Lambda Station: Alternate network path forwarding for production SciDAC applications

Maxim Grigoriev[1], Andrey Bobyshev[1], Matt Crawford[1], Phil DeMar[1], Vyto Grigaliunas[1], Alexander Moibenko[1], Don Petravick[1], Harvey Newman[2], Conrad Steenberg[2], Michael Thomas[2]

[1]Fermilab, PO BOX 500, Batavia, IL 60510, USA

[2]Caltech, 1200 East California Boulevard, Pasadena, CA 91125, USA

maxim@fnal.gov

Abstract. The LHC era will start very soon, creating immense data volumes capable of demanding allocation of an entire network circuit for task-driven applications. Circuit-based alternate network paths are one solution to meeting the LHC high bandwidth network requirements. The Lambda Station project is aimed at addressing growing requirements for dynamic allocation of alternate network paths. Lambda Station facilitates the rerouting of designated traffic through site LAN infrastructure onto so-called "high-impact" wide-area networks. The prototype Lambda Station developed with Service Oriented Architecture (SOA) approach in mind will be presented. Lambda Station has been successfully integrated into the production version of the Storage Resource Manager (SRM), and deployed at US CMS Tier1 center at Fermilab, as well as at US-CMS Tier-2 site at Caltech. This paper will discuss experiences using the prototype system with production SciDAC applications for data movement between Fermilab and Caltech. The architecture and design principles of the production version Lambda Station software, currently being implemented as Java based webservices, will also be presented in this paper.

1. Introduction

1.1. Motivations

The upcoming era of LHC high data volume distributed GRID computing implies extra requirements on resource management applications. There will be a large demand for network aware applications capable of reserving dedicated high performance network circuits. Correspondingly, there will also be a need for services capable of dynamic allocation of such high performance network paths. Lambda Station is a service that addresses the need for dynamic reconfiguration of the local network infrastructure to enable use of high bandwidth, alternate network paths by specific applications for designated traffic flows.

1.2. Basic terms

We assume that reader is familiar with networking terminology and further in the text we will refer to the Policy Based Routing as **PBR**. The **PBR Client** is the system or cluster and applications running on it and sourcing traffic flows that can be subject for Policy Based Routing. Also, the **Flow** is a stream of packets with some attributes in common, such as endpoints IP addresses (or some range of addresses), protocols, protocol's port (or range of ports) if applicable and differentiated services code point (**DSCP**). The Lambda Station (***λS***) is a host with special software, referred as ***λS*** API and designed to control traffic path across Local Area Network on demand of applications and interfacing Wide Area Network. The **ticket** is a placed alternative network path reservation on the Lambda Station from authenticated and authorized client, which goes through several states and has some finite time duration.

1.3. Lambda Station project

The Lambda Station project [1] was started about 3 years ago by Wide Area Networking group at Fermilab and network researchers from Caltech. The original goal of the project was to design a secure network service which will be able to configure alternate network paths between local production computing resources and advanced high performance network paths made available for specific data intensive applications. More details on the progress of the project can be found in [2] and [3]. In summary, Lambda Station is targeting on dynamic, per flow, alternate network path selection with graceful cutover and fallback. Among our goals were providing network awareness plugins for commonly used networked applications, and researching the behavior of network aware applications with flow-based network path forwarding. Last, but not least, a major goal was to incorporate the ***λS*** API within SRM [4], the production storage resource management software used within the CMS Tier-1 facility. This presentation will cover basic building blocks of the Lambda Station architecture, major interface calls, developed and deployed Perl based API, developed Java API and integration with SRM production environment at Fermilab US CMS [5] Tier1 site and Caltech Tier2 site.

2. Lambda Station API

2.1. Building blocks

There are several services working together on Lambda Station. The major one is ***λS Controller***. It manages persistence in the form of the active tickets queue, processes all tickets, and starts related services. These services are implemented as Java threads, where there is a thread for every incoming or out-coming ***λS request,*** and a single thread for ***λS-λS service*** dedicated to network topology, **PBR** clients definition propagation, and ***λS*** service parameters discovery. For example the simplest reservation lifecycle can be expressed as:

- reservation request is received on the ***λS request interface*** from the client
- authorization verified for the service request
- new open ticket is added to the tickets queue
- ***λS Controller*** forwards reservation request to remote ***λS*** for reciprocal action
- ***λS Controller*** sends request to **Network Configurator** to configure network devices about 5 minutes before the reservation activation time
- ***λS Controller*** sends another configuration request to **Network Configurator** at ticket expiration time to restore original configuration

Figure 1 illustrates how different ***λS*** services communicate with each other.

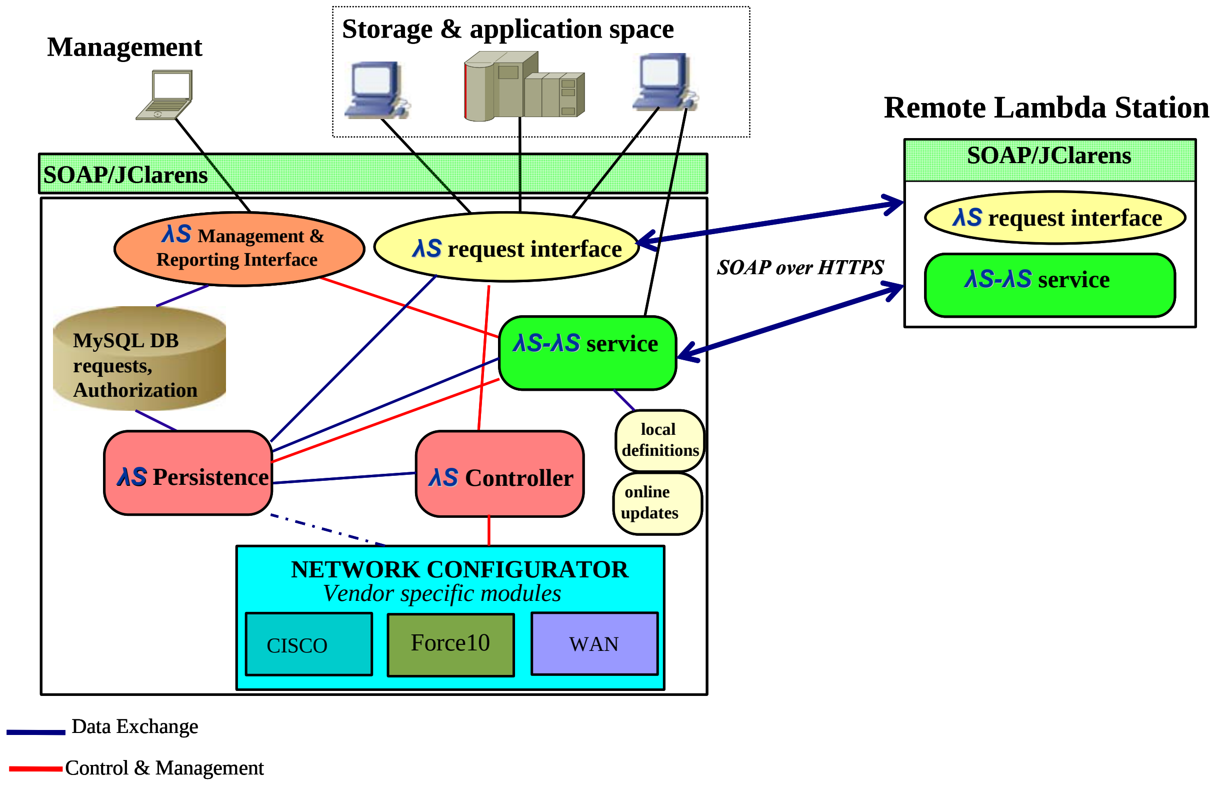

Figure 1 Lambda Station building blocks

2.2. Software layers

The design of the Lambda Station was chosen to be based on Service Oriented Approach principals. Therefore, every Lambda Station presents complete and independent set of webservices complied with WS-I Version 1.1 WebServices Interoperability Profile. This provided platform flexibility for Lambda Station clients and network aware applications by encapsulating any complexity of implementation within the simple, publicly available webservices interface described by the WSDL file. After implementing prototype ***λS*** server in Perl and testing functionality of the ***λS*** services on the WAN test-bed, we have started implementing the production ***λS*** server in Java. This idea was supported by portability of Java SDK, and the presence of a large variety of open source projects targeted on delivering easy to use webservices frameworks. The need for production quality solution led us to select the Tomcat [9] webserver with Apache's Axis [10] webservices framework. The Axis framework allowed us to utilize WSDL2Java converter, and gave us opportunity to build dynamically interface description classes around WSDL interface. The same Metadata-to-JavaObjects approach was used for building XML binding classes.

The JClarens [11] webservices toolkit was added to bring such features as reusable database connections pool, security certificates loading, user's session management, basic service configuration management, and background threads control. Another task was to provide support for our XML configuration files, and add validation into the process of unmarshalling of the incoming XML fragments inside of SOAP envelopes. After functionality tests of the several XML binding APIs, we decided to use Apache's XMLBeans [12] as the most mature product, as well as the one with most complete support for XML schema.

On the backend of the ***λS,*** we employed MySQL DB as universal storage for the JClarens internals and ***λS*** operational data. The schematic view of the interactions between different layers of the ***λS*** software can be seen in Figure 2.

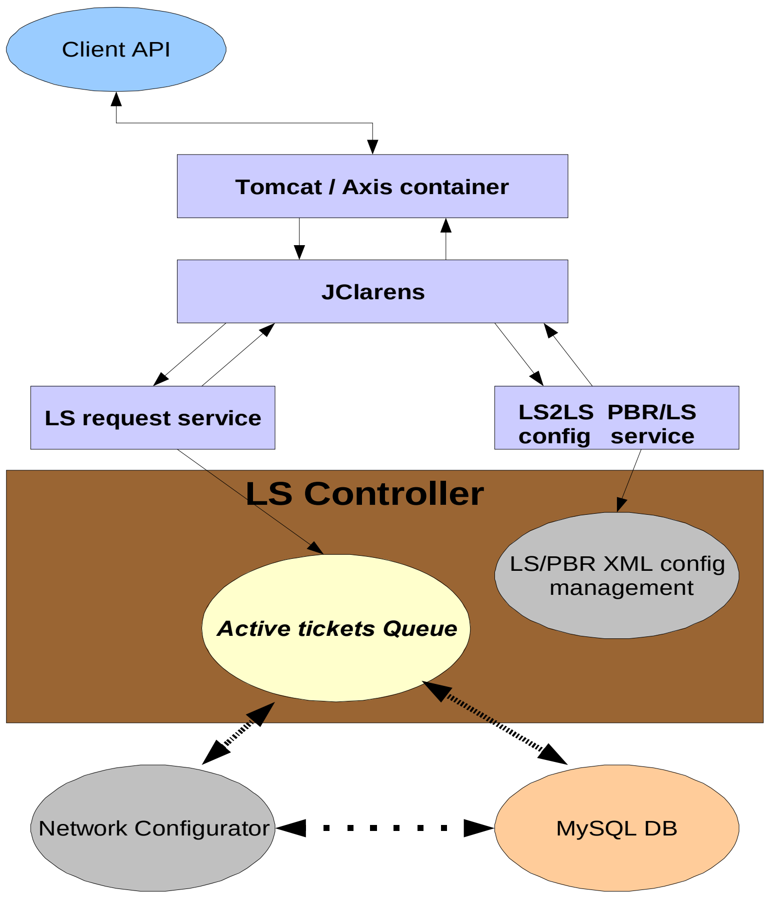

Figure 2. Interactions between different layers of the ***λS*** API

2.3. Network Configurator service

The **Network Configurator** service stands a little bit aside from other API, since it has been implemented only in Perl. The purpose of this service is to modify dynamic configurations of the local network devices. Therefore, this module is very vendor specific. In case of Fermilab's ***λS*** service, it is customized to work on **Cisco**™ routers and switches. Lambda Station may relay calls for **Network Configurator** through a separate web services interface. Alternately, this service may connect to the ***λS*** backend database to check the list of tickets that require network reconfiguration. In case of **Cisco**™ routers, the IOS version must support sequencing type of the named ACLs. Also, the interface on which PBR is applied needs to be configured with ***"ip policy route-map"***, route map needs to be configured as ordered list of match/action statements, and match criteria need to be associated with ACLs.

2.4. Basic interface calls

There are several calls which client needs to know in order to negotiate alternative path reservation with ***λS.***

- openSvcTicket
    - Major ***λS*** operational request, places alternative path reservation ("**ticket**")
    - Accepts **svcTicket** element as an argument, validated by XML schema
    - Returns udpated **svcTicket** XML element with **ticket ID**
- updateFlowSpecs
    - updates flow specification for the "**ticket**"
    - Accepts **svcTicket** XML element as an argument, validated by XML schema
    - Returns **true** or **false**
- getTicket
    - get **svcTicket** XML element with full information about placed "**ticket**"
    - Accepts **"ticket"** ID
    - Returns **svcTicket** XML element
- cancelTicket
    - cancel existed "**ticket**", ticket will be closed and network topology will be changed
    - back to production path
    - Accepts "**ticket**" ID
    - Returns **true** or **false**

2.5. "ticket" operational modes

Three operational modes should be considered for every openSvcTicket call. All modes are subject to TLS/SSL-based and rules-based authorization.

- **new** ticket
    - create a new "**ticket**"
    - client must be authorized for local ***λS*** and station must be authorized for remote ***λS***
- **join** ticket
    - join already active "**ticket**" (in case of multiple requests for the same flow)
    - existing "**ticket**" parameters will be reused
- **extend** ticket
    - extend already active "**ticket**"
    - **endtime** will be extended

3. Utilizing Lambda Station with SRM production environment

3.1. SRM production environment

At the present time, a Storage Resource Manager (SRM) at US CMS Tier1 in Fermilab controls more than 100 read/write nodes, about 1PB (PetaByte) of tape-backed pools, and more than 100TB in the resilient storage, spread across 650 worker nodes. The Tier2 facility at Caltech has 75 worker nodes with about 55TB in the resilient storage. The most recent observed SRM data traffic between Fermilab and Caltech is shown on the Figure 3.

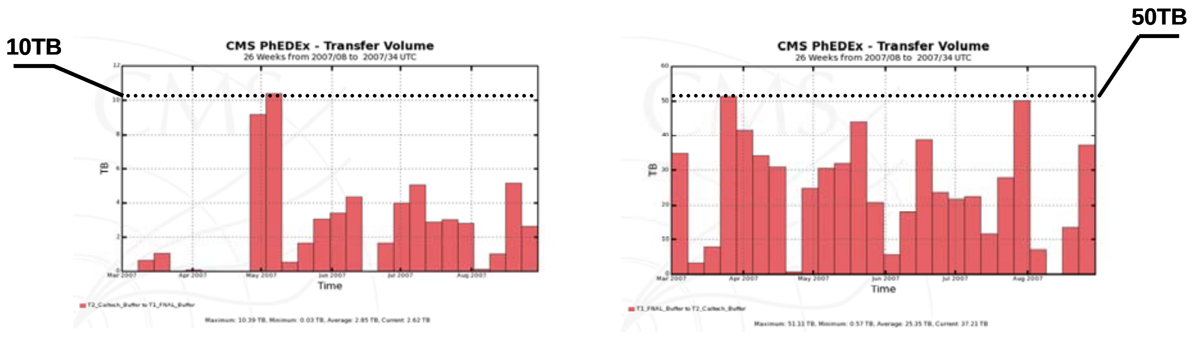

Figure 3. Traffic volume between Fermilab and Caltech, observed by PhEDEx

3.2. ***λS*** awareness in SRM

The ***λS*** awareness was introduced by injecting into the SRM API several CLI calls to the client ***λS*** API placed on the same host. On the Figure 4 there is a schematic view of the interactions between ***λS*** services and SRM.

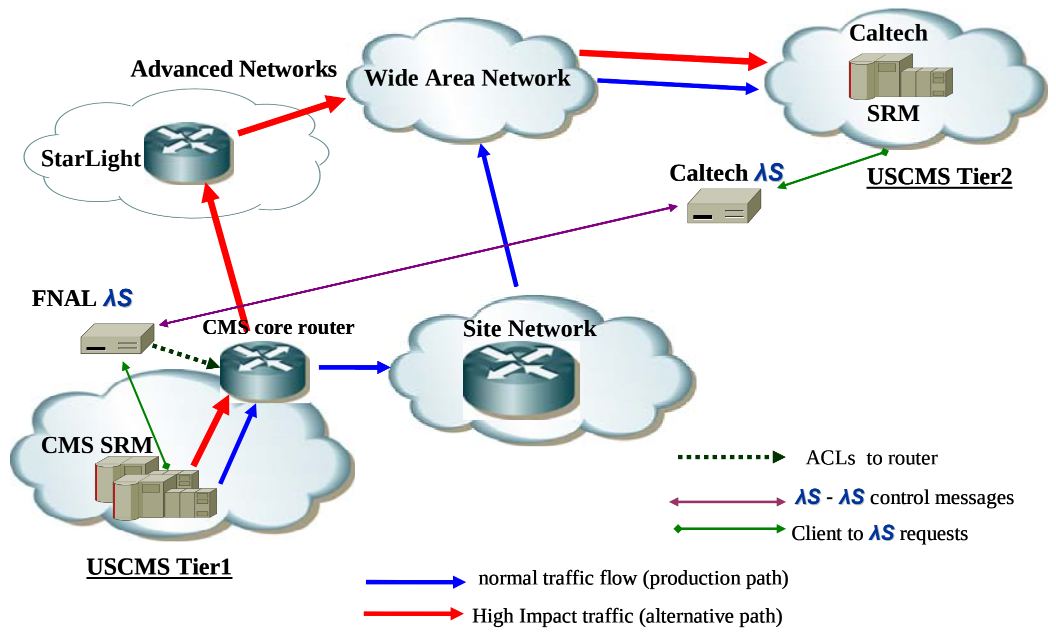

Figure 4. ***λS*** awareness in the SRM

Production US CMS **SRM** server sends request to ***λS*** to send Fermilab CMS Tier-1/Caltech Tier-2 traffic across the **Advanced Network** infrastructure. If ***λS*** service is available, the designated traffic gets re-routed through the alternative path. Currently, there were about 500 ***λS*** requests per day from the Tier-1 SRM. All requests are processed automatically, without intervention from the user.

4. Project accomplishments

The major milestones for the project to date are:

- Development and release of the Perl-based API version 1.0 (a fully functional prototype supporting whole cycle of ***λS*** functionality)
- Deployment of a high bandwidth, alternate network path between Fermilab and Caltech and successfully utilization of that path with ***λSs*** installed at Fermilab and Caltech.
- ***λS*** awareness build into well known applications, Iperf (lsiperf) and traceroute (lsTraceroute)
- Version 1.0 of the API integrated into the production release v.1.7.0 of the SRM.
- The ***λS***-aware production SRM running at Fermilab's US CMS Tier1 and Caltech Tier2 sites.
- Interoperable Java implementation of the ***λS***'s major components developed, with support for Perl and Java clients.

5. Future development

The main goal has been to release production quality ***λS*** Java API and integrate it with production SRM at Fermilab's US CMS Tier1 site, and at participating CMS Tier2 sites where there is a need for alternative path forwarding. In the past several months, another goal has emerged. There are several projects targeted on dynamic manipulation of network paths. Some of them are oriented towards Wide Area Networks (WANs), and are carried by research network providers. The OSCARS [13] project was started by ESnet two years ago, and delivers APIs capable of on-demand creation of end-to-end circuits, based on MPLS technology. The DRAGON [14] project has a goal to deliver software capable of Dynamic Resource Allocation via GMPLS Optical Networks. Finally, the TeraPaths [15] project, based at Brookhaven National Lab, is designed to provide end-to-end Virtual Network Paths with QoS Guarantees.

All of these projects operate with almost identical pieces of networking information, such as "flows", and all are trying to design their software as set of the secure independent web services. Optimally, OSCARS would interoperate with Lambda Station and TeraPaths, in order to achieve end-to-end reservation guarantee. However, there was no effort made initially to synchronize development activities of the various projects. Our new goal is to develop a common interface standard for these projects to promote interoperability. Lambda Station and the other projects need to collaborate on uniform messaging protocol in order to orchestrate their webservices into the truly versatile enterprise bus defined by Service Oriented Architecture. We need to formalize authentication and authorization mechanisms and describe set of XML schemas for common interface calls and common data objects. This task will take some effort in the beginning but then will allow easily accommodate any new control plane networking API. We are looking forward to present some level of integration in the nearest future and more information can be found on the collaborative Wiki page [16].

6. References

[1] Lambda Station project webpage, http://www.lambdastation.org

[2] P. Demar et al., "LambdaStation: A forwarding and admission control service to interface production network facilities with advanced research network paths", Proceedings of CHEP2004, Interlaken, Switzerland, September 2004

[3] A. Bobyshev et al., "Lambda Station: Production Applications Exploiting Advanced Networks in Data Intensive High Energy Physics", Proceedings of CHEP2006, Mumbai, India, September 2006

[4] Fermilab SRM Wiki page, https://srm.fnal.gov/twiki/bin/view/SrmProject/WebHome

[5] US CMS, http://www.uscms.org

[6] WS-I Basic Profile Version 1.1, Final Material, 2006-04-10
[7] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl
[8] gLite project, http://glite.web.cern.ch/glite/
[9] Apache Tomcat project, http://tomcat.apache.org
[10] Apache Axis project, http://ws.apache.org/axis/
[11] JClarens project, http://clarens.sourceforge.net/jclarens/
[12] Apache XMLBeans project, http://xmlbeans.apache.org
[13] ESnet OSCARS project, http://es.net/oscars
[14] DRAGON project, http://dragon.east.isi.edu/twiki/bin/view/Main/WebHome
[15] Terapaths project, https://www.racf.bnl.gov/terapaths
[16] LambdaStation-TeraPaths-OSCARS integration Wiki page, https://wiki.internet2.edu/confluence/display/CPD/LambdaStation+and+TeraPaths