A Computational Model to Support In-Network Data Analysis in Federated Ecosystems

Ali Reza Zamani[1], Mengsong Zou[1], Javier Diaz-Montes[1], Ioan Petri[2], Omer Rana[2], and Manish Parashar[1]
[1] Rutgers Discovery Informatics Institute, Rutgers University, USA
[2] School of Computer Science & Informatics, Cardiff University, UK

Abstract

Software-defined networks (SDNs) have proven to be an efficacious tool for undertaking complex data analysis and manipulation within data intensive applications. SDN technology allows us to separate the data path from the control path, enabling in-network processing capabilities to be supported as data is migrated across the network. We propose to leverage software-defined networking (SDN) to gain control over the data transport service with the purpose of dynamically establishing data routes such that we can opportunistically exploit the latent computational capabilities located along the network path. This strategy allows us to minimize waiting times at the destination data center and to cope with spikes in demand for computational capability. We validate our approach using a smart building application in a multi-cloud infrastructure. Results show how the in-transit processing strategy increases the computational capabilities of the infrastructure and influences the percentage of job completion without significantly impacting costs and overheads.

Keywords: software-defined networks, in-transit, smart buildings, cloud federation, CometCloud

1. Introduction

There has been recent interest in moving away from centralized, large-scale data centers to a more distributed multi-cloud setting (as demonstrated by significant interest in cloud federation and interoperability efforts). Such a multi-cloud environment is often formed by a network of smaller virtualized infrastructure runtime nodes with an unstructured architecture. On the other hand, network providers are increasingly becoming potential sources of general purpose computation. They are minimizing the amount of network-specialized hardware hosted in their data centers and moving towards the use of commodity hardware. This strategy follows state of the art networking approaches, such as Software-defined networking (SDN) and Network Functions Virtualization (NFV). Software-defined networking (SDN) in particular is an approach devised

to simplify network management through abstraction of lower-level functionality. Specifically, SDN separates control plane (where to send data) from data plane (data forwarding functions). This enables the software-based control plane to be run on commodity servers and to leverage the latest-generation of processors, which are faster than embedded-class processors in most switches [1]. On the other hand, NFV goes a step further and extends the as-a-service cloud model to offer networking functions on-demand using virtualization techniques. The key reason for using virtual machines (VMs) is the possibility of elastically scaling functions by simply adding or removing VMs based on data workload characteristics. This approach promises, as the cloud, a reduction in capital expenses and fast delivery of new functionality. As in the case of SDN, this approach is also implemented on commodity hardware [2].

Data centers managed and operated by network providers form a significant part of the current Internet infrastructure, as there is a large number of such data centers that are almost ubiquitous across the world. These data centers may not be as powerful as computational data centers, hosted by cloud providers or traditional high performance computing (HPC) providers. However, their ubiquity and the fact that we have to necessarily use them when moving data over the Internet, make them a very interesting source of pervasive computing at the edge of the network. Understanding how the availability of commodity servers within such "network data centers" can contribute towards data processing would enable an effective way to extend the boundaries of a cloud system – from a high end, often localized data center, to multiple distributed data centers that can process data while it is in transit from source to destination.

In this paper we propose a model to leverage the use of computational capabilities within such network data centers to offer general computation services co-located with network services. In this way, we can use more efficiently the resources of these network data centers while providing an extra source of revenue for those who operate and manage them. Hence, spare capacity within such network data centers can be more efficiently utilized and monetized. We extend the network controller capabilities to not only offer information about the network topology but also to identify sources of computation. We envision an ensemble of network data centers that can optimize the data routes based on flows and offer *in-transit* computational capabilities.

The rest of the paper is organized as follows. Section 2 presents our motivating use case. Section 3 presents our in-transit computational model, followed by the proposed in-transit optimization strategies in Section 4. Section 5 defines the problem of allocating workload using our computational model. Section 6 presents the implementation of our in-transit computational model. Section 7 describes evaluation and results. Section 8 collects the related work. Finally, Section 9 presents the conclusions and ongoing activities.

2. Motivating Use Case

An instrumented built environment, which can consist of single/multiple buildings (homes, office buildings, sports facilities, etc), provides a useful scenario to validate the use of in-transit analysis capabilities. Depending on the number of sensors within a single building, the frequency at which data is captured from such sensors and the particular data analysis objective (e.g. reduce energy consumption, improve efficiency of HVAC (heating, ventilation and air condition) function, improve comfort levels based on occupancy, etc), the computational capability requirements can vary significantly. In some instances such data is often analyzed off-line (in batch mode) to enable improvements in building design or to support long term facilities management. In other instances (evidenced by recent use of such instrumented environments), real time analysis needs to be carried out (over intervals of 15 to 30 minutes generally) to enable better energy efficiency and use of such infrastructure. When multiple such buildings are considered (e.g. within a business park, University campus or a housing association), the overall computational requirement can increase considerably.

In order to maintain a comfortable living environment, it is often necessary to consider multiple objectives that may have conflicting targets, e.g. minimum energy consumption, minimum CO_2 emission, or maximum comfort level. Optimization for the building operation stage (which can also include facilities management) requires different approach compared to the building design stage, e.g., some key design variables can no longer be changed (to find the most optimum solutions for design). It needs to take the as-built building environment to find the optimum solutions either against single or multi-objectives. To provide practical real time decision making in building energy management based on real time monitored data, it is necessary to develop a 'behaviour' of a building energy system by using various simulation tools. During the process, domain experts are often involved in order to identify the main use cases and scenarios with associated input parameters and feasible outputs. In the modelling process, different relevant components have to be assessed and calibrated iteratively, and the developed building energy simulation model is then executed (as the calculation engine) within a generic optimization program. In this work we execute multiple EnergyPlus[1] instances, a software that requires significant computational resources to run, with different input parameter ranges.

Various types of sensors are used to monitor energy efficiency levels within a building, such as: (i) solid-state meters for accurate usage levels, (ii) environmental sensors for measuring temperature, relative humidity (RH), carbon monoxide (CO), and carbon dioxide (CO_2) levels, (iii) temperature measurements using both mechanical (e.g., thermally expanding metallic coils) and electrical means (e.g., thermistors, metallic resistance temperature detectors (RTD), thermocouples, digital P-n junctions, infrared thermocouples) to pro-

[1] http://apps1.eere.energy.gov/buildings/energyplus/

vide sufficient accuracy. When dealing with large buildings such as sports facilities, the accuracy of these sensors is often questionable, largely because of the significant drift that occurs after initial calibration. In some buildings, there are specific requirements for sensors when monitoring CO_2 concentration, air flow, humidity, etc and these sensors are more expensive to use and deploy. We use sensor data from the $SportE^2$ project pilot called FIDIA [3], a public

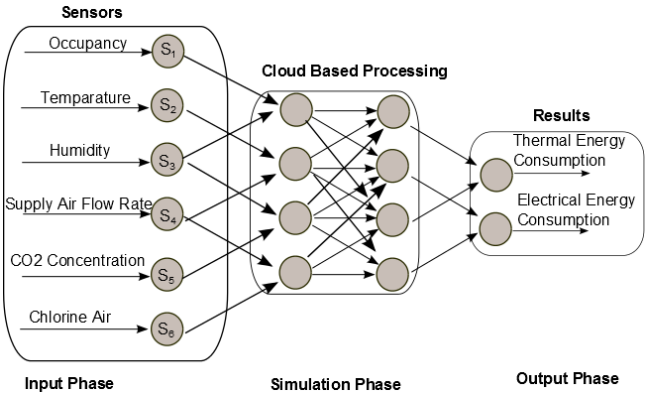


Figure 1: Energy optimization scenario.

sports building facility, located in Rome, Italy. $SportE^2$ is a research project co-financed by the European Commission FP7 programme under the domain of Information Communication Technologies and Energy Efficient Buildings. This project focuses on developing energy efficient products and services dedicated to needs and unique characteristics of sporting facilities. The building has metering capability to determine consumption of electricity, gas, biomass, water and thermal energy. This data can be accessed through a specialist interface and recorded for analysis. The sub-metering of thermal and electrical consumption within grouped zones (gym/fitness and swimming pool is also provided along with "comfort" monitoring by functional area: gym, fitness room and swimming pool). In these areas the Predicted Mean Vote (PMV) index is used (which measures the average response of a group of people to a thermal sensation scale – such as hot, warm to cool and cold) – it is one of the most widely recognized thermal comfort models, and is measured as a function of the activity performed within a particular part of the building. Additional details can be found in [4].

3. In-transit Computational Model

Novel networking approaches, such as SDN and NFV, require a central orchestrator or controller that manages routing tables and other network functions. Such orchestrator has knowledge of the overall network architecture that it is used to, for example, optimize traffic routes based on utilization and available bandwidth. We propose to extend such a controller to enable the management of computational capabilities available at each network data center.

The idea is to collect information about the available computational capabilities together with the rest of the network information (e.g., traffic, bandwidth, etc). Since networking is a critical service, its infrastructure is typically over-dimensioned to easily deal with spikes in demand and failures [1]. Hence, we could enable general purpose computation on network data centers by offering the idle capacity of their resources to application users. For example, we could share commodity hardware by deploying different types of VMs that isolate general purpose computation from network functions. In this way, we could easily displace or terminate general purpose computation if resources are required to support network operations.

Offering computational capabilities in a scalable manner requires having a service local to each data center that provisions computational resources and allocates workload onto such resources. In its simplest instance, this service could be a queue that allocates computation onto VMs and interacts with the service that allocates network functions to prevent interference. More advance computation services can involve creating a virtual staging area which allows data to be diverted directly from the network to local resources for computation and subsequently placing it back into the network after processing [5].

Due to limited computational capabilities, network data centers are not intended to perform large computations but rather perform small operations on data that is moving through the network. This could be the basis for new business models that not only consider bandwidth reservation but also the computation that can be performed along the data path. For example, a client may require certain amount of bandwidth and latency to satisfy his/her Quality-of-Service (QoS – e.g. throughput in a workflow), but due to the current status of the network it may not be possible to satisfy such requirements within some pre-defined constraints. However, a controller may estimate that QoS could still be satisfied with lower bandwidth and latency requirements, if in-transit computation is carried out when the data is moving from source to destination. Although data is moving slower through the network, certain results will be available when it reaches the destination. Hence, it would be possible to maintain the QoS of the system (e.g., throughput) by utilising spare capacity within network data centers.

4. In-transit Optimization Approach

We consider a marketplace where a manager within a built environment needs to decide whether a given workload should be computed using local resources or it should be outsourced to a remote site. Typically, computation within a single building may be carried out locally – generally utilizing computational resources that are geographically local to the sensors producing the data. However, when multiple buildings are considered, such as in a smart city scenario, it may be necessary to outsource computation to another sites. This enables computational resource sharing, which not only improves resource utilization, but also increases the resilience to failures and peaks in the demand of the whole infrastructure. Unlike in the single building case, when outsource

workload to external sites, it necessary to consider both the computational and data transfer costs. Additionally, we need to identify where to compute the workload and how to take advantage of in-transit computation, by leveraging computational power embedded within the network infrastructure (e.g., SDN switches, network data center). We consider that we have an SDN Controller that knows the topology of the network as well as the computation offered by each node, as defined in previous section.

In our scenario we consider that the service level agreement (SLA) of a job (in this instance, an energy simulation with a particular set of input parameters) includes a deadline, a budget, and a minimum job completion ratio. Using terminology common in distributed systems, we differentiate between a job and a task. A job contains multiple tasks, and the job completion ratio measures the total number of tasks completed within a given deadline compared to the total number of tasks that make up a single job. In our marketplace, various resource providers are asked to send their offers given a SLA. The building manager or client evaluates all offers and selects the one that maximizes the utility of his/her decision function. For example, a client may select the provider that offers a higher completion ratio of a job, subject to deadline and budget constraints. In this paper we devise two strategies to improve the utility of the decision function by leveraging in-transit computation.

Traditional client: This strategy allows clients to transparently improve the utility of their decision function while operating as always. In this strategy, the client continues to rely on computational data centers (resource providers) to ensure his/her QoS are met. Hence, this client does not wish to include in his/her decision function the possibility of performing in-transit computation. In this case only the resource providers that can meet the required SLA offer their services to the client. In this way the client is certain that his/her SLA can be met and a decision is taken using the information of those offers. Once the decision has been made, the controller can optimize the route from client to the selected resource provider and allocate in-transit resources along the data path. In this instance, in-transit computation is carried out without the client's intervention – it is transparent to such client.

In-transit aware client: This strategy aims at maximizing the utility of a client's decision function by taking into account all sources of computation from source to destination. Thus, we envision this strategy being used by a client that wishes to influence his/her decision by the computational capabilities available in-transit, along the data path, and at the destination resource provider. As opposed to the previous strategy, in this case the client receives offers from all resource providers interested in computing the workload, even from those that cannot meet the required SLA by themselves. Next, the client asks the controller to optimize the route to each one of these resource providers. Once all the information is collected and evaluated by the client, a decision that maximizes the utility of the decision function and meets the required SLA can be taken considering all sources of computation in the system as well as the optimized routes to them.

Both in-transit optimization strategies are subject to two constraints. First, the time that can be allocated to in-transit computation is limited by the time at which data must reach the destination (i.e. the computation is scheduled to start). We assume that the destination resource provider can give us an estimation of the time when the job is scheduled to start. Although we are using cloud services, a resource provider may delay the execution of a job if it considers that the SLA can be met (e.g., complete the required number of tasks) within the deadline. The second constraint is that the cost of in-transit computation plus the cost of computation at the destination provider cannot exceed the budget. One must consider that if in-transit computation is performed, then less computation needs to be done at the destination resource provider.

Next, we formalize the in-transit optimization problem proposed in this paper. The problem to optimize is similar for both strategies, the only difference is that in the first case we optimize a single time for the selected resource provider, while in the second case we optimize for all candidate resource providers.

5. Problem Definition

A client needs to compute a job J, composed of k tasks $\{j_1, ..., j_k\}$, that is generated at the client's location – this location is defined as source s. In case the client wishes to outsource a job, it is necessary to identify where to compute the workload and how to take advantage of in-transit computation. The place to compute the workload may be a remote resource provider – its location is defined as destination d. The service level agreement (SLA) of a job J includes: a deadline ($Deadline(J)$) by which results have to be returned to the client; a budget ($Budget(J)$) that sets the maximum amount available to spend on computing job J; and a minimum job completion ratio $CRatio(J)$ defined as the ratio between the k' completed tasks and the overall number of tasks k, where $k' \leq k$, of job J.

We define two types of computational resources, namely computational data centers (resource providers or sites) and network data centers (in-transit resources). Resource providers or sites collectively form a federation and are the main sources of computation in our infrastructure. When outsourcing workload in our scenario, these sites represent the source s and the destination d. On the other hand, network data centers offer resources at the edge of the network. We consider that we have a set of q network data centers $R : \{r_1, ..., rq\}$. Both sites and network data centers are equipped with SDN routers to ensure control over the network. We consider that there is some waiting time $W(J)$ before a job J can be executed at resource provider d. During this time, the job is idle and it is using storage space at the destination resource. Hence, we would like to identify and configure a data path that leverages in-transit computation to take advantage of $W(J)$ for a job. We define the following variables:

- $P(r_i)$ is the average number of tasks that resource r_i completes per unit of time.

- $E(r_i)$ is the amount of time spent computing in resource r_i.
- $CE(r_i)$ is the cost per unit of time of using resource r_i for computation.
- $T(r_i, r_k)$ is the amount of time spent transferring data between resources r_i and r_k.
- $CT(r_i, r_k)$ is the cost of reserving a network channel per unit of time.
- $W(J)$ is the waiting time before job J can start its computation at destination resource.
- $CRatio(J)$ is the completion ratio of job J, i.e. the ratio between completed tasks and total number of tasks composing job J.

To leverage in-transit computation and minimize the amount of time a job is idle at destination, the objective of our problem becomes maximizing the amount of tasks completed in-transit, which is defined as follows:

$$\max \sum_i P(r_i) * E(r_i) \tag{1}$$

subject to being ready to compute at destination resource d at the scheduled time (2), performing computation within the given deadline (3), keeping costs within the given budget (4), and making sure that the completion ratio is satisfied (5):

$$\sum_i E(r_i) + Transfer(J) \leq W(J) \tag{2}$$

$$\sum_i E(r_i) + Transfer(J) + E(d) \leq Deadline \tag{3}$$

$$Cost(J) \leq Budget \tag{4}$$

$$\frac{\sum_i [P(r_i) * E(r_i)] + P(d) * E(d)}{k} \geq CRatio(J) \tag{5}$$

where the $Transfer(J)$ is the overall transfer time of a job, which is defined as the sum of the time spent transferring data from source (s) to first network data center (r_i), the sum of the time spent transferring data between network data centers $\in R$, and the time spent transferring data between the last network data center (r_k) and destination (d):

$$Transfer(J) = T(s, r_i) + \sum_i^q \sum_{k \neq i,k}^q T(r_i, r_k) + T(r_k, d) \tag{6}$$

$Cost(J)$ is the overall cost of computing job J, defined as:

$$Cost(J) = CostExecMid + CostExecDest + CostNet \quad (7)$$

where the cost of computing in-transit ($CostExecMid$) is defined as:

$$CostExecMid = \sum_{i} [CE(r_i) * E(r_i)] \quad (8)$$

the cost of computing at the destination resource d ($CostExecDest$) is defined as:

$$CostExecDest = CE(d) * E(d) \quad (9)$$

and the cost of transferring data associated with a job ($CostNet$) is defined as:

$$CostNet = T(s, r_i) * CT(s, r_i) + \sum_{i}^{q} \sum_{k \neq i,k}^{q} [T(r_i, r_k) * CT(r_i, r_k)] + T(r_k, d) * CT(r_k, d) \quad (10)$$

subject to $E(r_k) \neq 0$. Note that the time and cost of returning results to the client is negligible as only a few parameters are sent.

6. Implementation

We extend our federation model [6] to expose in-transit capabilities to participant sites of the federation. Figure 2 shows our architecture. We include a service, called Controller, that is aware of the network topology, using SDN technology, and also has information about the available computational capabilities of each network data center. Each network data center has a SDN router and a set of computational resources – referred as in-transit resources in Figure 2. This service can be consulted by different federated sites to optimize workload scheduling using the strategies proposed in Section 4.

This federation model is built using the CometCloud framework [7]. CometCloud is an autonomic framework for enabling real-world applications on software-defined federated cyberinfrastructure, including hybrid infrastructures integrating public and private Clouds, data-centers and Grids. The CometCloud federation is created dynamically and collaboratively, where resources/sites can join or leave at any point, identify themselves (using security mechanisms such as public/private keys), negotiate terms of federation, discover available resources, and advertise their own resources and capabilities [8].

CometSpace [9] provides tuple-space like abstraction for coordination and messaging in the federation model. Specifically, we define two types of coordination spaces. First, a single management space (*CometCloud Federation Space*)

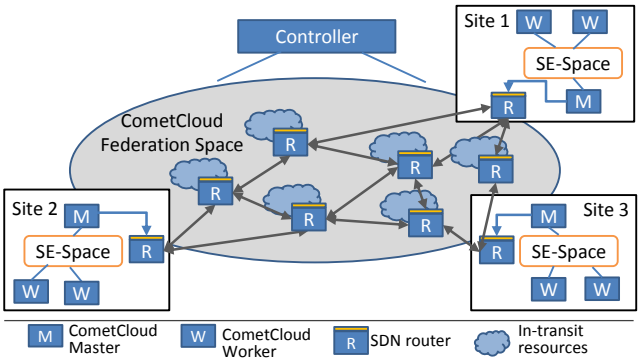


Figure 2: Federated architecture that exposes in-transit capabilities.

spans across all resource sites creating and orchestrating the federation. Second, multiple shared execution spaces (*SE-Space*) are created on-demand during application workflow executions to satisfy computational or data needs. Execution spaces can be created within a single resource site, or can burst to others, such as public clouds or external HPC systems.

7. Evaluation

7.1. Configuration of Testbed

We deployed our federation model on the Amazon EC2 cloud platform. In our experiments, we used a total of 8 VM instances that emulated different geographically distributed sites. 3 VMs represented site resources: Site1, Site2, and Site3. The other 5 VMs were in-transit resources: Mid2, Mid3, Mid4, Mid5, and Mid6 – located between sites, see Figure 3. As described in section 4, our model considers that we can have computational resources (either from a resource provider or from a network data center) co-located with each SDN router. An overview of the deployed testbed is depicted in Figure 3. We considered a geographically distributed infrastructure, which was emulated by configuring different network bandwidths connecting the sites. Specifically, three in-transit resources were located near Site3, the other two were located near Site2, and all of them were located at identical distance from Site1. This network configuration is inspired in data obtained from previous experiments [4, 8].

In this work, we considered three different infrastructure scenarios: (i) Base: in-transit resources and sites have the same computational power; (ii) Higher: in-transit resources are less powerful than those at the resource providers' sites; and (iii) Highest: in-transit resources are much less powerful than site resources. The cost was set to be proportional to the computational capabilities in resources. Table 1 collects various scenarios using different types of EC2 VMs, and Table 2 collects the characteristics of the different types of VMs, based on Amazon EC2.

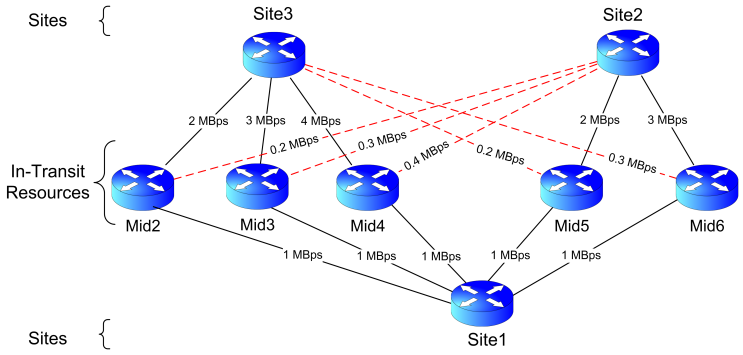


Figure 3: Infrastructure. Solid lines indicate high bandwidth links and dashed lines indicate slow bandwidth links. We assume each SDN router is co-located with computational resources.

Table 1: Infrastructure Scenarios.

Scenario	In-transit Resources	Site Resources
Base	c4.2xlarge	c4.2xlarge
Higher	c4.2xlarge	c4.4xlarge
Highest	c4.2xlarge	c4.8xlarge

All instances were deployed with Mininet [10] and the network between them was configured to emulate a SDN environment among these 8 mininet instances. Each VM had one mininet host and one mininet switch. Switches were connected to each other using Generic Routing Encapsulation (GRE) tunneling[11]. Bandwidth allocation for data links was implemented in the hosts using token bucket filter. Routing tables and connections were controlled by a POX SDN controller (POX is a python based SDN controller). We had an additional VM designated as the controller of the network. The controller managed network connections using two types of connections: (i) UDP was used for gathering information; and (ii) TCP was used for regular communication and establishing data paths. TCP rules for each switch were installed in a proactive manner. That is, every time a switch connected to the controller (i.e. when switch starts), the controller would install rules.

We implemented our in-transit optimization approach, described in Section 4, as follows. We defined our protocol for communication between the controller and hosts using UDP packets. We established that switches would

Table 2: Resource Properties

Resource Type	vCPU	ECU	Memory	Price ($/Hour)
c4.2xlarge	8	31	15	0.464
c4.4xlarge	16	62	30	0.928
c4.8xlarge	36	132	60	1.856

forward all UDP packets to the controller unless a specific destination was included in the packets. We used this rule to enable communication between client and controller. Specifically, when the source site (i.e. client) wanted to communicate with the controller, it would send a UDP packet without destination field instantiated. This packet would be automatically forwarded to the Controller. Upon receipt of this packet, the controller would send UDP packets to all in-transit resources (hosts) asking for their status and capabilities information. Since those UDP packets would have a specific destination, switches knew where to send them (i.e. in-transit resources). Then, the controller would gather all replies, analyze their information, and create a plan. This plan was returned to the client and included a data transfer path as well as information about the allocated in-transit computation.

For example, in the case of the building scenario, each building has a Building Manager Service (BMS) that periodically collects all the information from sensors of the building under its control. Hence, this BMS acts as a client in our infrastructure and needs to decide where the workload is computed and collects the results to establish new setpoints. Using the network configuration defined before, this BMS can contact resource provider sites as it would in a non-SDN scenario, e.g., via TCP socket communication. However, when interacting with the controller, the BMS simply needs to send an UDP message that is automatically forwarded to the controller. Using this approach, the use of SDN does not involve complex changes in the client and changes in the network are transparent as well.

We considered a use case where several smart buildings required analysis of their sensor data to optimize energy consumption. Based on the FIDIA pilot [3], we considered that these buildings could generate three type of jobs, as described in Table 3. In our previous work we focused on studying the cost of using SDN [6], in these experiments we focused instead on the feasibility of using in-transit computation to meet job completion deadlines. Hence we established a budget large enough to satisfy every decision, and the limiting factor was the deadline of the job. For all types of jobs, we established in the SLA that at least 60% of the tasks, within a job, must be completed before the deadline (to achieve a particular accuracy target). Hence, the decision function for each job was to maximize its completion ratio subject to the deadline and cost.

Table 3: Job Information.

JobType	Data Size(MB)	Budget	Deadline(s)	Tasks†
JobType1	10	20	120	10
JobType2	20	30	150	20
JobType3	30	40	180	30

† – A job is composed of a set of tasks

In each experiment, a total of 326 jobs were generated and inserted, using a Poisson distribution, from Site1 to a federated marketplace. Once a job was

inserted in the federation, different sites (i.e. Site1, Site2, and Site3) offered their services using a blind auction mechanism. The marketplace was based on the problem specification identified in Section 5, which established how to calculate costs and satisfy required SLAs.

We made use of two strategies: **No in-transit computation:** in these experiments only resource providers' sites can perform computation – labeled as "No In-Transit" in the experiment results. **In-transit computation:** in these experiments in-transit resources can contribute computational resources. We used both strategies defined in section 4:

- *Traditional client*, where in-transit optimization happens after a destination site (resource provider site) has been selected – labeled as "In-Transit" in the experiment results.
- *In-transit aware client*, where in-transit optimization is taken into account when selecting a destination site – labeled as "In-Transit2" in the experiment results.

7.2. EnergyPlus Optimization

In these experiments, we considered that jobs were computed using EnergyPlus. Table 4 shows the completion time for each job type when computed using EnergyPlus. EnergyPlus independently computes each one of the tasks composing a job, and it was assumed that all tasks within a job had similar completion time (i.e. job completion time divided by number of tasks composing job).

Table 4: Time to completion of EnergyPlus job types.

JobType	c4.2xlarge	c4.4xlarge	c4.8xlarge
JobType1	80 s	40 s	20 s
JobType2	100 s	50 s	25 s
JobType3	120 s	60 s	30 s

First, we evaluated the effect of our in-transit strategies on the job acceptance rate. In our experiments, the acceptance rate was limited by the ability of resources to complete jobs within the given SLA (i.e. deadline and completion ratio). We used three different infrastructure scenarios (see Table 2): Base, where in-transit resources were equivalent to provider resources (sites); Higher, where provider resources were twice as powerful as in-transit resources; and Highest, where resources were four times more powerful than in-transit resources.

Figure 4 collects the results of this first experiment. It can be observed that in-transit resources helped to increase the number of accepted jobs in the federation, when the computational capacity of resource providers (sites) was not enough to satisfy the overall workload. This was the case for scenarios Base, Higher, and Highest, as the no in-transit strategy shows. In these three cases, we observe an increase (with reference to the no in-transit case) of up to an

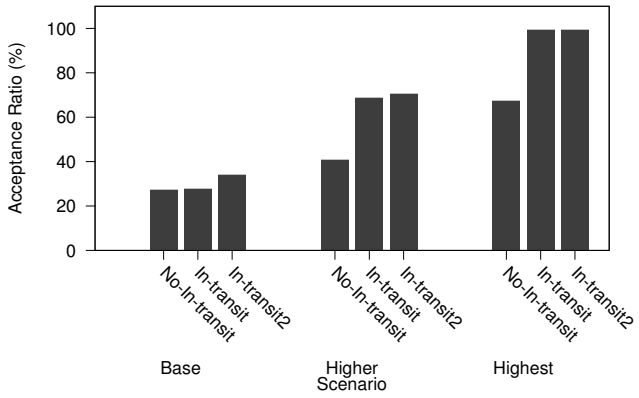


Figure 4: Job Acceptance Ratio. "in-transit" corresponds to the strategy *Traditional client* and "in-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

19% for the Base scenario, around a 41 % for the Higher scenario, and around 31% for the Highest scenario. The Highest scenario was able to accept all jobs submitted for the In-transit strategies (i.e. In-transit and In-transit2) due to the high-end capabilities of the federation sites.

In order for us to better understand the nature of these improvements, we collected the number of computations, expressed in core/hours, that each resource contributed to the federation. Figure 5 collects the overall normalized core/hours used for the different scenarios.

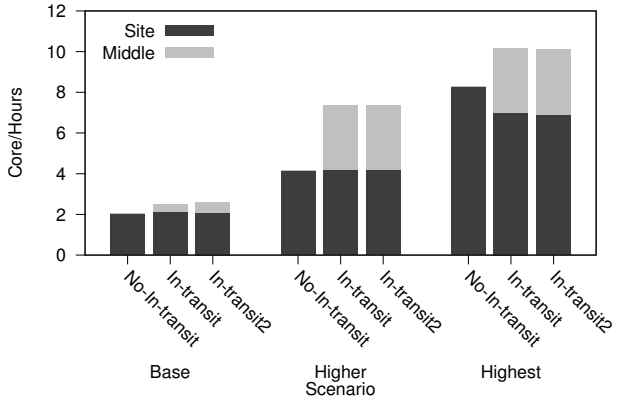


Figure 5: Normalized Core/Hours. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

The Base scenario shows that in-transit resources (labelled as Middle in Figure 5) contributed to increase the overall core/hours of the system between

17% and 21%, when compared with the case where no in-transit strategy was used. However, this additional core/hours does not significantly increase the acceptance ratio, as it can be observed in Figure 4. The main reason is that our strategies are focused on aiding resource providers (labelled as Site in Figure 5) and are influenced by the performance of these resources. In this scenario many jobs are rejected due to lack of computational capacity at the end sites, and therefore there were not many jobs with idle/wait time that could benefit from in-transit computation. On the other hand, in the Higher scenario, many more jobs were accepted by the federation, as in-transit resources were able to increase the available capacity (core hours) up to 44%, of which approximately 42 % was contributed by in-transit resources. Finally, the Highest scenario shows that most of the workload was computed by the resources owned by the provider. In this scenario in-transit resources represented approx. 18% of the overall computation.

Next we analyzed if in-transit computation can increase the completion ratio of accepted jobs. As previously explained, a job is composed of tasks and the SLA specifies the minimum completion ratio for a job to be accepted. However, from the client's perspective higher completion ratio means better accuracy of the resulting energy usage/efficiency model (thereby leading to greater potential energy savings and comfort levels within a building). Figure 6 shows that the completion ratio for the in-transit strategies improved up to 25%, when compared with the no in-transit strategy. Additionally, the in-transit2 strategy achieved a lower completion ratio than the in-transit strategy. The reason is that this strategy aims at maximizing the number of jobs accepted at the expense of reducing the completion ratio.

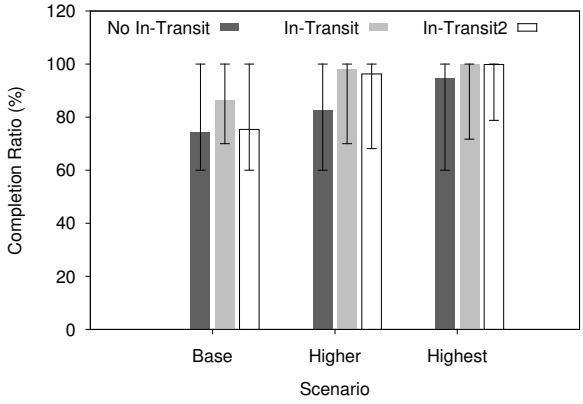


Figure 6: Job completion ratio. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

We have also collected information regarding the overheads involved when computing jobs in the system. These overheads include the time a job spent transferring data in the system (network overhead), shown in Figure 7, and

the time that a job was waiting idle to be computed (queue time), shown in Figure 8. Figure 7 shows that, on average, the use of our in-transit strategies increased the time jobs spent transferring data between 4–29%. This increase in the network overhead was caused by the particularities of our use case, which required all data to be available at the data center prior to initiating any computation. Therefore, when allocating the workload across multiple data centers (in-transit and core cloud) the overheads increased. This drawback does not affect other types of applications, such as video analytics, where data processing can be overlapped with data transfer. Nevertheless, while the network overhead increased, we can observe in Figure 8 that the idle time of the jobs executed using the in-transit strategies decreased up to 80%, when compared with the no in-transit case.

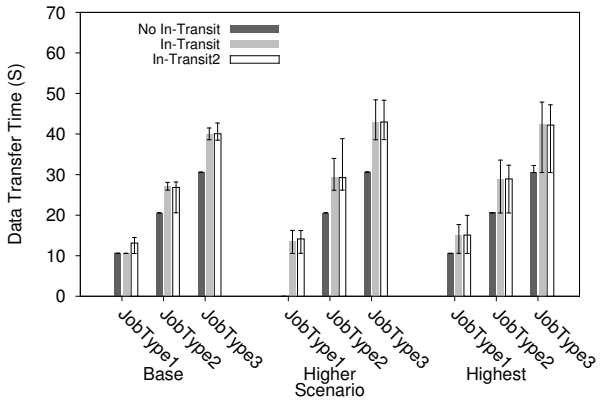


Figure 7: Network Overheads per job. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

Finally, we studied the impact of our in-transit strategies on the cost of computing jobs and the revenue generated from the system. Figure 9 shows the average cost of each type of job for different scenarios and computational strategies. These results show that the average cost of computing a job increased by up to 22% on average. In these scenarios the in-transit computation contributed to a higher completion ratio, hence the increase in cost.

We have also studied how the generated system revenue is distributed amongst data center sites and in-transit resources. Figure 10 contains these results. For the Base scenario, introducing in-transit resources (labeled as Middle) did not affect the revenue of the sites, in fact it increased their revenue by a 3 % and it also increased the overall wealth/revenue within the system. On the other hand, in the Higher scenario, the revenue of the sites was increased by a 1 % when introducing in-transit resources. In the Highest scenario, the site revenue decreased around a 6% when using the in-transit strategies. However, in all cases the overall revenue of the system increased between a 17 and a 48 %.

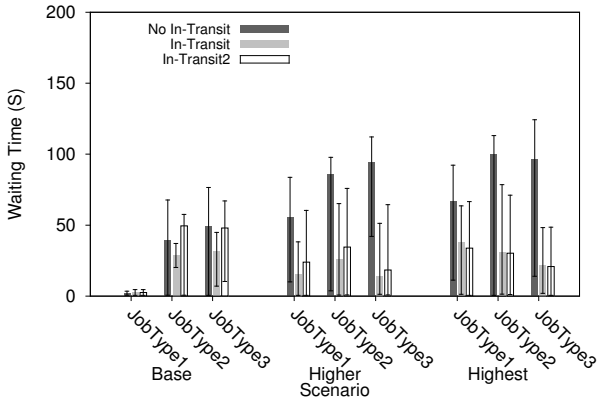


Figure 8: Idle time Overheads per job. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

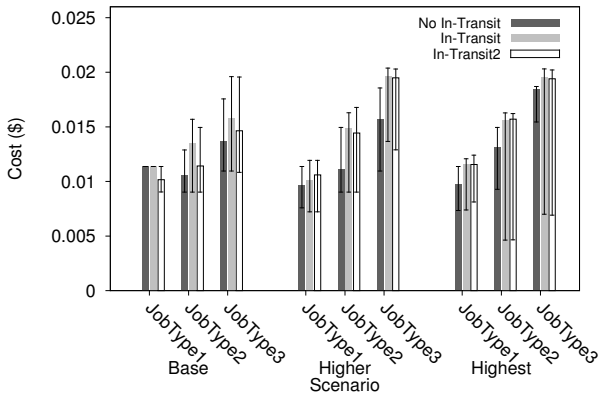


Figure 9: Average Job Cost. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

7.3. Complementing EnergyPlus Optimization with Neural Networks-based Approximate Models

The previous experiments showed that EnergyPlus optimization requires large amounts of computational resources, which affected the number of jobs that the system was able to accept and the associated job completion ratio (i.e. the number of tasks within a job which could be completed by a deadline). In order to: (i) reduce potential resource requirements; (ii) improve use of in-transit resources, we suggest the use of trained artificial neural network (ANN) models alongside EnergyPlus. The ANN based model can complement the execution of EnergyPlus, and provide an approximation of EnergyPlus output.

We map an ANN based optimization process as a function $f(a) : I_a \rightarrow R_a$,

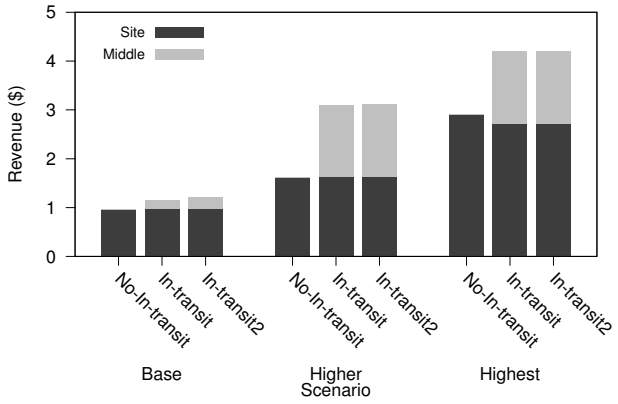


Figure 10: Overall Revenue. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

where I_a is a set representing the input of the ANN model (the same inputs used for EnergyPlus) and R_a is the out from the learned ANN model (the same outputs as EnergyPlus). These input/output sets (I_a/R_a) are based on multiple executions of EnergyPlus over time, representing historical data recorded in the building facility over time. Each execution of EnergyPlus generates data that can be used to subsequently train an ANN model. As the size of this training data grows, the ANN model can replace an EnergyPlus execution and provide an approximate result in a shorter time interval. We assume a three phase execution when an ANN has been used (phase 1 and 2 taking place primarily on the data centre resources):

- Phase 1: EnergyPlus simulations are executed based on measured parameter values from the building. A minimum of 30 simulations are required to acquire enough data to progress to phase 2. The only jobs executed in this phase are generated by EnergyPlus.

- Phase 2: EnergyPlus simulation is co-scheduled with ANN training. Jobs submitted to the system now comprise of both ANN training and EnergyPlus instances. Once the required mean square error has been achieved on the ANN training, Phase 3 is initiated. The workflow used for training the ANN is presented in Algorithm 1. It should be noted that in order to adequately sample the input parameter I_a space adequately, in some instances we may need to generate artificial EnergyPlus simulations.

- Phase 3: Now that the trained ANN model is available, this is deployed over in-transit nodes. The ANN model has a considerably lower execution time (as outlined in results in this section) compared to EnergyPlus simulation. However, ANN model output is very much dependent on the training/test data set used. Any change in building use will render this model inaccurate, requiring us to restart from Phase 1.

Algorithm 1 ANN based optimisation

```
completed = false;
Execute EnergyPlus (30 times);
Train ANN model with EnergyPlus results;
Proc Optimisation()
while !completed do
    Acquire building data;
    Update ANN input/output data
    Test ANN output;
    Update ANN model;
    if accuracy>threshold then
        completed = true;
    else
        Repeat Optimisation();
    end if
end while
```

The ANN training process has two important parameters: (i) number of EnergyPlus simulations needed to train the ANN and (ii) time period over which building data has been acquired (corresponding to different seasons/weather patterns in which the building has been used), identifying the time period after which the ANN needs to be re-trained. The number of EnergyPlus simulations (reported as "threshold" in Algorithm 2) required to train the ANN model is a factor of total time over which data has been acquired and building size. In the FIDIA pilot used for experiments reported in Section 7.1, the ANN training is based on data from 30 EnergyPlus simulations. Effect of weather on the building parameters (outdoor temperature, outdoor humidity, etc.) triggers re-training (two weather patterns were considered in this work). An ANN execution takes (on average) around 5 seconds, compared to an EnergyPlus simulation time of around 2 minutes.

Algorithm 2 presents the steps required to deploy the ANN model over the in-transit architecture. In Algorithm 2, EnergyPlus input is formed of sensor readings, whereas ANN training input comprises a combination of sensor readings and EnergyPlus simulation outputs. ANN output generates optimisation results to implement as building set-points.

In our experimental setup we consider three categories of jobs: a) EnergyPlus jobs, b) ANN training jobs and c) ANN execution jobs.

- *EnergyPlus jobs* – are regular EnergyPlus simulations which can lead to an optimized model of the building. Their characteristics are described in Table 3 and their computational complexity in Table 4.

- *ANN training jobs* – are used to (re)train the ANN up to the required error rate. The number of iterations involved can vary depending on the complexity of the building being considered. The characteristics of a training job are shown in Table 5 and its computational complexity in

Algorithm 2 In-transit workflow with ANN

```
1: Update ANN model
2: Move ANN model to in-transit nodes;
3: for i=1 to number of weather patterns do
4:    Refresh sensor reading;
5:    Execute EnergyPlus with new data;
6:    while EnergyPlus output >= threshold do
7:       Move EnergyPlus output data to in-transit nodes;
8:    end while
9:    Run ANN in-transit;
10: end for
11: Write output data to set-points;
```

Table 6. We consider that EnergyPlus simulations leave their results in the cloud data center and therefore training jobs do not required to transfer any data.

- *ANN jobs* – execution of a trained ANN model at a lower accuracy than EnergyPlus jobs. These types of jobs are of low complexity as the ANN execution does not necessitate high computing specifications. In practice, any of our EnergyPlus jobs can be replaced with an ANN job. As in the case of regular EnergyPlus simulations, the ANN jobs can also be of three types (JobType1, JobType2, and JobType3). Their characteristics are shown in Table 3 – the computational complexity of the ANN jobs is described in Table 6. The computation of an ANN job is always completed at 100%, unlike regular EnergyPlus jobs.

Table 5: EnergyPlus ANN Training Job Information.

JobType	Data Size(MB)	Budget	Deadline(s)	Tasks†
Training	0	∞	∞	30

† – A job is composed by a set of tasks

Table 6: Time to completion of each ANN job type.

JobType	c4.2xlarge	c4.4xlarge	c4.8xlarge
Training	1200 s	1200 s	1200 s
JobType{1-3}	5 s	2.5 s	1.25 s

These experiments used the same workload, computational strategies (No In-transit, In-Transit, and In-Transit2), and scenarios (Base, Higher, and Higher) as before. The only difference was that an ANN training job was inserted

at the beginning of the execution to train the ANN (it is assumed that we already had enough EnergyPlus simulations to have data for training). Once the training job was completed, approximately 20 minutes, the trained ANN was made available to replace EnergyPlus simulations when needed. Thus, every time a regular EnergyPlus job was going to be rejected, the system evaluated if it could complete such job using the trained ANN.

First, we studied the acceptance ratio of jobs. Figure 11 collects the results and differentiate EnergyPlus jobs (E+) from ANN jobs (ANN). The base scenario, which had low computational performance, shows that a significant number of jobs were computed using ANN instead of EnergyPlus. In total, between 21% to 33% of the jobs were accepted and completed using EnergyPlus, and between 26% to 39% were accepted and completed using ANN. Our in-transit strategies helped to increase the number of jobs completed using EnergyPlus between 21% to 36%, but at the cost of decreasing the overall number of accepted job by up to an 8%. In the Higher and Highest scenarios, the system also used ANN jobs to increase the overall acceptance ratio. In these cases, our in-transit strategies increased the overall acceptance ratio between 14% to 19%.

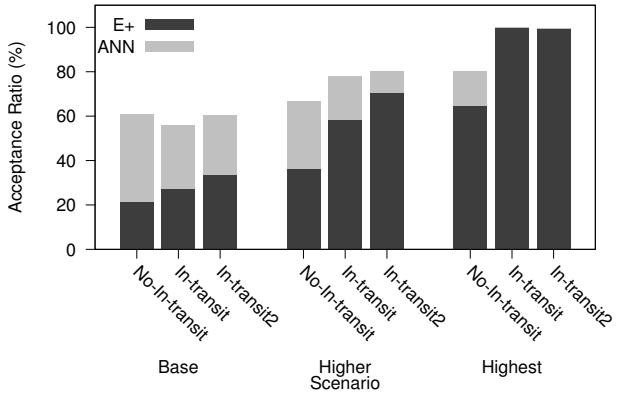


Figure 11: Job Acceptance Ratio. "in-transit" corresponds to the strategy *Traditional client* and "in-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

Figure 12 combines data about the core/hours used by the system to compute the accepted jobs. The results for the Base scenario show that the In-transit strategies used between 23% to 47% more core/hours than the No In-transit strategy. However, both the No In-transit and the In-transit2 accepted a similar number of jobs, which shows that the increase in the number of core/hours was mainly caused by the number of EnergyPlus jobs computed, as the computational complexity of ANN jobs was low. In general, the Higher and Highest scenarios were able to perform more computation in transit, by using resources located at the network data centers (Middle). Specifically, between a 30% to 43% of the overall computation took place at the network data center resources for the Higher and Highest scenarios. In the Base scenario only between 18%

and 20% of the total core/hours were used at resources in the Middle. This was mainly caused by lower performance of computational resources through the whole federation, resulting in rejection of numerous jobs.

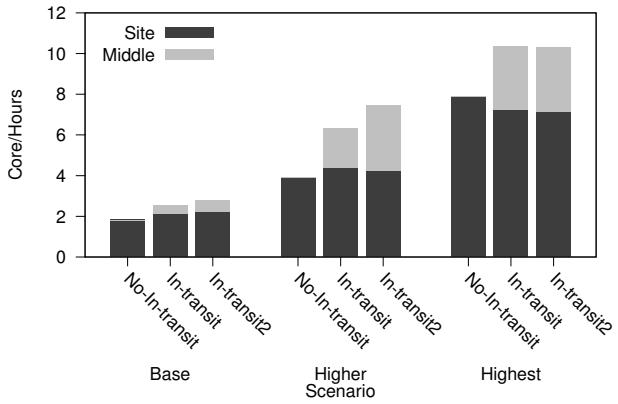


Figure 12: Normalized Core/Hours. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

595 Figure 13 collects the average completion ratio of the EnergyPlus jobs (ANN jobs are always completed at 100%). These results show that the in-transit strategies help to achieve higher job completion ratios, which in this case the average job completion ratio increased between 8% to 25%.

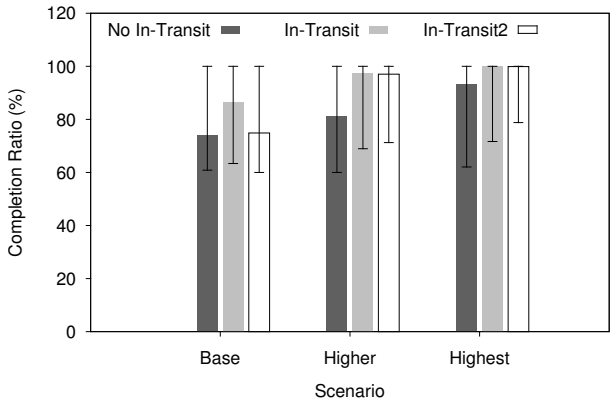


Figure 13: Job completion ratio. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

We also collected information regarding the overheads involved when com-
600 puting jobs in the system. These overheads include the time a job spent transferring data in the system (network overhead), shown in Figure 14, and the time

that a job was waiting idle to be computed (queue time), shown in Figures 15a, and 15b for jobs computed using EnergyPlus and for jobs computed using ANN, respectively.

Figure 14 shows that, on average, the use of our in-transit strategies increased the time jobs spent transferring data by up to 34%. As it was mentioned in the previous section, this was due to the fact that our use case required all data to be fully transferred before starting execution and to increase the number of accepted jobs.

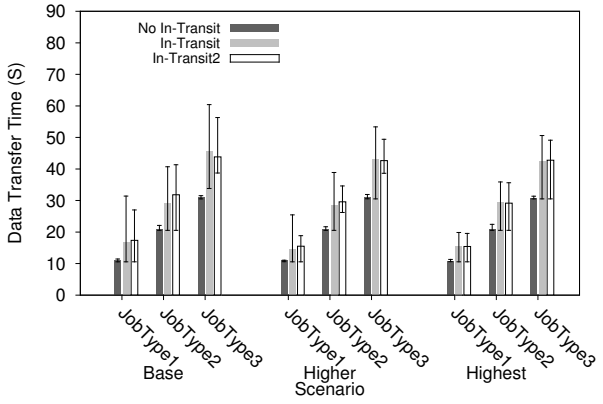
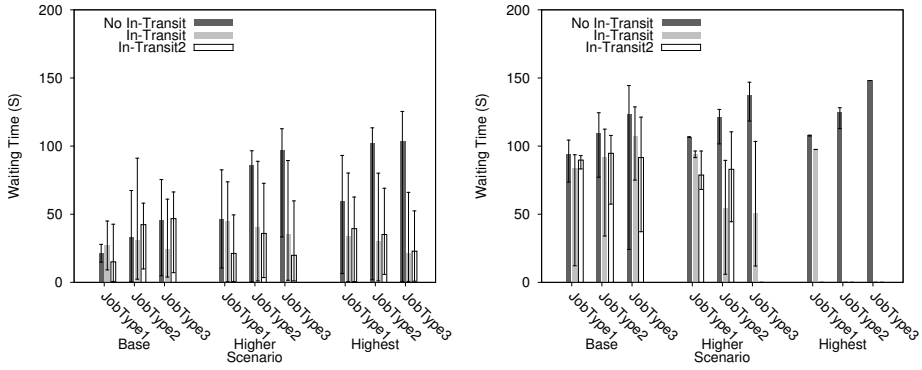


Figure 14: Network Overheads per Job. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

Since the computational time required to compute a job using EnergyPlus and ANN were significantly different, while the deadline was the same, we used two figures to collect the waiting time overheads of the jobs computed using EnergyPlus, Figure 15a, and those that were computed using a trained ANN model, Figure 15b. Comparing both figures, it can be observed that the ANN jobs experienced longer queue times as the scheduler tried to place them in the appropriated place. It can also be observed that using the in-transit strategies decreases the average waiting time by up to 79%, in the case of EnergyPlus jobs, and up to 63%, in the case of ANN jobs.

Figure 16 illustrates the monetary cost required to compute jobs in our experiments. Results show that when using the in-transit strategies, the cost of computing a job increased on average. The main reasons were that the average completion ratio of the jobs is higher and that more EnergyPlus jobs are accepted, as we mentioned earlier. Since EnergyPlus jobs are computationally expensive, the associated monetary cost also increases. In general the average monetary cost of completed jobs increased by up to 58%. Additionally, the dispersion of the cost per job type increased significantly, which was caused not only by the completion ratio of the EnergyPlus jobs, but also by the reduced cost of the ANN jobs.

Finally, Figure 17 shows the overall revenue of the federation for the different



(a) EnergyPlus (b) ANN

Figure 15: Idle Time Overheads per Job. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

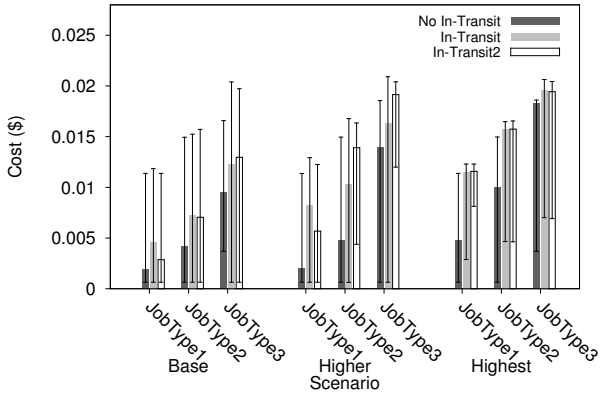


Figure 16: Average Job Cost. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

scenarios. Results show how in these experiments, the use of in-transit resources does not affect negatively the total revenue obtained by the end sites. On the contrary, when comparing the No In-transit strategy against the two in-transit ones (In-Transit and In-Transit2), it can be observed that the overall revenue of the sites increased by up to 35% in the Base scenario, up to 42% in the Higher scenario, and around 34% in the Highest scenario. Additionally, the network data center resources (Middle) were able to generate significant revenue for the federation, amounting to a 48% increase in total.

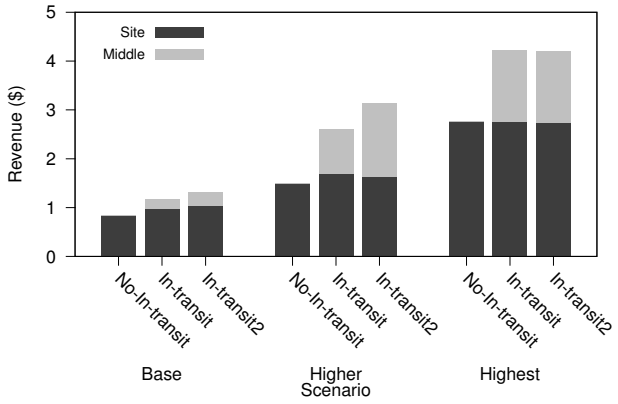


Figure 17: Overall Revenue. "In-transit" corresponds to the strategy *Traditional client* and "In-transit2" corresponds to the strategy *in-transit aware client*. Scenarios refer to Table 1.

7.4. Discussion

The results presented in this paper show the benefits of exploiting computational capabilities found at the network data centers. The proposed computational model allowed us to leverage these in-transit capabilities to increase the number of computation that the infrastructure was able to perform as well as the supported QoS levels. We used a smart building energy calculation use case and tested several scenarios to show the feasibility and benefits of our proposed computational model by making use of in-transit data analysis. We performed two sets of experiments, one where energy optimization jobs were computed exclusively using EnergyPlus and another where energy optimization jobs could be computed using EnergyPlus and alternatively a trained ANN for those cases where the EnergyPlus computation could not meet the SLA. In both experiments, the use of in-transit strategies had a significant impact on the computation. This was shown using different metrics, such as the number of accepted jobs, the completion ratio, overall core/hours used by the experiment, overheads, monetary costs, and overall revenue of the experiment.

In this section we discuss the effects of using a trained ANN to compute energy optimization jobs when using EnergyPlus is not feasible due to SLA constraints. Comparing the acceptance ratio, experiments showed that when using ANN optimization, the number of EnergyPlus jobs accepted decreased by up to 22% for the No In-transit strategy. The scenarios that used the in-transit strategies also experienced a decrease in the number of jobs computed using EnergyPlus by up to 21%. This was caused by a higher utilization of the resources affected by two factors: i) the scheduler used ANN jobs to replace certain EnergyPlus jobs, and ii) we had to compute the ANN training job that were computationally intensive. However, the overall acceptance rate increased between 11% to 55% when combining the use of EnergyPlus and the trained ANN model. The biggest improvement was observed in the Base scenario, where

about half of the accepted jobs were computed using the ANN model.

There were almost no changes in the completion ratio of the EnergyPlus jobs that were accepted and processed by the system. However, we observed some variations in the number of core/hours and revenue of the system. In general, we observed that the amount of core/hours and revenue decreased by up to 11% when using the No In-transit strategy. On the other hand, when using the in-transit strategies, both the overall number of core/hours and revenue of the system increased by up to 47% .

8. Related Work

Software Defined Networks (SDNs) are efficacious instruments for network intensive applications. Many related studies have implemented SDN oriented solutions in order to ease the communication between different networking domains or to optimize various performance parameters within a complex system. Nunes et al. [12] have described the concept of SDN and the various layers involved in such systems. Others have described security challenges faced by SDN [13, 14]. The most recognized protocol to enable a server (SDN controller) to control network elements (such as switches) is OpenFlow [15].

In relation to SDNs, the SWITCH project[16] addresses a number of existing industrial requirements for developing and executing time critical applications in Clouds. SWITCH provides an interactive environment for developing applications and controlling their execution, a real-time infrastructure planner for deploying applications in Clouds, and an autonomous system adaptation platform for monitoring and adapting system behavior. The work outlined here differs from the SWITCH project in utilizing a combination of edge, in-network capability alongside data center/Cloud infrastructure.

In the field of active networking, communication patterns are used for addressing specific user requirements [17]. An active network refers to a specific capability to execute tasks within the network over active elements such as switches that have processing capability. Lefevre et al. [18] developed an active network architecture (A-Grid) to support QoS-related metrics for Grid data transport services in addition to other data transport services such as reliable multicast and dynamic service deployment. The architecture employs QoS management at intermediate active routers, and in principal, it is similar to the in-transit processing employed in our approach. Understanding how application requirements get mapped to such an architectue has not been fully addressed in existing work. The proposed work maps constraints associated with an application into capability of each component.

Another emerging research topic is the availability of network resource reservation systems such as ESNET's OSCARS [19] and UltraScience Net [20]. These types of systems can provide on-demand dedicated bandwidth channels to user applications. The main idea in resource reservation systems is that a virtual single-switch abstraction is added on top of networks facilitating both a bandwidth reservation system and SDN processing.

This paper primarily focuses on mechanisms that make use of SDN-based processing capability to improve cloud federation performance. Using SDN to improve the performance of applications that need to process large amounts of data has been discussed by other authors. Wang et al. have analyzed the potential performance improvement of big data application by integrating network control directly within the application [21]. In [22] Xiong et al. show how performance of data analysis applications can be improved using SDN within distributed data storage environments. Das et al. propose improving application performance by choosing particular routes and making more effective use of dynamic links [23]within a network. In [24] Wang et al. integrate network resources into datacenter orchestration and provide isolated virtual networks for improving service quality. Similarly, Luo et al. [25] and Miyamoto et al. [26] propose to transform network bandwidth into manageable resources, and provide guaranteed virtual networks. Furthermore, in-transit computation has been used to aid HPC computation [27, 28]. In our research, the focus is on improving performance of federated cloud systems by using SDN-based approaches, and unlike mentioned papers our target is not only considering networking resource but also dealing with other computational and storage resources through the use of an SDN controller.

Our work shares commonalities with these related studies from the perspective of achieving performance improvements, through federated systems at different computational capabilities (at the network edge, in-transit and within a data center). The proposed work also has similar interests in achieving the overall objective of providing greater transparency into the network's state, enabling users to make more informed decisions and to adapt to changes that may appear in the network. However, the main differentiating aspect of the work reported here is the adoption of an application-centric methodology, whereby SDN capability can be employed efficiently based on a set of user-defined constraints, subsequently validated in a building energy scenario. The proposed work also contributes to supporting in-transit processing to enable maximizing job acceptance and completion ratios (within an application specific deadline). These metrics (acceptance and completion ratios) are often critical in real-time applications, with increasing impact on cost and business *value*.

9. Conclusions

This paper introduced a computational model that enables the use of latent computational capacities and capabilities found in network data centers. We showed how integrating software-defined network (SDN) technology in our federated infrastructure can enable the use of resources located at the network data centers to perform in-transit computation of data that is being transferred. We used a smart building energy calculation use case and tested several scenarios to show the feasibility and benefits of our proposed computational model by making use of in-transit data analysis.

Experimental results showed that when the computational capacity of resource providers is limited, enabling in-transit computation can improve job ac-

ceptance ratio and their average completion ratio. This holds even in cases where in-transit resources are not very powerful compared with the computational resource providers' resources. The main reason is that the computational capacity available in the federation increased, which proves that our approach was able to extract latent computational cycles from in-transit resources. However, experiments showed that only using the idle/waiting time of jobs to perform in-transit computation limits the practicability of the in-transit approaches. For this reason, we are currently working to explore additional in-transit strategies in which both in-transit resources and end sites are part of an optimization problem to decide how computation can be effectively distributed across each. Moreover, we are defining a new business model that takes into account not only the capabilities of the resources, but also their location in the infrastructure (e.g., at the edge vs. deep into the infrastructure). We also plan to extend our prototype to consider more than one in-transit resource to enable allocating resources in more complex data path. Finally, we would like to explore scenarios in which the computation of certain jobs or buildings have different priorities.

Acknowledgements

This research is supported in part by NSF via grants ACI 1339036, ACI 1441376. Chameleon is supported by NSF OCI-1419152. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2). The scenario outlined in this work comes from the European "Energy Efficiency for European Sports Facilities" (SportE2) project (http://www.sporte2.eu/). We are grateful to Prof. Rezgui (Cardiff University) for access to this scenario.

References

[1] S. Jain, A. Kumar, S. Mandal, et al., B4: Experience with a globally-deployed software defined wan, in: ACM SIGCOMM 2013, 2013, pp. 3–14.

[2] OpenCloud, Nfaas - network function as a service (2014).

[3] FIDIA project. http://www.sporte2.eu/fidia-sport (Last accessed on June 2016.).

[4] I. Petri, O. F. Rana, Y. Rezgui, H. Li, T. Beach, M. Zou, J. D. Montes, M. Parashar, Cloud supported building data analytics, in: CCGrid, 2014, pp. 641–650.

[5] T. Jin, F. Zhang, Q. Sun, et al., Poster: Leveraging deep memory hierarchies for data staging in coupled data-intensive simulation workflows, in: IEEE CLUSTER, 2014, pp. 268–269.

[6] I. Petri, M. Zou, A. Zamani, J. Diaz-Montes, O. F. Rana, M. Parashar, Integrating software defined networks within a cloud federation, in: CCGrid, 2015.

[7] J. Diaz-Montes, M. AbdelBaky, M. Zou, M. Parashar, Cometcloud: Enabling software-defined federations for end-to-end application workflows, IEEE Internet Computing 19 (1) (2015) 69–73.

[8] J. Diaz-Montes, Y. Xie, I. Rodero, et al., Federated computing for the masses - aggregating resources to tackle large-scale engineering problems, CiSE Magazine 16 (4) (2014) 62–72.

[9] Z. Li, M. Parashar, Comet: A scalable coordination space for decentralized distributed environments, in: Intl. Workshop on Hot Topics in Peer-to-Peer Systems, 2005.

[10] Mininet project. http://mininet.org (Last accessed on June 2016.).

[11] GRE Tunneling. http://lartc.org/howto/lartc.tunnel.gre.html (Last accessed on June 2016.).

[12] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Communications Surveys Tutorials 16 (3) (2014) 1617–1634.

[13] S. Scott-Hayward, G. O'Callaghan, S. Sezer, Sdn security: A survey, in: Future Networks and Services (SDN4FNS), 2013 IEEE SDN for, 2013, pp. 1–7.

[14] S. Shin, G. Gu, Attacking software-defined networks: A first feasibility study, in: ACM SIGCOMM workshop on Hot topics in software defined networking, 2013, pp. 165–166.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[16] SWITCH project. http://www.switchproject.eu (Last accessed on June 2016.).

[17] D. L. Tennenhouse, D. J. Wetherall, Towards an active network architecture, SIGCOMM Comput. Commun. Rev. 37 (5) (2007) 81–94.

[18] L. Lefevre, C.-d. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J.-P. Gelas, M. Maimour1, Active networking support for the grid, in: I. Marshall, S. Nettles, N. Wakamiya (Eds.), Active Networks, Vol. 2207 of Lecture Notes in Computer Science, 2001, pp. 16–33.

[19] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, W. Johnston, Intra and interdomain circuit provisioning using the oscars reservation system, in: 3rd Intl. Conf. on Broadband Communications, Networks and Systems (BROADNETS), 2006, pp. 1–8.

[20] N. Rao, W. Wing, S. Carter, Q. Wu, Ultrascience net: network testbed for large-scale science applications, IEEE Communications Magazine 43 (11) (2005) S12–S17.

[21] G. Wang, T. Ng, A. Shaikh, Programming your network at run-time for big data applications, in: ACM SIGCOMM Workshop on Hot topics in software defined networks, 2012, pp. 103–108.

[22] P. Xiong, H. Hacigumus, J. F. Naughton, A software-defined networking based approach for performance management of analytical queries on distributed data stores, in: ACM SIGMOD Intl. Conf. on Management of data, ACM, 2014, pp. 955–966.

[23] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, P. D. Desai, Application-aware aggregation and traffic engineering in a converged packet-circuit network, in: National Fiber Optic Engineers Conference, Optical Society of America, 2011.

[24] Z. Liu, X. Wang, Y. Qi, J. Li, Livecloud: A lucid orchestrator for cloud datacenters, in: CloudCom, 2012, pp. 341–348.

[25] M.-Y. Luo, J.-Y. Chen, Software defined networking across distributed datacenters over cloud, in: CloudCom, 2013, pp. 615–622.

[26] T. Miyamoto, M. Hayashi, K. Nishimura, Sustainable network resource management system for virtual private clouds, in: CloudCom, 2010, pp. 512–520.

[27] K. Moreland, R. Oldfield, P. Marion, et al., Examples of in transit visualization, in: Intl. Workshop on Petascal Data Analytics: Challenges and Opportunities, PDAC '11, 2011, pp. 1–6.

[28] J. C. Bennett, H. Abbasi, P.-T. Bremer, et al., Combining in-situ and in-transit processing to enable extreme-scale scientific analysis, in: SC'12, 2012, pp. 49:1–49:9.