Dynamic Photonic Lightpaths in the StarPlane Network

Paola Grosso[a,*] Damien Marchal[a] Jason Maassen[b] Eric Bernier[c] Carol Meertens[a] Li Xu[a] Cees de Laat[a]

[a]*System and Network Engineering Group, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*

[b]*Dept. of Computer Science, Vrije Universiteit, De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands*

[c]*Nortel Networks, 3500 Carling Avenue, Ottawa, Ontario K2H 8E9, Canada*

Abstract

The StarPlane project enables users to dynamically control network photonic paths. Applications that run on the Distributed ASCI Supercomputer (DAS-3) manipulate wavelengths in the Dutch research and education network SURFnet6.

The goal is to achieve fast switching times so that when the computational pattern in the computing clusters changes, the underlying network topology adapts rapidly to the new traffic flow. StarPlane develops: - the software to perform optimal traffic engineering; - the management plane to map the users' request into directives for the network control plane, as well as the integration to the DAS-3 Grid middleware. We present here some preliminary results obtained with few selected Grid applications that make use of StarPlane.

Key words: Photonic networks; application-controlled networks; wavelength-selective switches.

1 Introduction

StarPlane is a research project funded by the Netherlands Organization for Scientific Research (NWO) and carried out at two Dutch Universities: the University of Amsterdam (UvA) and the Vrije Universiteit Amsterdam (VU).

* Corresponding author.
Email address: `grosso@science.uva.nl` (Paola Grosso).

The goals of the project is to build an *application-controlled photonic network*. The e-Science community relies nowadays on distributed collaborations with researchers interacting remotely with each other and exchanging large datasets. Such scientific applications require better performance, isolated environments and guaranteed scheduling.

The regular Internet is based on best-effort Layer3 IP routing; it has great flexibility but it is slow and unpredictable; on the other hand, dedicated *lightpaths* as the ones available in *lambda Grids*(1), offer good performance and good guarantees on the Quality of Service (QoS). Giving end users access to a dedicated connection has been implemented in many of the current research and education networks (NRENs), among them the Dutch NREN SURFnet. SURFnet6 deploys multiple fiber optic rings that connect academic and research institutes around The Netherlands. One of these rings connects the universities in Leiden, Delft and Amsterdam, the locations of the DAS-3 clusters. Into this ring eight wavelengths constitute the StarPlane lightpaths. The distinctive features of StarPlane in comparison with other similar initiatives are the accent on fast reconfiguration times, in the orders of seconds or subseconds, the use of photonic equipment in the core, and tight coupling between the user application and the networking services offered.

The rest of the article is organized as follows. Sec. 2 outlines network architecture of StarPlane; Sec. 3 describes the technical challenges related to the setup of the full end-to-end paths; Sec. 4 introduces the middlewares aspect and most specifically the management plane of StarPlane; Sec. 5 gives details on some of the most promising applications currently using StarPlane.

2 The StarPlane network

The Distributed ASCI SuperComputer (DAS) is a collection of five clusters located in four Dutch Universities: the UvA and the VU in Amsterdam, the University of Leiden and the Technical University in Delft. DAS is now in its third implementation and is called DAS-3. The 270 dual-CPU nodes are integrated into a single large-scale distributed system that provides a test-bed for Grid applications and distributed systems. In the following two sections we describe the network setup from the cluster to the photonics network and in the core of the SURFnet6.

2.1 From the cluster node to SURFnet6

Each of the five clusters has a head node and number of regular cluster nodes, from a minimum of 32 in Leiden to a maximum of 85 at the VU. All nodes have three ways to communicate: the *local*, the *inter-cluster* and the *wide-area* connection. Fig.1 shows a schematic representation of the network architecture used by DAS-3 nodes.

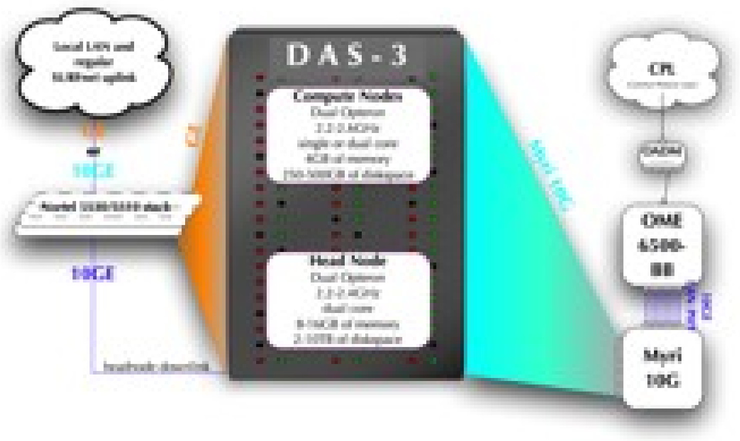

Figure 1. Network architecture in DAS-3

The local connections are Ethernet links to the University network, and from there to the regular Internet. The bandwidth is 1GbE for the cluster nodes and 10GbE for the head nodes. These links are used to move data between clusters in different locations when the application does not explicitly request the use of the faster and dedicated StarPlane links.
Cluster nodes in the same location communicate with each other using the high-bandwidth and low-latency interconnection via a Myrinet switch. The Myrinet protocol is particularly useful for parallel computing techniques based on MPI - Message Passing Interface.
Last, all cluster nodes have access to the photonic network of StarPlane. The Myrinet switches in all sites are equipped with an Ethernet card. 10 GE links go from this card to the OME (Optical Multi-service Edge) switch in the SURFnet domain. It is the OME that maps each of the Ethernet links into a wavelength of the photonic core.

2.2 CPL

CPL (Common Photonic Layer) is the Nortel WDM equipment used in SURFnet6 to deliver Adaptive All Optical Intelligent Networking solutions. In includes all components necessary to carry DWDM signals on optical fibers. SURFnet uses the optical multiplexers, optical amplifiers and the eROADM (Enhanced Reconfigurable Optical Add/Drop Multiplexer). The eROADM is a CPL functional module that enables dynamic, on the fly, optical branching to up to five different optical paths in addition to facilitating basic add/drop of individual wavelengths. Any wavelength (color) can be added or dropped on any port.

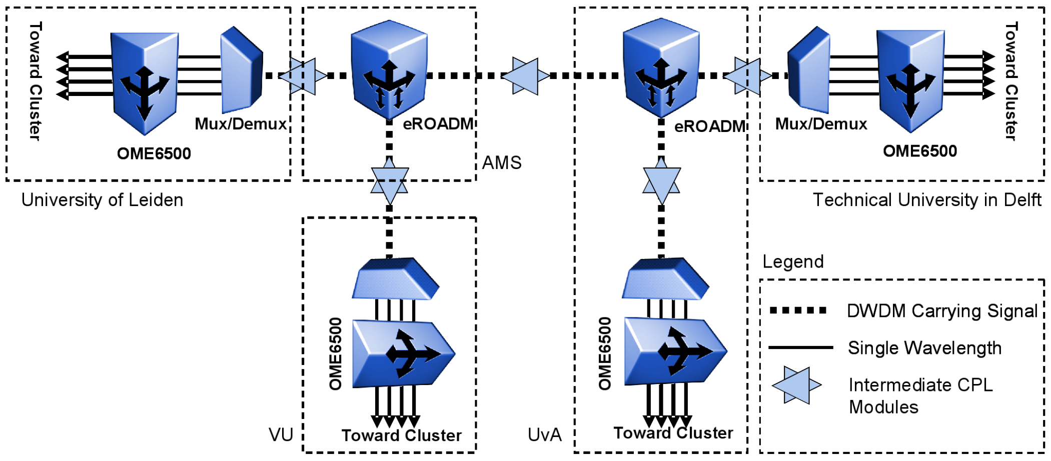

Figure 2. Overview of the Common Photonic Layer in StarPlane

In StarPlane there are two eROADM sites capable of switching the signals coming from 3 pair of fibers each. They allow to route wavelengths from and to any of the four DAS-3 locations. Fig. 2 shows the logical connectivity of photonic network implemented. The OME6500s at the DAS-3 sites translate the optical signal into a predetermined wavelength and insert it through the CPL multiplexer on the photonic network. The eROADMs in Amsterdam switch the wavelengths toward the final destination where the CPL de-multiplexer isolates the individual wavelength for delivery to the OME.

The eROADM are built out of WSS (Wavelength Selective Switch). The WSS can connect up to five different nodes at an eROADM site while supporting local add/drop traffic. These devices route wavelengths in and out of fibers: given a series of wavelengths coming into the device, they can send subsets of colors to different outputs. A WSS has incoming and outgoing fibers, and Micro Electro Mechanical mirrors, that tilt to reflect appropriate wavelengths toward the right output ports.

At the heart of CPL resides the Domain Optical Controller (DOC), a user-accessible end-to-end intelligent provisioning and management software that enables in-service insertion and deletion of wavelengths, all the while maintaining the active network channels in an optimized state. In addition, the DOC tracks and corrects for aging and very slow changes in operating conditions to ensure consistent system reliability and continual optimal performance.

In order to maintain the performance of all channels to maximum performance

DOC uses a multitude of monitoring points and actuators during wavelength insertion and optical switching. Due to the worst case engineering design for DOC the typical insertion and switching time for a wavelength could be in the order of 10 minutes. As a proof of concept for Starplane we will bypass the DOC and command the WSS directly in order to test the gain obtained through fast switching of the wavelengths through optical network.

3 StarPlane Engineering Challenges

In StarPlane there are two network layers to be manipulated to achieve path (re-)configuration times of the order of seconds or ultimately sub-seconds: Layer 0, for the CPL equipment in the core of the network, and Layer 2, from the SURFnet6 OME to the cluster nodes. Efficient configuration of both these layers has its own unique challenges. Let's define t as the total setup time of a StarPlane lightpath, bw_l the lighpath bandwith, bw_i as the bandwidth over the Internet cloud and d as the amount of data that have to be transmitted. The equality point, when both transmission paths are equivalent, is given by the following equation:

$$d = \frac{t * (bw_l * bw_i)}{bw_l - bw_i} \tag{1}$$

From Eq.1, the lower the setup time the smaller the amount of data needed to make a StarPlane path equivalent to the Internet path; with an amount of data larger than this minimum the dedicated path becomes the preferred choice for transmission. The setup time in StarPlane is the sum of the times needed in the core at Layer 0 and at the edges at Layer 2: $t = t_{L0} + t_{L2}$. The goal of StarPlane is to make the setup time t as small as possible.

All the wavelengths in the StarPlane color band are now operational. The lightpaths passing through the eROADM can be switched on demand using DRAC, the management tool from Nortel that we will describe in more detail in 4.1. The (re-)configuration time at Layer 0 (t_{L0}) is determined by the time that DRAC takes to moves one color from one destination to another. Currently this time is in the order of 10 minutes, as we describe in Sec. 2.2. As there are no easy solutions to decrease this setup time we are considering a caching approach, where the StarPlane management plane maintains all the lighpath pre-configured, also in absence of requesting applications. Subsequent requests for path will be served in few seconds instead of minutes as the paths are already available at Layer 0 and do not need to be configured on the fly.

As the photonic paths provide an isolated environment at Layer 0 we need to build a similar infrastructure at Layer 2, the highest network layer present in

StarPlane. This means providing a node-to-node isolated virtual network. As we described in Section 2, applications using the lightpaths first need to move data through the Ethernet card on the Myrinet switches. Unfortunately there is very little control on the behavior of these cards. To provide an isolated Layer 2 environment in the lowest setup time possible (smallest t_{L2}) we need both VLAN functionalities as well as the possibility to configure Spanning Tree Protocol (STP) priorities as to obtain the optimal Ethernet topology. We are currently looking at the adding extra Ethernet switches between the Myrinet and the OME to achieve this.

4 StarPlane middlewares

In StarPlane DRAC provides the photonic switching service, the StarPlane Management Plane (SPMP) builds new services used by the higher level middlewares and user-applications, and the Grid execution service, KOALA, is in charge of the overall orchestration.

4.1 DRAC

The primary goal of DRAC (Dynamic Resource Allocation Controller) is to enable a high degree of coupling between applications and networks, resulting in an improved application network experience, while optimizing equipment investments and operational expenses. Applications have the means to directly drive their share of network resources within a pre-defined authorization policy. Network resources include bandwidth, quality of service (QoS), security, and more.

DRAC provides a service interface with an abstracted view of the network. It enables applications to control the network, yet without requiring applications to interface directly with diverse and constantly evolving network protocols, features, and devices. The interface to applications is bi-directional, enabling network performance and availability information to be abstracted upwards towards the application.

In StarPlane DRAC was designed to control the CPL network and to allow for inter-layer routing from Layer 0 to Layer 2 of the network. When an application needs to reconfigure the underlying network, it reserves a point to point link from DRAC using a web service interface. DRAC then computes the availability of the connection, determines the best route for the signal and requests the connection from CPL using the Domain Optical Controller. The DOC in turn actions the eROADM to create the connection.

4.2 The StarPlane Management Plane: SPMP

The StarPlane Management Plane services are accessible through three sets of functions: one is for querying the service availability, one is to reserve the service and one to finally use it. The SPMP wraps around the photonic services provided by DRAC. Nevertheless as StarPlane relies on the public interfaces of DRAC, the added abstraction layer reduces its capability to take optimal decisions in terms of scheduling and resource allocation. This is circumvented by requesting all the colors in advance, the caching strategy we mentioned already in Sec. 3. The alternative approach to the "abstraction problem" would be to have DRAC offering a *bypass interface* with reduced abstraction.

To use StarPlane an application has to negotiate its requested services, and their quality, using the querying interface. The expressiveness of the query language is essential. Most of the query languages offer a set of hardcoded possible queries, hardcoded constraints and reduced and/or conjunctions between queries. We are exploring if advanced queries are desirable and suitable for StarPlane. One of the possible choices for extended expressiveness is to use logical programming (2) and semantic-based logical reasoning. A logical program contains *variables* connected by logical formulas (*constraints*). The program is said to be satisfiable if for all of its variables a value can be assigned that does not violate any of the constraints. The major difference with prior approaches is that users can implement their own complex functions and constraints. An analogy can be made with SQL, where a classical query looks like `select X, Y in .... where X.capacity >= 10 and Y.capacity <= 10.`, while logical programming is closer to the recent *recursive functions* extension of SQL (3). Once a query has been satisfied, a binding between the variables and the service has been computed and it becomes possible for the user to directly use the resource or to reserve it. The result of the reservation is a reservation ID, that allows at run-time to activate the use of the locked resources or to cancel the reservation.
Interactions between DRAC and the SPMP, as well as interactions between the higher level Grid managers and the SPMP are mediated via Web Services interfaces.

4.3 Grid execution service: KOALA

Each DAS-3 clusters provides a local resource scheduler, SGE (4). Jobs can also be submitted to the individual clusters by using Globus (5). In addition to the local schedulers, DAS-3 also offers the Koala grid scheduler (6; 7), which currently offers both processor and data co-allocation. Processor co-allocation can be used to run jobs that simultaneously require multiple execution sites.

As the local schedulers do not support processor reservation, the processor co-allocation can only provide a *best-effort* Quality of Service. If required, Koala can also transfer input files needed by the application to the appropriate sites. Since these files can potentially be large, Koala also performs data co-allocation, by taking into account the necessary time to transfer the files to the execution sites. When multiple potential execution sites are available, it will choose the one(s) which require the least time for the file transfers. Finally, Koala also offers fault-tolerance. When (part of) a job execution fails, the job will be aborted and can resubmitted automatically. The job submission will only fail if the submission attempts exceed a user-defined threshold.

In StarPlane we are planning several extensions to Koala to support network reservation. The first extension is to have Koala taking advantage of the dynamic optical network when performing data co-allocation. Instead of simply estimating file transfers times using the internet, the available lightpaths and their setup times can also be taken into account when determining the optimal strategy. The outcome of this heuristic may vary substantially depending on the photonic setup time that can be achieved (see Eq. 1). Even in the current state, where setup times are of the order of 10 minutes, transfer of very large input files using lightpaths instead of the regular Internet is already beneficial.

A second extension to Koala would allow the user applications to allocate network bandwidth as part of their job. Currently, to submit a job, the user provides an RSL (5) description of the job to Koala. This job description format can easily be extended with an additional field which describes the required network bandwidth between the job components. An example of this extended description is given in Appendix A.1. By using this information, Koala can then request a lighpath between the execution sites. By requesting lightpath availability information from the StarPlane Management Plane and processor availability information from the local job schedulers, Koala can estimate at what time both the requested resources are likely to be available.

4.4 Other Grid services

In StarPlane we are also experimenting with the SmartSocket and at the Enhanced Socket API communication libraries and with the JavaGAT.

SmartSockets (8) is a Java library that simplifies creating socket connections in environments where the connectivity is severely limited by firewalls, network address translation (NAT) or non-routed networks. The primary focus of SmartSockets is on ease of use. It automatically discovers a wide range of connectivity problems and attempts to solve them with little or no support from the user, using solutions such as port forwarding, TCP splicing and SSH tunneling. In addition, SmartSockets can also automatically select the most appropriate network interface on multi homed machines (which have multiple network addresses). As a result, applications using SmartSockets automatically use the StarPlane network once the photonic network is configured. No changes to the application are required.

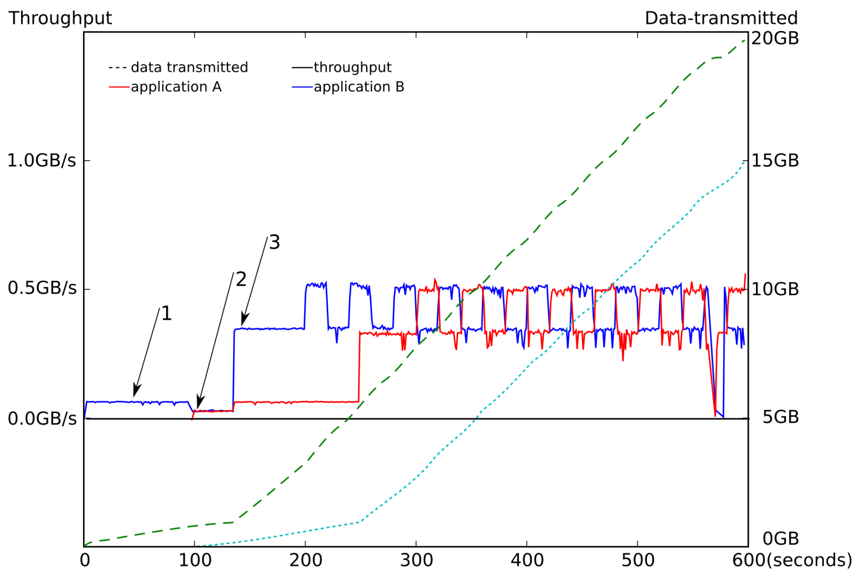

Figure 3. Benchmarking of two client-server applications executed on two clusters. In(1) only one traffic flow through the ethernet network, in (2) the second application is started impacting the first one. In (3) the first application receives a lighpath and redirect the traffic to it; in (4) the same happens to the second application

Implemented in C, the Enhanced Socket API is an experimental library that implements low level interaction between the communication library itself and StarPlane. The library can request lighpaths to SPMP if the two endpoints are on different clusters. This library mimics the BSD-socket API and provides a hijacker-wrapper based on dynamic loading (LD_PRELOAD) that allows to use the new functions without requiring any modification or recompilation of the user applications. Fig. 3 is a capture of a the run of two client-server applications that have been hijacked. The client and server are on two different cluster sites. It can be seen at the startup that the regular Ethernet network is used (point 1 and 2 in the drawing). The throughput for the two applications is 110 MB/s (point 2). Once a lighpath is available (point 3 and 4), the library redirects the traffic to the myrinet interface and StarPlane. When lighpaths are used the throughput per application is around 400MB/s (800MB/s in total).

The JavaGAT (9) project can also be extended to take advantage of the dynamic lighpath of StarPlane. The JavaGAT is a middleware abstraction layer. It provides a middleware-independent API, and contains robust mechanisms to automatically map these API calls onto whatever middleware is available at runtime. As a result, grid application programmers only need to learn a single API to obtain access to the entire grid. JavaGAT has a modular design that allows it to easily be extended with support for other grid middleware layers. Therefore, by adding new modules to the JavaGAT that interface with the StarPlane Management Plane, the new dynamic networking services can be used by the user-applications based on the JavaGAT API. Although some applications may want to use the dynamic networking functionality explicity, many other applications may also profit from StarPlane. By extending the existing JavaGAT modules to make use of the dynamic network whenever it is appropriate, the performance of many JavaGAT operations (such as site to site file transfers) could be significantly improved. Many JavaGAT based applications can benefit from these improvements without the need to change a single line of application code.

5 Applications on StarPlane

We will now take a closer look at two applications, Awari and SCARIe, currently using StarPlane.

Awari is an application that solves an ancient two-player board game using retrograde analysis. This board game is played using 48 stones divided over 12 pits (traditionally, holes in the ground). Players can move and capture stones according to some rules (10). Although Awari is specifically designed to solve a game, its behavior is very similar to model checking and verification applications (11). These applications, like Awari, focus on searching a state space which can be very large. The state space searched in Awari contains approximately 900 billion board positions. The only way to handle such a large state space is by distributing it over multiple machines. This distribution, however, also results in a extremely high communication volume. On a single cluster, running on 144 nodes, Awari required a sustained total bandwidth of almost 29 Gbits/s for a large portion of the runtime. Awari is implemented using asynchronous communication. Although this does not reduce the communication volume in any way, it does make the application latency insensitive, thereby improving its performance. Earlier work at the VU (10) showed that it was possible to completely solve the game of Awari on a single cluster. Althought the insensitivity to latency should, in theory, also make Awari suitable for multi cluster runs, the high bandwidth requirements could not be fulfilled by the regular internet links. StarPlane, however, is capable of meeting the requirements.
Recent work (12) showed that with the combination of DAS-3 and StarPlane it was indeed possible to run Awari efficiently using three DAS-3 sites. Although the high bandwidth provided by StarPlane was essential for this success, some additional optimizations had to be applied to the application to reduce the send overhead per message, to improve the performance of global synchroniza-

tion, and to reduce the effect of flow-control problems caused by the higher latency of the wide-area links. Ultimately, running Awari on three DAS-3 sites using StarPlane only caused a 15% performance drop compared to the single cluster version.

The SCARIe project is a cooperation between the JIVE, the UvA and SARA that focuses on providing a software correlator for Very Long Base Interferometry, VLBI. VLBI is a technique in astronomy in which several distant radio telescopes observe the same object simultaneously. By using VLBI, astronomers can make detailed images of cosmic radio sources with unsurpassed resolution. To process the final picture all the data need to be at the same location. In recent years networks have taken the place of couriers in delivering the data from the telescope to the correlation centers. This approach is called e-VLBI. An e-VLBI experiment may need to transmit data to a centralized computation center from up to 16 radio-telescopes, each gathering data at 1 Gbps and located all around the earth. Currently processing is done by a dedicated hardware performing a signal processing operation called correlation. Hardware correlators are highly efficient while pure software correlators have the advantage of greater flexibility, rapid prototyping and testing of new VLBI operational modes. SCARIe is a very good example of an application that needs high networking performance, an isolated environment for real-time operation as well as scheduling for synchronization with radio telescopes. The correlation task is following a *hierarchical master-worker* model. We have successfully implemented it on top of MPI and the SATIN programming model (13). In our experiments Starplane delivers 400MB/s of throughput between the cluster sites and allows us to distribute medium correlation jobs to the DAS-3 Grid. It is very important for the SCARIe project to have guaranteed quality of service on nodes and network performances. For this reason we are working closely with the SCARIe team in charge of the on-demand virtual-network (traffic isolation).

6 Future work and conclusions

The preliminary results we obtained from monitoring tools and user tests using the Enhanced Socket API (see Sec. 4.4) show unexplained behaviors of the throughput in the StarPlane network. Our first goal is to investigate the origin of the oscillations visible in Fig. 3. We suspect this is due to unexpected interactions between cluster nodes, the Myrinet switches and the OMEs.

In Sec. 4.2 we described our current work on the use of logical programming (queries) for the interaction with the SPMP. To solve the logical queries it is necessary to have a detailed knowledge base of the underlying infrastructure and resource's status. One possibility we will be investigating is the use of Semantic Web-based models for such descriptions. Starting from our experience with the RDF-based Network Description Language (NDL) (14) we will apply this approach to StarPlane. Semantic reasoning offers a good level of flexibility and enough detail of information to make decisions over Grid services, and specifically dynamic network services.

The StarPlane project researches novel ways to offer users higher control of the network and its services. It achieves this by allowing applications running on the DAS3 to control lightpaths in a portion of SURFnet6. The development of the StarPlane Management Plane allows the integration of Grid middleware on one side and the network components on the other. Currently switching times are in the order of minutes: this makes StarPlane suitable only for long applications. Integration of StarPlane with some Grid services running the DAS-3, such as KOALA and JavaGAT, is underway. Two selected scientific applications, AWARI and SCARIe, are experimenting with all the new StarPlane services. Future work will focus on reducing the switching time, as this opens up StarPlane also to small, short-lived applications, and on the traffic engineering techniques to provide isolated virtual environments. We investigating the use of logics and Semantic-Web models for the description of the network infrastructure and the interactions between applications and Grid services.

Acknowledgments

We wish to thank SURFnet for their continuous support, in particular Roeland Nuijts, Bram Peeters and Antoon Prins. We also thank Nico Kruithof and the rest of the SCARIe project. The StarPlane research is supported by SURFnet, the NWO grant 643.000.504 and the BSIK program GigaPort.

References

[1] C. de Laat, P. Grosso, Lambda grid developments, history - present - future, in: P. Clarke, C. Davenhall, C. Greenwood, M. Strong (Eds.), Lighting the Blue Touchpaper for UK e-Science: Closing Conference of ESLEA Project, National e-Science Centre, 2007.
[2] P. Blackburn, J. Bos, K. Striegnitz, Learn Prolog Now!, Vol. 7 of Texts in Computing, College Publications, 2006.
[3] J. Melton, A. Simon, SQL:1999: understanding relational language components, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
[4] W. Gentzsch, Sun grid engine: Towards creating a compute power grid,

in: CCGrid, 1st International Symposium on Cluster Computing and the Grid, 2001.
[5] The globus toolkit, http://www.globus.org.
[6] H. Mohamed, D. Epema, The design and implementation of the koala co-allocating grid scheduler, in: European Grid Conference, 2005.
[7] H. Mohamed, D. Epema, Experiences with the koala co-allocating scheduler in multiclusters, in: Proceeding of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005), 2005.
[8] J. Maassen, H. E. Bal, Solving the Connectivity Problems in Grid Computing, in: Proceedings of The 16th IEEE International Symposium on High-Performance Distributed Computing (HPDC), Monterey, CA, USA, 2007.
[9] R. V. van Nieuwpoort, T. Kielmann, H. E. Bal, User-friendly and reliable grid computing based on imperfect middleware, in: Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07), 2007.
[10] J. W. Romein, H. E. Bal, Solving awari with parallel retrograde analysis, IEEE Computer 36 (10) (2003) 26–33.
[11] J. Barnat, L. Brim, I. Cerna, P. Moravec, P. Rockai, P. Simecek, Divine - a tool for distributed verification, in: Proceeding of 18th International Conference on Computer Aided Verification (CAV 2006), 2006.
[12] K. Verstoep, J. Maassen, H. E. Bal, J. W. Romein, Experiences with fine-grained distributed supercomputing on a 10g testbed, in: Accepted for publication in the Proceeding of the 8th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2008), 2008.
[13] R. V. van Nieuwpoort, J. Maassen, T. Kielmann, H. E. Bal, Satin: Simple and efficient Java-based grid programming, Scalable Computing: Practice and Experience 6 (3) (2005) 19–32.
[14] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, C. de Laat, Using the network description language in optical networks, in: Tenth IFIP/IEEE Symposium on Integrated Network Management, 2007.

A Appendix: Resource querying

A.1 KOALA's scheduler: Example of an bandwidth aware RSL query

In the following example, a job description is given of a 30 minute job consisting of three seperate components. The first two components require 50 processors each, and must be run on the VU and UvA sites of the DAS-3 (as indicated by the "resourceManagerContact" option). The third component requires 25 processors and may be placed anywhere on the DAS-3.

```
+(&( executable = "awari" )
    ( label = "subjob 0" )
    ( count = "50" )
    ( resourceManagerContact = "fs0.das3.cs.vu.nl" )
    ( bandwidth = "subjob 1:20G" "subjob 2:10G"))

(&( executable = "awari" )
    ( label = "subjob 1" )
    ( count = "50" )
    ( resourceManagerContact = "fs2.das3.science.uva.nl" )
    ( bandwidth = "subjob 0:20G" "subjob 2:10G"))

(&( executable = "awari" )
    ( label = "subjob 2" )
    ( count = "25" )
    ( bandwidth = "subjob 0:10G" "subjob 1:10G"))
```