INCA: An Architecture for In-Network Computing

Abdulazaz Ali Albalawi, Asit Chakraborti, Cedric Westphal, Dirk Kutscher, Jeffrey He, Quinton Hoole*

ABSTRACT

We present some results on integrating computing with networking so as to optimize the placement of workloads within a distributed network. We describe INCA, an In-Network Computing Architecture that allows clients to request functions that are then instantiated at a place within the network that attempts to meet both the QoE constraints of the application and the incentives of the operator of the network. We have implemented INCA, including network monitoring capability as well as a function placement optimization capability. In our evaluation, INCA demonstrates the benefit of a joint optimization of the networking and computing aspects.

CCS CONCEPTS

• **Networks** → **Layering; Middle boxes / network appliances; Cloud computing; In-network processing;**

KEYWORDS

In-Network Computing, Service Virtualization

ACM Reference Format:

Abdulazaz Ali Albalawi, Asit Chakraborti, Cedric Westphal, Dirk Kutscher, Jeffrey He, Quinton Hoole. 2019. INCA: An Architecture for In-Network Computing. In *1st ACM CoNEXT Workshop on Emerging in-Network Computing Paradigms (ENCP '19)*, December 9, 2019, Orlando, FL, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3359993.3366649>

1 INTRODUCTION

With the advent of more light-weight platforms (light-weight VMs, unikernels, mobile code) it has become possible to decompose larger application into smaller units, instantiate

*Email: aalbalaw@ucsc.edu, asit.k.c@gmail.com, cedric.westphal@futurewei.com, dku@dkutscher.net, jeffrey.he@huawei.com, quinton.hoole@futurewei.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ENCP '19, December 9, 2019, Orlando, FL, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-7000-4/19/12...\$15.00

<https://doi.org/10.1145/3359993.3366649>

them in milliseconds timeframes. This has led to rethink architectures and best practices, for example introducing serverless services and stateless functions.

However, existing networking abstractions, protocols and corresponding infrastructure services are inadequate for connecting light-weight compute functions dynamically, responding to events in the physical word and to changing network conditions and user behavior. Chaining light-weight, potentially short-lived instances of compute functions by relying on DNS-redirection, connection-based communication abstractions and security mechanisms could make both the development as well as the actual operation of such applications to complicated, inefficient and insecure.

We propose INCA (for In-Network Compute Architecture), a framework that answers the need for placing the computation within the network in real time at the appropriate location. This framework monitors the network performance in between INCA nodes and optimizes in real time the placement of functions for meeting a given performance objective (for example, minimization of the response time for a specific function).

INCA is leveraging Sapphire – a genuinely distributed computing framework [10] – and is applying Information-Centric Networking concepts (name-based forwarding, named functions, caching, forwarding strategies) to provide better visibility into edge resource availability and to accelerate resource allocation and function execution.

The key contribution of this paper is a framework to support the joint optimization of networking and computation within the network, allowing the placement of workloads in a manner that satisfies both the application requirements and the optimization of the network resources. We have implemented INCA, integrated in the Sapphire system and evaluated its performance on a range of evaluation scenarios to demonstrate its feasibility.

The paper is organized as follows: first, Section 2 contrasts the related work. We then consider the requirements for such a system in Section 3. We describe the design of INCA in Section 4. We implemented INCA and detail this implementation in Section 5. Section 6 evaluates the performance of INCA in a range of scenarios. Finally, Section 7 concludes the paper.

2 RELATED WORK

Programmability in the network [9] is being supported by languages such as P4 [1]. The P4 framework provides programmability to switches in the network, but is restricted into the computation that can be performed via a match-action switch architecture.

Serverless computing instantiates functions on demand, albeit stateless functions primarily. [8] gives a comprehensive overview of serverless computing, outlining its definition, working mechanisms, advantages, drawbacks and future directions. INCA supports stateful functions as well, in addition to serverless applications.

[3, 7] proposed architectures that supports flow-based any-cast to services, and allows for these services to be mobile. Service Oriented Networking [4] provides the mechanisms required to deploy a replicated service instance in the network and to route client requests to the closest instance in an efficient manner. INCA as well uses the service name to appropriately identify the executing node.

[2] introduced Named-Function Networking (NFN), an extension of Named-Data Networking that supports invocation of function by name. [5] uses unikernels to deploy Named Functions as a Service. These proposals require to change the network underneath, unlike INCA.

NFV (see for instance [6]) is an example of a set of functions (namely, Network Functions) that are being virtualized, distributed and scaled up and down within the network based upon some optimization formulation. INCA generalizes this to any service, not just network functions.

While INCA is related to all these previous works, none provides a comprehensive framework to support the joint optimization of networking and computing.

3 ARCHITECTURAL REQUIREMENTS

INCA defines a set of general principles that can be applied to different environments, including edge analytics, big data processing, in-network edge computing, context-aware computing. Namely, INCA is a network architecture that puts computing first in the network. Distributed computing is its first order principle.

Currently, the Internet follows a connection-oriented model. It uses host-based addressing to set up transport and security pipes to connect the different nodes, both at the network layer and at the overlay layer. The network cannot really support applications, outside of CDN-like overlays that are heavy-weight and application-specific. Service chaining is possible only in controlled environments, and without transport layer security. The current Internet is dominated static servers and mobile clients connected via overlays.

On the other hand, INCA is a distributed system where the computation resource at different locations can be leveraged

by applications on-demand. These applications can range from light-weight, stateless functions to long-lived, stateful services. In INCA, the app store is replaced by a λ -store, a repository of services that can be quickly deployed on-demand. The functions can be instantiated on the end user's device, the home gateway, the base station, the Central Office) or the cloud data center.

These services are deployed according to specific objectives that include economic considerations (such as consolidation of the functions on a few platforms); performance (ideal function location, load balancing); and security (control over function location, access to the functions and data distribution).

INCA should enable a client to request a specific service or application from the network; the network then forwards the service request to a node that can respond to the request, or alternatively the network instantiate the function at a node that satisfies the application requirements; the user can then obtain the service from this instance. The instantiation of the service is optimized with respect to the user's context, the network conditions and the available resources.

The key issues are to provide a mechanism that is flexible, that can be *incrementally deployed*, and that allows for a wide range of applications to be supported. In addition, we would like INCA to be *expressive, to instantiate a wide range of services, to be secure, to optimize the allocation of resource, to scale to a large number of users, to support mobility (of the function and of the user), and to support heterogeneous devices*. Meeting these requirements is a design objective of INCA.

For INCA, we selected Sapphire [10] as the basis for the architecture. Sapphire is a distributed programming platform that makes it easy to program mobile and cloud applications. Sapphire's key design feature is its distributed runtime system, which supports a flexible and extensible deployment layer for solving complex distributed systems tasks, such as fault-tolerance, code-offloading, and caching.

Sapphire is designed so that application developers need to specify the application logic and the deployment logic in separate but structured way. By providing deployment abstractions, Sapphire hides from the application developer the mechanisms to distribute the components of the application. This is attractive for INCA, as we can leverage these mechanisms to integrate the placement of the application components within the network.

In particular, Sapphire allows application developers to wrap their function into Sapphire objects (SO). These SOs are then associated with a deployment manager which specifies how the function can be distributed in the network. Deployment managers can specify where to deploy the SOs, what type of reliability to provide, where to cache the data, what consistency level is required, etc.

Figure 1: Architectural view of INCA

Each supported node in Sapphire must provide a *deployment kernel* that tracks, addresses and potentially migrates the SOs. Calls from one SO to another are routed by the deployment kernel.

Sapphire also provide a centralized coordinator, denoted here as the *Object Management System (OMS)*. Object that requires global coordination register with the Object Management System, so that Sapphire knows where these objects are located and how to connect them.

Sapphire code has been open sourced and is available on GitHub.

4 DESIGN ISSUES

INCA provides different features from Sapphire; namely INCA needs to provide *network and function awareness* in order to deliver a joint optimization of the network and compute layers.

INCA is designed as a mechanism that spans the different network elements. Each element (client, access point, edge node, data center, etc) is equipped with a layer that enables the INCA functions to be deployed. Figure 1 describes the INCA layer. Similar to Sapphire, each node participating in the INCA network must run the function deployment layer (Sapphire kernel server) and register with the controller (Sapphire's Object Management System, OMS). However, these modules are updated to support the INCA goals.

Figure 2 shows the elements of the INCA framework.

Controller (Enhanced OMS): INCA uses a distributed controller that enhances the Sapphire OMS (which is currently a centralized entity). This is a hierarchical controller architecture, where the edge controller manages the resource and the objects located at a specific edge cluster, and registers

Figure 2: Functional Elements of INCA

with one or more cloud controllers. A lightweight protocol synchronizes the edge controller with the cloud controller(s), and specifies what resource are available in the cloud to the edge controller. Edge controllers that are adjacent (under adjacency rules specified by the INCA operators) also periodically exchange information regarding the status of their resource (availability of functions and CPU usage).

Clients register with the OMS located at the edge node (edge controller discovery can be achieved using a variety of mechanisms that we do not discuss here).

To keep allocation of resources manageable, we constrain it to the edge controller being able to forward service requests to the cloud controller. This allows the system to reduce the amount of information to transmit between edge and cloud controllers.

The edge OMS performs the authentication of the clients (through any available mechanism) and ensures that the functions requested by the client are consistent with the access control and other applicable policies.

As in Sapphire, the Object Management System tracks the location of the Objects. However, there is no centralized, global view in INCA. The edge controllers track the location and properties of all the Objects that were issued by one of its clients; the cloud controllers track the objects their own clients requested and the objects instantiated by their dependent edge controllers within the cloud cluster.

The INCA OMS is enhanced with a network monitoring agent. This agent probes the network between edge controllers that are adjacent, and between the edge controllers and the cloud controllers that they are dependent of. This is perform via a combination of active probing and passive monitoring of messages in between these controllers.

Node Architecture: Each INCA node contains a kernel server (KS) that, in addition to the Sapphire functionality, includes a network monitoring agent. This agent periodically relays the network conditions to the OMS that it is associated

with. Each node is responsible to support the execution of services as assigned by the OMS.

Routing and Forwarding: INCA relies on IP routing in the underlay. However, it can be integrated with a SDN controller to provide some service-based paths in between the kernel servers. Similarly, the network monitoring agents may keep track of several routes in between INCA nodes, if available.

Function Caching Proxy: We introduce here this Function Caching Proxy, as a mechanism to provide Information-Centric Networking properties to our framework. Upon registering with INCA, the client connect with a Function Caching Proxy (FCP) that sits at the edge of the network. The client then obtains the location of its OMS using a DNS query. When the client issues a request, it takes the form of a Function Manifest sent to the local OMS. The function request includes the Function Name, but also an object ID (OID) to disambiguate multiple requests for the same function name.

The FCP holds a Function Forwarding Base (FFB) that can be used to forward function requests to the kernel servers that can process it. This FFB is populated by the OMS as part of the Measurement Updates. The FFB only keeps track of the functions related to the server located within a region near the FCP, and does not have a global scope. In our implementation, the FFB lists the function name, the corresponding servers' IP addresses, a priority level. It could also include some instantiation cost or some function affinity.

The FCP is allowed to intercept the Function Manifests, and to process them without the client's knowledge. It forwards the request to the Kernel Server that satisfy client's priorities based on the forwarding table (FFB) updated by the OMS.

If a client moves to a new region, a new FCP with a different forwarding table that fits the client's new region will intercept the function manifest to process it.

Function Placement & Function Store: Functions execute in a JVM or in containers using the GraalVM framework. This allows for ease of application development and for code reuse over multiple platforms.

Upon receiving a request for a service, the edge controller performs an optimization step to decide where to execute the service. The service can specify a level of QoS that it is expecting through either a specific INCA labeling scheme, or through registering the service characteristics with the OMS ahead of time.

The INCA labeling scheme assigns a label to service request that can specify which scope the service is allowed to use (edge, local, cloud) and can also express the CPU requirement, some delay bounds and/or some bandwidth sensitivity of the service. This is used as input in the placement optimization algorithm.

Using the network conditions information, the CPU load and the service requirements allows the OMS to allocate the service to the appropriate kernel server.

Periodic monitoring of the kernel servers may trigger a service migration to improve performance.

Function caches are distributed throughout the INCA network. The distributed OMS tracks what functions are available locally in the cluster of servers that are registered with it, and in adjacent clusters. Functions can be in one of several state: non-existent (the function is not available locally and would need to be downloaded and instantiated); cold (as in binary code that is stored but not loaded); or warm (for the active functions).

The INCA OMS uses this information for delay sensitive application, potentially forwarding requests to warm instances rather than cold or non-existent.

Security Framework: Authentication to controller is done by presenting INCA credentials. The SPIFFE mechanisms could be leveraged to delegate trust for objects that have dependencies onto other objects on other nodes.

5 IMPLEMENTATION

We have implemented INCA as an extension of the Sapphire code base. In order to support INCA requirements, clients in INCA express their functions requests to the OMS through the use of labels. These labels specify certain information relayed to the function such as the function name, function's priority (latency sensitive, CPU intensive, High Bandwidth, etc.) and the client's region. Based on this information, the OMS will select the best suitable kernel server to serve the client's request.

To allow the OMS in INCA to optimize function placement in the network, we have built network monitoring agents that report network conditions to the edge OMS. These network agents are located near the clients and can be instantiated by the OMS whenever it is needed. Also, clients in INCA can volunteer to provide stats regarding their function request.

It is essential (say, for load balancing) for the OMS and the kernel server to have periodic updates at a regular interval. Initially, these kernel servers will register with the OMS by listing the regions they support and available systems resources.

We also implemented some specific functions to test INCA, including a face recognition application and some HTTP server applications (in addition to leveraging existing applications built for Sapphire).

6 EVALUATION

We evaluated INCA under different scenarios to validate the benefit of the approach.

Figure 3: Evaluation Testbed

Figure 4: INCA baseline comparison

First, we tested INCA's ability to detect changes in the network and optimize the function placement. We compare to an "Edge" function placement policy that places the workloads at the edge first, a "Cloud" first policy and a round-robin ("RR") policy. All nodes (clients, servers, and OMS) in our network are running as Docker containers running on the same machine (see Figure 3).

We consider a topology that consists of two edge clusters, and a cloud network. The 200 clients connect to the network via the edge clusters to request a latency-sensitive application. We increase and decrease the latency to connect to the edge and to the cloud.

INCA is able to detect when the latency towards the cloud is competitive with that at the edge. Figure 4 shows the results for our scenarios for INCA compared with Edge, Cloud and RR. For INCA, the average latency for client's application completion was around 5.7ms while the lowest average for a non-INCA case was for Edge (7ms latency).

Next considered 800 clients connecting through only one edge instead of two.

We used fixed latencies between clients, edge, and cloud, where the latency between clients and cloud is higher than client to edge. We created a synthetic traffic model based on the HTTP distribution of requests indicated. As before, INCA is able to instantiate the workload so as to optimize for the delay, displayed on End-to-end delay CDF for all

four scenarios are shown in Figure 5b. For INCA, the average latency for clients application completion was around 22.8ms while the lowest average for a non-INCA case was for RR, with average latency around 23.9ms.

Next we examine INCA's ability to load-balance clients' requests among multiple servers on the same edge to achieve low average latency. The topology here consists of two servers representing two edges. The clients can connect to both edges in this multihoming scenario. Access to the edge from the client network has an additional 3ms latency.

We compare our results to a non-INCA system where multi-homing is also supported but clients requests are be only served in RR fashion. 100 clients request to deploy a latency sensitive application. We vary the round trip time to connect to either server. A non-INCA system does not make use of any information on decision making for future client requests. INCA is able to detect that latency to one of the servers is more than the other and adapt resource allocation.

Figure 6 shows the results: INCA clearly utilize the network resources efficiently by taking advantage of multi-homing. The average latency for client's application response was around 5ms while the average latency for RR was around 7.7ms.

The next evaluation scenario assesses INCA's ability to optimize the workload placement based upon the availability of the specific function at a node. Here, a number clients using different set of latency sensitive applications connect to an access network which in turn connects to a set of edge kernel servers and a set of cloud kernel servers. Initially, none of the kernel servers have the server code locally. The code modules are retrieved from a function repository (HTTP) server in the cloud.

The topology we consider consists of one edge with 10 kernel servers, and an HTTP server located at the cloud. Without INCA, the tasks are allocated with round-robin policy. With INCA, the tasks are allocated preferentially to the kernel servers that have executed the given application in the recent past. Clients measure the service access latency defined as the time that expires before a service is responsive after a client request for service creation is made. INCA outperformed the vanilla cases for all clients as it reuses kernel servers that are "warm," namely with the code downloaded and ready to execute (see Figure 7a and 7b). 75% of the functions requests for all clients were satisfied in less than of 150ms compared to vanilla cases where only 30% of the client's requests were satisfied in less than 150ms.

The final evaluation scenario assesses INCA's ability to optimize the workload placement based upon applications' requirements. For this scenario, a number of clients are using two types of applications, latency sensitive applications and CPU intensive applications. The latency sensitive workload is based on the HTTP distribution model we used in previous

(a) End-to-end latency

(b) Empirical CDF

Figure 5: INCA baseline comparison

Multihoming

Figure 6: Multihoming Evaluation Scenario

(a) End-to-end latency

(b) Empirical CDF

Figure 7: Performance with Awareness of Code Location

scenarios. The CPU workload is designed to consume the system's resource where they run for a minimum of 3 seconds. This will cause applications running on the same system to compete for resources and affect applications performance. Without INCA, the tasks are allocated via a round-robin policy without regard to the applications' requirements. With INCA, the CPU intensive tasks are allocated to the cloud as it's assumed to have more CPU resources compared to the

Figure 8: Mix Apps Evaluation Scenario

edge. For latency sensitive tasks, INCA allocate them to the best suitable server either in edge or cloud.

As before, INCA is able to instantiate the workload so as to optimize for the application requirements. Figure8 shows the empirical CDF for the latency sensitive applications. There, 50% of the functions requests for all clients were satisfied in less than of 11ms compared to RR where only 35% of the client's requests were satisfied in less than 11ms.

7 CONCLUSIONS

We have presented INCA, an In-Network Computing Architecture. We have designed INCA to support easy deployment of applications by application developers over heterogeneous platforms. At the same time INCA provides machinery to flexibly allocate resources in a manner consistent both with the end-user QoE requirements and the network operator's incentives. To do so, INCA monitors the network and receives as input the expected level of service (CPU, delay, bandwidth) and performs an optimization based upon these parameters (according to an operator specified policy) to allocate resource to the service.

We have implemented INCA, leveraging the features of Sapphire, an existing distributed programming framework. Our evaluation results on an actual system shows the clear benefit of a joint network-and-compute optimization.

REFERENCES

- [1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95.
- [2] C. Tschudin et al and M. Sifalakis. 2014. Named functions and cached computations. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*.
- [3] M.J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordström, J. Rexford, and D. Shue. 2010. Service-Centric Networking with SCAFFOLD. In *Princeton University, Computer Science, Technical Report TR-885-10*.
- [4] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, and F. Schamel. 2014. Service oriented networking. In *2014 European Conference on Networks and Communications (EuCNC)*.
- [5] Michał Król and Ioannis Psaras. 2017. NFaaS: Named Function As a Service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking (ICN '17)*.
- [6] AT&T Labs and AT&T Foundry. 2017. AT&T Edge Cloud (AEC) - White Paper. (2017).
- [7] E. Nordström et al. [n. d.]. Serval: An End-Host Stack for Service-Centric Networking. In *USENIX NSDI'12*.
- [8] Mike Roberts. 2016. Serverless architectures. *online: <https://martinfowler.com/articles/serverless.html>* 4 (2016).
- [9] Haoyu Song. [n. d.]. Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane. In *ACM SIGCOMM HotSDN '13*.
- [10] I. Zhang et al. [n. d.]. Customizable and Extensible Deployment for Mobile/Cloud Applications. In *ACM OSDI'14*.