Enabling Run-time Resource-aware Task Placement in Fog Scenario

Domenico Iezzi*,1, Michele Zanella*,1,
Giuseppe Massari*,1, William Fornaciari*,1

* *Dipartimento di Elettronica, Informatica e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5 20133 Milan, Italy*

ABSTRACT

The amount of data produced at the edge of the network led to the introduction of the Fog Computing paradigm, as a way to assist resource constrained devices in processing and delivering data on behalf of the Cloud. This intermediate layer of computing infrastructure and services, closer to the edge, provides better response times thanks to the physical proximity between devices and computing nodes, while introducing new challenges regarding privacy, fault tolerance and task allocation strategies. In this work we introduce a framework for distributed computation of tasks, which extends a resource-aware programming model to abstract the underlying complexity of task and data allocations.

KEYWORDS: Resource Management, Programming Model, Task Offloading, Fog Computing

1 Introduction

Nowadays, world is experiencing a new computing era. Thanks to the advent of Internet of Things and Cloud computing, a huge number of embedded devices (sensors, actuators...) can be interconnected, enabling the possibility to acquire a lot of data that can be further processed remotely for a variety of purposes. Unfortunately, the management and transmission of all raw information from the edge to the Cloud requires ever larger data centers and it considerably increases network bandwidth demand, making it not sustainable or very expensive in the future. Moreover, some use-case scenarios (e.g., automotive, emergency management...) have real-time constraints and require an high responsiveness which are not satisfied using a Cloud solution, due to latency and unreliability introduced by the Internet network. Thus, in recent years, new approaches aim at shifting data processing back to the edge devices, where data is generated, in order to: (a) extract small-sized information from the raw data and avoid an high bandwidth consumption; (b) reduce the transmission latency to the Cloud, thus enabling faster response; (c) increase the reliability of the system leveraging local nearby devices. At this regard, the Fog computing paradigm was introduced by Cisco in 2012 and it includes a layer between the edge and the Cloud able to provide computational and storage services. This layer is composed by a set of devices

[1]E-mail: {name.surname}@polimi.it

(e.g., embedded boards, smartphones, gateways, local servers…) located nearby the data sources. In particular, the architecture can be considered as a bi-dimensional resource space in which data and computational tasks can be distributed among the devices both "horizontally" (i.e., between devices in the same layer) and "vertically" (i.e., between devices in different layers)[ZMGF18].

2 Run-Time Distributed Resource Management

In this context, since the highly distributed and heterogeneous nature of the system, both in terms of devices typology and hardware architectures, we need a management system able to deal with (a) mobility and availability of resources, (b) energy and computational capabilities constraints of devices. In this sense, our work includes an open source resource manager, the BarbequeRTRM[BMF15], which fit the aforementioned needs and has been extended in a distributed version. In particular, the different instances of the BarbequeRTRM, running on the different devices, can communicate to each other through the *RemoteProxy*. While the *Distributed Manager* module is in charge of discovering new available devices and synchronizing the current status of the systems.

However, the complexity of such infrastructure requires to be hidden from a developer perspective, demanding new tools and programming model. In fact, in order to improve an efficient utilization of the system, applications have to be composed by different modules, which are called "tasks" in our jargon, that can be dispatched on separated devices. In this way, we can represent an application through a *task graph*, which is a direct graph, where the arcs represents the mapping between the kernel (i.e., a specific workload to be executed), the buffers (i.e., memory space in which read/store data) and the events. The nodes can be buffers or kernels and two adjacent nodes cannot be of the same type. Exploiting the task-graph and application information, the instances of the resource manager follow a distributed strategy to allocate the task at system-level, optimizing energy efficiency and QoS, while implement a centralized strategy to pick the best resource allocation at device-level, in order to optimize local metrics (e.g., temperature, load…) and applications cohabitation.

To integrate the application with the resource management, we adopt a resource-aware programming model developed for a deeply heterogeneous HPC context inside the MANGO EU project[FAA+18, AFM+18]. Then, in order to make it compatible with the aforementioned distributed system, we develop the *BeeR* framework[Iez18], which allows an effective offloading of tasks on remote targets.

3 The BeeR Framework

BeeR is an extension of the MANGO framework with the goal of targeting remote embedded devices (IoT, mobile, smart objects) connected in a network, that can be exploited to execute computational workloads. These can be components of a complex application (i.e., tasks), or standalone functions which returns some data. The framework consists of two main components.

The *BeeR daemon* is a standalone daemon application which is usually installed on remote devices. This daemon listens for requests coming from applications that may be running on network nodes or general purpose computers. Through a TCP connection, the daemon is

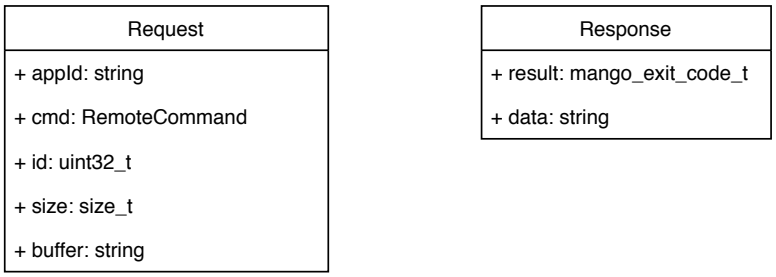

Figure 1: Request and Response classes used to represent message sent

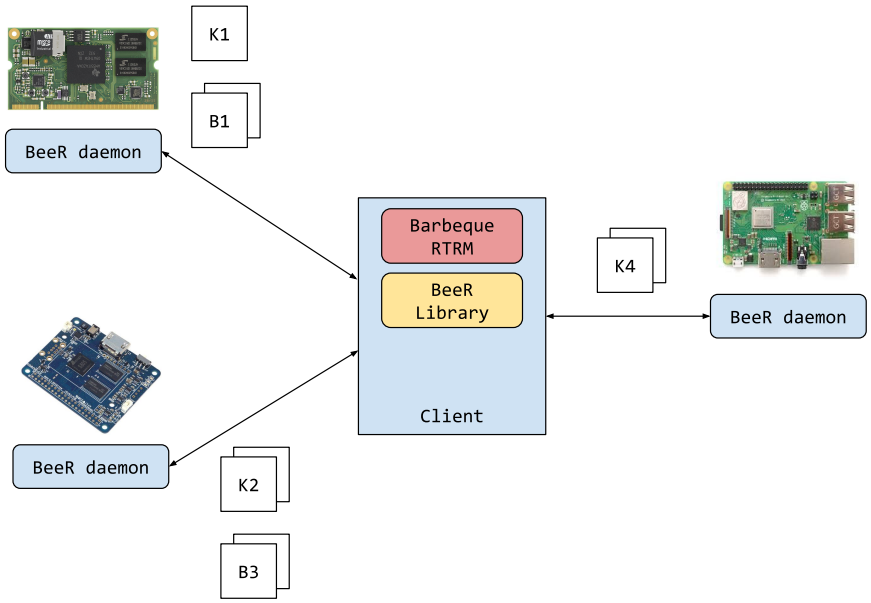

Figure 2: Typical scenario where a BeeR application offload kernels and buffers to remote devices running the daemon

able to receive *kernels* and *buffers* of the aforementioned task-graph applications. Once the data is ready, firstly the buffers are allocated using shared memory API, then the tasks are executed with references to the shared memory objects containing the respective buffers. Finally, after waiting for the completion of the tasks through a specific remote call, the application retrieves the results.

The protocol consists of a request and a response, defined as C++ classes as shown in Figure 1. The requests contains an application id, which is unique to the application that generated the call, a *RemoteCommand* field representing the action to be executed, and three field (id, size, buffer) which are optionally used by some function (e.g a buffer write operation requires the specification of all the three fields). The response class contains a result code and optionally a string containing a message or a buffer for some specific commands. The framework leverage object serialization to send instance of requests and responses back and forth.

On the other side there is the client application exploiting the *BeeR library*, an extended

version of the MANGO libraries providing a programming model which abstracts the underlying complexity of task allocation and resource management and gives fine-grained configuration mechanisms. In particular, it defines function for (a) registering binary programs as kernels and assign an id, (b) read and write buffers, (c) start remote execution of kernels, (d) manage the occurrence of remote events (e.g. kernel termination) and (e) retrieve statistics about kernel execution. As shown in Figure 2, the library layer directly communicates with BarbequeRTRM in order to provide run-time management of remote tasks considering not only device resources, status and capabilities, but also application QoS requirements. Depending on the type of application and its requirements, the resource manager can decide whether to offload the task to remote devices or keep the computation local and use edge devices only for data gathering, depending on the metrics chosen for the current application.

Devices and applications coexists in a many to many relationship, meaning that a single daemon instance can run kernels coming from multiple applications from different sources, and a BeeR application can scatter its kernels to different device.

4 Conclusions

In the context of Fog computing, the complexity of the system requires a distributed resource management approach along with a proper programming model able to deal with heterogeneity and mobility of resources, as well as application modularity. At this regard, our work presents an extension of a suitable resource manager and a resource-aware task-offloading framework. We are planning to evaluate the framework on a real cluster, containing some low-performance embedded systems acting as edge devices and others that could act as Fog nodes. This will be done by running a ported version of the Pathfinder algorithm from the Rodinia suite of GPU parallel benchmarks.

References

[AFM+18] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, and M. Zanella. Managing heterogeneous resources in hpc systems. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '18, pages 7–12, New York, NY, USA, 2018. ACM.

[BMF15] P. Bellasi, G. Massari, and W. Fornaciari. Effective runtime resource management using linux control groups with the barbequertrm framework. *ACM Trans. Embed. Comput. Syst.*, 14(2):39:1–39:17, March 2015.

[FAA+18] J. Flich, G. Agosta, P. Ampletzer, D. Atienza Alonso, C. Brandolese, E. Cappe, A. Cilardo, L. Dragic, A. Dray, A. Duspara, W. Fornaciari, E. Fusella, M. Gagliardi, G. Guillaume, D. Hofman, Y. Hoornenborg, A. Iranfar, M. Kovac, S. Libutti, B. Maitre, J. M. Martinez, G. Massari, K. Meinds, H. Mlinaric, E. Papastefanakis, T. Picornell, I. Piljic, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zanella, M. Zapater, and D. Zoni. Exploring manycore architectures for next-generation hpc systems through the mango approach. *Microprocessors and Microsystems*, 61:154 – 170, 2018.

[Iez18] D. Iezzi. Beer: an unified programming approach for distributed embedded platform. 2018.

[ZMGF18] M. Zanella, G. Massari, Andrea G., and W. Fornaciari. Back to the future: Resource management in post-cloud solutions. In *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications*, INTESA '18, pages 33–38, New York, NY, USA, 2018. ACM.