COMPUTERIZED AUTOMATIC SYSTEMS

IMPLEMENTATION OF AN APACHE SPARK COMPUTING CLUSTER BASED ON RASPBERRY PI MICROCOMPUTERS

Halyna Vlakh-Vyhrynovska, PhD, As. Prof.,
Bohdan Boretskyi, BSc Student
Lviv Polytechnic National University, Ukraine,
e-mail: bohdan.boretskyi.ir.2021@lpnu.ua

https://doi.org/10.23939/istcmtm2025.02.092

Abstract. The paper presents the implementation of an Apache Spark distributed computing cluster based on Raspberry Pi microcomputers. The solution consists of three Raspberry Pi 4 devices (one master node and two worker nodes), each equipped with 8 GB of RAM and a high-speed network connection. The cluster configuration was optimized by adjusting the SPARK_WORKER_MEMORY and SPARK_WORKER_CORES parameters to maximize the use of available hardware resources. Secure communication between nodes was established through authentication using 4096-bit SSH keys. The functionality of the cluster was tested using a test application that demonstrated efficient distribution of computational load across nodes. The developed solution costs $400, which is four times less than the cost of using equivalent cloud resources for one year. The results show that the Raspberry Pi cluster provides all the necessary capabilities for practical learning of distributed computing technologies, offering physical access to all system components at a low cost.

Key words: Apache Spark, distributed computing, Raspberry Pi, microcomputers, cluster, big data processing.

1. Introduction

Distributed computing is a fundamental component of modern information technologies, enabling efficient processing of big data and execution of complex algorithms [1]. Commercial cloud solutions often obscure the low-level aspects of cluster operation, making it difficult to develop a deep understanding of distributed system architecture.

This paper explores the feasibility of building an Apache Spark computing cluster based on Raspberry Pi microcomputers. The proposed approach combines low-cost hardware with flexible configuration options, allowing the cluster to be adapted to various research tasks.

The relevance of this solution lies in the value of direct access to the system's hardware components, which enables the study of all layers of its organization – from the physical arrangement of nodes and network infrastructure to software configuration and parameter optimization.

A full cluster deployment "from scratch", including operating system installation, network setup, secure access provisioning, and Apache Spark stack configuration, contributes to a deeper understanding of component interaction in a distributed environment and the principles of its operation [2].

2. Drawbacks

Modern approaches to learning distributed computing technologies have several limitations:

1. Hiding low-level infrastructure details and lack of physical interaction with hardware components in cloud-based solutions [3].
2. High cost of using cloud services compared to deploying physical computing resources [4], [5].
3. Inability to monitor thermal conditions and energy consumption during cloud-based computations [6].
4. Limited control over network infrastructure configurations [7].

3. Goal

The aim of this paper is to develop a cost-effective, energy-efficient, and fully functional computing cluster for distributed processing using Apache Spark based on Raspberry Pi microcomputers. To achieve this goal, the following tasks were undertaken:

– Design and build the architecture of a cluster system based on Raspberry Pi microcomputers that implements the required distributed computing functionality.

– Set up and configure the network infrastructure to ensure stable communication between cluster components and their access to the Internet.

– Configure the Apache Spark software components with parameters optimized for efficient use of the limited hardware resources of Raspberry Pi devices.

– Set up access to cluster nodes using asymmetric SSH keys.

– Perform experimental testing of the cluster using a basic data processing task to verify its functionality and efficiency.

– Compare the costs of the implemented solution with those of equivalent cloud-based services.

4. Literature Review

Distributed computing is a data processing model in which tasks are executed in parallel across multiple interconnected nodes to improve performance and scalability when working with large volumes of data [1]. In such systems, tasks are distributed among nodes, enabling scalable computation, faster data processing, and uninterrupted operation even if individual components fail. A cluster-based architecture with centralized resource and task management is most commonly used [8].

One of the leading frameworks for distributed computing is Apache Spark – a high-performance open-source engine that provides high-level APIs for Java, Scala, Python, and R [9]. Unlike traditional solutions that store intermediate results on disk [10], Spark uses in-memory processing, significantly increasing data processing speed. Its architecture includes the Spark Core and a number of specialized libraries: Spark SQL for structured data processing; MLlib for machine learning; GraphX for graph computations; and Structured Streaming for real-time data processing – making the platform suitable for a wide range of tasks.

The concept of Resilient Distributed Datasets (RDD) [11] forms the foundation of Spark's architecture. RDDs provide an abstraction that enables fault tolerance without writing intermediate results to disk, considerably enhancing system performance compared to earlier implementations of distributed computing. Spark executes iterative data processing tasks up to 26 times faster than MapReduce, which makes it a highly effective solution for machine learning and interactive analytics.

The architecture of Apache Spark follows a "master-worker" model, where the Cluster Manager plays a central role by coordinating communication and distributing computational tasks among worker nodes running on separate machines [11], [12]. This approach ensures the system's scalability and fault tolerance under increasing workloads. The platform supports several deployment modes, including Standalone, YARN, Mesos, and Kubernetes, allowing infrastructure to be adapted to technical constraints and user requirements.

Existing Raspberry Pi-based cluster implementations confirm the effectiveness of this platform for distributed computing. For instance, TopADDPi is an energy-efficient cluster for topology optimization tasks, built using open-source software that demonstrates high performance at minimal cost [13]. The study presented in [14] describes a cluster of 25 Raspberry Pi 2B devices, designed to explore the relationship between energy consumption and computational performance in an educational setting. Moreover, [13] discusses an environment that integrates Raspberry Pi with Google Colab for teaching the fundamentals of parallel and heterogeneous computing, highlighting the flexibility and accessibility of this hardware platform.

Further evidence of the educational value of such solutions is provided in [15], where Raspberry Pi clusters are effectively used to teach core concepts of parallel and distributed computing. The proposed teaching modules include a preconfigured software environment that enables quick deployment of fully functional clusters without complex infrastructure setup. Practical applications demonstrated improved student understanding of fundamental cluster system principles and the development of hands-on skills in the field.

5. Methodology

5.1. Operating system configuration for cluster nodes

The official Raspberry Pi Imager tool was used to flash Raspberry Pi OS (64-bit) onto MicroSD cards for all cluster nodes. During the installation process, key parameters were preconfigured, including user credentials and SSH access activation.

Static IP addressing for the cluster nodes was implemented by modifying the cmdline.txt file directly on the MicroSD cards. The IP address is defined using the following parameter:

ip=<static-ip>::<gateway-ip>:<netmask>:<hostname>:<interface> hostname=<system-hostname>

where:

– <static-ip> -the static IP address assigned to the node
– <gateway-ip> - the IP address of the router/gateway for Internet access
– <netmask> - defines the subnet range
– <hostname> - network identification name
– <interface> -specifies the network interface (e.g., eth0 for Ethernet)
– hostname=<system-hostname> - sets the OS hostname

The cmdline.txt file was configured as follows:

For the master node:

ip=192.168.137.2::192.168.137.1:255.255.255.0:raspberrypi-master hostname=raspberrypi-master

For worker node 1:

ip =192.168.137.3::192.168.137.1:255.255.255.0:raspberrypi-worker1 hostname=raspberrypi-worker1

For worker node 2:

ip =192.168.137.4::192.168.137.1:255.255.255.0:raspberrypi-worker2 hostname=raspberrypi-worker2

This configuration ensures that each node retains its static IP address after reboot and maintains consistent network identification within the cluster.

5.2. Physical Configuration of the Cluster

The physical setup of the cluster involved mounting three Raspberry Pi 4 devices on a multi-level stand to ensure efficient cooling (Fig. 1). Each device was powered by a dedicated 5V/3A power supply. The network infrastructure was implemented using a 5-port Mercury router connected via Cat 6 Ethernet cables, providing speeds of up to 1 Gbps. One Raspberry Pi was configured as the master node, while the other two served as worker nodes.

A client laptop was also connected to the network for managing the cluster and providing Internet access through the Internet Connection Sharing (ICS) feature. This configuration allowed all Raspberry Pi nodes to use the laptop's Internet connection. The correctness of the network connections was verified by the activity indicators on the router's ports.

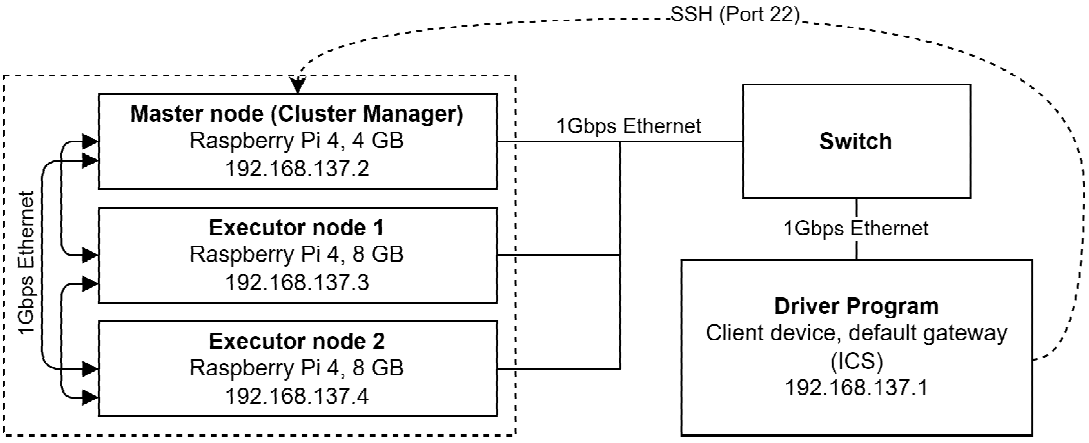

Fig. 1. Physical and network architecture of the cluster

5.3. SSH configuration and secure access

Asymmetric SSH key-based authentication was configured to manage the cluster from the client laptop. Initially, password-based authentication was used to connect the client device to the master node (ssh 192.168.137.2). RSA keys with a length of 4096 bits were then generated on the client device (ssh-keygen -t rsa -b 4096), after which the public key was transferred to the master node and added to the list of authorized keys. The same steps were repeated for each of the two worker nodes.

To ensure seamless interaction among all cluster components, passwordless authentication was additionally configured. A separate pair of SSH keys was generated on the master node, and the public key was distributed to both worker nodes using the ssh-copy-id utility. The correctness of the configuration was verified through test connections to each node.

5.4. Internet connection sharing configuration

To provide Internet access to the Raspberry Pi devices, the Internet Connection Sharing (ICS) feature available in the Windows operating system was used. In the settings of the network adapter with an active Internet connection, the option "Allow other network users to connect through this computer's Internet connection" was enabled. This ensured routing and address translation between the cluster's local network (192.168.137.0/24) and the external network. On all Raspberry Pi nodes, the default gateway and DNS server were set to 192.168.137.1. Internet access was then used to update system packages and install the necessary software, including Apache Spark.

5.5. Software installation

The necessary software was deployed on all cluster nodes after updating the system repositories (apt upgrade). In addition to OpenJDK 17 and Python 3.12, Apache Spark version 3.5.5 was installed as the core framework for distributed computing. To ensure proper system operation, relevant environment variables were configured: SPARK_HOME was set to point to the installation directory, the system PATH variable was updated to allow access to Spark utilities from the command line, and JAVA_HOME was defined to ensure correct interaction with the Java Virtual Machine [16].

5.6. Spark cluster configuration

To create an efficient computing environment, the cluster nodes were configured according to their designated roles. On the master node, which functions as the Cluster Manager, the following parameters were set: SPARK_MASTER_HOST with the node's IP address, SPARK_MASTER_PORT (7077), and SPARK_DRIVER_MEMORY (3 GB) to ensure effective management of computational processes.

On the worker nodes, which have more RAM (8 GB), the parameters SPARK_WORKER_MEMORY (6 GB) and

SPARK_WORKER_CORES (4) were configured [16]. These values are optimal for maximizing the use of available hardware resources for parallel computations. All configuration parameters were added to the spark-env.sh file on the respective nodes to retain settings between cluster startup sessions and ensure stable operation within the cluster environment.

5.7. Launching and testing of cluster

The Apache Spark cluster can be launched in two different ways.

The first method involves starting the components step by step. First, the Master service is launched on the master node:

start-master.sh

After the successful launch of the Master node, the Worker process is initialized on each worker node with the specified address of the Master:

start-worker.sh spark://192.168.137.2:7077

This command connects the worker node to the driver via the spark:// protocol on port 7077.

An alternative and faster method is to use a command that launches all cluster components simultaneously:

start-all.sh

This command automatically starts the Master process on the local machine and Worker processes on all nodes listed in the conf/workers configuration file. This significantly simplifies cluster startup, especially during regular use. The successful deployment of the cluster can be confirmed via the Spark Master web interface available at http://192.168.137.2:8080, which provides information about the connected worker nodes and their computational resources.

It is worth noting that the start-worker.sh command also accepts additional parameters for more flexible resource configuration, such as -c to specify the number of CPU cores and -m to define the amount of memory.

6. Conclusions

6.1 Functional Testing

The implemented computing cluster was successfully tested by running a basic application that performs a parallel computation of the sum of numbers from 1 to 999.999. The Python code used for the test is shown below:

```
import logging
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum

logging.basicConfig(level=logging.INFO,
format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger("ClusterTest")

def main():
    os.environ['SPARK_LOCAL_IP'] = '192.168.137.1'

    spark = SparkSession.builder \
        .appName("HelloWorldInSpark") \
        .master("spark://192.168.137.2:7077") \
        .config("spark.driver.host", "192.168.137.1") \
        .getOrCreate()

    nums_df = spark.range(1, 1000000)
    result_df = nums_df.select(sum("id").alias("sum"))

    logger.info(f"Result:")
    result_df.show()

    spark.stop()

if __name__ == '__main__':
    main()
```

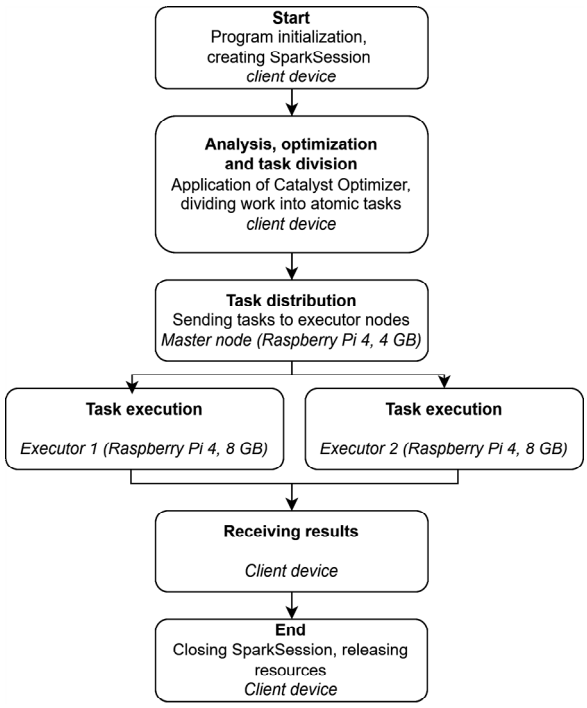

Fig. 2. Execution algorithm of the Spark application

The application was executed from a client device (a personal laptop) using the PySpark library, which was configured to connect to the master node of the cluster. Spark successfully distributed the computational load across the worker nodes and completed the data processing, returning the result 499 999 500 000 to the client device. This confirmed the functionality of the

deployed cluster. Task execution monitoring via the Spark Master web interface demonstrated efficient utilization of both worker nodes' resources.

The execution of the Spark application on the developed cluster follows a well-defined algorithm (Fig. 2), illustrating the interaction between system components [12]. The process begins on the client device with the creation and optimization of a logical execution plan. This is followed by the distribution of atomic tasks among the worker nodes via the Master node, parallel execution of computations on the worker nodes, and aggregation of results on the client device.

6.2. Cost analysis: raspberry pi cluster vs aws cloud environment

One of the key advantages of the assembled cluster is its low cost compared to commercial cloud-based solutions. Table 1 presents the cost structure for building such a system.

Table 1. Cost of implementing the Raspberry Pi Cluster

Component	Quantity	Unit Price, $	Total Price, $
Raspberry Pi 4 (8 GB RAM)	2	95	190
Raspberry Pi 4 (4 GB RAM)	1	75	75
MicroSD cards (64 GB)	3	15	45
Power Supplies (5V/3A)	3	10	30
Router	1	25	25
Ethernet Cables (Cat 6)	3	5	15
Multi-level Cooling Stand	1	20	20
Total			$400

To evaluate the economic feasibility of the solution, a cost comparison was conducted with equivalent computing resources in the AWS EC2 cloud environment. Equivalent virtual machines based on the ARM-based Graviton2 architecture were selected:

– for the master node (4 GB RAM, 4 cores) – t4g.medium

– for the worker nodes (8 GB RAM, 4 cores) – t4g.large

The use of ARM processors in the t4g series allows a more accurate comparison with Raspberry Pi than traditional x86-based architectures. Although the Raspberry Pi features 4 physical cores while the t4g instances have only 2 vCPUs [17], the newer Graviton2 cores with higher clock speeds deliver comparable performance.

Based on AWS pricing [17] – $0.0336/hour for t4g.medium and $0.0672/hour for t4g.large – the annual cost of operating an equivalent cluster configuration is summarized in Table 2:

Table 2. Annual Cost of Equivalent AWS EC2 Virtual Machines

Component	Monthly cost, $	Annual cost, $
t4g.medium (master)	24.54	294.53
t4g.large (worker) × 2	49.09 × 2	1178.10
Data transfer costs (approx.)	15.00	180.00
Total		$1652.63

The cost comparison shows that the Raspberry Pi cluster, with a total price of $400, is approximately four times cheaper than equivalent AWS cloud services over the course of one year.

7. Conclusions

As a result of this study, a functional Apache Spark computing cluster based on Raspberry Pi microcomputers was successfully implemented. The deployed cluster demonstrated its practical applicability for studying distributed computing technologies and conducting research on a cost-effective platform.

The developed cluster consists of three Raspberry Pi 4 nodes, including one master node and two worker nodes. Each device in the cluster configuration is equipped with 64 GB of storage on MicroSD cards. The worker nodes are equipped with 8 GB of RAM and quad-core Cortex-A72 processors.

To ensure efficient operation, the Apache Spark configuration parameters were optimized. The cluster's network infrastructure was implemented using a 5-port Mercury router. For secure access to the nodes, authentication based on 4096-bit asymmetric SSH keys was configured.

The functionality of the developed cluster was validated by successfully running a test Spark application that performs parallel computation of the sum of numbers from 1 to 999,999. Execution analysis confirmed that the computational load was effectively distributed across the worker nodes and that the results were correctly aggregated on the master node.

From a financial perspective, the Raspberry Pi cluster represents a viable alternative to cloud-based solutions, as its total cost of approximately $400 is more than four times lower than the annual cost of equivalent AWS EC2 virtual machines.

The key value of the developed cluster lies in its accessibility as an educational tool for hands-on learning of distributed computing principles. Unlike virtual or cloud-based solutions, a physical cluster provides direct interaction with all system components.

Conflict of interest

The authors declare that there are no financial or other potential conflicts of interest regarding this work.

References

[1] F. Dai, M. A. Hossain, and Y. Wang, "State of the Art in Parallel and Distributed Systems: Emerging Trends and Challenges", *Electronics*, vol. 14, No. 4, p. 677, Feb. 2025. DOI: 10.3390/electronics14040677.

[2] V. Thesma, G. C. Rains, and J. Mohammadpour Velni, "Development of a Low-Cost Distributed Computing Pipeline for High-Throughput Cotton Phenotyping", *Sensors*, vol. 24, No. 3, p. 970, Feb. 2024. DOI: 10.3390/s24030970.

[3] A. Alakuu and D. K. Dake, "Cloud Computing in Education: A review of Architecture, Applications, and Integration Challenges", *IJCA*, vol. 186, No. 66, pp. 49–65, Feb. 2025. DOI: 10.5120/ijca2025924472.

[4] S. Younus, K. Kumar, I. A. Kandhro, A. A. Laghari, and A. Ali, "Systematic Analysis of On Premise and Cloud Services", *IJCC*, vol. 13, No. 3, p. 10063641, 2024. DOI: 10.1504/IJCC.2024.10063641.

[5] A. A. Abdulle, A. Farah Ali, and R. H. Abdullah, "Cost-Benefit Analysis of Public Cloud Versus In-House Computing", *IJETT*, vol. 70, No. 6, pp. 300–307, Jun. 2022. DOI: 10.14445/22315381/IJETT-V70I6P231.

[6] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: a survey on software technologies", *Cluster Comput*, vol. 26, No. 3, pp. 1845–1875, Jun. 2023. DOI: 10.1007/s10586-022-03713-0.

[7] G. Agapito and M. Cannataro, "An Overview on the Challenges and Limitations Using Cloud Computing in Healthcare Corporations", *BDCC*, vol. 7, No. 2, p. 68, Apr. 2023. DOI: 10.3390/bdcc7020068.

[8] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, "Exploring the Potential of Distributed Computing Continuum Systems", *Computers*, vol. 12, No. 10, p. 198, Oct. 2023. DOI: 10.3390/computers12100198.

[9] "Spark Overview". Apache Software Foundation [Online]. Available: https://spark.apache.org/docs/latest/

[10] P. Sewal and Hari Singh, "Performance Comparison of Apache Spark and Hadoop for Machine Learning based iterative GBTR on HIGGS and Covid-19 Datasets", *SCPE*, vol. 25, no. 3, pp. 1373–1386, Apr. 2024. DOI: 10.12694/scpe.v25i3.2687.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica., "Spark: Cluster Computing with Working Sets", 2010 [Online]. Available: https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf

[12] N. Ahmed, A. L. C. Barczak, M. A. Rashid, and T. Susnjak, "A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters", *J. Big Data*, vol. 8, no. 1, p. 107, Dec. 2021. DOI: 10.1186/s40537-021-00499-7.

[13] Z.-D. Zhang *et al.*, "TopADDPi: An Affordable and Sustainable Raspberry Pi Cluster for Parallel-Computing Topology Optimization", *Processes*, vol. 13, No. 3, p. 633, Feb. 2025. DOI: 10.3390/pr13030633.

[14] M. Cloutier, C. Paradis, and V. Weaver, "A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement", *Electronics*, vol. 5, No. 4, p. 61, Sep. 2016. DOI: 10.3390/electronics5040061.

[15] E. Shoop, S. J. Matthews, R. Brown, and J. C. Adams, "Hands-on parallel & distributed computing with Raspberry Pi devices and clusters", *Journal of Parallel and Distributed Computing*, vol. 196, p. 104996, Feb. 2025. DOI: 10.1016/j.jpdc.2024.104996.

[16] "Spark Configuration." Apache Software Foundation [Online]. Available: https://spark.apache.org/docs/latest/configuration.html

[17] "Amazon EC2 On-Demand Pricing." AWS [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/