Towards the Future Internet
G. Tselentis et al. (Eds.)
IOS Press, 2009

doi:10.3233/978-1-60750-007-0-238

Above the Clouds: From Grids to Service-oriented Operating Systems

Lutz SCHUBERT[a,1], Alexander KIPP[a] and Stefan WESNER[a]
[a]*HLRS – University of Stuttgart, Germany*

Abstract. Over recent years, resource provisioning over the Internet has moved from Grid to Cloud computing. Whilst the capabilities and the ease of use have increased, uptake is still comparatively slow, in particular in the commercial context. This paper discusses a novel resource provisioning concept called Service-oriented Operating Systems and how it differs from existing approaches of Grids and Clouds. The proposed approach aims for making applications and computers more independent of the underlying hardware and increase mobility and performance. The base architecture and functionality will be detailed in this paper, as well as how such operating systems could be deployed in future workspaces.

Keywords. operating systems, network, grid, service oriented architecture, many core, mobile grids, distributed computing

Introduction

The Grid concept is understood in this paper as a set of distributed resources integrated in a fashion allowing remote execution of processes and applications with different requirements towards the underlying resources, such as computational power. Next to execution of tasks, it is also often considered a space for storing vast data amounts. These types are often referred to as *computational* or *storage* Grids. Depending on whether execution / storage is supposed to be stable non-regarding changes in the Grid structure (i.e. failure of individual resource nodes), we speak of *managed* grids.

Though these do not comprise the full usage scope of Grids, it nonetheless reflects the base commonality of most use cases, namely the capability to provide (managed) resources in a fashion that they can be used remotely with little to no impact on the local execution of processes, respectively extending the capabilities of local resources.

Cloud computing has recently emerged from this movement as a means to provide in particular computational resources, even though storage clouds are gaining in popularity too. From the Grid perspective, Clouds are only passive resources in the sense of nodes in the computational or data grid case – they do not offer the enhanced capabilities of Virtual Organisations or similar, nor do they easily plug in into existing grid structures (such as GT4). Typically cloud and similar resource providers expose their own proprietary APIs which imply that the resources are used in a more manual fashion than originally envisaged by the Grid. From this point of view, clouds are only extended single Grid nodes. However they can be seen as an intermediary step to enabling dynamic outsourcing in a form that extends local resources.

[1] Corresponding Author: Lutz Schubert, HLRS – High Performance Computing Centre, University of Stuttgart. Nobelstr. 19, 70569 Stuttgart, eMail: schubert@hlrs.de

This paper will examine the current problems in resource provisioning and uptake in commerce, as well as proposing a means to overcome these problems (section 1 & 2). We will present the notion of Service Oriented Operating Systems (SOS) that integrate dynamic resource pools and execute processes across these (section 3 & 4). We will then compare this model with current related approaches (section 5) and conclude by examining the relevance of resource fabrics for the future internet.

1. From Grids to Clouds and their limitations

Early Grid solutions had been introduced to overcome the limitations of computing systems and integrate geographically dispersed resources into a metacomputer [19]. This concept had been further extended to allow for integration of more heterogeneous resources. Prominent realisations of this are the EGEE [20] and DEISA [21] infrastructures. While they are well perceived and used within the scientific community their uptake for business or general purpose applications remains quite limited.

The move from proprietary solutions towards Service Oriented Grids [22] [23] has increased uptake within the community as critical aspects such as commercial level security, exploitation of Web Service standards and frameworks decreased the entry level border for programmers and companies significantly and is further developed utilizing semantic technology towards Service Oriented Knowledge Utilities [24]. Grids realising cross-organisational collaborations remain still comparatively complex and competing standards for key aspects make investment in this technology risky.

Cloud computing addresses the major limitations of Grid solutions and convinces by its (seeming) simplicity. They pick up the concept of Virtual Hosting Environments [23] [2] and realise them in an efficient and easy to use fashion. In order to be commercially viable, outsourcing approaches like Clouds have to take extreme security and legal precautions, as typically the data transported to and computed on external resources is sensitive to the respective customer. The complexity of outsourced computational resources was one of the reasons to make the approach of grid- / web service based Virtual Organisations [14] so attractive for enterprises.

Existing Cloud solutions only partially address such business requirements so far and clearly attract users without specific requirements e.g. related to the confidentiality of their data. The realization of Grids over Clouds (cf. Figure 3) would enable such combined features but require additional mechanisms to bridge the gap between the Grid and the Cloud environment.

Current approaches e.g. by Amazon [5] or SalesForce' [4] suffer in particular from their proprietary APIs and their clear disjunction from local machines: whilst network storage can already be easily integrated into the user's operating system given that it is static, resources, services and applications on a network level cannot. Even though the web service approach would allow for this integration due to its standard interface approach, usage is not easy and typically comes at the cost of speed.

Web Services and Grid put the user in control of the actions and integrations – something that very few users are willing to put up with, let alone have the knowledge and capabilities to do so. For low level resources this is furthermore not even required.

Service-oriented Operating Systems treat resources as what they are: resources, i.e. they are treated by the OS in the same way as local devices. As we will show in the discussion section, even web based applications effectively similar to computational resources with very limited capabilities (namely execution of the respective process).

2. Resource Fabrics: Building up the Service-oriented Operating System

For a SOS, the network builds a mesh of potential resources, similar to P2P where the nodes build up the network. The main OS instance will identify remote resources and their capabilities – ranging from speed, availability etc. to type and functionality.

Such a resource mesh consists of all machines available in the network that run the base infrastructure (see below) and integrates them *virtually* into a local environment. The Service-oriented Operating System therefore treats the web environment as a *resource fabric* where individual providers are subject to constant changes.

A "resource fabric" consists of resources exposed over the internet and made available on a low Operating System level. Grid and Cloud provide and integrate their resources on higher levels, close to the actual application layer. Note that even though clouds can host virtual machines (and thus OS) they are still subject to the underlying infrastructure and require a full operating system from the user.

Implicitly, classical approaches suffer from all according deficits and usage & control are completely up to the user. Extending capabilities implies reprogramming and deployment is restricted by the middleware (cf. Figure 3, left). Infrastructure and operating system even often clash, so that additional steps are required to compensate for the "deficiencies", ranging from security to slow message processing. In other words, efficient usage of resources on the upper layers requires in-depth expertise.

2.1. Virtualizing the Environment

The Service-oriented approach aims to overcome these obstacles by integrating resources right on the OS level – the fabric thus becomes generally transparent. In other words, which resources, applications and devices can be used does not only depend on the local deployment, but also on the machines available in the fabric.

Service-oriented OS require an intermediary layer that virtualises the hardware underneath it. This is already a common approach in most Operating Systems, such as the Hardware Abstraction Layer (HAL) [7], in order to reduce resource management overhead. Classical OS however treat networking on a higher layer with only the network card represented on the hardware layer, so as to account for different manufacturers. The SOS extends this concept by introducing cross network virtualisation which integrates resources across the web. The operating system therefore makes use of double blind invocation techniques (cf. [1], [2]), which renders resources and their location more independent from the calling infrastructure (Figure 1). This concept was already introduced by the IST project TrustCoM [3].

Double blind invocation acts on both sender and recipient side to enhance control, i.e. applications invoke hardware non-regarding their interface and only with respect to their functionality (printer, storage, computation etc.) – this goes beyond the scope of mere low-level devices and can circumscribe applications too. The Virtual Environment layer will identify appropriate resources and providers on basis of the required capabilities – note that the OS is *not supposed* to take higher-level aspects, such as user profiles, into consideration at this layer, as these are application-specific. During usage, interaction with the resource will either take place locally without intermediary messaging or across the network. In both cases, the recipient will not directly expose the resource, but through a reduced OS instance that grants secure access control and routing capabilities, i.e. it is completely up to the provider, which resources and services to expose or where to host them. Thus, the recipient's infrastructure can span a

local network. This is principally identical to cloud providers such as SalesForce which expose a common interface to a dynamic and fully managed cloud farm, thus reducing the user's overhead to select and maintain appropriate resources.

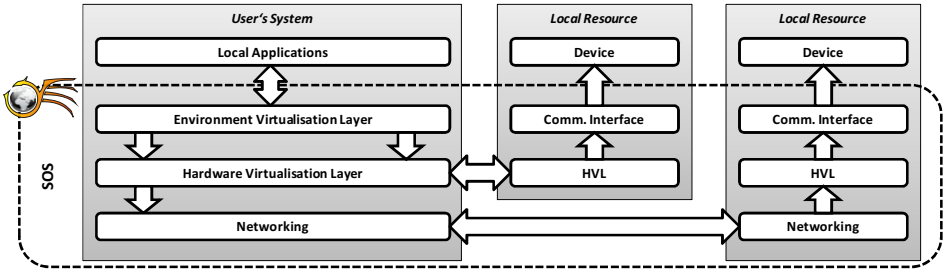

Figure 1: virtualization using double blind invocation in SOS

The communication interfaces also allow introduction of standard messaging structures, thus granting standard manufacture-independent access. With the remote instance hosting a reduced OS, the main instance can exploit the capabilities whilst the remote instance can take care of actual hardware interaction. As device drivers are the main source for Operating System crashes, this approach improves stability significantly, and reduces the risk of incompatibilities between different drivers accordingly.

2.2. Sub Instances

We can distinguish different types of resources and hence sub-OS instances, or micro Kernels, that can be integrated (cf. Figure 2):

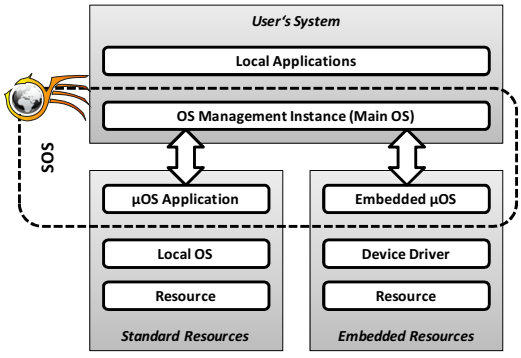

Figure 2: main kernel instances

(1) in classical, standalone mode, the OS will make use of local resources which do not require their own kernel, though this is possible with new hardware architectures and would grant additional flexibility at run-time – this however exceeds the scope of this paper and will be addressed in a future publication.

(2) dedicated resources which are only intended for usage in the SOS context which require embedded micro kernels that take over messaging and management, as well as incorporation of the driver. All access to these resources are routed via the same instance, leading to straight forward request queuing. More intelligent queues and self-management capabilities are specific to the device and may extend the kernel.

(3) resources used in multiple contexts and in particular used locally by the owner for his / her own purposes, need to provide the extended functionality *on top of* existing

operating systems, i.e. in the form of a standalone kernel wrapping as an application. The standalone kernel thereby allows designation of specific resources and applications for provisioning without requiring additional technical know-how: as the SOS operates on the code basis and not on the application level, dedicated resources of any type will become available to the main instance under the restrictions specified by the user. This allows hosting of applications equally to computational and storage resources.

2.3. Integrating Clouds and Grids

It is obvious that the virtualisation layer acts as a reduced Grid infrastructure incorporating classical resource management strategies. Notably, SOS does *not* provide the same support as current Grid related approaches, as these are not relevant for base resource infrastructures but add extended functionalities on top of it. Bearing the end-user in mind, a usable system should not exceed the capabilities on the cost of usability, but should realise all requirements in a stable and easy to use fashion. The system is thereby designed open enough to allow for flexible extensions on higher levels, i.e. integrating research results in the area of QoS, semantic annotation and reasoning etc.

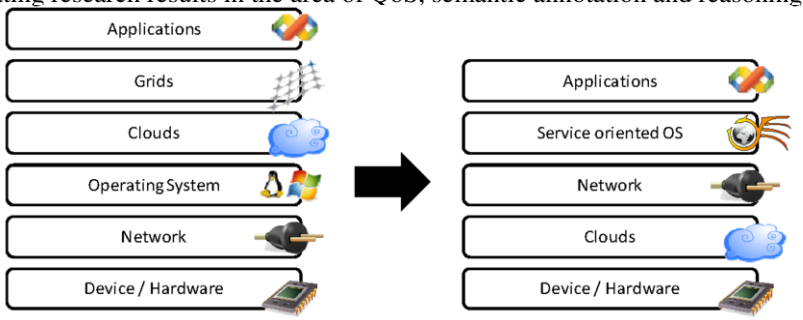

Figure 3: layer model without (left) and with (right) Service oriented Operating Systems

Currently, usage and research of Grid technologies in particular suffer from the lack of available, performing and stable infrastructures that are easy to use. With the SOS a new resource model is announced that incorporates main features relevant for cross-internet resource integration and usage in an efficient and managed fashion.

Along the same line, Clouds typically provide managed computational resources over a common interface, and are thus considered dedicated resources by the SOS. Accordingly, the operating system can principally make use of cloud computing and storage given the right interface, i.e. the right extension of the micro kernels. This would allow easy integration and hence easy exploitation of these capabilities.

With Service-oriented Operating Systems, the classical layer model (Figure 3, left) would change to the one depicted on the right in Figure 3 – note that Clouds themselves act *on top* of the network but would be integrated below the network (on resource level) from the end-user perspective acting on SOS.

3. The Service Oriented Operating System Concept

Service oriented Operating Systems would execute applications and processes *across* the resource fabric – i.e. as opposed to classical distributed operating systems or common grid middleware, the SOS concept makes use of resources on code / data level.

In other words, not only are services hosted by different providers, but more importantly, the executing code is distributed across the fabric, making use of specific conditions according to requirements.

This has the particular advantage that the *usability* of the system, i.e. the look & feel is adapted to the user's needs whilst extending the capabilities of the machine beyond the local resources. The operating system thus allows for new business models, where not only access to services is granted on a per-use basis, but also where whole code segments can be shipped and executed under license restrictions and limited to usage time, payment etc. This paper will *not* investigate the business aspects though for the sake of the underlying technical details (see e.g. [1]).

In order to achieve cross fabric code execution, SOS aggregates and virtualises the whole fabric as a distributed, annotated virtual memory for the execution layer – in other words the application, process or service runs in its own virtualised environment similar to a hypervisor, yet with the difference that all execution codes basically share the same environment *across* distributed resources and thus can exchange data easily.

The virtualisation technique bases on the same double blind invocation methodology introduced above: each application is hosted in its own virtual memory environment with the Service oriented OS mapping invocations between different such execution environments according to the respective setup and requirements.

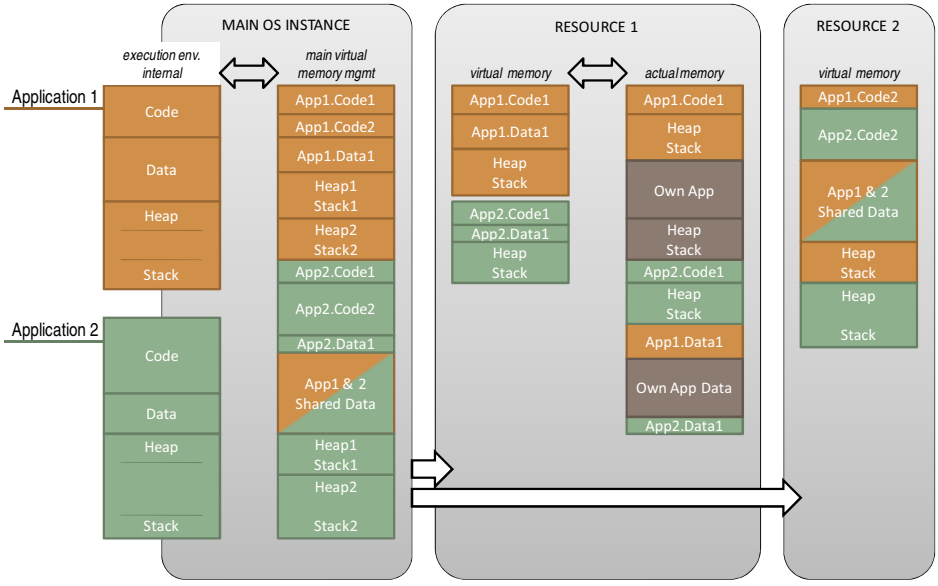

Figure 4: example distribution of applications across the resource fabric

Figure 4 depicts the virtual execution environment of each application and its actual mapping across the resource fabric: it can be noted that from the main OS instance (i.e. end-user side) all applications *appear* local, even though the actual code is distributed across resources according to execution specifics.

3.1. Code Segmentation

Even though processes typically perform a uniform functionality, they are nonetheless signified by a series of sub-routines and methods with individual requirements towards

infrastructure and hosting environment. One can thus e.g. distinguish between user interface and actual execution code in the background which may be idle for multiple cycles, when no specific user input requires actions. Obviously, this depends highly on the type of application, but it clearly shows that different code parts of one and the same application may have different requirements towards its hosting infrastructure. Notably, there are already commercial applications which address similar issues [8].

Service oriented Operating Systems can apply different methods to identify the code segment requirements: the principle approach bases on annotating the virtual memory in order to identify usage frequency and dependencies between segments, thus allowing assessments of couplings and basic infrastructure requirements. It also gives a rough indication of usage frequency, and thus of the user profile. Note that the accuracy of the segmentation and the according requirements increases over time, so that initial estimations may lead to performance losses. This is principally compensated through a "pessimistic" approach where requirements are initially estimated to be higher than the usage figures indicate. The details of such an annotated virtual memory management are beyond the scope of this paper and will be published separately.

Depending on these indicators, code blocks may be moved to remote low performance computers, to devices near the user with little delays in communication or somewhere in-between. The overall OS architecture thereby makes no distinction between different connection modes of the resources, i.e. the lower layers take care of the actual communication non-regarding the upper layers' requests. All resources are assigned with additional information related to their capabilities, such as connection bandwidth, response delay, computational speed etc. Information about these resources is gathered partially through ad-hoc publication, partially through run-time evaluation.

With the application requiring fast available (i.e. highly connected) resources, local ones will be preferred over remote ones, whilst e.g. computational intense processes with little interaction requirements may select remotely hosted resources that are only accessible via the internet. In other words, the lower layer of the OS takes care of communication modes relevant for the upper execution layer.

This makes the system transparent to the resources and thus also capable of hosting many-core platforms. As will be discussed below, this does not make the system more performing with respect to individual applications, but makes better use of resources when multiple threads are executed, in particular multiple applications in parallel.

3.2. Code Execution

Even with Quad-Core processors being more and more common, the typical use case is signified by multiple applications running in (pseudo)parallel and not with individual applications being executed in parallel threads. Accordingly, code execution is in its essence still sequential, with a main process steering the execution of individual applications and processes on different cores.

The Service-oriented Operating System is no exception to this rule: it is not proposing new methods to automatically parallelize code, even though the execution of the *overall* process may be distributed across multiple resources within the fabric. In other words, the basic behavior is identical to a many-core platform with the addition that the actual application or process may be broken up into sub-processes and executed remotely *according to the overall sequential execution.*

As illustrated, during "normal" pseudo-parallel execution of services and applications on e.g. a quad-core PC, each core is responsible for a dedicated set of

processes between which it constantly switches according to the multitasking concepts (Figure 5, left). In the case of SOS, the applications may be distributed across the resource fabric and executed effectively sequentially, whilst each processor switches between local and remote processes like in the classical case (Figure 5, right).

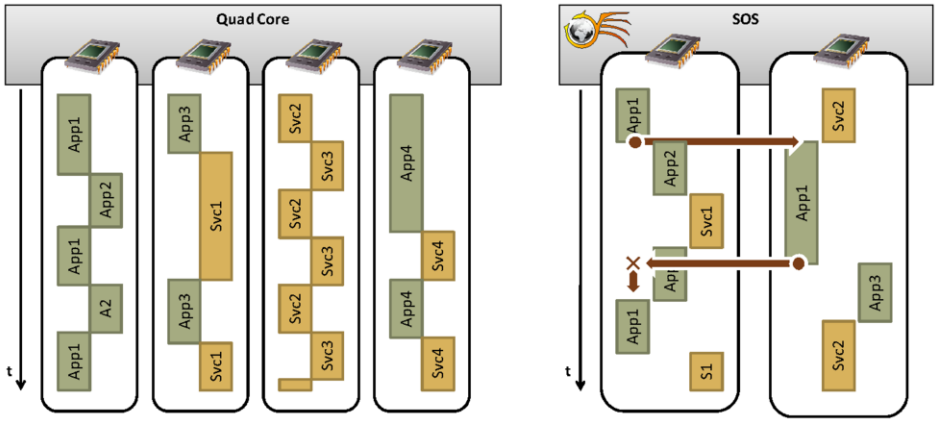

Figure 5: pseudo-parallel execution on a quad core PC (left) and execution distributed between two cores in the resource fabric (right)

One of the major problems during distributed execution does not consist in the distribution of the threads, but in handling conflicting invocations between currently active threads, i.e. maintaining the priority whilst distributing computational power between processes (cf. Figure 5, red arrows).

4. Critical Issues & Solution Approaches

Obviously, there are more critical issues involved in distributed code execution in the way presented here, than "just" scheduling between processes on a single node or core. Within this section, we will briefly discuss some of the major issues. It has to be kept in mind that Service-oriented Operating Systems such as presented here are not fully realized as yet, but still in an experimental research stage. It will also be noted that some of the issues have been addressed before, though in most cases not focusing on the full scale of such a distribution and execution model.

1. Stability and Reliability in Dynamic Networks

Resource fabrics are subject to potentially high volatility, since resources may become inaccessible or fail. This applies equally to network provided resources, as to nodes in a HPC cluster. Since in the SOS case, *essential* execution code may be hosted on a remote resource, losing connection to one of these resources will hence lead to the according process failing on a global level. It has to be noted though that the processes' main instance (the "provider") will have the full code available prior to its distribution. The information lost is related to the process *state* (or context) at a given time.

Where such resource loss can be predicted in advance, graceful shutdown is a simple means to circumvent process failure, as this allows moving the according code (or data) blocks to other OS instances in time. In many cases, resource loss cannot be predicted however and additional means need to be provided. The classical way of compensating node loss in grid-like environments consists in dedicated replicated

execution [13] or checkpointing [12]. However, checkpoints are stored only in specific intervals so as to reduce impact on execution – thus, failures may result in backtracking. In particular for short code blocks, this approach would lead to too high disruption.

SOS proposes to regard each application context change as a low-level checkpoint in which all execution information is stored in a highly available memory (RAM). Less available memories will lead to significant latencies during execution. Extended virtual memory management allows not only replicated memory across the web (with the according latency) but also asynchronous updating of memory with remote resources (without latency). The latter approach is effectively comparable to checkpointing on OS layer – however, with the Virtual Memory Manager being extensible, additional methods for delta propagation etc. can be applied to reduce the time-window between checkpoints, making them effectively identical to the actual execution timeframe.

2. Consistency in Distributed Shared Memory

Shared memory machines suffer from significant consistency issues comes to concurrent access to the same memory location. This is particularly true for parallel processes and applications that share memory (i.e. data). Numerous literature on this issue exists from both distributed systems perspective, as well as for concurrent thread execution and principally no further additions are needed here (see e.g. [17] [18] [25]).

Note that SOS is not a system for automatic parallelisation so that the issue is not of direct concern for the concept. Consistency is however an implicit issue, in particular for processes sharing data access, although in these cases, consistency maintenance is mostly up to the developer. There nonetheless needs to be a means to assess the state of data and to prevent essential failures due to concurrent access (read / write protection). As SOS shifts full code contexts between processing nodes, data is effectively replicated in multiple locations so that no single view exists.

The approach chosen by SOS consists in the hierarchical nature of the Virtual Memory Manager: as each executing resource hosts its own manager which obviously relates to different physical (local) addresses, the consistent view between the individual managers consists in an application specific virtual memory environment which implies a (virtual) data heap and stack. Consistency across these process specific views is maintained by the main OS' memory manager which maps all processes and memory locations across the fabric. With each execution shift, the "global" view on the memory state is updated to reference to the local view of the according executing resource, respectively considered blocked, if strong consistency protection is to be applied.

3. Scheduling

Scheduling is a classical issue in the area of distributed and grid computing [15] [16] – up to now, no general scheduling mechanism has been developed which is able to fulfill all relevant criteria. With SOS, this issue becomes even more critical: remote hosts may feed more than one main OS instance, so that the concurrent needs are not as easily evaluated as within a single main OS instance, in particular when the remote instance hosts a main instance for its own purposes competing over available resources.

As there is no general solution to this issue, SOS proposes the most straightforward approach, allowing more scenario specific approaches on a higher level. Effectively, this approach is basing on a fair-share scheduling discipline which takes an assigned priority as an adjustable weight into account. Priority weights are decided foremost by the resource owner and secondary by the process owner – as such, a resource owner can assign local processes with higher priorities than individual hosted

instances. Priority weighing may be applied to reduce the risk of malevolent assignments by "greedy" consumers.

4. Communication Latency

Communication latency is already critical within the close proximity and high bandwidth of nodes within a single cluster, even more so does the comparatively high latency of the web affect system execution [25] [26]. However, as plenty of web applications have already shown, this impact is *perceived* differently, depending on the type of application [27]. As SOS does not primarily aim at execution of applications with high performance computing needs, but instead at extending the resource scope for average application developers and users, perceived average performance is more relevant than optimizing communication paths, which is critical in HPC processes. Notably, dedicated HPC developers will explicitly specify the requirements of the individual threads and code blocks so that SOS will not have to identify "most suiting" resources according to own assessments.

In other words, primary concerns are maintenance of user interactivity so that no latency is perceptible, and increment of the *overall* performance in comparison to local execution. *Optimization* is thereby currently no concern. Accordingly, the impact of network latency depends completely on the type of application and its distribution across the web (cf. below).

5. Security

With remote instances hosting multiple processes from different main instances, security becomes a crucial concern: though the virtual memory manager creates exclusive instances, there is currently no way to protect the data from the resource owner or malicious attacks from other instances. Nonetheless, there exist numerous approaches to network security and data protection mechanisms that we do not consider this a specific concern of the operating system architecture as yet.

6. Code & Memory Block Relationships

"Live" code block assessment as described in section 3.1 is accompanied with the risk of frequent code block shifts during execution when new resources become available or the assessment changes during remote execution. Also, competition with new processes may lead to constant reallocation which obviously impacts on efficiency considerably. An outstanding issue to solve consists hence in the right cut-off between the gains from data shifts versus the loss of maintaining code blocks at the same site.

5. Conclusions

Bringing distributed capabilities down to the operating system is not a new approach as such: MPI is a means to write distributed (parallel) applications [9], and e.g. the IP project XtreemOS moves essential grid functionalities onto the OS level [10], thus increasing performance and reducing complexity. Further to this, the .NET Framework 3.5 brings essential networking and distribution features directly to the application layer [11] – future Windows versions may incorporate these features into the OS layer.

All these indicate a clear trend towards shifting the management overhead further away from the user and building up a distributed resource environment on OS / process execution level, so basically realizing a first step towards the resource fabric. However,

none of these approaches distributes actual execution code dynamically across different device types, thus exploiting the full resource fabric and in particular building up the future operating system that allows for dynamic low level resources and n-core support.

We have described in this paper how future OS models will realise a shift from application layer resource sharing to low-level resource webs, called “resource fabrics”. Such operating systems will not only shift a great amount of management overhead away from the user, thus making resource and application sharing more attractive, but it will also improve performance through reduction of messaging overheads.

These future models will enable flexible resource integration and distributed execution of code as originally envisaged by the Grid community (see e.g. [28]). Recent development in the area of Service Oriented Architectures, as well as the advances in computer and network architectures have now reached a stage where this original vision becomes feasible in a complete new fashion, namely the resource fabric.

Future operating systems building upon this model will thus significantly increase the dynamicity and stability of the main execution core, as it will become more independent of the resource fabric.

Acknowledgments

The work presented in this paper bases on results from collaborations and evaluations of the research projects TrustCoM (http://www.eu-trustcom.com), CoSpaces (http://www.cospaces.org) and BREIN (http://www. gridsforbusiness.eu) and has been partly funded by the European Commission’s IST activity of the 6th Framework Programme. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

[1] Schubert, L., & Kipp, A., 2008. Principles of Service Oriented Operating Systems. *in print*.

[2] Haller, J., Schubert, L., & Wesner, S., 2006. Private Business Infrastructures in a VO Environment. In P. Cunningham, & M. Cunningham *Exploiting the Knowledge Economy - Issues, Applications, Case Studies*. p. 1064-1071.

[3] Wilson, M.; Schubert, L. & Arenas, A. (2007), 'The TrustCoM Framework V4', http://epubs.cclrc.ac.uk/work-details?w=37589. Accessed January '08

[4] SalesForce, 'force.com - platform as a service', http://www.salesforce.com. Accessed June '08

[5] Amazon, 'Amazon Elastic Compute Cloud (Amazon EC2)', http://aws.amazon.com/ec2. Accessed June '08

[6] Hinchcliffe, D. (2008), 'The Next Evolution in Web Apps: Platform-as-a-Service (PaaS)', Technical report, Hinchcliffe Company, http://bungee-media.s3.amazon aws.com/whitepapers/hinchcliffe/hinchcliffe0408.pdf. Accessed August '08

[7] Englander, I. (1996), The Architecture of Computer Hardware and Systems Software: An Information Technology Approach, Wiley.

[8] Moskowitz, J. (2008), 'Making SoftGrid Apps Work On the Road', *Windows IT Pro* August '08.

[9] Gropp, W. (2000), Using MPI: Portable Parallel Programming with the Message-passing Interface, MIT Press.
[10] Cortes, T. et al (2008), 'XtreemOS: a Vision for a Grid Operating System', Technical report, XtreemOS, Project no. IST-033576, http://www.xtreemos.eu/publications/research-papers/xtreemos-cacm.pdf. Accessed September '08
[11] Microsoft, 'Microsoft .NET Framework 3.5', http://www.microsoft.com/net/. Accessed November '08
[12] Elnozahy, E.; Alvisi, L.; Wang, Y. & Johnson, D. (2002), 'A survey of rollback-recovery protocols in message-passing systems', *ACM Computing Surveys* **34**(3).
[13] Birman, K. P. & Joseph, T. A. (1989), Exploiting replication in distributed systems, *in* Sape Mullender, ed., 'Distributed Systems', ACM, New York, NY, USA, pp. 319-367.
[14] Saabeel, W.; Verduijn, T.; Hagdorn, L. & Kumar, K. (2002), A Model for Virtual Organisation: A structure and Process Perspective, *in* 'Electronic Journal of Organizational Virtualness', pp. 1-16.
[15] Doulamis, N.; Varvarigos, E. & Varvarigou, T. (2007), 'Fair Scheduling Algorithms in Grids', *IEEE Trans. Parallel Distrib. Syst.* **18**(11), 1630-1648.
[16] Phan, T.; Ranganathan, K. & Sion, R. (2005), Evolving Toward the Perfect Schedule: Co-scheduling Job Assignments and Data Replication in Wide-Area Systems Using a Genetic Algorithm 'Job Scheduling Strategies for Parallel Processing', Springer Berlin / Heidelberg, pp. 173-193.
[17] Adve, S. V. & Gharachorloo, K. (1995), 'Shared Memory Consistency Models: A Tutorial', Technical report, Rice University ECE.
[18] Mosberger, D. (1993), 'Memory consistency models', *SIGOPS Oper. Syst. Rev.* **27**(1), 18-26.
[19] Catlett, C. & Smarr, L. (1992), 'Metacomputing', *Communications ACM* **35**(6).
[20] Laure, E. et al (2004), 'Middleware for the next generation Grid infrastructure', (EGEE-PUB-2004-002)
[21] DEISA Primer, http://www.deisa.org/files/DEISAPrimer-V1-1.pdf, Accessed August '08
[22] Surridge, M.; Taylor, S.; Roure, D. D. & Zaluska, E. (2005), Experiences with GRIA - Industrial Applications on a Web Services Grid, *in* 'E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing', IEEE Computer Society, Washington, DC, USA, pp. 98-105.
[23] Dimitrakos, T. et al (2003), An Emerging Architecture Enabling Grid-based Application Service Provision, *in* '6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003)'.
[24] Jefferey, K. & Snelling, D. (2004), 'Next Generation Grids 2', Technical report, EC, ftp://ftp.cordis.lu/pub/ist/docs/ngg2_eg_final.pdf. Accessed May '08
[25] Delaney, D.; Ward, T. & McLoone, S. (2006), 'On Consistency and Network Latency in Distributed Interactive Applications: A Survey - Part I', *Presence: Teleoperators & Virtual Environments* **15**(2), 218-234.
[26] Delaney, D.; Ward, T. & McLoone, S. (2006), 'On Consistency and Network Latency in Distributed Interactive Applications: A Survey - Part II', *Presence: Teleoperators & Virtual Environments* **15**(4), 465-482.
[27] Ruddle, A.; Allison, C. & Lindsay, P. (2001), Analysing the latency of WWW applications, *in* 'Computer Communications and Networks', pp. 116-121.
[28] Foster, I. & Kesselman, C. (1998), Computational grids, *in* 'In VECPAR', Morgan Kaufmann, pp. 15-52.