Domus – An Architecture for Cluster-Oriented Distributed Hash Tables

José Rufino[1,⋆], António Pina[2], Albano Alves[1], and José Exposto[1]

[1] Polytechnic Institute of Bragança, 5300-854 Bragança, Portugal
{rufino, albano, exp}@ipb.pt
[2] University of Minho, 4710-057 Braga, Portugal
pina@di.uminho.pt

Abstract. This paper presents a high level description of Domus, an architecture for cluster-oriented Distributed Hash Tables.

As a data management layer, Domus supports the concurrent execution of multiple and heterogeneous DHTs, that may be simultaneously accessed by different distributed/parallel client applications. At system level, a load balancement mechanism allows for the (re)distribution of each DHT over cluster nodes, based on the monitoring of their resources, including CPUs, memory, storage and network. Two basic units of balancement are supported: *vnodes*, a coarse-grain unit, and *partitions*, a fine-grain unit. The design also takes advantage of the strict separation of object *lookup* and *storage*, at each cluster node, and for each DHT. Lookup follows a distributed strategy that benefits from the joint analysis of multiple partition-specific routing information, to shorten routing paths. Storage is accomplished through different kinds of data repositories, according to the specificity and requirements of each DHT.

1 Introduction

Certain classes of applications require large *dictionaries*, which are data repositories that store <key, data> records, accessed by its unique <key>.

Distributed Hash Tables (DHTs) are one of the most popular approaches to distributed dictionaries. Research in this domain has been prolific, ranging from 1st generation models, oriented to the cluster environment [1, 2, 3, 4], to numerous recent contributions, most exclusively focused in the area of P2P systems [5, 6].

Despite its abundance, most DHT designs have concentrated on the issues related to a single DHT deployment. Applications, however, may require the simultaneous availability of several DHTs, whether exclusively accessed by a single application, or shared by multiple applications.

Moreover, depending on certain basic attributes (*e.g.*, the type of hash function, the persistence level required for data records, etc.), DHTs may exhibit different properties that are suited to different application scenarios. The lack

⋆ Supported by the portuguese grant PRODEP III - 5.3/N/199.006/00.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 296–303, 2006.

of support for the above desired features, both at the design and deployment levels, lengthens the application development cycle and prevents the resources of the distributed/parallel environment to be more efficiently exploited.

In this paper we present a general description of Domus, an architecture aimed to support the creation, usage and management of *multiple DHTs*, to be *simultaneously* deployed at a cluster environment.

The design is compatible with the possibility to specify, for each DHT, the value of a basic set of attributes, thus supporting *heterogeneity of DHTs*.

At the execution environment, heterogeneity is also conveniently exploited through a load balancement mechanism that builds on the dissociation of the *addressing/lookup* and *storage* functions of the data records, once these functions may impose different requisites on the resources of the cluster nodes.

The discovery of a data record is based on a *distributed lookup* approach, that benefits from the combined analysis of multiple sources of routing information, available at each cluster node, to shorten routing paths.

The remaining of the paper is organized as follows: section 2 revisits previous work on specific issues of the architecture, section 3 presents a general view, sections 4 and 5 elaborate on the main components, and section 6 concludes.

2 Previous Work

Domus architecture gives continuity to our previous work.

In [7, 8] we have presented and evaluated a model for the balancement of the range (*address space*) R_h of an hash function h, over a set of heterogeneous cluster nodes. The model provides for the definition of two basic units of balancement: a) the *vnode*, a coarse-grain unit, and b) the *partition*, a fine-grain unit.

Whenever a cluster node n requires $\#V.n$ vnodes of a DHT, for a global number of $\#V$ vnodes then, ideally, n should be responsible for the fraction $Q^i.n = \#V.n/\#V$ of R_h, which defines the node's *ideal quota*. However, in reality, nodes are given a certain number of *partitions* of R_h, which may be viewed as slices of R_h, all of the same size, and mutually exclusive. Thus, if a node n is bound to $\#P.n$ partitions, for a global number of $\#P$ partitions, the node's *real quota* of R_h is given by $Q^r.n = \#P.n/\#P$.

Vnodes and partitions relate as follows: for each one of its vnodes, a node will be given a certain number of partitions; this number is dynamically adjustable so that, for every cluster node n enrolled in the DHT, $Q^i.n$ and $Q^r.n$ are maintained as close as possible, at every moment, even when the number of nodes that support the DHT and/or their ideal quotas change.

Depending on the number of nodes that support a DHT, and on the maximum deviation allowed between real and ideal quotas, the overall number of partitions bound to each node may be considerable, preventing any cluster node to maintain the full <partition, node> assignment/addressing table.

A well known solution to the previous problem is to interconnect the partitions of the DHT by an *application level topology* that supports a distributed lookup mechanism. Under such mechanism: a) each node needs to keep only a

small (logarithmic bound) number of <partition, node> associations, deterministically defined and b) looking up for any partition involves the participation of a limited (logarithmic bound) number of nodes.

However, in the context of our work, the direct application of this solution to our models is not recommended. Because a cluster node may be responsible for multiple partitions of the same DHT, it may be visited many times if a *routing chain* is exclusively based on the *conventional* distributed lookup algorithms, where lookup requests "hop between partitions".

We have thus studied algorithms for *aggregated routing* [9], suitable to *Chord* [6] and *de Bruijn* [10] graphs, which fit naturally in our models for the weighted distribution of the hash function range. By using *aggregated routing*, the combined routing information of various partitions is investigated to ensure that lookup requests "hop between cluster nodes", leading to shorter routing chains, when compared to conventional routing.

3 General View

A typical Domus deployment, as shown in figure 1, builds on four major components: i) *client applications* (a_i); ii) *DHTs* (d_j); iii) *services* (s_k); iv) *cluster nodes* (n_l). External applications and services, not represented, may also coexist. A Domus deployment or instantiations is named hereafter as *Domus cluster.*

In the same figure, *base services* allow for: a) the interaction between applications and/or services, through high performance message passing (as provided by RoCL [11]), b) the global monitoring of resources (as provided by Ganglia [12]) and c) the remote execution of services (*e.g.*, via RoCL, Ganglia or `rexec`).

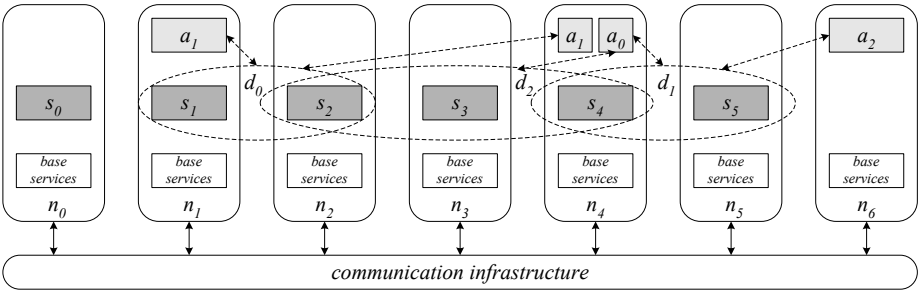

Fig. 1. A typical Domus cluster

Figure 1 also illustrates some basic architectural possibilities and constraints:

- one client application may access multiple DHTs;
- one DHT may be accessed by multiple client applications;
- one DHT may be implemented by multiple Domus services;
- one Domus service may support multiple DHTs;
- one cluster node runs no more than one Domus service, per Domus cluster.

Client applications interact with Domus services to explore DHT abstractions or for administration purposes. Interactions are conducted through the facilities offered by a user-level library which, among others, includes methods to: a) create/destroy, open/close and shutdown/restart a Domus cluster; b) add/remove Domus services; c) create/destroy, open/close and shutdown/restart DHTs; d) insert, modify, read, browse or remove data records from DHTs.

Domus also supports the user-level definition of a basic set of attributes for each DHT: a) the hash function; b) maximum deviation between ideal and real quotas of the DHT, for any node; c) initial distribution of the DHT; d) redistribution constraints; e) balancement thresholds; f) routing overlay and algorithms; g) storage media and block size; h) data access pattern (read-only/read-write).

4 Domus Services

Domus services are the fundamental building blocks of the Domus architecture: it's their cooperation that enables to deploy, access and manage multiple DHTs.

At each node, Domus services comprise an *addressing subsystem* (AS), a *storage subsystem* (SS) and a *balancement subsystem* (BS).

The AS and SS subsystems support the *address space* and *storage space*, respectively, of one or more vnodes, eventually from different DHTs. The BS subsystem monitors the utilization of certain node resources and, if necessary, triggers the migration of local vnodes to another Domus services.

The address/storage space of a vnode is the union/sum of the address/storage space of its partitions. The address space of a partition is its own slice of the hash function range. The storage space of a partition refers to the storage resources consumed to hold the data records whose keys map onto its address space.

The separate management of the address/storage space of vnodes and partitions allows increased flexibility: such spaces may be independently managed, by different services[1], which has proved to be useful for balancement purposes.

The possible enrollment of a service in more than one DHT, and at several levels, is illustrated by figure 2. The figure shows how the AS and SS subsystems of services $s_1, ..., s_5$ support the DHTs $d_0, ..., d_2$, for the scenario of figure 1.

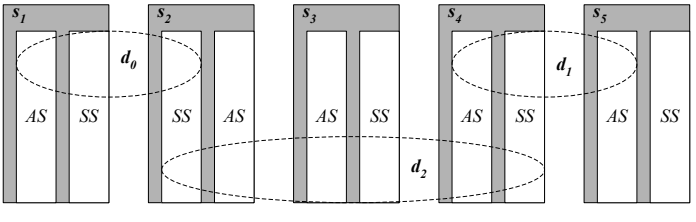

Fig. 2. A possible association of the AS/SS subsystems of $s_1, ..., s_5$ to $d_0, ..., d_2$

[1] Named *addressing/storage services* of the vnodes or partitions (the services whose AS/SS subsystem manage the address/storage space of the vnodes or partitions).

4.1 Addressing Subsystem

Lookup of Partitions. For a large number of partitions, a distributed approach to partition lookup is advisable. In our case, that approach must also be compatible with the decoupling of the addressing and storage functions of partitions; such implies that the lookup of a partition may involve two distinct lookups: i) the finding of its addressing service and ii) the finding of its storage service.

Domus supports the interconnection of the overall partition set of any DHT by a routing overlay, such as a *Chord* or *de Bruijn* graph, thus requiring the definition of a *routing table* (RT) per partition, residing at the partition's addressing service (in the AS subsystem). Each RT maintains *addressing references* (AR) to the addressing services of other partitions of the same DHT. Along with its RT, a *storage reference* (SR) to the storage service of the partition is also preserved.

Thus, finding the storage service of a partition starts by locating its addressing service, in order to obtain its SR reference. Relying in addressing services for the maintenance of SR references ensures that finding a storage service benefits from the same distributed lookup mechanisms used to find an addressing service.

Aggregated Routing. A Domus service may be the addressing service for several partitions of the same DHT. In such case, there will be many RTs locally available for that DHT. The combination of the totality or part of the RTs of a DHT, available at each service, allows routing decisions to be performed under enhanced algorithms (*aggregated routing*), that usually lead to shorter routing chains than *conventional routing*, which bases routing decisions in a single RT.

To make aggregated routing faster, all the addressing information available at each Domus service is brought into *addressing indexes* (AIs), one per each DHT. For a partition uniquely identified by its PID, the AI index maintains the RT table of the partition, its SR reference and an *access counter* (AC).

Figure 3 shows the AI index maintained by services s_3 and s_4, at their AS subsystems, for DHTs d_2 and d_1, accordingly to the roles defined in figure 2 for those services. SR references are explicitly represented, by dashed arrows.

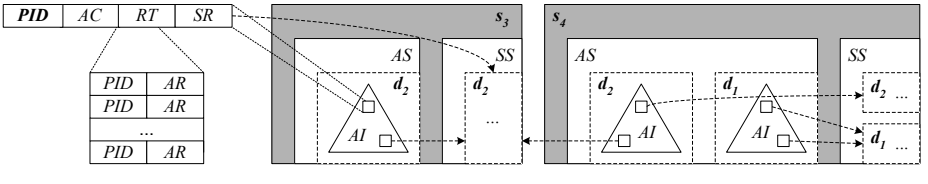

Fig. 3. A representation of the AI indexes of s_3 and s_4

Load Balancing. The AS subsystem measures the access rate to the local RTs of a DHT and, if this statistic surpasses a certain threshold on the average access rate per vnode, it will trigger the creation of a new vnode. The new vnode will "steal" some partitions from the others, thus the average number of partitions per vnode decreases which, in turn, lowers the average routing load per vnode.

The AS subsystem may also switch routing algorithms, if needed. Routing algorithms supported in Domus, for *Chord* and *de Bruijn* graphs, range from conventional routing methods, to more complex algorithms for aggregated routing; the first class of algorithms is lighter, but routing chains are longer; the second class is heavier, but routing chains are smaller. Because all these algorithms converge to the same final answer of a lookup request (though some converge faster than others), different algorithms may be used along the same routing chain.

4.2 Storage Subsystem

Repositories. The data records whose keys map onto the address space of a partition are saved in a per DHT *repository*, at the storage service of the partition. In this context, a repository is any data store capable of holding *dictionaries*.

The SS subsystem supports any kind of repository (and multiple instances), if provided proper *midleware* to interact with it. Each kind has specific properties, relevant to different application scenarios. Any Domus deployment should support at least a RAM-based (volatile) and a disk-based (persistent) repository.

The local repository of a DHT is identified by a *repository reference* (RR) – see figure 4. This is used to access the repository trough a *repository abstraction layer* (RAL) that presents a generic/unified interface to repositories, irregardless of their base storage platform. The RAL layer maps a high-level RR reference to a low-level handler, used to access the repository via appropriate middleware.

A specific *storage index* (SI) per DHT is also preserved. For each partition locally stored, it contains its PID, the *addressing reference* (AR) to its addressing service and a *storage counter* (SC). Conveniently, if the AS subsystem is allowed to inspect the SI indexes of the SS subsystem, a routing chain may end sooner.

Figure 4 magnifies the SS subsystem of s_4, already visible in figure 3; it shows the SI index for d_2 and d_1, including some AR back-references (also to s_4).

Load Balancing. Domus provides several mechanisms to balance the consumption of storage resources. Firstly, during the creation of a DHT, a correspondence between a vnode and a certain amount of storage is established; thus, if vnodes

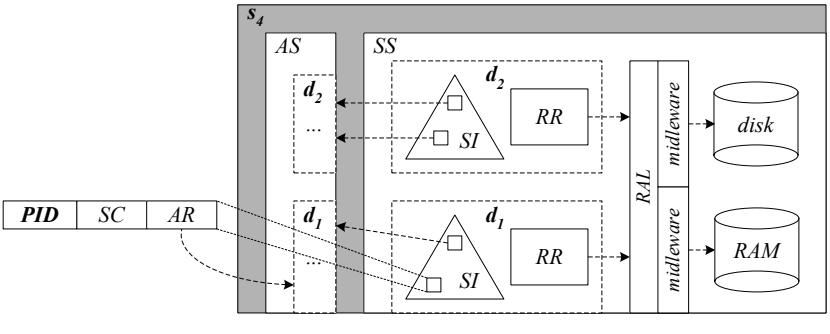

Fig. 4. A representation of the inner structure of the SS subsystem of s_4

become "full", some options are: a) to delete data records (the DHT may behave like a cache), or b) to create more vnodes for the data records in excess. Secondly, Domus may enforce fragmentation of user-level data records into DHT-level data blocks; these are uniformly spread over the partition set of the DHT, which has a twofold effect: it ensures that both the storage consumption and the access load are also uniformly spread over the Domus services that support the DHT.

4.3 Balancement Subsystem

The BS subsystem monitors the load on local resources (like CPU, main memory, disk and network), with the help of specialized tasks, provided by the base services (*e.g.*, Ganglia [12] and Domus-specific plugins). Such tasks collect raw information about the load/availability of node resources and maintain utilization statistics (*e.g.*, exponential moving averages). If the utilization of a resource surpasses a certain threshold, the BS subsystem triggers a *balancement event*.

The processing of a balancement event involves the selection of the set of RTs or data records, of a certain local vnode, that should be moved to another Domus service, in order to diminish the pressure on the overloaded resources. The selection is based on the analysis of the access rate and storage utilization statistics, maintained by the AS and SS subsystems, for RTs and data records. A *supervisor* service choses the service where RTs or data records should be moved.

5 Supervisor Service

Some tasks in a Domus cluster may require global coordination, such as: a) the creation/destruction, shutdown/restart of the Domus cluster; b) the adding/removal, shutdown/restart of specific Domus services; c) the creation/destruction, shutdown/restart and redistribution of specific DHTs. These tasks are conducted under the supervision of a well known Supervisor service).

The expansion of the set of Domus services, of a Domus cluster, may be *administrative*, as required by an administrative application, or *automatic*, as the result of the processing of balancement events. Contraction is administrative, typically motivated by the need to detach some cluster nodes from a Domus deployment; in such scenario, the addressing and/or storage responsibilities of the affected services must be transfered to other services, in other cluster nodes.

Shutting down a DHT requires saving all its RTs and data records in persistent repositories, at the cluster nodes that host the Domus services enrolled in the DHT. Afterwards, the DHT will enter an *off-line* state. The restart of a DHT brings it back to the *on-line* state. Shutting down a Domus service involves the local shutdown of its supported DHTs. Finally, shutting down an entire Domus cluster involves the shutdown of the Domus services and of the Supervisor.

Balancement events are serialized by the supervisor and comprise the following two phases: 1) the discovery (or instantiation) of Domus services with the necessary available resources, and 2) the coordination of the transfer of RTs or data records, from the overloaded service to the selected recipients.

6 Conclusions

The main contribution of this paper is the definition of an architecture for cluster-oriented DHTs, aimed to support multiple, heterogeneous and balanced DHTs.

The design allows the user-level specification, separately for each DHT, of a basic set of attributes, providing for heterogeneous DHTs.

It also supports the decoupling of the routing and storage functions of DHTs, as a strategy to achieve global load and storage balancement, in the highly dynamic execution environment of the cluster.

To our knowledge, Domus is novel in the combined support for all these features, and also in the distributed lookup enhancements that allow to accommodate aggregated routing algorithms.

A prototype version of Domus is being developed in the context of SIRe[2], as a simple implementation of the work described here.

References

1. Litwin, W., Neimat, M.A., Schneider, D.: LH*: Linear Hashing for Distributed Files. In: Procs. of ACM SIGMOD - Int. Conf. on Management of Data. (1993)
2. Devine, R.: Design and implementation of DDH: a distributed dynamic hashing algorithm. In: Proceedings of the 4th Int. Conference on Foundations of Data Organization and Algorithms. (1993)
3. Hilford, V., Bastani, F., Cukic, B.: EH* – Extendible Hashing in a Distributed Environment. In: Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference. (1997)
4. Gribble, S., Brewer, E., Hellerstein, J., Culler, D.: Scalable, Distributed Data Structures for Internet Service Construction. In: Proceedings of the Fourth Symposium on Operating Systems Design and Implementation. (2000)
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content–Addressable Network. In: Proceedings of the ACM SIGCOMM'01. (2001)
6. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balkrishnan, H.: Chord: A Scalable P2P Lookup Service for Internet Applications. In: Proceedings of ACM SIGCOMM'01. (2001)
7. Rufino, J., Pina, A., Alves, A., Exposto, J.: Toward a dynamically balanced cluster oriented DHT. In: Proceedings of the International Conference on Parallel and Distributed Computing and Networks (PDCN'04). (2004)
8. Rufino, J., Alves, A., Pina, A., Exposto, J.: A cluster oriented model for dynamically balanced DHTs. In: Proceedings of IPDPS '04. (2004)
9. Rufino, J., Pina, A., Alves, A., Exposto, J.: Agregated routing for a cluster oriented DHT. Technical report, Dep. of Informatics and Communications, Polytechnic Institute of Bragança, Portugal (2004)
10. Bermond, J.C., Liu, Z., Syska, M.: Mean Eccentricities of de Bruijn Networks. Technical report, Université de Nice-Sophia Antipolis (1993)
11. Alves, A., Pina, A., Rufino, J., Exposto, J.: RoCL: A Resource oriented Communication Library. In: Proceedings of Euro-Par 2003. (2003)
12. D. Sacerdoti, F., Katz, M.J., Massie, M.L., Culler, D.E.: Wide Area Cluster Monitoring with Ganglia. In: Proceedings of the IEEE Cluster 2003 Conference. (2003)

[2] Scalable Information Retrieval environment (grant FCT POSI/CHS/41739/2001).