Cyberinfrastructure programming with COMPSs

Francesc Lordan*, Daniele Lezzi*, Rosa M. Badia*
*Barcelona Supercomputing Center (BSC)
E-mail: {francesc.lordan, daniele.lezzi,rosa.m.badia}@bsc.es

Keywords—Distributed Computing, Workflow Managers, Programming Models, Edge Computing, Internet of Things, Cloud Computing

I. Extended Abstract

Cyberinfrastructures are compositions of computing systems, data repositories and storage systems, data collection instruments and visualization environments all linked together to improve research productivity and enable breakthroughs not otherwise possible.

As for previous computing infrastructure transformations, developers had to adopt new execution paradigms to exploit them. Distributed systems deprecated the traditional monolithic model in favor of service-oriented applications. To exploit the Cloud and offer SaaS, developers embraced the microservices model so that services could adjust the number of instances of each microservice to the current workload. Bringing down the computation from the Cloud to the Edge mitigates the network issues - latency and bandwidth - and enables new opportunities. On Fog infrastructures, computing devices can join in or leave at their own will. For dealing with such dynamicity, microservices became serverless and stateless.

IoT devices have sensors that permanently produce data. These devices can monitor this data themselves and, when a certain condition is met, trigger a response that may require heavy computation capabilities, or they can provide other devices with this information so they process it on a real-time basis. To support the former scenario, developers can turn to Function as a Service (FaaS): functions executed on the underlying platform in a serverless, stateless approach. To support the latter, developers need to code using stream processing frameworks such as Kafka streams or Spark Streams.

A. The COMPSs Programming Model

COMPSs [1], [2] is a framework that aims to ease the development of distributed applications. Its core it is a task-based programming model [3], [4] and a runtime toolkit executed along with the application which automatically detects its parallelism and orchestrates the execution of its tasks on the available computing nodes. Thus, it is able to exploit distributed, heterogeneous and highly-dynamic infrastructures while keeping the code totally unaware of the infrastructure and parallelism details.

To develop a distributed application with COMPSs, programmers code the logic of the application in a sequential, infrastructure-unaware fashion with no API invocations as if the code was to be run on a single-core computer. For the runtime system to detect the tasks composing the application, developers select a set of methods whose invocations create new asynchronous tasks by annotating them with the @task tag. In order to guarantee the sequential consistency of the code, the runtime system monitors the data values accessed by each tasks to find dependencies among them. So the runtime can better exploit the application parallelism, developers need to describe how the method operates (reads, generates or updates) on each data value by indicating its directionality (IN, OUT, INOUT, respectively).

The code snippet in Figure 1 shows a sample application with three tasks: two of them (generateReport) process their respective input files generating two Report objects that are merged in a third task – merge_report.

```
@task(file=FILE_IN, returns=Report)
def generate_report(file):
    ...
    return report;

@task(r1=INOUT, r2=IN)
def merge_report(r1, r2):
    ...

def main(files):
    final_report = None
    for file in files:
        partial_report = generateReport(file)
        if final_report:
            merge_report(final_report, partial_report)
        else:
            final_report = partial_report
```

Fig. 1. Sample COMPSs application using Python

B. Envisaged Computing Patterns

In cyberinfrastructure environments, authors envisage three possibilities that require computation. On the first one, known as *sense-process-actuate*, the infrastructure is supposed to give a proper response to an event detected on one of its sensors; for instance, turning on the AC system when temperature in a room reaches more than 27 degrees. A second approach that might require computation is stream processing. Sensors may continuosly produce data that need to be processed in real-time to produce a response or update the data visualization components. For instance, a camera could video-stream the inside of the room, and the AC system could be turned when the processing of such stream identifies typical reactions to heat. Finally, the third scenario that might require computation is batch processing. Cyberinfrastructures collect, generate and store big amounts of data. All this information can be analyzed to produce new knowledge; for instance, machine learning techniques could be applied to improve the models detecting people reactions and change the behavior of the infrastructure.

C. Runtime System Infrastructure

Unlike previous environments with a single entry point to start an application, on cyberinfrastructures any device can detect an event and trigger a computation. Thus, the each device must be able to work in a standalone manner and run the computation in an efficient way: detecting the parallelism inherent in the computation and run the detected tasks on the computing elements embedded on the device. For doing so, each node of the infrastructure will host the execution of a COMPSs agent, a deamon process waiting for execution requests. When a the node triggers a new execution, it contacts this agent which runs the corresponding code detecting the tasks composing it and orchestrates their execution on the available computing resources.

On the one hand, the computing resources locally embedded on the same device might not be enough to host the expected computation. On the other hand, not all the devices of the infrastructure will trigger executions; indeed, even those that raise events may not compute anything temporarily. Thus, if COMPSs agents were able to only exploit the local processors, most of the computing power would remain idle. In order to better exploit the computing power of the infrastructure and shorten the execution time of the application, COMPSs agents must interact with each other and offload part of their computation. For that purpose, one COMPSs agent can call the same method that started a computation on another agent to execute a task in the remote agent. This agent will run the task code, analyze it detecting new tasks on the task code and orchestrate the execution of these tasks on other agents of the infrastructure.

For infrastructures with a small number of devices, a COMPSs agent can individually manage the execution of the tasks it has detected on any of the other COMPSs agents composing the infrastructure. However, the bigger the infrastructure is, the higher the complexity of the scheduling problem becomes. Assigning tasks to resources may become a computational load bigger than the actual computation to perform, or for very large infrastructures, it may simply not fit in resource-scarce devices.

To solve that problem, we propose to organize all the agents of the infrastructure in a hierarchic way. The topology of such organization can be represented as an acyclic graph where each node of the graph represents an agent of the infrastructure, and edges illustrate the possibility of one agent to submit tasks onto the agent represented by the other end of the edge. Thus, one agent can decide to run tasks on its local devices or offload them onto any of the other agents with whom it is directly connected on the hierarchy. In turn, the receiving agent could decide to run the task on its local computing devices or onto any of its direct neighbors. Thus, a computation triggered by any of the agents could use as many resources of the infrastructure as the computation needs to ensure that the application runs and it does so in the shortest execution time possible without adding a significant overhead due to the scheduling.

An important aspect to bear in mind when dealing with cyberinfrastructures is the potential mobility of its components. Smartphones, tablets and more complex devices attached to a battery with wireless network connectivity may be part of the system. These kind of devices may join in the infrastructure or leave it at their own will. When a device joins in, it is included in the system topology to add its resources into the pool so other agents can use them and, on the other hand, the mobile device can exploit the rest of the infrastructure to enhance its performance. When a device gets disconnected from the infrastructure, a communication link is broken and an edge of the graph is removed. At that point, the agents at both ends of the edge need to re-schedule the execution of the tasks already offloaded onto the other agent and assign them to resources to which it remains connected. Thus, any computation already started can go on and finish its execution.

Good criteria to build the topology are the stability and latency of the network. Establishing the topology according to that ensures that agents will always try to offload tasks onto nearby resources, on the fog, rather than submitting them to the Cloud achieving a higher performance. Besides, in the case of network disruption, the system could remain usable even without the cloud.

II. Acknowledgment

This work is supported by the Horizon 2020 research and innovation programme under grant 730929 (mF2C Project).

Salis, A., Mancini, G., Bulla, R., Cocco, P., Lezzi, D. and Lordan, F. (2019) Benefits of a Fog-to-Cloud Approach in Proximity Marketing. In: Mencagli G. et al. (eds) Euro-Par 2018: Parallel Processing Workshops.

Lordan, F. and Badia, R.M. (2017) COMPSs-Mobile: Parallel Programming for Mobile Cloud Computing. In Journal of Grid Computing, vol 15, 357.

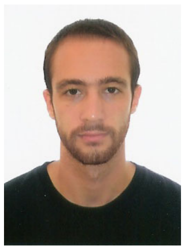
References

[1] R. M. Badia and et al., “COMP superscalar, an interoperable programming framework,” *SoftwareX*, vol. 3, pp. 32–36, 12 2015. [Online]. Available: http://dx.doi.org/10.1016/j.softx.2015.10.004

[2] Workflows and Distributed Computing - Barcelona Supercomputing Center (BSC). (2018) COMP Superscalar. [Online]. Available: http://compss.bsc.es

[3] F. Lordan *et al.*, “ServiceSs: an interoperable programming framework for the Cloud,” *Journal of Grid Computing*, vol. 12, no. 1, pp. 67–91, 3 2014. [Online]. Available: https://digital.csic.es/handle/10261/132141

[4] E. Tejedor *et al.*, “PyCOMPSs: Parallel computational workflows in Python,” *The International Journal of High Performance Computing Applications (IJHPCA)*, vol. 31, pp. 66–82, 2017. [Online]. Available: http://dx.doi.org/10.1177/1094342015594678



Francesc Lordan obtained the Ph.D. in Computer Architecture from the Universitat Politecnica de Catalunya in 2018 after defending his Ph.D. thesis: ”Programming Models for Mobile Environments”. Since 2010, Francesc is part of the Workflows and Distributed Computing group of the Barcelona Supercomputing Center. His efforts have focused on the COMPSs programming model: a task-based model for developing parallel applications running on large distributed infrastructures such as clusters, supercomputers, grids and clouds. Francesc has published more than 20 articles in International conferences and journals, and he has been directly involved in the European projects mF2C, ASCETIC and OPTIMIS. He has also provided support to other collaborative projects such as Venus-C, EU-Brasil OpenBio, Transplant and the Human Brain Project. His research focuses on programming models that aim to ease the development of parallel applications by hiding the technical concerns of heterogeneous and distributed infrastructures.