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

Domenico Iezzi, Michele Zanella
Giuseppe Massari, William Fornaciari
{name.surname}@polimi.it

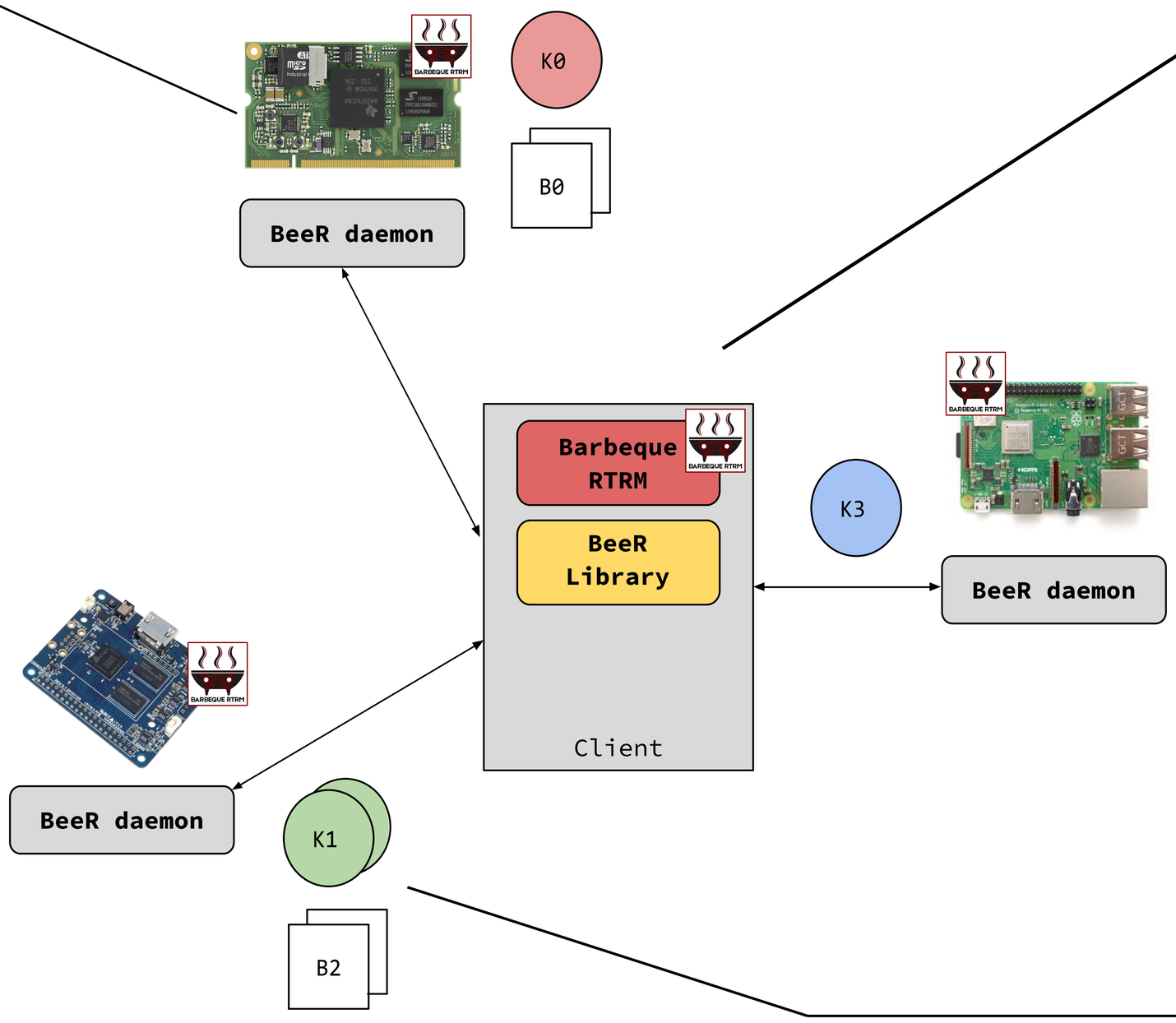
OUR RESEARCH

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. Despite the improvements in network technology, the huge amount of data coming from the edge is pushing bandwidth requirements, storage and computational demand to unsustainable levels. Thus, in recent years, new approaches aim at shifting data processing back to 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 includes a layer between the edge and the Cloud able to provide computational and storage services. In this context, given the highly distributed and heterogeneous nature of the system[1], 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.

THE BEER FRAMEWORK

BeeR framework allows tasks and data, composing an application, to be executed on remote devices.
Each device connected through the network will run an instance of *BeeR daemon*. This is in charge of:
(a) receiving application requests,
(b) handling incoming tasks and
(c) managing their execution.

The client application is compiled with *BeeR library*, which provides an easy to use API for:
(a) registering the task-graph,
(b) initializing the context and components,
(c) starting the execution of each kernel,
(d) handling the occurrence of remote events (e.g. task termination),
(e) retrieving run-time profile information.



RUN-TIME DISTRIBUTED RESOURCE MANAGEMENT

The open source resource manager, **BarbequeRTRM**[2], fits the aforementioned needs and has been extended in a distributed version.
The different instances of the BarbequeRTRM, running on the different devices, can communicate to each other through the *RemoteProxy*.
The *Distributed Manager* module is in charge of discovering new available devices and synchronizing the current status of the systems.

Exploiting the task-graph and application information, the instances of the resource manager follow two-level stratgy:
(a) a distributed strategy to allocate the task at system-level, optimizing energy efficiency and/or QoS,
(b) 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.

BENCHMARKING

We evaluated the framework by porting the *PathFinder* benchmark from the Rodinia suite of parallel benchmarks. It finds the path with the lowest accumulated weight from the bottom to the top of a matrix, where each cell contains a natural number representing the weight of a node. It splits the source matrix by columns into smaller sub-matrices and define a task for each, so that the computation can happen in parallel. Each task will produce an array of results which can be finally merged.

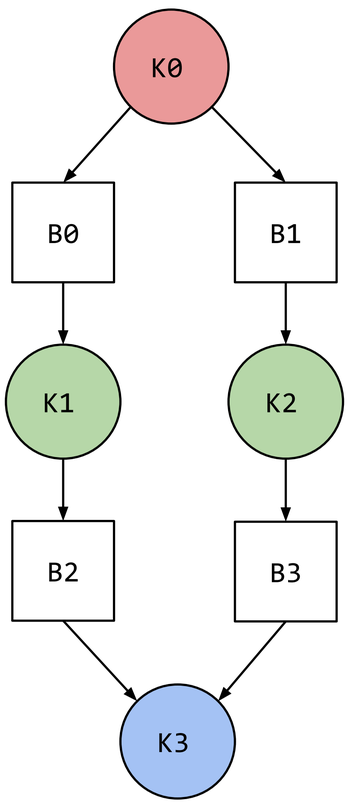


Figure 2

TASK-BASED PROGRAMMING MODEL

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[3].
In fact, to improve an efficient utilization of the system, applications have to be composed by different modules, which are called "*tasks*", that can be dispatched on separated devices.

In this way, we can represent an application through a **task-graph** (shown in the Figure 2), which is a direct graph, where the arcs represents the mapping between the kernels (i.e., a specific workload to be executed), the buffers (i.e., memory space in which read/store data) and the events.

REFERENCES

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

[2] P. Bellasi, G. Massari, and W. Fornaciari.
Effective runtime resource management using linux control groups with the BarbequeRTRM Framework.
ACM Trans. Embed. Comput. Syst, 2015

[3] 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, 2018.

[4] D. Iezzi.
Beer: an unified programming approach for distributed embedded platform.
Master's thesis, Politecnico di Milano, 2018

FUTURE WORKS

We plan 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.

Moreover, we will address the privacy and security issues by implementing a distributed key exchange protocol and an encryption layer as an extension of the BeeR framework.