User Guide
AGORA application handler

Draft version

April 3, 2018

Contents

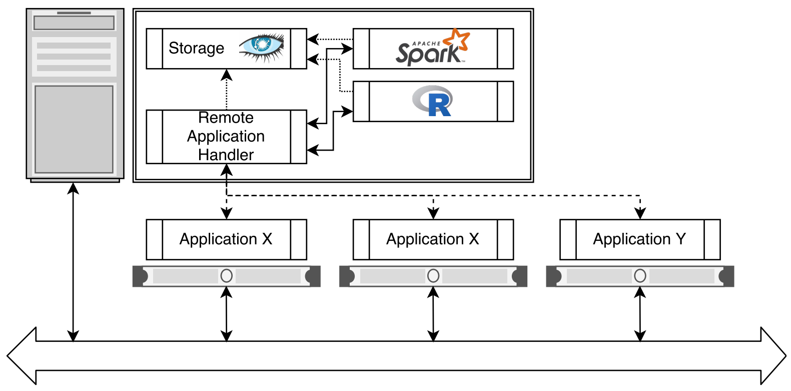

Figure 1: Overall architecture of the AGORA application handler.

1 Introduction

The main goal of AGORA framework is to perform an online distributed Design Space Exploration. The key concept is that on one hand we have the AGORA remote application handler that orchestrate the task; on the other hand, we have several instances of the application that explores the required configurations.

Figure 1 shows the overall structure of the framework. In particular, all the instances of the application executes on production nodes, for examples computing nodes of an HPC center, while the AGORA remote application handler runs in a dedicated server. The communications between instances of application and the remote handler leverage the lightweight MQTT protocol, using the publish-subscribe pattern. From a logical point of view, the remote application handler acts as a server, while the instances of application act as clients.

The typical workflow starts when the first instance of an unknown application begins to execute and notify its existence to the server. The latter will ask to an application instance information about the application itself (e.g. the list of metrics of interest or the Design of Experiments required) and computes the set of configurations to explore. At this point the server dispatch to clients the required configuration and it expects the observed performance of the application. Once we have explored all the required configurations, it computes the application knowledge to broadcast to the clients. In this way, all the instances of the application might start to leverage these information to automatically tune themselves.

The implementation of the agora application handler is designed to scale and to handle crash of clients. In particular, the agora application handler uses a threadpool to process incoming message from clients, which might be instances of different applications. It leverage Apache Cassandra[1] to store information about clients and uses a flexible plugin system to build the model of a metric of interest, starting from the observations.

The remainder of the manual is structured as follows: at first we describe in more

[1]homepage: `http://cassandra.apache.org`

details the remote application handler, then we discuss how the local application handler interacts with mARGOt. The integration process is totally hidden by using the high-level interface mARGOt heel. Please, refer to the mARGOt heel user guide for further details. The last section of the document explain how to start the AGORA remote application handler.

2 Remote application handler

The remote application handler is designed to orchestrate the Design Space Exploration of an unknown application in a distributed fashion. Which means that each instance of the application contributes at building the knowledge, exploring a different configuration, without restarting the server. To achieve this goal, we need to define three concepts: how to explore the Design Space (i.e. the Design of Experiments), what is the application knowledge and how to predict the behavior of a metric of interest.

2.1 Design of Experiments

Within this context, the Design Space Exploration (DSE) aims at exploring the Design Space to characterize the behavior of the application. The Design of Experiments (DoE) aims at selecting which configuration must be explored. Given the design space of each software knob of the target application, the most common DoE is exploring all the combinations (named full-factorial). However, since the design space grows exponentially with the number of software knobs, and with the admissible values of each software knob, a full-factorial exploration might be unfeasible. This is a well known topic in literature, therefore are available different techniques that aims at selecting the most meaningful configurations given the method used to generate the application knowledge.

The remote application handler, enable the user to select which DoE technique to use during the DSE. To have a list of all the supported DoE in the current version of AGORA, please refer to help command. Moreover, to produce a more robust knowledge of the application, it is possible to set the minimum number of times that each configuration must be observed.

2.2 The application knowledge

In the mARGOt user guide, the application knowledge was implicitly defined by the list of Operating Points. Since we aim at modeling the application behavior at runtime, we define the application knowledge as the full-factorial combination of all the admissible values of all the software knobs and of the features of the application.

Therefore, the final goal of the AGORA framework is to produce a list that relates a configuration, for each possible feature, with the expected value of the metrics of interest. In particular, it characterize a metric with its mean value and with its standard deviation. Which is delivered to each client of the application.

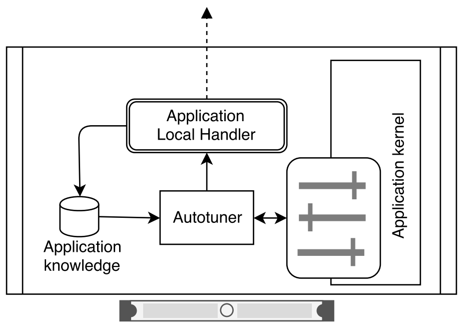

Figure 2: The architecture of the local application handler.

2.3 How to generate the application knoweldge

During the DSE, the remote application handler dispatch configuration to explore to the available clients. However, since the selected DoE may explore only a subset of the Design Space and since we can't force the exploration of data features, we need a technique to generate the application knowledge.

This topic is a well known topic in literature, therefore are available different techniques that aims at solving this problem. Since each technique shines in a particular context, the remote application handler let the user to select its preferred technique, with metric granularity. Moreover, since a metric might be totally application specific, e.g. the elaboration error, AGORA implements a plugin system that enable the user to write its own technique to derive a metric. To have a list of all the supported prediction methods in the current version of AGORA, please refer to help command.

3 Local application handler

The local application handler has three main goals: 1) it provides to the remote application handler information about the actual application, 2) it manages the mARGOt application knowledge to perform a DSE, and 3) provides to mARGOt the final application knowledge.

Figure 2 shows the architecture of the local application handler. While the mARGOt autotuner is synchronous with respect to the execution flow of the target application, the local application handler is a separate thread that communicates with the remote application handler. In particular, after the initialization, it notifies the existence of a new instance of the application to the server and then it waits for messages.

If the server has no information of this application, the local handler send to the remote handler all the required information. If the server ask to this client to explore a configuration, the local handler manage the application knowledge to force mARGOt on selecting the given configuration. If the server send the application knowledge, the local handler replace the old one with the new one. The only synchronous operation, with respect to the application, happens after each elaboration when mARGOt sends to the remote handler the observed performance of the application.

From the integration point of view, if we are using mARGOt the high-level interface, no additional code is exposed to the end-user. However, all the required information must be defined in the adaptation configuration file, as described in the mARGOt heel user guide.

4 How to start the remote application handler

Since the AGORA application handler implementation requires several dependencies, by default is disabled. It is possible to enable it, using the cmake flag *-DWITH_AGORA=ON*. The build system will automatically download and compile the Cassandra C/C++ client and the Apache Paho MQTT client library.

The remote handler executable is named *agora* and expose several flags to configure its behavior. Please, use the *–help* command option to have the full list. In particular, it is possible to configure parameters of the MQTT connection, parameters to the storage connection and the parameters of AGORA itself.

For the MQTT connection, it is possible to specify the url of the broker, the username and password required to authenticate to the broker (if needed) and the communication quality of service. For the storage conncetion, it is possible to specify the url of one cluster of the databse, and the usename and password required for the authentication (if needed). You need to start the MQTT broker and the Cassandra database before of running the AGORA application handler.

The server requires only two parameters to successfully start its execution: the workspace folder and the plugin folder. The workspace folder is the path where the application is able to store files to generate the application knowledge, while the plugin folder is the path to the available plugins. The standard ones, shipped with AGORA, are located in *<repository_root>/agora/plugins*.

5 Acknowledgement

This work is based on the M.Sc. Thesis of Cristiano Di Marco: "A distributed framework supporting runtime autotuning for HPC applications