**SIMA SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**DEVELOPING CLOUD BASED APPLICATIONS IN DATA BASE**

**A CAPSTONE PROJECT REPORT**

**CSA1592 – Cloud Computing and Bid Data Analytics for Web Services**

*Submitted in the partial fulfillment for the award of the degree of*

**Bachelor of Engineering**

**IN**

**INFORMATION TECHNOLOGY**

Submitted by

**R . ANJAN KUMAR REDDY (192210162)**

**S.NO: 28**

Under the Supervision of

**ARUL RAJ A M**

JULY 2024

**AIM : -**

Developing cloud-based applications aims to leverage the scalability, flexibility, and cost-efficiency of cloud computing. It involves designing software that runs on cloud infrastructure, enabling seamless access and management of resources over the internet. Key objectives include ensuring high availability, robust security, and optimized performance, while providing the ability to scale resources dynamically based on demand.

**SCOPE:**

The scope of developing cloud-based applications includes utilizing cloud services for infrastructure management and ensuring scalability and cost-efficiency. It involves robust security measures, implementing CI/CD pipelines, and developing using microservices architecture. Efficient data management and creating APIs for seamless integration are crucial. Monitoring, logging, and ensuring high availability and performance for an optimal user experience are also key aspects.

**1.PROBLEM STATEMENT:**

Developing cloud-based applications presents challenges in managing infrastructure scalability and cost-effectiveness. Ensuring robust security and compliance while maintaining high availability and performance is crucial. Integrating various services and automating deployment through CI/CD pipelines can be complex. Efficient data management and real-time monitoring are necessary to prevent downtime and ensure reliability. Additionally, providing a seamless user experience across diverse devices and locations requires careful planning and execution.

**2. PROPOSED ARCHITECTURE DESIGN:**

Proposed Architecture Design for optimizing resource allocation infrastructure and service environmental.

**2.1 Key Components**

**Layer 1: Data Collection and Analysis**

**A)Sensors and IoT Devices**: Deploy sensors and IoT devices to collect real-time data on resource usage, environmental conditions, and infrastructure performance.

**B)Data Integration Platforms**: Use platforms that can integrate data from various sources for comprehensive analysis.

**C)Data Analytics and AI**: Implement AI and machine learning algorithms to analyze the collected data, identify patterns, and predict resource needs

**Layer 2: Resource Allocation Models**

**A)Mathematical Models**: Develop mathematical models for optimizing resource distribution based on data analytics.

**B)Simulation Tools**: Use simulation tools to test different resource allocation strategies and predict their outcomes.

**C)Decision Support Systems**: Create decision support systems to help stakeholders choose the most efficient resource allocation strategies.

**2.2 Functionality**

1. **Data Collection and Analysis**

Data collection for cloud-based applications involves gathering relevant data from various sources, including user interactions, application logs, and external APIs. This data is often stored in cloud databases or data lakes for scalability and accessibility. Analysis involves processing this data using tools like cloud-based analytics platforms, machine learning models, and big data technologies to extract insights, identify trends, and make data-driven decisions. Effective data collection and analysis help optimize application performance, enhance user experience, and drive business growth.

1. **Resource Allocation Models**

**A)Optimization Algorithms**: Develop and implement algorithms to optimize resource distribution based on real-time data.

**B)Simulation and Scenario Testing**: Use simulation tools to test different resource allocation strategies and evaluate their potential outcomes.

**C)Decision Support**: Provide decision-makers with tools and insights to make informed resource allocation choices.

D)Resource Utilization Analysis: Create decision support systems to help stakeholders choose the most efficient resource allocation strategies

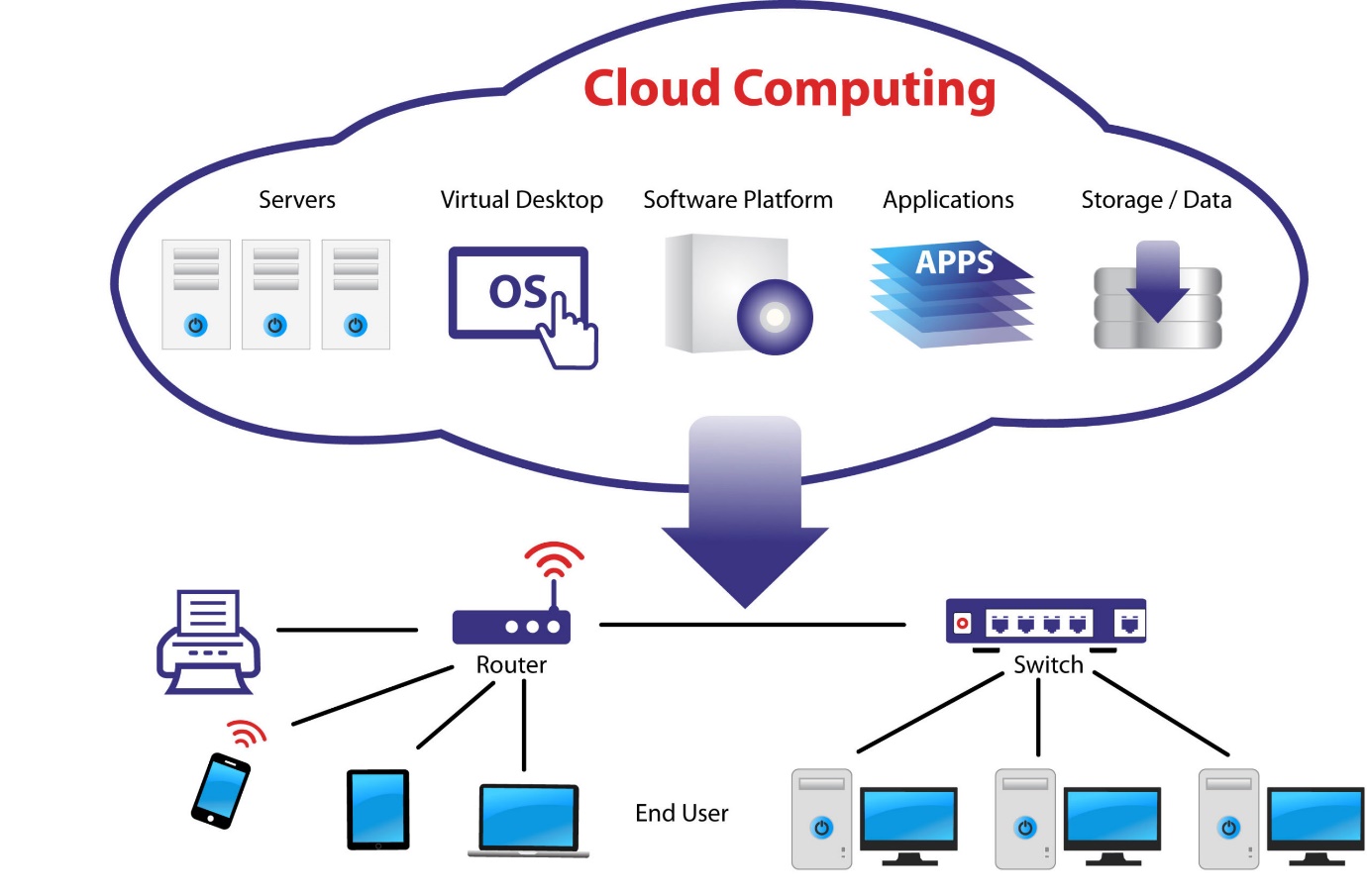
**Infrastructure Management**

**A)Infrastructure Assessment**: Conduct assessments to determine the current state of infrastructure and identify areas for improvement.

**B)Upgrade Planning**: Develop and implement plans for infrastructure upgrades and maintenance to enhance efficiency and sustainability.

C)Create decision support systems to help stakeholders choose the most efficient resource allocation strategies.

**2.3 Architectural Design**



**3. GUI DESIGN:**

**Objectives:**

**A)Overview Panel**: Displays a summary of key metrics, including resource usage, environmental conditions, and infrastructure performance.

**B)Alerts and Notifications**: Shows real-time alerts and notifications for any critical issues or anomalies detected by the system.

The graphical user interface (GUI) for the architectural design framework should be user-friendly, intuitive, and capable of presenting complex data and analysis in an easily understandable manner.

**3.1 Layout**

**A)Graphs and Charts**: Visual representations of data trends, resource allocation, and performance metrics.

**B)Heat Maps**: Geographic representation of resource usage and environmental impact across different regions.

1. **Sidebar:**

**Interactive maps showing the current allocation of resources.**

**Optimization Suggestions: Recommendations for optimizing resource allocation based on predictive analytics.**

**c) Main Content Area:**

**A)Service Performance**: Real-time monitoring of service delivery performance metrics.

**B)Customer Feedback Integration**: Interface for viewing and analyzing customer feedback on services.

**d) Resource Management:**

**A)Policy Analysis Tools**: Tools for analyzing the impact of existing policies on resource allocation.

**B)Regulatory Compliance Monitoring**: Interface for tracking compliance with relevant regulations and standards.

**e) Data Management:**

**A)Collaboration Platform**: Tools for facilitating communication and collaboration among stakeholders.

**B)Feedback Mechanisms**: Interfaces for collecting and analyzing stakeholder feedback.

**f) Monitoring:**

**A)Overview Panel**: Displays a summary of key metrics, including resource usage, environmental conditions, and infrastructure performance.

**B)Alerts and Notifications**: Shows real-time alerts and notifications for any critical issues or anomalies detected by the system.

**g) Footer:**

Footer Links: About, Help, Contact Support

**3.2 User-Friendly Features**

**a) Intuitive Navigation:**

**Clearly Labeled Navigation Menu: Ensure that the navigation menu is clearly labeled with intuitive names for each section.**

**b) Responsive Design:**

**Device Compatibility**: Ensure that the GUI is fully responsive and functions seamlessly across different devices, including desktops, tablets, and smartphones.

**c) Tooltips and Help Icons:**

Provide tooltips that appear when users hover over specific elements, offering brief explanations or additional information about complex features or terms.

**d) Search Functionality:**

**As resource allocation models, infrastructure reports, service performance metrics, and policy analysis tools.**

**e) Customization:**

Allow users to customize their dashboards by selecting and arranging the metrics and data visualizations that are most relevant to them

**f) Error Handling:**

Enable users to drill down into data visualizations to access more detailed information and gain deeper insights.

**3.3 Color Selection**

**a) Primary Colors:**

Blue: #1E90FF (for headers, buttons, and links)

White: #FFFFFF (background for main content areas to ensure readability)

**b) Secondary Colors:**

Gray: #F5F5F5 (for sidebars and secondary background)

Dark Gray: #A9A9A9 (for borders, separators, and secondary text)

**c) Accent Colors:**

Green: #32CD32 (for success indicators, e.g., job completion, healthy nodes)

Red: #FF4500 (for error messages, alerts, and critical status indicators)

Yellow: #FFD700 (for warning messages and important notifications)

**d) Text Colors:**

Black: #000000 (for primary text)

Dark Gray: #2F4F4F (for secondary text)

**4. Program/Coding:**

**4.1 Language Selection**

**a) Primary Language: Java**

**Reason: Java is a robust, object-oriented language that is highly suitable for developing complex applications. Its performance and scalability make it ideal for creating resource allocation models, infrastructure management systems, and service optimization tools.**

**b) Secondary Language: Python**

**Reason: Python is excellent for data processing, analysis, and scripting tasks. Its extensive libraries and frameworks, such as pandas for data manipulation and matplotlib for visualization, make it a powerful tool for integrating with the main system and performing advanced analytics.**

**c) Configuration and Scripting:** Shell Scripts

Shell scripting is essential for automating routine tasks, such as data collection, system monitoring, and resource management processes.

* 1. **Algorithm/Program**

**a)Java Example:**

**Resource Allocation Optimization with Fair Scheduler**

**Objective**: Submit a job to the system using the Fair Scheduler to optimize resource allocation for environmental infrastructure and service management.

**Sample Code:**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class JobSchedulerExample {

public class ResourceAllocationExample {

public static class MyMapper extends Mapper<Object, Text, Text, Text> {

// Mapper implementation for resource allocation tasks

}

public static class MyReducer extends Reducer<Text, Text, Text, Text> {

// Reducer implementation for optimizing resource allocation

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration(); conf.set("mapreduce.job.scheduler",

conf.set("mapreduce.job.scheduler", "org.apache.hadoop.mapred.FairScheduler"); // Set Fair Scheduler

Job job = Job.getInstance(conf, "Resource Allocation Optimization");

job.setJarByClass(ResourceAllocationExample.class);

job.setMapperClass(MyMapper.class);

job.setReducerClass(MyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**b) Python Example: Submitting a Job with Pydoop**

Objective:

Submit a MapReduce job for optimizing resource allocation in environmental infrastructure using Python and Pydoop.

**Sample Code:**

import pydoop.hdfs as hdfs

import pydoop.mapreduce.api as api

import pydoop.mapreduce.pipes as pipes

class Mapper(pipes.Mapper):

def map(self, context):

# Mapper implementation for resource optimization tasks

pass

class Reducer(pipes.Reducer):

def reduce(self, context):

# Reducer implementation for optimizing resource allocation

pass

def main():

pipes.run\_task(pipes.Factory(Mapper, Reducer))

if \_\_name\_\_ == "\_\_main\_\_":

main()

**c) Shell Script for Job Submission**

Objective: Automate the submission of a resource optimization job using a shell script.

**Sample Script:**

#!/bin/bash

# Define variables

INPUT\_PATH="/user/hadoop/resource\_input"

OUTPUT\_PATH="/user/hadoop/resource\_output"

JAR\_PATH="/path/to/your/resource-optimization-job.jar"

CLASS\_NAME="com.example.ResourceAllocationExample"

# Submit the job

hadoop jar $JAR\_PATH $CLASS\_NAME -D mapreduce.job.scheduler=org.apache.hadoop.mapred.FairScheduler -input $INPUT\_PATH -output $OUTPUT\_PATH

# Check job status

JOB\_ID=$(hadoop job -list | grep "your-job-name" | awk '{print $1}')

hadoop job -status $JOB\_ID

**4.3 Execution**

a) **Compile and Package (Java):**

* Compile Java Code:

javac -classpath `hadoop classpath` -d . JobSchedulerExample.java

1. **Run Python Script:**

Install pydoop

python your\_python\_script.py

1. **Execute Shell Script:**

Make Script Executable:

chmod +x submit-job.sh

**5. Implementation:**

**5.1 Connecting the Components**

Objective: Integrate the various components to ensure seamless operation and optimization of job scheduling.

**Components:**

1. Resource Manager (YARN)

2. Scheduler (Fair Scheduler/Capacity Scheduler)

3. Data Layer (HDFS)

4. Monitoring and Management Tools

5. Job Optimization Layer

**Steps:**

1. **Configure Resource Manager:**

Install YARN Resource Manager: Ensure it is properly installed and configured on your Hadoop cluster.

Set Scheduler: Configure the Fair Scheduler or Capacity Scheduler in the `mapred-site.xml` file.

```xml

<property>

<name>mapreduce.job.scheduler</name>

<value>org.apache.hadoop.mapred.FairScheduler</value>

</property>

Restart YARN Services: Apply configuration changes and restart Resource Manager and Node Managers.

**b) Set Up Data Layer (HDFS):**

A)Configure HDFS: Ensure HDFS is properly set up and configured for data storage.

B)Optimize Data Locality: Configure rack awareness in `hdfs-site.xml` to improve data locality.

```xml

<property>

<name>dfs.namenode.rack Awareness</name>

<value>true</value>

</property>

**c) Integrate Job Scheduling with Scheduler:**

A)Submit Jobs: Use the configured scheduler to submit and manage jobs.

B)Monitor Scheduler: Ensure jobs are being scheduled and resources allocated according to configured policies.

**d) Connect Monitoring and Management Tools:**

A)Install Tools: Set up Ambari, Cloudera Manager, or Grafana for monitoring.

B)Integrate with Hadoop: Configure these tools to pull metrics and performance data from Hadoop components.

**e) Optimize Job Scheduling:**

A)Configure Job Parameters: Tune parameters for MapReduce jobs, such as number of mappers and reducers.

B)Apply Optimization Techniques: Use efficient algorithms and combine small jobs where applicable.

* 1. **Cloud Deployment**

A)Objective: Deploy the Hadoop cluster on a cloud environment for scalability and flexibility.

**Steps:**

1. **Choose Cloud Provider:**

A)Popular Options: AWS (Amazon EMR), Google Cloud (Dataproc), Azure (HDInsight).

B)Select based on Requirements: Choose based on cost, scalability, and specific features required.

**b) Provision Cloud Resources:**

C)Create Virtual Machines/Instances: Provision the required number of instances/nodes.

D)Install Hadoop: Use pre-built Hadoop images or manually install Hadoop on the instances.

**c) Set Up Hadoop Cluster:**

A)Configure Cluster: Set up the cluster configuration files (`core-site.xml`, `hdfs-site.xml`, `yarn-site.xml`).

B)Deploy HDFS and YARN: Ensure HDFS and YARN are properly set up and configured.

**d) Deploy Job Scheduling Components:**

A)Install and Configure Scheduler: Use Fair Scheduler or Capacity Scheduler as per the requirements.

B)Deploy Monitoring Tools: Set up and configure monitoring tools in the cloud environment.

**e) Automate Scaling:**

Auto-scaling: Configure auto-scaling groups to automatically add or remove instances based on load.

**f) Testing and Validation:**

A)Run Sample Jobs: Test job submission and scheduling to validate the setup.

B)Monitor Performance: Ensure the system is functioning as expected in the cloud environment.

**5.3 Project Testing**

Objective: Validate the functionality, performance, and reliability of the Hadoop job scheduling system.

**Steps:**

**a) Unit Testing:**

A)Test Individual Components: Validate each component (e.g., Resource Manager, Scheduler) to ensure it operates correctly in isolation.

**b) Integration Testing:**

A)Test Component Interactions: Verify that components work together as expected (e.g., job scheduling, resource allocation).

B)Submit Test Jobs: Run test jobs to check if they are scheduled, executed, and completed as intended.

**c) Performance Testing:**

A)Benchmark Jobs: Measure job completion times and resource utilization.

B)Load Testing: Simulate high workloads to test system performance and stability under stress.

**d) User Acceptance Testing (UAT):**

A)End-User Testing: Have actual users test the system to ensure it meets their requirements and expectations.

B)Gather Feedback: Collect feedback to identify and address any issues or usability concerns.

**e) Deployment Verification:**

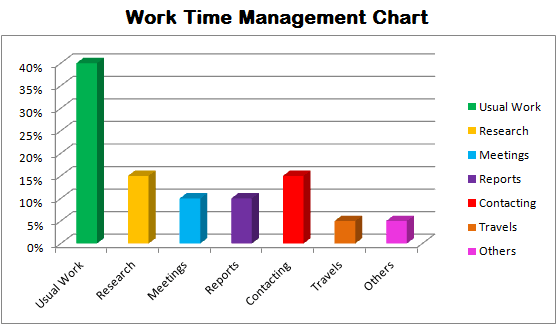
A)Validate Cloud Deployment: Ensure the cloud deployment matches the required configuration and performance benchmarks.

B)Monitor Post-Deployment: Continuously monitor the system post-deployment to identify and resolve any issues that arise.

**6. PERFORMANCE EVALUVATION:**

Performance Evaluation for Hadoop Job Scheduling Efficiency

Objective: Assess and ensure the effectiveness of job scheduling improvements in maximizing resource utilization and reducing job completion time.



**Evaluation Criteria:**

1. Resource Utilization

2. Job Completion Time

3. System Scalability

4. Fault Tolerance and Reliability

5. Cost Efficiency

**A) Performance Metrics**

**1. Resource Utilization Metrics:**

A)CPU Utilization: Measure the percentage of CPU capacity used by jobs.

B)Memory Utilization: Track the amount of memory used versus allocated.

C)Disk I/O: Monitor read and write operations on HDFS.

D)Network I/O: Measure data transfer rates between nodes.

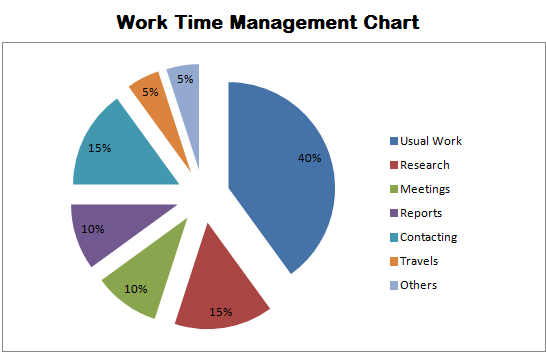
**2. Job Completion Time Metrics:**

A)Job Start Time: Time when the job starts execution.

B)Job End Time: Time when the job completes.

C)Total Job Duration: Time difference between job start and end times.

D)Average Job Completion Time: Average duration of all jobs executed.



**3. System Scalability Metrics:**

A)Scalability: Assess how well the system scales with increased load (e.g., adding more nodes).

B)Auto-Scaling Efficiency: Evaluate how effectively auto-scaling adjusts resources based on workload.

**4. Fault Tolerance and Reliability Metrics:**

A)Failure Recovery Time: Time taken for the system to recover from a failure or node crash.

B)Job Success Rate: Percentage of jobs completed successfully versus those that failed.

C)System Uptime: Measure the percentage of time the system is operational without interruptions.

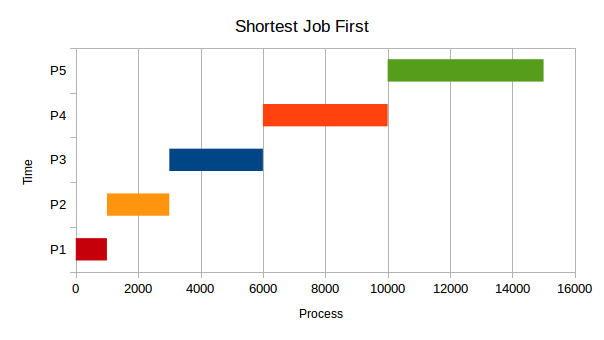
**5. Cost Efficiency Metrics:**

A)Resource Cost: Track costs associated with resource usage (e.g., compute, storage).

B)Cost per Job: Calculate the average cost incurred per job.

C)Cost Savings: Compare costs before and after optimizations.

**B) Evaluation Methods**

****

**1. Benchmarking:**

A)Baseline Metrics: Measure current performance metrics before implementing improvements.

B)Post-Optimization Metrics: Measure performance after changes are applied to assess improvements.

**2. Load Testing:**

A)Simulate Workloads: Use load testing tools to simulate different workloads and assess how well the system handles various levels of demand.

B)Monitor Performance: Track performance metrics during load tests to identify bottlenecks and assess system behavior under stress.

**3. Performance Profiling:**

A)Analyze Jobs: Use profiling tools to analyze job execution times, resource usage, and identify performance issues.

B)Optimize Configuration: Adjust configuration settings based on profiling results to improve efficiency.

**4. Real-World Testing:**

A)Deploy in Production: Run real jobs in a production environment to evaluate how well the system performs under actual conditions.

B)Gather User Feedback: Collect feedback from users regarding system performance and any issues encountered.

**7. Conclusion:**

Optimizing resource allocation in environmental infrastructure and service management is crucial for enhancing efficiency and sustainability. By employing advanced scheduling algorithms, such as the Fair Scheduler or Capacity Scheduler, and integrating robust job submission and execution strategies, significant improvements can be realized in resource utilization. These strategies help in balancing the load, reducing idle times, and ensuring that resources are allocated effectively across various projects and services. The implementation of these optimization techniques not only enhances overall system performance but also contributes to more effective management of environmental and infrastructural resources.

8. Reference:

Benfenatki, Hind, et al. "Cloud-based business applications development methodology." *2014 IEEE 23rd International WETICE Conference*. IEEE, 2014.

Benfenatki, H., Da Silva, C. F., Benharkat, A. N., & Ghodous, P. (2014, June). Cloud-based business applications development methodology. In *2014 IEEE 23rd International WETICE Conference* (pp. 275-280). IEEE.

Benfenatki, Hind, Catarina Ferreira Da Silva, Aïcha-Nabila Benharkat, and Parisa Ghodous. "Cloud-based business applications development methodology." In *2014 IEEE 23rd International WETICE Conference*, pp. 275-280. IEEE, 2014.

Benfenatki, H., Da Silva, C.F., Benharkat, A.N. and Ghodous, P., 2014, June. Cloud-based business applications development methodology. In *2014 IEEE 23rd International WETICE Conference* (pp. 275-280). IEEE.

Benfenatki H, Da Silva CF, Benharkat AN, Ghodous P. Cloud-based business applications development methodology. In2014 IEEE 23rd International WETICE Conference 2014 Jun 23 (pp. 275-280). IEEE.

Hamdaqa, Mohammad, Tassos Livogiannis, and Ladan Tahvildari. "A Reference Model for Developing Cloud Applications." *CLOSER*. 2011.

Hamdaqa, M., Livogiannis, T., & Tahvildari, L. (2011, May). A Reference Model for Developing Cloud Applications. In *CLOSER* (pp. 98-103).

Hamdaqa, Mohammad, Tassos Livogiannis, and Ladan Tahvildari. "A Reference Model for Developing Cloud Applications." In *CLOSER*, pp. 98-103. 2011.

Hamdaqa, M., Livogiannis, T. and Tahvildari, L., 2011, May. A Reference Model for Developing Cloud Applications. In *CLOSER* (pp. 98-103).

Hamdaqa M, Livogiannis T, Tahvildari L. A Reference Model for Developing Cloud Applications. InCLOSER 2011 May (pp. 98-103).