Journal of Advanced Zoology

ISSN: 0253-7214
*Volume **44** Issue **S-6 Year 2023** Page **586:592***

Asynchronous Execution Platform for Edge Node Devices

Pratyush Singh*, Rithik Kumar Jha D, Sai Krishna, Rohit Raj, Dr. K Amuthabala

School of CSE, Reva University
Email: emailtops369@gmail.com, rithikkumar53@gmail.com, saik9714@gmail.com, rohitraj13may@gmail.com, amuthabala.p@reva.edu.in

**Corresponding author's E-mail: emailtops369@gmail.com*

Article History

Received: 06 June 2023
Revised: 05 Sept 2023
Accepted: 29 Nov 2023

Abstract

A Asynchronous distributed execution platform which enables efficient and seamless task submissions on a remote node from a cluster of edge node devices using a reactive framework and provide real-time metrics of execution persisted on elastic database. Queues are used for job submission along with different compute units delivering the infrastructure for execution of submitted jobs. The proposed system is a generic framework that can be used in any enterprise web application where execution on a remote node is required. Through this we aim to provide an enterprise grade solution for task submission and management on a remote machine, using new, efficient technologies like SpringBoot and RabbitMQ. There is a demand in remote computing and huge workloads that cannot be executed on small local machines, our system can be used directly or indirectly by incorporated in other solutions.

Keywords: *Asynchronous Execution Platform, Queue Based delivery for execution, Generic execution platform, remote execution, Remote Execution Platform.*

1. Introduction

The exceptional growth in data processing has created a demand for a scalable and flexible platform to orchestrate and manage complexities of handling execution on cluster of compute nodes by providing seamless deployment, execution, testing and monitoring of user submitted scripts.

This project aims at running user-submitted tasks which can be a Python script or some other type of script. Our method involves a Master-worker design for handling and processing of requests. We chose to go with Master-Slave architecture to increase modularity.

The usual method of achieving this objective is having a centralized system accepting user jobs, but the problem here is the user is bothered with the intricate details of the way communication is achieved and configurations done between the centralized server and the user. As a user this is tedious and doesn't provide a generic platform to work upon. Hence, we propose the use of different technologies to make a seamless reactive framework that will accept scripts from the user and execute it on the desired compute unit, this will enable user to a whole lot of big and small applications like research applications, cloud, enterprise application.

The CRUD (Create, Read, Update, Delete) operations about user jobs are handled by a Management API which will be implemented using Spring Boot. The Management API will be responsible for handling requests from the user, collecting the status regarding an already submitted job. Spring is chosen for its substantial number of packages and classes which are easy to implement. Testing applications based on the Spring framework are comparatively easy.

As developers, we focus more on the convention part of application building and usually want to avoid the complexity that arises because of different deployment environments. This is made possible by Spring Framework as it helps build large enterprise applications without focusing much on the deployment side of things. The developer is not burdened with the tedious process of configuring the runtime environment.

Active Messaging Queue is based on the RabbitMQ, a popular message broker which helps application communicate, a message can contain anything of our choice, using this we will communicate between

the Master and worker sides of the system. RabbitMQ offers an assortment of highlights to let you compromise execution performance with dependability, including persistence, delivery affirmations, publisher confirms, and high availability, Flexible Routing, Clustering, Highly Available Queues, etc.

Literature Survey

The paper "A Lightweight Execution Framework for Massive Independent [1] Tasks" presents a "lightweight framework for executing many independent tasks efficiently on grids of heterogeneous computational nodes." The authors' system dynamically groups tasks of different granularities and dispatches the groups onto distributed computational resources concurrently.[1] Three procedures have been conceived to improve the productivity of computation and resource utilization. "One strategy is to pack up to thousands of tasks into one request. Another is to share the effort in resource discovery and allocation among requests by separating resource allocations from request submissions. The third strategy is to pack variable numbers of tasks into different requests, where the task number is a function of the destination resource's computability." This framework has been implemented in Gracie, a computational grid software platform developed by Peking University, and used for executing bioinformatics tasks. We describe its architecture.[1]

In the paper "Agent-based platform to support the execution of parallel tasks",[2] the authors had proposed "the use of agent technology to improve the management, flexibility, and reusability of grid-like parallel computing architectures."[2] The creators have introduced a universally useful agent-based architecture which can oversee and execute independent parallel tasks through one or a few heterogeneous computer networks – or even Internet hubs – abusing cutting edge agent versatility abilities.

The task execution requests are stored as Job class instantiations in a queue in which they wait until free node slots are available. The server informs via the requester service to the user about the task assignment and the number of resources available in the whole system at each moment. [2]"The process starts when the server agent module is initialized in conjunction with the Web server in a particular computer. The main container and the basic agents are created and the services initialized. From that moment, the server can receive task execution requests from the user through the request server and to register offers from computer nodes."[2]

In the paper WorkflowDSL: Scalable Workflow Execution with Provenance [4] by THARIDU FERNANDO The objective of this paper is to implement a parallel workflow execution framework for scientific workloads without any specialized code. The Author was motivated by the knowledge gap between the scientific domain experts and the technical experts providing means of the execution. There is no fruitful result until and unless this gap is reduced to an understandable level. To overcome this problem Author mentions the use of Sintific Workflow management Systems that were created specifically, SWfMSs provide scalable execution, where workflows "can exploit more computing resources as needed (i.e., using a cluster or cloud) which will significantly lower the workflow execution times. Normally, the workflow tasks are modelled as a Directed Acyclic Graph (DAG) where the vertexes and edges represent tasks and data dependencies respectively." Using such workflow representations and using SWfMSs it is possible to execute them in parallel by applying mechanisms such as "functional and data parallelism."

In Clustered Workflow Execution of Retargeted Data Analysis Scripts [3]

By Daniel L. Wang, Charles S. Zender, and Stephen F. Jenks University of California, Irvine Irvine, California, USA. The authors emphasize the huge volume of data created from the introduction of supercomputing and the bottleneck of current computing infrastructure.

Authors of the paper have implemented, a workflow execution framework to provide computation services integrated with data services. The target application is geoscience data analysis, whose data-intensive and computationally light characteristics categorize a class of computing that is not well-served by existing solutions and will grow in importance as data production continues to explode.[3]A phrasher converts the script into a DAG (Directed Acyclic Graph). After the user has submitted the task as specified in the framework, the engine begins its execution phase, in which it removes commands from the ready list(A list where all the user-submitted scripts are waiting and dispatch them to worker node with idle CPU slots.[3] "The interface is graphical and simple enough to empower scientists with the ability to construct ad-hoc workflows without worrying about execution."[3].

Summarization of survey

- Most of the work in the development of a execution framework is related to workflow management systems, there is a lack of generic model for Independent task loads.
- Huge gap in knowledge between Domain Experts and Technical Experts.
- Most of the proposed system(s) focuses on DAG to form workflow and execution pipeline is left dangling without any major research work.
- Several models include centralized execution platform model.
- Developers lack the freedom to choose the resource.
- Other Papers depicts the importance of processing scripted tasks but fail to present a regular framework which solely focuses on the efficiency of task execution.
- No research which addresses the problem of Tedious function implementations that is in the way of users who just want to run Independent tasks or orchestrate their own flow.
- Whole class of users like scientific researcher, independent analysts, etc. are left unserved with a need of some regular framework for execution of scripts.
- To conclude there is a lack of some regular model for execution platforms, that can orchestrate, manage and monitor the independent user submitted task(s) and focuses on doing just that with good efficiency.
- Lack of research where the proposed system doesn't bother the user with the task of setting up the complex infrastructure before execution of scripts.
- Research available for some applications of executor platform like workflow management and scientific analysis but no generic framework is proposed yet.

Methodology

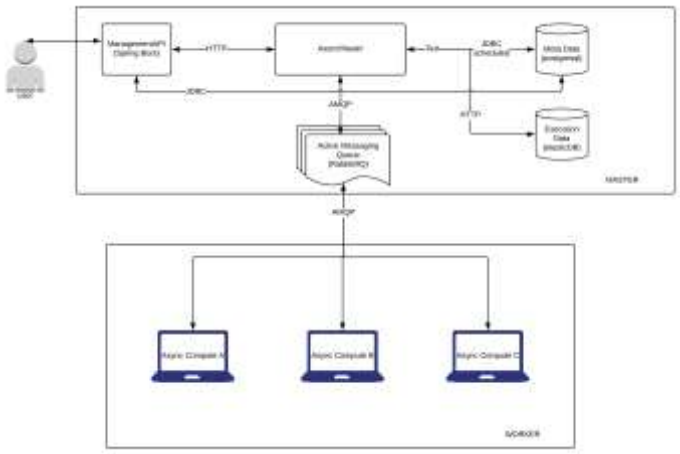

Fig. 1 Core Architecture

Java is used as the base language, chosen because of its superior capabilities in object-oriented programming as well as adequate support for required computing libraries. Java is accepted as an industry standard when it comes to developing business applications. Java supports all the toolkits and libraries that are required for this project.

A. Master-Worker

Master-Worker design is proposed where the worker is a cluster of computing units that will perform the executions as per the instructions received from the Master's side. The Master and the Worker (Client) will be packaged together, this means that in the same modular architecture we can specify which is acting as master and which device is acting as a worker. This binding can be done during the initial setup process.

B. The Master

The user will hit the management API through HTTP with all the relevant job details like job name, script(s) path, and a compute unit of their choice. The Management API will then receive the HTTP request from the user and places the metadata for the said job in the Database. It also stores the scripts in a shared directory that is constantly checked for any unexecuted jobs by the Asynchronous master.

The Asynchronous master is scheduled to check the database at a specified interval. The asynchronous master will read the job from the database and will upload the script for execution to the worker's side and then one of the units will execute the script, after execution the metrics and the result (if available) are sent back to the Async Master. The Asynchronous master will communicate/listen from an active messaging queue that is implemented using RabbitMQ, a popular open-source message broker. The Queue(s), The Asynchronous master, and The Compute unit cluster will be using AMQP (Active Messaging Queuing Protocol) to communicate. AMQP is a well-known generic protocol for messaging.

The Management API, Spring Boot is used for its development. Spring boot is a web application toolkit designed on top of the Spring Framework. Spring is a well-known framework that is used extensively in the industry to create fast and robust web applications. Spring boot is used to create microservices in our project. Spring Boot will help to make our program(s) efficient with the use of small code and will give us the ability to use JDBC with full efficacy. Spring Boot uses the Spring Framework as a foundation and improves on it. It simplifies Spring dependencies that will allow us to run our application straight from a command line, no application container required. Spring Boot will mostly help in monitoring several components along with the explicit configuration of these components [11-16].

The Flow of Operation

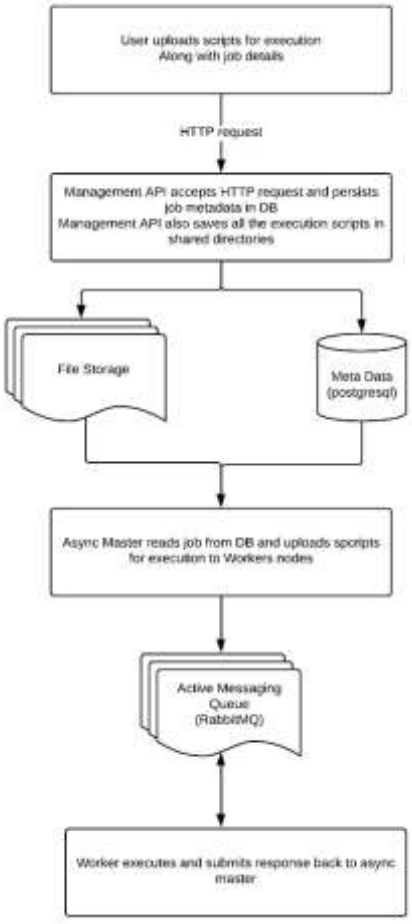

Fig. 1 Flow of Operation

Custom Message Structure(s)

There are two custom message structures that are defined in the architecture. Both the message structures implements java.io.Serializable, this enables the architecture to be secure and it lets Java handle the integrity , authenticity, and security of the message. Even if the Message Structure is known there can be NO interception of data while it's being transferred.

The messages use project Lombok for the use of Getter Setter setup. Lombok enables us to correctly initialize and send the message efficiently.

The first-class Message is the one that Master Node will send to one of the Worker Nodes, this is named as CommandSend in the architecture. CommandSend includes the worker's name, the Command, file path of the script/workload. Worker name is also the Queue name for the respective workers.

The second-class of message is the acknowledgment message which will be sent after the execution is completed by the Worker to the master. This acknowledgment message contains the worker's name, the output, and time taken for execution. This is named as CommonAck in the architecture.

The Queue Structure

The custom message structure defined will be transmitted to the working nodes using Routing methods of RabbitMQ. Each node will have its own Queue which will be listening for messages from the Master Side. This method can also be called as selective subscribe method this will enable the nodes to communicate when necessary.

To accommodate the communication back from the nodes to the Producer in this case the Async Master, the architecture will use a single callback Queue using which the back communication will be achieved. The message can be transmitted to a single queue from the nodes after the script has been executed or when requested by the user, the statistics, along with other necessary data will be aggregated into another custom message structure.

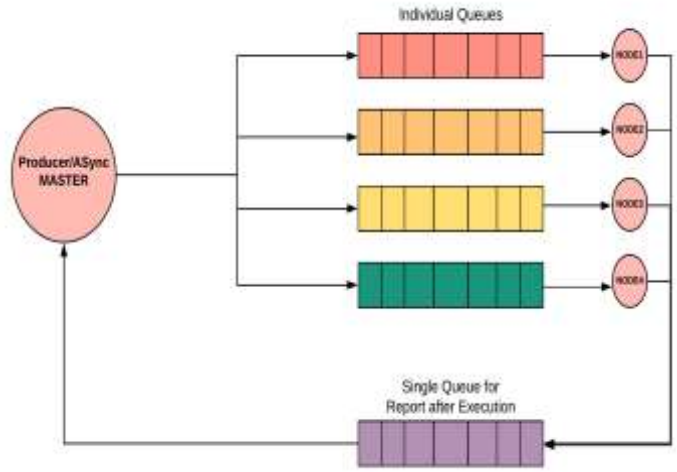

Fig. 2 Queue's Arrangement

This way we achieve seamless communication between the nodes, given that the initial setup process with the working side of the nodes is done. RabbitMQ allows for this behavior.

Execution Engine

The execution engine in the architecture is responsible for running the command part from the message in the shell. To achieve this, we are using conventional java process builders.

As soon as the command is sent a timer is started to calculate absolute execution time that will be used in the common acknowledgment message structure. A buffered reader is used to read the output from the shell of the worker sub process that was created earlier. Before the buffered reader starts the process right after execution, the timer that was started earlier is stopped. Then the execution engine sets data for the acknowledgment which includes worker's name, time elapsed, and the output.

3. Results and Discussion

We have achieved execution of a simple python command the command can be any acceptable shell command that is provided by the user. After the script is served to the master side and a worker name, the engine automatically executed the provided instructions and return the output, time elapsed and the name of the worker. A sample of the output generated by the worker using a simple python file to output a triangular pattern is illustrated in below figure.

Comparing to existing solutions like SCOM, Shiny Framework, or even using SSH to execute a task on a remote machine the time elapsed in the whole operation is greater than half a second (Mostly because of the required manual configurations), Whereas our proposed system is very fast, producing experimental results in less than 0.07 seconds because there is no need to configure apart from the initial setup.

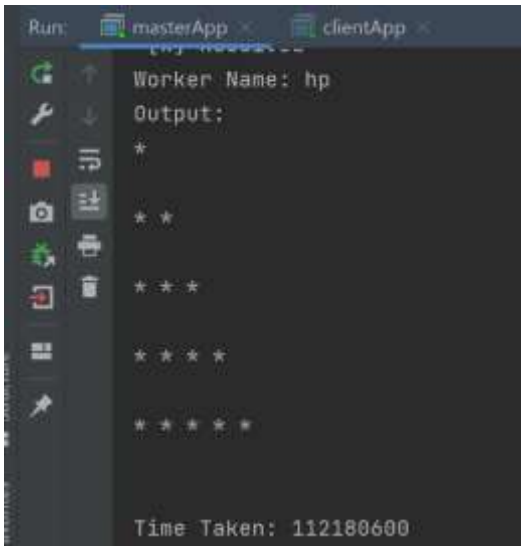

Fig. 4 Initial Output

To calculate the time taken by the current version of the framework can be calculated by the time taken since initiation subtracted by time taken to run the script by worker.

Total Time taken in test: 59,123,400 nanoseconds

Time Taken by test program: 21,931,600 nanoseconds

Time taken by Framework: 37,191,800 nanoseconds.

4. Conclusion

Through this proposed architecture, we will achieve execution of user-submitted jobs on one of the compute unit from a cluster of edge node devices, using a reactive framework along with Queues and Statistics of job execution is returned to the user. Hence, developing a generic application model which can be further developed upon.

References:

AH Omar Baabood, Prajoona Valsalan, Tariq Ahmed Barham Baomar, IoT Based Health Monitoring System, Journal of Critical Reviews , Vol. 7, Issue. 4, pp. 739-743, 2020.

Bindhia K Francis, Suvanam Sasidhar Babu, Predicting academic performance of students using a hybrid data mining approach, Journal of Medical Systems, 43:162, 2019. https://doi.org/10.1007/s10916-019-1295-4

David Sánchez, David Isern, Ángel Rodríguez-Rozas, Antonio Moreno-"Agent-based platform to support the execution of parallel tasks" - 2011Intelligent Technologies for Advanced Knowledge Acquisition (ITAKA), Department of Computer Science and Mathematics, University Rovira i Virgili,

Erik Elmroth, Francisco Hern´andez, and Johan "A Light-Weight Grid Workflow Execution Engine Enabling Client and Middleware Independence" – 2018 - Sweden

Li Hui, Yu Huashan, Li Xiaoming -"A Lightweight Execution Framework for Massive Independent Tasks" – 2008 Peking University, Beijing 100871, P.R.China

Maciej Malawski "Towards Serverless Execution of Scientific Workflows – HyperFlow Case Study" - AGH University of Science and Technology Department of Computer Science Krakow, Poland - 2019

P. Amuthabala and Dr.M. Mohanapriya, "Pattern Based Technique in Evaluation of Data Quality on Complex Data" Department of Computer Science and Engineering, Karpagam Academy of Higher Education, Karpagam University, Coimbatore, Department of Computer Science and Engineering, Karpaga – 2020

P. Amuthabala and M. Mohanapriya, " Cost Effective Framework for Complex and Heterogeneous Data Integration in Warehouse" – Department of Computer Science and Engineering, Karpagam Academy of Higher Education, Coimbatore, India - 2020

P. Amuthabala, R. Santosh "Robust analysis and optimization of a novel efficient quality assurance model in data warehousing", Department of Computer Science and Engineering, Karpagam Academy of Higher Education, Coimbatore, India. – 2020

PK Sadineni, Comparative Study on Query Processing and Indexing Techniques in Big Data, 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), pp. 933-939, 2020.

PM Surendra, S Manimurugan, A New Modified Recurrent Extreme Learning with PSO Machine Based on Feature Fusion with CNN Deep Features for Breast Cancer Detection, Journal of Computational Science and Intelligent Technologies, Vol. 1, Issue. 3, Pp. 15-21, 2020.

Sajay KR, Suvanam Sasidhar Babu, Vijayalakshmi Yellepeddi, Enhancing The Security Of Cloud Data Using Hybrid Encryption Algorithm, Journal of Ambient Intelligence and Humanized Computing, 2019. https://doi.org/10.1007/s12652-019-01403-1

Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. "Serverless Computation with OpenLambda" – 2016 - University of Wisconsin, Madison

Sudhan Murugan Bhagavathi, Anitha Thavasimuthu, Aruna Murugesan, Charlyn Pushpa Latha George Rajendran, A Vijay, Raja Laxmi, Rajendran Thavasimuthu, Weather forecasting and prediction using hybrid C5.0 machine learning algorithm International Journal of Communication Systems, Vol. 34, Issue. 10, Pp. e4805, 2021.

Tarragona, Catalonia, Spain & Center for Mathematics and its Applications, Department of Mathematics, Instituto Superior Técnico, Lisboa, Portugal

THARIDU FERNANDO "WorkflowDSL: Scalable Workflow Execution with Provenance" – 2018

Wang, DL Zender, CS Jenks "Clustered workflow execution of retargeted data analysis scripts" – 2008-University of California, Irvine, California, USA