

EDAS: Providing an Environment for Decentralized Adaptive Services

Rüdiger Kapitza
Dept. of Comp. Sciences, Informatik 4
University of Erlangen-Nürnberg
rrkapitz@cs.fau.de

Franz J. Hauck
Distributed Systems Laboratory
University of Ulm
franz.hauck@uni-ulm.de

ABSTRACT

As the idea of virtualisation of compute power, storage and bandwidth becomes more and more important, grid computing evolves and is applied to a rising number of applications. The environment for decentralized adaptive services (EDAS) provides a grid-like infrastructure for user-accessed, long-term services (e.g. webserver, source-code repository etc.). It aims at supporting the autonomous execution and evolution of services in terms of scalability and resource-aware distribution. EDAS offers flexible service models based on distributed mobile objects ranging from a traditional client-server scenario to a fully peer-to-peer based approach. Automatic, dynamic resource management allows optimized use of available resources while minimizing the administrative complexity.

Categories and Subject Descriptors

C.2.4 [Computer Communication Networks]: Distributed Systems—*Distributed applications*; D.2.12b [Software]: Software Engineering Interoperability [Distributed objects]

General Terms

Design, Management

Keywords

Grid Computing, Fragmented Objects, Resource Management, Adaptability

1. INTRODUCTION

Infrastructures for grid computing aim at virtualizing a group of computers, servers, and storage as one large computing system. Resource management is a key issue in such systems, needed for an efficient and automated distribution of tasks on the grid. Such grid infrastructures are often deployed at enterprise level, but projects like SETI@home [1]

have demonstrated the feasibility of more decentralized grids as well. Current grid computing infrastructures don't provide sufficient support for the execution of distributed, user-accessed, long-term services as they are designed to solve compute- or data-intensive tasks with a more or less fixed set of parameters. The common three-phase approach of resource discovery, system selection and job execution fails for services that change their resource demand over time due to interactive user access and run for a long period of time. Instead an infrastructure for long-term services has to place services based on their current demand and their estimated future requirements. If the distribution turns out to be wrong (e.g. a node gets overloaded) the service has to be migrated within the grid (e.g. to a more powerful and less loaded node). Migration however is expensive as the whole state of a service has to be transferred. Additionally a non-replicated service is not accessible during migration. Therefore the resource management has to avoid migration if possible. Furthermore a service concept has to be provided that evades overload in the first place, and secondly inhibits service unavailability if migration can't be avoided.

EDAS [2] aims at providing a grid-like infrastructure for user-accessed, long-term services that allows the dynamic adaptation at run-time, provides a management infrastructure, and offers system-level support for scalability and fault tolerance. Nodes can dynamically join and leave the infrastructure, and all management tasks, especially the resource management, are decentralized. The environment is built upon our AspectIX [3] middleware infrastructure, which directly supports QoS-based, dynamic reconfiguration of services.

The resource management focuses on the execution of services that have a long, potentially infinite, operating time. These services are organized in projects. Each project has a distributed execution scope called a *service environment*. Such an environment possibly spans multiple institutions. Each institution represents an administrative domain that can support a project with a fixed set of resources. Our approach supports the adaptive resource management of all projects in scope of an institution based on an algorithm inspired by the diffusive algorithms for decentralized load-balancing [4]. It is not known how to optimally subdivide these resources for the services as the resource demand of services can change over time or even frequently fluctuate. To provide resources as needed, our approach automatically rededicates evenly free or not needed resources between service instances across projects and nodes. The whole process

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2nd International Doctoral Symposium on Middleware '05, November 28–December 2, 2005, Grenoble, France
Copyright 2005 ACM 1-59593-267-4/05/11 ...\$5.00.

of rededication is scalable as it is decentralized and respects certain limits like the physically available resources of a node and the amount of resources dedicated to a project. In cases where rededication is not possible, the migration of the demanding service is initiated.

EDAS further supports flexible service models, including a fully centralized client/server structure, completely peer-to-peer based systems, and various configurations in between that allow a controlled use of peer resources based on the fragmented object model [5]. The overall goal is to provide a generic service architecture that allows to implement the service functionality once, and then, ideally, run this service with different service models and adapt it at run-time, thereby scaling from a single user local instance to a multi-domain-spanning scalable service.

To reduce the implementation effort of such services a framework has been developed that supports the run-time evolution from a traditional client/server scenario to an active replicated server with clients interacting in a hybrid peer-to-peer architecture as known from Napster. In a long-term-service grid infrastructure, active replication has various benefits: Replicas can join and leave the object group and therefore replicas can be migrated without service unavailability. Load of non-modifying requests can be evenly distributed across the replicas making overload situations less likely. Finally a certain amount of node crashes can be tolerated.

The following section describes the used features of AspectIX followed by a brief overview of the core components and concepts of EDAS. Section 4 explains the self-managing and rededication concepts of distributed adaptive resource management. Section 5 describes the framework for decentralized adaptive services. Section 6 describes related work and finally Section 7 concludes the paper.

2. BASIC MIDDLEWARE

The EDAS environment is based on the AspectIX middleware. At its core, it provides a CORBA-compliant ORB and, as such, supports heterogeneous distributed systems. Furthermore AspectIX supports the fragmented object model [5] that is used to implement and provide decentralized adaptive services.

In the fragmented object model, the distinction between client stubs and the server object is no longer present (Fig. 1). From an abstract point of view, a fragmented object is a unit with unique identity, interface, behavior, and state, like in classic object-oriented design. The implementation of these properties however is not bound to a specific location, but may be distributed arbitrarily on various fragments. Any client that wants to access the fragmented object needs a local fragment, which provides an interface identical to that of a traditional stub. However internal distribution and interaction is not only transparent on the outer interface of the distributed object, but may even change dynamically at runtime. This allows the fragmented object model to adapt to changing environment conditions or quality of service requirements. It offers to change the service model on demand from traditional client-server to a peer-to-peer based approach and all kind of intermediate stages by migration and exchanging of fragments.

3. EDAS CORE COMPONENTS

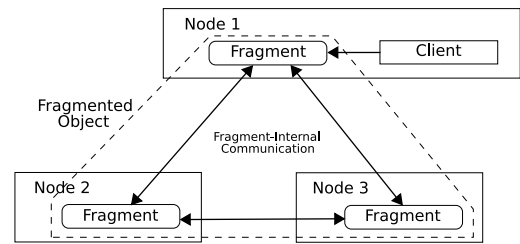


Figure 1: Fragmented object on three nodes

EDAS has three major components: Every node that actively supports decentralized adaptive services provides a *home environment*. The home environment basically manages resources of one or more nodes belonging to the same administrative domain or institution. The *service environment* is spread over a set of domains that support a certain project and relies on basic services from the corresponding home environments. The service environment supports the execution of services belonging to the same project. Finally, the decentralized adaptive service is dynamically distributed within the scope of an associated service environment.

The home environment has the role of a mediator between the nodes of an institution and one or more service environments, each running a set of services. Fig. 2 shows three domains each running a home environment that spans all nodes of the respective domains. Every node provides a set of resources. Each domain has a manager who can use that home environment to assign resources to service environments and to revoke them. Apart from providing system load and all kinds of resource-usage information to the service environment, the home environment also notifies about all important system events like a node shutdown or crash.

A service environment represents a scope of distribution for one or more services. Usually, a service environment is owned by one organization or community and dedicated to one project. A service manager can start, stop, and configure services through the interface of the service environment and decides which resources provided by home environments are accepted.

In most cases a service environment is spread over more than one administrative domain as shown in Fig. 2. One of the main tasks of the service environment is to support the migration of services or service components especially between different home environments. The service environment thereby takes available resources, the requirements of the services, and the policies provided by the service manager into account. The migration of service components can be necessary for various reasons, like node shutdown, resource constraints, and the growth or shrinkage of a service environment.

4. DECENTRALIZED RESOURCE MANAGEMENT

Resource management for long-term services has other requirements than resource management in common grid computing environments. For instance even in the context of traditional grid systems it is very difficult to determine or even only estimate the resource requirements of a task [6]. For long-term services this is even harder, and it is likely that the resource demand frequently changes. This turns

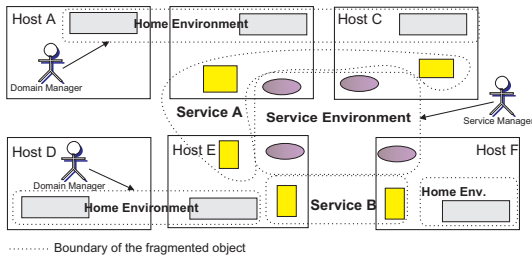


Figure 2: EDAS Scenario

the optimal distribution of services over a set of nodes into a difficult problem. In general the migration of services is a prerequisite of EDAS as it offers a solution if the initial distribution decision was wrong (e.g. initial start of previously unknown service) or the demand of services has changed substantially. But migration is costly, since the whole state of a service possibly including program code has to be transferred. If the service is not replicated it will be unavailable during migration. Taking this into account a resource management should place services and adaptively rededicate resources between services and nodes as needed to prevent migration. As EDAS aims at providing a grid-like infrastructure for a large set of nodes that can join and leave the system all resource management tasks have to be decentralized to be scalable and should not require global knowledge. The resource management can be structured into the following different tasks:

- Adding and changing the assigned resources of a service environment
- Automatic placement of service at startup time and during migration due to overload
- Keeping all kind of resource limits, especially the limits of service environments
- Compensate leaving and crashed nodes

In the next sections we will describe what kind of resource limits there are and how to do resource management based on these basic conditions.

4.1 Resource Limits

Our approach manages resources on two stages, the node level and the service-environment level. At the node level we monitor usage of all kind of physical resources like disk space, memory, CPU and network bandwidth but also logical ones like open files, sockets and threads. The entity of monitoring is a service or a service component in case of a decentralized adaptive service. Every service has so called *local limits* that restrict the resource usage in context of the current node. If a service runs the risk of exceeding such a local limit the home environment tries to extend the limits or notifies the responsible service environment if rededication is not possible. Reaching a local limit can be caused by two reasons: an overloaded node or an overloaded service environment. In the first case service migration might help, in the second case migration to another administrative domain might be an option, or simply reducing resource demand by stopping the service. Of course there could be more than one service of the same service environment at a node.

Therefore the assigned resources of a service environment at a node can be easily computed by summing up all local limits of its services.

Furthermore each node has *node limits* that restrict the overall usage of certain resources. A node limit must never exceed the physical resource (e.g. disk space) of a node and might be much smaller, e.g., for supporting local users. The sum of all local limits at a node must never exceed the node limit of a resource. Therefore observing and controlling the local limits will keep the node limits and preventing overload.

$$NodeLimit_{Node} \geq \sum_{i=1}^{numberOfLocalLimits_{Node}} LocalLimit_i$$

At the institution level the resource usage of a service environment and its associated services is also restricted by so-called *global limits*. These limits determine the maximum resource usage of a project in scope of a home environment. The sum of all local limits on all nodes of the institution for a certain project therefore never exceeds its global limit.

$$GlobleLimit_{SE} \geq \sum_{i=1}^{numberOfLocalLimits_{SE}} LocalLimit_i$$

4.2 Adaptive Resource Rededication

We start with a straight-forward implementation to describe the principal workflow. Then we propose an approach for a more efficient solution and discuss its problems.

If a new project should be supported by a home environment it is first necessary to identify the nodes that offer sufficient unassigned resources to start a service. This can be achieved in a naive implementation by using a flooding approach like it is done by the Gnutella protocol assuming the nodes are connected in a random graph. These resources then can be assigned to the service environment of the new project which further on can start services on these nodes. Of course a home environment supports usually numerous projects. Each of these projects has resource shares on various nodes, some of them occupied by services, other free and unused.

As the resource demand of a service changes it might be possible that a service reaches its local limit if the service is under high demand. What happens next depends on the overall resource usage of the service environment and the resource consumption at the local node. If the service environment has not reached its global limit and the node is not overloaded the dependent local limit of the service should be extended simply by reducing a local limit at another node of the same service environment. When all resources of the node are assigned to other service environments there are two possibilities. All resources are used by services, so we have to migrate a service, or the resources are assigned but not used. In the later case we shall rededicate resources and assign them to the demanding service environment. Finally the service environment might have reached its global limit. In this case the resource consumption has to be reduced either by migrating the service to another domain and its depended home environment or simply by bounding resource usage and if this is not possible, stopping the service.

In contrary to the setup of a new service environment which is not time critical and a less frequent task the adaptation of local limits occurs frequently and needs to be done

almost immediately. Thus it is not an option to use broadcast searches for rededication. Instead a more efficient approach with a bounded complexity is necessary. The same applies for detecting if a global limit is reached by a service environment.

Currently we investigate if this can be achieved by using a diffusive algorithm[4] like it is used for decentralized load balancing. Thereby all nodes of a system are partitioned in groups that overlap partially. The union of all groups achieves a full coverage. Group members frequently exchange load information and balance the load by migration.

In our case we aim not at balancing the load but the amount of available free resources of a service environment. Each node that supports a certain service environment is at least connected to another node that supports the same project. This way it always should be known if a service environment has still enough resources and therefore if a service can grow. There still remain open issues like if the diffusively balanced free resources should be tightly connected to the real resources, comparable to reservations. In this case there might be problems if a node supports several service environments which all have services running at the node and a node limit is exceeded which would require service migration. In fact it can be needless as the services might not use all the assigned resources but the diffusive algorithm caused the limit overrun by equally balancing the free resources of all supported service environments. If we remove the mapping between free resources and real resources of a node we can evade these situations. However it gets more complicated to determine the free and unassigned resources of a home environment.

4.3 Placement of Services

Independent of the mapping of free resources the placement of a service is, as already stated, a difficult problem. Distributing the services equally over all nodes would surely prevent migration in the average case even if resource demand of services changes. However if the resource demand of services varies highly and the grid is clogged by many projects it might be that a service can't be placed because the free resources are too scattered.

A different approach would be to consider it as an variant of the bin-packing problem that aims at packing items in bins by optimizing the number of used bins. In our case we need an online approach as the items are not known in advance and we have a multi-dimensional problem since a service has various resource requirements. The number of bins is bounded as we have a finite number of nodes in our grid. An algorithm for this problem has recently been proposed by Epstein and van Stee in [7].

On the downside this algorithm needs to know all nodes and their actual usage. As the placement of a service is not a time critical problem again a flooding based approach might offer a solution. To reduce the number of answers only nodes that provide sufficient resources need to reply. It has also to be considered to transform the algorithm to a distributed one. Another problem might be that the algorithm optimizes the occupancy too strong. Therefore demand changes of service can lead to overloaded nodes and causing migration. We believe this can be prevented by not only considering the actual resource consumption to determine the resource demand of a service but taking the previous demand

into account.

5. DECENTRALIZED ADAPTIVE SERVICE MODEL

In EDAS a decentralized, adaptive service normally matches a traditional service accessed by users like a web server, an instant messaging server or a source code repository. Such a service is represented by a fragmented object. This object expands or shrinks in the scope spanned by the associated service environment depending on the service demands and for fault-tolerance reasons. Usually every part of the object is mobile and can be migrated if necessary. Each service has at least two interfaces: one for management tasks and another service specific for the end user. The management interface offers methods to start, stop, and configure service instances.

As this set of features requires an enormous implementation effort to do it anew for each service implementation we support the development of decentralized adaptive services through a framework and an extended version of IDL in combination with a special IDL-compiler [8]. The core idea is to develop a service in usual client/server fashion as it is done in plain CORBA. This service then can be started and executed on the grid as a common servant. Additionally it should be possible to migrate the service. This can be achieved by using value type based approach to describe the service state as done in [9] or using the language supplied serialization mechanisms.

As we would like to tolerate node crashes and the service should be available during migration we support the active replication of the service. This is achieved by generating special client-side stubs that communicate with one of the replicas. To keep the connection between clients and the replicated object we use time-bounded references [10] that restrict the migration but make the usage of location services (to cope with outdated references) obsolete. The replicas are synchronized via a group communication framework.

The IDL extension consists of additional modifiers that affect code generation for client and server side. These are `retain` to mark non-modifying operations which allows faster responses and load balancing of those requests. Furthermore one can mark methods as `local` which indicates that they can be locally processed. In this case the IDL-compiler creates placeholder for local execution. Apart from methods that are usual static this is useful to implement client-side contribution and interaction. For example if a client-stub offers a method which results in a file transfer it is possible to integrate a custom protocol that forwards a modified request to the replicated object which returns not the file as in the common case but URLs that point to clients that previously requested the file. Now the client-stub fetches the data from the offered location and responds as if it was supplied by the server object. This peer-to-peer based behavior as known from Napster is transparent to the client and can be switched on and off depending on environment conditions like load and community as needed. Finally we provide another modifier to mark administrative operations. If a method is marked with `admin` an authentication is necessary. The method to authenticate is pluggable and might be by pass-phrase, internet address or any other authentication scheme. This modifier facilitates the creation of service management methods.

6. RELATED WORK

Grid infrastructures like the Globus-Toolkit [11] provide services and mechanisms for distributed heterogeneous environments to combine resources on demand to solve resource consuming and compute intensive tasks. Due to this orientation they focus on different service models, provide no support for object mobility if even supporting a distributed object approach at all. But most important they follow a different resource management approach as they target the parallel execution of a large number of short and midterm tasks.

JavaSymphony [12] and Ibis [13] provide object mobility but are limited to the Java programming language and focus on object oriented high performance computing.

Actively replicated objects are provided by Jgroup [14] based on RMI. On top of this basic middleware a replication management layer has been implemented called ARM [15]. JGroup focus on the active replication of objects but lacks support for more flexible services like EDAS does. ARM can be compared to EDAS but supports no resource aware distribution.

Fog [16] and Globe [17] are basic middleware environments that support the fragmented object approach. Globe considers replication and caching. Both systems lack support for resource aware distribution.

7. CONCLUSION AND ONGOING WORK

Based on the fragmented object model and the architecture of the EDAS environment, decentralized adaptive services can be easily designed, implemented and executed.

As described the resource management can be decomposed in two main problems that have to be solved. Controlling and managing of resource limits including ensuring that the assigned resources are available (even in the context of node crashes) and the autonomous placement of services. For both problems we offer a solution, a currently implemented simulation environment will verify their feasibility. In a next step the resource management will be integrate in an already implemented prototype of the EDAS architecture.

As described we have already an early implementation of the framework for the decentralized adaptive services. This framework has to be extended to smoothly interact with the resource management and the EDAS architecture. In a final step we need to implement some services that verify the usability of the whole EDAS project.

8. REFERENCES

- [1] D. Werthimer S. Bowyer J. Cobb D. Gedye D. Anderson W. T. Sullivan, III. A new major seti project based on project serendip data and 100,000 personal computers. In *Proc. of the Fifth Intl. Conf. on Bioastronomy*, 1997.
- [2] Hans Reiser Rüdiger Kapitza, Franz J. Hauck. Decentralized, Adaptive Services: The AspectIX Approach for a Flexible and Secure Grid Environment. In *Grid Services Engineering and Management (GSEM 2004)*, Erfurt, Germany, 2004. Springer.
- [3] Hans P. Reiser, Franz J. Hauck, Rüdiger Kapitza, and Andreas I. Schmied. Integrating fragmented objects into a CORBA environment. In *Proc. of the Net.ObjectDays*, 2003.
- [4] Tiberiu Rotaru and Hans-Heinrich Nägeli. Dynamic load balancing by diffusion in heterogeneous systems. *J. Parallel Distrib. Comput.*, 64(4):481–497, 2004.
- [5] M. Makpangou, Y. Gourhant, J.-P. Narzul, and M. Shapiro. *Fragmented objects for distributed abstractions*.
- [6] Jennifer M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. pages 15–23, 2004.
- [7] Leah Epstein and Rob van Stee. Optimal online bounded space multidimensional packing. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 214–223, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.
- [8] Hans P. Reiser, Martin Steckermeier, and Franz J. Hauck. IDLflex: a flexible and generic compiler for CORBA IDL. In *Proc. of the Net.ObjectDays (Erfurt, Germany, Sep. 10-13, 2001)*, 2001.
- [9] Rüdiger Kapitza, Holger Schmidt, and Franz J. Hauck. Platform-independent object migration in CORBA. In *Proc. of the OTM'05 Conferences (DOA, Agia Napa, Cyprus, Oct 31-Nov. 04, 2005)*, 2005.
- [10] Rüdiger Kapitza, Hans P. Reiser, and Franz J. Hauck. Stable, time-bound references in context of dynamically changing environments. In *MDC'05: Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems - Workshops (ICDCS 2005 Workshops)*, 2005.
- [11] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.
- [12] Thomas Fahringer and Alexandru Jugravu. Javasymphony: new directives to control and synchronize locality, parallelism, and load balancing for cluster and grid-computing. In *JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 8–17, New York, NY, USA, 2002. ACM Press.
- [13] Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E. Bal. Ibis: an efficient java-based grid programming environment. In *JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 18–27, New York, NY, USA, 2002. ACM Press.
- [14] Alberto Montresor, Renzo Davoli, and Ozalp Babaoglu. Middleware for dependable network services in partitionable distributed systems. *SIGOPS Oper. Syst. Rev.*, 35(1):73–96, 2001.
- [15] H. Meling and B. Helvik. Arm: Autonomous replication management in jgroup, 2001.
- [16] Mesaac Makpangou, Yvon Gourhant, Jean-Pierre Le Narzul, and Marc Shapiro. Fragmented objects for distributed abstractions. In *Readings in Distributed Computing Systems*.
- [17] Philip Homburg, Leendert van Doorn, Maarten van Steen, Andrew S. Tanenbaum, and Wiebren de Jonge. An object model for flexible distributed systems. In *Proceedings of the 1st Annual ASCI Conference*, pages 69–78, 1995.