures). Synthetic workloads

```
If the service is found in at least one downer's local table
        Pick free port with lowest index amongst hits
        If none was picked,
                If request came from a downer,
                        Set to use an upper
                Else,
                        Increase request collision count
        Else,
                If request came from a downer
                        AND switch isn't root
                        AND there's a free upper,
                        Set to use an upper
Else,
        Set to use an upper

If an upper is to be used,
        If the switch is not root,
                Pick an upper which is free

        If switch is root,
                Increase request collision count
        Else,
        If no free uppers,
                Increase request collision count

If a port was chosen,
        Forward Request message through chosen port
Else,
        Reschedule request and break partial connection
```

Fig. 5. Pseudocode describing the implementation of level-global knowledge's search in a switch.

```
If the service is found in at least one downer's local table
        Pick free port with lowest index amongst hits
        If none was picked,
                If request came from a downer,
                        Set to use an upper
                Else,
                        Increase request collision count
        Else,
                If request came from a downer
                        AND switch isn't root
                        AND there's a free upper,
                        Set to use an upper
Else,
        Set to use an upper

If an upper is to be used,
        If the switch is not root,
                Pick an upper which is free

        If level is greater than ONE,
                On each downer that's not the requestor
                        Generate a Question message

        If switch is root,
                Increase request collision count
        Else,
        If no free uppers,
                Increase request collision count

If a port was chosen,
        Forward Request message through chosen port
Else,
        Reschedule request and break partial connection
```

Fig. 6. Pseudocode describing the implementation of level caches' search in a switch.

were produced with uniform distribution of services and requests. The arrival time of the requests was governed by a Poisson distribution (Workload 2 in the figures).

3.3. Level-global knowledge results

The level-global knowledge model has the advantage of locating services immediately if they exist in the same subnetwork and there are enough communication resources (downers and data links) available in the subnet. This model performs better if the requests are uniformly made in the whole system, and the services among the nodes are uniformly distributed across the nodes (workloads 2 and 3 in Fig. 7). This is due to the fact that locality is the most exploited characteristic of the system. The performance advantage also grows if the processing time remains similar across all service requests (workloads 2 in Fig. 7).

In experiments where the requests are not distributed uniformly (workload 1) certain nodes request more services of a specific kind or they ask for more services than others. This system is outperformed by level caches (workload 1 in Fig. 7). The greedy algorithm tries to first satisfy any request with local resources available at hand. For this reason, and also that requestors relatively close to each other may ask for services of different types, it is possible that nodes closer to the requestor become overloaded with requests and nodes

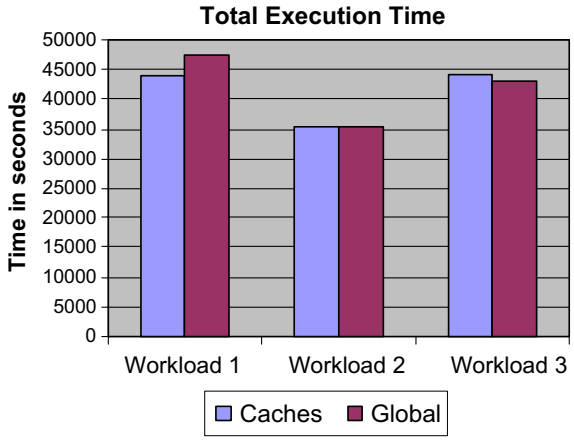

Fig. 7. Total execution time using level caches and level-global knowledge.

from adjacent subnetworks remain unloaded. This scenario could lead to a system that is unbalanced since there would be tasks that do not completely exploit parallelism and resource availability.

In these scenarios, the algorithm starts distributing the load to neighboring subnetworks once the system is already unbalanced. Load balancing techniques, like process migration, are needed to bring the system back to a balanced state. Moreover, the excessive use of locality in the subnetworks increases their local traffic. As a consequence, there is an increase in the number of collisions present in a circuit-based architecture (Figs. 10 and 11), while in packet switched architectures there is an increase in the number of lost packages due to buffer overflow. Therefore, the service lookup algorithms need to consider these factors in order to provide load balancing without causing substantial overhead in the system, or significant increase on the complexity of the switches.

Simulation experiments were performed with traces obtained from real supercomputers. The level caches algorithm proved to reduce the queue time by up to 20% for unbalanced loads, which means that single jobs are served faster than the level-global knowledge algorithm (Fig. 8). The level-global knowledge algorithm needs to consider other variables such as processing time, saturation of the nodes and network traffic with respect to other points of the network, in order to have a more uniform distribution and scheduling of the tasks. Fig. 9 shows how level-global knowledge incurs a more unbalanced system for the first workload, when compared to level caches on the same workload, due to request locality. Although the maximum node utilization is the same, the average score is less by 7%, which means that there were more unloaded nodes when others were fully loaded. This is also reflected by a larger standard deviation. This behavior is not desirable for this kind of architecture because the local systems tend to be saturated with local requests even when remote servers are available and communications are not a major problem for data transfers are done over an optical medium.

3.4. Level caches results

As explained previously, level caches present the same lookup algorithm, with the slight difference that when service information is not present in a cache, a question message is broadcast through the downers available in the switch. This design was chosen because, besides its not increasing traffic in overloaded subnetworks, it promotes the use of those that are available and whose nodes are likely to be without any heavy workload.

It is interesting to point out that fewer collisions and request reschedules were present in service table level caches even when it would seem natural to expect more due to the injection of new traffic by Question messages and cache entry replacement. The number of collisions in the architecture was reduced up to 17% while using real workloads (Figs. 10 and 11). Furthermore, as previously mentioned in some scenarios, the system is able to complete the task faster than level-global knowledge (Fig. 7). This is because in a cold start, when the system does not have the entries for that specific service in its local subnetwork, the switches will forward the

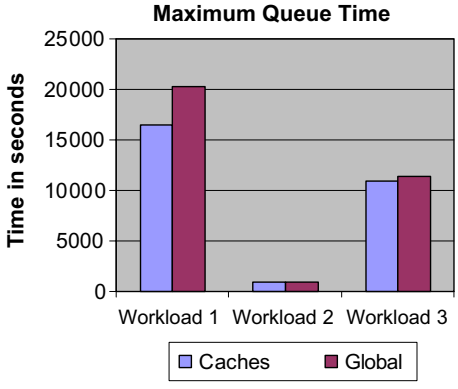

Fig. 8. Maximum queue time using level caches and level-global knowledge.

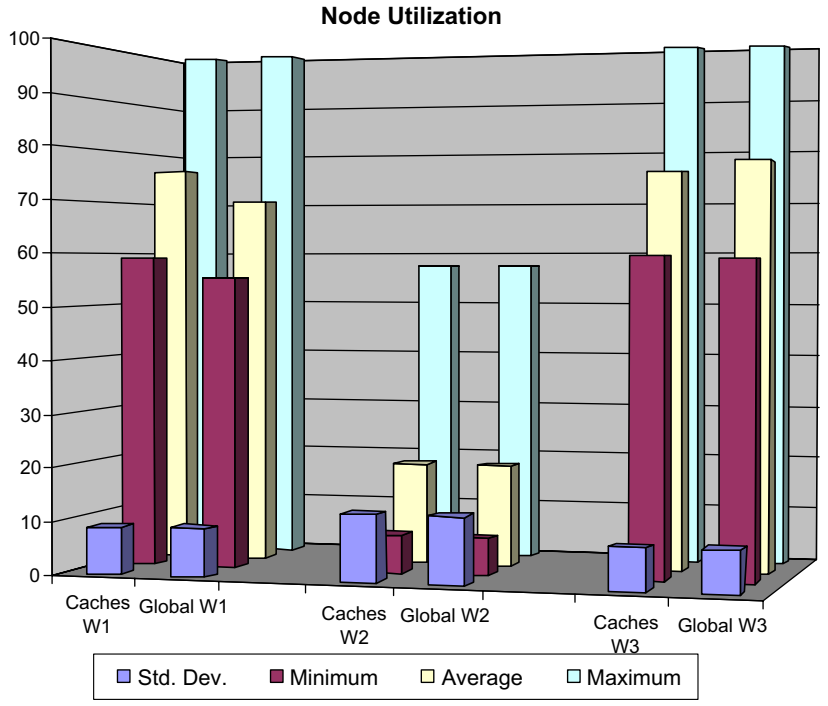

Fig. 9. Node utilization using level caches and level-global Knowledge.

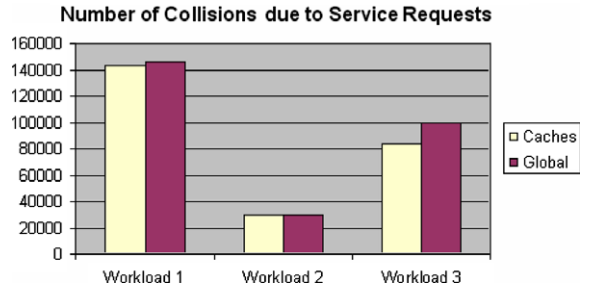

Fig. 10. Number of conflicts due to service requests.

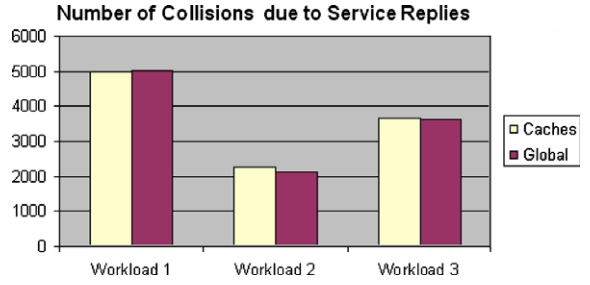

Fig. 11. Number of conflicts due to service replies.

request upwards until the level caches are updated properly. This effectively works as a load pseudo-balancing and distribution algorithm.

It is important to observe two phenomena affecting the level caches algorithm. The first phenomenon is due to the algorithm design; Level Caches between switches of the same level have a tendency to be consistent

between them. It means that the information they have regarding the same subnetwork is similar most of the time. This allows the system to be connection independent, which confirms the statement (Section 3.1) that system balance does not depend on the upper that is taken to forward a particular request because the parents mostly agree in their perspective of the system.

The second phenomenon happens when the switches do not agree in their perspective of the subnetwork and occurs because it is saturated with communications and the updates are rescheduled. In this case, Service Tables tend to have entries containing services that have been used, avoiding question messages to be successfully propagated through congested networks. Therefore, future requests cannot be satisfied through those networks. Instead, the system has preference for unloaded subnetworks.

The most important contribution is that level caches perform very similar to level-global knowledge. The importance of this result is the nonexistent loss of performance when relinquishing global knowledge. This means that this architecture can be scaled without the loss of performance and the increase in complexity that are usually present when scaling a network's size.

4. Conclusions

Service address routing has been presented and characterized for service search and discovery. SAR is an important paradigm applicable at many different levels. In this paper, focus was on tightly coupled scalable computer clusters using least common ancestor networks. The performance was tallied by means of simulation and using incomplete knowledge (level caches) proved more scalable than an equivalent algorithm using level-global knowledge while providing similar performance. Although counter-intuitive, the result is important and opens a new area of discussion on distributed, network-embedded directory implementation and search.

This work opens a new field thanks to the introduction of service address routing using LCANs. As such, much research is possible under the light of this novel idea. The search algorithms presented here are a first step into demonstrating the power of SAR in a hierarchical network such as the LCAN. Further research could be carried out on the impact of service distribution and information distribution on different fully distributed search algorithms. Or, conversely, study the differences between fully distributed search algorithms based on given data processing capabilities of heterogeneous distributed systems. It is also important to study load distribution and balancing algorithms in order to improve, if possible, node utilization.

The use of SAR's unique routing paradigm can benefit existing and future technologies throughout different levels of integration; from nanotechnology to typical distributed systems (computer clusters and the GRID). For instance, in embedded systems, emerging chip design and implementation technologies allow in-network intelligence for systems on chip. Applications of these systems can vary from personal handheld devices (PDAs, cellphones, and GPS) to special-mission unmanned vehicles for military and space exploration purposes.

References

[1] E. Caron, F. Desprez, DIET: a scalable toolbox to build network enabled servers on the grid, Int. J. High Perform. Comput. Appl. (2005).
[2] D. Cornhill, A survivable distributed computing system for embedded application programs written in ADA, Ada. Lett. III (3) (1983) 79–87.
[3] S. Marti, V. Krishnan, Carmen: A Dynamic Service Discovery Architecture, Mobile and Media Systems Laboratory, HP Laboratories, Palo Alto, 2002, September 16th.
[4] MPI: a message-passing interface standard, <http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html>.
[5] MPI-2: extensions to the message-passing interface, <http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html>.
[6] Parallel workload archive, <http://www.cs.huji.ac.il/labs/parallel/workload>.
[7] PVM parallel virtual machine, <http://www.csm.ornl.gov/pvm/pvm_home.html>.
[8] G.F. Pfister, The varieties of single system image, in: Advances in Parallel and Distributed Systems, IEEE, 1993.
[9] Isaac D. Scherson, C.-K. Chien, Least common ancestor networks, VLSI Design 2 (4) (1995) 353–364.

[10] I.D. Scherson, D.S. Valencia, Service address routing: a network architecture for tightly coupled distributed computing systems, in: Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN) in Las Vegas, Nevada, USA., November 2005.
[11] A.S. Tanenbaum, M. van Steen, Distributed Systems; Principles and Paradigms, Prentice Hall, 2002.
[12] S. Vinoki, CORBA: integrating diverse applications within distributed heterogeneous environment, Commu. Mag., IEEE 35 (2) (1997) 46–55, February.
[13] J. Waldo, The Jini architecture for network-centric computing, Commun. ACM 42 (7) (1999) 76–82, July.