DiProNN Resource Management System

Tomáš Rebok

Faculty of Informatics
Masaryk University
Botanická 68a, 602 00 Brno
xrebok@fi.muni.cz

Abstract. The Distributed Programmable Network Node (DiProNN) we proposed earlier allows user-controlled processing of passing user data in a network. The node takes advantages of virtualization principles, which make DiProNN being able to accept not only standalone active programs (the programs performing the stream processing), but the whole virtual machines with their own operating systems, and their own set of active programs running inside them. Furthermore, thanks to the virtualization principles we proposed innovative programming model which makes the programming of complex stream processing applications for DiProNN much easier.
However, when the node is being used in multiuser environment, the users must share its limited resources, namely its CPU time, network I/O, disk I/O, and memory subsystem. In this paper, we present the resource management (RM) system of the DiProNN node including resource discovery process performed during node startup. Furthermore, we present the specification of DiProNN users' resource requests and DiProNN resources reservation process itself. In the end of the paper, we briefly describe the DiProNN prototype implementation enhanced by the resource management system described.

1 Introduction

In [1] and [2] we proposed the architecture of the *Distributed Programmable Network Node (DiProNN)*, that uses the principle called "Active/Programmable Networks"—an attempt how to build an intelligent and flexible network using current networks serving as a communication underlay. The proposed node employs the virtualization principles [3], that enable DiProNN users to upload not only standalone active programs (the programs performing the stream processing on the node), but the whole virtual machines with their own operating systems and their own set of active programs (APs) running inside them. Moreover, the usage of virtual machines (VM) principles enabled us to propose an innovative programming model proposed in [4].

However, the application of such a programmable node in a shared network environment pose new challenges in the design of its resource management (RM) system. Thus, more resources are shared among the users of the node—the CPU cycles, state storage capacity, and data storage together with traditional networking components like packet queues on network interfaces. To enable sharing of all these resources within the DiProNN by its users we employ the RM system as described in this paper. The system is based on a simple per-VM resource reservation mechanism, that enable DiProNN users to precisely reserve the resources they need for their proper DiProNN session processing.

The main goal of this paper is to illustrate the RM system employed in DiProNN, and to show how DiProNN users can request reservations of DiProNN resources to ensure proper processing of their data stream(s). The paper is organized as follows: for the purposes of this paper, the Section 2 briefly describes the DiProNN architecture, while the following Section 3 describes the DiProNN resources discovery process in detail. The Section 4 then briefly presents the RM-enabled prototype implementation we created, while in the next section we present works related to the ours one. Finally, in Section 6 we conclude and present our future work.

2 DiProNN: Distributed Programmable Network Node

DiProNN architecture we propose assumes the infrastructure as shown in Figure 1. The computing nodes form a computer cluster interconnected with each node having two connections—one *low-latency control connection* used for internal communication and synchronization inside the DiProNN, and at least one[1] *data connection* used for receiving and sending data.

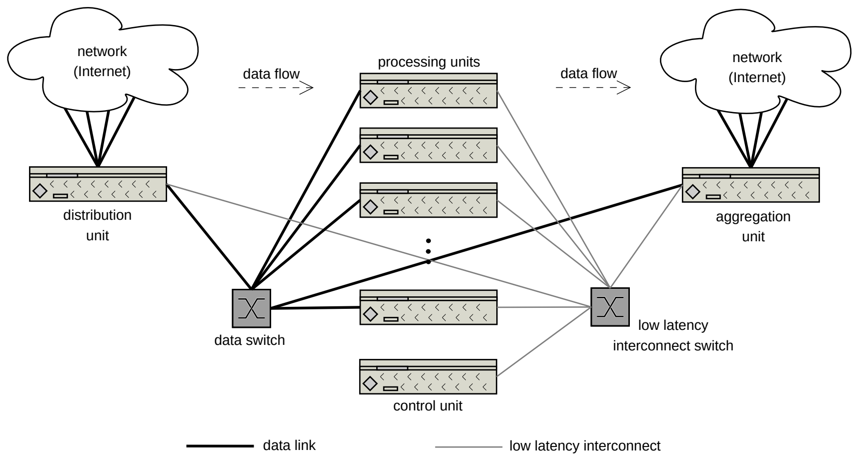

Fig. 1. Proposed DiProNN architecture.

[1] The ingress data connection could be the same as the egress one.

For the purposes of this paper, the most important DiProNN units depicted in the Figure 1 are the *Control Unit* and the *Processing Unit(s)*[2].

The *Control Unit* serves as the main control point of the whole DiProNN node. It communicates with all the other DiProNN units, collects all the necessary information and decides about various events in DiProNN (e.g., VM migration, accepting/refusing new DiProNN sessions, collecting all the accounting information, etc.). From the DiProNN RM system point of view, the Control Unit maintains information about the actual state of all the DiProNN resources. It communicates with the Resource Management module of all the units (see later) and collects all the information about the actual state and usage of their resources monitored. For example, the information about available resources is then used for decisions about accepting or refusing new DiProNN sessions and for decisions about necessary virtual machine(s) migration in order to use the DiProNN resources in an efficient way.

The *Processing Units* having their architecture depicted in the Figure 2 are the only DiProNN units necessarily hosting the virtualization mechanism and virtual machines (usually called *domains*), where user uploaded APs and VMs performing passing stream processing run. The Processing Units' Service domain depicted in the Figure 2 has to manage the whole unit functionality including uploading, starting and destroying of the virtual machines, communication with the Control Unit, and a session accounting and management. To provide all of these the Service domains run set of modules depicted in the figure—for the purposes of this paper, the most important ones are the *Control module* controlling the Processing Units' behavior and functionality, the *VM/AP Management module* managing all the virtual machines and active programs running on the unit, and the *Resource Management (RM) module* providing monitoring and controlling of units' resources.

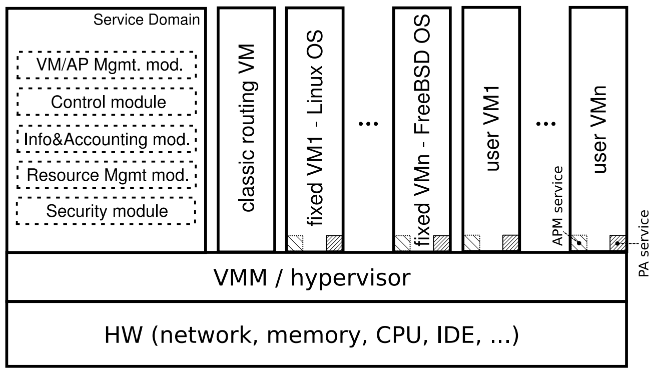

Fig. 2. DiPRoNN Processing Unit Architecture.

[2] Further details about the whole DiProNN architecture as well as the architecture of all the DiProNN units can be found in [1] and in [2].

3 DiProNN Resources Discovery Process

The resource discovery process takes place during DiProNN initialization, which is the process starting with all the DiProNN units startup to the moment, when the node is ready for new sessions establishment. During the process, not only the hardware, but also the software DiProNN resources have to be discovered.

By the software DiProNN resources we mean all the active programs providing the built-in DiProNN functionality, and all the (fixed) virtual machines serving as an execution environment for standalone APs[3]. The hardware resources are the ones we have already mentioned—CPU time, network, memory, and storage subsystem access.

During the initialization, all the DiProNN units have to start their necessary service utilities, and all the Processing Units have to register their fixed virtual machines and active programs running inside them at first. The APs registration is performed using the *Active Program Manager (APM) service* running inside each VM (see Figure 2), which sends the information about all the built-in APs running inside each VM to the VM/AP Management module of the relevant Processing Unit.

When the APs registration process finishes, the units registration process takes place. During the registration, all the DiProNN units (their Control module) send the registration request to the DiProNN Control Unit. When the unit registration process finishes, the available DiProNN HW & SW resources collection process takes place.

3.1 DiProNN HW & SW Resources Collection

After the successful registration of the units, all the RM modules running on the units are contacted by the *DiProNN Resource Management module* running on the Control Unit to receive information about their both HW and SW resources available. This information is periodically obtained by the module and, as mentioned in the Section 2, serves mainly for the built-in APs discovery and the decisions about new standalone APs and VMs placement, and VMs migrations.

For resources discovery and information exchange among the units, the use of a standardized monitoring software tool is highly desirable. The examples of such a tool are the *Ganglia* [5]—a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids—and the *GLUE schema* [6], which is an information model for Grid entities described using natural language and enriched with a graphical representation using UML Class Diagrams.

The resources discovery and monitoring software tool has to collect, and to the DiProNN Resource Management module has to provide following set of information about:

[3] As mentioned in Section 1, DiProNN users are enabled to upload both the virtual machines running set of APs performing requested stream processing, or the standalone APs if no VMs are necessary. But such standalone APs have to run in any VM, and thus, some fixed VM(s) serve as an execution environment for them.

- *virtual machines* running on all the DiProNN Processing Units. The information consists of a type of the VM (whether it can serve as an execution environment for standalone APs), VM reference name (the name used when referencing APs running inside them, see [4]), VM operating system information (used for standalone APs requiring specific OS type for their run) and simple description provided to DiProNN users.
- *built-in APs* running in the fixed VMs. The information provided contains the AP reference name, the VM reference name the AP is running in, and simple description of the AP functionality (provided to DiProNN users when requesting the overview of built-in APs running in the node).
- *HW resources* available on the DiProNN units. The resources monitored cover information (status and utilization) about all the CPUs, network I/O, disk I/O, and memory subsystem.

3.2 Resource requests specification

As described in [4], the DiProNN users specify their HW resource requests in so called *DiProNN programs*. The specification of requested resources may be specified in both VM parameters section(s) and standalone AP parameters section(s) (see [4]). The resources are specified using the `CPU_REQUEST`, `MEM_REQUEST`, `NET_REQUEST` and `DISK_REQUEST` keywords as depicted in following example:

```
{ AP name="first_AP" ref="first_AP_reference";
     # AP parameters
     CPU_REQUEST = 60    # percent
     MEM_REQUEST = 128   # MB
     DISK_REQUEST = 100  # MB
     NET_REQUEST = 1000  # Mbps
     inputs = ...;
     outputs = ...; }
```

Fig. 3. Example part of a DiProNN program specifying resource requests.

The DiProNN programs requesting new DiProNN session establishment are through the Distribution Unit delivered to the Control Unit, which depending on the actual usage of DiProNN resources obtained from the DiProNN Resource Management module (running on the Control Unit) decides about the accepting/rejecting of the DiProNN session request. If the request could be satisfied, the HW resource allocation process take place.

3.3 Resource allocation process

Since DiProNN does not rely on specific virtualization system, the real resource allocation process as well as the precise specification of the amount of requested resources (as depicted in the Figure 3 depend on the features of the virtualization system used.

4 Prototype implementation

To test the RM system described we enhanced our previously created basic prototype implementation based on the Xen Virtual Machine Monitor (VMM) [7]. The new DiProNN implementation we created covers all the three basic aspects of the RM system described—the resource state and utilization information gathering, users' resource requests specification, and the requested resource allocations themselves.

Since the resource requests specifications we use are exactly the same as depicted in the Figure 3, they are not described in this section again. Furthermore, because of limited number of pages, for the comprehensive performance tests of the DiProNN prototype being described let us refer the reader to [8].

4.1 Resource state and utilization information gathering

To gather all the resource information we decided to use the Ganglia monitoring tool mentioned in previous section (mainly because of its simple and powerful design and simple extensibility).

The Ganglia comprises of two daemons—the *Ganglia Monitoring Daemon* (`gmond`) and the *Ganglia Meta Daemon* (`gmetad`). For our RM-enabled DiProNN implementation, the `gmond` served as a gathering agent running on particular DiProNN units, while the `gmetad` served as an aggregation agent running on the DiProNN Control Unit only.

Since the original `gmond` is able to monitor the CPU units, network I/O, disk I/O, and memory subsystem only, to enable monitoring of SW resources (built-in APs and VMs) we created a dynamically loadable module (so called *agent* in Ganglia terminology) extending the available metrics. The created agent, called `VMAPmon_module`, directly communicates with the VM/AP Management module of the relevant Processing Unit and collects all the information about built-in APs and fixed VMs running on the unit. The `gmond` then provides this information together with the information about the CPU, network, disk and memory to the `gmetad` daemon.

The `gmetad` collects all the SW and HW information provided by the `gmond`s of all the DiProNN units and provides it to the DiProNN Resource Management module running on the Control Unit. The information is then used for accepting/rejecting of new DiProNN sessions (depending on the resources required vs. resources available) and decisions about suitable VMs for standalone APs uploading.

4.2 HW resource allocations

In our RM-enabled prototype, the resource allocations are realized using common resource-controlling tools provided with the Xen VMM system. If a Processing Unit hosts a VM (or a standalone AP) belonging to an accepted DiProNN session, the Control Unit sends to its VM/AP Management module relevant part of the DiProNN program including the resource requests specified. The VM/AP

Management module then in cooperation with the Resource Management module running on the same Processing Unit proceeds the resource allocation itself.

As mentioned previously, the RM-enabled DiProNN prototype implementation being described is able to reserve following set of resources:

- *CPU time* – based on available DiProNN schedulers evaluation we did (see [8]), for our DiProNN prototype we decided to use the *credit scheduler* in so called *work consuming (wc) mode*. The scheduler accepts two attributes for each VM—so called *weight* representing the domain's CPU priority, and so called *cap* optionally fixing the maximum amount of CPU a domain will be able to consume.

  In our case, based on the CPU request the VM/AP Management module tries to find a CPU having enough free computing time, which the relevant VM will be dedicated to (if there is any, otherwise the session request is rejected). If two or more VMs do have to share one physical CPU unit, their *weight* attribute of the credit scheduler is set appropriately so that all the VMs do receive proper portion of the CPU time.
- *Memory* – for the maximum memory information we use the `memory` parameter of the VM configuration file. The memory is allocated and dedicated to the VM during its startup.
- *Network bandwidth* – for the network bandwidth reservations, we use the `Traffic Control (tc)` tool available under Linux, namely its shaping functionality. All the domains are reserved (and restricted to) their requested network bandwidth specified.
- *Disk capacity* – if a domain requests an additional disk space than the one available in the VM image, using the `disk` parameter of the VM configuration file we assign it a partition (previously created) of the size requested.

5 Related Work

Thanks to amazing flexibility, the programmable networks principles became very popular and thus various architectures of programmable routers have been proposed. This section briefly depicts only those ones mostly related to our work.

C&C Research Laboratories propose the CLARA (CLuster-based Active Router Architecture, [9]) providing customizing of media streams to the needs of their clients. Computational resources available in CLARA could be also reserved—the hierarchies of schedulers are created, making it possible to divide the computational resources available on a computing router.

The [10] presents an architecture of the *QuaSAR* router, trying to improve the Quality of Service guarantees provided to applications transporting time-sensitive data across the best-effort Internet. The architecture uses the virtual machine techniques to assign an individual virtual *routelet* to each network flow requiring QoS guarantees—the appropriate amount of resources is then assigned to each (virtual) routelet.

The PlanetLab project [11] also introduces their approach to isolate shared network resources in their Xen VMM-based *XenoServers*.

6 Conclusions and Future Work

In this paper, we have presented the resource management system of the Distributed Programmable Network Node (DiProNN) we proposed. The RM system we described deals with all the three basic aspects we mentioned—the resource state and utilization information gathering, users' resource requests specification, and the requested resource allocations themselves. The RM-enabled DiProNN prototype implementation was also depicted.

Concerning the future challenges, we would like to explore the DiProNN sessions scheduling from global point of view—the scheduling of APs/VMs to suitable DiProNN nodes when there are more DiProNN nodes able to participate on the processing of given DiProNN session.

Acknowledgement: This project has been supported by research intent "Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems" (No. 102/05/H050) and "Optical Network of National Research and Its New Applications" (MŠM 6383917201).

References

1. Rebok, T.: DiProNN: Distributed Programmable Network Node Architecture. In: ICNS'08 Conference Proceedings. Fourth International Conference on Networking and Services. (2008) 67–72
2. Rebok, T.: DiProNN: Distributed Programmable Network Node Architecture. In: CGW'07 Conference Proceedings. (2008) 283–290
3. Smith, J.E., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Elsevier Inc. (2005)
4. Rebok, T.: DiProNN Programming Model. In: MEMICS'07 Conference Proceedings, MEMICS 2007 (2007) 978–80
5. Massie, M., Chun, B., Culler, D.: The Ganglia Distributed Monitoring System: design, implementation, and experience. Parallel Computing **30**(7) (2004) 817–840
6. Andreozzi, S., et al.: GLUE Schema Specification–version 1.3. OGF: https://forge.gridforum.org/sf/go/doc14185 (2005)
7. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., m, P.B., Neugebauer, R.: Xen and the Art of Virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA (2003)
8. Rebok, T.: Měření plánovacích algoritmů prototypu aktivního směrovače s podporou QoS. `http://www.fi.muni.cz/~xrebok/DiProNN_QoS/DiProNN_QoS.pdf` (2008)
9. Welling, G., Ott, M., Mathur, S.: A cluster-based active router architecture. IEEE Micro **21**(1) (2001) 16–25
10. Sventek, J.;McIlroy, R.: Resource virtualisation of network routers. In: International Workshop on High Performance Switching and Routing (HPSR), IEEE (2006)
11. Warfield, A., Hand, S., Harris, T., Pratt, I.: Isolation of Shared Network Resources in Xenoservers. Technical Report PDN–02–006, PlanetLab Consortium (2002)