ISSN: 2641-2039
DOI: 10.33552/GJES.2021.07.000659
Global Journal of Engineering Sciences
Iris Publishers

Mini Review

On Developing a Runtime Environment for Heterogeneous Distributed Computing

Ahmed Abdelmoamen Ahmed*

Department of Computer Science, Prairie View A&M University, Prairie View, TX, USA

***Corresponding author:** Ahmed Abdelmoamen Ahmed, Department of Computer Science, Prairie View A&M University, Prairie View, TX, USA.

Received Date: February 18, 2021

Published Date: February 26, 2021

Abstract

This paper presents the design and implementation of a distributed runtime environment to support the deployment and execution of distributed actor-based applications on connected heterogeneous computing machines. The distributed runtime environment consists of connected runtime environments, which support the execution of individual application components (actors) and managing their communication. The programmability benefits of our runtime environment are evaluated by developing a computational-intensive application across three connected heterogeneous devices. Finally, we evaluated the performance of the developed prototype experimentally using different metrics.

Introduction

It is becoming increasingly important to support the deployment and execution of parallel and distributed systems across heterogeneous machines. One way to support the functional needs of such systems is by implementing them using Actors [1]. There is a growing number of implementations of Actors, including production languages such as Scala [2] which supports actors through its Akka library [3].

In this paper, we present a runtime environment will offer high-level primitives supported by a middleware implementing fundamental mechanisms required by distributed actor-based applications. Our solution to such support is implemented for Scala/Akka programming language1. The paper also presents a prototype implementation of a computational-intensive application, which is developed on the top of the runtime environment, to illustrate the ease with which new applications can be programmed.

Design and Implementation

The runtime environment enables programmers to write actor programs which can run on a large number of devices. Actors act the building blocks for an application which will be deployed on the top the runtime environment. The actors are connected in a dataflow to form the application. This simplifies actor migration between runtimes and matching of actor requirements with runtime capabilities. If the runtime does not meet the requirements posed by a currently deployed actor, then the actor will be automatically migrated to a runtime that can satisfy the requirements.

As shown in Figure 1, the system architecture consists of connected runtime environments running on heterogeneous computing devices, which support the execution of actors and managing their communication. We added a gatekeeper component to decide whether to deliver or postpone the delivery of messages for an actor according to some priority settings set by the programmer. When a sender actor sends a message –through its local runtime environment– to another actor hosted in a remote runtime environment, the gatekeeper dispatches the inbound message to the message dispatcher component, which in turns transport the message to the remote receiving actor using the routing actor.

Word counter application

We implemented a distributed word count application that calculates a running word count from a continuous stream of sentences. This application involves many of the structures and patterns required for more complex computation which can be supported by our runtime environment. We used the master slave architecture to show how our runtime environment can be used to break a computationally intensive task down into small subtasks for individual distributed actors to handle.

An Akka router is an actor which route messages to other actors called routees. The *router* actor is responsible for distributing tasks among its *routees* using different routing strategies. In this application, we used the round-robin-group routing algorithm in which messages are sent to *routees* in a round-robin fashion. The router actor can also deploy its created children on a set of remote hosts. We used two laptops and one Raspberry Pi 3 device to test the application. The Raspberry Pi served as the master node and the laptops served as worker nodes. First, the local runtime –which is deployed at the Raspberry Pi– created a master actor, which instructs the router to create and deploy 10 child actors on the other two remote hosts in a round-robin fashion.

In order to deploy *routees* remotely, we wrapped the *router* configuration in a RemoteRouterConfig by attaching the remote addresses and ports of the destination nodes. Once the *routees* are deployed at the remote

runtimes, they start executing the word-counting task and send results back to the master actor through the router. Particularly, worker actors read the input text line-by-line, count and index words before sending results back to the master actor. Once a new message arrives at the master node, the master actor forwards the indexed words to a mapper-aggregator worker which is responsible for mapping and aggregating these words from the two workers to display an up-to-date view of results, as shown in Figure 2.

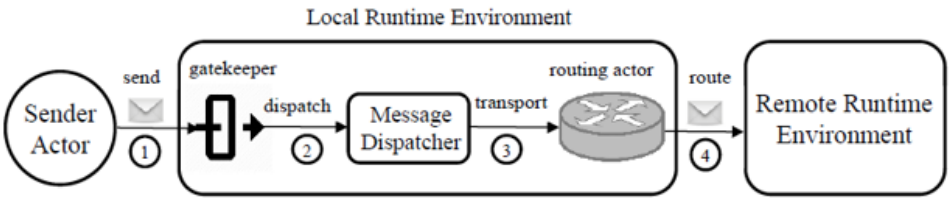

Figure 1: System Architecture.

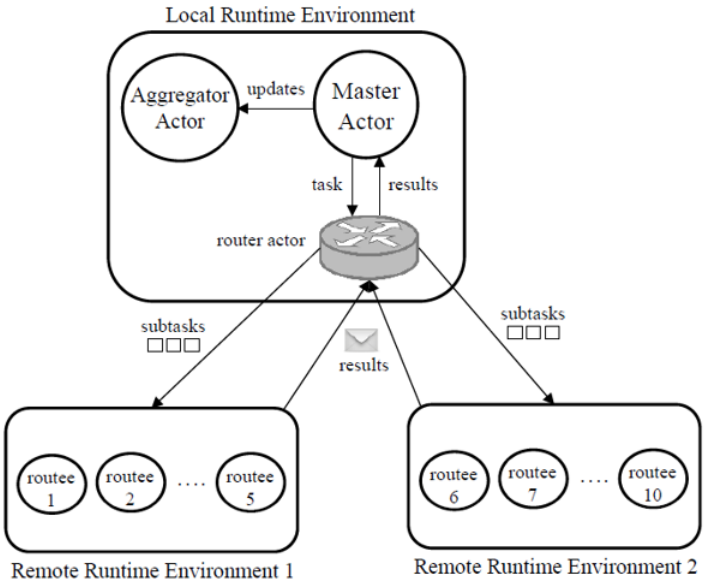

Figure 2: Word Counter Application.

[1]Available online: https://github.com/ahmed-pvamu/Actor-Based-Runtime-Environment-Distributed-Computing

Evaluation

This section presents our experimental evaluation of the runtime environment for performance and scalability. We used one Raspberry Pi 3 device with a 1.2GHz quad-core ARM Cortex-A53 processor and 1GB of RAM running Raspbian OS. We used two Windows laptops: the first is equipped with a 2.6GHz Intel Core™ i5-2540M CPU processor and 16GB of RAM, while the second is equipped with a 2GHz Intel R Core™ i3-5005U CPU processor and 8GB of RAM. We used Scala version 2.13.0-M1 with Akka version 2.5.19 running on JVM 1.8. Each experiment was carried out 10 times.

Fibonacci numbers sequence calculator

We carried out a simulation to calculate the Fibonacci sequence which is the series of numbers (0, 1, 1, 2, 3, 5, 8, ...) following this simple mathematics rule: xn = xn-1 + xn-2, where n is the Nth Fibonacci position and the Nth Fibonacci position is found by adding up the two numbers before it.

We ran a set of experiments to determine the effect of changing the Nth Fibonacci position on the total computational time. In this experiment, we calculated the Fibonacci numbers corresponding to Nth Fibonacci position for large values of N of up to 1M. We distributed these intensive computations across the Raspberry Pi and the two laptops equally so that they collectively calculate the target position. Specifically, we used the Raspberry Pi device as the master node and the two laptops as worker nodes.

Figure 3 show the results at the Raspberry Pi 3 and the laptops. It shows an increasing trend between the Nth Fibonacci position and the CPU computational time. Although the CPU time stays roughly under 15 seconds for the core-i3 laptop and stays roughly under 13 seconds for the core-i5 laptop, it jumps to approximately around 80 seconds for the Raspberry Pi device. This experiment demonstrates the benefits of leveraging the high-performance computing power of worker machines to process this highly intensive computation.

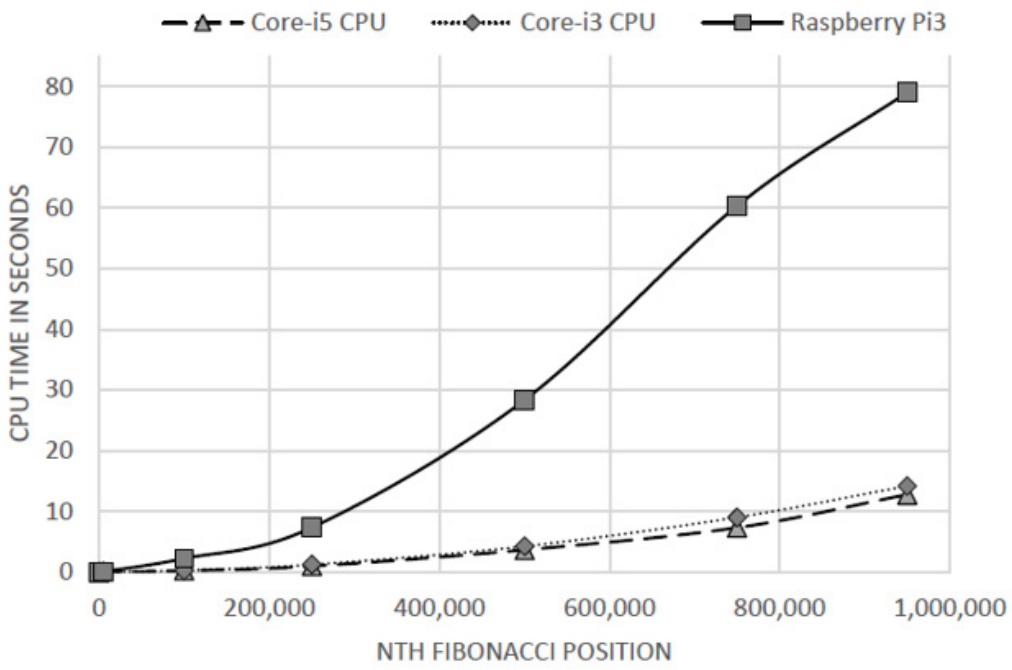

Figure 3: The Effect of Changing the N[th] Fibonacci Position on Computational Time.

We noticed that as the number of actors used in the computations increases, the total CPU time decreases until reaching a certain point, on which we could not observe noticeable differences in the computational time. This may be justified by the extra overheads of initiating the actors and communication delay. These results suggest that having a large number of actors is not necessary to improve the overall performance of intensive computations. Therefore, programmers need to find an equilibrium between the number of actors and the number of CPU cycles required to carry out the calculations.

Conclusion

This paper presented an approach to support the efficient execution of actor-based applications in Akka. Particularly, we described our design and implementation of a distributed runtime environment over which such class of distributed applications could be implemented relatively easily across heterogeneous computing machines. We carried out several sets of experiments for evaluating the performance and scalability of our system, paying particular attention to establishing the relationship between the distribution

of computations and the total computational time for executing them. The results showed that the computational time depends on various granularity characteristics of the systems, most notably the sizes of the computations assigned to individual machines.

Acknowledgement

None.

Conflict of Interest

No conflict of interest.

References

1. G Agha (1986) Actors: A Model of Concurrent Computation in Distributed Systems. Cambridge, MA, USA.
2. (2020) The scala programming language.
3. (2020) Akka programming language.