SSICLOPS

Measurements, experiments, system analysis report on SSICLOPS intra-cloud scenarios

SSICLOPS Deliverable D.4.2.

Jukka Kommeri,[1] Seppo Heikkilä,[1] Tapio Niemi,[1] Felix Eberhardt,[2] Max Plauth,[2] Andreas Polze,[2] Stefan Klauck,[3] Matthias Uflacker,[3] Maël Kimmerlin,[4] Jose Costa-Requena,[4] Emil Kowalczyk,[5] Dariusz Bursztynowski,[5] Paweł Parol,[5] and Gerhard Hasslinger[6]

[1]*Helsinki Institute of Physics, Finland*
[2]*Operating Systems and Middleware Group, Hasso Plattner Institute, Germany*
[3]*Enterprise Platform and Integration Concepts Group, Hasso Plattner Institute, Germany*
[4]*Department of Communications and Networking, Aalto University, Finland*
[5]*Orange Polska S.A., Poland*
[6]*Deutsche Telekom, Germany*

July 29, 2016

This paper has received funding from the European Union's Horizon 2020 research and innovation program 2014–2018 under grant agreement No. 644866 ("SSICLOPS"). It reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

Contents

Abstract

The first results for the solutions developed by the partners in the SSICLOPS project are presented and evaluated. The improvement in solving problems and bottlenecks indicated by SSICLOPS are discussed in detail and concluded. In particular, a Hyrise-R extension was implemented and different replication schemes were tested. High Energy Physics (HEP) performance was studied in an OpenStack testbed with the focus on the impact of network latencies on the performance and energy consumption. Caching methods were analysed based on simulation. Lastly, the first version of a virtual Home GateWay (vHGW) application was developed using open-source components run in an OpenStack environment. Although the results are related to varied use cases, they together contribute to the final result of the SSICLOPS project.

Executive Summary

This report provides and discusses the results obtained under the evaluation of solutions developed in the SSICLOPS project. They are based on running testbeds or simulations, according to the use cases considered in the project. The improvement in solving problems and bottlenecks investigated by SSICLOPS are based on metrics defined for each use case individually. The results presented in this report are related to other parts of the project such as stack modelling (work package "Private Cloud Infrastructure"), determination policy language (work package "Policy and Security") or optimization of cloud interconnection solutions (work package "Federated Private Clouds"). All of the use cases, allow to improve the design and implementation of inter-cloud orchestration as the final result of the SSICLOPS project. The individual summaries for the particular use cases are presented as follows.

Hyrise-R is a scale-out extension of the in-memory database Hyrise where users send queries, as Hyper-Text Transfer Protocol (HTTP) JavaScript Object Notation (JSON) query plans or stored procedures, to a query dispatcher, which propagates requests to appropriate database instances. Data altering requests are processed by the master instance. The master sends logs, describing the data changes, via Transmission Control Protocol (TCP) to the replica nodes in an eager or lazy manner. Adding replicas can scale read-only and mixed workloads linearly. In general, the amount of exchanged data compared to the processing within the database is low. For eager replication, the master node has to wait that replicas acknowledge the log reception. In this case, network latency can affect the transaction latency. Bandwidth is important when returning large result sets, or performing large data updates and sending these to replicas.

The network performance affects the computing performance and energy efficiency of High Energy Physics (HEP) computing in an OpenStack cloud testbed. Our results indicate that the network latency, either caused by a simulator or physical distances between the sites, has a negative impact on the computing performance. High latency both increases processing times and the total energy consumption. Moreover, the effect of latency on the execution time and energy consumption of a computation job, increases when bandwidth is small. Parallelism, e.g. multiple cloud instances sharing the limited network resource, also adds more latency and increases processing times. The results reflect the current software environment used for HEP computing. We conclude that new data transfer protocols and advanced caching mechanisms could improve performance. Planned activities will cover the inclusion of mechanisms which could also provide additional profiles of workload for this use case.

The evaluation of caching methods reveals substantial performance deficits of usual caching strategies following the Least Recently Used (LRU) principle. For the standard model of independent Zipf distributed requests (IRM), we confirm that LRU hit rates in the range 10 - 50% generally leave further 10 - 20% absolute hit rate potential unused compared to score based strategies, which include statistics about past requests. While similar experience has been made also in other studies for strategies involving more complex updates, we recommend a score-gated LRU variant which often even undercuts low LRU update effort. The clients of F-Secure's cloud services query attributes of files and URLs from a large set of known objects, but not all objects are queried equally often. By implementing a caching intermediate server and collecting statistics from a subset of clients, it was found out that most of the time a responses could be served from the cache. The volume of traffic during the measuring period did not cause excess load for the chosen test platform. For identifying bottlenecks in the intermediate server implementation, it will be necessary to increase the volume of requests. Research into more advanced caching strategies could enable us to limit the number of response to cache without sacrificing too much of the cache hit rate. Planned work is to study caching strategies in the context of a real-world content distribution application based on collected data.

Based on the results obtained so far, it can be observed that Network Function Virtualization Infrastructure (NFVI) for a virtual Home Gateway (vHGW) offers high bandwidth capabilities (gigabits per second) for transmissions between the Virtual Machines (VMs) involved. Measured round-trip time (RTT) values meet the requirements of typical LAN RTT (RTT less than 1 ms). Also the jitter level is very low (no more than 0.05 ms). Packet loss starts to exceed 1% for flows transmitted with a rate close to 1 Gbps or higher. Packet loss is well below 1% for flows of lower rate. Both an overlay Virtual eXtensible Local Area Network (VxLAN)-based network as well as an underlay native OpenStack network can be considered as logical network structures incurring deterministic delay. It must be stressed, however, that the tested scenario was centralized in a single datacenter and assumed homogeneous utilization of the datacenter, i.e., only the vHGW service running without another background traffic present. Future work will cover tests of service performance assuming a more diversified load of the datacenter. Tests related to computational resources utilization and energy consumption will also be carried out.

Abbreviations

ALICE	A Large Ion Collider Experiment
BBF	Broadband Forum
BNG	Broadband Network Gateway
CDN	Content Delivery Network
CDNI	Content Delivery Networks Interconnection
CERN	European Organization for Nuclear Research
CGNAT	Carrier Grade Network Address Translation
CMS	Compact Muon Solenoid
COTS	Commercial off-the-shelf
CPE	Customer Premise Equipment
CPU	Central Processing Unit
CVMS	CERN Virtual Machine Filesystem
DBMS	DataBase Management System
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
DNS	Domain Name System
DPI	Deep Packet Inspection
DSL	X Digital Subscriber Line
E2E	End-to-end
EPC	Evolve Packet Core
FXS	Foreign eXchange System
HEP	High-Energy Physics
HGW	Home Gateway
HTTP	Hyper-Text Transfer Protocol
I/O	Input / Output
IaaS	Infrastructure-as-a-Service
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPTV	Internet Protocol Television
IRM	Information Resources Management
ISP	Internet Service Providers
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicators

L2	Data Link Layer in Open Systems Interconnection Model
L3	Network Layer in Open Systems Interconnection Model
LAN	Local Area Network
LHC	Large Hadron Collider
LRU	Least Recently Used
MAC	Media Access Control
NAS	Network Attached Storage
NAT	Network Address Translation
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NUMA	Non-uniform memory access
OLAP	OnLine Analytical Processing
OLTP	OnLine Transaction Processing
OPNFV	Open Network Function Virtualization
PAT	Physics Analysis Toolkit
PON	Passive optical network
POP	Point of Presence
PPP	Point to Point Protocol
QoE	Quality of Experience
QoS	Quality of Service
RTT	Round Trip Times
SAN	Storage Area Network
SDN	Software Defined Network
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TPC	Transaction Processing Performance Council
TTM	Time To Market
UPnP	Universal Plug-and-Play
URL	Uniform Resource Locator
USB	Universal Serial Bus
vHGW	Virtual Home Gateway
VM	Virtual Machine
VoIP	Voice over IP
VxLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network
WiFi	Wireless Fidelity
WLCG	World Wide LHC Computing Grid

1. Introduction

The purpose of this work is to provide and to evaluate first results for the solutions developed by the partners in the SSICLOPS project. The methodology of evaluation was to compare results obtained at the initial phase of project to ones obtained after 18 months of duration. This allows to indicate improvements in solving problems and bottlenecks investigated by SSICLOPS. Although, results presented in this document are obtained for four individual use cases, they together contribute to other parts of the project (Private Cloud Infrastructure, Policy and Security, Federated Private Clouds) to improve the design and implementation of inter-cloud orchestration as the final result of the SSICLOPS project.

Individual points from the particular use cases (and thus partners) are presented as follows.

The growing performance demand for database systems, caused by an increasing number of users who utilize OnLine Analytical Processing (OLAP), search, and read-only applications, requires scalability and elasticity. Load balancing with read-only database replication is a way to cope with these requirements. Hyrise-R is an implementation of master replication. A master database instance processes all write transactions and sends logs to replica instances to keep them synchronized. This document presents the query performance and communication characteristics of Hyrise-R.

The HEP Workload use case focuses on processing particle physics applications in a distributed cloud computing environment. In the case of HEP data analysis jobs, which fetch constantly data from remote locations, both network delay and bandwidth can become bottlenecks and cause delays for the analysis. These delays can have a significant impact on overall performance. We have analysed this effect by using simulations and running realistic test cases in geographically different locations connected by the Internet. The obtained results reflect the current software environment used for HEP job processing.

The concept of a vHGW assumes that most of the traditional HGW functionality is shifted from a dedicated physical device to computing resources managed by a Telecom operator. It can be accomplished for instance through launching network functions on VMs instantiated on IT resources (Commercial off-the-shelf servers) of a next generation PoP node or a classical datacenter. This is in line with the NFV-based approach. The results contained in this report include a description of the vHGW testbed based on open source software components and initial evaluation of the network-related characteristics of the solution.

Most services provided on the Internet are nowadays supported by content delivery networks or clouds, which include distributed cache servers to shorten transport paths and delays. Consequently, large portions of Internet traffic are delivered from caches in datacenters, which have to handle high workloads from a large user population. Caching efficiency benefits from a pattern which focuses user requests to a small set of highly popular Zipf distributed requests. On the other hand, simple caching strategies are prevalently used, which don't include knowledge about the popularity of objects from the request history. We study the efficiency of web caching strategies in a trade-off between low complexity of cache updates and involvement of information about past requests for improving the cache hit rate.

In the object reputation cloud service use case, a geographically diverse group of clients are querying a server to learn about attributes of common files and Uniform Resource Locators (URLs). We have implemented an intermediary server capable of caching and distributing this data to clients, and have collected and analysed anonymous data on how queries are distributed in the key space. The efficiency of the caching will affect the performance of the system, and ultimately end user experience.

In the following sections, the achievements of partners are presented using a unified description pattern. First, each use case is shortly recalled to indicate the problem statement. After that, the metric for performance measurement is presented and discussed. Next, the particular experiment testbeds are described in detail and finally, performance results are concluded. The last section of this report summarizes the overall results achieved and provides the work plan for the next period of the project.

Note, that the original "Use Case 3: Efficient and secure cloud bursting" is intentionally omitted and replaced by "Use Case 5: Content Distribution and Caching". This reflects the changes proposed by the consortium to the Description of Action for the SSICLOPS project (see Amendment request, reference No AMD-644866-9).

2. Use case 1: In-Memory Databases in the Cloud (HYRISELab)

2.1. Use case description

In-memory database systems are well-suited for enterprise workloads, consisting of transactional and analytical queries. A growing number of users and an increasing demand for enterprise applications can saturate or even overload single-node database systems at peak times. Better performance can be achieved by improving a single machine's hardware but it is often cheaper and more practicable to follow a scale-out approach and use additional machines.

The large amount of read queries and the possibility to distribute them among several nodes make the concept of database replication desirable for enterprise applications. Krüger et al. show in an analysis of a modern enterprise system that more than 80% of OLTP and more than 90% of OLAP requests are read queries [23]. SAP HANA and HyPer's scale-out version ScyPer propose master replication to cope with growing OLAP demands [28, 33]. Replica instances can execute read queries on snapshots in parallel without violating any consistency or isolation requirements (see Figure 1).

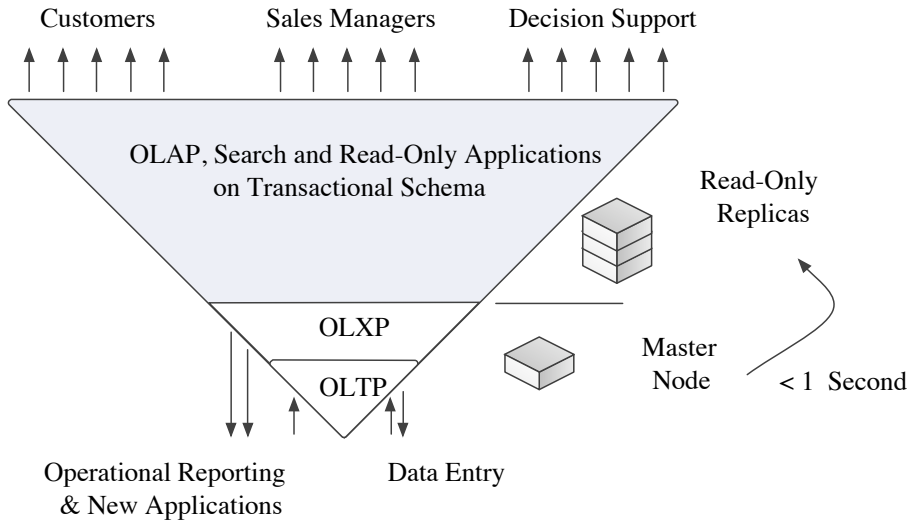

Figure 1: Read-only replication [33].

We implemented master replication for the in-memory database Hyrise. Hyrise is an in-memory research database project, initiated by the investigation of an optimal storage layout. Hyrise supports flexible hybrid table layouts, i.e., storing attributes corresponding to their access patterns to optimize cache locality [16]. On the one side, a columnar arrangement is well-suited for attributes which are often accessed sequentially, e.g., via scans, joins, or aggregations. On the other side, attributes accessed in OLTP style queries, e.g., projections of few tuples, can be stored in a row-wise manner. Over time, many in-memory database concepts were developed, evaluated, and integrated into Hyrise. By exploiting a main-delta architecture, Hyrise is well-suited for mixed workloads. Tuples in the main partition are stored dictionary compressed with a sorted dictionary. This allows efficient vector scanning and supports range queries without decoding complete columns. Data modifications are inserted in the write-optimized delta partition. Using an unsorted dictionary for the delta is a trade-off for better write and reasonable read performance. The periodic merge process moves tuples from the delta to the main partition [23]. Hyrise exploits an insert-only approach and multi-version concurrency control with snapshot isolation as default isolation level [37]. That is why Hyrise can process writes without delaying read queries. Hyrise features a task-based query execution framework to execute dynamically arriving transactional queries in a timely manner, even while complex analytical queries are executed [45].

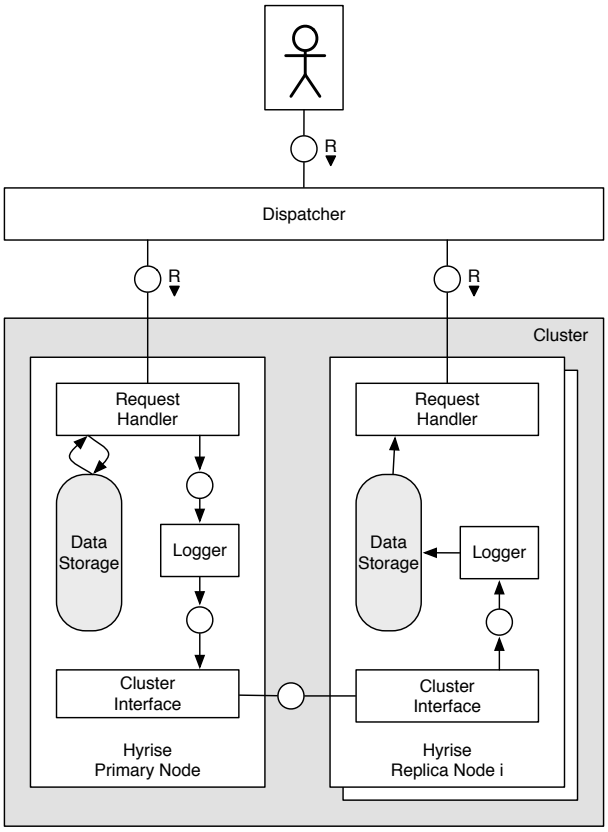

Figure 2: Hyrise-R architecture.

For the scale-out extension, called Hyrise-R [38], we added a Cluster Interface to the Hyrise core. Figure 2 shows the architecture of a Hyrise-R cluster, consisting of a query dispatcher, a single Hyrise primary/master node, and an arbitrary number of replicas. Users send their database requests to a query dispatcher, which acts as a load balancer for read requests. The dispatcher parses the queries for data altering operations, i.e., inserts, updates, and deletes. The master node processes all write workloads. Data changes are written into a local log, implemented as a ring buffer [37]. Besides storing the log to persistent memory, the primary node sends it to the replicas, which update their data storages accordingly.

2.2. Metrics

Query performance

As a (scale-out) database system, Hyrise-R's main metrics relate to query performance, e.g., the number of queries which can be processed per second. However, it is impossible to describe the performance of a database system with a single number without knowledge about the kind of queries, i.e., how complex their calculations are and whether they just read or also modify data. Manegold et al. propose a cost model based on main memory accesses [24], which could be used to describe query complexity. Another approach to measure database performance is running standardized workloads. The Transaction Processing Performance Council (TPC) defines database benchmarks for objective database performance data, e.g., TPC-C as on-line transaction processing and TPC-H as ad-hoc decision support system. Besides pure query performance, a primary goal of Hyrise-R is scalability, i.e., the performance behavior when adding replicas to the database cluster.

Communication

Communication metrics are important to describe the performance of a distributed database system. Hyrise-R has the same query interface as Hyrise. Users specify their database requests as HTTP POSTs with JSON payloads to describe the query plans or stored procedures to execute. The query dispatcher parses the request body and redirects writes to the master instance. Reads can be redirected to any cluster instance. The master instance sends physical logs via TCP, using the socket library nanomsg[1], to the replicas to keep them in sync.

For Hyrise-R, we can measure the number of processed HTTP requests per second: on the one hand for single Hyrise instances, on the other hand for the dispatcher. The time to synchronize the replica instances is another metric. We can specifically investigate whether the synchronization overhead influences the query performance.

[1] http://nanomsg.org/ (Accessed July 2016)

In general, the amount of exchanged data compared to the processing within the database is low. Bandwidth is important when returning large result sets, or performing large data updates and sending these to the replicas.

2.3. Experiments

Query performance

We defined a number of experiments for evaluating Hyrise-R [38]. For these measurements, we run Hyrise-R on five instances of Amazon's EC2 c4.8xlarge with Intel Xeon E5-2666 v3 processors, 60 GB main memory, and SSDs as persistent storage. Four instances were running Hyrise instances and one instance was running the dispatcher and the Apache HTTP server benchmark tool.

- Read-only scale-out. To measure the scaling of a Hyrise-R cluster, we executed analytical queries, which scan the complete table and aggregate the values. A single Hyrise instance (zero replicas) is our baseline. We increased the number of replica instances up to three.
- Mixed workload scale-out. We tried to enhance the write performance for mixed workloads by increasing the number of replica instances. The idea is that when increasing the number of replicas, the master node has to process less read requests and can increase the write throughput in this way. The writes were transactional queries, far less complex than the reads.
- Write-only performance. We measured the write-only performance to test whether the replica synchronization mechanisms can become a bottleneck. We sent only write queries to the dispatcher, which redirects all requests to the master. The master instance processes the requests, writes the logs to its SSD, and sends it in addition to all replica instances asynchronously.

Communication

To measure the requests per second, Hyrise has a database operation, called NoOp, which does nothing, i.e., it is implemented as function that just returns. We use a query plan consisting of only such NoOp to measure the number of requests Hyrise(-R) can process per second, which includes the HTTP request handling, query plan parsing, and query scheduling. The following listing shows a corresponding JSON query plan.

```
{
  "operators": {
    "no_op": {
      "type": "NoOp"
    }
  },
  "edges": [
    ["no_op","no_op"]
  ]
}
```

We used the Apache HTTP server benchmark tool to send NoOp to a Hyrise instance. With the tool we could specify the number of concurrent requests and use persistent connections. We submitted the requests on the database server and another computer in the network.

As an example of an HTTP Hyrise request which actually queries data, we present query 1 of the CH-benCHmark. The CH-benCHmark is a combination of TPC-C and TPC-H (see Subsection 2.2). The presented query *"reports the total amount and quantity of all shipped orderlines given by a specific time period. Additionally it informs about the average amount and quantity plus the total count of all these orderlines ordered by the individual orderline number."* [2]

```
select   ol_number,
         sum(ol_quantity) as sum_qty,
         sum(ol_amount) as sum_amount,
         avg(ol_quantity) as avg_qty,
         avg(ol_amount) as avg_amount,
         count(*) as count_order
from     orderline
where    ol_delivery_d > '2007-01-02 00:00:00.000000'
group by ol_number order by ol_number
```

A corresponding Hyrise query plan is:

```
{
  "operators": {
    "get_orderline": {
      "type": "GetTable",
      "name": "ORDER_LINE"
    },
    "filter_date": {
      "type": "SimpleTableScan",
      "predicates": [{"type": "GT", "in":0, "f":"OL_DELIVERY_D",
                "value":"2007-01-02 00:00:00.000000", "vtype": 2}]
    },
    "project": {
      "type": "ProjectionScan",
      "fields": ["OL_NUMBER","OL_QUANTITY","OL_AMOUNT"]
    },
    "hashbuild": {
      "type": "HashBuild",
      "key": "groupby",
      "fields": ["OL_NUMBER"]
    },
    "groupby":{
      "type": "GroupByScan",
```

[2] https://db.in.tum.de/research/projects/CHbenCHmark/?lang=de (Accessed July 2016)

```
      "fields": ["OL_NUMBER"],
      "functions":[
        {"type": "SUM", "field": "OL_QUANTITY", "as": "SUM_QTY"},
        {"type": "SUM", "field": "OL_AMOUNT", "as": "SUM_AMOUNT"},
        {"type": "AVG", "field": "OL_QUANTITY", "as": "AVG_QTY"},
        {"type": "AVG", "field": "OL_AMOUNT", "as": "AVG_AMOUNT"},
        {"type": "COUNT", "field": "OL_QUANTITY", "as": "COUNT_ORDER"}
      ]
    },
    "sort": {
      "type": "SortScan",
      "fields": [0]
    }
  },
  "edges": [
    ["get_orderline","filter_date"],
    ["filter_date","project"],

    ["project", "hashbuild"],
    ["hashbuild", "groupby"],

    ["project", "groupby"],
    ["groupby", "sort"]
  ]
}
```

2.4. Results

Query performance

One of the main targets of Hyrise-R is to improve the performance of Hyrise by a scale-out approach.

- Read-only scale-out. In this experiment, only read requests are sent to the cluster and the number of replica is increased from one to three. The performance is compared with a single node setup (zero replicas). As we can see in Figure 3, Hyrise-R scales linearly for read-only workloads.
- Mixed workload scale-out. In this experiment, read and write requests are sent to the cluster, whereby the reads dominate the workload and there is no prioritization of writes. The reads had to access far more data (full table scans) than the writes. Figure 4 shows the results. An increasing number of replicas can process an increasing number of read requests. In this way, the ratio of writes for the master increases, which results in a growing number of writes per second for mixed workload. This scales until the master processes writes exclusively.
- Write-only performance. To propagate the commits throughout the cluster, the master instance sends log data to the replicas. The data exchange increases with an increasing

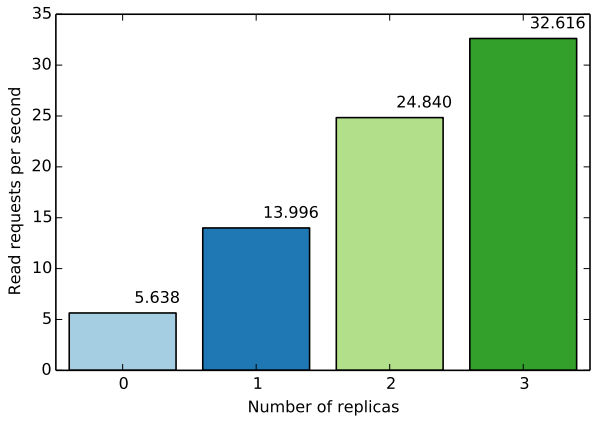

Figure 3: Read-only scale-out [38].

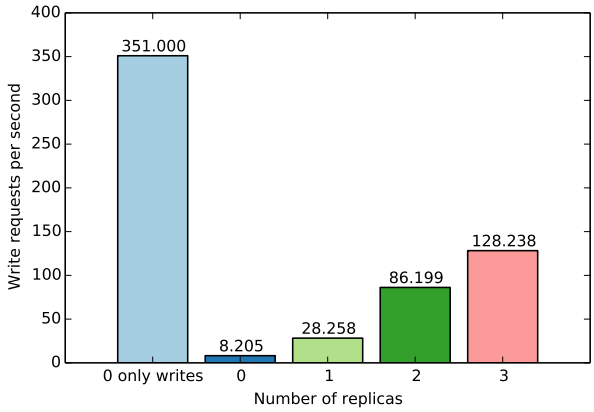

Figure 4: Mixed workload scale-out [38].

number of replica nodes. As we can see in Figure 5, there is no impact on the communication. At least in our closely connected experiment setup, the overhead of communication was too small to degrade performance.

Communication

Table 2 shows the measured database requests per second for a single Hyrise instance. The round-trip time for the remote running Hyrise server, measured with the ping tool, was about 0.220ms. The complete HTTP request had a size of about 250 bytes. We assume that 20,000 requests per second are a limit of Hyrise's internal processing (rather than a network bottleneck), which includes request parsing and scheduling. As peaks, we measured up to 50,000 requests per second.

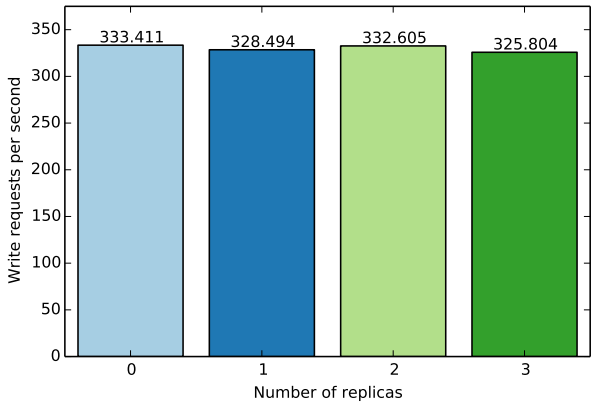

Figure 5: Write-only performance [38].

Table 2.: Hyrise requests per second, measured with Apache tool.

concurrent requests	local requests per second	remote requests per second
1	11,500	4,500
2	12,900	8,600
4	14,500	11,800
8	16,700	17,200
16	18,400	21,000
32	20,000	22,500

2.5. Summary

Database replication is a possibility to cope with the increasing OLAP demand. We implemented master replication for the hybrid in-memory database Hyrise. A Hyrise-R cluster has a dedicated master instance for transaction processing. We presented the Hyrise JSON query format which is sent as HTTP payload to the cluster nodes. The master node propagates data modifications via TCP to the replicas in an *eager* or *lazy* manner. We performed several experiments to benchmark the read-only, mixed-workload, and write-only performance of Hyrise-R. In addition, we investigated Hyrise's limits for requests per second.

3. Use case 2: High-Energy Physics Workloads

3.1. Use case description

High Energy Physics (HEP) studies elementary particles by using large particle accelerators, such as the Large Hadron Collider (LHC) at CERN, for producing millions of high-energy particle collision events. In order to understand physics phenomena, one must go through a very large quantity of measurement samples. A single high-energy physics analysis can process millions of events [35]. This work can be easily parallelized because there are no dependencies among these events. Particle physics events are stored in database-like containers, ROOT files [7]. The required computing resources for CERN LHC data analysis are divided among 11 tier-1 sites and 155 tier-2 sites of computing centers world-wide using grid/cloud computing paradigms [9]. The distributed nature of HEP computing poses some extra overhead when the data needs to be accessed from a site that is geographically very distant. This often happens since the grid infrastructure used at CERN, World Wide LHC Computing Grid (WLCG), spans from Japan to USA. Although, WLCG was designed before the era of cloud computing, also different cloud solutions have been studied and the OpenStack cloud suite has been found suitable for HEP computing [6, 29].

At a high level, cloud computing is a collection of servers, or hypervisors, that run mixed sets of virtual machines processing various workloads. The hypervisors share their processing and networking resources among a set of virtual machines. In the case of HEP data analysis jobs, which fetch constantly data from remote location, the network can become a bottleneck and cause delays for the analysis. The delays can have a big impact on overall performance.

HEP computing used in the LHC experiment is both CPU and data intensive. It comprises mainly the analysis of large amounts of both generated and measured data. The generated data is produced with simulations and compared against the real data measured by the experiments at CERN. In this use case description, we focus on the experiments in which the Helsinki Institute of Physics (HIP) is participating: the Compact Muon Solenoid (CMS) and A Large Ion Collider Experiment (ALICE). We only discuss off-line data processing, i.e. the data permanently stored after on-line filtering.

HEP computing consists of simulating and analyzing events in particle physics. Events depict high energy collisions of accelerated particles. In order to understand physics phenomena,

one must go through a very large quantity of measurement samples. In the case of the CMS experiment, and the search for the Higgs Boson tens of petabytes of data were processed. This was done with the resources of the World Wide LHC Computing Grid (WLCG).

ROOT files are most commonly accessed with the XRootD protocol, that runs on top of TCP, [15]. The performance of XrootD is a well-studied topic. These studies mainly focus on storage performance [14, 25], data federation [8], and scalability [10, 44]. Energy efficiency has not been considered, nor the effect of network delay on the performance of HEP computing. Therefore, we study in particular the effect of networking in cloud environment on the performance and, especially, energy efficiency of HEP computing. In HEP the data and computing is geographically distributed all over the globe.

3.2. Metrics

The key problem examined in the HEP use case is the performance of HEP software accessing locally and remotely located data sets. In particular, the goal is to understand the effect of latency and throughput to HEP job execution time and energy usage.

The HEP job execution time is measured from the start of the job execution to the end of the job execution. In case of multiple parallel jobs, the start times are synchronized and the ending time is the time when one of the jobs finishes the job execution.

The energy consumption is measured over the same period of time as the job execution time. The energy consumption is measured either with an external energy meter or by calculating from processor energy counters.

In addition, the network performance was measured during each job by recording the transferred bytes every second. Also, a more detailed analysis was done for a few jobs by collecting the network traffic and analyzing it with Wireshark (retransmissions, duplicate ACKs, etc). This more detailed analysis was done in order to understand the reasons for observed performance changes.

3.3. Experiments

There are different kinds of HEP workloads: simulation, reconstruction, analysis, etc. In this study, we used as our workload a process that transforms real physics event data into a more compact form that can be eventually used by the physicist on a standard PC hardware. The transformation process of a single event has two phases. In the first phase, the events are selected based on their suitability for the current analysis. Then, in the second phase, the event data is transformed and stored in a more space saving structure.

HEP computation uses special software packages. In the case of the CERN CMS experiment, CMS software framework (CMSSW) [11] is used. CMSSW is distributed to computing nodes with the CERN Virtual Machine Filesystem, CVMFS [27]. CVMFS is a centrally managed software repository that contains several versions of various HEP software frameworks. It can be mounted directly onto computing nodes. The software is cached locally when it is being used. In the case of CMS analysis, the job can cache about 1 GB of data or program code.

Since the location of the data and networking conditions have a big impact onthe performance of the computation, our goal is to measure this effect of distance on both computation time and energy consumption. In our tests, the OpenStack[1] cloud platform was used. OpenStack is a open-source platform for cloud computing consisting of individual projects, i.e. services, that are responsible for computing, networking and storage, among other services. OpenStack services are designed to be deployed on multiple nodes, with a scalable number of compute nodes. Virtual Machines running on OpenStack are called instances.

Measurements were done using three different cloud setups. In all the setups, we had a HEP client that reads data from storage server and does the transformation. In every test, the client was run in a cloud instance and the data server on a separate instance or on a separate server outside the cloud. The same workload was used in all the tests and its run times and energy consumption in different network conditions were measured. Tests were repeated several times to get reliable results.

Local data

First tests were done using a single physical server with suitable hardware for energy measurement. This single server OpenStack cloud installation was setup with DevStack[2]. In our DevStack installation, all of the OpenStack components run on the same hardware. The setup used the following hardware: Fujitsu Esprimo Q910 computer with quad-core Intel(R) Core(TM) i5-3470T CPU @ 2.90GHz, 8 GB of RAM and 8 GB of swap. As an operating system, it had Ubuntu 12.04.

Physics workload was run in an OpenStack instance. OpenStack instances can have different amounts of resources; number of virtual processors (VCPU), the amount of memory (RAM), and the size root disk and swap disk. These different configurations are called flavors[3]. Two types of flavors were used in the test environment: when running only one job the instance was assigned two virtual CPUs, 4 GB of memory and two GB of swap, and when running two jobs it was assigned one virtual CPU, 3 GB of memory and 1 GB of swap.

As a storage server for ROOT files, we used a ProLiant BL280c G6 blade computer with 16-core Intel Xeon CPU E5640 @ 2.67GHz and 68 GB of RAM. The server was in the same local area

[1] https://www.openstack.org/software
[2] http://docs.openstack.org/developer/devstack/
[3] http://docs.openstack.org/openstack-ops/content/flavors.html

network of computer science department of Aalto University as the cloud setup and were initially connected with 100MbE, which was upgraded to 1GbE for comparison. The server was installed with Ubuntu 14.04 and xrd server version 4.1.3.

We measured aspects such as processor (CPU), memory (RAM), power, cached data and network traffic statistics. Power measurements were done using Running Average Power Limit (RAPL) [17]. RAPL is an Intel technology that measures the power consumption in Sandy Bridge CPUs and above. Network traffic has been recorded with Tshark, which is the command line version of the Wireshark[4] packet analyzer.

The workload was run in varying conditions, including network delay, packet loss, packet duplication, packet corruption, limited network throughput, parallel jobs, and different operating system cache and CVMFS cache configurations. We use the term throughput to describe the actual network transport capacity, i.e., bits per second. The network limitations were simulated using the classless queuing disciplines (qdisc) tool, except for limited throughput simulated with Wondershaper[5]. The OS cache was cleared by freeing pagecache, dentries and inodes, and CVMFS cache with CVMFS tool `cvmfs_config wipecache`.

In addition to the previously described single node cloud system, we installed a separate cloud, which was able to run more Virtual Machines, but lacked the ability to measure energy consumption. We used the same blade hardware and operating system as previously for the storage server. In these tests, we used three blades, which were installed using Puppetlabs OpenStack module[6]. The OpenStack controller node, networking node and compute node were installed on their own hardware. All the nodes were connected to the same gigabit network switch. Data was served from the same node as where the controller was installed, but not within OpenStack. Physics analysis was run in an OpenStack instance. Tests with this cloud setup were done using varying amounts of Virtual Machines. In a similar way as in the single node tests, latency was simulated using qdisc.

Remote data

Two OpenStack (release 2015.1.2) installations were used in this study. One was located in CERN (Meyrin, Switzerland) and another in the Aalto University (Espoo, Finland). Both of the deployments used three physical machines. The physical machines used in CERN were Dell PowerEdge R210 rack servers, and in Aalto a HP blade servers in HP BladeSystem c7000 Enclosure. The roles of these three machines were computing, networking and other services. Power consumption was measured only over the physical machine running computing service. Both of the OpenStack instances were configured with routable IP addresses in order to be accessible from outside.

[4]https://www.wireshark.org/
[5]http://www.lartc.org/wondershaper/
[6]https://github.com/puppetlabs/puppetlabs-openstack

The OpenStack installation in CERN was used to run three identical computing jobs in parallel. Each job was run in a separate Virtual Machine (VM). The jobs accessed a 1.4 GB physics data file hosted in XRootD servers. Two of these XRootD servers were hosted inside the Aalto and CERN OpenStack installations. Two other XRootD servers were deployed onexisting OpenStack VMs: one in the CERN IT department and one in Kajaani in Finland.

In this study the duration of the HEP job processing and energy usage were collected. The duration was measured from the start of job processing until the first VM finished the processing. Energy usage was collected during this same period of time.

3.4. Results

The workload was the same in all of the tests. Depending on the setups, energy was measured either with an energy meter or by calculating from processor energy counters.

Local data

Running the workload is both CPU and network intense. Figure 6 shows the relation between power consumption and network traffic, both appearing in synchronous cycles. The base power consumption of the hypervisor is typically less than five watts. Results have a 30-second period of time at both ends when the virtual machine is running idle, i.e., no workload. This demonstrates the base power consumption and other base statistics on the hypervisor and the virtual machines. Remote resources from the XRootD server are downloaded in distinct parts. Most of the time there is no traffic between the server and the hypervisor. A closer look at the network throughput peaks shows that there is a short but constant peak that uses all the available bandwidth.

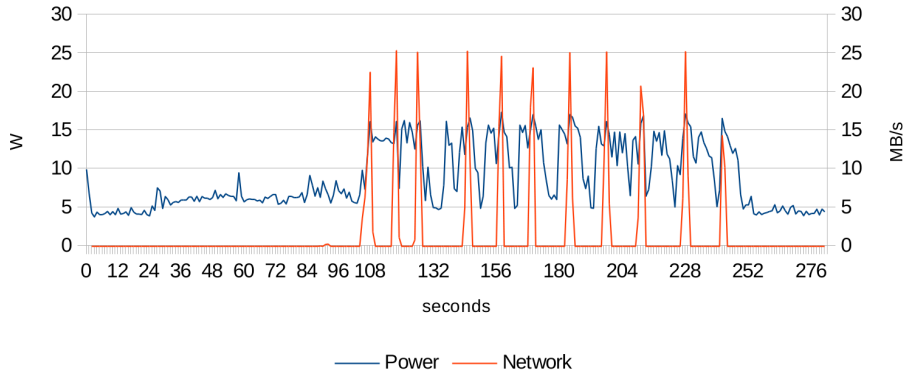

Figure 6: Relation between power consumption and network traffic on the hypervisor.

Network delays are typical to Wide Area Networks (WANs) and have a clear impact on the workload runtime. As Figure 7 shows, an increase of 75 milliseconds in network delay, causes

the run time to increase by over ten seconds. In addition, the power consumption increases by 34 percent. The effect is less evident when the same test is repeated in a network with more bandwidth. In Figure 8, we have the results of the same measurements in gigabit Ethernet. Run times are shorter with 1GbE, but energy consumption depends on the latency. The effect of bandwidth is summarized in Table 3. In a gigabit LAN, the run times decrease roughly by ten percent when compared to that of a 100 Mbps network. Latency does not seem to have a big impact on the energy consumption when using a1 GbE network, but has an impact in the 100 MbE network.

Table 3.: Comparison of execution times in different networks with no added delay.

Parallelism	Execution time (s)		Δ
	100MB/s	1 GB/s	
1 VM	281	255	-9 %
2 jobs, 1 VM	363	327	-10 %
2 jobs, 2 VMs	315	278	-12 %

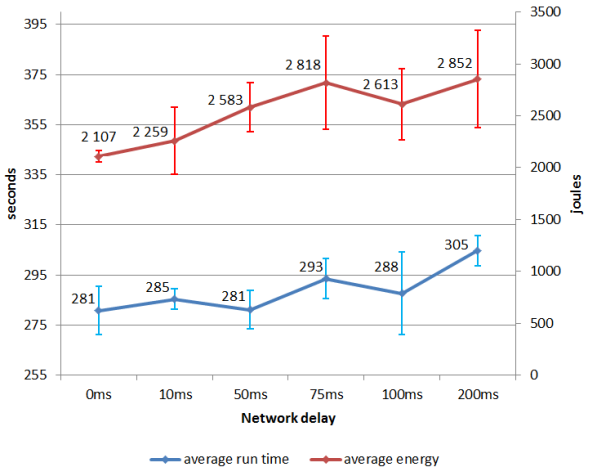

Figure 7: Comparison of workload run times and total energy consumptions with different network delays in 100Mb/s Ethernet.

The workload downloads and stores roughly 400 MB of data in the CERN VM File System (CVMFS) local cache. If the cache is empty, this data is downloaded from CERN servers at the beginning of executing the workload. Otherwise no data exceeding 10 KB in total is downloaded from CERN. As the total size of required tools is 400 MB, setting the CVMFS cache limit lower than that, affects the processing time and network traffic. As Table 4 shows, no data needs to be downloaded if the cache limit is high enough and the data has been previously cached. On the contrary, the workload cannot be executed at all if the cache limit is too low.

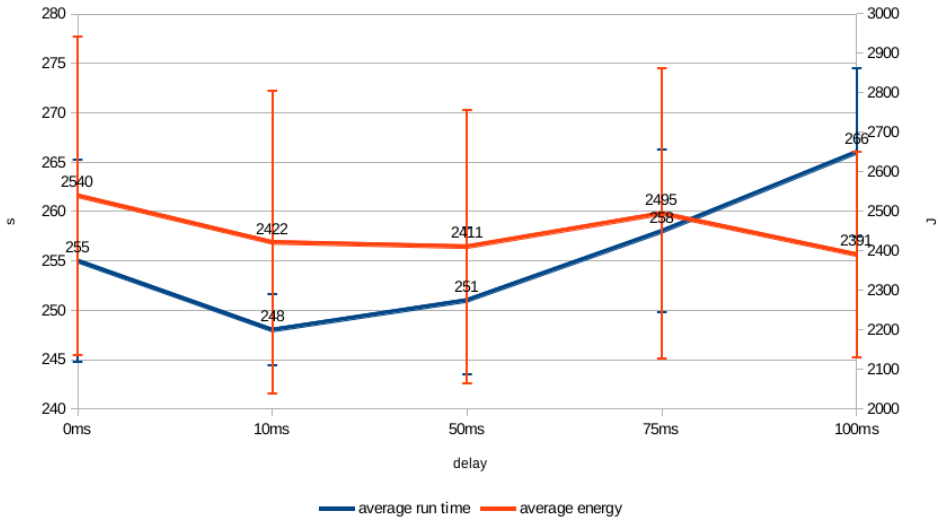

Figure 8: Comparison of workload run times and total energy consumptions with different network delays in 1Gb/s Ethernet.

In Table 5, we have a summary of parallel workload tests. It shows that the latency does not increase significantly even though ten Virtual Machines are sharing a single physical interface. Flows have different latencies depending on the direction, but this is similar with all workloads. There are some changes in the maximum values, but means and medians are about the same.

We tried to stress the shared interface even more by adding five additional Virtual Machines generating HTTP traffic by downloading large images from a university server. The results of this addition were similar to the results of the 10 VM workload.

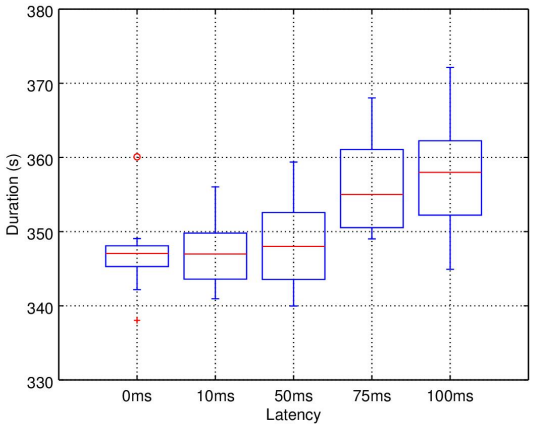

Figure 9: Job execution times with ten OpenStack instances in parallel and with added latency.

Similarly to previous single server measurements, we tested how added latency affects the execution time when running multiple Virtual Machines in parallel. Figure 9 shows how

Table 4.: Comparison of execution times and network traffic from CERN. When the cache limit is sufficiently less than 300 MB, the workload cannot be executed.
* First run after CVMFS cache clear
** The following runs (average)

Cache limit	Data from CERN (MB)		Execution time (s)
	*	**	**
200 MB	-	-	-
300 MB	422	300	249
500 MB	315	0	166
5000 MB	320	0	165

Table 5.: Round trip times between XRootD server and OpenStack instance in milliseconds.

	1VM	10VM
min	0.11	0.03
max	238.97	294.71
median	4.38	4.46
mean	7.67	10.14
stdev	25.32	32.08

execution times increase when we add more latency. This measurement was done with 1, 5 and 10 Virtual Machines. The effect of added latency was greater with 1 VM where 100 ms caused 6.9 % increase to execution time as with 10 VMs it is 2.3 %.

From the same 10VM tests we got the throughput values, that are shown in Figure 10. These results show a relation between latency and throughput as the maximum throughput decreased 41% when 100ms latency was added.

Remote data

Figure 11 and Figure 12 show job run times and energy usage, respectively, when data is hosted on an XRootD server in different physical locations. The energy usage is directly related to the processing time. Only the OpenStack energy usage is slightly higher because the measurement includes also the XRootD server hosting. Thus, it would be possible to estimate the energy usage by measuring only the processing time.

There are at least two possible causes for the differences between sites: the network latency and throughput. The network latencies to Aalto and Kajaani are 46.1±0.3 ms and 49.6±0.2 ms, respectively, while inside CERN the latencies are less than one millisecond. The network

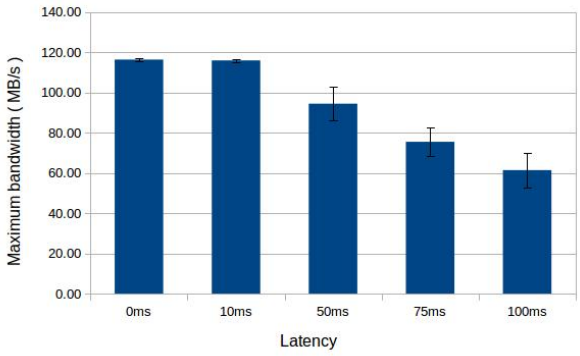

Figure 10: Maximum throughput with ten OpenStack instances in parallel and with added latency.

throughput to Aalto and Kajaani are 23.2±2.7 MB/s and 20.0±1.6 MB/s, respectively, while inside CERN 58.3±4.9 MB/s.

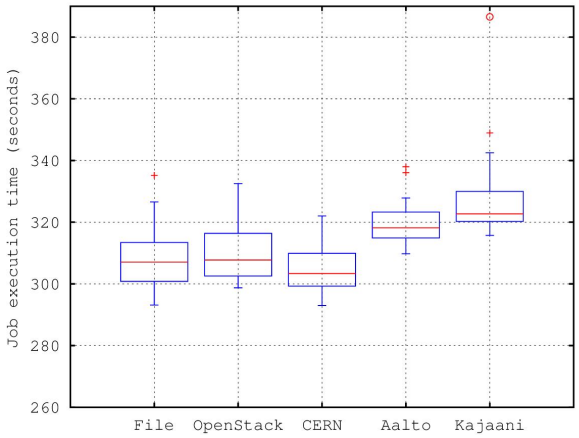

Figure 11: Job execution times of OpenStack VMs.

The effect of network throughput and latency can be examined by comparing local and remote sites with a 20MB/s fixed throughput that is achievable with both sites. Figure 13 shows that there is around 12.5 % difference between CERN and the throughput-limited Kajaani site. Throughput limitation removes 70 % of this difference so the effect of latency seems to explain around 30 % of the difference. A detailed analysis of the network traffic showed that only 0.01 % TCP packets were retransmitted and also the TCP window size increased quickly to around 3 MB. Thus network problems do not explain the observed differencies. With the current HEP software stack the best option is to prefer nearby data sources with low latency.

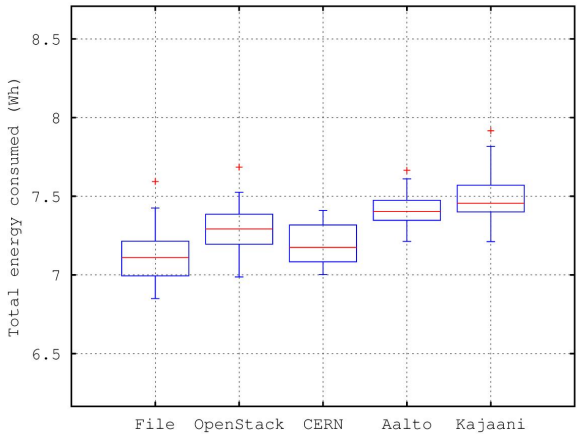

Figure 12: Energy usage of OpenStack VMs.

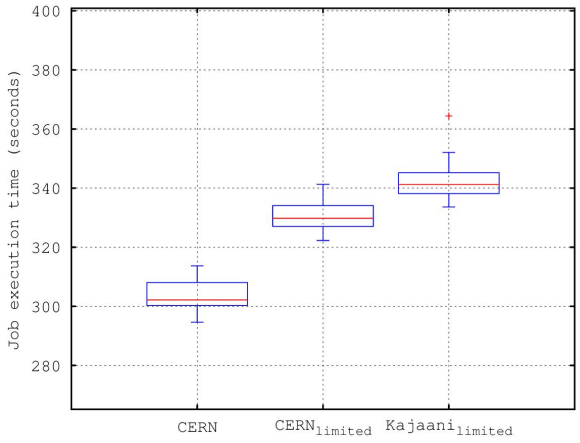

Figure 13: Job execution times with throughput limits.

3.5. Summary

High Energy Physics computing at CERN uses a large computing grid/cloud distributed around the world. This naturally poses long distances between the sites and slows down the network connections among them. To alleviate this, we studied how networking performance affects computing performance and energy efficiency on High Energy Physics computing in an OpenStack cloud testbed. We used both simulated network latencies in laboratory network and several geographically distant sites connected by the Internet to measure how different latencies change computing performance when processing HEP workloads.

Our results indicate that the network latency, either caused by a simulator or physical distances between the sites, has a negative impact on the computing performance. High latency both increases run times and the total energy consumption. Additionally, we also noticed that the

contribution of latency, to the execution time and energy consumption of a computation job, increases when bandwidth is small. Parallelism, multiple cloud instances sharing the limited network resource, also adds more latency and increases job run times.

The obtained results reflect the current software environment used for HEP job processing. New data transfer protocols or advanced caching mechanism could diminish the observed differencies. Instead, the used network infrastructure and computing hardware is unlikely to change significantly in the near future.

4. Use case 4: Network Function Virtualization in NGPOP

4.1. Use case description

Worldwide deployed Home Gateway (HGW) devices can provide different sets of functions, starting with very simple ones (e.g. mostly physical LAN/WAN interfaces and L2 forwarding) up to complex networking features. This is mainly because of several reasons like Internet Service Provides' (ISPs) networks architecture specific requirements, offered service catalogue, various technical limitations, etc. Since HGW types can differ significantly OAM operations became very complex and it is very difficult to manage a large set of HGW types, especially for large scale deployments. Also the introduction of new features or services is often an issue. It can be slow or even not possible on some devices due to limited resources and capabilities.

The primary goal of introducing virtual Customer Premise Equipment (vCPE) concept is to decouple devices' functionality installed in customers' premises (which customers are given by ISPs) from the lifecycle of network applications and services. The idea of vCPEs is to move most of the logic from user devices "to the network" making them simpler and cheaper. With this approach, significant benefits for both operator and customer can be envisaged:

- CAPEX and OPEX reduction – through replacing expensive user equipment with cheaper boxes (not requiring a constant development of their functionality) and possible reduction of the total number of devices needed to be installed at customer premise by the operator.
- OAM simplification – through evolving from heterogeneous environment of boxes (each requiring individual OAM procedures and patterns) towards more unified operational model where different devices support the same limited set of functions.
- TTM (Time To Market) improvement – through shortening the time for implementing new network and service functions to end-users.

Typical examples of virtual CPE appliances are:

- virtual Set-Top-Box (vSTB)
- virtual Home Gateway (vHGW), also known as virtual Residential Gateway (vRGW)

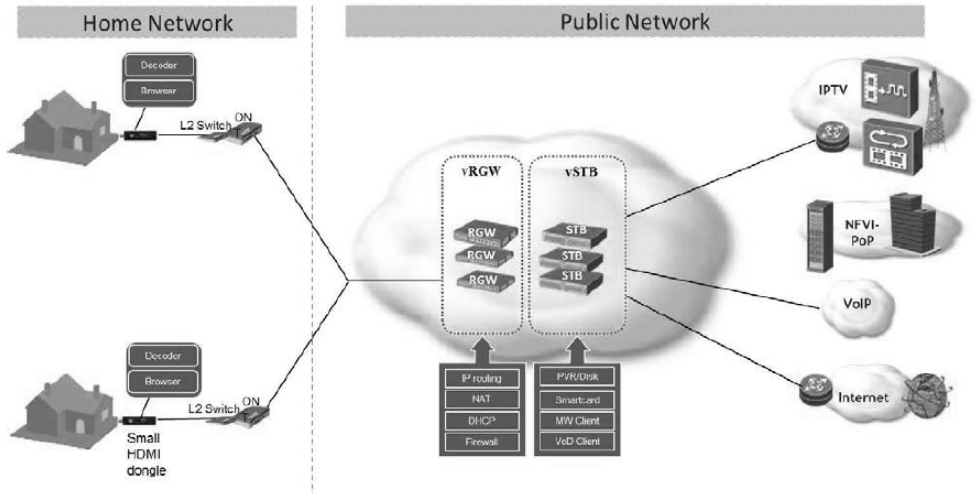

Figure 14: Virtual CPE concept illustration (source: [1])

The concept of vHGWs assumes that the most of traditional L3-aware HGW functionality is shifted from the physical device to network resources managed by the ISP. The end-user HGW becomes then a much simpler device with a limited set of functions supported. At the same time, most of the logic is executed "in the network". It can be accomplished for instance through launching network functions on VMs instantiated on IT resources (COTS servers) in a next generation PoP node or datacenter. This is in line with the NFV-based approach. It brings opportunities for noticeable advantages for both ISP and customers:

- Applying the vHGW approach allows ISPs to achieve significant cost savings. Since the aggregated cost of the CPEs (Customer Premise Equipment) that end-users are given by the operator has a great impact on the total expenditures, using a simple, cheap L2 box instead of sophisticated L3 device is a desirable change.
- Network-located vHGW brings a high flexibility and TTM advantage in implementing new network and service functions to end-users. There is no need to upgrade HGW firmware whenever a new feature is required because it can be done "in the network" in a more automated and faster way – through modifications in the software installed on VMs.
- Since L2 traffic is forwarded by a simplified HGW located at the user premise up to the vHGW instance located "in the network", all end-devices from user home network can be identified by their MAC address. This brings the possibility to process traffic flows corresponding to each end-device individually in the sense of Service Function Chaining (SFC) which is a significant improvement as compared to the traditional L3 HGW scenario.
- Applying the decomposition of HGW functionality into functional blocks executed on VMs and introducing capabilities for processing particular traffic flows individually can facilitate troubleshooting CPE issues.

4.2. Metrics

Since network-based aspects are crucial for the SSICLOPS vHGW solution, the following network-related metrics have been identified for the purpose of the considered testbed:

- bandwidth (throughput)
- RTT
- jitter
- packet loss

It is worth noting that some of the above mentioned metrics (e.g. bandwidth or jitter) represent general terms in telecommunications and therefore they are relevant for a wide range of technologies, systems and network types (sometimes with a specific meaning). However, in this document they are considered in the context of IP networks as the SSICLOPS vHGW solution is implemented in a network environment based on the TCP/IP protocol suite.

Unlike traditional analogue telecommunications where bandwidth refers to the frequency domain utilized for transmitting signals (for various technologies) and therefore is measured in Hz, in IP networks this term is more related to maximum achievable throughput between two end-points in the network. Since in general those transmission capabilities can be different for two directions (e.g. due to asymmetric links, congestions, traffic engineering mechanisms applied on the network nodes like rate limitation settings, etc.) the bandwidth is measured individually as maximum download speed (bit-rate) and maximum upload speed in the particular network topology, configuration state and network conditions.

The RTT (round-trip time) metric is be described as the length of time the IP packet needs to traverse from source to destination plus the length of time required to receive an acknowledgment (at the source) of reception the packet at the destination point. It is worth noting that RTT is essentially different from E2E delay (end-to-end delay) which is a one-way delay between two end-points. E2E delay consists of several delay components like processing delay, queuing delay, transmission delay and propagation delay considered as the sum of each delay component type over all network nodes and links forming the E2E path for the IP packet from a given source to a given destination. E2E delay is a direction-oriented metric which means that E2E delay value for the path from the source to the destination can be different from the E2E delay value for the opposite direction path. It is because both paths can be different, which is usually the case for Wide Area Networks. Unlike E2E delay, RTT is independent of the directions. It is simply the time required for message exchange between two end-points in the sense of IP-based transmission.

Jitter is the variation of network delay (i.e. E2E delay) between two end-points. This term usually causes confusion because there is no common definition for jitter in the area of IP networking. As a consequence, also the methodology for measuring jitter can be different. One approach is

to define jitter as a standard deviation of E2E delay distribution over a period of time. Another possibility is to directly measure the Instantaneous Packet Delay Variation (IPDV) value, which is a difference in E2E delay between successive packets. Such measurements should be conducted in a given time period in order to obtain a distribution of results. Jitter can be defined in such case as the maximum or the mean value of that IPDV distribution.

Unlike jitter, packet loss has a much clearer definition. Packet loss is the percentage of packets lost during transmission from source to destination with respect to the total number of packets sent. In general, several causes of losing or dropping packets in IP networks can be identified. Some of them could be link-related (issues with distance, propagation conditions) while other ones are more node-related (congestions, hardware or software issues). Dropping the packets intentionally by network devices also might be the case in some situations.

4.3. Experiments

There are ongoing works in the area of Home Gateway virtualisation. The Broadband Forum Architecture and Migration working group has recently finished standardization works with TR-317 document [12] which defines an architectural model named NERG (Network Enhanced Residential Gateway). This model assumes that functionality supported by traditional HGWs can be distributed between two components (see Figure 15):

- Bridged Gateway (BRG)
- virtual Gateway (vG)

The aim of the TR-317 document is to specify the requirements for both BRG and vG as well as E2E requirements for supporting legacy and new broadband services deployed in the novel architectural framework.

The BRG unit, which is supposed to be located at the customer premise, becomes a simplified version of the traditional HGW device – with a limited set of functions. It mainly provides various physical interfaces, e.g. WAN, Ethernet LAN, WiFi, FXS, USB, etc. From the traffic processing perspective it performs L2 forwarding including MAC learning and QoS as well as it supports mechanism for customer authentication. On the other hand, most of the HGW logic is executed by the network-based vG. The considered set of functions may cover DHCP internal server, DNS proxy, NAT, ALG, port mapping and triggering, QoS, traffic shaping, security functions (firewall, antivirus, parental control), DLNA, UPnP proxy, etc. The current version of TR-317 document does not specify the technical environment for implementation of vG entities. The possible options are the following:

- vG instances reside on well-known legacy ISP's equipment
- vG instances are hosted on COTS servers

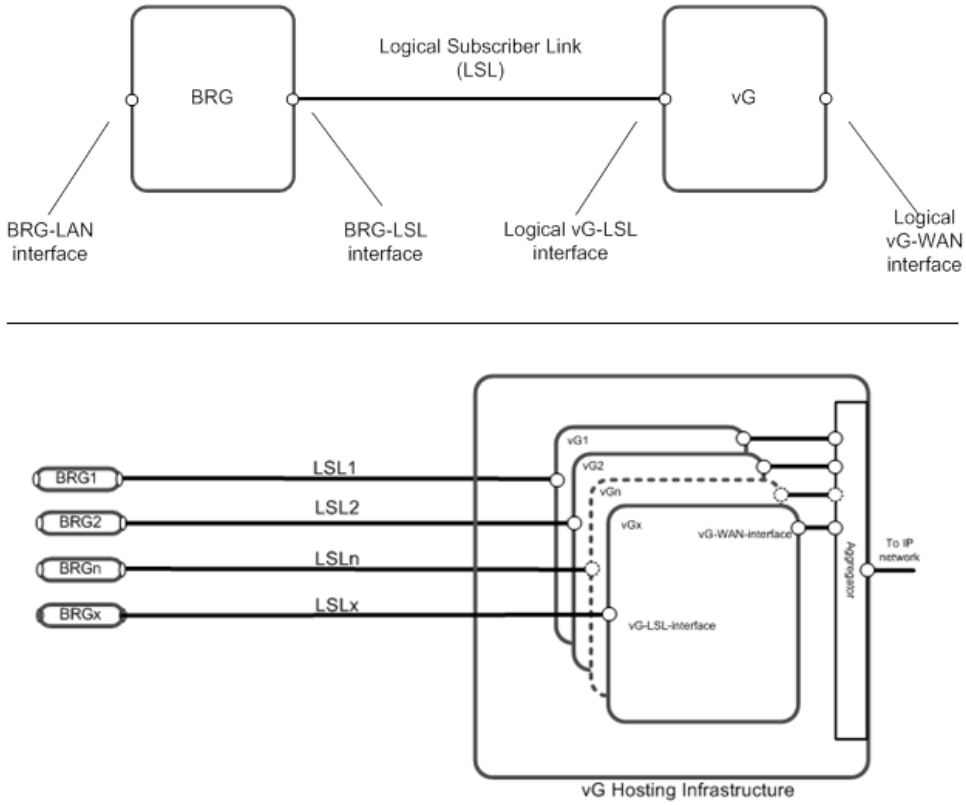

Figure 15: NERG overview (source: [12])

- vG functionality is distributed between ISP's network nodes and COTS servers

The traffic exchanged between customer end-devices and the network is passed through the Logical Subscriber Link (LSL) which is a logical point-to-point L2 connection between the BRG and the vG. This is a customer specific connection, i.e. each BRG is connected to its corresponding vG instance via a dedicated LSL link. It is worth noting that the LSL represents a logical connection, not a direct one. This means that, alongside the physical path between BRG and network-located vG, instance other network entities are assumed to be present, e.g. Access Network node, Aggregation Network node, BNG, etc. Depending on architectural assumptions and configuration scheme which is chosen to be applied in the network LSLs can be established in different fashions (flat Ethernet with VLAN tags or GRE, VxLAN, L2TPv3 tunnels).

The BBF TR-317 document provides the NERG architecture specification (with detailed requirements concerning BRG and vG functionality) which is supposed to be the main recommendation for future commercial deployments in ISP' networks. However, since the SSICLOPS vHGW is more a PoC-alike solution than a commercial application, it does not need to be fully compliant with the considered NERG model:

- There is no need for supporting legacy services like VoIP and IPTV. For simplification, only Internet access service is assumed to be provided to end-users.
- SSICLOPS implementations of BRG and vG entities do not have to meet all the requirements regarding network features and mechanisms which are specified for BBF NERG components.
- Management aspects are out of scope of the SSICLOPS vHGW. BRG and vG entities implemented within the testbed are not assumed to be managed by any kind of typical EMS/NMS platform.
- Since QoS-related traffic engineering is not a critical issue addressed in SSICLOPS, not all QoS mechanisms and functionality considered within BBF NERG architecture are mandatory to be implemented in the SSICLOPS vHGW solution.

On the other hand, the following aspects were taken into account when designing the SSICLOPS vHGW testbed:

- One of the key aspects is the physical infrastructure for vG deployment. For the SSICLOPS vHGW it is assumed that functionality covered by the vG component is to be executed as VNFs (Virtual Network Functions) on VMs hosted on COTS servers since they are a target technical environment for the SSICLOPS testbed. For that purpose a SSICLOPS NFVI has been established. This NFVI hosts different VNFs including dedicated vG VMs for every user. Some of those network functions can form a SFC domain. The NFVI can also host control plane components, e.g. SDN controller, orchestration functions' modules, etc.
- In order to avoid complexity the SSICLOPS vHGW solution cannot to be used by real end-users. Instead of having physical BRG devices aggregated in the sense of an access network and connected to the SSICLOPS vHGW platform BRG, functions are emulated on VMs.
- The SSICLOPS vHGW architecture leverages SDN and NFV mechanisms. This facilitates a flexible service chaining. In that context the following aspects are taken into account:
  - VNFs being part of a service chaining domain and having the possibility to add new functions
  - SFC patterns for the different traffic flows incoming from user end-devices
  - orchestration-related aspects
- The SSICLOPS vHGW implementation is based on open source components.

Figure 16 shows the basic high level conceptual view on vHGW architecture in SSICLOPS.

The current SSICLOPS vHGW testbed is based on two compute nodes, Dell PowerEdge R630 servers. The hardware configuration is as follows:

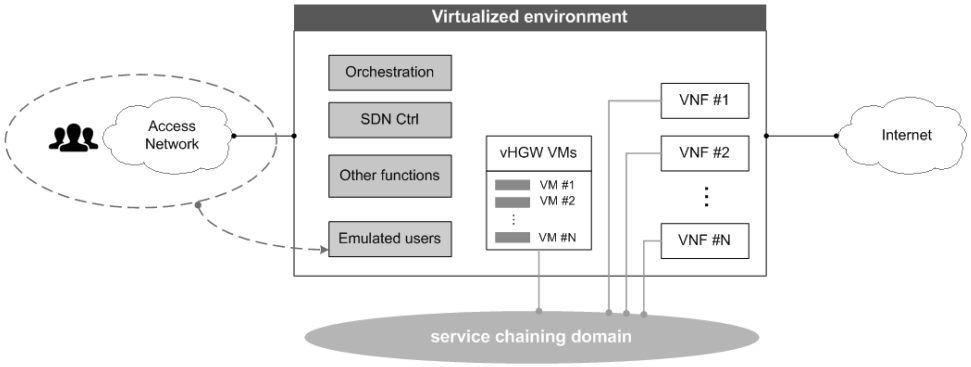

Figure 16: High level conceptual view on vHGW in SSICLOPS

- 2 x Intel Xeon E5 2.5 GHz CPUs with 12 cores/24 threads for each physical machine
- 128 GB of RAM per physical machine
- 600 GB of available logical Hard Drive space in RAID 10 configuration (physically 2x600 GB Hard Drives having 15k rpm each)

The mentioned hardware infrastructure is managed by an OpenStack (Kilo release) platform. The test environment will be extended in the future with other compute nodes having a similar configuration to the existing ones.

For the purpose of the SSICLOPS vHGW implementation, appropriate subnetworks have been created utilizing built-in networking mechanisms natively offered by OpenStack Neutron software component (virtual networks and routers). Figure 17 shows the high level view on considered network topology.

As depicted in Figure 17, the network design consists of several subnetworks. Each one emulates a different networking domain of an ISP' broadband IP network architecture. Thanks to such an approach the SSICLOPS vHGW testbed network topology reflects the idea of hierarchical structure which is relevant for both legacy and modern IP-based broadband networks. Thus, the following subnetworks have been created within the vHGW testbed:

- subnet #1 (end-users)
- subnet #2 (PoP #1)
- subnet #3 (PoP #2)
- subnet #4 (control plane software)
- subnet #5 (transport network)
- subnet #6 (external network - with Internet access)

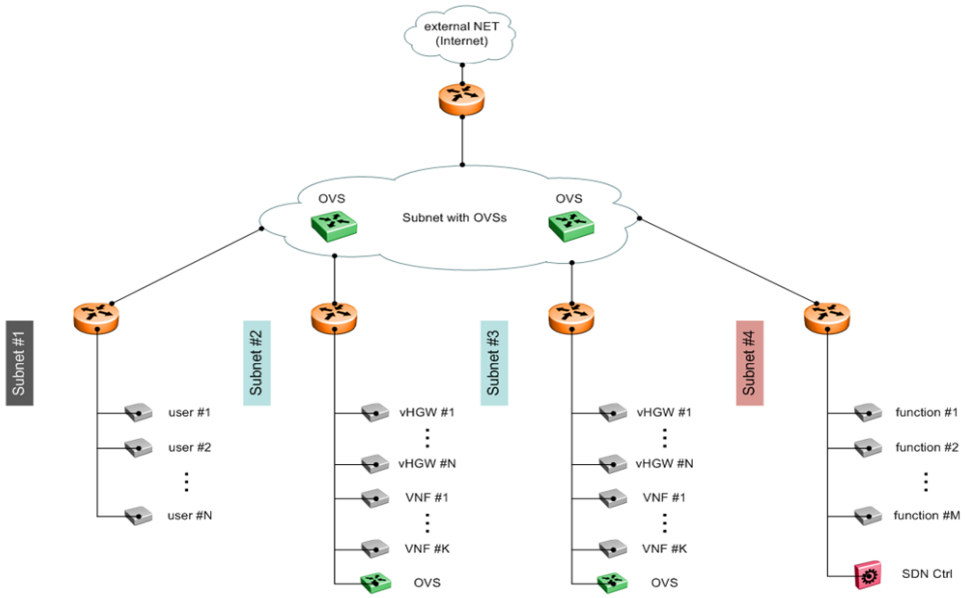

Figure 17: High level view on network topology within SSICLOPS vHGW testbed

The subnet #1 emulates the end-devices (like PCs, laptops, smartphones, etc.) of end-users. Every user's home network equipment can be emulated on a dedicated VM (one VM per user) or one common VM for multiple users can be utilized as well. In order to accomplish that an OVS (Open vSwitch [30]) is installed on the VMs instantiated in the considered subnet. OVS is an SDN-capable virtual multilayer switch which allows to create multiple software-based bridges. Those bridges can be individually or massively programmable e.g. via the OpenFlow protocol. An OVS bridge can act as a BRG unit according to NERG architectural model described briefly in the previous section. On one hand it provides the interfaces (LAN side) for connecting emulated user end-devics while on the other hand it is capable to establish the LSL link (WAN side) to the instance of vHGW corresponding to the particular user. The LSL link can be realized e.g. in the sense of VxLAN tunnel. Optionally, also an additional VLAN tag can be used. Since the OVS bridge is SDN-aware, it can be easily configured to forward traffic incoming from different end-devices at L2 up "to the network" through the LSL in order to follow the NERG architecture principles. Emulation of user end-devices leverages native Linux networking mechanisms, especially Linux network namespaces. Obviously, the full hardware and software specification of any end-device is not reflected through this simplified emulation. It is more about providing a TCP/IP stack of an emulated end-device. Having applied Linux network namespaces all end-devices are separated i.e. each of them acts as independent node in the testbed network.

Subnet #2 represents the internal network of single next generation PoP (Point of Presence). This concept aims at introducing a new class of functional node into the ISP network architecture. The considered node is assumed to integrate a wide range of network and service functions (e.g. access termination and aggregation, BNG, EPC, vCPE, CDN, etc.) into a single topological entity. Such

functions are originally executed across existing network domains, including access, aggregation, core network segments and service platforms, as well as end-user networks alongside the end-to-end (E2E) path. Having applied the PoP-based approach, selected functions can be collocated and hosted in one physical location, which opens the door for optimization in the general sense. PoPs offer a wide technical landscape for implementing various functions in the sense of the NFV-based approach through leveraging virtualization techniques and performing intra-node SDN-based service chaining. That is also the case for the SSICLOPS vHGW implementation. The considered subnet #2 hosts VMs which represent dedicated vHGW function sets corresponding to appropriate end-users as well as VMs with shared vHGW logical blocks. The key component of the subnet is the SDN-aware switch (deployed as a VM with OVS installed) which enables flexible traffic forwarding within the PoP. To accomplish that, an overlay network is built between each of the vHGW-related VMs and SDN switch VMs through establishing VxLAN tunnels. Such, approach allows to configure VMs to exchange user traffic not directly but via the SDN switch and, in consequence, service chains can be spread in a flexible manner (see Figure 18).

Subnet #3 is realized in a similar way as subnet #2. The only difference is that it acts as another PoP entity. The presence of two independent PoP nodes within the SSICLOPS vHGW testbed enables performing workload migration tests within the testbed. In particular, VMs representing some vHGW functions can be moved from its nominal PoP to another one and then one can assess the impact on performance caused by such an operation. In order to emulate transmission-related aspects of the network, subnet #5 has been created. It is comprised of OVS switches which are capable to introduce bandwidth or latency modifications. In such a configuration distance-dependant issues existing in real network environment between users' premises and various network nodes (like PoPs) can be taken into account for the tests performed within the considered testbed. In order to allow end-devices residing in subnet #1 to access the Internet, subnet #6 has been created.

Subnet #4 host VMs with various control plane applications like the SDN controller which is responsible for managing flows traversing accross OVS-based overlay network within the SSICLOPS vHGW testbed. Also VMs performing orchestration functions reside there.

The tests conducted in SSICLOPS vHGW environment were performed using the following tools:

- *ping* tool
- *iPerf* software: both *iPerf2* [21] and *iPerf3* [22] were used

One of the most popular methods to measure RTT is to use a *ping* tool. *Ping* is a software utility available in most of operating systems both dedicated for user end-devices like MS Windows or Linux and network devices like Cisco IOS. *Ping* is commonly used to test the mutual reachability of hosts accross IP network. It measures the RTT by sending ICMP messages from the originating host to a given destination point in the network that are echoed back to the source.

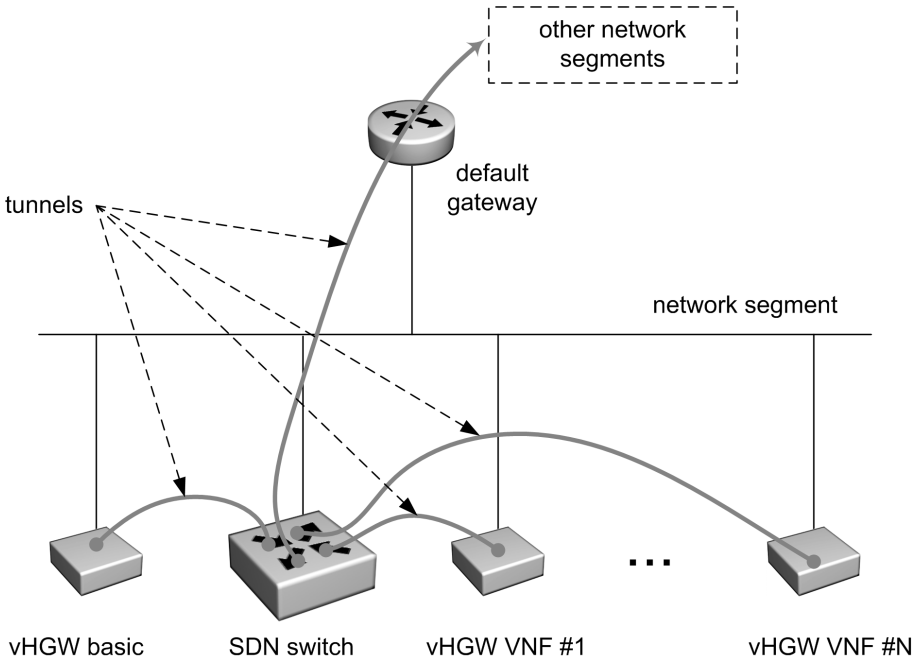

Figure 18: PoP internal overlay network

iPerf is a an open source software commonly used as a network testing tool suite. *iPerf* is capable to create TCP or UDP streams being transmitted between two selected end-points in the network. Through running appropriate tests utilizing different capacities of TCP and UDP *iPerf* allows to measure a wide range of network metrics like bandwidth, jitter or packet loss in order to provide statistics about network links. The results can be displayed on the command line, forwarded to the GUI front end (called *jPerf*) or exported to the *json* file (feature available only in *iPerf3*) for further data processing. The *iPerf* is a relevant utility for both wired and wireless networking environment and can be installed on multiple operating systems.

For the purpose of the tests described below, both utilities were used in their advanced mode which allows to execute commands with set of various parameters (e.g. time intervals between samples, total test duration, test modes, format of results, etc.).

The aim of the first test (TEST #1) was to measure the maximum throughput between the SDN-switch VM (VM #1) and another "generic" VM (VM #2) which was supposed to host any kind of VNF within the SSICLOPS vHGW testbed. The test was performed using *iPerf* tool by transmitting TCP streams between two VMs over 5 minutes (300 seconds) period. Two different test cases were taken into account:

- case A - SDN-switch VM and VNF VM were instantiated on the same compute node
- case B - SDN-switch VM and VNF VM were instantiated on different compute nodes

The objective of the second test (TEST #2) was to measure RTT between the SDN-switch VM and VNF VM. The test was performed using *ping* tool by sending ICMP messages from VM #1 to VM #2 over a 5-minute period. Again, two different test cases (the same as for the previous test) were considered.

The aim of the third test (TEST #3) was to measure jitter and packet loss between the SDN-switch VM and VNF VM. The test was performed using *iPerf* software by transmitting UDP packets between two considered VMs over a 5-minute period. Different values of UDP packet data rates were configured during the test (1 Mbps, 10 Mbps, 100 Mbps and 1 Gbps). Also for that test, two subcases (case A and case B) were analyzed. *iPerf* uses the IPDV-based approach for measuring jitter.

Another test (TEST #4) was performed in order to assess the potential impact of intra-PoP processing on delay. The test scenario is depicted in Figure 19. The SDN-switch VM was programmed to execute the traffic forwarding between two end-points (input and sink) connected to two different Linux network namespaces which were assigned to the same logical subnet in order to ensure mutual reachability on IP layer in easy way. The SDN-switch (with appropriate configuration) was forwarding the traffic from the input to the sink in the sense of SFC via VNF VMs (i.e. from SDN-switch to a VNF, then back to the switch and to the next VNF and so on). The number of VNFs in the chain was increased one by one for each test subcase. The VNFs were configured to immediately return the received packet to the SDN-switch without performing any VNF-specific packet processing. The reason for that was the fact that the aim of the test was to assess the delay introduced by intra-PoP forwarding plane, not by VNF processing which in general can be different for different VNF types. During the test, RTT between the input and the sink was measured for different length of SF chain i.e. number of VNFs forming the chain.

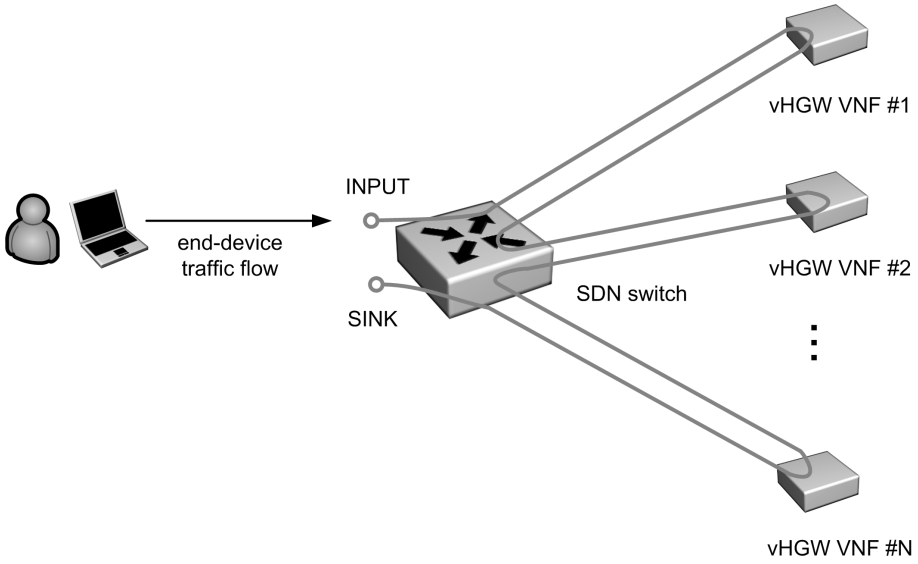

Figure 19: Service function chaining (SFC) test scenario

4.4. Results

The results of the TEST #1 are presented in Figure 20 and Figure 21, the results of the TEST #2 are presented in Figure 22 and the results of the TEST #3 are presented in Table 6.

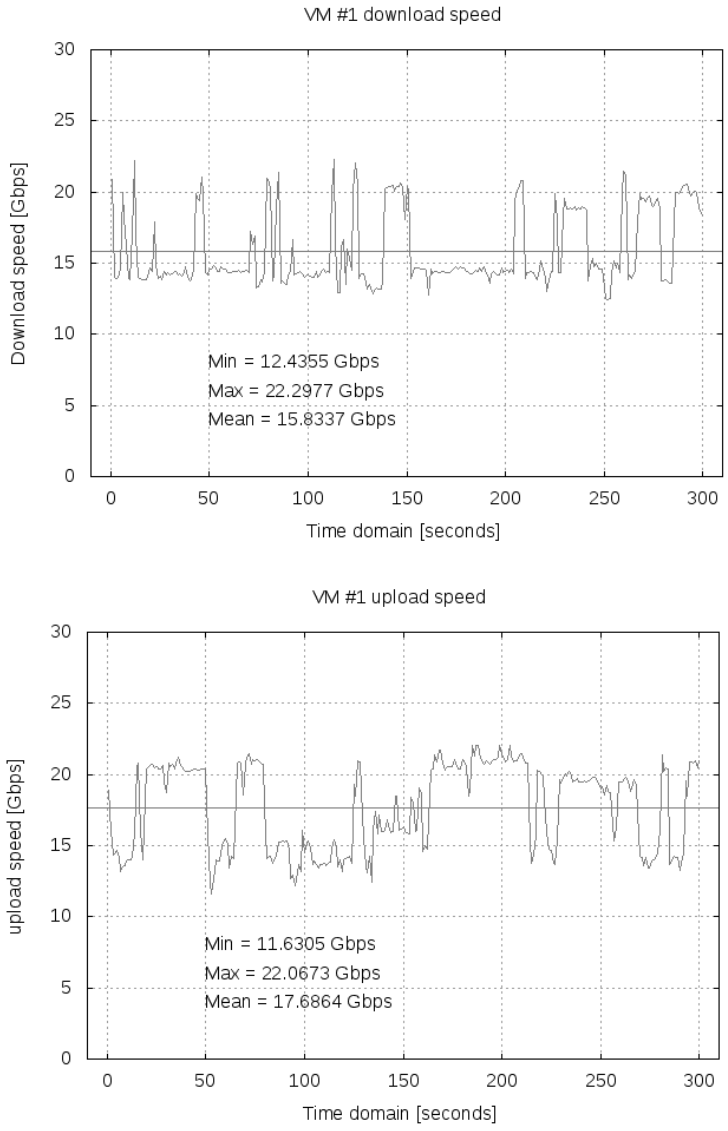

Figure 20: Maximum download and upload speed of VM #1 - case A

The results presented in Figure 20 and Figure 21 show very high bandwidth capabilities (gigabits per second) for transmission between SDN-switch VM and VNF VM for both considered test cases. This is because for case A both VMs were instantiated on the same compute node and in consequence the traffic between VM #1 and VM #2 can have been exchanged internally, i.e. without outgoing from the physical server.

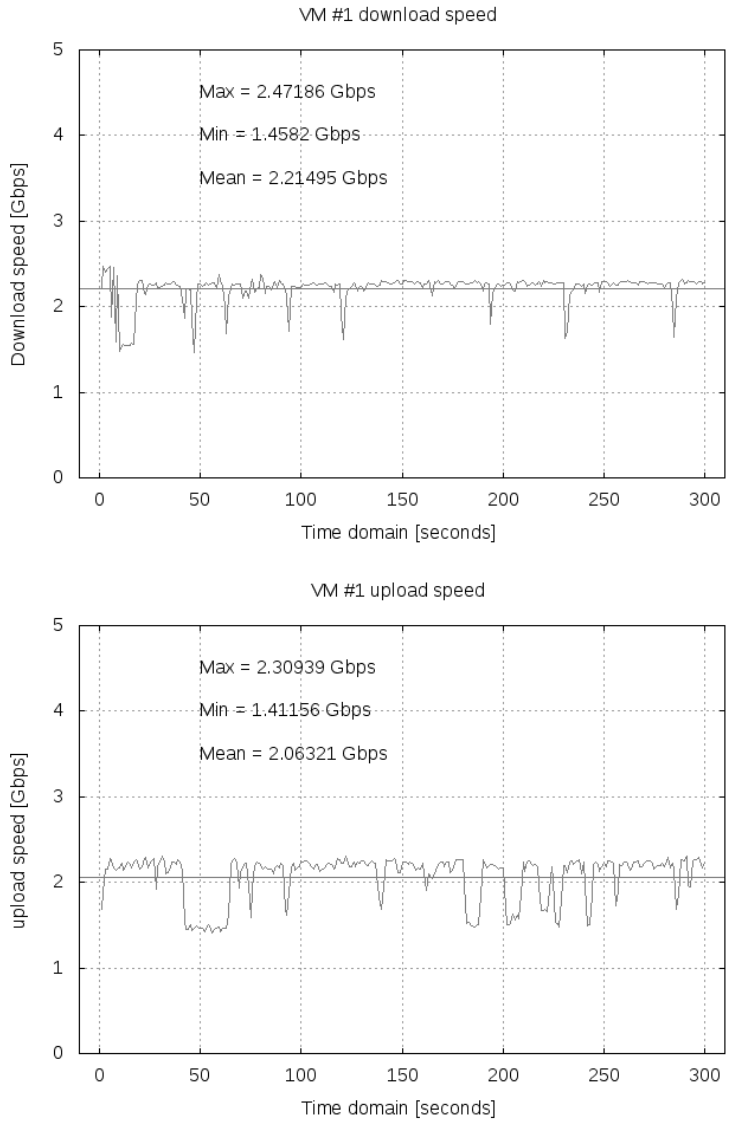

Figure 21: Maximum download and upload speed of VM #1 - case B

Also for RTT metric a certain difference in obtained results for cases A and B can be observed (see Figure 22). But still, for both cases measured values meet the RTT requirements for LAN (RTT less than 1 ms).

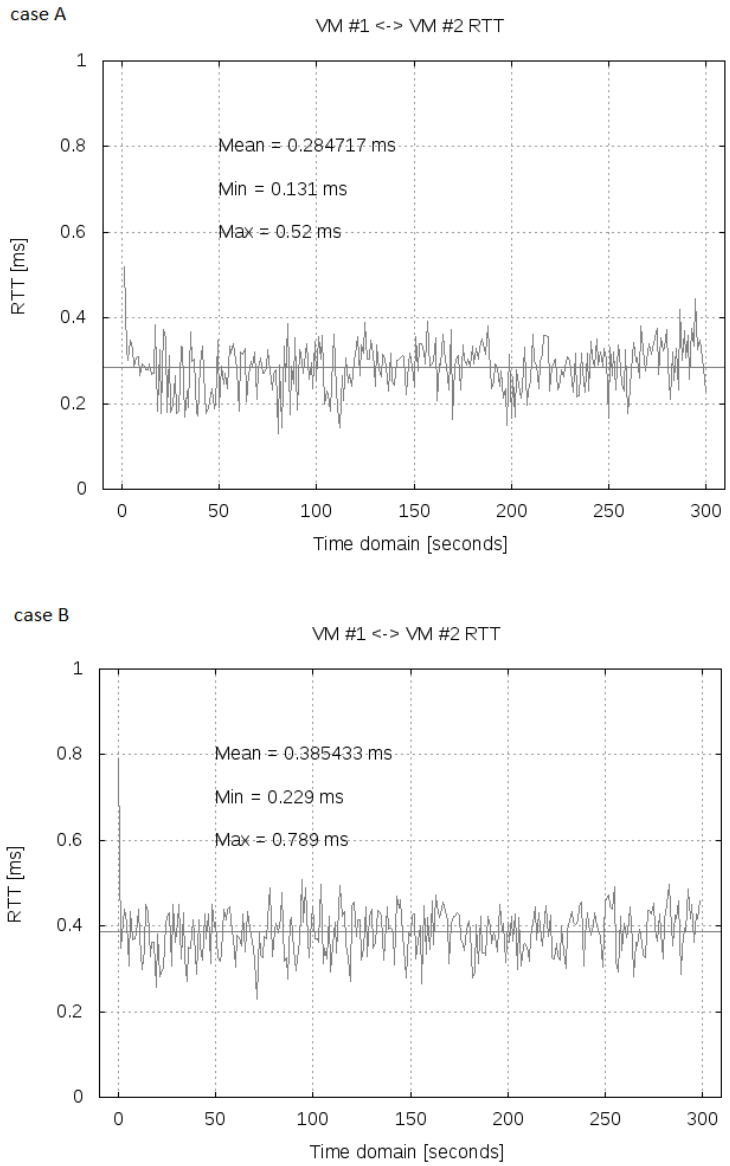

Figure 22: RTT between VM #1 and VM #2 - case A & B

Concerning jitter and packet loss there is in practice no difference between cases A and B. The jitter level is very low (no more than 0.05 ms for any test subcase). The similar situation is for the packet loss metric. It starts to exceed the level of 1% for flows transmitted with the rate close to 1 Gbps or higher. It is much less than 1% for flows of lower rate (see Table 6).

Table 6.: Jitter and packet loss for transmission from VM #1 to VM #2

UDP flow rate [Mbps]	case A		case B	
	jitter [ms]	packet loss [%]	jitter [ms]	packet loss [%]
1	0.033	0	0.022	0
10	0.012	0	0.012	0
100	0.010	0.000117	0.010	0.000078
1000	0.017	1.65	0.011	1.95

Figure 23 shows the results of the TEST #4. As can be observed RTT values increase together with the number of VNFs taking part in the SFC. This function can be approximated by a linear regression line of a slope value equal to 0.282 and an intercept value equal to 0.473.

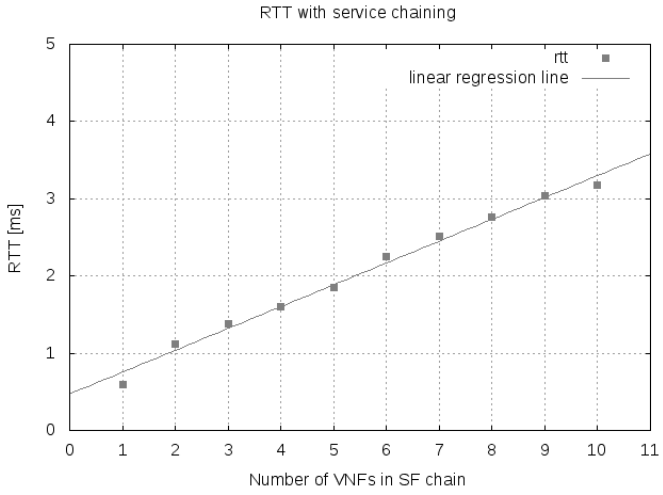

Figure 23: RTT with service chaining

The obtained results show that an overlay SDN-based PoP internal network and SDN-switch VM (leveraging OVS) as well as an underlay OpenStack-based virtual network established for the purpose of the SSICLOPS vHGW testbed can be considered as logical network structures of sufficient stability and deterministic delay. However, it is easy to notice that for case A those capabilities are significantly higher (about 15 Gbps versus 2 Gbps for case B).

4.5. Summary

The idea of an NFV-based vHGW assumes that most of the traditional advanced HGW logic is shifted from the physical device to network resources (e.g. COTS servers being part of next generation PoPs) managed by the ISP. For the purpose of the SSICLOPS vHGW solution NFVI has been established. This implementation is based on open source solutions and it is capable to host a set of VNFs. Some of those network functions can form a service chaining domain. The NFVI can host also control plane components, e.g. SDN controller, orchestration functions' modules, etc. Also appropriate subnetworks have been created utilizing built-in networking mechanisms natively offered by OpenStack Neutron software component (virtual networks and routers). This network topology reflects the idea of a hierarchical structure which is relevant for both legacy and modern IP-based broadband networks. It emulates both the end-devices of subscribers and network entities like next generation PoPs (a new class functional node in the ISP network architecture). The key component of these PoPs is an SDN-aware switch (deployed as a VM with OVS software installed) which enables flexible traffic forwarding within the PoP. To accomplish that, an overlay network is built with the use of VxLAN tunnels. Since networking aspects are very important for the the SSICLOPS vHGW solution, the network-related metrics like bandwidth, RTT, jitter and packet loss have been identified for the purpose of the considered testbed. Several tests have been conducted. Based on the obtained results it can be observed that the SSICLOPS NFVI offers high bandwidth capabilities (gigabits per second) for transmission between VMs. Measured RTT values meet the requirements of typical LAN RTT (RTT less than 1 ms). Also the jitter level is very low (no more than 0.05 ms). Packet loss starts to exceed the level of 1% for flows transmitted with the rate close to 1 Gbps or higher. It is much less than 1% for flows of lower rate. Both an overlay VxLAN-based network as well as an underlay native OpenStack network can be considered as logical network structures of deterministic delay. In future, also another tests will be conducted, e.g. tests related to computational resources utilization and energy consumption.

5. Use case 5: Content Distribution and Caching

5.1. Use case description

Web caching systems are widely applied at global scale on the Internet to improve the delivery of streaming, IPTV and many other IP services. Today, a major portion of IP traffic is transferred via distributed Content Delivery Networks (CDNs) and cloud architectures [18][4][32][39]. The main benefits are reduced load and delays as well as higher throughput, when caches serve requests to popular data on shorter transport paths to the users. The ongoing extension of datacenters in CDNs, clouds and ISP networks has strengthened the availability of caching infrastructures within IP networking domains. Caches are also present in local networks, in HGWs and browsers on the users' end devices.

In this use case we study the efficient support of content and service delivery via caching servers from the perspective of network providers as well as in the overall view including options and benefits for content/CDN providers and the users. In a main work stream, we evaluate the performance of existing and new web caching strategies in terms of the achievable hit rate for large user populations. We focus on a class of Score-Gated Least Recently Used (SG-LRU) strategies, which combine the simple update effort of the basic LRU policy with the flexibility to keep the most important content in the cache according to a predefined score function.

Simulations are performed to evaluate caching efficiency assuming Zipf request patterns, which have been confirmed manifold in the access to popular web platforms for various services including video streaming [3][20][41]. The simulations are based on a new inversion method for generating random Zipf variates [19]. Second order statistics are used to estimate the precision of the hit rate as the main performance measure.

We analyze the potential hit rate gain of new web caching strategies for the Independent Request Model (IRM) as a standard reference scenario. Moreover, we consider dynamic request patterns over time which are visible in the request statistics of popular web sites. Therefore, we investigated the daily top-1000 page requests that have been made available for Wikipedia since 2015. Our extended simulations show the impact of varying popularity of objects on the cache hit rates and to which extent the IRM is valid as a realistic assumption for large user populations.

5.2. Caching Scenario Classification

We address several preconditions in order to clarify our main focus within the broad spectrum of caching applications.

5.2.1. Web caching vs. caches in computing and database systems

Caching for the support of IP services has basic communalities with caching in computer and database systems, i.e. to get faster access to a part of the data that fits into a cache of limited size, but the workloads and the effects on costs and benefits are entirely different for both cases. In computing and database systems, periodic sequences of requests are relevant, whereas random and Zipf distributed request sequences are usual for web access. Frequent LRU data uploads from an original or higher layer web server to a cache are much more expensive than for data in a local system. Therefore, results from this study on web caching strategies cannot be transferred to caches in computing systems and vice versa, although some technical discussions seem to mix up the different scenarios [31].

5.2.2. Cachable web content and HTTP caching guidelines

Caching is known to be inapplicable for a part of the web content, including highly dynamic content or one-timers, i.e. web pages that are only requested once over a long period. Moreover, a content owner or provider can mark objects as non-cacheable in network or in end system caches. Nonetheless, the major portion of IP traffic for video streaming and IPTV services is distributed from caches in CDNs and cloud architectures. A recent update of caching options for HTTP by the IETF provides guidelines on how to avoid inappropriate data in caches [13]. In our evaluations we always refer to cacheable data.

5.2.3. Cache updates per request versus daily updates

Web caching strategies basically assume updates to be done per request, but they can partly be deferred to low traffic periods to reduce peak hour traffic. Cache providers often combine updates per requests with daily updates [4][32][39]. In this work, we focus on updates per request. The need for fast updates depends on the dynamics in the popularity of objects.

5.2.4. Caches for large versus small population

Caching systems are often organized hierarchically with central caches serving a large population and lower level caches for smaller regions. Even a cache in a browser for a single user can save 20% of download traffic for repetitive requests. The request pattern is different for each user and varies for small communities. Our focus is on a large population with access to a large web platform, where the request pattern is known to be universal and characterized by a Zipf law [3].

5.2.5. Fixed versus variable size objects

Files and other objects representing cacheable web data have different sizes up to the GB range for videos. For small caches, bin-packing problems can arise and therefore size has been considered as an important factor in studies which assume complete files as the transport unit. However, coding schemes for video streams and files in client-server and peer-to-peer systems are nowadays segmenting data into small chunks in the KB range, while storage size is steadily increasing. Therefore, we simply assume objects of fixed size corresponding to data chunks. Files of different size can still be represented as sets of fixed size objects with equal score according to the file popularity. Moreover, Hefeeda and Saleh [20] suggest assigning linear decreasing scores to fixed size data chunks of the same video because users often stop video streams after some time.

5.3. F-Secure security cloud overview

Many of F-Secure's products are based on identifying potentially dangerous data inside computer systems, and acting upon it in various ways. Examples include preventing access to malware files, or warning users attempting to access web sites known for serving malware. Traditionally such products have been based on identifying dangerous content by using periodically updating analysis engines, running locally at the user's machine. They can consume a significant amount of user's CPU and memory resources. They also make it difficult to respond rapidly to new threats, because it's not practical to update the engines very often.

The F-Secure Security Cloud is a collection of cloud-based services designed to alleviate the shortcomings of these techniques, and in some cases replace them completely. They offer clients ways to send queries with files, URLs or other objects (or hashes of such), and receive in response the object's *reputation*, which consists of various numerical scores describing features of the object. This data is continuously changing, and it's important for clients to have access to the most recent data, so that they can rapidly react to new types of threats.

A single client may handle a large volume of objects, which can cause a large volume of object reputation queries. There's motivation to reduce the volume of these queries; in addition to consuming network bandwidth, requesting data from a remote server necessarily introduces some latency, which when large enough can have a negative impact on user experience.

Our goal is to implement a caching content delivery system for distributing this reputation data. The number and geographical distribution of clients will need to be taken into account; once complete, the system will be required to handle several billions of requests per day globally. The primary requirements for the system are to be able to cost-efficiently scale to the required number of requests, and efficiently cache reputation data to reduce network traffic and user-perceived latency.

5.3.1. Object reputation queries

Querying reputation for a hash consists of the following steps:

- Normalization - if there are several equivalent binary representations of an object, they must all be able to be converted to a single canonical representation. This is necessary for URLs.
- Hashing - a cryptographically secure hash is calculated from the object data.
- Reputation query - clients pack one or more hashes into a query message, send the query, and receive a response.

Object reputation data is represented as short (usually under 1KB) JSON strings. The response JSON object maps the hash(es) queried by the client to a set of values. These can include numeric estimates of how likely a file is likely to be malicious, whether a URL is known to point to malware, etc.

Cacheability was taken into account when designing the response format. A TTL value is included in each response. In addition, effort is taken to minimize the number of distinct responses to enable caches to do effective deduplication. For instance, for the vast majority of possible hashes, which are not known at all, a single response is sufficient. A large number of known safe files can similarly be all returned in the same response.

5.4. Metrics

For content delivery via video streaming, downloads and other applications, the following metrics are used to characterize the efficiency of caching:

- the cache hit rate regarding objects and regarding data volume,

- the cache utilization in terms of the number of responses and the delivered data volume from the cache per day,
- the fraction of cacheable content for transparent caching over multiple content platforms,
- the latency of responses from caches as compared to the original web site,
- the data throughput of responses from caches as compared to the original web site,
- the traffic load reduction due to shorter transport paths from caches,
- the busy hour traffic load reduction due to prefetching and shifts of content updates into lower load phases,
- the availability of content due to responses from caches as compared to the original web site,
- the required cache storage,
- the cross traffic overhead for messages and data exchange between data center locations to control caching in CDNs and clouds.

Those metrics apply to a single cache as well as multi-level caching architectures. The caching performance can be estimated based on combinations of those metrics with different components and weights from the perspective of content, cache and network providers and the users. Quality of service parameters are in the primary focus of users and content providers. Cache and network providers can evaluate the metrics to derive operational costs including energy consumption in comparison to the benefits as a business case analysis of a caching architecture design.

The cloud reputation service servers running in Amazon AWS and OpenStack based environments are instrumented to produce logs of server resource usage (CPU, network), and response times. These metrics will be used to derive the performance characteristics of the actual implementation, as well as the value of our chosen caching strategy. Since we are working with a large scale system, we are also interested in cost optimization and try to process as many concurrent requests as possible with a single instance. For this purpose, the correlation of CPU usage to network throughput is a useful metric. Later we plan to optimize the internals of the service even more by collecting metrics with a system profiler to identify possible bottlenecks in the application logic itself.

Finally, since the whole purpose of the service is to improve customer perceived service quality, we are also doing performance test runs with our own client to collect metrics from the client perspective. We have identified the following key metrics to be collected from the client tests:

- Response latency
  - Mean, standard deviation and min/max values
  - Comparison between single and batch queries

- Errors
  - Number of hard errors from the server which prevented the client from getting the correct result
  - Overall rate compared to the number of successful requests
  - Breakdown to timeouts from the API vs. errors in the content

5.5. Experiments

5.5.1. Caching strategies

The efficiency of a cache mainly depends on the replacement strategy for identifying and storing the most relevant objects. A simple and probably the most widely used caching strategy follows the Least Recently Used (LRU) principle. LRU always puts the requested object on top of the cache, while the bottom object of a cache is replaced if cache storage is exhausted. However, LRU can't offer flexibility to prioritize objects according to cost and benefit aspects, regarding their size, the transport path, the origin or other preferences from the content providers' or users' perspective.

The extension to Score-Gated SG-LRU admits a new external object to enter a full cache only if it has a higher score than the bottom object and otherwise puts the bottom object on top, instead of the requested object. Figure 24 illustrates (SG-)LRU updates for a usual implementation as a cyclic double linked list with a top pointer. In this way, SG-LRU opens flexibility to collect objects in the cache based on an arbitrary score function, e.g. for counting past requests to each objects or for more sophisticated policies to prefer most relevant objects in the cache.

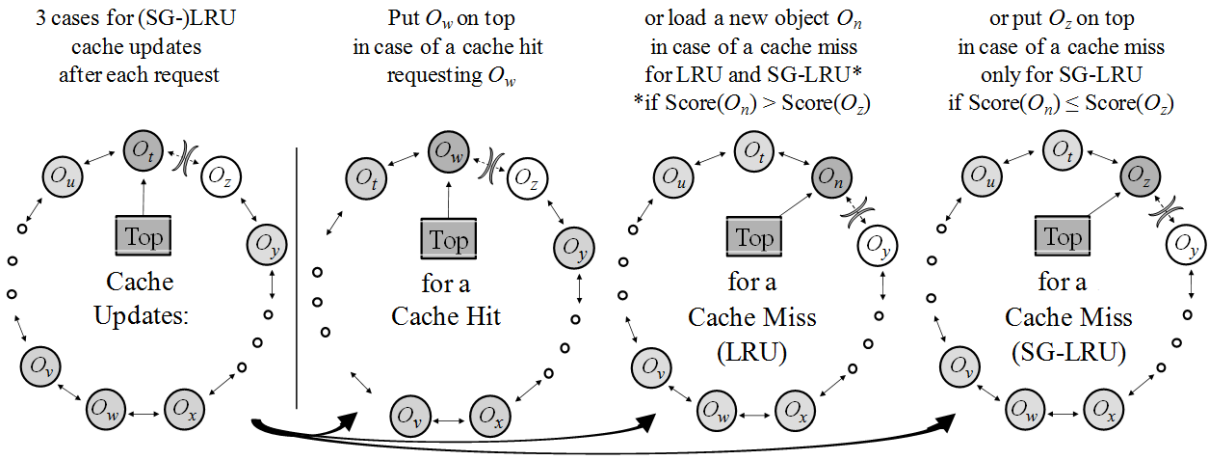

Figure 24: Updates for LRU and SG-LRU caches per request in a cyclic double linked list

If the score updates are simple, e.g., when only the score of the requested object is changing, SG-LRU requires only slightly higher update effort in the caching list than pure LRU. However, LRU is loading an object into the web cache for each request to an external object, i.e. for each cache miss, whereas SG-LRU can avoid most uploads for seldom requested, low-scored objects. Thus, SG-LRU usually even undercuts pure LRU update effort.

A simple update by incrementing the score of the requested object yields scores for counts of the number of requests per object, which corresponds to the LFU principle. The LFU is known to be optimal for the Independent Request Model (IRM), but LFU can't adapt to changing popularity. Therefore, other approaches with limited count statistics have been proposed and evaluated in literature [36][26]. A class of such strategies is attributed as LRFU spectrum [2], which includes LRU and LFU caching schemes as extreme cases. Two basic schemes covering LRU and LFU in this way are

- sliding window, restricting LFU to request counts within a window of the W most recent requests, and
- geometrical fading, introducing a decreasing factor ρ^k for the k^{th} recent request and ranking an item according to the sum of weight factors.

Both schemes behave similar for $\rho = 1 - 1/W$. The LFU principle is approached as one extreme for $W \to \infty$ and $\rho \to 1$. On the other hand, geometric fading is equivalent to LRU for $\rho \leq 0.5$, when the most recent request dominates the weights. Since geometrical fading can be implemented with lower update effort, we apply geometrical fading scores in our experiments for SG-LRU, which realize a count of past requests over a limited time frame whose extend depends on the parameter ρ. As the main new contributions of this study, we perform

- an extensive simulation study of the efficiency of SG-LRU caching strategies including advanced accuracy control estimation developed in [19],
- an exhaustive evaluation of the hit rate deficits of a pure LRU strategy compared to SG-LRU for the standard IRM model with Zipf distributed requests, and
- an extension of the study for dynamically changing popularity of objects applied to request pattern for Wikipedia pages.

5.5.2. Evaluation of web caching strategies

Next, we compare the achievable hit rates for LRU, SG-LRU and LFU in simulation studies. We start with a first evaluation example assuming independent and Zipf distributed requests. For caching evaluations of the IRM we consider three basic parameters:

- the size N of a fixed set of cacheable objects,
- the cache size $M (M < N)$, and

- the shaping parameter β of the Zipf distribution, which determines the request probabilities $z(r) = z(1)\ r^{-\beta}$ to the objects.

We simulate how the cache content is developing for a sequence of K requests, starting from an empty cache. During a filling phase of the cache until M different objects have been addressed, the caching strategies behave identical. As soon as the cache is full, pure LRU already has entered steady state regarding the object set in the cache. Thus, it is sufficient to exclude the cache filling phase as a non-representative start phase for pure LRU simulations. For LFU and SG-LRU, the convergence to a steady state depends on stabilizing score ranks of the objects, which takes much longer than the cache filling phase. LFU scores count the requests to each object. In a sequence of k successive requests, an object in rank r gets a binomially distributed number of requests in $[0, ..., k]$ with mean $kz(r)$. We exclude the first quarter of each simulation run from evaluations of the hit rate, in order to converge to stable score ranks when the run time pertains sufficiently long.

Figure 25 (a) shows SG-LRU evaluation results for an example with $N = 10^6$ objects, a cache for $M = 1000$ objects and independent Zipf distributed requests with $z(r) = 0.01337r^{-0.8}$. In this case, the LRU hit rate is $h_{LRU} \approx 10\%$, which is also confirmed by the Che approximation [36], whereas LFU achieves the maximum hit rate under IRM assumptions:

$$h_{LFU} = z(1) + ... + z(M) \approx 20.68\%$$

The score gated SG-LRU results for different ρ fall between both extremes.

For sufficiently long simulation runs with $K > 10^7$ requests, the hit rate is observed to stabilize, where SG-LRU stays close to h_{LRU} for $\rho \leq 1 - 10^{-3}$ and comes close to h_{LFU} for $\rho \leq 1 - 10^{-6}$. For shorter simulations, the SG-LRU hit rate is still increasing with K towards a saturation level depending on ρ. Since a fading factor ρ has similar effect as sliding window with window size $W = 1/(1 - \rho)$, it is obvious that the number of simulated requests is partly smaller than W. Then the scores are still evolving in a transient phase with increasing hit rates. On the other hand, the SG-LRU results in are close to their saturated maximum level as soon as $K > 10\,W = 10/(1 - \rho)$ requests are evaluated.

The results confirm that SG-LRU with scores based on previous requests limited by sliding window or geometrical fading can fully adapt to LRU as well as LFU hit rates as extreme cases based on a single parameter, ρ or W respectively. The parameter can be automatically tuned to approach the LFU hit rate up to $h_{LFU} - \epsilon$ in the IRM case, because the SG-LRU hit rate is monotonously increasing with ρ and W.

5.5.3. Hit rate estimators and variance of the simulation results

For simulations based on the IRM we can estimate the hit rate in two ways [19]:

- by counting the hits or

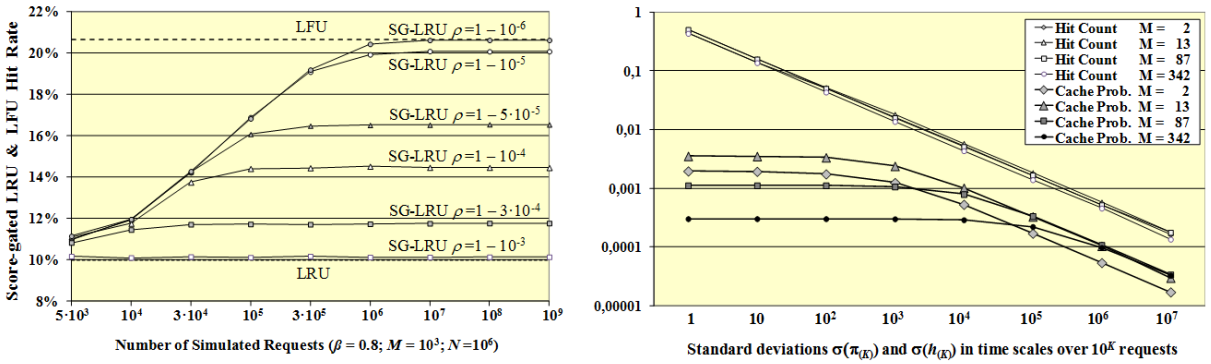

Figure 25: (a) SG-LRU hit rate in simulations of different run times (b) 2nd order statistics

- by the sum of the request probabilities of all objects in the cache over the considered request sequence.

We denote the 2nd order statistics, i.e. the standard deviations of both estimators for simulation runs over K requests by (i) $\sigma(h_{(K)})$ and (ii) $\sigma(\pi_{(K)})$, respectively. Figure 25 (b) shows a 2nd order analysis of both hit rate estimators during a simulation for SG-LRU caching with geometrical fading scores for independent Zipf requests ($\beta = 0.8; N = 1000$ objects; $\rho = 0.9999$). Four different cache sizes ($M = 2$, 13, 87 and 342) are considered, which achieve about 10%, 25%, 50% and 75% SG-LRU hit rate. The four curves for the standard deviation of the hit count are almost linear $\sigma(h_{(K)}) = \sigma(h_{(1)})/\sqrt{K}$. The four standard deviation curves for the sum of cache probabilites $\sigma(\pi_{(K)})$ as hit rate estimator start at a low level $\sigma(\pi_{(1)}) < 0.005$ already for snapshots of a single request, are remaining almost constant over several time scales and finally are also decreasing. On all time scales we observe that $\sigma(\pi_{(K)}) < \sigma(h_{(K)})$ and therefore prefer the sum of probabilities of cached objects as hit rate estimator.

5.5.4. Cloud reputation queries

Several of F-Secure's existing products employ the Security Cloud. For experiments, it was necessary to create a compatible service capable of collecting anonymous statistics of queried object hashes. The new service implementation would also serve as a testbed for any improvements that could be made based on the results. The server acted as a proxying intermediary between our backend systems and the clients, falling back to fetching fresh data from the backend if a non-expired cache entry for the queried hash was not available. The size of the cache was not limited in the experiments, and entries were invalidated based on TTL only. The TTL values were issued by the backend, varying from some minutes for objects for which detailed analysis results are pending, to several days for objects known to be safe.

Due to the expected large volume of requests, it was not practical to create a system that would immediately be able to serve all clients. Instead, we deployed a small number of servers, and

directed to them either requests originating from internal test automation, or requests from a randomly selected subgroup of end users. Upstreaming of logs of the queried object hashes was implemented, and the data was initially stored to an Elasticsearch database for analysis.

The software was initially deployed in two environments: Amazon AWS, and an internal OpenStack based platform. For both, log upstreaming was enabled to collect data of queried objects, cache hit rates, and processing latencies. The OpenStack deployment received queries from internal test automation, and was primarily for testing the stability and performance of the software under load. The role of the AWS deployment was to gather statistics of queries from real clients, in order to get information on how the queries are distributed, and to be able to research caching strategies.

The AWS instance type used was m2.small, which is designed for use cases where application will not generate constant high CPU load. For the rate of requests received during tests, it showed to be sufficient, with CPU load staying near idle levels. For redundancy, two servers were deployed between a load balancer, which also enabled updating the deployed software without interruption in service.

5.6. Results

5.6.1. Exhaustive LFU/LRU Hit Rate Evaluations for Zipf Distributed Independent Requests

In several measurement driven case studies, alternative strategies are compared to LRU, but it remains open how the evaluated performance benefits depend on the system parameters. A number of studies have evaluated LRU web caching efficiency in terms of the hit rate [31][5][42]. They partly recommend LRU, whereas on the other hand, alternatives are shown to achieve higher hit rates [20][2][26][34]. We extend and complement the experience based on measurement by a generalized simulative evaluation of the gain of LFU in comparison to LRU for the IRM with Zipf distributed requests. Therefore we extend our web caching simulations over the entire relevant range of the three characteristic parameters: the number of objects N, the cache size $M (M < N)$, and the shaping parameter β of the Zipf distribution.

Figure 26 shows results from extended evaluations for different size $N = 10^4$ and $N = 10^6$ of the set of objects. The achievable hit rate is compared for LRU and LFU for varying cache sizes M and for $\beta = 0.5, 0.6, ..., 1$, actually with 0.9999 instead of 1. Beyond simulations, the LFU optimum hit rate under IRM equals the sum of request probabilities $h_{LFU} = z(1) + ... + z(M)$. In case of LRU, the Che approximation can be applied and is confirmed to provide accurate results. We also checked that SG-LRU closely approaches the LRU and LFU results as extreme cases.

Regarding the potential advantage of alternatives over LRU for independent Zipf distributed requests, we observe a 10% - 20% absolute hit rate gain of LFU over LRU for the entire relevant range, i.e. whenever LRU achieves a hit rate in the range 10% - 50%. The relative gain is largest especially for small caches. When LRU hit rates are below 10%, then the LFU optimum is at least twice as high. Moreover, the caching efficiency can be expressed in terms of the LFU versus LRU cache size required to obtain a demanded hit rate. In order to achieve e.g. 20% hit rate, LRU requires 3-fold to 8-fold larger cache size than LFU.

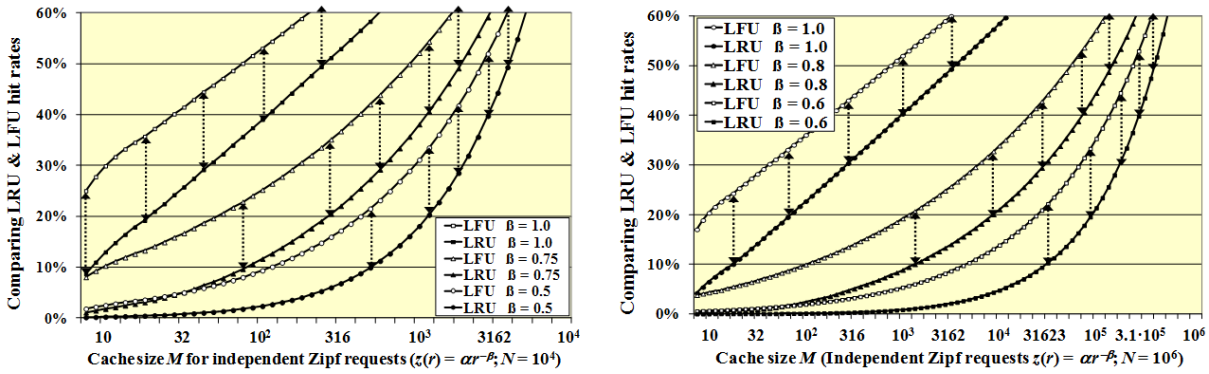

Figure 26: Comparing LFU/LRU hit rates for IRM Zipf requests

Finally, we evaluate the potential for improvement of LRU for grid points combining the parameters over the relevant range

- $\beta = 0.4, 0.5, ..., 1.1$ for Zipf distributed requests due to IRM,
- $N = 10^2, 10^3, ..., 10^7$ for the number of objects, and
- $M = M_{1\%}, M_{2\%}, ..., M_{99\%}$ for the cache size,

where $M_{x\%}$ is the minimum cache size required to obtain an LRU hit rate of $x\%$ depending on β and N. On the whole, $8 \cdot 6 \cdot 99 = 4752$ simulations have been performed to cover this range. Each simulation runs at least 10^8 requests in the evaluation phase in order to reduce the standard deviation of the hit rate results below $5 \cdot 10^{-5}$. The results of the detailed simulation study are summarized in Figure 27. It is already visible from that the difference between LRU and LFU hit rates is primarily depending on the achieved LRU hit rate. For each combination of β and N we first evaluate the minimum cache sizes $M = M_{1\%}, M_{2\%}, ..., M_{99\%}$ to obtain LRU hit rates of 1%, ..., 99% and then compute the LFU hit rates for the same cache size. shows the minimum, mean and maximum absolute LFU gain for all 8·6 cases with the same LRU hit rate of $x\%$.

In fact, we can confirm at least 9.8% potential gain whenever the LRU hit rate is in the range of 10% - 50%. The mean LFU gain in this range is 13.7% and the maximum goes up to 15% - 20%. The maximum is most often reached for the case $\beta = 1.1, N = 100$. Then, the LRU and LFU cache sizes required for $x\%$ hit rate are often unchanged for several values of $x\%$. The minimum gain is due to different cases, most of which are in the largest set of items $N = 10^7$. Again, the

relative differences in LFU versus LRU caching efficiency are largest for small caches. When the LRU hit rate is, e.g., 5% then an LFU cache of equal size achieves at least 12.8% hit rate and 15% in the mean over all 8·6 cases of $N \times \beta$ combinations.

The additional storage required to compensate for 10% - 20% LRU hit rate deficit ranges from about twice the capacity for $N = 1000$ objects up to 10-fold and more storage required in examples with $N = 10^7$ for large content platforms. Measurement based studies show similar curves for moderate LRU hit rates up to 25%, which indicate even 10 - 100-fold higher storage requirement to obtain 10% - 20% more LRU hit rate.

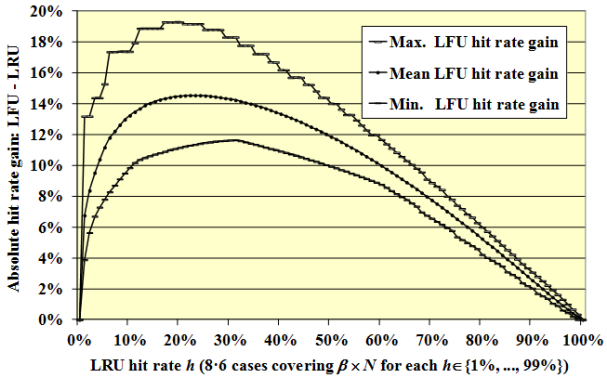

Figure 27: LFU gain over LRU for IRM Zipf requests: Result summary of 4752 cases as a grid on the relevant $\beta \times N \times M$ range

5.6.2. Evaluation for Dynamic Popularity based on Daily Wikipedia Page Request Statistics

Content Popularity Dynamics over Time

In contrast to the IRM assumption, many measurement studies indicate how the popularity of Internet content varies over time for user-generated content, e.g., videos, especially for file sharing systems, and for IPTV applications [41][5].

In principle, unpredicted changes in content popularity make caching less efficient, if such dynamics is high enough to render content useless only a few requests after being loaded into the cache. The effect of popularity dynamics on cache hits mainly depends on the user population served by the cache and its request frequency, which can count into millions per day.

In fact, we experience higher dynamics in the day by day Wikipedia request pattern than the previously referred studies on video, P2P and IPTV platforms. Nevertheless, Zipf distributed requests and the conclusions on caching efficiency of the previous sections are confirmed to be still relevant.

Daily Wikipedia top-1000 statistics and Zipf request pattern

For realistic access pattern of a popular web platform, we refer to statistics being published by Wikipedia [43]. Even if the data volumes are smaller than for popular video streaming platforms, we expect similar dynamic Wikipedia page request pattern. We evaluate daily statistics of the number of requests to the top-1000 pages, which are provided since August 2015.

In a first step, we adapt Zipf distributions to daily top-1000 request distributions. We measure the deviation of the Wikipedia top-1000 requests for the cumulative distribution function $W_{d,CDF}(k) = R_d(k)/R_d(1000)$ and a Zipf adaptation $Z_{CDF}(k)$, where $R_d(k)$ is the sum of requests to the top-k pages ($k = 1, ..., 1000$) at day $d = 1, ..., 184$ for the half year period from Aug. 2015 - Jan. 2016. Our measure of deviation $\Delta_{W_d \leftrightarrow Z}(k)$ is defined by the difference in the ranks, for which both CDF distributions achieve the same level, i.e. we define

$$\Delta_{W_d \leftrightarrow Z}(k) = j_k - j \quad \text{where} \quad Z_{CDF}(j_k) \leq W_{d,CDF}(k) < Z_{CDF}(j_k + 1).$$

Therefore, the Zipf parameter β is determined in order to minimize the mean absolute rank deviation $\sum_k |\Delta_{W_d \leftrightarrow Z}(k)|)/1000$. We found the minimum according to the considered deviation criterion always in the range $0.5 < \beta < 0.6$ except for two days in the time frame with larger β up to 0.75.

Figure 28 (a) shows the minimum, maximum and mean absolute rank deviations of Zipf distributions adapted to each daily top-1000 request statistics for optimum β. All rank deviations are in the range [–27, 9] and mean absolute rank deviations are always less than 10. On 4 out of 184 days, the Wikipedia top-1000 requests can be perfectly matched by a Zipf distribution with rank deviations of only –1, 0 or 1 on all 1000 ranks.

The largest deviations from Zipf distributions are experienced in the top-10, whereas the tail of the top-1000 distributions can be closely fitted. This is due to a large statistical variation of requests especially for the top-1 page, ranging from $3{\cdot}10^5$ to $6{\cdot}10^6$ requests, while the total number of daily top-1000 request is less variable between $2.4{\cdot}10^7$-$3.4{\cdot}10^7$. When we consider the distribution given by the mean number of top-k requests over increasing time periods, then we experience a convergence very close to a Zipf distribution.

Daily Wikipedia top-1000 statistics and IRM cache hit rates

In a first evaluation of caching efficiency based on the Wikipedia data, we assume independent requests per day with request probabilities according to the top-1000 request frequencies and separated simulations for each day.

Figure 28 (b) shows SG-LRU results for several fading factors r and for cache sizes 25 and 200, respectively. Although the variability in the daily hit rates is high, from 4.3% - 24.9% for LRU with $M = 25$ and from 29.3% - 54.9% with $M = 200$, the potential gain for SG-LRU is almost constant each day close to the maximum LFU hit rate under IRM conditions. The mean

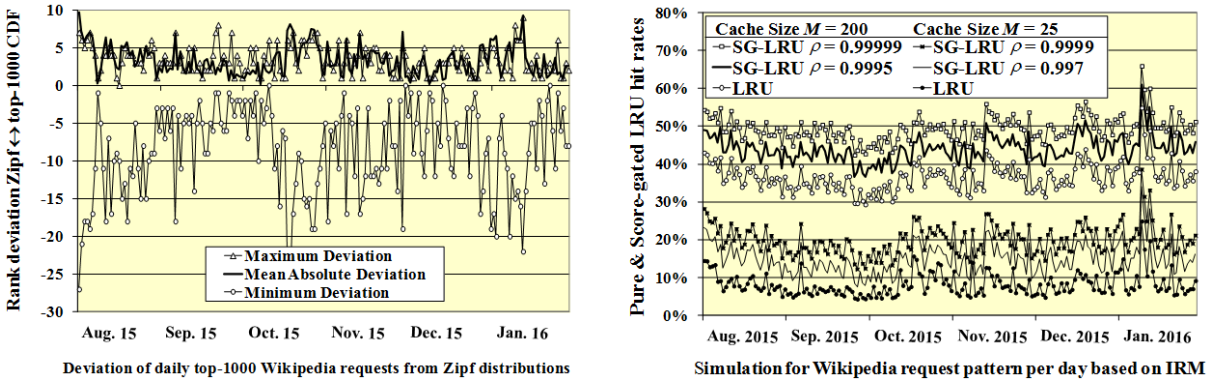

Figure 28: (a) Deviations of Zipf adaptations (b) Cache hit rate for top-1000 daily Wikipedia requests

LRU hit rates over 184 days are 8.2% for $M = 25$ and 36.5% for $M = 200$, as compared to mean SG-LRU hit rates of 20% $(M = 25, \rho = 0.9999)$ and 49.4% $(M = 200, \rho = 0.99999)$. This confirms the 10% - 20% gain similar to the results in also for a real request pattern due to Wikipedia statistics. For those evaluations, all requests are assumed to address top-1000 web pages of a day. With regard to a 10-fold higher number of requests to all Wikipedia web pages, larger cache sizes are required in order to achieve the same hit rates.

Cache simulation under daily changing request pattern

From the daily top-1000 page request statistics we can also gain insights into the dynamics of page popularity. The rate of change in the top-k pages is expressed in Table 7 by the fraction of requests addressing top-k pages of the previous day.

The evaluation is again based on 184 days from Aug. 2015 - Jan. 2016. At least more than half of the requests for $k = 25$ and even 75% for $k = 1000$ are referring to yesterday's top-k pages. On the other hand, 20% - 24% of the requests are for new pages which didn't appear among yesterday's top-1000.

Top-k pages: k =	**25**	**50**	**100**	**200**	**500**	**1000**
Y's Top-k → Top-k	54.7%	58.8%	62.4%	66.7%	71.9%	76.1%
1 – (Y's Top-1000 → Top-k)	22.6%	21.8%	21.2%	20.8%	20.9%	23.9%

Table 7.: Fraction of requests to yesterday's top-k pages

The fluctuation within the top-200 is captured day by day in Table 7, with an upper curve for the total number of requests to a steadily changing set of top-200 pages for each day. A dark gray surface represents the requests to the previous day's top-200 and a light gray surface to previous top-1000 pages.

We simulate cache evaluations for the half year time frame. On each day, we assume constant request probabilities $R_d^{(i)}/R_d$ to the pages, where $R_d^{(i)}$ is the number of requests to the page on rank i in the top-1000 statistics on day d. R_d denotes the total number of top-1000 requests on that day. For a new day, the cache starts with the content from the end of the previous day and it is left to the (SG-)LRU strategy to adapt the cache content to the current request.

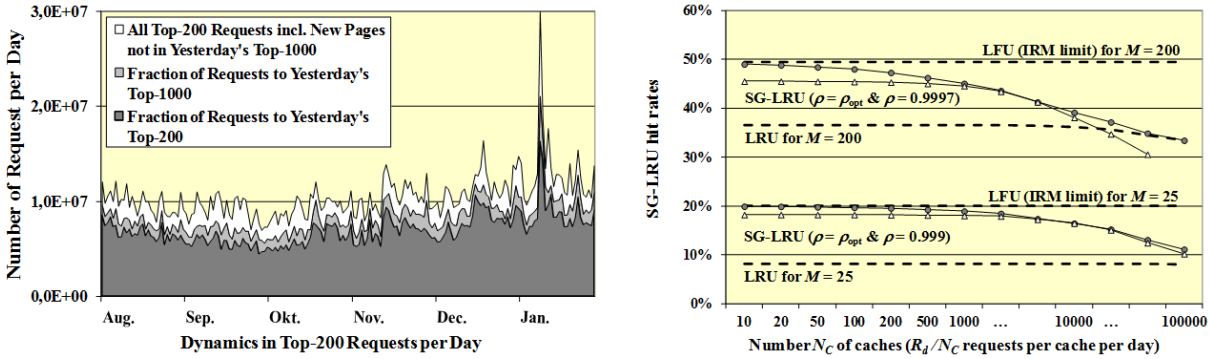

Figure 29: (a) Changes in the daily top-200 requests (b) Hit rates incl. daily dynamics

With a single cache for all requests, the phases adapting to daily changes are negligible within the total number $R_d \approx 2{\cdot}10^7$ of top-1000 requests per day, such that the IRM hit rate is still obtained. Instead, we assume a number N_C of caches, each of which is serving a fraction $1/N_C$ of the user population and thus a fraction $\lceil R_d/N_C \rceil$ of the daily Wikipedia requests. Then we simulate $\lceil R_d/N_C \rceil$ requests with constant request probabilities per day. The number of requests $\lceil R_d/N_C \rceil$ per cache per day is a characteristic factor for the caching efficiency. Figure 29 (a) shows simulation results again for requests only to the top-1000 Wikipedia pages and for cache sizes of $M = 200$ with hit rate curves in the range $> 30\%$ as well as for $M = 25$ with hit rates $< 30\%$, respectively. We still apply the SG-LRU caching strategy with a geometrical fading score function. The results shown in Figure 29 (b) fall into three categories depending on N_C:

- In the range $N_C \leq 200$ (for $M = 25$ even up to $N_C \leq 1000$), the LFU hit rate limit for independent requests (IRM) is closely approached by SG-LRU with an appropriate fading factor ρ.
- In a range $500 \leq N_C \leq 10\,000$, the SG-LRU hit rate doesn't exploit the LFU limit under IRM conditions, but still significantly improves over pure LRU.
- In a range $N_C > 10\,000$, SG-LRU does not essentially improve pure LRU hit rates for geometrical fading scores.

Note that the number N_C of caches corresponds to the number of requests per cache per day: $\lceil R_d/N_C \rceil \approx 2{\cdot}10^7/N_C$. The first category roughly corresponds to $\lceil R_d/N_C \rceil > 2{\cdot}10^7/200 = 100\,000$.

As a difference compared to the static IRM case, SG-LRU hit rates are now increasing up to an optimum ρ_{opt} and decreasing beyond ρ_{opt}. In the case $N_C = 10\,000$ and $M = 25$, we observe SG-LRU hit rates going up to 16.6% for $\rho \approx 0.999$ and then falling down to 11.4% already for $\rho = 0.9999$. With dynamically changing request pattern, we experience SG-LRU hit rates $h(\rho)$ generally as a function of ρ starting to increase from LRU for $\rho < 0.5$ towards an optimum ρ_{opt} with maximum hit rate h_{max} and then falling for $\rho > \rho_{opt}$ down to hit rates even below the LRU hit rate. This behaviour allows to automatically determine ρ_{opt} and the maximum hit rate. For further study, we are testing more sophisticated and further optimized score functions, trying to predict rising popularity of new pages.

For dynamics on the time scale of hours, the Wikipedia statistics has no data available per page. Therefore we leave an analysis in smaller time scales open for future work based on more detailed alternative traces. If we interpolate a daily change in the requests to a page from R_d to R_{d+1} over the hours as a monotonous trend, then the dynamics is smoother, leading to improved caching efficiency as compared to a hard shift from R_d to R_{d+1}. From our current experience, we agree to that dynamics in the time scale of days/weeks is more relevant.

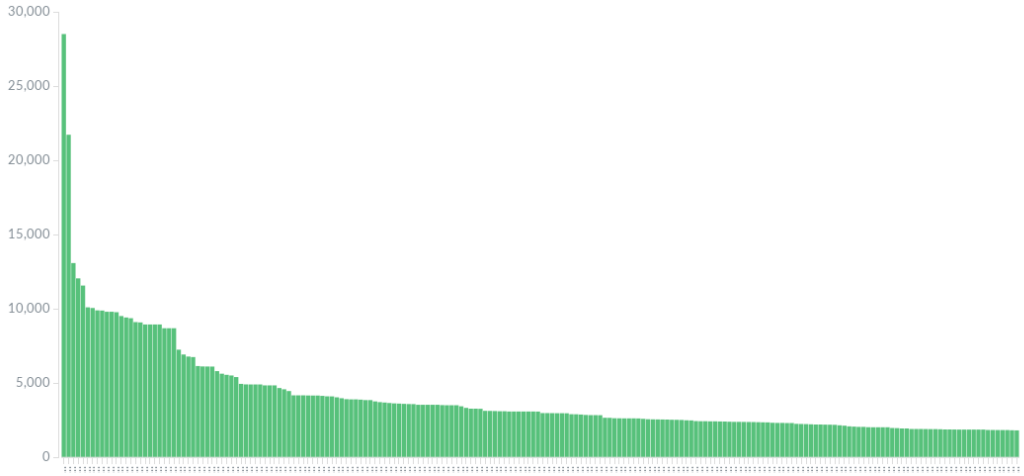

Figure 30: Distribution of most queried file hashes from a 7 day period

5.6.3. Object reputation cloud query statistics

For initial data collection from the object reputation cloud service, a randomly chosen pool of users was directed to the service deployed to AWS. The clients were mostly querying for attributes of common files present on Android-based mobile platforms, mostly .apk application files. During the seven day measuring period, the volume of queried objects was about 650000 per day.

The objective of the test was to determine how the queried keys were distributed, and what kind of a cache hit rate is achievable, if the cache size were not limited by anything else than backend-issued TTLs for individual entries.

As can be seen in 30, the distribution of keys resembled a Zipf-like distribution. Cache hit rate during the period was approximately 60 % (see 31).

Figure 31: Number of cache hits and misses on the servers during a 7 day period

Times taken by the server to generate responses were measured, and are shown in 32. There seems to be a clear divide between queries that could be served directly from cache, and queries which required a backend query.

5.7. Summary

We devoted detailed simulation studies to the efficiency of caching for usually observed Zipf distributed request pattern. The LRU caching strategy is compared to score gated SG-LRU methods, combining low LRU update effort with flexible score-based selection of the cache content.

In a first part on the standard model of independent (IRM) Zipf distributed requests, we confirm that LRU hit rates in the range 10 - 50% generally leave further 10 - 20% absolute hit rate potential unused compared to SG-LRU, as also shown in several measurement studies for more complex caching strategies [26].

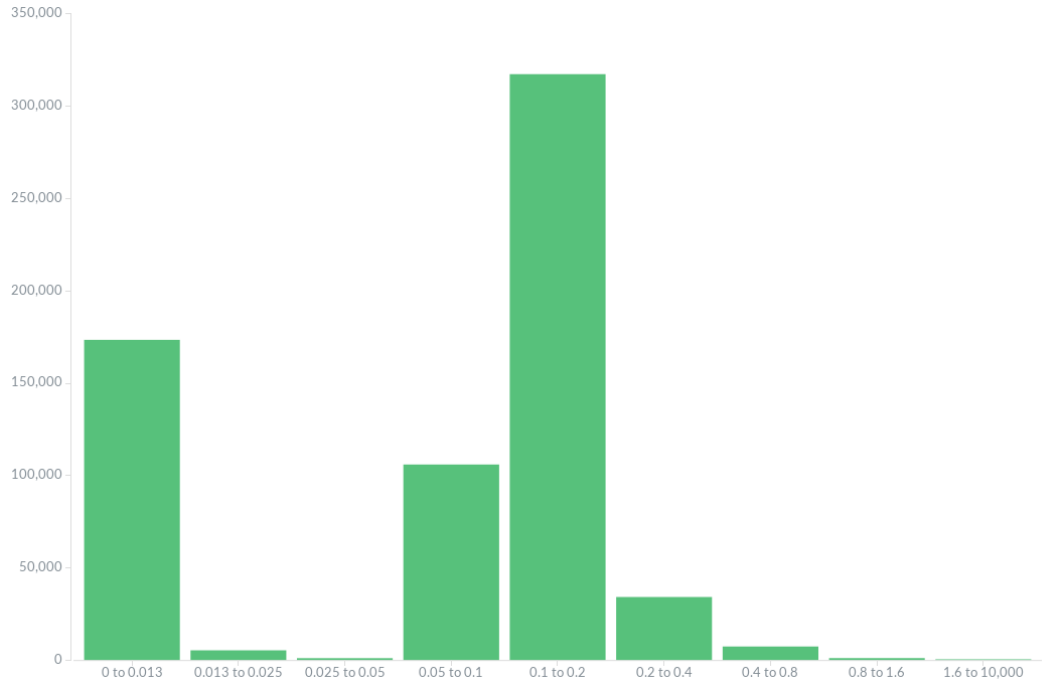

Figure 32: Distribution of response times in a 7 day period, collected at server

In a second part, we include dynamic popularity changes based on Wikipedia page request statistics, which again exhibit Zipf-like request pattern. Although $> 20\%$ new pages appear among the top-1000 pages every day, the cache hit rates for Wikipedia pages are still close to IRM conditions for the Zipf-like daily request distribution when a cache serves a large population, i.e. when a cache handles at least 100 000 requests per day. Dynamic request pattern have more impact on the performance of smaller caches, leading to less homogeneous results also depending on local environments and user preferences.

On the whole, the results indicate that caching efficiency is not restricted to the prevalently considered LRU case, but can go up to the essentially higher LFU hit rate limit for Zipf distributed requests. Including statistics about past requests by the proposed SG-LRU method provides a simple extension to exploit the hit rate potential beyond pure LRU for IRM and for more realistic dynamic content popularity scenarios. For future work, we plan to optimize SG-LRU score functions for request pattern based on an extended set of web access measurements.

F-Secure's security cloud services provide clients with information on attributes of known files and URLs. By implementing an intermediate caching server, we were able to study the queries originating from a randomly selected subset of clients, and test the effect of caching responses at the intermediate proxy server. In future, the collected data can be used to study caching strategies in context of a real-world content distribution application. We plan to leverage the simulation methods and results presented in section 5.6 in order to pick the best available strategy for the request distribution observed in our application.

6. Conclusions

This reports provides and discusses the evaluation of solutions developed by the partners in the current period of project duration time. The methodologies of evaluation of particular solutions are presented as well. The results presented in this report are related to other parts of the project such as stack modelling, determination policy language or optimization cloud interconnection solutions. All of them, together contribute to the development of inter-cloud orchestration as the final result of the SSICLOPS project. Specific conclusions related to particular use cases are summarized as follows.

Hyrise-R is a scale-out extension of the in-memory database Hyrise where users send queries, as Hyper-Text Transfer Protocol (HTTP) JavaScript Object Notation (JSON) query plans or stored procedures, to a query dispatcher, which propagates requests to appropriate database instances. Data altering requests are processed by the master instance. The master sends logs, describing the data changes, via Transmission Control Protocol (TCP) to the replica nodes in an eager or lazy manner. Adding replicas can scale read-only and mixed workloads linearly. In general, the amount of exchanged data compared to the processing within the database is low. For eager replication, the master node has to wait that replicas acknowledge the log reception. In this case, network latency can affect the transaction latency. Bandwidth is important when returning large result sets, or performing large data updates and sending these to replicas.

The network performance affects the computing performance and energy efficiency of High Energy Physics (HEP) computing in an OpenStack cloud testbed. Our results indicate that the network latency, either caused by a simulator or physical distances between the sites, has a negative impact on the computing performance. High latency both increases processing times and the total energy consumption. Moreover, the effect of latency on the execution time and energy consumption of a computation job, increases when bandwidth is small. Parallelism, e.g. multiple cloud instances sharing the limited network resource, also adds more latency and increases processing times. The results reflect the current software environment used for HEP computing. We conclude that new data transfer protocols and advanced caching mechanisms could improve performance. Planned activities will cover the inclusion of mechanisms which could also provide additional profiles of workload for this use case.

The evaluation of caching methods reveals substantial performance deficits of usual caching strategies following the Least Recently Used (LRU) principle. For the standard model of independent Zipf distributed requests (IRM), we confirm that LRU hit rates in the range 10 - 50% generally leave further 10 - 20% absolute hit rate potential unused compared to score

based strategies, which include statistics about past requests. While similar experience has been made also in other studies for strategies involving more complex updates, we recommend a score-gated LRU variant which often even undercuts low LRU update effort. The clients of F-Secure's cloud services query attributes of files and URLs from a large set of known objects, but not all objects are queried equally often. By implementing a caching intermediate server and collecting statistics from a subset of clients, it was found out that most of the time a responses could be served from the cache. The volume of traffic during the measuring period did not cause excess load for the chosen test platform. For identifying bottlenecks in the intermediate server implementation, it will be necessary to increase the volume of requests. Research into more advanced caching strategies could enable us to limit the number of response to cache without sacrificing too much of the cache hit rate. Planned work is to study caching strategies in the context of a real-world content distribution application based on collected data.

Based on the results obtained so far, it can be observed that Network Function Virtualization Infrastructure (NFVI) for a virtual Home Gateway (vHGW) offers high bandwidth capabilities (gigabits per second) for transmissions between the Virtual Machines (VMs) involved. Measured round-trip time (RTT) values meet the requirements of typical LAN RTT (RTT less than 1 ms). Also the jitter level is very low (no more than 0.05 ms). Packet loss starts to exceed 1% for flows transmitted with a rate close to 1 Gbps or higher. Packet loss is well below 1% for flows of lower rate. Both an overlay Virtual eXtensible Local Area Network (VxLAN)-based network as well as an underlay native OpenStack network can be considered as logical network structures incurring deterministic delay. It must be stressed, however, that the tested scenario was centralized in a single datacenter and assumed homogeneous utilization of the datacenter, i.e., only the vHGW service running without another background traffic present. Future work will cover tests of service performance assuming a more diversified load of the datacenter. Tests related to computational resources utilization and energy consumption will also be carried out.

The future work will be continued to develop inter-cloud orchestration solutions which utilise and span individual result of particular use cases. This allows to reach open-source software-defined cloud infrastructure. Specific conclusions presented above allow to determine a generic one for the final results of the SSICLOPS project.

Bibliography

[1] E. G. N. 001. "Network Functions Virtualisation (NFV), Use Cases". (Oct. 2013).

[2] D. L. et al. "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies". *IEEE Trans. Computers* 50.12 (2001), pp. 1352–1361.

[3] L. B. et al. "Web Caching and Zipf-like Distributions: Evidence and Implications". *IN INFOCOM*. 1999, pp. 126–134.

[4] N. K. et al. "Optimally Designing ISP-Operated CDN". *IEICE Transactions* 96-B.3 (2013), pp. 790–801.

[5] S. T. et al. "Temporal locality in today's content caching: why it matters and how to model it". *Computer Communication Review* 43.5 (2013), pp. 5–12.

[6] P. Andrade, T. Bell, J. van Eldik, G. McCance, B. Panzer-Steindel, M. C. dos Santos, S. Traylen, and U. Schwickerath. "Review of CERN Data Centre Infrastructure". *Journal of Physics: Conference Series* 396.4 (2012).

[7] I. Antcheva, M. Ballintijn, B. Bellenot, and M. Biskup. "ROOT?A C++ framework for petabyte data storage, statistical analysis and visualization". *Computer Physics Communications* 180.12 (Dec. 2009), 2499?2512.

[8] L. A. T. Bauerdick, K. Bloom, B. Bockelman, D. C. Bradley, S. Dasu, J. M. Dost, I. Sfiligoi, A. Tadel, M. Tadel, F. Wuerthwein, A. Yagil, and the Cms collaboration. "XRootd, disk-based, caching proxy for optimization of data access, data placement and data replication". *Journal of Physics: Conference Series* 513.4 (2014). URL: http://stacks.iop.org/1742-6596/513/i=4/a=042044.

[9] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, I. Fisk, M. Girone, J. Harvey, B. Kersevan, P. Mato, R. Mount, and B. Panzer-Steindel. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN, 2014.

[10] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky. "XROOTD-A Highly scalable architecture for data access". *WSEAS Transactions on Computers* 1.4.3 (2005).

[11] F. Fabozzi, C. Jones, B. Hegner, and L. Lista. "Physics Analysis Tools for the CMS Experiment at LHC". *Nuclear Science, IEEE Transactions on* 55 (2008), pp. 3539–3543.

[12] B. Forum. "TR-317, Network Enhanced Residential Gateway". (July 2016).

[13] C. Fricker, P. Robert, and J. Roberts. "A versatile and accurate approximation for LRU cache performance". *IEEE Proc. 24th International Teletraffic Congress* (2012).

[14] R. Gardner, S. Campana, G. Duckeck, J. Elmsheuser, A. Hanushevsky, F. G. Hönig, J. Iven, F. Legger, I. Vukotic, W. Yang, and the Atlas Collaboration. "Data federation strategies for ATLAS using XRootD". *Journal of Physics: Conference Series* 513.4 (2014), p. 042049. URL: http://stacks.iop.org/1742-6596/513/i=4/a=042049.

[15] T. Z. Gerd Behrmann Dmitry Ozerov. "Xrootd in dCache - design and experiences". *International Conference on Computing in High Energy and Nuclear Physics (CHEP)*. 2010.

[16] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden. "HYRISE - A Main Memory Hybrid Storage Engine". *PVLDB* 4.2 (2010), pp. 105–116. URL: http://www.vldb.org/pvldb/vol4/p105-grund.pdf.

[17] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. "Measuring energy consumption for short code paths using RAPL". *ACM SIGMETRICS Performance Evaluation Review* 40.3 (Dec. 2012), pp. 13–17. DOI: 10.1145/2425248.2425252.

[18] G. Hasslinger and F. Hartleb. "Content delivery and caching from a network provider's perspective". *Computer Networks* 55.18 (2011), pp. 3991–4006.

[19] G. Hasslinger, K. Ntougias, and F. Hasslinger. "Performance and Precision of Web Caching Simulations Including a Random Generator for Zipf Request Pattern". *Measurement, Modelling and Evaluation of Dependable Computer and Communication Systems - 18th International GI/ITG Conference, 2016, Münster, Germany, April 4-6, 2016, Proceedings*. 2016, pp. 60–76.

[20] M. Hefeeda and O. Saleh. "Traffic modeling and proportional partial caching for peer-to-peer systems". *IEEE/ACM Trans. Netw.* 16.6 (2008), pp. 1447–1460.

[21] *Iperf2 documentation*. URL: https://iperf.fr/iperf-doc.php#doc.

[22] *Iperf3 documentation*. URL: https://iperf.fr/iperf-doc.php#3doc.

[23] J. Krüger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, and A. Zeier. "Fast Updates on Read-Optimized Databases Using Multi-Core CPUs". *PVLDB* 5.1 (2011), pp. 61–72. URL: http://www.vldb.org/pvldb/vol5/p061_jenskrueger_vldb2012.pdf.

[24] S. Manegold, P. A. Boncz, and M. L. Kersten. "Generic Database Cost Models for Hierarchical Memory Systems". *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*. 2002, pp. 191–202. URL: http://www.vldb.org/conf/2002/S06P03.pdf.

[25] H. Matsunaga, T. Isobe, T. Mashimo, H. Sakamoto, and I. Ueda. "Managed Grids and Cloud Systems in the Asia-Pacific Research Community". Springer US, 2010. Chap. Performance of a disk storage system at a Tier-2 site, pp. 85–97. DOI: 10.1007/978-1-4419-6469-4_5.

[26] N. Megiddo and D. Modha. "Outperforming LRU with an Adaptive Replacement Cache Algorithm". *IEEE Computer* 37.4 (2004), pp. 58–65.

[27] R. Meusel, J. Blomer, P. Buncic, G. Ganis, and S. S. Heikkilä. "Recent Developments in the CernVM-File System Server Backend". *Journal of Physics: Conference Series* 608.1 (2015), p. 012031.

[28] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. "ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics". *Datenbanksysteme für Business, Technologie und Web (BTW), 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings*. 2013, pp. 499–502. URL: http://www.btw-2013.de/proceedings/ScyPer%20A%20Hybrid%20OLTPOLAP%20Distributed%20Main%20Memory%20Database%20System%20for%20Scalable%20RealTime%20Analytics.pdf.

[29] C. O'Luanaigh. *OpenStack boosts Tier 0 for LHC Run 2*. Tech. rep. CERN, 2014.

[30] *Open virtual Switch documentation*. 2016. URL: http://openvswitch.org/support/.

[31] P. Panchekha. *Caching in theory and practice*. 2012. URL: tech.dropbox.com/2012/10/caching-in-theory-and-practice.

[32] PeerApp. URL: www.peerapp.com.

[33] H. Plattner. "The Impact of Columnar In-Memory Databases on Enterprise Systems". *PVLDB* 7.13 (2014), pp. 1722–1729. URL: http://www.vldb.org/pvldb/vol7/p1722-plattner.pdf.

[34] S. Podlipnig and L. Böszörményi. "A survey of Web cache replacement strategies". *ACM Comput. Surv.* 35.4 (2003), pp. 374–398.

[35] S. Ponce and R. D. Hersch. "Parallelization and Scheduling of Data Intensive Particle Physics Analysis Jobs on Clusters of PCs". *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. 2004. DOI: 10.1109/IPDPS.2004.1303280.

[36] M. N. R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Caching*. RFC 7234. 2015. URL: https://rfc-editor.org/rfc/rfc7234.txt.

[37] D. Schwalb, M. Faust, J. Wust, M. Grund, and H. Plattner. "Efficient Transaction Processing for Hyrise in Mixed Workload Environments". *Proc. 2nd International Workshop on In Memory Data Management and Analytics, IMDM 2014, Hangzhou, China, September 1, 2014*. 2014, pp. 16–29. URL: http://www-db.in.tum.de/hosted/imdm2014/papers/schwalb.pdf.

[38] D. Schwalb, J. Kossmann, M. Faust, S. Klauck, M. Uflacker, and H. Plattner. "Hyrise-R: Scale-out and Hot-Standby through Lazy Master Replication for Enterprise Applications". *Proc. 3rd VLDB Workshop on In-Memory Data Mangement and Analytics, IMDM@VLDB 2015, Kohala Coast, HI, USA, August 31, 2015*. 2015, 7:1–7:7. DOI: 10.1145/2803140.2803147.

[39] A. Sharma, A. Venkataramani, and R. Sitaraman. "Distributing content simplifies ISP traffic engineering". *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA*. 2013, pp. 229–242.

[40] SSICLOPS Project. *SSICLOPS Project Web Site*. Feb. 2015. URL: https://ssiclops.eu/.

[41] D. Stutzbach, S. Zhao, and R. Rejaie. "Characterizing files in the modern Gnutella network". *Multimedia Syst.* 13.1 (2007), pp. 35–50.

[42] D. Wessels. *Temporal locality in today's content caching: why it matters and how to model it*. Wesley, 2004.

[43] Wikipedia. URL: wikitech.wikimedia.org/wiki/Pageviews%5C_API.

[44] S. de Witt and A. Lahiff. "Quantifying XRootD Scalability and Overheads". *Journal of Physics: Conference Series* 513.3 (2014). URL: http://stacks.iop.org/1742-6596/513/i=3/a=032025.

[45] J. Wust, M. Grund, and H. Plattner. "TAMEX: a Task-Based Query Execution Framework for Mixed Enterprise Workloads on In-Memory Databases". *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt, 16.-20. September 2013, Koblenz, Deutschland*. 2013, pp. 487–501.