Design and Evaluation of a Scalable Smart City Software Platform with Large-Scale SimulationsA

**Cite as:**

**A Esposte, E Santana, L Kanashiro, F Costa, K Braghetto, N Lago, F Kon. Design and evaluation of a scalable smart city software platform with large-scale simulations in Future Generation Computer Systems, Volume 93, pp. 427-441, 2019.**

Arthur de M. Del Espostea, Eduardo F. Z. Santanaa, Lucas Kanashiroa, Fabio M. Costab, Kelly R. Braghettoa, Nelson Lagoa, Fabio Kona,∗

*aDepartment of Computer Science - University of S˜ao Paulo*

*bInstituto de Inform´atica - Universidade Federal de Goi´as*

# Abstract

Smart Cities combine advances in Internet of Things, Big Data, Social Networks, and Cloud Computing technologies with the demand for cyber-physical applications in areas of public interest, such as Health, Public Safety, and Mobility. The end goal is to leverage the use of city resources to improve the quality of life of its citizens. Achieving this goal, however, requires advanced support for the development and operation of applications in a complex and dynamic environment. Middleware platforms can provide an integrated infrastructure that enables solutions for smart cities by combining heterogeneous city devices and providing unified, high-level facilities for the development of applications and services. Although several smart city platforms have been proposed in the literature, there are still open research and development challenges related to their scalability, maintainability, interoperability, and reuse in the context of different cities, to name a few. Moreover, available platforms lack extensive scientific validation, which hinders a comparative analysis of their applicability. Aiming to close this gap, we propose InterSCity, a microservices-based, open-source, smart city platform that enables the collaborative development of large-scale systems, applications, and services for the cities of the future, contributing to turn them into truly smart cyber-physical environments. In this paper, we present the architecture of the InterSCity platform, followed by a comprehensive set of experiments that evaluate its scalability. The experiments were conducted using a smart city simulator to generate realistic workloads used to assess the platform in extreme conditions. The experimental results demonstrate that the platform can scale horizontally to handle the highly dynamic demands of a large smart city while maintaining low response times. The experiments also show the effectiveness of the technique used to generate synthetic workloads.

*Keywords:* Smart Cities, Smart Urban Spaces, Middleware, Simulation, Scalability, Open Source, Microservices

# Introduction

Most large cities around the world are challenged by problems related to population growth, shortage of re- sources, air pollution, traffic congestion, and public safety,

20

5 among many other modern urban problems. Research on

Smart Cities seeks to mitigate these problems using Infor- mation and Communication Technology (ICT) to leverage the city infrastructure and resources as part of a highly integrated, diverse, and large-scale smart cyber-physical

25

10 environment. The goal is to enable systems and applica-

tions that will ultimately turn the city into a smart envi- ronment, contributing to improve the quality of life of its citizens.

A number of industry initiatives have been success-

30

15 fully adopted in cities around the world, tackling differ-

AThis research is part of the INCT of the Future Internet for Smart Cities funded by CNPq, proc. 465446/2014-0, CAPES, proc. 88887.136422/2017-00, and FAPESP, proc. 2014/50937-1 and

35

15/24485-9.

∗Corresponding author

*Email address:* [kon@ime.usp.br](mailto:kon@ime.usp.br) (Fabio Kon)

ent urban problems. They combine cyber-physical systems with mobile applications to enable the realization of bike sharing services, bus tracking systems, and smart traffic lights, which are now widespread in large urban centers. Smart navigation systems such as Waze and Google Maps are examples of smart city applications capable of serving millions of users by combining geolocation, crowd-sensing, and real-time data. The early adoption of ICT in the con- text of urban problems has significantly impacted multiple application domains. Although several of the mentioned systems handle scalable data processing and services, they do not offer an integrated infrastructure to facilitate the development of other smart city applications through fa- cilities that can be shared across multiple domains and services.

In the academia, several efforts have been devoted to the study and development of the smart city paradigm, leading to a proliferation of initiatives around the world, targeting different domains such as urban mobility, energy management, and healthcare. Despite their early success, there is a need for more effective solutions for data integra- tion and sharing as several of those initiatives have been

*Preprint submitted to Future Generation Computer Systems February 22, 2019*

developed using ad-hoc approaches [[1].](#_bookmark44) They neither fol- 95

low a common set of practices and standards nor consider

40 the need for data and resource sharing among the different systems in the city [[2,](#_bookmark45) [3].](#_bookmark46) This hinders the development of smart city solutions that are both sustainable in the long term and reusable in cities with different characteris-100 tics, thus creating market islands and raising extensibility,

45 adaptability and interoperability issues.

The use of smart city platforms facilitates the fast de- velopment, deployment, and operation of integrated, high- quality smart-city services and applications [[3].](#_bookmark46) Such plat-105 forms typically implement a set of common functional re-

50 quirements in the form of reusable services that applica- tion developers can use, including services for the inte- gration of Internet of Things (IoT) devices, as well as for data storage and processing, and for context-awareness.110 Similarly, the architectures of those platforms may meet

55 multiple non-functional requirements such as adaptabil- ity, privacy, interoperability, and evolvability, according to the specificities of the problem that they intend to solve. In particular, in this paper, we are mainly interested in115 designing and evaluating such platforms for scalability, a

60 non-functional requirement for future cyber-physical sys- tems that is specially critical in the context of smart urban spaces, which may encompass entire cities, with millions

ing platform characteristics in real-world scenarios. This difficulty stems both from the significant deployment and usage hurdles in most platforms and from the lack of ade- quate evaluation techniques.

Middleware platforms for smart cities usually imple- ment complex distributed architectures, requiring consid- erable effort in documentation and automation to run, configure, and test them. Also, these platforms are of- ten implemented as non-open-source software, hampering their use and study by third parties, as well as the repro- ducibility of results and collaborative development. Con- sequently, few projects are truly deployed in production environments, as most of the existing infrastructures are experimental and small in scale.

Even if this was not the case, the research community still needs to advance in the direction of more sophisti- cated methods, tools, and benchmark strategies to enable a comprehensive evaluation of smart city platforms. One challenging related problem is the generation of represen- tative workloads to assess the performance and scalabil- ity of smart city software platforms. Two distinct funda- mental actors must be accounted for: (1) client applica- tions, which generate requests using the platform facilities, and (2) the underlying IoT devices, which continuously produce sensed data and/or receive actuation commands.

of sensors, actuators and humans in the loop [[3,](#_bookmark46) [4].](#_bookmark47)

A smart city platform must handle a large number of

65 devices that are part of or are integrated with the city in- frastructure. It needs to store and process large volumes of data related to the city, which are continuously pro-

120

In this context, the use of randomly generated synthetic workloads or a failure to consider either of these external actors can result in experiments that, although useful for evaluating the system in a limited context, do not repre- sent realistic scenarios. This has been observed in most of

duced and consumed by devices and client applications.125 At the same time, the platform must be able to support

70 thousands of requests from users and from services that run on top of it. The scalability demands thus vary ac- cording to the characteristics of the city, as well as those of the deployed applications and services. For instance, a130 city may start with a pilot project in one of its districts

75 and then expand to other parts of the city as the required infrastructure becomes available.

Despite its importance, platform scalability has not yet received the necessary attention in smart city research.135 According to a previous systematic study of the litera-

80 ture [[3],](#_bookmark46) most of the works that supposedly meet scalability requirements only present superficial discussions of design and implementation decisions that can lead to a scalable architecture. Moreover, they often do not provide scientific140 evidence of the feasibility of their approaches. Sanchez et

85 al. [[5]](#_bookmark48) highlighted that several IoT projects were able to present concrete solutions through new technologies and architectural models, but failed to present conclusive vali- dation of the proposed solutions, which are often restricted145 to proofs-of-concept. However, demonstrating the actual

90 scalability of smart city platforms presents significant chal- lenges due to the lack of available infrastructure for real experimental setups, as well as the lack of comprehensive

the reported evaluations of smart city platforms, as exem- plified by [[6,](#_bookmark49) [7,](#_bookmark50) [8].](#_bookmark51)

In a real smart city, citizens dynamically interact with the cyber-physical infrastructure, triggering multiple events and changing the context observed by pervasive devices and systems. Similarly, platform clients, such as end-user applications and city management services, may vary their interaction behavior with the platform due to events ob- served in the physical world and real-time data provided by the city. This dynamic behavior of citizens and plat- form clients needs to be captured and modeled to generate meaningful and representative synthetic workloads.

Available benchmark tools focus on generating work- loads for Smart Cities or IoT platforms by extrapolating real sensor traces from various contexts or by emulating the behavior of users based on Web traces. The use of smart city simulators can reduce the problems related to workload generation by allowing large-scale experiments on smart city platforms based on more realistic scenarios. Such simulators implement models that reproduce smart city environments based on the behavior of citizens, IoT devices, and other aspects of the city, such as vehicles, buildings, and lamp posts [[9].](#_bookmark52) Although data produced by a simulator is still synthetic, it reflects the individual be- havior of involved actors and their interactions, emulating

datasets.

This state of affairs derives from the difficulty in assess-

150

the dynamic behavior of a city and adapting the simula- tion accordingly. However, making a simulator interact

155

with a smart city platform is not a trivial task. Notably, the scalability of the simulator itself needs to be addressed, as well as its integration with the target platform.

In 2016, we started the development of InterSCity[1](#_bookmark1), an open-source, microservices-based platform that provides a cloud-native software infrastructure to support the de- velopment of smart city projects, applications, and ser-

205

city platforms. Section [3](#_bookmark4) explores the design and imple- mentation details of the InterSCity microservices archi- tecture related to scalability. Section [4](#_bookmark20) presents a brief overview of InterSCSimulator, its integration with the In- terSCity platform, and the modeled smart city scenario. Section [5](#_bookmark31) describes the experimental methodology to as- sess the platform using the modeled scenario and a com-

160

165

vices. InterSCity has a microservices architecture that210 contributes to its scalability. Its modularization through small, single-purpose and loosely-coupled interconnected services enables the evolution of its design, which is a valu- able property to adequately meet the continuously chang- ing demands of novel smart city contexts and applications.

In this paper, we investigate the scalability of the Inter-

SCity platform using InterSCSimulator, a scalable smart

prehensive scalability analysis. Finally, Section [6](#_bookmark43) presents our conclusions and discusses future work.

# Related Work

Research on smart city platforms comprises a wide range of areas and related problems. However, in the

170

city simulator [[9].](#_bookmark52) In previous work [[6],](#_bookmark49) we introduced the InterSCity platform and provided preliminary exper- imental results indicating the feasibility of our approach towards achieving a scalable architecture. Although we

215

context of this work, we are mainly concerned with the

scalability of these platforms. Therefore, we divide rele- vant related work into two categories: (1) projects that propose novel smart city platforms that address scalabil- ity challenges, and (2) tools and methodologies to generate

have maintained the main InterSCity modules in relation

220

workloads for experiments on smart city platforms.

175

180

185

190

to our previous work, we have refined important archi-

tectural decisions regarding microservices communication and deployment, enabling greater scalability and the in- tegration with the simulator. Also, we have implemented several improvements in the microservices code base re- lated to database queries, caching mechanism, and data se- rialization to avoid performance issues. This paper brings three new fundamental contributions:

225

A detailed exploration of the current architecture and implementation of the InterSCity platform, sup- ported by a deeper discussion of the design decisions230 made to meet scalability requirements and more com- prehensive experiments considering realistic scenar- ios and covering broader requirements than the ones of the previous version presented in [[6].](#_bookmark49)

*•*

235

Advances on performance evaluation of smart city platforms, with the proposal of a new method for workload generation that addresses the dynamism of smart cities by using large-scale simulations inte-

*•*

* 1. *Scalable Smart City Platforms*

A number of smart city platforms have been proposed in the literature to address challenges in the development and deployment of smart city applications and services. One of the most relevant smart city initiatives is Smart- Santander [[10],](#_bookmark53) which includes an infrastructure to collect city data using more than 20,000 IoT devices, together with a software platform to store this data and provide it to other applications. The city of Santander has less than 200,000 inhabitants; we did not find any discussion on the use of this platform in more complex contexts, such as in a large metropolis. Also, to the best of our knowledge, no evaluation of the scalability of the SmartSantander plat- form has been published yet.

CiDAP [[11]](#_bookmark59) is a big data analytics platform for smart cities deployed in the SmartSantander testbed. The main objective of this platform is to use the data collected from the testbed and analyze it to understand the city’s behav- ior. CiDAP uses a set of Big Data tools, such as Apache

195

200

grated with the platform.

An application of the simulation-based experimental strategy to evaluate the InterSCity platform based on a Smart Parking application scenario. We pro- vide an extensive analysis of the platform’s scalabil- ity properties, advancing the knowledge on the use of the microservices architectural pattern to develop smart city solutions.

*•*

This paper is organized as follows. Section [2](#_bookmark0) discusses related work targeting either architectural proposals for scalable smart city platforms or the generation of work- loads for scalability and performance evaluation of smart

1<http://interscity.org/software/interscity-platform>

240

245

250

255

Spark and NoSQL databases, to improve the scalability of the platform. Access to data collected by the plat- form is possible through a REST interface. Experiments showed that CiDAP can handle almost 300 sensor data up- dates per second. However, the authors failed to evaluate their platform scalability considering other city-scale re- alistic scenarios. Moreover, their platform does not meet key functional requirements to support the development of smart city applications, such as resource discovery and actuator support.

Similarly to CiDAP, the Scallop4SC (SCALable LOg- ging Platform for Smart City) [[12]](#_bookmark61) and the platform pro- posed by Girtelschmid et al. [[13]](#_bookmark62) rely on the use of Big Data tools such as Hadoop and Storm. Both works collect data from the city infrastructure, store the data in NoSQL databases and process the data using services implemented

260

265

270

275

280

285

290

295

300

with such tools. Likewise, the InterSCity project takes ad-310 vantage of existing open source tools to leverage its func- tionalities and performance. However, neither of the two projects mentioned above present experiments to demon- strate the maximum workload supported by the platforms.

DIMMER is a microservices-based smart city IoT plat-315 form focused on energy efficiency and management [[14].](#_bookmark63) This is one of the few previous works that explore the use of microservices to build smart city solutions, provid- ing important discussions regarding the design and orga- nizational impact of this architectural style in this con-320 text. The creators of DIMMER state that microservices enable horizontal scalability, data partitioning, and scaling through functional decomposition. However, no experi- mental evaluation of the platform was reported to support these statements. In contrast, and supported by experi-325 mental results, our work closely examines the impact of adopting a microservices architecture in a more compre- hensive smart city context.

OpenIoT[2](#_bookmark2) is an open source middleware for the de- velopment of IoT-based applications. It has an API to330 manage a Wireless Sensor Network and a directory service to discover deployed sensors; it also has a component for service definition and access. A Smart City project called Vital [[15]](#_bookmark64) was built on this platform.

CitySDK is an API that provides data collected from335 the city for use in the development of smart city appli- cations and services. According to Pereira et al. [[16],](#_bookmark70) CitySDK is available for Amsterdam, Helsinki, Lamia, Lis- bon, and Rome using a reference implementation of the API. The primary requirements handled by the API is340 the use of static and dynamic data, caching for enabling offline use, interoperability, distribution, and scalability. The platform provides a set of REST services that call a data management layer which in turn accesses a MongoDB database. Despite the deployments of CitySDK and the345 stated scalability requirement, for the best of our knowl- edge, there are no studies about the maximum workload supported, neither numbers about the real scenarios al- ready faced by the platform.

The CityPulse framework facilitates the creation of city services providing a distributed system for real-time data stream processing, data analytics, and semantic discovery. The main goal is to provide cross-domain data integration, tackling the problem of silo applications for smart cities. The framework deals with data from different formats and

350

quality to create reliable city services. Puiu and Barnaghi

smart cities. To this end, the project specifies a reference architecture composed of Generic Enablers (GE), which are software components that implement general-purpose functions that can be combined through open APIs to pro- vide middleware facilities for application development. FI- WARE offers an open specification for each of its GEs, de- scribing its essential functionalities, interfaces, and APIs, along with an open source solution that implements the specification. FIWARE GEs have been exploited in vari- ous domains [[18,](#_bookmark73) [19,](#_bookmark74) [20,](#_bookmark75) [21].](#_bookmark76) These works usually combine a very small set of GEs to provide the basis for the devel- opment of different prototype applications for end users. Such projects demonstrate that smart applications can benefit from FIWARE components as reusable building blocks. However, these works also evidence the technical disparity between the different GEs in terms of technical quality, maturity, ease of integration, documentation, and provided functionalities.

The projects discussed above describe tools and ar- chitectures for the design of smart city platforms whose features and principles have influenced the design of the InterSCity platform. However, most of these works failed to either provide a thorough explanation of how they ad- dress the scalability problems or perform scientific vali- dation of the scalability of the proposed platforms. Such problems are heightened in cases where the documentation is lacking or the source code is not available, thus limiting the analysis of scalability and exploratory work by the re- search community to just considering the discussions that were presented in the original publications. To tackle these problems, our work deepens the discussion about the use of a microservices architecture on smart cities via the open- source InterSCity platform. In addition, this paper intro- duces reproducible scientific experiments to demonstrate the scalability of the platform. To this end, a companion experimentation package is also made available[3](#_bookmark3) which can be used and extended by the research community.

* 1. *Workload Generation for Smart City Platforms*

One of the major difficulties to evaluate the scalability of smart city platforms is workload generation. Such plat- forms interact with two layers that depend on each other at runtime: the city IoT infrastructure and client appli- cations. The city dynamics (e.g., people and cars moving across the city, car accidents, traffic jams, and weather conditions) directly impact the infrastructure. For exam-

305

[[17]](#_bookmark72) present a study case of the framework using a Travel Planer application. However, no scalability analyses or discussion is presented in the paper.

FIWARE is a multi-million Euro initiative funded by the European Commission whose goal is to develop a large collection of open source components and tools to facilitate the development of Future Internet applications, including

355

360

ple, if in a specific region many cars are moving in a spe-

cific direction, intelligent traffic lights should be kept open to optimize the flow. Conversely, changes in the cyber- physical infrastructure affect the city behavior as well. For instance, if a driver is heading to an available parking spot and, in the meantime, a sensor detects that the spot has been occupied by another car, the driver will keep looking for another available parking spot, increasing the number

2OpenIoT - https://github.com/OpenIotOrg/openiot

3<https://github.com/LSS-USP/interscity-k8s-experiment>

365

370

375

380

of vehicles moving around. Therefore, a realistic workload in a smart city context must consider the dynamic inter- action between the two layers. However, since real-world deployments of smart city platforms are still rare, there is420 a lack of real traces of this kind of systems that could be used to better characterize workloads.

IoT and smart city benchmarks, such as RIoTBench [[22]](#_bookmark54) and CityBench [[23],](#_bookmark55) can be used to generate workloads re- lated to a city’s IoT infrastructure. These benchmarks are425 based on real traces of IoT devices scattered across the city. Their main goal is to assess distributed stream processing systems, which are a crucial part of a smart cities plat- form architecture. RIoTBench includes four stream-based workloads derived from real IoT observations of smart cities and personal fitness devices, with peak stream rates that range from 500 to 10,000 messages/sec, and different frequency distr[ibutions[22].](#_bookmark54) CityBench, in turn, focuses on RDF stream processing engines. It includes real-time IoT data streams generated from various sensors deployed in the city of Aarhus, Denmark [[23].](#_bookmark55) Although these works

430

represent significant steps towards the evaluation of smart

they do not encompass the two layers (city IoT infrastruc- ture and client applications) and their cross-interactions. Our proposal to solve this problem is to use a scalable agent-based smart city simulator that manages the behav- ior of both the IoT infrastructure and the client applica- tions at the same time, while also enabling the analysis of the consequences of their interactions via the platform during the simulation. The following sections present this approach.

# The InterSCity Platform

Future smart cities will demand high-quality research in multiple areas. The InterSCity project [[27]](#_bookmark60) is a multi- disciplinary consortium that aims to develop scientific and technological research to address key challenges related to the software infrastructure of smart cities, focusing on Networking and High-Performance Distributed Comput- ing, Software Engineering, Data Analysis, and Mathemat- ical Modeling. The project aims at enabling the develop-

385

city platforms, they are limited to a set of specific function- alities (such as data stream processing and the handling of several IoT devices at the same time) and encompass only the behavior of the underlying IoT layer, failing to consider the behavior of citizens and applications.

Another technique to generate workloads that repre-

435

440

ment of reusable open-source technologies and methods to

support future smart cities while advancing the state-of- the-art.This section presents the InterSCity platform[5](#_bookmark6), an open-source, microservices-based middleware to support the development of smart city applications and to enable novel, reproducible research in the field.

The InterSCity platform offers a set of high-level, Web-

390

395

sent the city’s IoT infrastructure is the use of simulation.

SenSE (Sensor Simulation Environment) [[24]](#_bookmark56) is an open source synthetic traffic workload generator, developed to simulate complex environments, such as smart cities. This tool can generate massive amounts of heterogeneous sensor data simultaneously, being able to simulate tens of thou- sands of sensors based on pre-defined probability distribu- tions [[24].](#_bookmark56) However, it is also limited to the IoT layer.

445

To generate workloads representing client applications of smart city platforms, common HTTP workload genera-

450

[4](#_bookmark5)

based services that provide facilities to manage heteroge- neous IoT services and resources, as well as to support the discovery of city resources based on context data, to store and process data, and to intermediate actuation com- mands. These services cover essential features required to support integrated smart city applications in different do- mains. The InterSCity platform intermediates communi- cation between smart city applications and the IoT ser- vices of the city, offering high-level abstractions that hide the complexity of city-scale data management and the in-

400

tion tools can be used, such as ApacheBench

and HTTPerf [[25](#_bookmark57)]h. erent particularities of the communication with the un-

However, characterizing the workload to represent the fre-

quency and peaks of user requests is not easy. Workload characterization is based on the analysis of measurements

455

derlying IoT devices.

In addition to **scalability**, the main design require- ments for the platform include: **flexibility** and **extensi-**

405

collected from the infrastructure during its operation [[26].](#_bookmark58)

When real representative traces from smart city applica- tions are not available, workload characterization is not a trivial task. This aspect is important because cities

and their citizens do not behave the same way during the

**bility**, achieved via modular, decoupled, distributed ser- vices that enable independent evolution of components and facilitate the addition of new features; **interoperability**, achieved with the adoption of open, well-accepted stan-

410

course of a day, week, month or year. In addition, there is the occurrence of unexpected events. Thus, a realistic workload must include such variations.

To the best of our knowledge, no previous work ex- plores the dynamic interactions among the different actors in a city to characterize experimental workloads. The ap-

460

465

dards and protocols; code writing **productivity** and **re-**

**liability**, achieved with the reuse of robust, highly-tested open-source tools, libraries, and frameworks. By meeting these requirements, we were able to design an evolvable architecture facilitating the addition of new features and resolving performance bottlenecks.

Therefore, the platform design is based on a microser-

415

proaches described above present significant limitations in

generating realistic workloads for smart city platforms as

vices architecture from the outset, aiming to provide a

4<https://httpd.apache.org/docs/2.4/programs/ab.html>

5The InterSCity source code is available at [http://interscity.](http://interscity.org/software/interscity-platform) [org/software/interscity-platform](http://interscity.org/software/interscity-platform) under the MPL 2.0 license

470

475

480

485

490

modular, evolvable, and scalable middleware infrastruc-525 ture that can be easily extended and shared among multi- ple research groups, smart city initiatives, and code de- velopment communities. Microservices are an architec- tural style that promotes the modularity of a system in terms of a set of distributed, fine-grained, independent530 services that collaborate via network protocols such as Web services or remote procedure calls [[28].](#_bookmark65) The microser- vices model emerged from the software industry efforts to build large-scale distributed systems combining the guide- lines of traditional Service Oriented Architecture, Domain- Driven Design, continuous delivery, on-demand virtual-535 ization, infrastructure automation, and small autonomous teams [[29].](#_bookmark66)

Up to this date, the InterSCity platform has already been used in four software development courses in differ- ent Brazilian universities to support the development of smart cities projects, reaching nearly 100 undergraduate and graduate students. Also, a Smart Cities Hackathon was promoted at the University of S˜ao Paulo in which teams developed hardware and software solutions based on the platform. These allowed us to experiment with the platform in different domains such as mobility, health,

540

sports, interaction with social networks, and street light-

vices, avoiding unnecessary overheads of object serializa- tion and deserialization; and refinement of the queries and database indexes most commonly used by applications.

InterSCity leverages the microservices architecture as its base strategy to achieve scalability. This is due to three fundamental reasons, which we explore in further detail in the remaining of this section:

Functional Decomposition: the microservices archi- tecture promotes modularity via single-purpose, small services that communicate through lightweight mech- anisms to achieve a common goal. Also, functional decomposition leads to scalability as the system work- load is split among distributed microservices.

*•*

Design and Technology Heterogeneity: different de- sign and technology decisions can be made for dis- tinct microservices, although they do not exclude general choices that apply to the entire system; and

*•*

Independent Deployment: each microservice can be deployed, replicated, and replaced independently.

*•*

* 1. *Functional Decomposition*

ing. Each of these events served as a valuable opportunity

545

By implementing a microservices architecture, the In-

terSCity platform decomposes its functionalities across a

495

500

to detect non-optimal design decisions leading to refine-

ments of the platform implementation and documentation. Recently, research groups from other universities started to use the platform as a basis for their research. For ex-

550

ample, the authors of ContextNet extended the InterSCity

platform to include support for Complex Event Processing [[30].](#_bookmark67) With such experiences, we have been collecting feed- back from the users of the platform, which have helped us

to refine the endpoints provided in our API and to add

set of small, interconnected, collaborative services. Fig- ure [1](#_bookmark7) presents an overview of the InsterSCity architec- ture, exposing its main microservices: Resource Adaptor, Resource Catalog, Data Collector, Actuator Controller, Re- source Discovery, and Resource Viewer. The figure also highlights the top and bottom boundaries of the system, which are accessed by smart city applications and by IoT systems, respectively, through Representational State Trans-

505

new useful features, such as the support for geolocalized searches. Besides, it also motivated some external contri- butions either through patches of code or bug reporting.

A broader description of the InterSCity platform was

555

fer (REST) APIs over the Hypertext Transfer Protocol

(HTTP).

The architecture of InterSCity supports decentralized management of city resources, dividing functional respon-

sibilities and data persistence across the microservices.

510

515

520

presented in a previous paper [[6],](#_bookmark49) covering high-level de-

560

tails of its architecture, modules, interfaces, design prin-

ciples, and development methods, such as DevOps tech- niques and practices used by open-source communities. Therefore, in this paper, we focus on presenting the refined decisions in the systems-level design and in the implemen-

565

tation of the platform to meet its scalability requirements.

Compared to the previous version of InterSCity, the main enhancements we introduced in the new version de- scribed in this paper did not impose significant changes in the high-level system components. The main changes im-

570

proved platform internal aspects such as deployment pro-

cedures, communication mechanisms among the microser- vices, and improvements in the microservices code base to resolve performance bottlenecks. Such enhancements in- clude using caching mechanisms to save the latest sensor

575

data and avoid database locking when reading from tables

that receive many writes; providing finer-grained API end- points for internal communication between the microser-

The Resource Adaptor is a proxy microservice that sim- plifies communication between external IoT systems and the rest of the platform. It provides a single entry point for the underlying IoT gateways to register and update resources on the platform, post sensed data from those resources, and subscribe to events that indicate actuator commands. As a proxy, Resource Adaptor is responsible for validating the requests, augmenting them with additional metadata and required adaptations. The adaptor either broadcasts the requests to the entire system or calls a spe- cific microservice to handle the request synchronously.

The Resource Catalog is a vital microservice of the In-

terSCity architecture since it manages the static data of city resources, working as a catalog for these resources. Af- ter registering a new resource, the Resource Catalog assigns it a new Universally Unique IDentifier (UUID) and asyn- chronously notifies the resource creation event to other microservices. Both client applications and IoT gateways

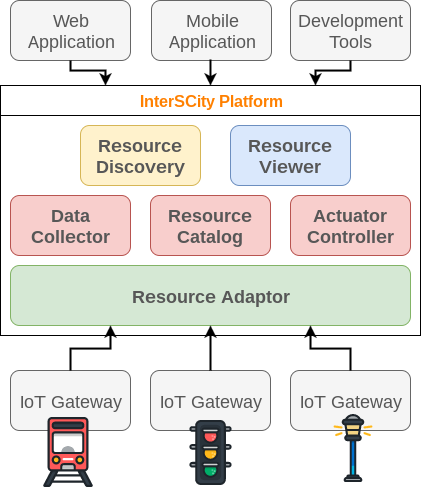


Figure 1: The InterSCity platform architecture (extracted from [[6])](#_bookmark49)

610

615

620

625

630

For instance, one may search for all available city resources capable of monitoring the environment located within a radius of 1 kilometer from a specific location, and whose temperature sensors are scoring above 18 ◦C. Similarly to Resource Dicovery, the Resource Viewer microservice also orchestrates the Data Collector and Resource Catalog to

gather both static data and sensor data for visual presen- tation. It is a front-end microservice for presenting general and administrative information regarding city resources which are useful, e.g., for platform management.

As a result of its functional decomposition, the work- load of InterSCity is handled by separate distributed ser- vices, cooperating towards its scalability. The partitioning of the system workload is also reflected on the database tier, since each microservice has its database and manages a small set of the system’s data. Different smart city con- texts may lead to different ways of splitting the workload of the platform among its microservices in a non-uniform way. For instance, in a city with a large number of sensors, two microservices of the InterSCity platform may be used more often: Resource Adaptor and Data Collector. In an- other type of scenario, a city with actuator resources might have management systems that send actuation commands to the underlying IoT devices, thus placing a higher de- mand on Actuator Controller.

must use a UUID in later interactions that target a specific resource, such as to get and publish sensor data.

635

* 1. *Design and Technology Heterogeneity*

Since microservices communicate via standardized pro-

580

585

City resources may have functional capabilities to pro-

vide data and receive commands, which are respectively supported by sensors and actuators coupled to the re- source. InteSCity splits the responsibility for managing sensors and actuators between two microservices: Data Collector and Actuator Controller. Data Collector is respon- sible for storing and providing access to data collected by city resources. An item of sensor data consists of either

640

tocols, each microservice can be built with the most appro- priate stack of technologies for its specific purpose. They can also be maintained in separate repositories, enabling polyglot persistence and technology diversity. Table [1](#_bookmark8) shows the technologies used by InterSCity services.

Although we embrace the diversity of technologies in InterSCity, we apply this principle with some caveats to

context information or an event linked to a resource capa- bility that is observed at a particular time. Data Collector

645

avoid increasing overall system complexity and the prolif-

eration of practical problems since a radical adoption of

technology diversity could lead to an unmaintainable ar-

590

595

600

605

provides an API to allow access to both current and his-

torical context data of city resources using a rich set of filters that can be accessed via search endpoints. The Ac- tuator Controller microservice, in turn, enables client ap- plications to send commands to change the state of city resources that have actuator capabilities. It is responsible for receiving and validating actuation requests from clients and asynchronously notifying the underlying IoT gateway through Webhooks. This microservice also records the history and tracks the status of actuation requests made through its API.

Client applications need an easy way to find the re- sources available in the city so they can interact with them through the platform. For this purpose, the Resource Dis- covery microservice provides a sophisticated context-aware search API. It orchestrates the Data Collector and Resource Catalog to offer a set of filters by combining their results based on location and static data (i.e., resources’ type and capabilities), and by matching rules for latest context data.

650

655

660

chitecture [[31].](#_bookmark68) For this reason, we initially opted to use Ruby-based tools to develop the foundation services of the platform due to the high productivity and flexibility of this language and the smooth learning curve for newcom- ers who aim to contribute with the InterSCity codebase. Technological and design aspects related to contributions in the form of new microservices are discussed with the project team and may use other technical options accord- ing to their purpose. However, if the new microservices are supposed to be maintained within the main platform repository, they must have at least one maintainer that has fluency in the adopted technologies and must conform to the quality requirements of the project, such as having highly automated test coverage.

In addition to the technology aspects mentioned above, several other design decisions permeate the development of a microservice for the platform. One of the main issues is to define strategies to enable a microservice to scale hori- zontally while offering its functionalities with acceptable

Table 1: Technology Stack of InterSCity Microservices

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service** | **Language** | **Framework** | **Database** | **Cache** |
| Actuator Controller | Ruby | Ruby on Rails | MongoDB |  |
| Data Collector | Ruby | Ruby on Rails | MongoDB | In-memory MongoDB (Percona) |
| Resource Adaptor | Ruby | Ruby on Rails | PostgreSQL | Redis |
| Resource Catalog | Ruby | Ruby on Rails | PostgreSQL | Redis |
| Resource Discovery | Ruby | Ruby on Rails |  |  |
| Resource Viewer | Javascript | EmberJS |  |  |

665

670

675

680

685

performance. Several decisions may affect the internal implementation of a service, such as the use of caching mechanisms, database schema and indexing, the choice of algorithms, and API design. Decisions regarding how a microservice interacts with the whole ecosystem of the platform are also relevant in this perspective, such as com- munication protocols, discovery of other services, and the handling of asynchronous messages, to name a few.

InterSCity implements two fundamental communica- tion mechanisms to support the scalable orchestration of its microservices: (1) Synchronous communication over HTTP and (2) Asynchronous messaging through the Ad- vanced Message Queuing Protocol (AMQP)[6](#_bookmark10).

For HTTP communication, we employ the *API Gate- way* design pattern[7](#_bookmark11) by using Kong[8](#_bookmark12), a distributed, scal- able open-source gateway that aims at facilitating the or- chestration of microservice APIs. Kong is backed by NG- INX[9](#_bookmark13) and extends its HTTP Web server features to offer a dynamic management layer for the microservices HTTP APIs. In summary, the API Gateway receives all incoming HTTP requests, determines which InterSCity microservice should respond to a specific request, and forwards the re-

quest to the identified microservice. For this purpose, we

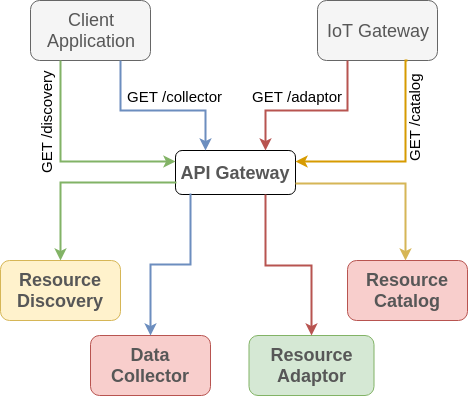


Figure 2: InterSCity API gateway

pattern[10](#_bookmark14), to register itself with the API Gateway as a target for an existing URI rule. In the example, distinct Resource Catalog instances register themselves as targets

use URI-based rules: the system uses the path part of an HTTP request to identify the target microservice of

710

for requests that match the */catalog* URI rule, indicated

by the black arrows. This approach enables the dynamic creation of new instances of a service to split and dis-

690

the request. Figure [2](#_bookmark9) illustrates this strategy, highlighting

the interaction of a client application and an IoT Gateway

with the platform through the HTTP APIs of its microser-

tribute the increasing workload using a round-robin strat- egy, without affecting other parts of the system or requir-

695

vices, which in turn are accessed via the API Gateway.

A clear advantage of implementing the above men- tioned API Gateway strategy is that clients only need to interact with a single entry point (host address and port) to access all InterSCity facilities, instead of keeping refer- ences to multiple microservices.

The API Gateway also enables scalability as it lever-

715

720

ing further reconfiguration. Round-robin load balancing is

represented by the blue and yellow arrows in Figure [3.](#_bookmark16)

The API Gateway constantly monitors the availability of target instances by health checking the endpoints in or- der to adjust its load balancing accordingly by not sending new requests to unhealthy nodes. It identifies microservice instances as healthy or unhealthy based on the status code in HTTP responses. A success code indicates a healthy

700

705

ages the load balancing feature of NGINX so that it is

possible to easily deploy multiple instances of the same ser- vice. Figure [3](#_bookmark16) illustrates this by presenting the main inter-

725

actions among clients, the API Gateway, and the available

microservice instances. Whenever we run a new instance of a microservice, the microservice uses Kong’s REST API, which is an implementation of the *Self-registration* design

endpoint, whereas non-success status codes, timeouts and TCP errors denote an unhealthy endpoint.

As Kong allows the registration of service instances, the API Gateway also supports the *Server Side Discov- ery* design pattern[11](#_bookmark15). This pattern facilitates HTTP-based communication among microservices in the platform since instead of having to manage references to all running in-

6<https://www.amqp.org/> 7<http://microservices.io/patterns/apigateway.html> 8<https://getkong.org/>

9<https://www.nginx.com/>

10<http://microservices.io/patterns/self-registration.html> 11[http://microservices.io/patterns/server-side-discovery.](http://microservices.io/patterns/server-side-discovery.html)

[html](http://microservices.io/patterns/server-side-discovery.html)

730

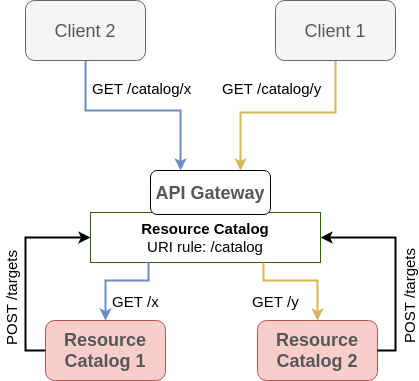


Figure 3: InterSCity load balancing HTTP requests

stances, clients need to know only the address of the API

765

770

775

780

platform, a new message is published on topic **data stream** with a routing key of the form **uuid.capability.others**, where *uuid* is the unique identifier of the resource, *capabil- ity* is the type of the sensor data, and *others* represents any additional information regarding the posted sensor data that could also be used to build the routing key. Similarly, subscribers must inform a binding key when defining a queue to receive messages from a topic.

Figure [4](#_bookmark17) shows an example of an InterSCity topic-based message exchange using routing keys. When the Resource Adaptor receives sensor data from the underlying IoT in- frastructure, it publishes the data on topic *data stream*, which has consumer queues for Data Collector, Resource Catalog, and other services interested on the topic. As Data Collector is responsible for saving the history of sen- sor data, it is interested in any data on this topic (binding key: **#**). On the other hand, Resource Catalog is only interested in sensor data that contains geolocation infor- mation so that it can update the location of moving city resources on its database (binding key: **#.location.#**). Other microservices could be interested in any data from a

735

740

745

Gateway, using URI-based rules to communicate with the target microservice, as shown in Figure [2.](#_bookmark9) A direct ad- vantage of using the API Gateway design pattern is the guarantee that requests for a given microservice will be handled by any of its instances in an independent way, al- lowing for efficient load-balancing. This is an important feature since the number of service instances and their lo- cations may change dynamically.

Although services provide well-defined RESTful APIs, we adopt asynchronous communication whenever possible to avoid the additional latency of blocking synchronous request-reply interactions. To achieve this, a microservice may use the publish/subscribe design pattern rather than directly exchanging messages with other services. Inter- SCity carries out asynchronous messaging by using Rab- bitMQ[12](#_bookmark18), a widely used, lightweight, open-source messag-

785

specific type of sensor, such as temperature data (binding

key: **\*.temperature.#**), regardless of which city resource provided the data.

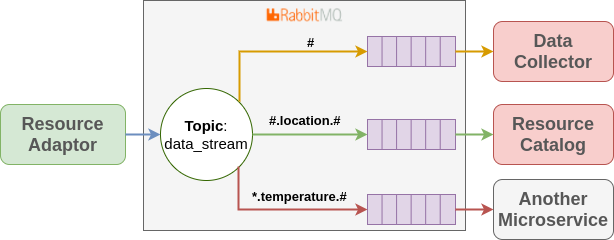


Figure 4: InterSCity Asynchronous messaging

To further decrease the latency of communication among microservices and thus improve its scalability, InterSCity

ing middleware that implements the AMQP protocol.

Events of different types may generate data that need

790

employs a worker-based strategy for asynchronous messag- ing. InterSCity services that need to receive asynchronous

750

755

760

to be broadcast to other modules in the platform. Exam- ples are the registration of a new IoT resource, the recep- tion of sensor data, and requests to actuators. Each type of event is mapped to a *topic*, to which interested services can795 subscribe to receive new messages through queues main- tained by RabbitMQ. By default, each subscription will create a new queue. Messages sent to a specific topic are pushed to all of its subscribed queues.

InterSCity thoroughly exploits RabbitMQ messaging800 features by using routing keys that enable the use of more sophisticated criteria to route messages to subscribers. A routing key is a list of dot-separated strings related to metadata about InterSCity’s internal abstractions. They are added by publishers when sending messages to Rab- bitMQ. When a city resource publishes sensor data on the805

12<https://www.rabbitmq.com/>

messages from the platform must implement a background worker for each subscribed topic in RabbitMQ. By design, these services support the addition of more workers to im- prove the processing rate of the queued jobs by implement- ing the *Competing Consumers* design [pattern[32].](#_bookmark69) Figure [5](#_bookmark19) presents this design pattern, where additional workers read messages from the same queue concurrently and as fast as possible, enabling parallel processing of background tasks. As workers read from the same queue, messages are not replicated for each of them. Consequently, in the example, messages **X** and **W** will be processed in this order by the first two workers that finish their current jobs, represented by messages **Y**, **Z**, and **A**.

* 1. *Independent Deployment*

An important aspect of the InterSCity architecture is that its components are designed as single-purpose, inde-

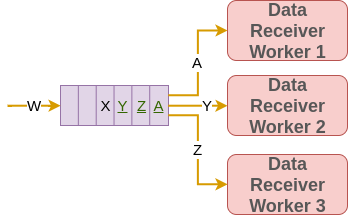


Figure 5: Competing Consumers design pattern

pendently deployable services. Two sets of deployment-

855

860

croservices into Docker[13](#_bookmark21) containers and used them in ad- dition to the containers provided by the community for the supporting tools (i.e., RabbitMQ, Kong, MongoDB). Moreover, we use Kubernetes[14](#_bookmark22) to support the deployment of Docker containers throughout a cluster of virtual ma- chines in a cloud infrastructure.

Kubernetes is an open source project that aims at au- tomating the management of container-based applications by offering tools for supporting two critical tasks. First, it provides a declarative structure through YAML files for developers to define the desired state for the containers that comprise the managed system [[33]](#_bookmark71) in compliance with the *Infrastructure as Code* (IaC) principle, one of the pil-

810

815

820

825

830

835

840

845

related design decisions need to be considered: those re- lated to the entire system, and those that concern individ-865 ual microservices. The former corresponds to decisions re- lated to the choice of cloud provider, the allocation of com- puting resources for the system, the packaging of service instances (i.e., virtual machines, containers, and physical hosts), the handling of common procedures (e.g., update,870 backup, and monitoring), and the deployment of comple- mentary services used by the system (e.g., databases, mes- sage brokers, and proxies). The latter concerns the distri- bution of microservice instances across the available hosts (e.g., single service per host, multiple services per host),875 service configuration, failure recovery, and scaling of a mi- croservice to appropriately respond to changes in the work- load.

Another related issue refers to the deployment of in- dividual microservice instances across the different tiers880 of the distributed execution environment: cloud, fog, and edge. For instance, a microservice that interacts heavily with city infrastructure resources, such as Resource Adap- tor, could be dynamically deployed at the edge to reduce latency. Conversely, a microservice that requires signifi- cant computational power or a more global view of the system, such as Resource Viewer, could be deployed in the885 cloud. This aspect of microservice deployment, especially considering dynamic redeployment, is the subject of ongo- ing research in the project.

The above examples of decisions related to the deploy- ment of microservices-based systems highlight the multi-890 ple trade-offs and challenges faced by engineers, as also observed in [[31].](#_bookmark68) DevOps techniques and tools are very successfull in achieving a reliable and reproducible deploy- ment process, as well as in improving the operating envi- ronment [[29,](#_bookmark66) [31].](#_bookmark68) The inherent complexity of the Inter- SCity distributed microservices environment requires the895 use of such techniques to automate the necessary proce- dures. These include automated tests and the upgrading of microservice instances in the production environment, as well as support for more complex tasks, such as scaling and distributing microservices automatically based on the

lars of DevOps [[34].](#_bookmark77) Second, the Kubernetes engine runs

the specified configuration on the cloud provider to man- age the deployment of the system by scheduling containers across the available machines.

In dynamic smart city contexts, with varying demands throughout the day and random citizen behavior, Inter- SCity must be able to individually scale out the stressed services to properly support workload fluctuations, rather than scaling the entire system as a whole. As we designed InterSCity microservices as stateless services, we can place several copies of the same microservice behind a load bal- ancer (i.e., Kong). Likewise, multiple instances of the same microservice leverage the worker-based approach to dis- tribute processing jobs across asynchronous workers. By monitoring computational resources used by the platform microservices, Kubernetes is capable of adjusting the num- ber of instances for each microservice to adequately and automatically support a varying workload.

# Smart City Simulation for Workload Generation

This section discusses the fundamental steps to enable simulation-based workload generation. This approach is meant to support comprehensive performance and scala- bility experiments on top of the InterSCity platform, espe- cially considering realistic, large-scale smart city scenarios. In particular, we present InterSCSimulator, a smart city simulator for the behavior of city actors and their inter- actions in large-scale settings. We also describe the sim- ulated scenario and how we implemented the integration between the InterSCity platform and InterSCSimulator to perform the experiments.

* 1. *InterSCSimulator*

InterSCSimulator is an open-source, scalable simula- tor for large-scale smart city scenarios [[9]](#_bookmark52) developed in the context of the InterSCity project. In its current version, the simulator provides a set of mobility models for cars,

850

monitoring of services and computational resources.

For this purpose, we packaged all the InterSCity mi-

900

pedestrians, buses, and the subway. Previous work shows

13<https://www.docker.com/> 14<https://kubernetes.io/>

905

910

915

that InterSCSimulator is capable of simulating an entire city such as S˜ao Paulo, with more than 10 million software agents virtually moving across tens of thousands of streets. The simulator is implemented in Erlang, a language suit- able for the development of highly parallel and distributed applications based on the actor model.

Figure [6](#_bookmark25) shows the simulator architecture. InterSC- Simulator builds on top of the Sim-Diasca simulator [[35]](#_bookmark78) to perform general-purpose simulation activities such as Simulation Time Management, Random Number Gener- ation, Load Balancing, and Base Actor Modeling. Inter- SCSimulator contains the city street map graph and the required actors for Smart City simulation such as cars, buses, subways, and sensors. Actors execute tasks during

a simulation clock tick, which represents the simulation of

**Required Inputs**



**config.xml**

**map.xml**

**trips.xml**

**Optional Inputs**

**subway.xml**

**Scenario Definition**

**Simulation Engine**

**Output**

**events.xml**

**park.xml**

Figure 7: InterSCSimulator components

* 1. *Smart City Scenario*

one second in the real world. Lastly, the top layer com-

935

To evaluate the InterSCity platform, we implemented a

Smart Parking scenario on InterSCSimulator, which uses

920

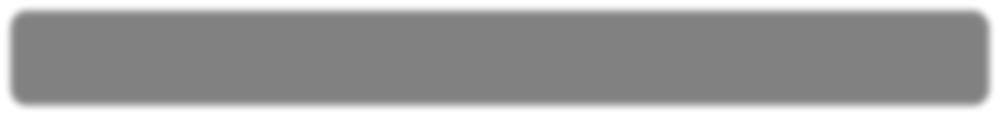
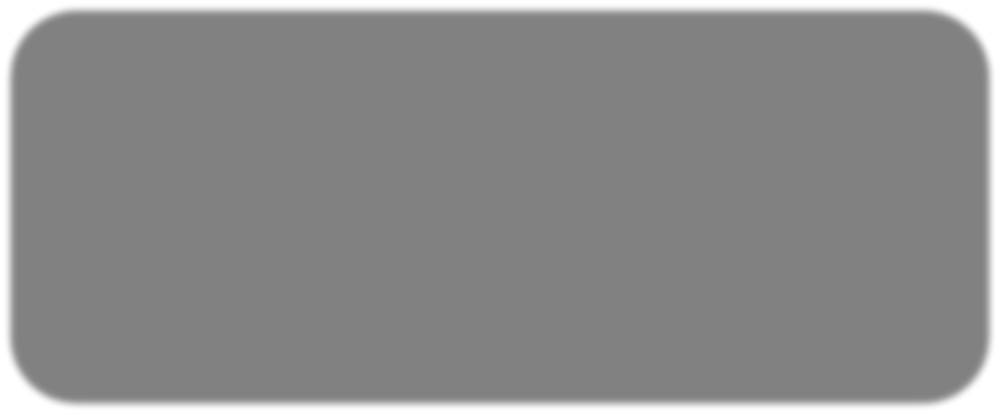
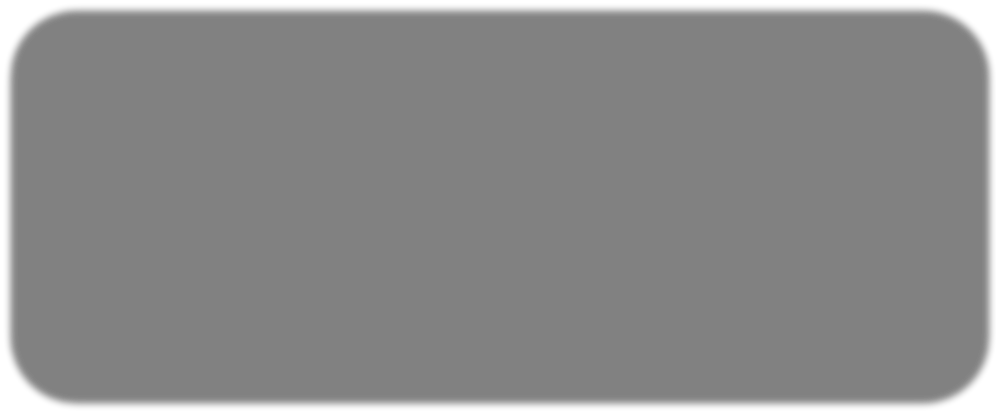
925

930

prises the scenarios that can be implemented using the

Smart City Model.

940



**Smart City Scenarios**

**InterSCSimulator**

**Pedestrians**

**Vehicles**

**Buses**

**Parking Spots**

**City Map**

**Sensors**

**Deployment Manager**

**Sim-Diasca**

**Load Balancer**

**Result Manager**

**Time Manager**

**Random Manager**

**Base Actor Model**

945

950

955

Figure 6: InterSCSimulator Architecture

InterSCSimulator has two main components: **Scenario**960 **Definition**, which receives the input files and creates the simulation scenario, and **Simulation Engine**, which exe- cutes the algorithms and models and generates the simula- tion output. For mobility simulations, it receives three re- quired files as input: **map.xml**, with the city street graph,965 **trips.xml**, describing all the trips that must be simulated, and **config.xml**, with general options such as the total simulation time. There are other optional files depending on the simulated scenario such as **park.xml**, containing all the parking spots and **subway.xml** with the subway970 network graph of the city. At the end of a simulation, the simulator generates an output file with all the events that occurred during the simulation. Figure [7](#_bookmark23) presents the In- terSCSimulator components and their inputs and outputs.

975

the existing mobility models in addition to new parking spot actors. This scenario considers several car drivers using a Smart Parking mobile application backed by the InterSCity platform to assist in the difficult task of find- ing a free parking spot in a large city like S˜ao Paulo. This application offers geolocated real-time information about parking spaces. When using the mobile app, a car sends its current location to the server application, which then an- swers with a list of the closest available parking spots. In this scenario, we simulated the behavior of drivers that use the mobile application, the behavior of the IoT devices in- stalled in parking spots, and the interactions among them. We extended the simulator with models to support the monitoring of parking spots and to allow drivers to find parking spots in the city. A parking spot actor generates events when a car parks on it or leaves it. It simulates the behavior of a real smart parking infrastructure sup- ported by cyber-physical systems to detect the presence of cars based on technologies that are commonly used in smart parking solutions, such as Wireless Sensor Networks (WSN), Light Dependable Resistor (LDR) sensors, Infra-

Red (IR) sensors, and magnetic sensors.

The simulation model consists of the following flow for a single car agent: (I) the car starts its trip from an origin to a destination point; (II) when the car is close to arriving at its destination, it requests the nearest free spots to the Smart Parking application; (III) the application asks the platform to find the nearest available spots considering the user’s location; (IV) with the data returned by the platform, the simulator changes the route of the car to the closest parking spot returned; (V) the driver arrives at the spot and finishes its trip; (VI) the simulator updates the status of the chosen parking spot on the InterSCity platform, marking it as unavailable.

If the platform does not find any parking spots that match the request of a car driver, the corresponding sim- ulation agent may return to step **II** increasing the search radius. After three failed attempts, the agent stops using the application and completes its execution in the simu- lation. Another special case might occur during step **V**,

980

985

990

995

1000

1005

1010

1015

1020

1025

as the target parking spot might become unavailable (e.g., by being taken by another vehicle), requiring the car to return to step **II**.

The interactions between the InterSCSimulator and1030 the InterSCity platform required the integration of these two systems. Such interactions impose workloads on the platform both at the top layer, as the simulator consumes data as a client application, and at the bottom layer, as it updates the status of the underlying IoT infrastructure of smart parking spaces. The details of this integration will1035 be presented in Section [4.3.](#_bookmark26)

To generate a meaningful workload to evaluate the platform, we modeled a realistic scenario based on real data from the very large city of S˜ao Paulo. This data was used as input to the simulator to define the number1040 and distribution of parking spots around the city and the trips undertaken by drivers, including their origins, desti- nations, and departure times. The data used to generate workloads is detailed bellow:

1045

**Origin-Destination (OD) Survey:** we created the simulated trips based on the OD survey per- formed by the subway company of S˜ao Paulo[15](#_bookmark27). This survey describes the trips of 200,000 people and ex- trapolates the data to the entire population of the1050 city. The survey includes information on the ori- gin, destination, transportation mode, and depar- ture time. We used this data to define the behavior of car agents. To generate the load for the platform experiments, we simulated the traffic in S˜ao Paulo1055 during peak hours, from 5:40 am to 8:40 am. In the OD survey, there are 492,976 cars that start their trips during the considered time interval.

*•*

**OpenStreet Maps:** to create the city graph used in the simulation, we used the map from OpenStreet1060 Maps. This map contains all the streets and junc- tions of the city, together with a number of attributes, such as length, capacity, and speed limit. Such infor- mation is used by the simulator to define the routes taken by cars as they perform their trips, as well as1065 to simulate the impact of traffic on the speed of cars.

*•*

**Parking Spots:** we created the simulated parking spots based on data from OpenStreet Maps and from *Zona Azul* [16](#_bookmark28) (the rotary parking service of the city1070 of S˜ao Paulo).

*•*

To show the distribution of parking spaces and drivers destinations throughout the city, Figure [8](#_bookmark29) presents two heat maps: (a) the simulated distribution of parking spots1075 across the city; and (b) the distribution of trip destina- tions throughout the entire simulation. It is worth noting that parking spots with IoT infrastructure are significantly

more concentrated than trip destinations. This may lead to situations where drivers do not find available parking spots after three attempts. In such cases, the user agent stops using the application and finishes its execution.

* 1. *Simulator and Platform Integration*

Enabling the scalability evaluation on the InterSCity platform using InterSCSimulator requires an integration of the two systems. To fully explore the dynamics of the Smart Parking scenario on the workload generation, the simulator needs to produce workloads for both the sta- tus updates of parking spots (the IoT infrastructure) and the requests performed by drivers to find available parking spaces (using the client application).

For this integration, we implemented a two-way com- munication interface between the simulator and the plat- form. In one direction, the simulator updates the status of the *parking spots* in the platform based on the simu- lated IoT devices. This is carried out directly via the plat- form’s RabbitMQ instance, by publishing update data on the **data stream** topic. In the other direction, the simu- lated users of the application perform synchronous HTTP requests to the platform. To avoid blocking the simula- tor while clients wait for a reply from the platform (and, therefore, distorting the simulated passage of time), we im- plemented an intermediate agent that runs independently of the simulation cycle to perform these HTTP requests and wait for the responses.

Figure [9](#_bookmark30) details, in the context of the Smart Parking scenario, the integration of InterSCity and InterSCSimula- tor in terms of the interactions between their components. When a car (represented by its simulation agent) is close to arriving at its destination, it asks a controller agent for a nearby available parking space. Note that there are mul- tiple controller agents, each one managing a set of parking spots in the simulation. As we do not want to halt the simulation waiting for the HTTP response from the plat- form, the controller sends a message to an external Erlang agent asking for an available spot nearby the latitude and longitude of the car’s destination. This discovery agent is responsible for actually making the HTTP request to the platform. The agent keeps blocked until it receives the re- ply. After that, the identifier of the suitable parking spot is sent back to the controller inside the simulator. The controller marks this spot as unavailable and passes its identifier to the car. The car retraces its route to park in the right place. This workflow simulates a single driver us- ing the Smart Parking application. When a parking spot becomes available or occupied, the controller agent sends an AMQP message to the platform, via the RabbitMQ instance, informing the event to the platform.

Finally, we added an idle agent to the simulator which sleeps for a time very close to one second to ensure that the simulator operates in real time (i.e., at the same rate

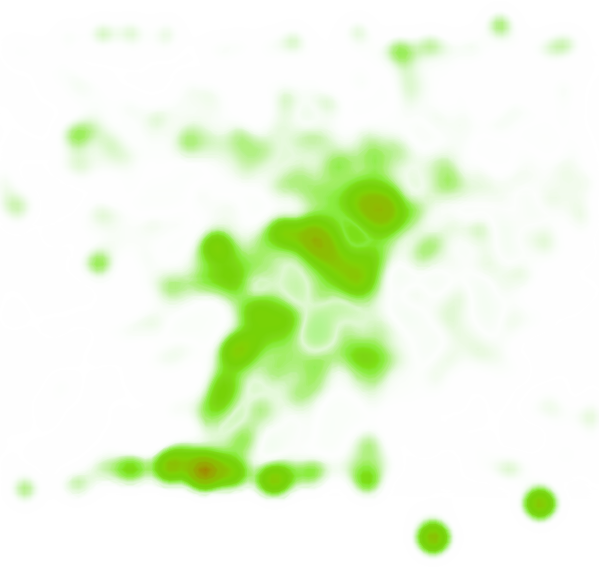
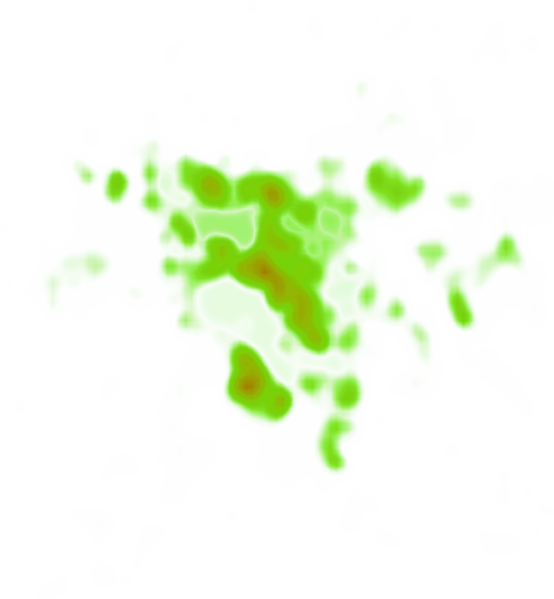
15Origin-Destination Survey - <http://goo.gl/Te2SX7>

16[http://www.cetsp.com.br/consultas/zona-azul/](http://www.cetsp.com.br/consultas/zona-azul/mapa-zona-azul/mapa-zona-azul.aspx)

[mapa-zona-azul/mapa-zona-azul.aspx](http://www.cetsp.com.br/consultas/zona-azul/mapa-zona-azul/mapa-zona-azul.aspx)

1080

as the actual physical system). Each simulation step (cy- cle) is a simulated second of the day in which all agents are given the opportunity to perform some action. In its



* + 1. Parking spots distribution (b) Destinations distribution

Figure 8: Heat maps of the distributions of parking spots and car trip destinations in the City of S˜ao Paulo

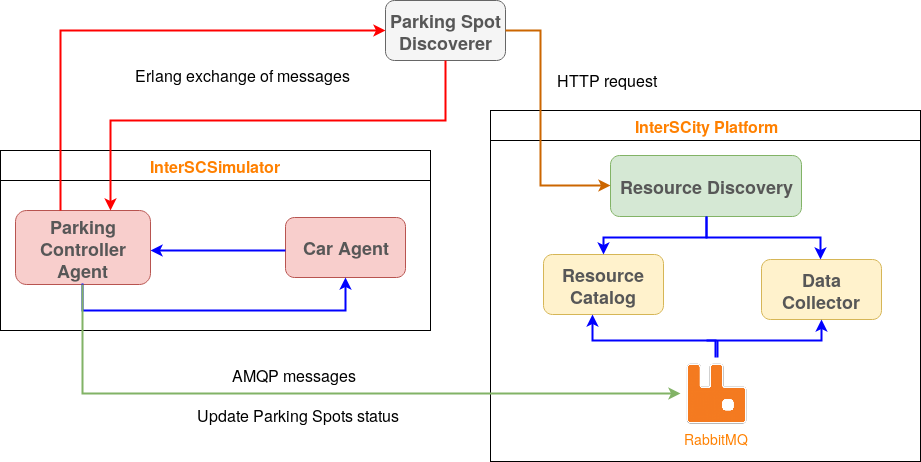


Figure 9: The 2-way integration of the InterSCity platform and InterSCSimulator

1085

1090

normal operation mode, the simulator runs as fast as pos- sible, simulating multiple cycles under one real second. In many cases, this is interesting because the simulation re-1095 sults (which may take hours) come faster. However, in our case, the goal is to produce a realistic load to evaluate the platform, so we want to process as close to real time as possible.

1100

* 1. *Simulating Other Scenarios*

Note that other scenarios could be simulated by defin- ing a proper model of the city dynamics and creating asso-

ciated abstractions in the InterSCSimulator. For example, one could consider extending the platform to deal with a Smart Grid scenario in which each home or building in the city could behave both as energy consumer and producer. In this case, one should program new InterSCSimulator agents [[9]](#_bookmark52) to model, for instance, the energy consumption behavior of families in a residence, workers in an office, and machines in a factory, as well as a simplified behavior of wind turbines and solar panels (e.g., modeling how much energy they produce). The simulator could then receive as input (1) a map of the city’s energy distribution grid,

1105

1110

1115

1120

1125

1130

1135

1140

1145

1150

(2) historical data about the level of solar radiation and1155 wind speed, and (3) a list of buildings and homes capable of producing electricity from solar and wind energy.

With this setting, the simulator would be able to gen- erate a realistic workload, consisting of events related to energy consumption and energy production, generated by1160 the different simulation agents. This workload could then be used to assess, for example, a Smart Grid monitoring and accounting application built on top of the smart city platform.

1165

# Experimental Evaluation and Analysis

Using the integrated environment described in Section [4,](#_bookmark20) we performed a comprehensive experiment to assess the scalability properties of the InterSCity platform. The ex-1170 periment consisted of: (I) running a production-like in- stance of the platform in a cloud environment; (II) en- abling an auto-scaling mechanism for the platform’s mi- croservices based on the variation of the workload; (III) set- ting up the simulator in an isolated environment; (IV) start11-75 ing the simulation of the Smart Parking scenario; (V) mon- itoring the platform’s performance and its usage of the computational resources during the entire simulation; and

(VI) analyzing the obtained results.

Section [5.1](#_bookmark32) details the execution environment of the experiments. In Section [5.2](#_bookmark38) we discuss the results from a set of repeated rounds of the experiment on the InterSCity platform.

* 1. *InterSCity Configuration*

To conduct the experiments, a production-like instance of the InterSCity platform pre-populated with the resources available in the city (parking spots) and their initial states (available) is required. Only the microservices Resource Catalog, Resource Discovery, and Data Collector are used in the Smart Parking scenario. The Resource Adaptor is

1180

not used since all communication with sensors happens

through RabbitMQ (as detailed in Section [4.3),](#_bookmark26) whereas the Actuator Controller is not required as the scenario does not include actuators.

We used Docker containers for InterSCity microservices

1185

and supporting services. In all experiments we used the

Google Cloud Platform[17](#_bookmark33) (GCloud), which is an appropri- ate infrastructure to run InterSCity in a production envi- ronment as the platform is a cloud-native system. More specifically, we used the Kubernetes Engine, a set of tools

1190

provided by GCloud based on Kubernetes, to schedule

the deployment of containers throughout a GCloud clus- ter. Among other tasks, we used Kubernetes to also au- tomate the restarting, replication, and scaling of contain- ers. Thus, Kubernetes increases the reproducibility of our

1195

experiments by ensuring the correct application of deploy-

ment rules, configuration, and state. All the code used to

17<https://cloud.google.com/>

perform the experiments, as well as the Kubernetes con- figuration files, are publicly available in an online reposi- tory[18](#_bookmark35).

We divided the cluster into 5 different node pools so that Kubernetes could schedule the containers to the ap- propriate pools. Figure [10](#_bookmark34) presents these node pools and the number and type of the machines used by each one on GCloud[19](#_bookmark36). The Platform Pool comprises 25 machines of type *n1-standard-2* (2 virtual CPUs and 7.5GB of mem- ory) and runs both InterSCity microservices and Kong. There are three additional node pools composed of *n1- highmem-2* machines (2 virtual CPUs and 13GB of mem- ory), which execute the support services of the InterSC- ity environment. Both MongoDB and PostgreSQL pools have 5 nodes running distributed, fault-tolerant instances of their respective database systems, whereas RabbitMQ has a dedicated machine in an isolated node. MongoDB is deployed using the replica set strategy[20](#_bookmark37) to distribute *read* operations among secondary nodes (slaves), although *write* operations are always performed on the primary node (mas- ter). The same strategy is adopted for the PostgreSQL instance for optimizing the *read* operations performed by the Resource Catalog. Finally, the InterSCSimulator runs on its own *n1-highmem-16* machine (16 virtual CPUs and 104GB of memory), isolated from the rest of the services.

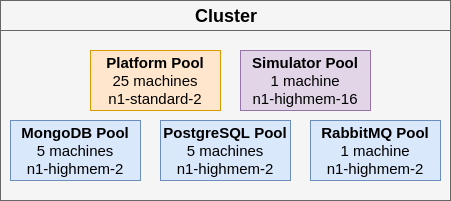


Figure 10: Cluster Node Pools for the experiments

For the Platform Pool, Kubernetes may schedule sev- eral containers for the same host depending on the avail- able computational resources. The distribution of contain- ers across the 25 nodes may differ from one experiment to another and is a variable that we did not control for dur- ing the performed experiments. To evaluate the impact of such variations on the analysis, we repeated the exper- iment 15 times and studied the variability of the results.

As we were interested in assessing the platform scal- ability considering a smart city scenario with a varying workload, we used automatic scaling for the Resource Cat- alog, Resource Discovery, Data Collector, and Kong services, as they are designed to scale horizontally. For this pur- pose, we specified a target of 60% of CPU usage for each of those services, enabling the system to increase or de- crease the number of containers per service. The system

18<https://github.com/LSS-USP/interscity-k8s-experiment> 19<https://cloud.google.com/compute/docs/machine-types> 20[https://docs.mongodb.com/manual/tutorial/](https://docs.mongodb.com/manual/tutorial/deploy-replica-set/)

[deploy-replica-set/](https://docs.mongodb.com/manual/tutorial/deploy-replica-set/)

1200

1205

1210

1215

1220

1225

1230

1235

1240

1245

1250

balances the workload to match the target CPU usage con- sidering the average CPU usage of the running containers, which is measured every 30 seconds. Initially, each service has four containers, which is set as the minimum number of running containers. This number may increase as long as computational resources are available in the Platform node pool. We run the containers behind a load balancing service.

Although we could benefit from GCloud’s elasticity properties by automatically adding and removing nodes to our cluster using its auto-scaling feature, this would in- troduce another level of uncertainty in our experiments, since in our experience the time taken to create new VMs may vary considerably. Thus, we created all the nodes in advance, before starting the experiments, keeping them running throughout the experiment.

* 1. *Scalability Evaluation*

To better analyze the behavior of the InterSCity plat- form, we ran multiple rounds of the experiment. Our ob- jective was to minimize the effects of uncontrollable vari- ables inherent to the environment in which the tests were performed, so as to evaluate important aspects of the sys- tem and to ensure that the observed results are good esti- mates for the general behavior of the system.

We performed a total of 15 rounds of the experiment. Each round ran for 3 hours, corresponding to the simu- lation of the morning peak traffic hours in S˜ao Paulo ac- cording to the scenario described in Section [4.2.](#_bookmark24) Figure [11](#_bookmark39) represents the average workload generated by InterSCSim- ulator on the platform during the experiment and the stan- dard deviation (black lines on top of each bar). It is worth noting the constant increase in the workload during the first 80 minutes. In the approximate interval of one hour between 60 and 120 minutes, we observed the highest load period of the experiment, whereas the maximum peak of requests happens after 80 minutes, corresponding to more than 113,000 requests in 10 minutes. In total, more than one million requests were performed to the platform during the experiment time. Since fulfilling each of these requests requires a complex set of operations with multiple internal steps, this translates to a very large computational load.

Figure [12](#_bookmark40) shows the dynamic creation and destruction of InterSCity containers due to the application of the auto- scaling strategy in a single round. The initial replication of Kong instances was enough to support the entire work- load during the entire experiment since it only performs the low-latency task of forwarding the incoming requests to the proper microservices. In turn, the three InterSCity microservices, which are truly responsible for handling the requests, were replicated according to the increasing work- load. Thus, the number of containers for these services varied from 4 to 25. It is important to mention that the InterSCity elasticity mechanism also reduced the number

1255

of containers as the demand decreased. As can be seen in

Figure [12,](#_bookmark40) among the InterSCity microservices, Data Col-

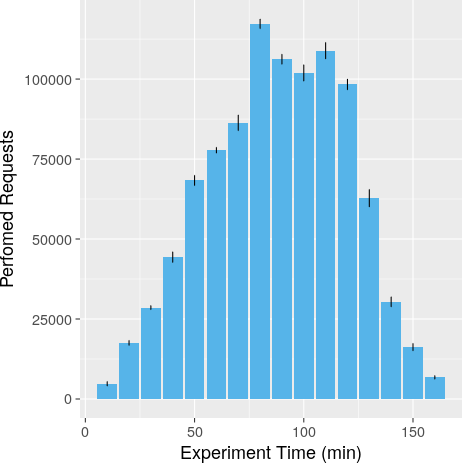


Figure 11: Average workload generated by the InterSCSimulator

lector was the microservice that consumed the least CPU time.

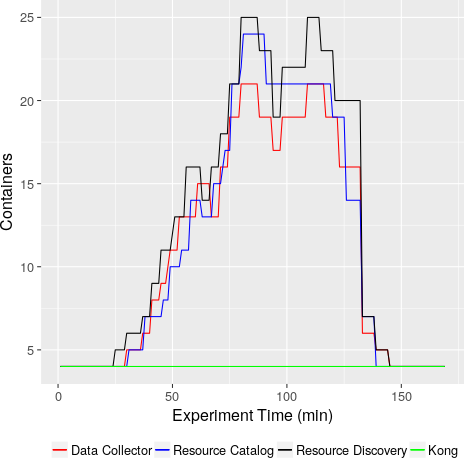


Figure 12: InterSCity services autoscaling

Figure [13](#_bookmark41) shows the average throughput of the Inter- SCity platform over the duration of the experiment. The throughput is defined as the rate of successful responses received by the simulator. The result indicates that the throughput closely matches the generated workload, as

1260

1265

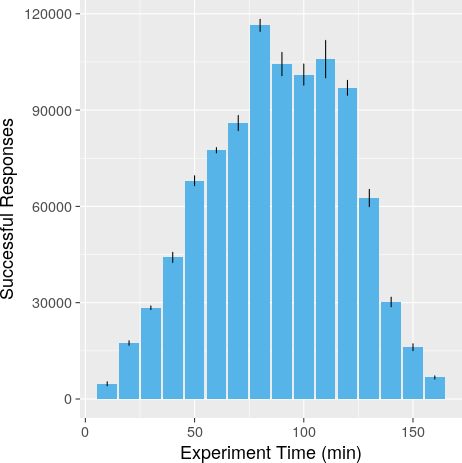
1270

1275

1280

1285

can be seen by comparing Figures [11](#_bookmark39) and [13.](#_bookmark41) Despite the1290 simulated variations in the drivers’ behavior throughout the experiment, the platform was able to handle the vary- ing demand thanks to its scalability and autoscaling sup- port features, described in Section [3.](#_bookmark4) However, we should mention that the throughput did not match the generated workload exactly, since some of the requests failed, rep- resenting nearly 0.6% of all requests on average. Failed requests include those that had responses with an HTTP error code, as well as those that were not completed due to connection refusal or timeout. But we consider that be- ing able to handle over 99.5% of the requests under high load is satisfactory; a typical user would see a failure every 200 requests, which is very good for this kind of real-time smart urban application.

1295

1300

Figure 13: Average InterSCity throughput

Another fundamental aspect of the system assessment is to analyze the performance of the platform to handle application requests with a varying workload. In this re- spect, we are mainly interested in analyzing the perfor- mance degradation and verifying whether the platform is scaling appropriately to serve its clients within acceptable response times. For this purpose, we collected the response time from the client’s point of view, as shown by Figure [14.](#_bookmark42) During most of the experiment duration, the platform was able to respond in less than one second. However, differ- ent from the throughput, the impact of the highest de- mand period on the observed response time is noticeable since, during a small time interval (after 110 minutes of execution), the average response time was greater than 1 second. The response time went back down to 500 mil- liseconds after that. But, we can see that even in periods of high-load, the response time was kept under 2s, which

1305

1310

1315

is a very good result for this kind of application.

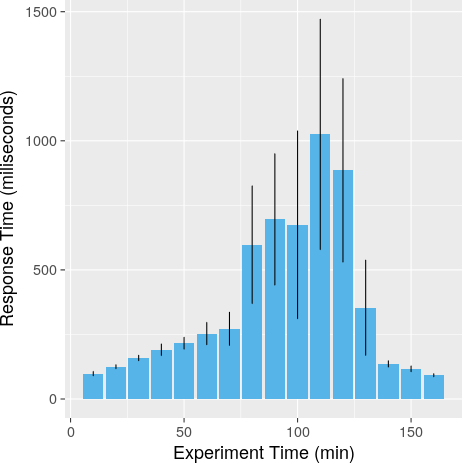


Figure 14: InterSCity Average Response Time

We should recall that the distribution of containers on the available nodes may impact the response time, as sev- eral containers may compete for computational resources if they are running on the same host machine. Moreover, although the system handles the autoscaling task every 30 seconds, we have no control over the time it takes for a container to be scheduled and become ready to receive requests. On the other hand, this distribution may also introduce a beneficial effect due to the scheduling of ser- vices that constantly interact with each other to the same machine, reducing network latency and unpredictability.

* 1. *Threats to Validity*

In this section, we clarify and discuss some important aspects of our experiment design as they may threaten the validity of our results and affect reproducibility. Most of the threats are inherent to the environment where the experiments were carried out, as there is considerable un- certainty concerning the provisioning of resources and ser- vices in cloud environments [[36].](#_bookmark79) As the Google Cloud Platform is not a controlled environment, it is impossible to have complete knowledge of the system. For instance, we cannot correctly specify all properties related to the communication network used within the cluster, such as network capacity, nor ensure constant bandwidth. Such variables may directly impact results such as the observed response time.

As discussed in Section [5.2,](#_bookmark38) although Kubernetes plays a crucial role in our experiments, it also raises some con- cerns for their validation. Firstly, we do not control the

1320

1325

1330

1335

1340

1345

1350

1355

1360

1365

1370

way Kubernetes distributes new containers across the avail1-375 able hosts of a node pool, except for the definition of the minimum computing resource requirements for executing

a specific container. As a consequence, on each round of the experiment, Kubernetes may distribute microservices differently, which may lead to a different load distribution1380 among the hosts in use. For instance, consider the case of

a new container that has the minimum requirement of 30% of available CPU and is allocated on a host whose CPU usage is already at 60%. The container would probably have less CPU time than if it had been allocated on a host1385 with less competition for resources or, at least, manifest higher latencies. Another important aspect regarding Ku- bernetes is the variation in the time spent for scheduling new containers, especially in the auto-scaling task. In pe- riods of increasing workload, the delay in the allocation of1390 new containers may directly impact the number of failed requests and the response time.

Finally, since we advocate for the use of more appro- priate scenarios to test smart city platforms, it is worth mentioning the aspects related to the realism of the smart1395 city simulations performed as part of the experiments. The conformance of the simulations with real-world fu- ture smart cities scenarios depends on good models. In this sense, the refinement of the models used to simulate car trips, as well as the behavior of drivers in search for1400 parking spots in a large city might change the workload generated on the platform. Moreover, in addition to Smart Parking, it is essential to evaluate the platform using other smart city scenarios, which may pose different demands and require others functionalities, such as actuation on1405 city resources.

# Conclusions and Future Work

Appropriate software platforms are critical to the fea-1410 sibility and widespread adoption of complex smart cyber- physical environments, especially future smart cities, re- quiring considerable development and research effort to ensure their technical quality. Despite the existence of numerous smart city projects, developing architectures to support city-scale environments and evaluating them is

1415

still challenging. Thus, this paper brought a comprehen-

sive discussion on the InterSCity platform microservices architectural design and presented experimental results to demonstrate its benefits on scalability and performance.

1420

For this purpose, we introduced a novel approach that uses

a smart city simulator to generate realistic workloads, sim- ilar to what one would expect in a metropolis with over 10 million inhabitants such as S˜ao Paulo. Such workloads

1425

are essential to enable the execution of more appropriate

experiments and tests.

With respect to the InterSCity platform, the exper- iment showed that its architecture is capable of scaling up and down horizontally to handle a varying workload. More precisely, it was able to handle more than one mil- lion complex requests during approximately 3 hours, con-

1430

sidering a Smart Parking scenario during S˜ao Paulo’s rush hour. Also, the platform response time remained accept- able, mostly 1 second or below, even at the highest demand time interval. We highlight the use of modern tools such as Kubernetes and Docker containers as essential means to achieve these results.

We presented a mechanism to evaluate the scalability and performance of the InterSCity platform via the in- tegration of the platform and the InterSCSimulator. To achieve a more representative workload for assessing the platform, we extended the InterSCSimulator models with real data gathered from a large metropolis. The simulator managed both individual behavior and interactions among IoT devices and the platform users considering a Smart Parking scenario. This approach contributed towards the characterization of more appropriate workloads for smart cities.

As future work, we want to integrate other smart city scenarios, enabling cross-domain experiments. We also in- tend to devote efforts to decouple the presented integration mechanism to allow experiments and tests of smart city platforms other than InterSCity with the same simulation- based approach.

There are still several open research challenges regard- ing smart cyber-phisical platforms applied to smart urban spaces, including: (I) meeting other critical non-functional requirements, such as security and privacy; (II) extending the core functionalities to better support application devel- opment; and (III) deploying a production instance of the platform on a real city. Also, we intend to perform further experiments to continue exploring the scalability and per- formance properties of the platform, mostly by considering other smart cities scenarios and features. Finally, based on our open source and open science approach, we encourage the community to leverage the contributions described in this paper and to contribute to the evolution of the field.

# References

[1]PCAST, T[echnology and the future of cities, report to the](https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf) [president,](https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf) Tech. rep., Executive Office of the President, United States (February 2016).

URL [https://www.whitehouse.gov/sites/whitehouse.gov/](https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf) [files/images/Blog/PCAST%20Cities%20Report%20\_%20FINAL.](https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf) [pdf](https://www.whitehouse.gov/sites/whitehouse.gov/files/images/Blog/PCAST%20Cities%20Report%20_%20FINAL.pdf)

[2]F. J. Villanueva, M. J. Santofimia, D. Villa, J. Barba, J. C. Lpez, Civitas: The smart city middleware, from sensors to big data, in: 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013, pp. 445–450. [doi:10.1109/IMIS.2013.80](http://dx.doi.org/10.1109/IMIS.2013.80).

[3]E. F. Z. Santana, A. P. Chaves, M. A. Gerosa, F. Kon, D. S. Milojicic, Software Platforms for Smart Cities: Concepts, Re- quirements, Challenges, and a Unified Reference Architecture, ACM Computing Surveys 50 (6) (2017) 78:1–78:37.

[4]R. Morabito, R. Petrolo, V. Loscr, N. Mitton, [Legiot: A](http://www.sciencedirect.com/science/article/pii/S0167739X17306593) [lightweight edge gateway for the internet of things,](http://www.sciencedirect.com/science/article/pii/S0167739X17306593) Future Generation Computer Systems 81 (2018) 1 – 15. doi:https:

//doi.org/10.1016/j.future.2017.10.011.

URL [http://www.sciencedirect.com/science/article/pii/](http://www.sciencedirect.com/science/article/pii/S0167739X17306593) [S0167739X17306593](http://www.sciencedirect.com/science/article/pii/S0167739X17306593)

1435

1440

1445

1450

1455

[5]L. Sanchez, J. A. Galache, V. Gutierrez, J. M. Hernandez,

J. Bernat, A. Gluhak, T. Garcia, Smartsantander: The meet-1505 ing point between future internet research and experimentation and the smart cities, in: 2011 Future Network Mobile Summit, 2011, pp. 1–8.

[6]A. M. D. Esposte, F. Kon, F. M. Costa, N. Lago, InterSCity:

A Scalable Microservice-based Open Source Platform for Smart1510 Cities, in: Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems, 2017, pp. 35–46.

[7]C. Pereira, J. Cardoso, A. Aguiar, R. Morla, [Benchmarking](https://doi.org/10.1007/s40860-018-0056-3) [pub/sub iot middleware platforms for smart services,](https://doi.org/10.1007/s40860-018-0056-3) Journal of Reliable Intelligent Environments 4 (1) (2018) 25–37. [doi:](http://dx.doi.org/10.1007/s40860-018-0056-3)1515 [10.1007/s40860-018-0056-3](http://dx.doi.org/10.1007/s40860-018-0056-3).

URL <https://doi.org/10.1007/s40860-018-0056-3>

[8]D. Preuveneers, Y. Berbers, Samurai: A streaming multi-tenant context-management architecture for intelligent and scalable in- ternet of things applications, in: 2014 International Confer-1520 ence on Intelligent Environments, 2014, pp. 226–233. [doi:](http://dx.doi.org/10.1109/IE.2014.43) [10.1109/IE.2014.43](http://dx.doi.org/10.1109/IE.2014.43).

[9]E. F. Z. Santana, N. Lago, F. Kon, D. S. Milojicic, InterSC- Simulator: Large-Scale Traffic Simulation in Smart Cities using Erlang, in: 18th Workshop on Multi-agent-based Simulation1,525 2017.

[10]L. Sanchez, L. Mun˜oz, J. A. Galache, P. Sotres, J. R. Santana,

V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodor- idis, et al., Smartsantander: Iot experimentation over a smart

Nejad, K.-H. Wolf, M. Dragone, J. Ye, C. R¨ocker, G. O’Hare (Eds.), Evolving Ambient Intelligence, Springer International Publishing, Cham, 2013, pp. 308–312.

[22]A. Shukla, S. Chaturvedi, Y. Simmhan, Ri[otbench: A real-time](http://arxiv.org/abs/1701.08530) [IoT benchmark for distributed stream processing platforms,](http://arxiv.org/abs/1701.08530) CoRR abs/1701.08530. [arXiv:1701.08530](http://arxiv.org/abs/1701.08530).

URL <http://arxiv.org/abs/1701.08530>

[23]M. I. Ali, F. Gao, A. Mileo, Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets, in: M. Arenas, O. Corcho, E. Simperl, M. Strohmaier,

M. d’Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin,

K. Thirunarayan, S. Staab (Eds.), The Semantic Web - ISWC 2015, Springer International Publishing, Cham, 2015, pp. 374– 389.

[24]I. Zyrianoff, F. Borelli, C. Kamienski, Sense–sensor simulation environment: Uma ferramenta para gerac¸˜ao de tr´afego IoT em larga escala, Anais do Sal˜ao de Ferramentas do SBRC (2017) 1134–1141.

[25]D. Mosberger, T. Jin, [Httperf&mdash;a tool for measuring web](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/306225.306235) [server performance,](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/306225.306235) SIGMETRICS Perform. Eval. Rev. 26 (3) (1998) 31–37. [doi:10.1145/306225.306235](http://dx.doi.org/10.1145/306225.306235).

URL [http://doi-acm-org.ez67.periodicos.capes.gov.br/](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/306225.306235) [10.1145/306225.306235](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/306225.306235)

[26]M. C. Calzarossa, L. Massari, D. Tessera, W[orkload character-](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/2856127) [ization: A survey revisited,](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/2856127) ACM Comput. Surv. 48 (3) (2016) 48:1–48:43. [doi:10.1145/2856127](http://dx.doi.org/10.1145/2856127).

1460

city testbed, Computer Networks 61 (2014) 217–238.

[11]B. Cheng, S. Longo, F. Cirillo, M. Bauer, E. Kovacs, Building

1530

URL [http://doi-acm-org.ez67.periodicos.capes.gov.br/](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/2856127) [10.1145/2856127](http://doi-acm-org.ez67.periodicos.capes.gov.br/10.1145/2856127)

1465

1470

1475

a big data platform for smart cities: Experience and lessons from santander, in: Big Data (BigData Congress), 2015 IEEE International Congress on, IEEE, 2015, pp. 592–599.

[12]S. Yamamoto, S. Matsumoto, S. Saiki, M. Nakamura, Using 1535 materialized view as a service of scallop4sc for smart city ap- plication services, in: Soft Computing in Big Data Processing, Springer, 2014, pp. 51–60.

[13]S. Girtelschmid, M. Steinbauer, V. Kumar, A. Fensel, G. Kot- sis, Big data in large scale intelligent smart city installations1,540 in: Proceedings of International Conference on Information In- tegration and Web-based Applications & Services, ACM, 2013, pp. 428–432.

[14]A. Krylovskiy, M. Jahn, E. Patti, Designing a smart city inter- net of things platform with microservice architecture, in: Future1545 Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on, IEEE, 2015, pp. 25–30.

[15]R. Petrolo, V. Loscri, N. Mitton, Towards a cloud of things smart city, IEEE COMSOC MMTC E-Letter 9 (5) (2014) 44–

[27]D. M. Batista, A. Goldman, R. Hirata Jr., F. Kon, F. M. Costa,

M. Endler, InterSCity: Addressing Future Internet Research Challenges for Smart Cities, in: 7th International Conference on the Network of the Future, IEEE, 2016.

[28]N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara,

F. Montesi, R. Mustafin, L. Safina, [Microservices: yesterday,](http://arxiv.org/abs/1606.04036) [today, and tomorrow,](http://arxiv.org/abs/1606.04036) CoRR abs/1606.04036.

URL <http://arxiv.org/abs/1606.04036>

[29]S. Newman, Building Microservices, O’Reilly Media, 2015. [30]A. Meslin, N. Rodriguez, M. Endler, A Scalable Multilayer Mid-

dleware for Distributed Monitoring and Complex Event Pro- cessing for Smart Cities, in: 4th IEEE International Smart Cities Conference, IEEE, 2018.

[31]A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices archi- tecture enables devops: Migration to a cloud-native architec- ture, IEEE Software 33 (3) (2016) 42–52. [doi:10.1109/MS.](http://dx.doi.org/10.1109/MS.2016.64)

[2016.64](http://dx.doi.org/10.1109/MS.2016.64).

[32]G. Hohpe, B. Woolf, Enterprise Integration Patterns: Design-

1480

48.

[16]R. L. Pereira, P. C. Sousa, R. Barata, A. Oliveira, G. Monsieur,

1550

ing, Building, and Deploying Messaging Solutions, Addison- Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

1485

1490

1495

1500

Citysdk tourism api-building value around open data, Journal of Internet Services and Applications 6 (1) (2015) 24.

[17]D. Puiu, P. Barnaghi, R. T¨onjes, D. Ku¨mper, M. I. Ali,

A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. Faraji-1555 davar, et al., Citypulse: Large scale data analytics framework for smart cities, IEEE Access 4 (2016) 1086–1108.

[18]S. D. Cola, C. Tran, K.-K. Lau, A. Celesti, M. Fazio, A het- erogeneous approach for developing applications with fiware, in: Service Oriented and Cloud Computing. Lecture Notes in Com-1560 puter Science, Vol. 9306, Springer, Cham, 2015. doi:https:

//doi.org/10.1007/978-3-319-24072-5\_5.

[19]P. Salhofer, Ev[aluating the fiware platform: A case-study on](http://hdl.handle.net/10125/50615) [implementing smart application with fiware,](http://hdl.handle.net/10125/50615) in: Proceedings of the 51st Hawaii International Conference on System Sciences1,565 2018, pp. 5797–5805. [doi:10.24251/HICSS.2018.726](http://dx.doi.org/10.24251/HICSS.2018.726).

URL <http://hdl.handle.net/10125/50615>

[20]C. Steinmetz, G. Schroeder, A. dos Santos Roque, C. E. Pereira,

C. Wagner, P. Saalmann, B. Hellingrath, Ontology-driven iot code generation for fiware, in: 2017 IEEE 15th Internationa1l570 Conference on Industrial Informatics (INDIN), 2017, pp. 38– 43. [doi:10.1109/INDIN.2017.8104743](http://dx.doi.org/10.1109/INDIN.2017.8104743).

[21]A. Bellabas, F. Ramparany, M. Arndt, Fiware infrastructure for smart home applications, in: M. J. O’Grady, H. Vahdat-

[33]A. M. Joy, Performance comparison between linux containers and virtual machines, in: 2015 International Conference on Ad- vances in Computer Engineering and Applications, 2015, pp. 342–346. [doi:10.1109/ICACEA.2015.7164727](http://dx.doi.org/10.1109/ICACEA.2015.7164727).

[34]W. Hummer, F. Rosenberg, F. Oliveira, T. Eilam, Testing idem- potence for infrastructure as code, in: D. Eyers, K. Schwan (Eds.), Middleware 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 368–388.

[35]T. Song, D. Kaleshi, R. Zhou, O. Boudeville, J.-X. Ma, A. Pel- letier, I. Haddadi, Performance evaluation of integrated smart energy solutions through large-scale simulations, in: Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on, 2011, pp. 37–42.

[36]A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, E. ghaz- ali Talbi, [Towards understanding uncertainty in cloud com-](http://www.sciencedirect.com/science/article/pii/S1877050915011953) [puting resource provisioning,](http://www.sciencedirect.com/science/article/pii/S1877050915011953) Procedia Computer Science 51 (2015) 1772 – 1781, international Conference On Computational Science, ICCS 2015. doi:https://doi.org/10.1016/j.procs. 2015.05.387.

URL [http://www.sciencedirect.com/science/article/pii/](http://www.sciencedirect.com/science/article/pii/S1877050915011953) [S1877050915011953](http://www.sciencedirect.com/science/article/pii/S1877050915011953)