A Convergence Architecture for GRID Computing and Programmable Networks

Christian Bachmeir, Peter Tabery, Dimitar Marinov,
Georgi Nachev, and Jörg Eberspächer

Munich University of Technology, Institute of Communication Networks
bachmeir@ei.tum.de, http://www.lkn.ei.tum.de

Abstract. GRID computing in the Internet faces two fundamental challenges. First the development of an appropriate middleware for provision of computing services in a scalable and secure manner. Second the distributed, scalable and fast connection of GRID processing components to the network. The latter is important to enable fast data exchange among the distributed components of virtualized super-computers.
In this work, we focus on the connection aspects of GRID architectures and advocate to enhance edge-routers with clusters of high performance computing machines.
We propose enhancing available GRID resources to a component-based programmable node. Our approach delivers three advantages: Use of the same hardware/software for GRID computing and Programmable Networks. Support of GRID computing through–then available–programmable services, like data transport and distribution in the GRID through Programmable Networks. Finally we see GRID computing as a leverage for the future deployment of Programmable Networks technology.

1 Introduction

In this paper we propose a convergence architecture for GRID computing and Programmable Networks. The primary design goal of our approach is to provide a flexible platform that can be integrated in a GRID (see Figure 1) and a Programmable Network simultaneously. The major benefit of our approach is the convergence of the two network-based, distributed computing technologies. Our approach can provide both, the execution of GRID processing jobs as well as the provision of Programmable Network services.

Basic idea of our approach is a spatial separation between *routing* of traffic, the actual *processing* of services and the *control* of the system across different machines.

Through this separation we provide a powerful, scalable programmable architecture that is integrated into a GRID. We introduce dedicated signaling hosts in our architecture, that provide access to the programmable node, based on standard web services. Using this open signaling approach our architecture can be integrated in GRID middleware.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 187–194, 2004.

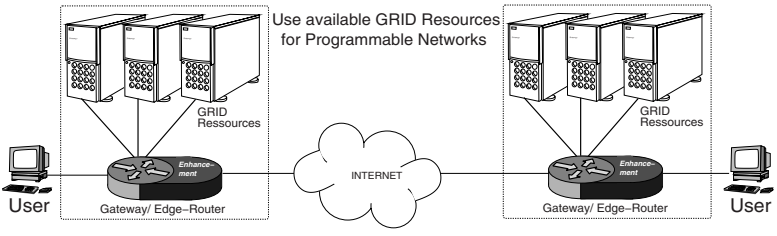

Fig. 1. Available Grid Ressources

Main contribution of this work is the integration of a new layer 4 encapsulation and proximity switching mechanism in programmable architectures, which **obsoletes explicit kernelspace-to-userspace communication** on processing machines. Based on layer 4 encapsulation and decapsulation, IP data packets are forwarded to to processing components by the router component.

Using standard UDP sockets, we introduce the concept of *Operating System as Execution Environment (OSaEE)*. We propose to execute service modules as standard userspace programs. These service modules can mangle original IP packets by receiving and sending them using **standard UDP sockets**. As we do not use kernel-userspace communication, the principle of OSaEE is inherently the same as processing GRID jobs.

Based on three mechanisms (layer 4 forwarding, OSaEE and a reliable signaling architecture) we present a robust architecture that bypasses security and fault tolerance limits which are currently faced by proposed programmable architectures, and which prohibit to use the resources of programmable architectures in a GRID.

This paper is organized as follows: In the next Section an overview of our architecture and the interworking with the GRID is presented. In Section 3 we show performance measurements of our prototype implementation.

2 Overview of the Proposed Architecture

In Figure 2, an overview of the programmable node architecture is shown. We derive our architecture from HArPooN [1], and propose to spatially separate routing functions from computing functions. The routing resp. switching is done in the **Routing Component**, an enhanced standard router. With *computing* we subsume the *signaling*, necessary for access and control, the provision of *programmable services* and the use of the resources in a GRID context. Our architecture executes these computing functions on machines in the **Computing Component**.

Key component of our architecture is a lightweight interface between the router and the computing cluster. We propose to enhance the architecture of the router with a traffic encapsulation and decapsulation module (see paragraph 2.1). When using our architecture as a programmable node this module is used to forward data packets to processing machines, which provide active and programmable services. When accessing GRID components, the module is bypassed.

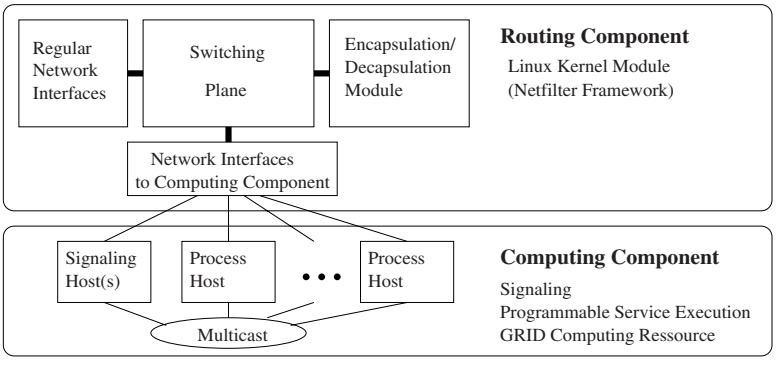

Fig. 2. Component based Programmable Router & GRID Architecture

2.1 Routing Component and OSaEE

In this section we elaborate on the encapsulation and decapsulation module, located in the routing component. Every time an IP packet arrives at the routing component, it is first checked whether the packet originates from the computing cluster of our proposed programmable architecture. If the IP packet does not originate from the computing cluster, the local database (provided by Module Control) is queried whether this IP packet belongs to a traffic flow, for which our architecture currently provides a service. If this is the case, the entire packet is encapsulated within a UDP packet[1]. The UDP/IP header is set to address the process host and the port number of the corresponding service module. Because of the encapsulation header, the packet does not continue its original way through the network, but is routed towards the processing machine and received by a service module with a corresponding UDP socket.

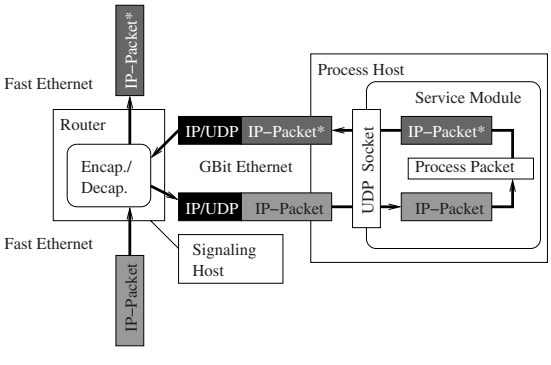

Fig. 3. Data flow in the component based Programmable Node

[1] Fragmentation due to encapsulation and enlargement of the packet size does not happen as we use GBit Ethernet between Routing and Computing Component, which allows "Jumbo packets" with a frame size up to 9 Kbyte

The processed IP packet is then sent by the service module as a payload using again the UDP socket. The IP destination address and the destination port number are set to the IP address of the routing component and a determined port number.

When the encapsulation/decapsulation mechanism of the routing component receives this encapsulated packet, it decapsulates the packet (deleting exterior IP and UDP headers). The remainder of the packet–the IP packet constructed by the service module–is routed by the routing component on its way through the Internet.

In some proposed architectures the loadable service modules are granted administrator privileges, due to kernel-userspace communication implementation issues. Obviously this is a back door for all kinds of attacks towards the system architecture. Therefore, security mechanisms are engineered around the service modules (e.g., the service module is prevented from deleting certain files on the system, it is not allowed to fork processes, etc.).

In case of architectures where loadable service modules are proposed as kernel modules [2], the issues concerning security is even a more severe one. Kernel modules cannot be controlled against security violations at all. Therefore architectures have been proposed which are using access policies, primarily building on trusted code sources.

When discussing security of programmable nodes against *internal* attacks, we felt like "reinventing the wheel". Therefore we propose to *make our architecture independent towards security issues like internal attacks*, through OSaEE (Operation System as Execution Environment)[2].

We consider the development of the security of standard operating systems (e.g. Linux) against internal attacks already at a mature stage. When developing OSaEE we intend to leverage that basis for our architecture.

The general goal of OSaEE is to use a standard operating systems on machines in the computing cluster, without any modifications as a platform to execute loadable programs. On that basis OSaEE enables us to use standard security features of the deployed operating system, e.g. the user management. Each service module (userspace program) is then started under a different user. Once a certain user consumes "too much" system resources or generates "too much" traffic, the control process of the system reacts, e.g. by killing offending processes of that specific user. By introducing OSaEE we see major advantages compared to state-of-the-art programmable architectures:

- Our architecture is simplified, leading to a higher robustness and usability.
- Using OSaEE we make our architecture independent towards security requirements regarding internal attacks of the system.
- Flexibility: Service modules can be written in a variety of programming languages

[2] External attacks, targeting or originating by the programmable node are clearly out of the scope of this work. We state that proposed mechanisms can be integrated in our architecture.

- Due to abstraction of network stack: Composition of complex services using special libraries (e.g. transcoding) in the userspace.
- Process Hosts may utilize different operating systems.
- Computing Resources can be used simultaneously in a GRID.

In Figure 3 the exemplary data flow of a single IP packet through HArPooN as described in paragraph 2.1 is shown. Due to the *Encap./ Decap. mechanism*, regular userspace programs are able to intercept entire IP packets that want to bypass the Router Component. On the processing machines in the computing cluster there is no modification necessary to enable this mechanism as the userspace programs only need to be able to use standard UDP sockets.

Besides improving issues for Programmable Networks, the adaptability of OSaEE also enables us to offer computing resources of our programmable architecture to a Computing GRID. The necessary flexible signaling mechanism is presented in the next paragraph.

2.2 Interworking: Layered Signaling

Because of the spatial separation between routing component and computing cluster, there is a need of synchronization between both. In our approach we introduce dedicated **signaling hosts** (see Figure 2) to perform this task. To enable fault tolerance our approach foresees supplemental signaling hosts, running in hot standby. The signaling hosts are basically responsible for three main tasks:

1. Control of the Router Component (entries in Encap./ Decap module)
2. Basic external signaling (load, start, and monitor a service or GRID jobs on our architecture).
3. Internal multicast-based signaling (for monitoring and controlling the Process Hosts)

These three mechanisms are general and service unspecific, therefor we subsume all three under the expression **Basic Signaling**.

It might be the case that for a specific service to work, there is a **Service or Job Specific Signaling** necessary. In case of a service (or a GRID job) that consists of modules which are distributed on numerous machines in the Internet, it is necessary to allow e.g. TCP connections between the different service modules for data exchange. We consider that kind of signaling as part of the service (or job) itself. In our approach, distributed service modules can communicate with each other, using standard sockets. The Routing Component does not affect these data streams. This separation between **general basic signaling** and (loadable) **service or job specific signaling** delivers a layered signaling architecture presented in Figure 4. In our implementation of the system, we use Java technologies[3] to realize the *general basic signaling*. Using the Java Web Services mechanism, the basic external interface can easily be provided, because

[3] Java 1.3.1, Netbeans IDE 3.5

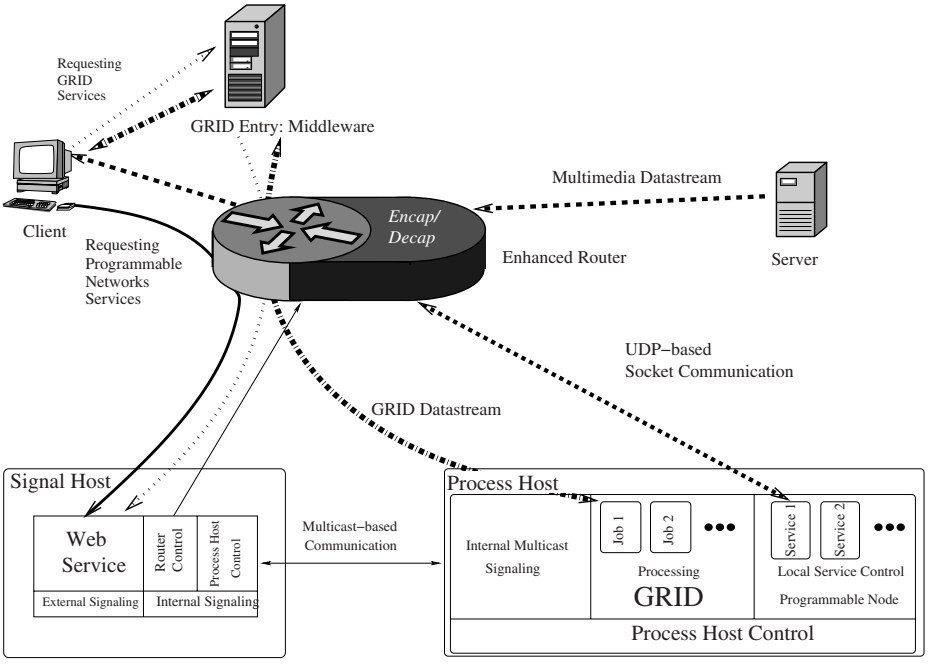

Fig. 4. Layered Signaling and embedding of signaling components in the architecture

the basic functionality is already implemented by the web service frameworks (like JAX-RPC[4], that we used).

The *Process Hosts Control Module*[5] is using the internal multicast-based signaling interface to access Process Hosts. The basic task of the multicast protocol is, to start and stop programs representing active services on the process hosts. Also other general, service unspecific information, e.g. monitoring, is exchanged via this interface. We implement that protocol based on multicast solely for reasons of fault tolerance (hot standby) and scalability of the whole system, when employing several Process Hosts. Our approach also provides a second signaling host, which can take over immediately the control of the respective services in case of a failure. Moreover the strict separation between Basic External Signaling and Internal Multicast based Signaling is also a security feature of our approach. It is not possible to start a service (or a GRID job) directly on a process host from outside of our architecture, only signaling hosts are allowed to do so.

3 Performance Evaluation of the Programmable Node

In this section, performance evaluation results of our approach are presented. In Figure 5 the perceived packet loss in our testbed is shown. As outlined in

[4] Java API for XML-based RPC: Simplifies Programming of Web Services
[5] Realized with Enterprise Java Beans: Performant execution of parallel beans

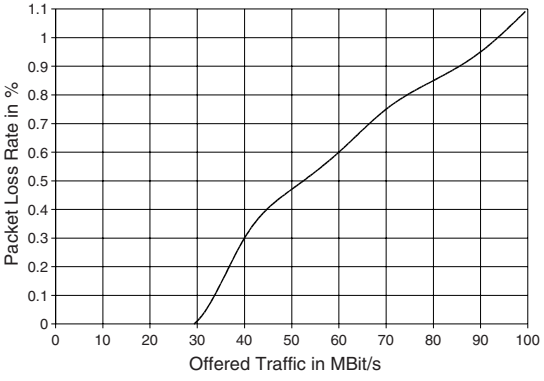

Fig. 5. Packet Loss in our prototype

figure 4, our prototype consists of five machines. One is a 700 MHz Linux-based Pentium 3 machine with four 1 GBit Ethernet cards, which represents the modified router. the others are Pentium 4 machines, acting as client, server, signaling host and one process host. Client and server are connected using 100 MBit Ethernet to the router. We send multimedia datastreams at constant bit rates from server to client. The data packets of the stream where intercepted at the router, encapsulated and forwarded to the Process Host. At the process host data packets were received, using UDP sockets and sent back[6] to the router. There they where decapsulated and forwarded to the client.

Although we used hardware with relatively low performance for the prototype implementation of the router, we perceived a considerably low packet loss. Even when offering 100 MBit/s of data only 1.1 % of the packets were lost. Compared to measurements of state-of-the-art implementations of Programmable Networks [3] our approach is clearly competitive.

We state that perceived losses will vanish entirely, when using a modified standard router hardware, instead of the PCI-Bus PC used in our prototype.

We did not measure our approach against multi machine based active and programmable node architectures like e.g. [4], as the used hardware cannot be compared. However we state if a high performance router–with proposed modifications–would be embedded in our architecture, the performance should be at least comparable to proposed architectures.

4 Conclusion

In this work we present a new architecture of a programmable node based on GRID computing resources. Foundation of our approach is, to keep changes at

[6] Data packets where not processed resp. modified at the Processing host, as we only measured for maximum throughput. Due to possible load balancing our architecture improves execution time of services anyway [1].

the actual router at a minimum, when transforming a regular router into an active and programmable router. Therefore we see a good chance to implement our proposal in future products and enable deployment of active and programmable services outside research environments.

In this work we present two major contributions: First, we provide a flexible platform that can be integrated in a GRID. We see a major benefit in this approach as available resources can be used concurrently for programmable network services as well as for computation of jobs in a GRID context. Besides the programmable part, our architecture delivers means to enhance the GRID, e.g. through the provision of dynamic configurable data transport mechanisms.

Second, we provide a new high performance architecture of a component based programmable router. Through the separation of routing, signaling and processing on different machines we improve local security and fault tolerance of the programmable node. The primary benefit of our architecture is in the provision of complex, resource-demanding active and programmable services.

We think that our approach is a powerful enhancement to the GRID. Based on the emerging deployment of GRID computing services, we see a potential for leveraging programmable networks technology.

References

[1] Bachmeir, C., Tabery, P., Sfeir, E., Marinov, D., Nachev, G., Eichler, S., Eberspächer, J.: HArPooN: A Scalable, High Performance, Fault Tolerant Programmable Router Architecture. In: Poster Session in IFIP-TC6 5th Annual International Working Conference on Active Networks, IWAN'2003, Kyoto, Japan (2003)

[2] Keller, R., Ruf, L., Guindehi, A., Plattner, B.: PromethOS: A dynamically extensible router architecture supporting explicit routing. In: 4th Annual International Working Conference on Active Networks, IWAN 2002, Zurich, Switzerland (2002)

[3] Conrad, M., Schöller, M., Fuhrmann, T., Bocksch, G., , Zitterbart, M.: Multiple language family support for programmable network systems. In: Proceedings of the 5th Annual International Working Conference on Active Networks, IWAN'2003, Kyoto, Japan (2003)

[4] Kuhns, F., DeHart, J., Kantawala, A., Keller, R., Lockwood, J., Pappu, P., Richards, D., Taylor, D., Parwatikar, J., Spitznagel, E., Turner, J., Wong, K.: Design of a high performance dynamically extensible router. In: DARPA Active Networks Conference and Exposition (DANCE), San Francisco (2002)