Towards Multi-Level Adaptation for Distributed Operating Systems and Applications

Djawida Dib, Nikos Parlavantzas, Christine Morin

Towards Multi-Level Adaptation for Distributed Operating Systems and Applications

Djawida Dib[1,⋆], Nikos Parlavantzas[1,2], and Christine Morin[1]

[1] INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
[2] INSA de Rennes, Campus de Beaulieu, 35708 Rennes cedex, France
{Djawida.Dib,Nikos.Parlavantzas,Christine.Morin}@inria.fr

Abstract. Distributed operating systems simplify building and executing applications on large-scale infrastructures, such as clusters, grids and clouds. These systems operate in a constantly changing environment characterized by varying application needs and varying physical infrastructure capabilities. To handle the diversity and dynamism of both the applications and the underlying infrastructures, the distributed Operating System (OS) should continually adapt to its changing environment. Two challenges arise in this context: how to design the distributed OS in order to facilitate dynamic adaptation, and how to ensure that OS-level adaptation does not conflict with application-level adaptation. This paper proposes to address these challenges by: (1) building the distributed OS as an assembly of adaptable services following the service-oriented architecture; and (2) using a common multi-level adaptation framework to adapt both the OS and the application layers in a coordinated way. Moreover, the paper presents experimental evidence of the usefulness of this approach in adapting the distributed shared memory service of a specific distributed OS.

Keywords: distributed operating system, service-oriented architecture, multi-level adaptation, distributed shared memory.

1 Introduction

Distributed Operating Systems (OSs) provide common services and abstractions for building applications on large-scale infrastructures such as clusters [14][3], grids [13][8] and clouds [1][2]. These systems operate in a constantly changing environment characterized by varying application needs and physical infrastructure capabilities. To deal with this dynamicity, it is essential for the distributed OS to support dynamic adaptation. For example, it should support dynamically modifying communication protocols depending on application usage patterns or network conditions.

Service-Oriented Architectures (SOAs) have recently emerged as a popular approach for building flexible software systems [15]. A service-oriented system is an assembly of services, where each service represents a well-defined function.

⋆ The research leading to these results is co-funded by the Brittany Region.

The flexibility of SOA results from the dynamic capability to publish, discover and use services as well as modify their implementations without impacting their consumers. While SOA has been widely applied to build distributed applications [20], it is less explored in building the supporting system software. In this paper, we propose to apply the SOA at the OS level in order to facilitate the OS dynamic adaptation.

Supporting dynamic adaptation of the distributed OS is introducing further challenges, such as detecting when adaptation is needed and deciding what and how to adapt in order to achieve specific objectives. One important challenge, which is not addressed by related research work, is ensuring that OS-level adaptation does not conflict with application-level adaptation. For example, a particular performance problem could be addressed by replacing a service at the application layer or by allocating more resources at the OS layer; performing both adaptations could be inefficient or damaging. To address this challenge, we propose to use a multi-level adaptation framework that coordinates the adaptation of both the application and the OS layers based on configurable policies. Experimental results are provided to demonstrate the usefulness of this approach.

This paper is organized as follows. Section 2 introduces the service-oriented OS. Section 3 discusses the need for a multi-level adaptation framework. Section 4 presents the multi-level adaptation architecture. Section 5 shows the usefulness of the proposed approach with the distributed shared memory example. Section 6 presents some related work. Finally, Section 7 concludes this paper and discusses future work.

2 Distributed Service-Oriented Operating Systems

Current distributed operating systems provide for distributed infrastructure users what a traditional OS provides for single computer users [3][14][13]. Specifically, a distributed OS extends in a transparent way traditional OS functionalities. For example, the memory management functionality in a traditional OS is extended to form the Distributed Shared Memory (DSM) functionality in a distributed OS. Previous research works have proposed, compared and classified several algorithms to implement each OS functionality (e.g., DSM [18] and resource management [11]). Depending on the system environment, usually only one algorithm provides optimal performance. For example, a resource management algorithm can be very efficient to manage a small number of resources but it could be less efficient than other algorithms to manage a lot of resources.

Furthermore, distributed OSs operate in a constantly-changing environment, characterized by two levels of dynamicity: the dynamicity of the underlying infrastructure (resources may connect and disconnect at any time leading to variable resource availability), and the dynamicity of the application requirements (such as resource requirements). To accommodate this dynamicity while providing the best quality of service to applications, we believe that the distributed OS should be dynamically adaptable. Specifically, we believe that the OS should support selecting the best algorithm for each functionality at runtime. The se-

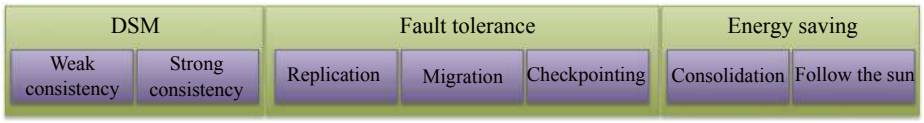

Fig. 1. Service-oriented operating system overview

lection could be done either according to the state of the whole system or in a more customized way, for each kind of application. In the latter case, two or more algorithms for the same functionality should be able to operate simultaneously, each one supporting the applications for which it was selected. For example, the fault tolerance functionality may use checkpointing for some applications while using replication for others.

To build such a dynamically-adaptable OS we propose to use SOA, thus enabling its well-known benefits in terms of interoperability and dynamic recomposition to be extended to the OS layer. Then, the OS is built as a composition of services, each service representing a distributed OS functionality. Figure 1 illustrates this idea with examples of some existing algorithms for three distributed OS services. The DSM service may use the weak or the strong consistency models, the fault tolerance service may use replication, migration or checkpointing, and the energy saving service may use a consolidation protocol [6] to minimize the number of used resources or use the "follow the sun" protocol [21] to utilize mostly sun energy.

The main challenge in building such a dynamically-adaptable OS is to determine when and how to change algorithms while taking into account the overhead of the change. Indeed, selecting the appropriate algorithm for each service is a complex task and involves thoroughly evaluating the algorithms as well as the parameters influencing their behaviors. To demonstrate this point, we investigate in Sect.5 two algorithms for the DSM service as well as the parameters influencing their behaviors.

3 Multi-Level Adaptation Framework

Apart from the distributed OS, it is frequently necessary to dynamically adapt the distributed applications themselves. This may be motivated by changes in user requirements, such as demands for higher precision in calculation results; or in the available computing resources, such as node failures or new nodes joining the system. The application adaptation is required in such cases to enable applications to work correctly despite all possible environment degradations, and to take advantage of new opportunities that can occur. Indeed, modern distributed applications are increasingly built according to SOA in which applications are composed of multiple services and are prepared to handle services appearing and disappearing at any time.

Traditionally, the adaptation of each layer of a distributed system is managed separately based on layer-specific knowledge and objectives. For example, there

are research works focusing exclusively on OS layer adaptation [17][19] and other works focusing exclusively on application layer adaptation [20]. However, the adaptation of one layer clearly impacts the others. As a result, adapting each layer separately can create conflicts, unpredictable results, or redundancies. For example, adapting the OS to accommodate reduced network bandwidth may be unnecessary if the application reacts to the same situation by replacing the remote service with a locally available one. To avoid such interference between adaptation actions, we propose to use a multi-level adaptation framework (such as SAFDIS [7]), capable of interacting and dynamically adapting both layers in a coordinated way. The framework must maintain knowledge about the whole system and perform adaptation actions according to application-specific and system-wide goals.

4 Architecture

This section presents a generic architecture to apply the concept of multi-level adaptation. This architecture is composed of two service-oriented layers, application and OS layers, which are on top of a physical infrastructure layer (see Fig.2). The considered physical infrastructure layer is composed of the hardware and a traditional OS installed on each node. All layers expose rich monitoring mechanisms and are controlled by the multi-level adaptation framework. Service-oriented layers expose also actuation mechanisms enabling the framework to trigger adaptation actions. The adaptation process of the framework follows the four main phases of the MAPE model [10]: Monitoring, Analysis, Planning and Execution. A description of the phases is given below.

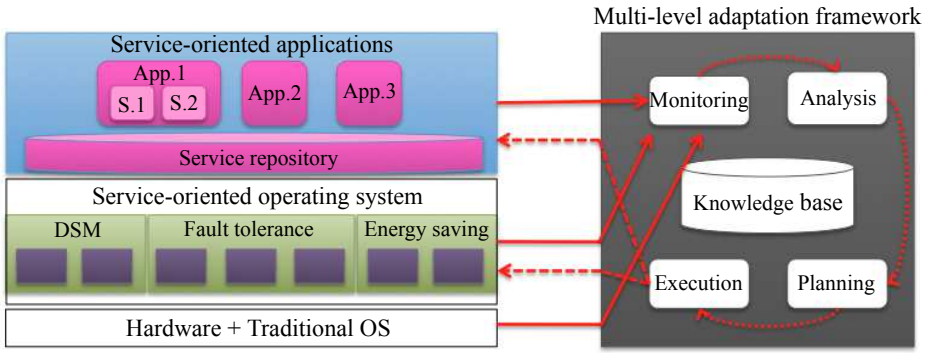

Fig. 2. Multi-level dynamic adaptation architecture – solid arrows represent the communication between sensors of each layer (sensors are not shown in the figure) and the Monitoring component of the framework; dashed arrows represent the communication between actuators of service-oriented layers (actuators are not shown in the figure) and the Execution component of the framework; dotted arrows represent the sequence of phases in an adaptation cycle.

1. **Monitoring:** the framework collaborates with the sensors of each layer in order to provide a dynamic view of the whole system. At the physical infrastructure layer, the sensors inform the framework about the availability and the workload of resources. At the OS layer, the sensors inform the framework about all available algorithms for each OS service, either statically at the start-up of the system or dynamically when a new algorithm component is added to the system. At the application layer, the sensors inform the framework about all available services in the repository as well as applications requirements. The mechanism used to monitor data may be either push or pull, depending on the nature of data and its frequency changes. The gathered information is stored in a knowledge base to be used later in the other phases.
2. **Analysis:** the framework analyzes the monitored data and uses policies to decide whether an adaptation is required or not. For instance, when the monitored data shows a deterioration of the system performance, the analysis policy determines if the OS, the application, or both should be adapted. The analysis policy triggers next phases only if the estimated overhead of the adaptation is less important than its benefit.
3. **Planning:** the framework defines a plan to apply the decided adaptation at the analysis phase. The plan consists of set of actions which are scheduled properly. For example, if both application and OS layers should be adapted then the planning strategy will specify which layer will be adapted first.
4. **Execution:** the framework collaborates with actuators of the service-oriented layers to perform the plan defined at the planning phase. At the OS layer, actuators enable either to switch between algorithms or to ensure that two (or more) algorithms coexist. At the application layer, actuators may modify application parameters or modify the application structure, such as replacing services or changing service interconnections.

5 Illustrative Examples

In this section we investigate two algorithms of the DSM functionality to illustrate the usefulness of the proposed architecture. The objective of this section is not to improve or to optimize existing DSM algorithms, but to demonstrate that each algorithm can be better than others in particular cases.

The DSM service makes the main memory of a cluster of computers look like a single memory, using replicated data consistency models. We have evaluated two algorithms of the sequential consistency model: Broadcast-on-write and Invalidate-on-write. Both algorithms guarantee that any read of a data copy returns the last value assigned to this data. This is done by invalidating all remote data copies when a process access in write mode its local copy. The main difference takes place once the write operation finishes. In the broadcast-on-write algorithm, the owner of the modified copy updates automatically all the other copies. Whereas in the invalidate-on-write algorithm, the invalidated copies will be updated on demand when a page fault exception occurs. If the OS uses the

broadcast algorithm, no fault page exception will occur. Nevertheless, some invalidated copies may be unnecessary updated if these copies are never subsequently used by their owners. Therefore, it is difficult to predict which algorithm will perform better. In this perspective we measure, in the next subsection, the execution time of a synthetic application with each algorithm while varying some parameters likely to impact the system performance.

5.1 Experiment Environment

To carry out these experiments, we have used and extended the Kerrighed OS [14]. The existing implementation of Kerrighed uses the invalidation algorithm for the DSM functionality. We have implemented the boadcast algorithm and an actuator to switch between the two algorithms at runtime. In addition to the Kerrighed modification, we have developed a synthetic distributed application that uses shared memory objects for its inter-process communications. This application consists of a configurable *number of processes* that share data objects, each process run on a different node. One process creates a configurable *number of shared data objects* with different *sizes* and all other processes can access them either in read or in write mode. The application *access pattern* follows the notion of *Write-Run* defined in [5]. A write-run is a sequence of write accesses to a shared object by a single process uninterrupted by any access of the other processes. The number of the sequential writes, called the write-run length, represents in these experiments the access pattern parameter. Attention has been paid to these parameters because of their dynamicity during the execution of a distributed shared memory application. The experiments were performed on the paradent cluster of the Grid'5000 site of Rennes that consists of Carri System CS-5393B nodes supplied with 2 Intel Xeon L5420 processors (each with 4 cores at 2.5 GHz), 32 GB of memory, and Gigabit Ethernet network interfaces. The results are discussed in the next subsection.

5.2 Results

Figure 4(a) shows the impact of the number of data object copies on the application execution time. The application parameters were set at one data object of 10 bytes size with a write-run length equal to one, only the number of data object copies was adjusted. We notice that the performance with both algorithms is equal when there are two data object copies. The invalidation algorithm performs better when there is only one copy, but the broadcast algorithm performs increasingly better when there are at least three copies.

Figure 4(b) shows the impact of the write-run length on the application execution time. The application parameters were set at one data object of 10 bytes size with 4 data object copies, only the write-run length was adjusted. The application execution times with both algorithms converge when the write-run length is set at 13. If the write-run length is smaller then the convergence point, the broadcast algorithm performs better and conversely if it is greater.

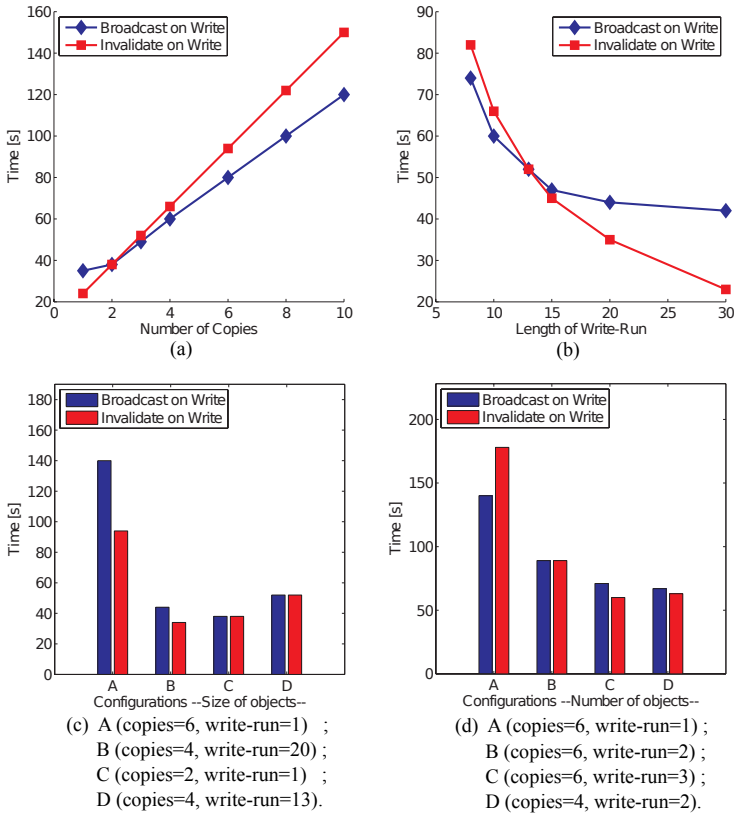

Fig. 3. Impact of (a) the number of data object copies, (b) the write-run length, (c) the data object size, and (d) the number of data objects on the execution time of the application.

Figure 4(c), compared to Fig.4(a) and Fig.4(b) shows the impact of the data object size on the application execution time. The application parameters were set at one data object of 10^3 bytes size. The number of data object copies and the write-run length were adjusted to define four configurations (A,B,C,D). Each configuration was previously tested with an object size set at 10 bytes (see Fig.4(a) and Fig.4(b)). This comparison shows that the object size does not impact the application execution time in our experimental environment.

Figure 4(d) shows the impact of the number of data objects on the application execution time. The application parameters were set at two data objects of 10 bytes size. The number of data object copies and the write-run length was adjusted to define four configurations. The first configuration (A), compared to its analogous one with one shared object in Fig.4(a), shows an increase of the application execution time with both algorithms. In configurations (B) and (C), we have increased the write-run length and there we notice that the invalidation

algorithm, respectively, catches up and overtakes the broadcast algorithm in terms of performance. In the configuration (D), we have decreased the number of data object copies and set the write-run length at 2. In this case the invalidation algorithm outperforms the broadcast algorithm. As a result, we conclude that the increase of the data objects number has an impact on the performance.

The results shown in Fig.3 demonstrate that the appropriate algorithm depends on application-layer parameters, thus showing the benefit of dynamically adapting the OS based on application-layer knowledge. Based on the data obtained from these experiments and machine learning techniques, we plan to generate a function that predicts from these parameters which algorithm will be better to use. When there is a change in parameters, the adaptation framework will use this function to decide whether to switch algorithms.

5.3 Possible Extensions

As a following step, we plan to validate our approach by providing support for adaptation involving multiple layers. We will first take into account the different network requirements of the algorithms. We then plan to extend the analysis phase to take into account not only the application-layer parameters but also the current state of the network.

Another planned extension is investigating coordinated adaptation across both OS and application layers through an application that transfers audio over the network and processes it in a parallel way using distributed shared memory. The audio compression rate of this application can be adapted based on network conditions. The multi-level framework enables this adaptation to be implemented in coordination with OS-level algorithm changes, thus avoiding unpredictable results. Indeed, performing this adaptation in an isolated way, without taking into account OS bandwidth needs, may cause instabilities. For example, if there is no coordinated adaptation and the bandwidth is low, the application increases its compression rate, freeing up available bandwidth. This causes the operating system to use a more bandwidth-demanding algorithm, which reduces the available bandwidth and the cycle continues. A coordinated adaptation, as enabled by our approach, avoids this problem

6 Related work

The dynamic adaptation of operating systems has been investigated under different angles. Seltzer and Small proposed a design for self monitoring and self adapting an extensible kernel [17]. They have defined a mechanism that uses online and offline analysis of the system behavior to adapt its policies. Teller and Seelam investigated the dynamic adaptation of OS policies in order to improve the application and the system performance. They have proposed a multi policy system that aims to satisfy different constraints by dynamically switching between policies. They have demonstrated their concept in the context of I/O scheduling [19].

The above works do not apply the service-oriented approach for structuring their OS. In this context Milanovic and Malek [12] investigated possible architectures to integrate the OS and the service-oriented computing. Among the investigated architectures, the closest one to our approach is to consider the OS as a set of collaborating services. The main difference with our model is the way of building the kernel. They propose to build it according to available kernel services while we propose to build it by selecting the most appropriate algorithm for each service according to its environment. Another interesting project in this context is SoOS, which tries to extend the limited capabilities of local devices with remote ones [16]. In this project local and remote resources are virtualized on top of the networking and Input/Output interfaces and are then made transparent to the running applications. Thus, the OS is considered as a collection of independent computational entities that appears to its users as a single coherent system. Unlike our proposal, this work focus on the physical infrastructure layer.

Recently, multi-level adaptation frameworks have also been investigated. The authors of [4] and [9] aim to provide a coherent solution to monitor and adapt service-based applications. The considered service layers are : business service, composition and coordination service, and the infrastructure service. The main difference with our proposal is the considered layers. They focus exclusively on service-oriented application layers while we also include the OS layer.

7 Conclusion

Modern distributed operating systems must operate reliably in constantly-changing environments. As a result, dynamic adaptability is an essential requirement for such systems. This paper made two contributions in this context. First, it proposed a service-oriented approach for building distributed OSs, which enables adapting the OS algorithms in a dynamic, on-demand fashion. Second, it proposed using a common framework to adapt both the OS and application layer in a coordinated way, thus avoiding any possible conflict or redundancy.

To demonstrate the usefulness of the proposed architecture, the paper investigated the DSM functionality and provided examples and experimental results using the Kerrighed distributed OS. As future work, we plan to implement further adaptation examples involving real applications on Kerrighed and extending the SAFDIS adaptation framework to control and adapt both layers.

Acknowledgment

We cordially thank the late professor Françoise André for her precious ideas and comments which helped to define the objectives of this work. We thank Dr. Louis Rilling for providing documentation support concerning Kerrighed. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Openstack, `http://www.openstack.org`
2. Opennebula, `http://opennebula.org`
3. Barak, A., Guday, S., Wheeler, R.G.: The MOSIX Distributed Operating System: Load Balancing for UNIX. Springer-Verlag New York, Inc. (1993)
4. Bratanis, K., Dranidis, D., Simons, A.J.H.: Slas for cross-layer adaptation and monitoring of service-based applications: a case study. In: Proceedings of the International Workshop on Quality Assurance for Service-Based Applications (2011)
5. Eggers, S.J., Katz, R.H.: A characterization of sharing in parallel programs and its application to coherency protocol evaluation. SIGARCH Comput. Archit. News (1988)
6. Ferreto, T.C., Netto, M.A.S., Calheiros, R.N., De Rose, C.A.F.: Server consolidation with migration control for virtualized data centers. Future Gener. Comput. Syst. (2011)
7. Gauvrit, G., Daubert, E., André, F.: SAFDIS: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications. In: Proceedings of the 36th EUROMICRO Conference (2010)
8. Grimshaw, A.S., Wulf, W.A., The Legion Team, C.: The legion vision of a worldwide virtual computer. Commun. ACM (1997)
9. Kazhamiakin, R., Pistore, M., Zengin, A.: Cross-layer adaptation and monitoring of service-based applications. In: Proceedings of the 2009 international conference on Service-oriented computing (2009)
10. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer (2003)
11. Klaus Krauter, Rajkumar Buyya, M.M.: A taxonomy and survey of grid resource management systems for distributed computing. Softw. Pract. Exper. (2002)
12. Milanovic, N., Malek, M.: Service-oriented operating system: A key element in improving service availability. In: Proceedings of the 4th international symposium on Service Availability. Springer-Verlag (2007)
13. Morin, C.: Xtreemos: A grid operating system making your computer ready for participating in virtual organizations. Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on (2007)
14. Morin, C., Gallard, P., Lottiaux, R., Vallée, G.: Towards an efficient single system image cluster operating system. Future Gener. Comput. Syst. (2004)
15. Papazoglou, M.P., Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. The VLDB Journal (2007)
16. Schubert, L., Kipp, A., Koller, B., Wesner, S.: Service oriented operating systems: Future workspaces. Ieee Wireless Communications (2009)
17. Seltzer, M., Small, C.: Self-monitoring and self-adapting operating systems. In: In Proceedings of the Sixth workshop on Hot Topics in Operating Systems (1997)
18. Tam, M.C., Smith, J.M., Farber, D.J.: A taxonomy-based comparison of several distributed shared memory systems. SIGOPS Oper. Syst. Rev. (1990)
19. Teller, P.J., Seelam, S.R.: Insights into providing dynamic adaptation of operating system policies. SIGOPS Oper. Syst. Rev. (2006)
20. Tosi, D., Denaro, G., Pezze, M.: Towards autonomic service-oriented applications. Int. J. Autonomic Comput. (2009)
21. Treinen, J.J., Miller-Frost, S.L.: Following the sun: case studies in global software development. IBM Syst. J. (2006)