HORIZON2020 FRAMEWORK PROGRAMME

TOPIC EUK-03-2016

"Federated Cloud resource brokerage for mobile cloud services"

D6.4
TripBuilder Use Case

Project acronym: BASMATI

Project full title: *Cloud Brokerage Across Borders for Mobile Users and Applications*

Contract no.: 723131

Workpackage:	6	Use Cases Definition, Experimentation and Validation	
Editor:	M.Coppola		CNR
Author(s):	M.Coppola		CNR
	Patrizio Dazzi		CNR
	Emanuele Carlini		CNR
Authorized by	Konstantinos Tserpes Young-Woo Jung		ICCS ETRI
Doc Ref:	D6.4		
Reviewer	K. Tserpes		ICCS
Dissemination Level	Public		

Document History

Version	Date	Changes	Author/Affiliation
v.0.10	10-09-2018	Collected contributions into standard template	M. Coppola, P. Dazzi, E. Carlini, V. Monteiro De Lira, F.M. Nardini, CNR
v.1.00	16-09-2018	Revised final version	K. Tserpes/ICCS

BASMATI Glossary

Term/Acronym	Definition
Mobile cloud services	Online services offered by cloud resources to support mobile apps. The backend of the mobile apps.
CP	Cloud Provider. The actor that provides the cloud infrastructure/resources, such as VMs
CSP	Cloud Service Provider. The actor that provides cloud services on top of a rent infrastructure from a CP
Cloudlet	Limited capacity infrastructures with virtualization capabilities, often used to support a limited amount of users or perform a limited set of operations on behalf of the central cloud infrastructure that hosts the complete application
Edge resources	Resources aimed to operate specialized functionality, located at the "edge" of the network infrastructure, thus, closer to the end users. Examples are (clusters of) RaspberryPis or cloudlets
BUDaMaF	BASMATI Unified Data Management Framework
KE	Knowledge Extractor
DM	Decision Maker
RB	Resource Broker
MVD	Mobile Virtual Desktop
DAS FEST	An 3-day long music festival taking place in Karlsruhe, Germany every July
ACE	Amenesik Cloud Engine. The cloud service deployment tool through which actual federation is achieved
BEAM	BASMATI Enhanced Application Model. An extension of the TOSCA specification
ASP	Application Service Provider. A Federation user that rents resource services in order to provide an Application services to End-users
Brokering	The matchmaking support provided by BASMATI platform to decide about the best cloud resources to exploit for the execution of the back-end of BASMATI applications. This activity regards the placement of the services or data on computational resources and storages belonging to the cloud data centre and the cloudlets within the federation.
End user	A user who benefits the various application and infrastructure services provided

	by the Cloud. Within BASMATI, the most typical example is exploiting the Cloud federation via a mobile device (possibly a laptop) using specialized apps or a web browser.
Federated user	A Federated User is any user whose identity is granted and recognized by the BASMATI identity services. These in turn may rely on identity providers federated within the BASMATI platform, but technical implications are not pertinent here. For what concerns BASMATI, a user whose identity is not tied to any BASMATI recognized identity service can be regarded as an Anonymous user.
Cloud user	The Cloud User is the owner of an application, that is, he/she controls an application that provides a service at the PaaS or SaaS level to a set of end-users. A Cloud user is a federated user. A Cloud user exploits the BASMATI platform to run its application service backend on IaaS resources from the federation. The service provider will use one or more of the cloud providers within the BASMATI federation. The service provider submits a description of its application (intended as a collection of services, with their functional specification) and a QoS that elaborate the non-functional specifications of the application.
Offloading	The ability of BASMATI platform supporting the runtime placement of the components composing the front-end of BASMATI applications on edge resources available nearby the end user. This activity takes place both when edge and mobiles exchange one each other their own workload or when such devices transfer some workload to the clouds or cloudlets. In BASMATI we often distinguish Front-end offloading, related to the mobile part of application, from Back-end offloading, concerning the server side of applications. The latter roughly translates to the known concept of Cloudbursting.
QoE	Quality of experience. It is a measure of a customer's experiences with a service. It may be related to some aspects of the QoS and QoP, but can also take into account other metrics.
Service handover	Service handover refers to the activity of transferring an active service between two computational resources (e.g. Cloudlets) with minimal or no disruption on the availability of the service. Ideally, service handover is transparent with respect to the user.
Situational Awareness	The ability of the BASMATI platform to recognise the “situation” characterising the actual combined status of users, applications and resources, aimed at achieving an effective and efficient management of applications and resources.
TBS	TripBuilder Batch Service
TOS	TripBuilder Online Service
RIR	Regional Information Repository
POI	Point of interest, a geographic location that is known as an interesting destination for a significant set of travellers. POIs are usually categorized according to their most relevant feature (e.g. place important for sport, art, music).
LOD	Linked Open Data
BEAM	BASMATI Enhanced Application Model, a TOSCA extension allowing to enhance TOSCA apps with BASMATI-specific information and metadata, used by the BASMATI platform support to manage the application over the Cloud federation.

Executive Summary

This report accompanies the demonstration deliverable D6.4 concerning the TripBuilder use case of the BASMATI project, a client/server application where two types of server activities (batch data collection and pre-processing versus online request management) are coupled together in order to provide an interactive service to end-users through typically mobile clients.

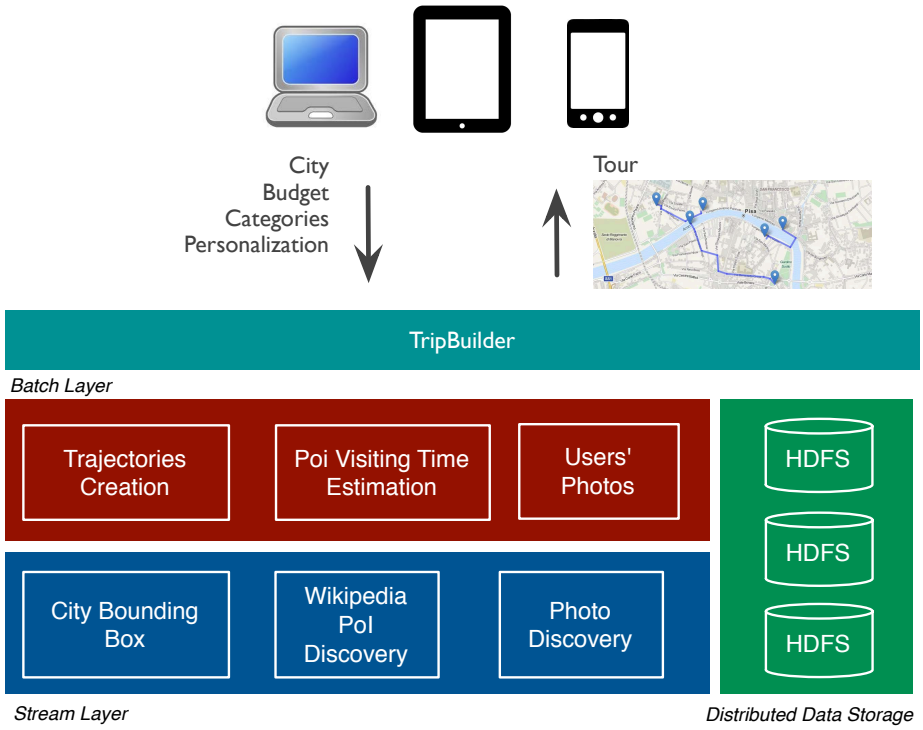
The report summarizes the main features and requirements of the Tripbuilder application, describes first its application scenario within BASMATI and the overall design principles of the use case. Then, the report describes the structure and organization of the use case, what features and components of the BASMATI architecture it leverages and the physical/virtual platform and systems employed, finally addressing the methodology and evaluation procedures defined in deliverable D6.5 for this use case as well as the expected demo output.

The demo will first show the functional behavior of the TripBuilder application, where a user is proposed a trip according to his/her preferences and time budget; then the behavior of the application servers under load will be shown, where the resource brokering and dynamic scale up supported by the BASMATI platform will allow coping with the increased load without impacting the end-user QoE.

Table of Contents

1 TripBuilder Application Description

TripBuilder is a system helping tourists to build their own personalized sightseeing tour. Given a target touristic area, the time available for the visit, and the tourist's profile (a set of preferences rating different categories of interest), TripBuilder provides its users with a time-budgeted tour that maximizes tourist's interests and takes into account both the time needed to enjoy the attractions and the time to move from one Point of Interest to the next one.



For the sightseeing tour generation TripBuilder exploits publicly available sources (Linked Open Data) to build a knowledge base and infer user interests. The sources currently exploited include Wikipedia, Flickr and Google Maps. Six main phases can be identified in the algorithm.

1. **PoI computation** – The LOD sources are used to build a model of the poins of interest (PoI) of the city (or cities), which is continuously updated.

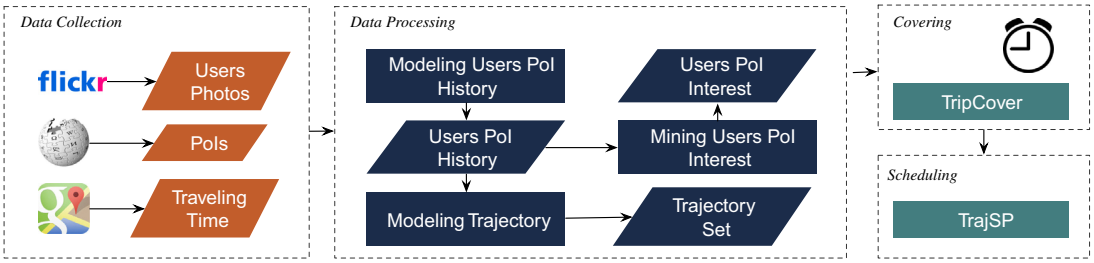
2. **User and PoI histories** – The same data sources are then used to build a model of past tourist itineraries across those PoIs, as well as the popularity of each PoI as related to each of the existing PoI categories.

3. **Trajectory computation** – Existing user trajectories extracted from the data are split and reprocessed in order to provide reliable segments for composing new trips.

4. **Travelling Time estimation** – By taking into account paths and time tags as well as Google Map data we compute an expected time cost of each itinerary segment.

5. **User-PoI interest evaluation** – A measure of the interest for the specific user for each of the available PoI is computed,

6. **Trip planning** – The actual trip plan is the result of two more sub-phases, namely TRIPCOVER and TRAJSP; the former employs an approximation algorithm to solve an instance of the Generalize Maximum Coverage problem, thus providing a set of trajectories maximizing user interest, subject to the time constraint, while the latter employs a heuristic approach to merge such trajectories into complete tours.



The computation phases are carried on by a complex infrastructure made up of four main blocks.

A. A ***stream layer*** based on Apache Storm that extracts and processes information from the LOD repositories,

B. A ***batch layer*** based on Apache Spark, that computes complex inferences and models from the available data, gaining knowledge from the combination of multiple data sources

C. A ***distributed data storage*** where all the data and knowledge extracted is accumulated, allowing the different modules of Tripbuilder to interoperate

D. The ***Tripbuilder Engine*** that receives user queries and exploits the knowledge in the distributed data storage to perform the TRIPCOVER and TRAJSP steps, providing answers to the queries

1.1 TripBuilder in BASMATI

TripBuilder is currently under evaluation for commercial exploitation. Its implementation should be considered as an industrial prototype, with no source code available for the code of the essential service, which is provided in executable format, and exploitation rights reserved except for the project's sake as a use case.

TripBuilder has been a valuable use case for the BASMATI project for several reasons.

1. it represents a typical application for people travelling "across regional borders" and accessing a service that can be duplicated and/or relocated to a different Cloud
2. it explicitly leverages "wisdom of the crowd" models, hence it is highly relevant and reactive whenever a large number of people concentrates is a specific area
3. it is mostly accessed by mobile devices;

The client part currently is browser-based, but a very simple application structure would allow additional functionalities as well as easy integration with the cloud brokering and computation offloading mechanisms of BASMATI.

The TripBuilder application for BASMATI consists in an online server that can be decomposed into a web front-end, a back end knowledge base and a module solving the combinatorial problem implied by each user query. The query computational latency is significant and critical, as it is part of the user perceived delay and thus directly impacts the quality of experience.

In the following we will call the TripBuilder Online Service (TOS) the service that answers user queries online leveraging a prebuilt repository of regional information (RIR). The TBS performs steps 4—6 of the sightseeing trip generation algorithm outlined in this section. We will call TripBuilder Batch Service (TBS) the off-line service that builds the regional information repository by crawling data sources and performing steps 1—3 of the algorithm.

In the following description, we will mainly speak of users meaning end-users, those who enjoy the TripBuilder service. For the sake of clarity we will call Application Provider the person/entity renting resources from the Cloud in order to provide the TripBuilder service, which generically is also a user of the BASMATI federation, but has the special role of application owner.

2 TripBuilder Use Case Scenario

End users of TripBuilder are a typical example of population satisfying the BASMATI project user model: a large set of people which travels across countries and need to efficiently access Cloud-based services, regardless of their current location. The region the users access the service from can be in principle completely unrelated to the one they are querying about (if the user is still

planning to go there) or it may be the same or relatively close to that (user just arrived, or is replanning part of his/her trip once there).

Of the two main components of TripBuilder, the TBS needs to crawl open data from other services. It is both computation-bound and network-bound. Its output (the RIR) is related to a specific city or geographical area. In this scenario the TBS can exploit at least two main features of BASMATI

- Scale up can be needed either in order to explore larger regions more quickly (distributing the load of more data to crawl and harder computational problems to solve) or in order to explore many separate regions concurrently (remove the constraint imposed by a dedicated, static hardware configuration)
- On demand allocation can be exploited to refresh the RIR of a specific region in order to 1) avoid the need to manage a fixed cluster of machines for the periodical update of data, 2) avoid network load centralization, and 3) dynamically select cost-effective resources.

The other main component of TripBuilder, the one which is actually the most critical for the QoE as it interfaces with the end users, is the TOS. The TOS needs to answer end-user queries about a city or region by combining information stored in the RIR, according to user input about its time budget, points of interest preferences and geographic location.
Each query amounts to producing an approximate result for the underlying NP-hard problem within a fixed deadline, as the QoE is influenced by the end-user latency. The required computational power becomes significant when several user queries reach the TOS server at the same time. The TOS too exploits the features of the BASMATI platform, although with slightly different goals.

- Scale up can be needed to allow more users from the same area to query the service without disrupting the service QoE (service latency reduction via load distribution and reduced service time)
- On-demand allocation on different Clouds of the BASMATI federation allows to serve more users from different areas using separate TOS server instances (network latency reduction via improved server placing) or with reduced cost (exploiting the tradeoff among network round trip time and application service time in order to select cheaper resources).

2.1 TripBuilder Use Case Design Principles

The design principles of the TripBuilder Use case were to show how the application can exploit scalability and flexibility in locating servers across intercontinental Cloud federations, taking into account user location, user preferences, resource availability and application QoE.

The use case research addressed deploying both the batch server and the online server through BASMATI, studying the application tradeoffs. A distributed architecture has been designed for

both the TBS and the TOS application servers, together with a BEAM encapsulation that allows the application to be automatically deployed on all Clouds of a BASMATI federation.

The implementation effort addressed mainly the online server part (the TOS). A primary goal was to show that scale up and on demand allocation over all clouds of the BASMATI federation was possible, in order to show the business advantage of an intercontinental federation.

Exploiting the user mobility knowledge to dynamically optimize server placement was also an option that the implementation had to make feasible.

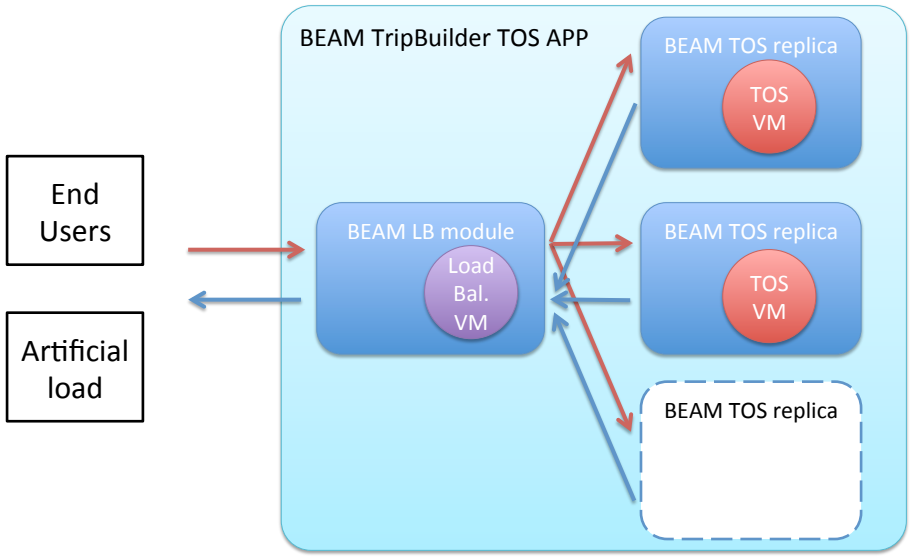
The TBS part of TripBuilder has been studied and its BASMATI enabled version has been designed. The approach is close to the one adopted for the TOS app, which is described in the next section, based on an on-demand activation of multiple independent servers, but the implementation is more complex. In order to spread the crawling load as well as the computational load, the TBS is already a parallel server relying on the SPARK and Hadoop frameworks. Deploying multiple copies of these frameworks to activate multiple TBS servers required too much implementation effort for the use case and was deemed to be less critical for the sake of demonstrating the exploitation of the BASMATI platform.

3 Use Case and Demo structure description

The TripBuilder use case implementation focuses on the adaptation of the TOS user front-end of TripBuilder, exploiting its deployment of top of BASMATI in order to achieve:

- Load distribution by dispatching queries to different server replicas.
- Guaranteed QoE by exploiting BASMATI SLA mechanisms to activate new replicas

The TOS application server is set up within a single VM, containing the RIR image that allows to answer user queries. The VM image so crafted contains thus both the actual TripBuilder code and data, and the application front-end serving the web user interface (built with HTTP, CSS and Javascript) to the end-user.



The single-VM module encapsulated within a distributed application architecture via the BEAM language. Within the application, a load-balancing VM distributes TripBuilder connectionless queries to multiple TOS server replicas.
TOS server replicas can be automatically spawn over resources from the Clouds of the federation. This brings into play the mechanisms provided by BASMATI in order to broker IaaS resources, according to the Application provider preferences and to an SLA specification, then to deploy the TripBuilder service on those resource.
The execution behavior of the replicas is monitored by BASMATI mechanisms, so that their measured performance falls within the range that is acceptable for the Application provider. For the TripBuilder use case, the main QOE constraint is that the service time of end-user queries is bounded. The TOS service is compute intensive, so its service time is influenced by the current server load, that is by the amount and type of concurrent queries by the set of users at any given time.

For this use case, the QoE constraint translates to an SLA constraint on the time required for each query. The response time of each query to each server is monitored and triggers an SLA violation if it exceeds the bounds. This will result in the SLA of the TripBuilder application being reevaluated, typically leading to a new IaaS resource being selected for deploying a new TOS replica which starts serving a share of the query stream. The opposite scale-down behavior can be triggered either by observing that the interval between received queries at a replica is too low, or that the load balancer module does not receive enough queries per unit of time.

In order to test the actual application management process, during the development we prepared a set of small program that can provide some artificial load, either leading to a reduced QoE for the real end-users and to a scale up reaction that compensates the QOE loss.

The load-imposing scripts directly access the TOS HTTP interface and replay a set of queries (or perform a stream of synthetic queries) with assigned speed and concurrency.

3.1 Exploitation of BASMATI features and components

The TOS application demonstrating the TripBuilder Use case will exploit the following BASMATI components in order to locate resources for, deploy and manage multiple instances of the TOS server.

- Application Repository
- ACE engine
- Decision Maker
- Knowledge Extractor
- Resource Broker
- Application Controller

In addition to those, the load balancing and dynamic QoE management will also exploit the following BASMATI components

- SLA Manager

3.2 Physical and virtual resources employed

The demo deployment will exploit multiple instances of the TOS server, which are activated when artificial load is added to trigger a delayed answer from the servers already active. The different VM allocated to the Tripbuilder TOS application will be spread over different Clouds accessed by means of BASMATI.

4 Use case Demo Evaluation

To evaluate the contribution of BASMATI to TripBuilder has been performed a proper set of evaluations. Deliverable D6.5 reports all the details regarding the evaluation process, contextualizing it in the frame of the project and in relation with the other uses cases of BASMATI. In this deliverable, such process is briefly summarized. In order to provide such information, it is presented the evaluation methodology (Sec. 4.1) and the expected output (Sec. 4.2).

4.1 Evaluation Methodology

The TripBuilder evaluation has been organized in four different phases, each one with specific goals and a different involvement and integration with the BASMATI technology. The evaluations have been successfully performed and demonstrated the viability of exploiting BASMATI solutions for TripBuilder use case.

4.1.1 Functional Evaluation

This evaluation step focused on the execution of plain TripBuilder on a Cloud. It has been aimed at validating the possibility of launching and executing TripBuilder on a Cloud by means of BASMATI technologies. To this end has been created a BEAM version of TripBuilder that has been loaded from the Application Repository and deployed by means of ACE.

4.1.2 Quality of Experience

This evaluation has been aimed at the definition of the QoE model governing TripBuilder when the frequency of requests to BASMATI varies. To this end has been integrated in TripBuilder a monitoring probe able to measure the latency of each trip planning request. The TripBuilder application has been executed on CNR premises without the involvement of other components. To conduct the evaluation, has been created an automatic generator of requests that has been configured to issue requests to TripBuilder at different frequencies. The probe measured the actual impact on the latency associated to each request, supporting the definition of the QoE model.

4.1.3 Multiple Instances

As it happens with most online applications, when the frequency of requests to TripBuilder increases, the application became quickly unusable, thus there is a need for creating multiple instances of the application when the workload gets higher. To assess the ability of BASMATI technologies in easing the task of generating multiple instances of an application, we designed a proper experiment. The ultimate aim was to validate the support of BASMATI to the creation of many instances of an application (expressed using the BEAM) without requesting users do directly dealing with the manual deployment of each single instance. To this end, has been conducted an evaluation involving the following components:

- **Application Repository**: where the BEAM describing BASMATI has been stored;
- **Decision Maker, Knowledge Extractor, Resource Broker**: to find and select the resources where to deploy TripBuilder instances;
- **Application Controller and ACE engine**: to actually perform the deployment and manage the application instances;

4.1.4 Load Balancing

Beyond the ability of seamlessly execute multiple instances of the TripBuilder application, we designed a further evaluation aimed at assessing the effectiveness of the runtime support of BASMATI, e.g. the ability of automatically fire and execute new instances of TripBuilder when the QoE of users decreases. This phase builds on the “Multiple Instances” one and evaluates the actual consequences of taking decisions on the overall amount of TripBuilder instances to maintain up and running, depending on the frequency of requests issued to the existing instances. To this end, the evaluation involved another component of the BASMATI toolkit, the SLA Manager. Such component is aimed at evaluating the actual service level provided by

TripBuilder to the users and, in case of misbehavior (meaning a miss of adherence to the SLAs) it does issue violations. When violations are received and processed, some actions can take place. In the case of TripBuilder, violations represent issues in providing the planned QoE and have as a consequence the creation (or reactivation) of a new instance of TripBuilder.

5 References

[1] http://tripbuilder.isti.cnr.it/ beta version of the tripbuilder application (single VM, fixed sets of demo repositories).