SOFTWARE-DEFINED MEASUREMENT TO SUPPORT PROGRAMMABLE NETWORKING FOR SOYKB

A Thesis

Presented to

The Faculty of the Graduate School

At the University of Missouri-Columbia

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Longhai Cui

Dr. Prasad Calyam, Advisor

DECEMBER 2015

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

SOFTWARE-DEFINED MEASUREMENT TO SUPPORT

PROGRAMMABLE NETWORKING FOR SOYKB

Presented by Longhai Cui,

A candidate for the degree of

Master of Science

And hereby certify that, in their opinion, it is worthy of acceptance.

_____

Professor Prasad Calyam

_____

Professor Trupti Joshi

_____

Professor Marjorie Skubic

ACKNOWLEDGEMENTS

I am deeply grateful to Dr. Prasad Calyam for his guidance and support. Next, I would like to thank Ronny Bazan Antequera, Yuanxun Zhang, Saptarshi Debroy, Matthew Dickinson, and Sripriya for their critical contributions to this research project.

Special thanks to Tony Zhu and Alex Berryman for guiding me step by step and sharing a lot of knowledge regarding network performance monitoring and software-defined networking.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

Campuses are increasingly adopting hybrid cloud architectures for supporting big data science applications that require "on-demand" resources, which are not always available locally on-site. Policies at the campus edge for handling multiple such applications competing for remote resources can cause bottlenecks across applications. To proactively avoid such bottlenecks, we investigate the benefits in the integration of two complementary technology paradigms of software-defined measurement and programmable networking. The integration inherently allows flexible end-to-end application performance monitoring and dynamic control of big data application flows using: (a) software-defined networking for transit selection to remote sites, and (b) pertinent selection of local or remote compute resources. Using the Soybean Knowledge Base (SoyKB) as an exemplar application, we demonstrate the benefits of software-defined measurement to support programmable networking.

As part of our study methodology, we first profiled the original data flows within SoyKB's use of iPlant public cloud resources, and identified bottleneck cases such as slow data transfer speeds, lack of performance information (e.g., such as cluster availability, job status and network health) and inflexible control of hybrid cloud resources to address application-specific needs. The profiling study motivated us to propose a new hybrid cloud architecture for SoyKB workflows that utilize end-to-end performance measurements to support a cost-optimized selection of sites for computation and effective traffic engineering at the campus-edge. We validate our approach for a SoyKB workflow use case that we setup on a wide-area overlay network testbed implementation across two geographically distributed campuses. Our performance results show a notable performance improvement in SoyKB remote data transfer flows that utilize iRODS and TCP tuning mechanisms in the presence of cross-traffic big data flows. Additionally,

we implement a SoyKB system that provides: (i) flexible workflow performance analytics at a glance to SoyKB researchers handling big data, and (ii) web service mechanisms for interfacing with popular dynamic resource management technologies such as OpenStack, HTCondor and Pegasus.

1. INTRODUCTION

1.1 Problem

Data intensive and big data applications in research fields such as bioinformatics, climate modeling, particle physics and genomics generate vast amounts of data that need to be processed with real-time analysis. The general data processing facilities and specialized compute resources do not always reside at the data generation sites on campus, and data is frequently transferred in real-time to geographically distributed sites (e.g., remote instrumentation site, federated data repository, public cloud) over wide-area networks. Moreover, researchers share workflows of their data intensive applications with remote collaborations for multi-disciplinary initiatives on multi-domain networks [1] leading to a shift in design of advanced network architectures [2].

Frequently, the Science DMZ (de-militarized zones) [1] has to compete with traditional general-purpose network (i.e., Layer 3/IP network) resource usage. With multiple applications accessing hybrid cloud resources and coupled with traditional campus traffic network, the result can be a significant amount of "friction" for science Big Data movement and can easily lead to performance bottlenecks. There is a need to provide dynamic network management of Science DMZ network resources instead of setting a static rate limit affecting all applications.

1.2 Domain of Interest

The research project is involved with many research individuals (including researchers from different departments and system administrators from different organization) and the over-all objective of this

research is to provide effective network provisioning solutions to meet different needs of data-intensive applications in an federated resource environment, while marking network programmability related issues a non-factor for the users. The author's contributions include: 1) Implementing essential scheduler functions of our network performance monitoring system, 2) Installing and maintaining necessary application/ monitoring software in the end-point servers to provide performance visibility, 3) Utilizing performance knowledge to support dynamic transit selection of data intensive applications, especially for soybean knowledge base (SoyKB) system and end-to-end data transfer optimization for SoyKB system.

There are two stages in this network management optimization – measurement and control [3]. In the measurement stage, we utilize our software-defined measurement platform to provide the network middleware service with the on-demand and complete awareness of the network performance based on the user requirements such as the where (servers, data transfer nodes, routers and switches etc.), what (bandwidth, delay and route-trace etc.) and when to measure securely. In the control stage, we utilize software-defined networking [4] technologies and adjust the network accordingly by changing the rules in control plane with the decision of the network middleware service. While the control stage have been put on a lot of effort and become extremely successful, there are not many studies on measurement especially about on how to utilize those measurements in a way to support adaptive control of network.

The remaining part of the paper is organized as follows. In chapter 2, we introduce the background knowledge of SDN and SDM for those who are not familiar with corresponding concepts. In chapter 3, describe the ADON architecture and its core components that we are referring as our blueprint and using as guideline. Chapter 4 shows ADON implementation on SoyKB. We detail the experiments and validate our optimization in Chapter 5. Chapter 6 concludes the paper.

2. BACKGROUND

This chapter provides the background knowledge of software-defined networking and software-defined measurement, which is essential to at least have basic concepts and idea to understand the remaining of the paper. Readers who are familiar with software-defined networking concepts and software-defined monitoring systems should feel free to skip this chapter.

2.1 Software-Defined Networking

In contrast to the traditional distributed network, where network operators can only get limited network visibility through network devices (such as switches, routers and data transfer nodes) and have to deploy the network manually in these end-point devices, software-defined networking is a new approach to computer networking that provides centralized control and network programmability. The key ideas are 1) to logically divide the network systems into control plane, which makes decisions about where to send the traffic and data plane which does nothing but simply forward the traffic, 2) to provide network programmability for the network engineers to deploy and change networks easily.

2.1.1 Need for Programmable Networks

As more and more applications trend to across different databases and geographically distributed servers nowadays, server-to-server communication generates way more traffic than user-server communication. The rise of cloud servers and big data science also put huge pressure to the conventional network architecture for a fine-grained control of server-to-server networking with ease of use and increased

security. Software-defined networking provides such solutions allowing organizations to replace a manual interface into networking equipment with a programmatic interface that can enable the automation of tasks (such as configuration and policy management) and can also enable the network to dynamically respond to application requirements.

2.1.2 OpenFlow

OpenFlow [4] is considered as the best SDN practice that provides open standard interfaces for the network vendors to follow. In the control plane, a remote controller is functioning as a brain, which can make intelligent decisions based on, for example, network performance and send commands (flow entries) to the OpenFlow Switches. In data plane, it utilizes the OpenFlow switches. Each OpenFlow switch has the following components 1) A Flow Table, that store flow entries which define the actions for the specific inbound and outbound traffic, 2) A Secure Channel that enables the packets to be transferred between OpenFlow switches and the remote controller 3) OpenFlow Protocol which provides an open standard controller-to-switch communication interface for network vendors to implement for their own network devices.

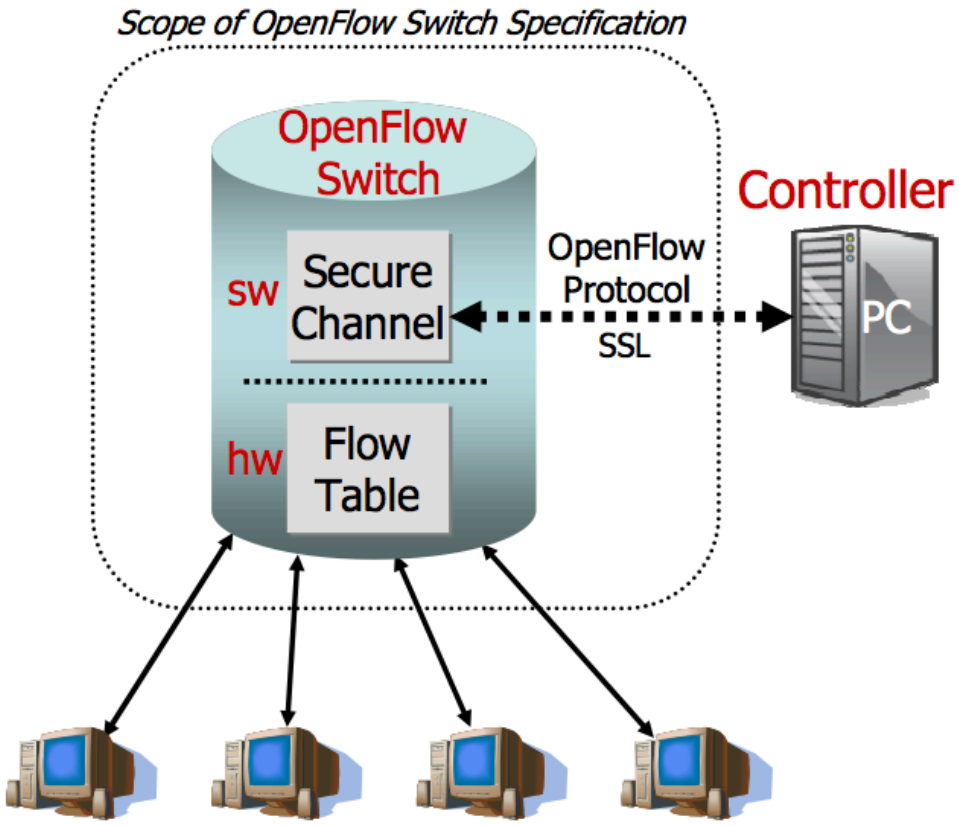

Figure 1: Idealized OpenFlow Switch. The Flow Table is controlled by a remote controller via the Secure Channel. [4]

Among these 3 components, the flow table is the one that we need to pay more attention here, since it has all the information (flow entries) about how the traffic is actually handled. These flow entries have header fields that separately identifies each flow entry and actions (such as forward, drop and modify-field) that defines the behavior of each flow in the order specified.

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN pri-or-ity	IP src	IP dst	IP proto	IP ToS bits	TCP/ UDP src port	TCP/ UDP dst port

Table 1: Fields from packets used to match against flow entries [5]

2.2 Software-Defined Measurement

Due to the recent bloom of the Big Data analytics among both academic and industrial communities, the need for having access to measurement analysis data of network performance on second-by-second basis to make better decisions and ensure the effective and efficient network management has never been such urgent. We have previously developed Narada Metrics [6] software-defined measurement and monitoring platform with perfSONAR [7] measurement points to address the needs of multi-domain application and network performance intelligence. All the measurement metrics are collected in our end-user software and can be accessible for authorized members to diagnose performance issues, identify end-to-end bottlenecks and determine the optimal network path.

2.2.1 System Features

Our software-defined measurement (SDM) platform is made up of four core features as following.

(1) Intelligent - In addition to the basic measurement data, it also provides other useful knowledge after data analysis. It utilizes adapted plateau detection (APD) algorithm for accurate detection of prominent single network path anomalies [8]. It also sends the smart notifications to the users about the network trends and anomalies based on their custom filters. The users can view the status of the their network at a glance from the monitoring dashboard with measurement points to provide notifications of critical anomaly events.

(2) Extensible - The system utilizes tools such as Ping, Traceroute, OWAMP (for one-way delay measurements) and BWCTL (for TCP/UDP throughput measurements) for end-to-end metrics such as

one-way delay, round-trip delay, jitter, loss, TCP /UDP throughput. However, it is not just a network performance monitoring system. It also provides the computation resource monitoring (such as CPU and memory usage of end nodes) and workflow status of data processing applications. The system can be extended, by adding new custom metrics to flexibly monitor the performance of any kind to suit measurement needs.

(3) Programmable - Centralized control and open RESTful APIs allow administrators to programmatically integrate deployment scripts, help desk workflows and other automation process. It provides line based APIs to integrate monitoring tasks within software-defined network infrastructure.

(4) Social and Secure – Today's end-to-end applications are spanning multiple segments of local, campus, regional and national network owned by different organizations where the monitoring data is only accessible for the organization owners. These organizations to setup policies through out system to share measurement resources and data with trusted enterprise users and external collaborators. It has a fine-grained access control to protect measurement data and resources. The system uses a role based access control (RBAC) policy engine to protect the performance monitoring resources.

2.2.2 System Architecture

Figure 2 shows our network performance monitoring deployment within a content delivery network. There are two main components – a Central Intelligence System (CIS) and multiple Measurement Point Appliances (MPAs). The CIS works as the "brain" of the system. It provides functions such as: discovery of geographically distributed MPAs, conflict-free scheduling of the measurement requests, collecting measurement results from MPAs, anomaly detection and user notification. We host our CIS in the Amazon

EC2 server and use another back-up server to balance the load and avoid single point of failure. The MPAs are end-point servers/ devices where the measurement software is actually installed to generate all the metrics (network metrics, computing metrics along with other custom metrics).

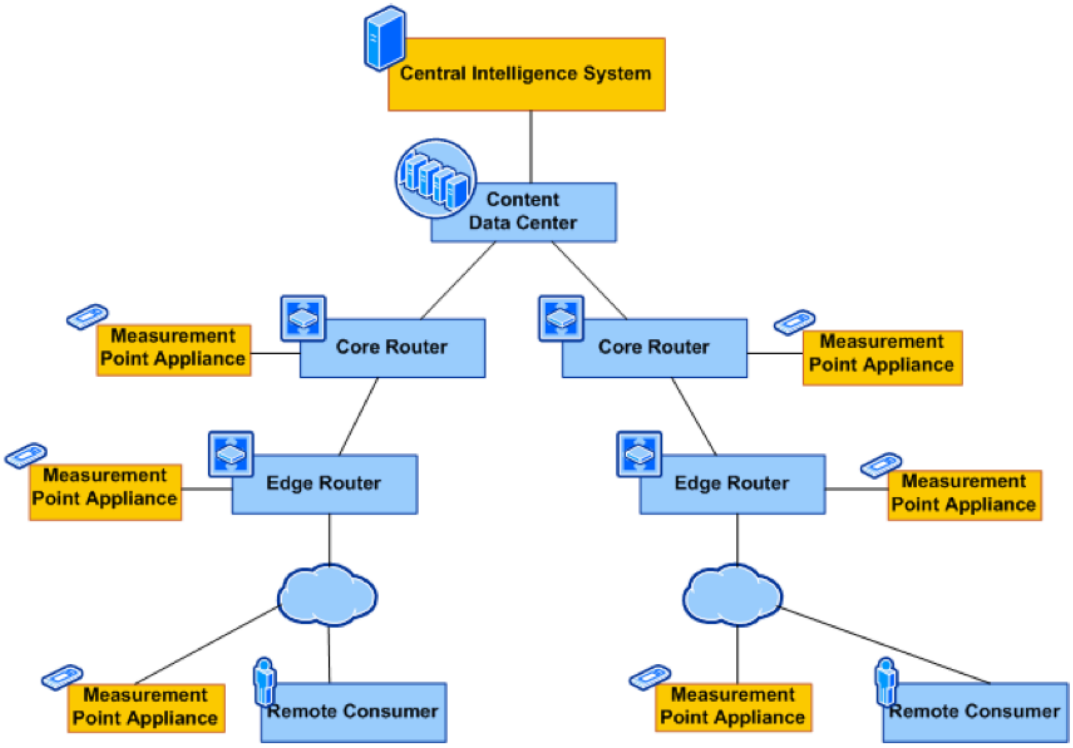

Figure 2: Intra-domain perfSONAR deployment within a content delivery network [6]

Our software-defined measurement system (through the CIS) enforces a conflict-free scheduling and measurement level agreement to ensure security level of measurement functions. This metric scheduling issue is resolved in our previous published paper [9] and the general idea is shown in the Algorithm 1. The author implemented the whole scheduler module for the Narada Metrics software-defined measurement system in a production level through test driven development process as part of this research work.

EDF-CE: For the given task conflict graph, find the measurement schedule during a hyperperiod

---

Input: task set Γ and task conflict graph
Output: start time st_{ij} and finish time ft_{ij} for each job τ_{ij} in a hyperperiod
begin procedure
1. Initialize rt_list with the ordered list of all release times in a hyperperiod
2. Initialize ft_list = {} /* ordered list of finish times*/
3. Initialize $pending\_job\_queue$ = {}
4. **do**
5. $time$ = get the next scheduling time point from rt_list and ft_list
6. add all newly released jobs at $time$ to $pending\_job\_queue$ in EDF order
7. **for** each job τ_{ij} in $pending\_job\_queue$ in EDF order
8. **if** τ_{ij} does not conflict with any of already scheduled jobs **and**
9. scheduling τ_{ij} at $time$ does not violate MLA constraint ψ
10. $st_{ij} = time$ and $ft_{ij} = time + e_i$
11. **if** ft_{ij} is later than the deadline of τ_{ij}
12. return error /* infeasible task set */
13. **end if**
14. remove τ_{ij} from $pending\_job\_queue$
15. add ft_{ij} to ft_list in order
16. **end if**
17. **end for**
18. **until** $time == hyperperiod$
end procedure

Algorithm 1: Conflict-free measurement scheduling algorithm [9]

3. ADON ARCHITECTURE REFERENCE

We previously proposed the early version of ADON: Application-Driven Overlay Network-as-a-Service for Big Data Networking [10] that we are referencing as the blueprint for our approach in this research project .The author made his own contributions to the extended ADON paper and we have submitted the newest version to the IEEE Transactions on Cloud Computing (TCC). The paper had already been "accepted with minor revision" by the time the author was finalizing this thesis for his graduation and it should be publicly available soon. We introduce the general architecture and the components of ADON in this chapter.

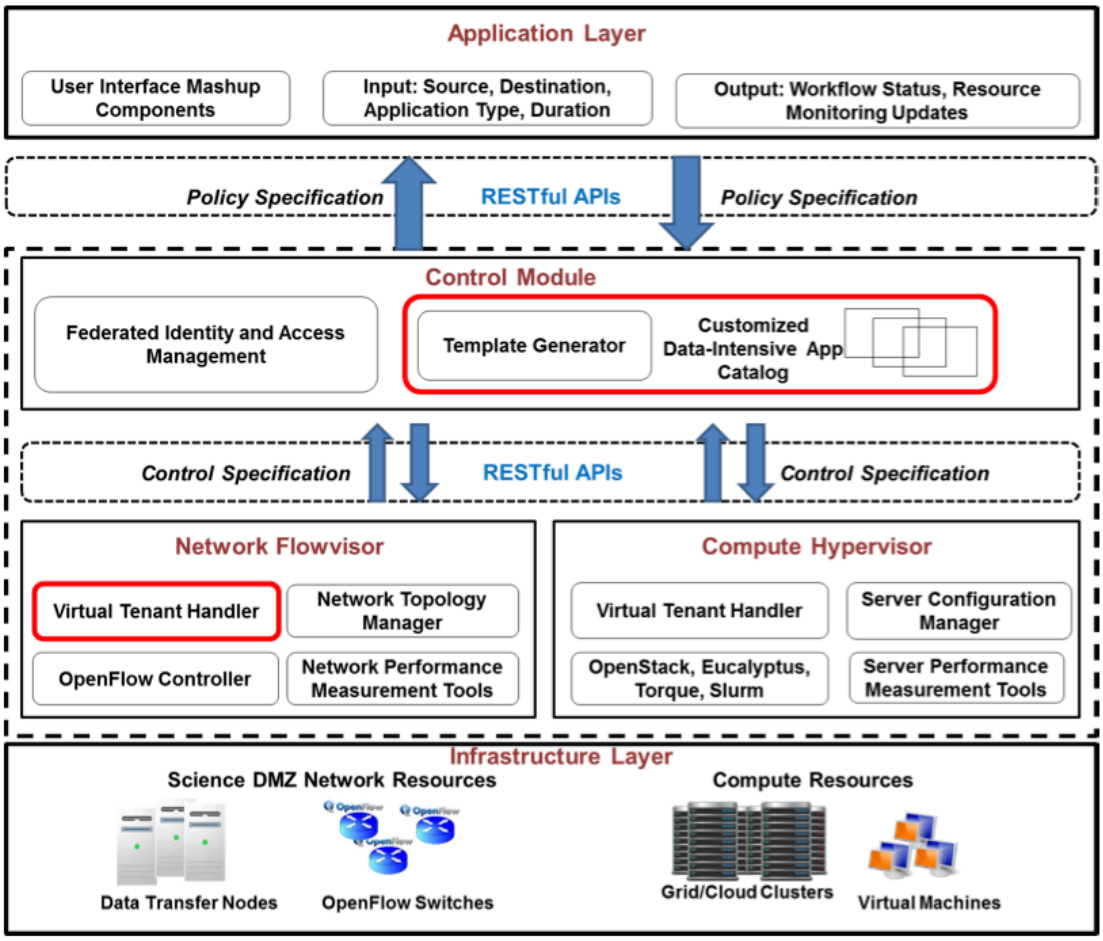

Figure 3: ADON reference architecture

Figure 3 shows the ADON architecture, which consists of the application layer, middleware layer and the infrastructure layers within which individual components interact with each other in to provision resources for incoming application requests.

The high-level application requirements along with Resource Specifications (RSpecs), Quality Specifications (QSpecs) and application priority are captured in the application layer. Depending upon the resources available in the infrastructure layer, the campus policy rules (maintained by the Performance Engineer) and those requirements captured in the application layer, the middleware will make (both

compute and network) resource assignments intelligently with the help of the performance monitoring of individual flows.

3.1 Custom Template Catalog

To allow composing and executing workflow pipelines for data-intensive applications in a reusable manner, ADON maintains a catalog of custom templates that contains RSpecs, QSpecs and corresponding best practice deployment solutions. The custom template catalog generator allows Performance Engineer to save a successfully configured template in a custom catalog database, which allows re-use for future flow provisioning instances.

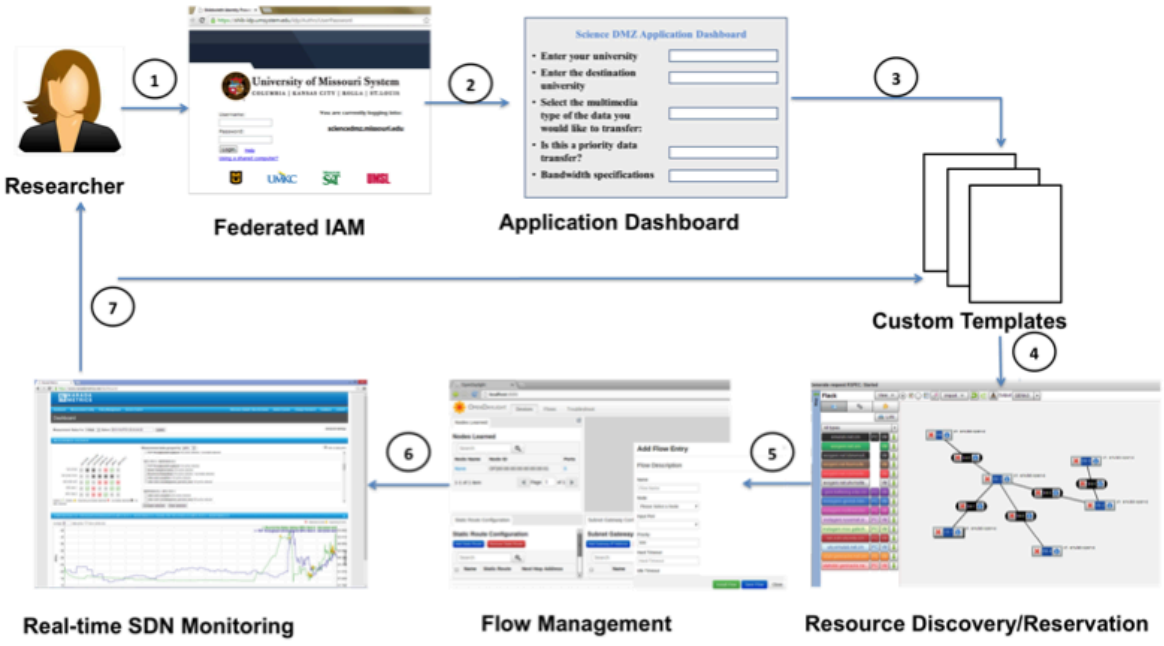

Figure 4: Sequential workflow of ADON during on-demand resource provisioning for a data-intensive application flow

Figure 4 shows how custom templates can be used as part of the sequential steps of ADON auto-orchestration during on-demand resource provisioning for a data-intensive applications. First, a researcher of a data-intensive application can securely request the ADON by authenticating with a Federated Identity and Access Management (Federated IAM) system that uses Shibboleth-based entitlements [11]. Subsequently, the researchers provide their application handling specifications through a simple and intuitive application dashboard mash-up, which are subsequently matched to a custom template that corresponds to the requested application type for resource discovery/ reservation of the necessary compute and network resources. Then, the custom template can be pre-configured by a "Performance Engineer" to apply specific resource descriptions and associated campus policies that can be interpreted by a network FlowVisor (i.e., proxy for OpenDaylight, POX [12] or Ryu [13]) to instantiate flows on intermediate OpenFlow switches within a data center. The software-defined monitoring systems are installed after successful deployment of the corresponding resources where users can visualize the performance of their application workflows. The custom templates also help configuring real-time network monitoring within the overlay network path to define triggers for dynamic resource adaption. Manual interventions that require the performance engineer attention can be minimized in cases where custom templates can be automatically re-applied through user 'self-service', if a similar specification was successfully handled previously.

3.2 Computation Location Selection

Figure 5 shows a typical data-intensive application life cycle where the final repository of processed data is reached through two possible paths involving either local (A.1) or remote (B.1) computation. The choice of

computation locations as well as storage destination is mandated by a comprehensive cost optimization which takes into consideration factors such as: availability of computation resources, residual utilization left-over from existing applications, and resource specification of the new application. In the case that data is processed locally, the resultant data will be locally stored (A.3) and a copy will be transferred to the Final Destination (A.2). Alternately, if the data is processed remotely, the resultant data will be stored in the Finial Destination (B.2) and a copy will be transferred to Local Store (B.3).

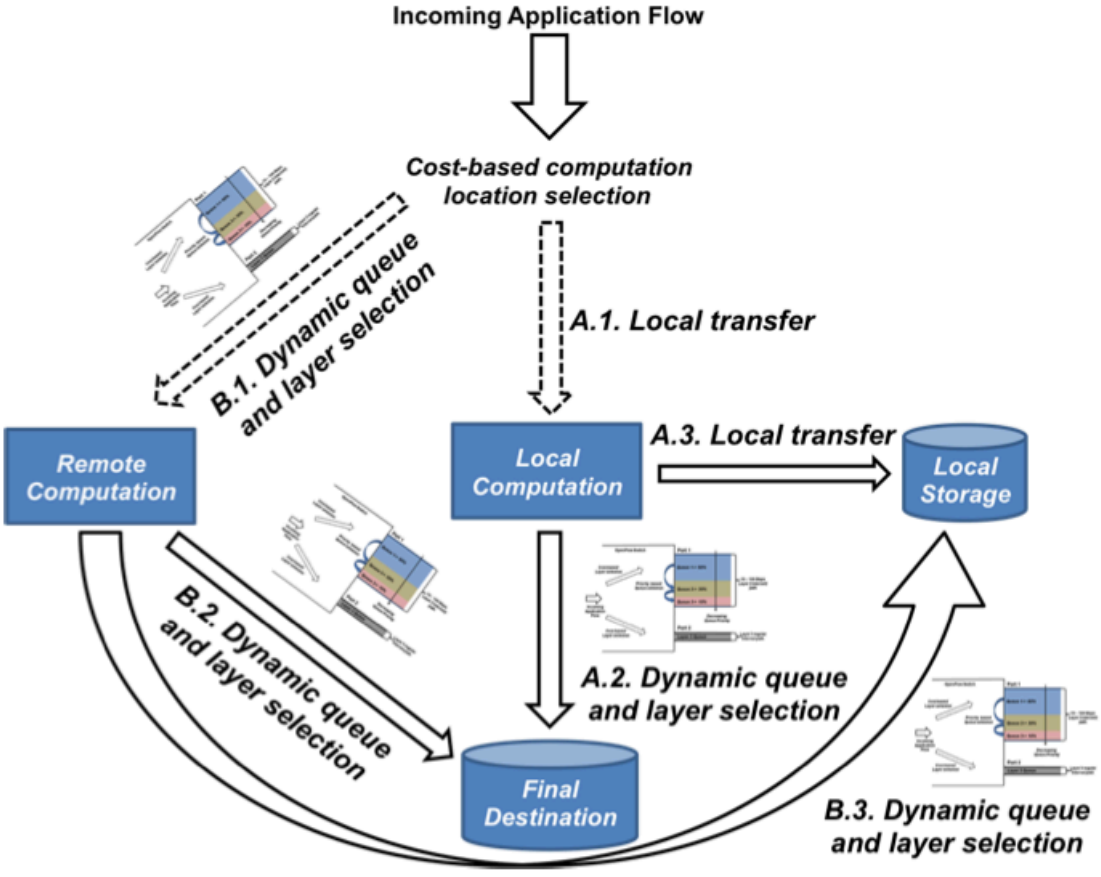

Figure 5: Illustration of computation location selection

The ADON middleware broker receives resource allocation requirements of any application a_x from the Template Generator represented as quality specification $^{QS}a_x$, resource specification $^{RS}a_x$ and the

application priority P_{ax}. The Performance Engineer is responsible for translating the resource requirements into QSpecs and RSpecs and forwards the translated information to the ADON middleware broker. Upon receipt if any such application resources allocation request, the broker is responsible for the allocation of optimal network and computational resources from a set of available resource R so that allocation not only satisfies the application QoS needs, but also minimizes the allocation cost. The specific cost function can be designed to minimize any performance or economic metric.

The Edge Optimization Algorithm (Algorithm 2) takes a new application with the corresponding RSpecs and QSpecs requirements as inputs, then it compares the local and remote processing costs (both network and compute) and takes the final decision on allocating compute resources, suitable Layer 2 or Layer 3 network resources.

Input: *RSpec* $RS_{a_{new}}$, *QSpec* $QS_{a_{new}}$, and Priority $P_{a_{new}}$ of new application a_{new}
Compute: $\mathcal{R}$=$\mathbb{R}$ - $\sum_{a_i \in A} R_{a_i}$
Compute: $P_{min} = min(P_{a_i}) \quad \forall \quad a_i \in A_{L2}$
Output: Resource allocation $R_{a_{new}}$
begin procedure
if $\mathcal{R} \geq R_T$ **then**
  /*Check for local computation*/
  $C_L = computeLocal(QS_{a_{new}}, RS_{a_{new}}, P_{a_{new}}, P_{new})$
  /*Check for remote computation*/
  $C_R = computeRemote(QS_{a_{new}}, RS_{a_{new}}, P_{a_{new}}, P_{new})$
  /*Compare computation costs*/
  return $\{R^C_{new} = RS^C_{new}, R^N_{new} = RS^N_{new}\}$
  if $C_L < C_R$ **then**
    $\mathcal{R}^L = \mathcal{R}^L - \{R^C_{new}, R^N_{new}\}$
    Include a_{new} to A
  else
    $\mathcal{R}^R = \mathcal{R}^R - \{R^C_{new}, R^N_{new}\}$
    Include a_{new} to A
  end if
else
  Push a_{new} to resource scheduler
end if
end procedure

Algorithm 2: Edge Optimization Algorithm

3.3 Dynamic Transit Selection

As shown in the earlier part of the paper, there are three layers (application, middleware, infrastructure layer) in the ADON architecture. It further decouples the middleware layer with 1) control module for managing user access control (Federated Access Management) and capturing application priority and specifications (Template Generator), 2) Network FlowVisor for managing network resource (Network Topology Manager), monitoring the network health (Network Performance Measurement) and managing data traffic (OpenFlow Controller) and also 3) Compute Hypervisor for managing compute resource (Server Configuration Manager) and monitoring the compute resource (Server Performance Measurement). By decoupling the system in such manner, ADON reduces the complexity and makes it easy for engineers to break down and understand the system.

The general purpose of the Network FlowVisor is to work as a "brain" of the network broker of the system and provide the intelligence for the comprehensive network provisioning. It is responsible for adaptive allocation of network resource and dynamic handling of the incoming flows where specific functions such as scheduling of the application flows by priority, selecting data transmit path based on real-time network performance, or even requiring more overlay network resource from the infrastructure layer in case of the traffic burst and many others, can be designed and implemented by engineers with their own strategy for different requirements. These functions and solutions are replaceable/reusable to meet different and changing requirements of the specific data intensive applications. We can also extend the existing functions and put more intelligence into the Network FlowVisor for increasing demand of new users.

In the process of providing these different functions, the Network FlowVisor follows some of these

common steps. It first interacts with the Template Generator in the Control Module to understand the requirement of the user applications. It then receives inputs from the Network Performance Measurement Module that provides network health status notifications such as flow statistics and other such QoS impacting factors. After that, it generates some solutions and makes decisions based on those user requirements and monitoring data. At last, it interacts with control and management components to send the final actions to the underlying resources in infrastructure layer. Among many functions that supports the Network FlowVisor, we detail how we are choosing the best data transfer path for the newly generated application flow with the help of the software-defined measurement. Other functions such as adaptive resource allocation and application flow scheduling within a determined network path are not the focus of this paper.

One of the SDN's advantages is the global and dynamic control of end network devises (such as switches and routers). In the traditional networks, the static edge routers that implement network protocol are responsible for selecting the data transfer path between two data transfer nodes. But these routers are only aware of their neighbor routers and do not have the global view of the whole network topology or the performance of the each links between the nodes, which makes it hard to select the best transit path.

We try to utilize the network performance measurement data and centralized control feature of software-defined networking to select the best data transfer path, which meets the application end users' expectation. The principles of our approach are 1) Keep it simple not complex. 2) Achieve essential then better 3) Stick to application users' requirement.

The dynamic transit selection is illustrated in Algorithm 3. Let us assume that each candidate path has n metrics (such as throughput, delay, jitter and loss), which we can measure from our network performance

monitoring system. Equation (1) defines the set of all metric weight W, which can be directly configured by the researchers with their preferences and application requirements from ADON application dashboard. These metric weight values will be fed into the custom template catalog along with other application specific data such as QSpecs. Equation (2) defines the set of all metric score S, which can be queried from our metric-score lookup table in the custom template database given measurement values for a metric kind. With principles above kept in mind, we define the final adjusted qualification score Q (W, S) of each candidate network transfer path for a specific application a_x as shown in Equation (3). The one with the maximum qualification score Q will be selected as the best candidate path.

$$W = \{w_i\}_{i=1}^{n} \quad (1)$$

$$S = \{s_i\}_{i=1}^{n} \quad (2)$$

$$Q(W, S) = \sum_{i=1}^{n} (w_i * s_i) \quad (3)$$

Algorithm 3: Qualification calculation equation for candidate transit path

The mapping between measurement values and metrics score is application specific. The network performance can either refer to the best practices from similar existing systems in the field or create a simple lookup table using QSpecs and RSpecs at first if there is not any. Let us take the bandwidth as an example. If QSpecs requests the minimum bandwidth of 1Gpbs for some data intensive application and RSpecs restricts the maximum of bandwidth of 10Gbps with existing underlying network resources, the lookup table can simply map the bandwidth measurement data and metric score in a linear relationship as shown in the Table 2 with upper bound score of 10 and lower bound score of 1. The dynamic transit selection broker can use this table as a reference to score bandwidth metric of each transfer path. The

custom template catalog will later be tuned upon growing knowledge base after configuring many similar applications and update the lookup table with more successful practices.

Bandwidth	Metric Score
1,000 Mbps	1
2,000 Mbps	2
3,000 Mbps	3
4,000 Mbps	4
5,000 Mbps	5
6,000 Mbps	6
7,000 Mbps	7
8,000 Mbps	8
9,000 Mbps	9
10,000 Mbps	10

Table 2: Bandwidth metric score lookup table example

Figure 6 shows the dynamic transit selection process block diagram. After communicating with application users, a performance engineer defines the QSpecs and RSpecs of the application. If a custom template for the same application or similar ones are already stored in the template catalog, the performance directly send the metric score lookup table along with user defined metric weight set W in Equation (1), to the dynamic transit selection module. If such templates do not exist in the template catalog, the performance engineer generates a metric score lookup table as part of network solution along with other necessary solutions to meet data storage, compute and other general-purpose requirements. Since we are periodically monitoring the health condition of available candidate data paths, we can get all the latest network metric

measurement data and feed them to dynamic transit selection module. Using these measurement data as input, we obtain the set of the metric score S in Equation (2) by referencing the metric-score lookup table. We take W and S as our function input; use Equation (1) to calculate the overall adjusted qualification score Q of each candidate paths and return the selected candidate that has the maximum qualification score. Finally, OpenFlow controller receives the result and sends the corresponding actions to the underlying OpenFlow switches to define the flow for this application.

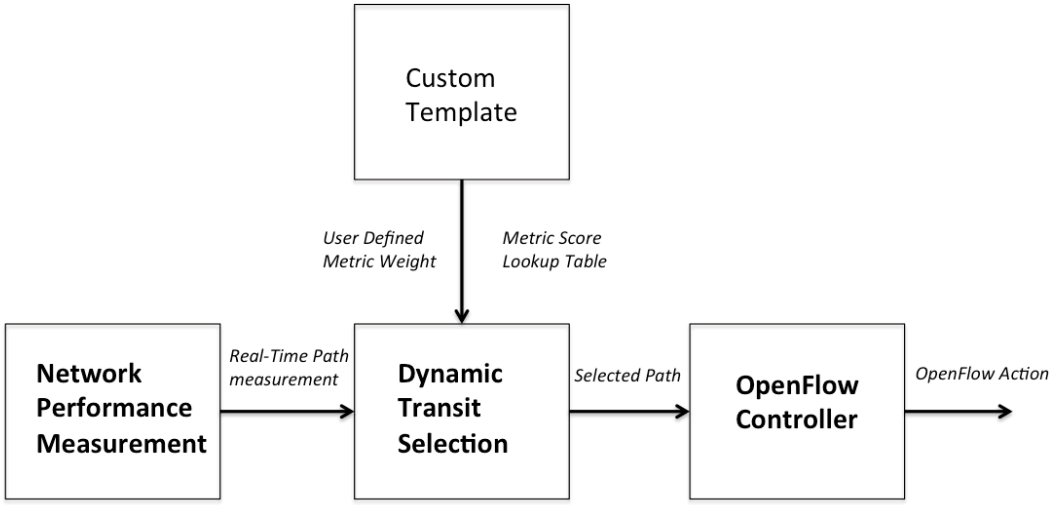

Figure 6: Dynamic Transit Selection Block Diagram

By decoupling the ADON architecture into fine-grained logical components, we separate complex responsibilities and assign them to people in different roles to make sure they can focus on their own related responsibilities and work with each other much easier. As we can see in the whole application transit selection process; end users just need to specify the quality requirements and performance preferences (metric weight) in application layer without worrying about any detail of the system

implementation and underlying resources, the performance engineers generate custom templates (including QSpecs, RSpecs and other useful information like metric score lookup table) for the application and delegate the responsibility to network FlowVisor; the software engineers implement the network FlowVisor logic to make decision and choose the best transit based on the measurement data from network performance measurement module and related useful information from custom template.

4. SOYBEAN KNOWLEDGE BASE CASE STUDY

Soybean Knowledge Base [14] is comprehensive and growing web resource for soybean's genetic and genomic data to meet the increasing need from the soybean community to have a one-stop interactive, web-based portal to browse, access and share knowledge about soybean.

Figure 7 shows the original data flow within the SoyKB infrastructure. The raw NGS data is mailed from partner institutions (e.g., China), pre-processed by PI Joshi and her team at MU, and later transferred to remote HPC sites for analysis. For the re-sequencing analysis, their team uses three available iPlant connected HPC infrastructures at: ISI (Information Science Institute) [15], TACC (Texas Advanced Computing Center) [16] or XSEDE [17]. The iPlant [18] resources at the U. of Arizona serve as SoyKB's Data Store, and can be accessed using iRODS [19]. Data can be replicated from U. of Arizona to the servers at other HPC sites over Internet2, which allows low latency data access when running the workflows on HPC resource. Pegasus [20] workflow system is used to control data movements, computations and execution in HPC environments. The execution environments are based on HTCondor 0, a specialized workload management system (i.e., job scheduler) for compute-intensive jobs in distributed environments, allowing users to place their jobs into a queue for batch computation. Finally, the analyzed data is sent to an iPlant virtual machine (Earnshaw Server in the Figure 7), which hosts the SoyKB website for researchers to view/download the data. In order to scale the SoyKB infrastructure and services to meet the KBCommons need to provide a comprehensive web resource at the front-end and backend powered by adequate resources to handle multi-omics data integration for a given species, there are certain limitations to be overcome in expansion activities.

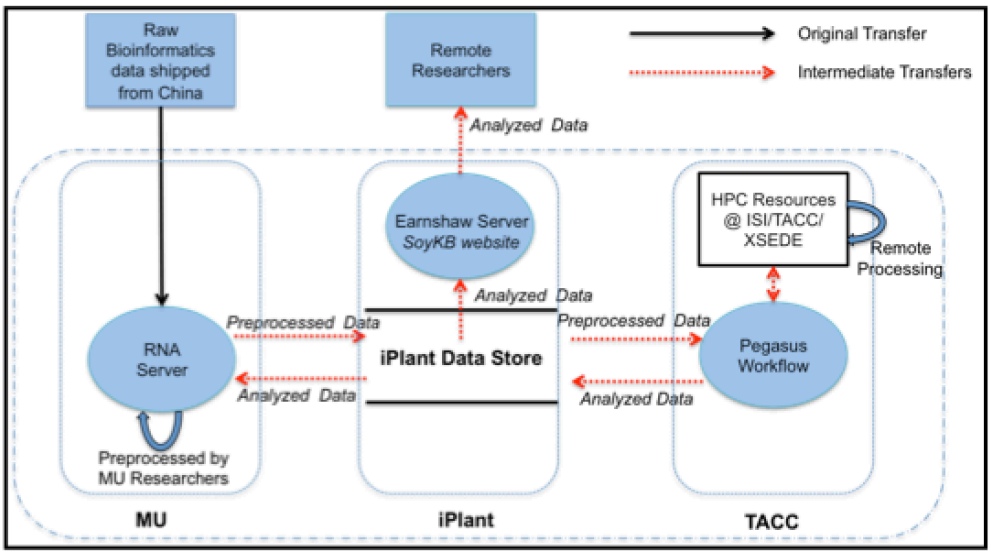

Figure 7: Original data flow within the SoyKB infrastructure

Those limitations in the original SoyKB infrastructure that motivated the KBCommons hybrid cloud architecture design and related services implementation. 1) The first, important step involves data transfer between MU and other remote compute sites such as iPlant, can take up to several days using traditional file transfer method for large data. Faster data transfer capability over the network through Science DMZ environments with advanced data transfer protocols and data transfer nodes can significantly improve efficiency and multiple KB workflows setup. 2) Also, the original setup neither provides easy access to resource availability and performance information (such as availability, status, wait time, network health etc.), nor flexible control of resources to address special needs, causing significant delays in workflow completion due to other jobs. To overcome the performance visibility limitations, there needs to be a dedicated effort to build a 'measurement plane' across the infrastructure to instrument resources with in both active and passive mode, and having services to effectively collect and analyze various workflow related custom metrics. For flexible control of compute and network resources, a 'control plane' needs to

be developed that not only allows local MU HPC resources to be added on-demanded to Pegasus workflow resource pools for computation regulation but also dynamically handles large data flows on important paths between the private and public cloud using software-defined networking for network regulation. 3) Lastly, the setup does not facilitate creation of 'templates' that can provide information about previous successfully executed workflow configurations and access control rules for a particular application. In order to truly support 'self-service' capabilities, we will need to create services that help with re-use of workflows for repeatable execution for different resource configurations.

To address these limitations and expand SoyKB for KBCommons, we designed an early vision of the infrastructure setup as shown in Figure 8. The new architecture will feature a set of core services as follows: 1) Template service: We develop custom template for our SoyKB extension efforts as an exemplar for other KBs to build upon. They help in abstraction of high-level policy and performance throughput requirements of KB users, and map them to lower-level control specifications implementable in an on-demand manner with virtualization technologies such as OpenStack [22] and OpenFlow. 2) Performance monitoring service: For end-to-end performance monitoring, we instrument the infrastructure with software-defined Narada Metrics MPAs (measurement point appliances) based on OnTimeSecure framework. Such setup will enable performing various custom metrics integration at system, network and application-levels into relevant measurement archives, and obtain timely accurate performance intelligence (e.g., log analysis and anomaly event notifications). We already started collecting measurements between MU, iPlant and ISI servers. With high level of manual coordination and measurement policy, we are able to address questions like: Who can initiate measurements within/between iPlant, MU and ISI resources? Who can view the measurement data of certain performance metrics? We create web-portals to provide SoyKB

users with relevant information such as statistics, graphs about current running processes, network health condition, workflow status notifications and use the analyzed data to troubleshoot data transfer and workflow throughput bottlenecks. 3) Data import service: We develop OpenFlow controller application that utilize performance analytics and allow intelligent provisioning of network resources based on the desired requirements of a given application, and the wide-area file system capabilities to import/store data at high-speeds. We conducted experiments on traffic engineering on MU's high-speed network connection to Internet2 AL2S performing transit selection between Layer 2 overlay network and regular Layer 3 network alternatives at the campus-edge. 4) Data analytics service: We broker the local and remote resources seamlessly using suitable OpenStack and HTCondor configurations. It will consequently allow Pegasus workflows for various KBs to be configured to utilize MU cluster resources or any external HPC site resources in an on-demand manner.

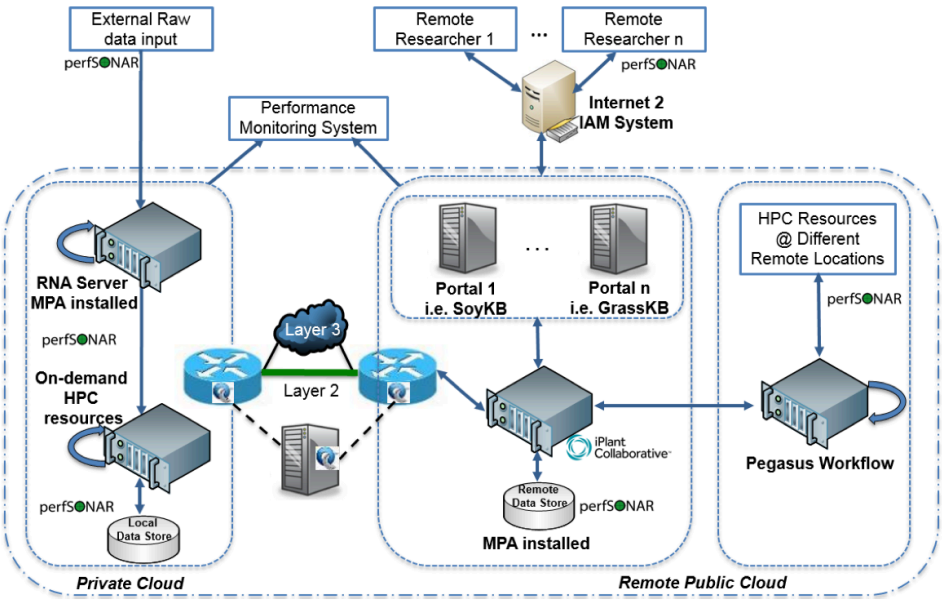

Figure 8: Expanded KBCommons workflow in hybrid cloud architecture

We validate our effectiveness of the software-defined measurement, the utilization of custom template, computation location selection and the dynamic transit selection approach through the real case study of a SoyKB application on a wide-area overlay network test-bed implementation across campuses in this chapter.

4.1 Custom Template for SoyKB Application

To have a better understanding of the SoyKB application and figure out the application requirement, we had frequent meetings with the SoyKB group in the early stage of the research. Survey for SoyKB application (Appendix) was conducted in order to abstract the various specifications (including QSpecs and RSpecs) for the custom template and provide a better performance and quality of experience for the soybean researchers. SoyKB handles the management and integration of soybean genomics and multi-omics data along with gene function annotations, biological pathway and trait information. The process consists of large data sets being transferred to MU for pre-processing and subsequently being transferred to either local or remote HPC sites (at: ISI, TACC or XSEDE) for analysis as shown in Figure 8. Following computation completion, the resultant processed data is transferred to MU and also to the iPlant Collaborative storage, in addition to becoming available for user browsing in SoyKB.

The above process can suffer from long run-time for the overall process due to workloads at HPC centers and slow network transmission of large data sets despite high bandwidth connectivity over paths involving Internet2 national high-speed network backbone. End-to-end performance monitoring is achieved through instrumentation of the infrastructure with perfSONAR measurement points based upon our Narada Metrics framework. Various custom metrics at the system, network and application-levels are integrated into

relevant measurement archives in order to obtain timely and accurate performance intelligence (e.g., log analysis and anomaly event notifications). A corresponding SoyKB application template can be: (i) RSpecs - HPC compute environment with local site and remote location speedup processing capabilities of raw genomic data, and (ii) QSpecs - high flow throughput with little or no packet loss to provide fast-enough data transfer to justify wide-area network path selection to move data to remote compute resources.

4.2. Measurement Support for SoyKB Application

The software-defined measurement is playing an important role for SoyKB application not only by providing the overall performance visibility to the performance engineers and end users, but also by supporting the various control logic, such as computation location selection and transit path selection. We have deployed our Narada Metrics Measurement Point Appliances (MPAs) in the corresponding MU and iPlant data transfer nodes. The following Figure 9 – 12 show some examples of network (such as TCP throughput and round-trip delay) and compute (memory usage) performance metrics, where PLSI1 represents the local data transfer server at University of Missouri hosted by SoyKB group and OLLIVANDER represents the remote virtual server in iPlant dedicated for SoyKB application flow test.

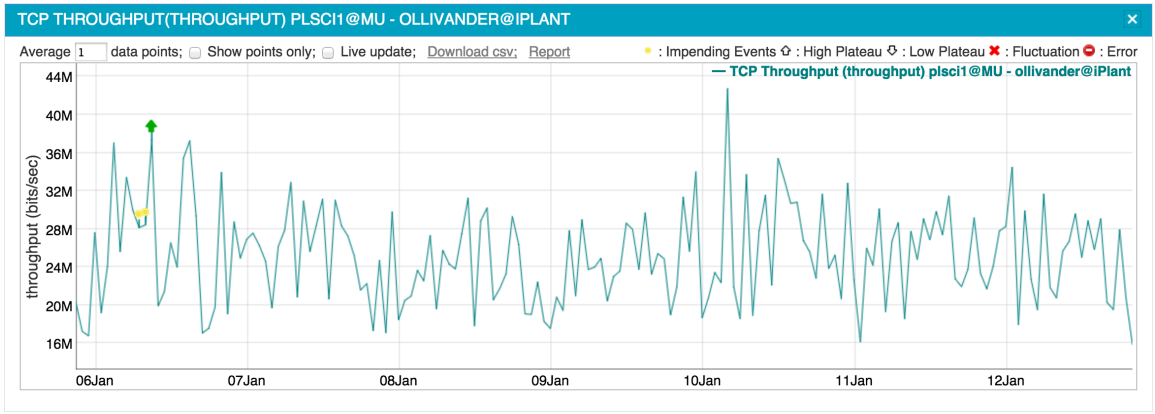

Figure 9: TCP throughput between PLSCI1 at MU and OLLIVANDER at iPlant

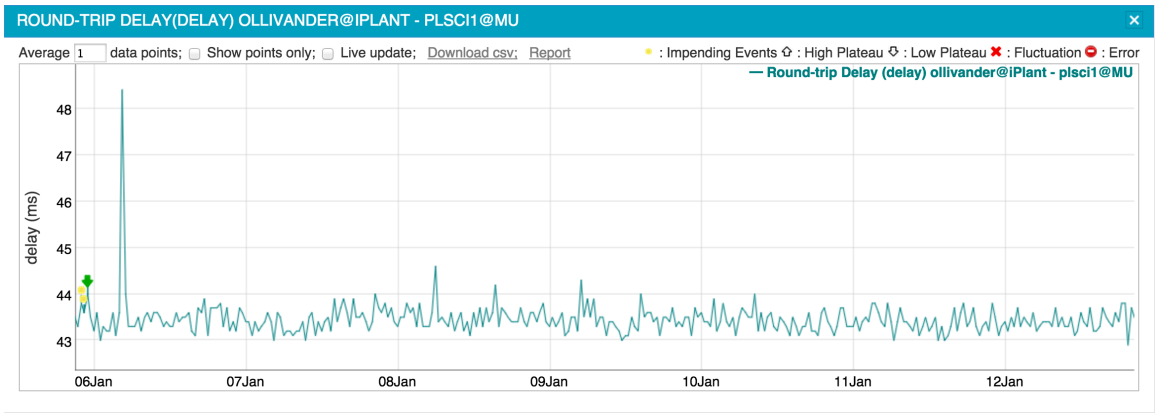

Figure 10: Round-trip delay between PLSCI1 and OLLIVANDER at iPlant

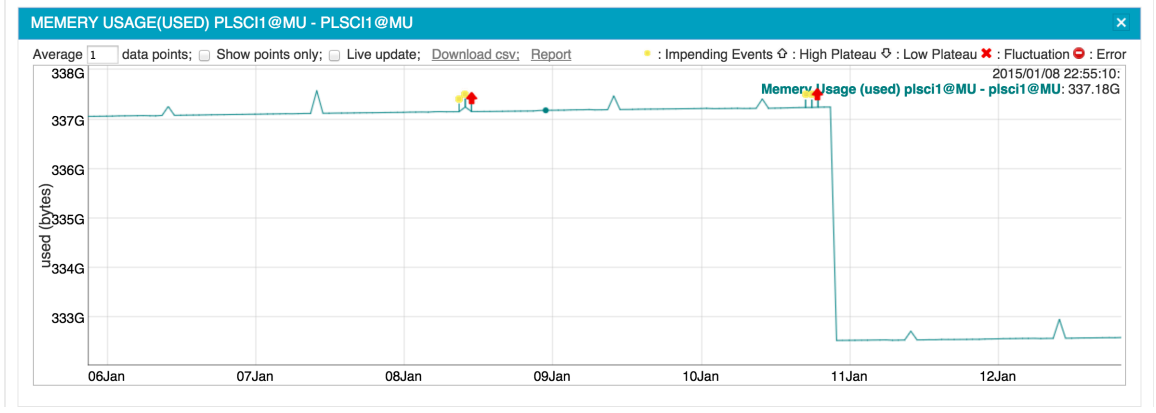

Figure 11: Memory Usage of PLSCI1 server at MU

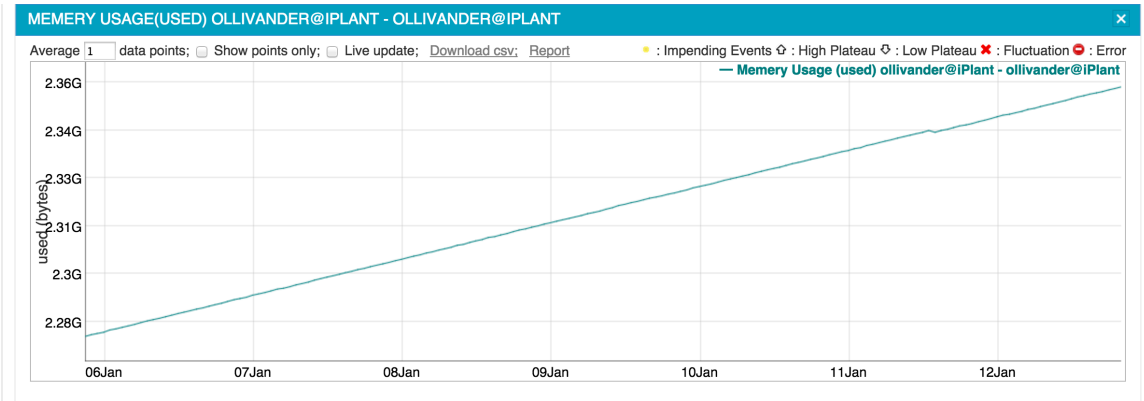

Figure 12: Memory Usage of OLLIVANDER server at iPlant

As shown in the figures above, SoyKB application uses the Pegasus workflow management system to process its raw data, which is capable of automatically locates the input data and necessary computation resources for flow execution. The workflow can be executed either in local ISI servers or remote resources like TACC, however, the resource mapping and task management is handled in the ISI. It has its own command line based tool for monitoring workflows. As we worked more closely with SoyKB group, we realized that there was an increasing need for them to visualize the monitoring result in a more intuitive manner. To solve this problem and improve the quality of experience (QoE) for the SoyKB group, we decided to do some extra work and provide them with a more intuitive graphic user interface (GUI) for their workflow monitoring. With the first initiative to let the SoyKB research group to visualize the monitoring result in a graphic manner, we started to integrate the Pegasus workflow management system and Narada Metrics performance monitoring system, since Narada Metrics is extensible and provides GUIs to show the measurement data. The author implemented the first version of this integration. We wrote a Python XML-RPC server that runs in the ISI listening to the local client requests from SoyKB users. The

server is capable of providing all the monitoring data that the Pegasus command line monitoring tool generates, including all workflows previously created by the user (Figure 13), most recent status (Figure 14) and detailed analysis data of a workflow (Figure 15) to the Narada Metrics CIS.

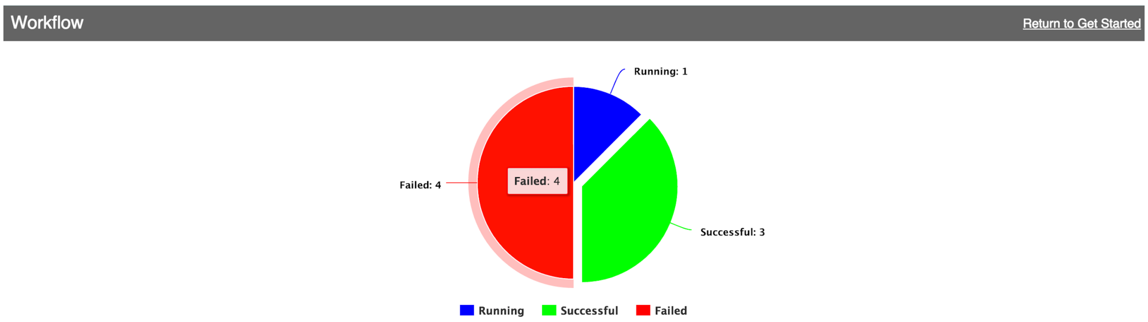

Figure 13: All workflows previously created by the user

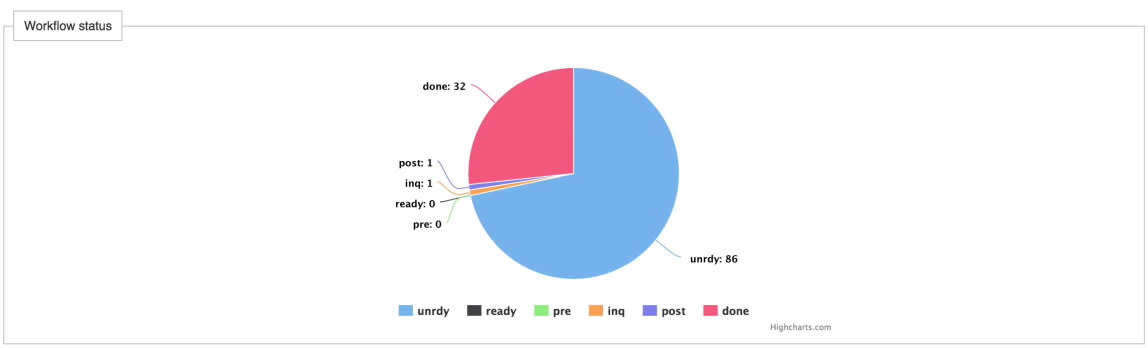

Figure 14: Most recent status of a workflow

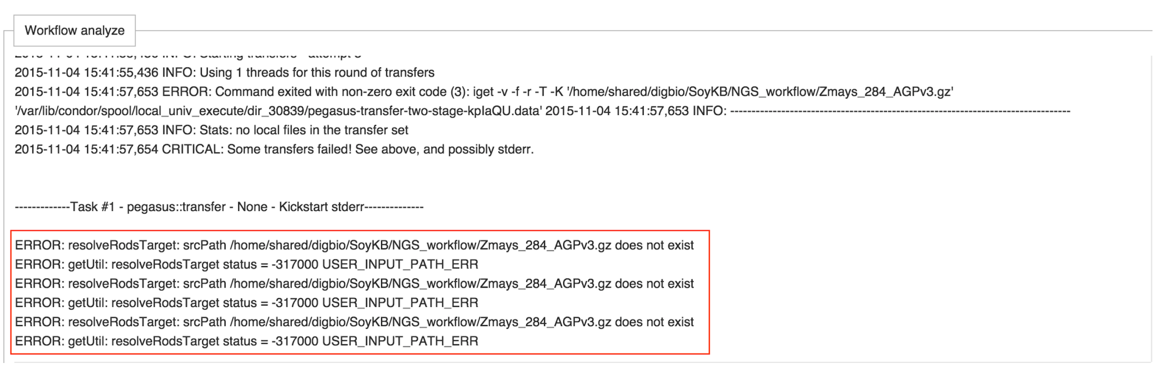

Figure 15: Detailed analysis data of a workflow

The Figure 16 shows the portion of our server side code, which is responsible for sending the most recent status of a running workflow. When a researcher starts running a workflow on ISI the server first reads the directory of the newly started workflow, then calls the Pegasus internal monitoring command Pegasus-status using Python sub-process module, parse the result in a desired manner and send them Narada Metrics every 5 seconds unless the workflow is done or failed. The current status data is stored in the CIS measurement data archive for the future reference. The code shown above is the first implementation of the Pegasus - Narada Metrics integration and it has changed significantly in the software development process. In addition to receiving measurement data passively, the Narada Metrics CIS is also able to actively communicate with MPA in the ISI system by calling the RESTful APIs. More specifically, it uses POST action to update the schedule of workflow custom metric (a.k.a. sets the remaining time to execute to 0) in MPA, which causes the MPA to execute the Pegasus monitoring commands immediately (using Python sub-process module). In either case, the user confidentiality (username and password) does not leave the ISI server. Plus, Narada Metrics uses two-way authentication scheme, which ensures that an MPA will only accept a measurement schedule from a known CIS, and in return a CIS will only accept measurement results from a known MPA [6].

```
class startMonitoring(multiprocessing.Process):
    def run(self):
        flag = True
        result = {}
        while flag:
            if pegasus_path !="":
                #Use subprocess to call pegasus-status command
                p = subprocess.Popen([r'pegasus-status', pegasus_path], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = p.communicate()
                #Output type is a list, the bandwidth information is in the first position of the list
                # the type is a string
                s = (output[0].split("%DONE\n")[1]).split("Summary")[0]
                #Parsing the result of pegasus-status
                vs = re.findall(r'\d+', s)
                # Order of the parsed results
                # READY   PRE  IN_Q  POST  DONE  FAIL %DONE STATE DAGNAME
                result['unready'] = int(vs[0])
                result['ready'] = int(vs[1])
                result['pre'] = int(vs[2])
                result['in_queue'] = int(vs[3])
                result['post'] = int(vs[4])
                result['done'] = int(vs[5])
                result['fail'] = int(vs[6])
                result['percent_done'] = float(vs[7])
                print result
                #Send the result to the NaradaMetrics CIS
                results(result)

            time.sleep(5)
            if result['percent_done'] == 100.0:
                flag = False

#thead list for different monitoring tasks.
threads = []
#Use xmlrpc protocol for server-client communication.
server = SimpleXMLRPCServer(("localhost", 8888))
#Register the add_task function for adding different monitoring tasks.
server.register_function(add_task, "add_task")
#Run forever until the KeyboardInterrupt event. Press control+c to terminate.
server.serve_forever()
```

Figure 16: Example code of Pegasus – Narada Metrics integration (workflow status update)

4.3 Computation Location Selection for SoyKB Application

In our ADON mathematical model, the function compute Cost() is a generic abstraction for calculating the cost of different applications. It can be defined and represented in different forms based on the specific requirements of each application. The purpose of calculating this cost is to decide: is computation performed locally (at MU) or does it use remote HPC resources (at ISI, TACC or XSEDE) and, do we use AL2S (Layer 2) or regular IP (Internet) network connections to transfer the data to the iPlant data store? In the SoyKB application, for example, it is vital for users to get the analyzed results in a timely manner after

they run the application with a large size of input data, which will speed up their bioinformatics research process.

To meet this requirement, we can define the cost as the total time taken, which is mainly comprised of data computation time and network transfer time. Once the application is formalized as a workflow, the Pegasus Workflow Management Service can map it onto available compute resources and execute the steps in appropriate order. However, it is possible that the service cannot find enough resources and needs to put the task in a prioritized queue. When there are many application tasks waiting in the Workflow Management Service queue, the SoyKB application has to wait for a long time to start the Pegasus workflow. As such, we have to take this time into consideration and denote it as T_q, see the notations used in Table 3. The Resource mapper component in Pegasus is responsible for diving the jobs and mapping these jobs to corresponding nodes to optimize performance. Since we are also able to get the data size that needs to be computed S_c, number of the worker node assigned N and the average compute throughput of each worker node Γ_c, we can easily calculate the total estimated computation time by adding the waiting time and actual computation time with the equation below. One thing to note is that, if currently any local HPC resources with Pegasus workflow are not available for SoyKB, we set the value of T_q as infinite because we are always using remote HPC resources at ISI, TACC and XSEDE, as shown in Algorithm 4.

All of the SoyKB research data is stored in the iPlant Data Center and is available to query through a website hosted in iPlant Atmosphere cloud platform that is similar to Amazon EC2. The iPlant collaborative uses iRODS to manage and transfer the data, which creates multiple TCP streams for transferring data. Since our network performance monitoring system can provide us the round-trip time

(RTT) delay of the transfer, we can configure the TCP buffer size and length of the processor input queue at the end nodes accordingly by following well-known TCP tuning practices [23] to achieve throughput that is close to the theoretical maximum throughput Γ_t. However, both the AL2S and regular Layer 3 network connections are not always able to provide the maximum bandwidth desired for SoyKB, especially when there are high levels of cross-traffic passing through the shared links, or source to destination physical distance is large enough to impact TCP behavior.

T_t	Estimated total time for getting results
T_c	Estimated data computation time
T_n	Estimated network transfer time
T_q	Estimated waiting time in the task submission queue
S_c	Size of the data to be computed
S_t	Size of the data to be transferred
N	Number of the worker nodes used for computation
Γ_c	Average compute throughput of each worker node
Γ_t	Theoretical maximum throughput after TCP tuning
Γ_a	Available TCP throughput measured by monitoring system

Table 3: Notations used in the SoyKB cost computation

begin procedure
/*computeCost()*/
$T_t = T_c + T_n$
$T_c = T_q + S_c \ / \ (N * \Gamma_c)$
$T_n = S_t \ / \ \min(\Gamma_t, \Gamma_a)$
end procedure

Algorithm 4: SoyKB application local or remote cost computation

In this scenario, the network monitoring system can again play an important role and provide the maximum bandwidth and the current throughput utilization of the network links to calculate the actual available

bandwidth left for the SoyKB data transfer Γ_a, or directly measure single thread TCP throughput and treat it as approximate available bandwidth. We can compare the theoretical TCP throughput and actual available bandwidth to get the estimated network transfer time, given that the size of data to be transferred S_t is known to our ADON-related support services. We particularly remark that the analyzed data will be significantly reduced after the computation process and could be only about 1/10th of the original raw data size. This indicates that even if the local HPC cloud resource is not as powerful as the remote cloud at TACC, it might still provide an increased overall performance by significantly reducing the estimated wait-time in the task queue for the computation part, and by reducing the size of the data to be transferred over the wide-area network segments.

5. EXPERIMENTAL EVALUATION

In this chapter, we first describe the wide area test-bed across MU and OSU [24] campuses from which we collected our experimental results. Then we show a case study where SoyKB application flow is handled effectively with other data intensive applications based on the network performance measurement. We also show some additional improvement we can achieve for SoyKB application from our experiments.

5.1 Testbed Setup

The testbed setup for the real-network case studies is as shown in Figure 17, which consists of the OSU and MU campuses connected through an extended VLAN overlay that involves an Internet2 AL2S connection by way of local regional networks of OARnet and GPN/MOREnet in Ohio and Missouri, respectively. Each Science DMZ has a matching DTN equipped with dual Intel E5-2660, 128GB of memory, 300GB PCI-Express solid state drive, and dual Mellanox 10 Gbps network cards with RoCE [25] support. Each Science DMZ has perfSONAR measurement points powered by Narada Metrics for continuous monitoring at 1 - 10 Gbps network speeds. A common Dell R610 node in the OSU Science DMZ is used to run the OpenFlow controller based on OpenDayLight for the NF-VTH functionality. The NEC switch on the OSU end is connected to OSU's 100 Gbps Cisco Nexus router at 10 Gbps, but has the ability to scale to 40 Gbps as the Science DMZ grows to support future researchers and applications. At MU, the Science DMZ features a Brocade VDX 8770 switch to attach various nodes in the Science DMZ, and a 100 Gbps Brocade MLXE router at 10 Gbps interface speeds, with the ability to scale to 100 Gbps speeds.

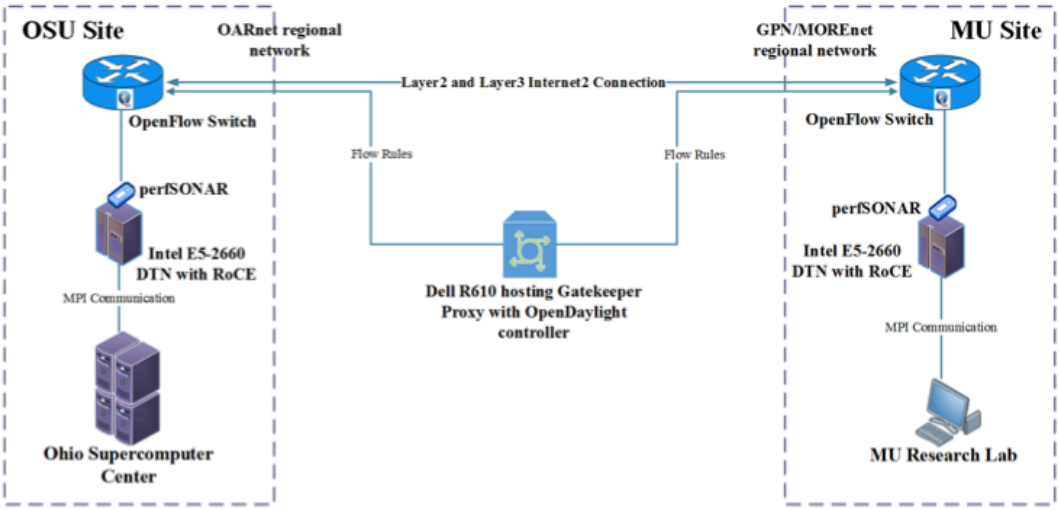

Figure 17: MU-OSU Science DMZ setup

5.2 Dynamic Control of SoyKB Application Flow

Unlike the real-time applications such as Neuroblastoma or video streaming whose goals are to reduce the jitter in order to provide better user experience, the SoyKB application focus is more on the time taken for the data transfer and data analysis across the distributed computing resource sites. Regarding the testbed configuration, we set the TCP buffer size to 256 MB on both sides of MU and OSU data transfer nodes. Prior to the experiment, we made sure that the TCP buffer size and processor input queue sizes were big enough to ensure that our path selection decision is solely dependent on the available bandwidth.

Figure 18 shows the orchestration timeline of data-intensive applications as seen by our ADON along with periodically sampled TCP throughout measurements. At time t1, the SoyKB application workflow is initiated, whose QSpecs requires at least 700 Mbps of single TCP throughput in the event that iRODS is not

able to create more than 1 thread. Predictably, the 10 Gbps Internet2 AL2S network connection between MU and OSU has significantly better throughput (around 5.5-to-6 Gbps) than the regular Layer 3 network connection with only around 650 Mbps achievable throughput when there are no other contending applications. Since ADON can get updated information from our Narada Metrics SDN monitoring system, it makes the decision to use AL2S for SoyKB data transfer, and calls a function in the OpenFlow controller for transferring traffic to a specific VLAN created for AL2S connection with a higher priority.

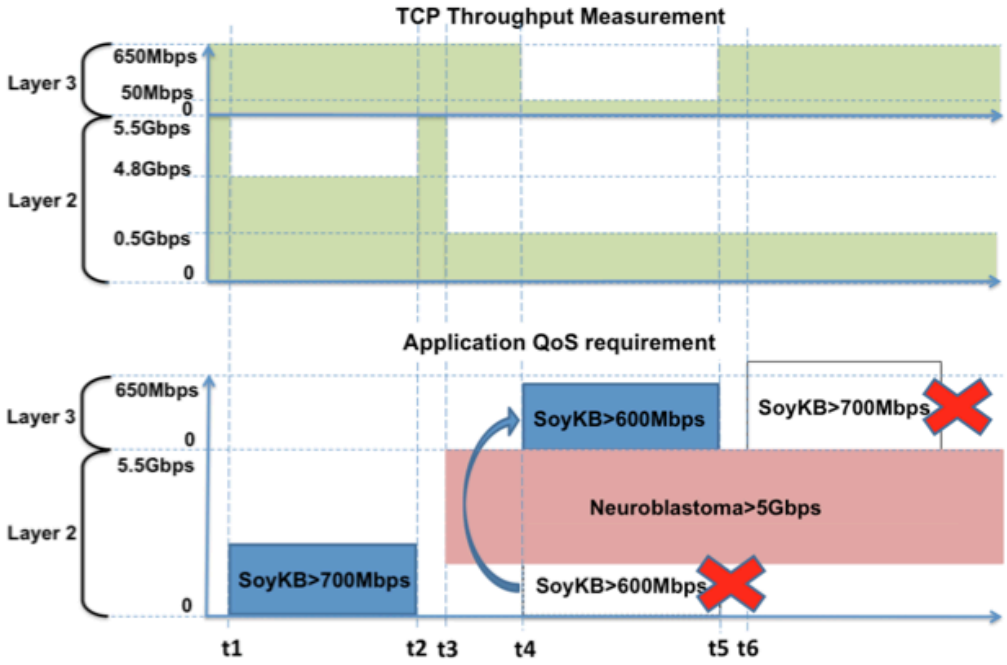

Figure 18: Timeline of OpenFlow switch handling SoyKB application workflows based on application QSpecs and periodic throughput measurement

Figure 19 shows the overall performance of the SoyKB transfer with different data sizes tested during the experiment run period from t1 to t2. The figure shows that due to absence of any other contending traffic,

the SoyKB flow can use the entire AL2S bandwidth achieving close to the highest achievable TCP throughput of 5.5 Gbps with the total transfer time in the order of tens of seconds. After t1, the achievable TCP throughput measured by Narada Metrics becomes about 4.8 Gbps from the original 5.5 Gbps, since the bandwidth test applications will compete against the SoyKB applications for the network bandwidth resource. At time t3, Neuroblastoma workflow which requires extremely high throughput and higher flow priority starts and the single TCP stream throughput test only shows around 500 Mbps, which does not meet the QSpecs of another SoyKB application workflow started at time t4 with at least 600 Mbps throughput requirements. That is when the ADON algorithm recalculates new estimated transfer times and achievable TCP throughput in Layer 3 (which currently had no other contending data-intensive traffic), and decides to use Layer 3 for SoyKB data transfer by making the priority of rule for the AL2S VLAN lower than the rule for Layer 3.

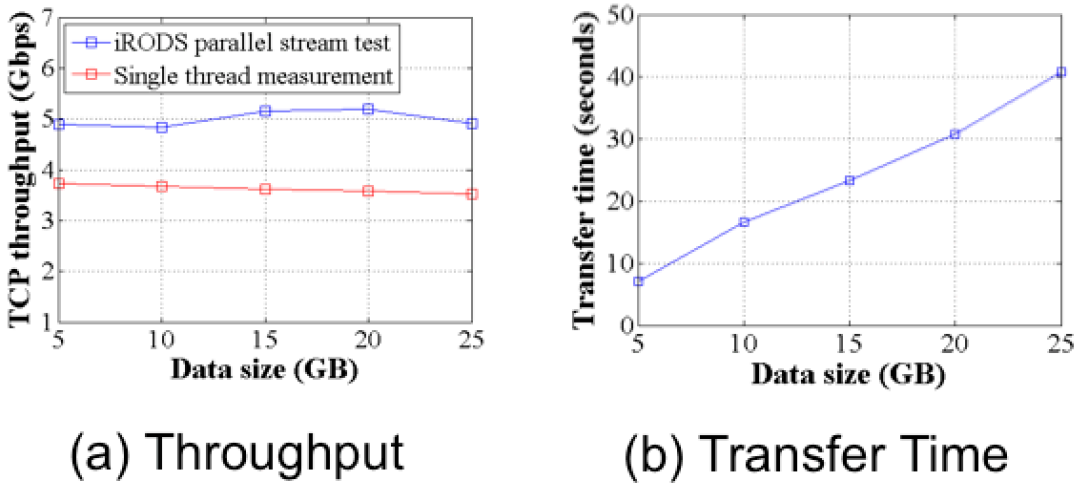

Figure 19: SoyKB Layer 2 performance without contending traffic

Figure 20 shows the performance of SoyKB application transfer without any other contending data-intensive flows in Layer 3. The data throughput reduced when compared to the Figure 19. However counter-intuitively, it still has considerably higher performance if we continued the SoyKB data transfer through Layer 2. The transfer performance in such a scenario is shown in Figure 21 where the SoyKB data is transferred through Layer 2 along with the contending Neuroblastoma traffic and does not even meet the minimum SoyKB QSpecs. Thus through intelligent provisioning with ADON service, we can save more than 100 seconds when transferring a 25 GB size file. In bioinformatics, it is very important for the users to transfer and analyze the data as soon as possible, and the time difference becomes very significant since the actual research data size can be up to several TBs for important applications such as the SoyKB. When we initiate another SoyKB application workflow requiring at least 700 Mbps single throughput at time t6, the ADON algorithm will reject the transfer task and push it to the Flow Scheduler, since neither network links can meet the QSpecs of the new workflow.

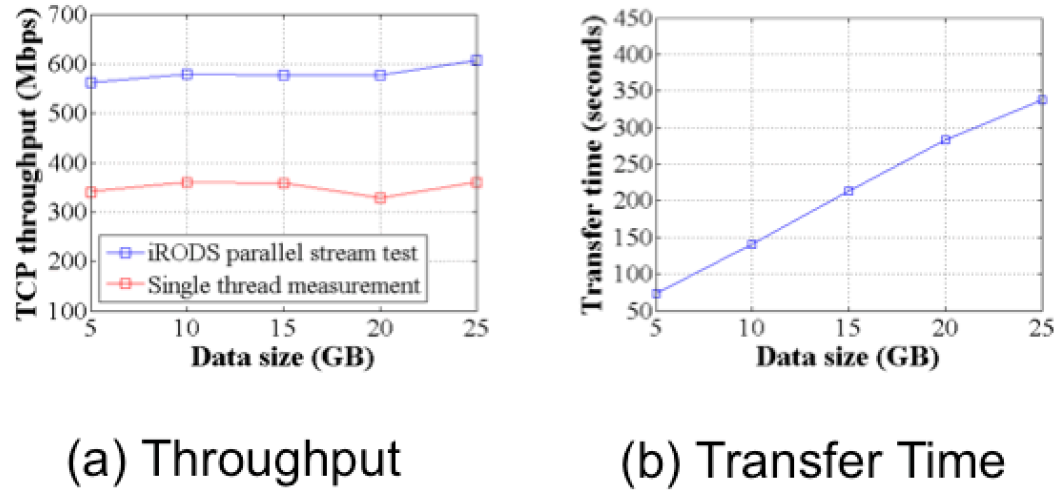

Figure 20: SoyKB Layer 3 performance without contending traffic

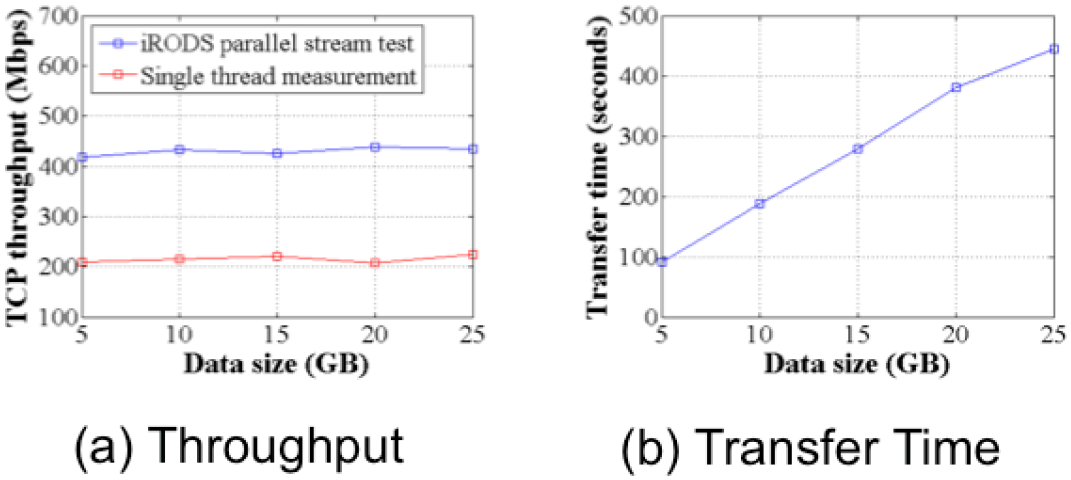

Figure 21: SoyKB Layer 2 performance with contending traffic

5.3 Additional improvement for SoyKB

Next, we perform additional experiments to demonstrate the performance improvements after the TCP tuning process. While performing above experiments, we found that the network infrastructure does not provide full advertised throughput due to protocol overhead, physical distance between the source and destination data transfer nodes, and end-point hardware limitation related issues. The MU-OSU AL2S link, which advertises 10 Gbps capacity, only had a maximum TCP throughput of 6 Gbps. When using tools such as iRODS that support parallel streams, or have multiple data transfers in parallel, we can configure the system settings such as Linux auto tuning TCP buffer limit and length of the processor input queue in both the MU and data transfer nodes to achieve closer to maximum TCP throughput in order to have fair sharing between flows. We found that it is important to change the configurations at both the source and destination edge nodes in order to ensure that the parallel stream setting performs at peak rates.

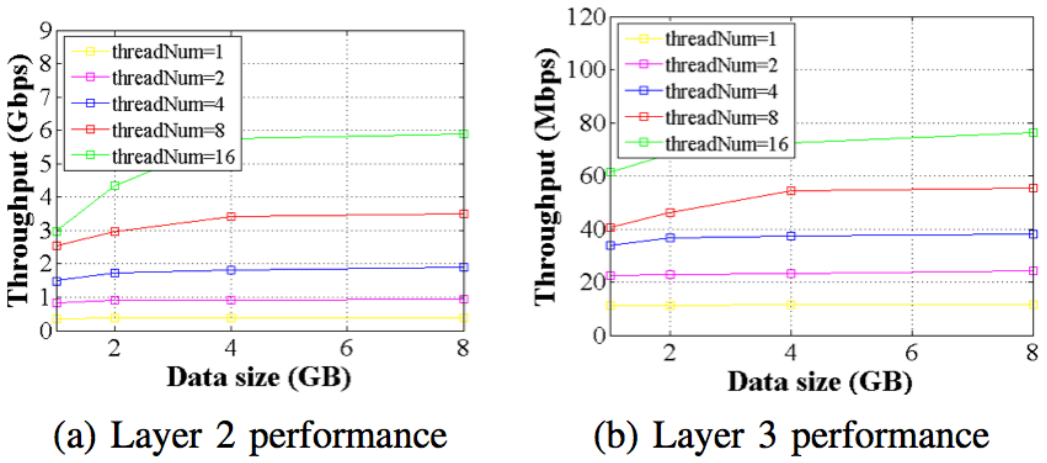

Figure 22: SoyKB transfer performance with different thread counts

Through Figure 22, we observe that the overall throughput increases when we use an increased number of TCP threads to move the data. However, the current iRODS system limits the maximum number of parallel threads to 16. Therefore, to achieve higher speed in next-generation networks, such as 100 Gbps Internet2 infrastructures, the iRODS system might need to support an increased number of parallel TCP threads. Increased number of TCP threads cannot always guarantee more throughputs in some cases, such as when the client side's bandwidth is scarce and single thread is able to consume all the available bandwidth. Too many open TCP connections can even degrade the performance since they will compete for the limited bandwidth causing protocol overheads and consume a lot of memory in the server side compute nodes, which we assume the reason the iRODS system made this limitation. Nevertheless, ADON capitalizes on the information regarding the number of threads that can be effectively used for SoyKB data throughput acceleration within the given Science DMZ environments.

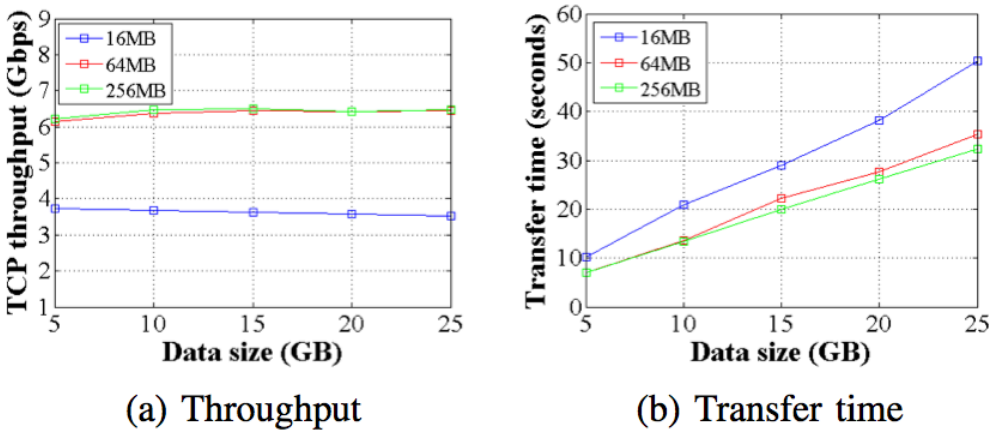

Figure 23: SoyKB Layer 2 transfer performance with different TCP buffer sizes

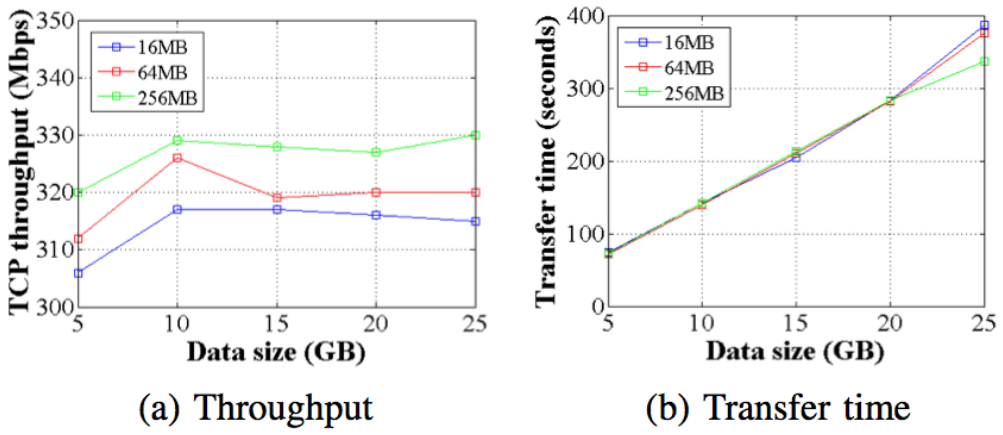

Figure 24: SoyKB Layer 3 transfer performance with different TCP buffer sizes

Figure 23 and 24 show the performance of Layer 2 and Layer 3 network connections with different TCP buffer sizes configured in both MU and OSU edge data transfer nodes. For both the cases, we keep the thread count at 16 in order to compare the respective maxima. In Layer 3, we were not able to improve the actual data throughput since the theoretical maximum throughput after TCP tuning in the data transfer nodes exceeds the throughput that the physical regular IP network infrastructure can provide. In contrast, when we increase the TCP buffer size to 64 MB from the original 16 MB, the data throughput increases by almost 50% in Layer2. The maximum throughput of Layer 2 is greater than the theoretical maximum TCP throughput with 16 MB TCP buffer size, which indicates that there is significant room for improvement by using multiple threads for TCP transfer with bigger buffer sizes (as shown with our experiments with increments of buffer size up to 256 MB). Hence, we can conclude that no further significant improvement of network performance can be obtained while transferring data through the 10 Gbps Layer 2 link beyond using the 64 MB TCP buffer size. Such a knowledge of the achievable peak performance over a certain

wide-area overlay network path can be updated within the corresponding ADON custom template, and can be used with the SDN monitoring system to manage expectations of the peak performance achievable for a given application workflow

6. CONCLUSION AND FUTUREWORK

In this thesis, we presented a new hybrid cloud architecture for SoyKB data flow that utilizes the software-defined measurement and software-defined networking referencing ADON architecture, in order to speed up the data transfer, provide the performance visibility to the users and control the resource dynamically based on the performance measurement analysis. We focus on how to utilize the performance measurement data to support cost optimized selection of sites for computation and dynamic transit selection for network engineering. Our testbed validation featured high-performance networking capabilities such as OpenFlow switches as well as local and remote HPC resources for SoyKB application usecase. We also showed the implementation of the SoyKB workflow performance monitoring within our software-defined measurement system that provides the performance visibility to the SoyKB researchers and some additional performance improvements with TCP tuning process we found in the previous experiments, which significantly increase the data throughput.

The major contributions of the author's work are as follows:

1. A hybrid cloud architecture for SoyKB data flow that utilize the software-defined measurement and software-defined to support dynamic resource control.

2. An effective network transit path selection mechanism based on real time network performance and user preference to determine the best transits for individual application flows.

3. Implementation of intelligent measurement scheduling broker for Narada Metrics software-defined measurement system that solves active measurement conflict problem to ensure accuracy for active measurements.

4. Implementation of end-to-end SoyKB workflow measurement as an extensible custom metric Narada Metrics to provide performance visibility to the SoyKB researchers and intelligence to resource broker technologies such as OpenStack, HTCondor and Pegasus.

5. Validation of ADON approach with SDM-SDN scheme with sufficient experiments in wide-area overlay network testbed across two campuses for SoyKB usecase that can be replicated and beneficial to other science big data applications.

As part of our future work, we are planning collect more compute-related performance metrics data from both local and remote compute resources, conduct experiments to utilize these metrics data to maximize the overall SoyKB throughput as soon as HPC resources at MU local side are ready to use.

7. BIBILOGRAPHY

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, “The Science DMZ: A Network Design Pattern for Data-Intensive Science”, Proc. of IEEE/ACM Supercomputing, 2013.

[2] H. Yin, Y. Jiang, C. Lin, Y. Luo, Y. Liu, “Big Data: Transforming the Design Philosophy of Future Internet”, IEEE Network Magazine, 2014.

[3] Yu, Minlan, Lavanya Jose, and Rui Miao. "Software Defined Traffic Measurement with OpenSketch." NSDI. Vol. 13. 2013.

[4] N. McKeown, T. Anderson, H. Balakrishnan, et. al., “OpenFlow: Enabling Innovation in Campus Networks”, ACM SIGCOMM Computer Communication Review, Vol. 38, No. 2, 2008.

[5] Pfaff, B. "OpenFlow Switch Specifications 1.0. 0.", 2009.

[6] P. Calyam, S. Kulkarni, A. Berryman, K. Zhu, M. Sridharan, R. Ramnath, G. Springer, “OnTimeSecure: Secure Middleware for Federated Network Performance Monitoring”, IEEE Conf. on Network and Service Management (CNSM), 2013. (https://www.naradametrics.com)

[7] A. Hanemann, J. Boote, E. Boyd, et. al., “perfSONAR: A Service Oriented Architecture for Multi-Domain Network Monitoring”, Proc. of Service Oriented Computing, 2005.

[8] Calyam, Prasad, et al. "Ontimedetect: Dynamic network anomaly notification in perfsonar deployments." Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on. IEEE, 2010.

[9] Calyam, Prasad, et al. "Enhanced EDF scheduling algorithms for orchestrating network-wide active measurements." Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International. IEEE, 2005.

[10] S. Seetharam, P. Calyam, T. Beyene, “ADON: Application-Driven Overlay Network-as-a-Service for Data-Intensive Science”, Proc. of IEEE Cloud Network- ing, 2014.

[11] R. Morgan, S. Cantor, S. Carmody, W. Hoehn, K. Klingenstein, “Federated Security: The Shibboleth Approach”, EDUCAUSE Quarterly, 2004.

[12] P. Calyam, S. Seetharam, R. Antequera, “GENI Laboratory Exercises Development for a Cloud Computing Course”, Proc. of GENI Research and Educational Experiment Workshop, 2014.

[13] Ryu - Component-based software-defined networking framework; http://osrg.github.io/ryu

[14] T. Joshi, M. Fitzpatrick, S. Chen, Y. Liu, H. Zhang, R. Endacott, E. Gaudiello, G. Stacey, H. Nguyen, D.Xu, “Soybean Knowledge Base (SoyKB): A web resource for integration of soybean translational genomics and molecular breeding”, Nucl. Acids Res, 2014.

[15] Information Science Institute - http://www.isi.edu

[16] Texas Advanced Computing Center - https://www.tacc.utexas.edu

[17] Extreme Science and Engineering Discovery Environment - https://www.xsede.org

[18] S. Goff, et al., “The iPlant Collaborative: Cyberinfrastructure for Plant Biology”, Frontiers in Plant Science, 2011.

[19] Integrated Rule-Oriented Data System (iRODS) - http://irods.org

[20] E. Deelman, J. Blythe, et. al., “Pegasus: Mapping Scientific Workflows onto the Grid”, Springer LNCS - Grid Computing, 2004.

[21] HTCondor Resource Manager - http://research.cs.wisc.edu/htcondor

[22] OpenStack - http://www.openstack.com

[23] ESnet Fasterdata Knowledge Base - http://fasterdata.es.net/host-tuning/linux

[24] P. Calyam, A. Berryman, E. Saule, H. Subramoni, P. Schopis, G. Springer, U. Catalyurek, D. K. Panda, "Wide-area Overlay Networking to Manage Accelerated Science DMZ Flows", Proc. of IEEE ICNC, 2014.

[25] P. Lai, H. Subramoni, S. Narravula, A. Mamidala, D. K. Panda, "Designing Efficient FTP Mechanisms for High Performance Data-Transfer over InfiniBand", *Proc. of ICPP*, 2009.