Behind the Report:

Unlocking Cost and Efficiency Gains with Next-Generation Single-Socket Servers

This methodology describes the test configurations and procedures used by Prowess Consulting's engineers to compare the performance and efficiency gains of consolidating from a dual-socket 14th-generation Dell™ PowerEdge™ R740xd server to a single-socket 17th-generation PowerEdge R570 server.

Methodology Overview

This document outlines how Prowess Consulting's engineers tested and validated the claims in the technical research report regarding consolidating workloads from dual-socket to single-socket servers. To provide a practical analysis, Prowess Consulting compared a commonly used dual-socket 14th-generation Dell™ PowerEdge™ R740xd server against a modern single-socket 17th-generation PowerEdge R570 server to represent the type of hardware a business might adopt during a typical server refresh.

Our comparisons focused on the performance, efficiency, and consolidation capabilities of each server. To quantify the benefits of this consolidation strategy, our tests included:

- Speech-to-text-to–sentiment analysis
- Apache HTTP Server™ benchmarking tools
- NGINX®/Redis® caching workloads.

We configured both servers with the latest firmware and Red Hat® Enterprise Linux® 9.5, and we established baseline CPU and memory performance using sysbench. Metrics captured included server throughput, latency, power consumption, and resource utilization, providing a comprehensive view of each system's real-world performance and operational efficiency.

Table 1 | Server platform configuration summary

Server Platform	Dell™ PowerEdge™ R570 Server (1S Intel® Xeon® 6787P Processor)	Dell™ PowerEdge™ R740xd Server (2S Intel® Xeon® Gold 5217 Processor)
Platform Setup Default Red Hat® Enterprise Linux® 9.5 installation with Python® 3, the Vosk speech-recognition models, Natural Language Toolkit (NLTK) sentiment libraries, and required audio-processing dependencies, and with no custom operating system (OS) tuning beyond standard best practices	Latest firmware and BIOS updates applied	
	Sysbench CPU and memory baseline validation prior to workload execution	

System Configuration

This section outlines the steps taken to initialize the systems before use.

Operating System Installation

Install Red Hat Enterprise Linux 9.5 on each of the target systems.

1. Log in to the Integrated Dell™ Remote Access Controller (iDRAC) interface.
2. To open the virtual console, click the **Virtual Console** icon.
3. To load the OS .iso:
   a. Click the **Virtual Media** button.
   b. Click the **Connect Virtual Media** button.
   c. In the **Map CD/DVD** section, click **Choose file**.
   d. Select the **RHEL 9.5** .iso image.
   e. Click **Map device**.
   f. Click **Close**.
4. To boot to the .iso:
   a. In the virtual console, click the **Boot** button, and then select **Virtual CD/DVD/ISO**.
   b. Click the **Power** button, and then select **Reboot system (warm boot)**.
5. To install the OS:
   a. When prompted, select **Install Red Hat 9.5**.
   b. From the initial window, leave the default **English** selections, and then click **Continue**.
   c. To specify the install location, click **Installation Destination**.
      i. Select the disk associated with the Dell™ Boot-Optimized Storage Solution (BOSS) virtual disk.
      ii. Leave the **Automatic** selection for the storage configuration.
      iii. Click **Done**.
   d. To create the root account:
      i. Click **Root Password**.
      ii. If selected, clear the checkbox for **Lock root account**.
      iii. Specify a **Password**, and then confirm it.
      iv. Click **Done**.
   e. To create the user account:
      i. Click **User Creation**.
      ii. Enter a full **Name**.
      iii. Enter a **User Name**.
      iv. Select **Make this user administrator**.
      v. Enter and confirm the password for the account.
      vi. Click **Done**.
   f. Click **Begin Installation**, and then wait for the installation to complete.
   g. Click **Reboot system**.
6. To log on, click the user name, and then enter the password.
7. To register the Red Hat Enterprise Linux system, click the **Register System** notice.
   a. Click **Subscription**.
   b. Enter **Login**.
   c. Enter **Password**.
   d. Click **Register**.
8. To check the IP, open a terminal window and run **ifconfig**, henceforth ***$SystemIP***.
9. To update to the latest Red Hat Enterprise Linux 9.5 components, run the following from the terminal window:

```
sudo subscription-manager release --set=9.5

sudo dnf update -y
```

10. To mount the data volume, take the following actions:
    a. To check the dev id of the Dell™ PowerEdge RAID Controller (PERC)-controlled volume, run:

```
sudo fdisk -l
```

b. To create the file system on that disk, run:

```
sudo mkfs.xfs /dev/${ActualDiskID}
```

c. To check the UUID of that disk ID, run:

```
ls -lah /dev/disk/by-uuid/ | grep ${ActualDiskID}
```

d. To create the working directory, run:

```
mkdir -p ~/benchmark

sudo chown user:user ~/benchmark/
```

e. Referencing the UUID, add the following to **/etc/fstab**:

```
UUID=${ActualDiskUUID}    /home/user/benchmark    xfs  noatime,nodiratime,largeio,swalloc,inode64  0 2
```

f. To mount the volume, run:

```
sudo systemctl daemon-reload

sudo mount -a
```

Workload Execution

This section outlines the steps taken to set up and run the AI sentiment analysis, web serving, and caching workloads.

Speech to Text and Sentiment Analysis

This section outlines the steps taken to convert audio files to text transcriptions and then perform a sentiment analysis on the content. Unless specifically noted, all of the following steps were applied to both systems.

System Setup

1. Open a Secure Shell (SSH) connection from the local workstation to the system under test (SUT) by running:

```
ssh user@$SystemIP
```

2. To install required packages, run:

```
sudo dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm

sudo dnf install -y python3.11 python3.11-devel gcc git sysstat numactl dstat
```

3. To install the Python® packages, run:

```
python3.11 -m venv ~/benchmark/speech-env

source ~/benchmark/speech-env/bin/activate

pip install --upgrade pip

pip install vosk nltk soundfile psutil pandas matplotlib seaborn

python -m nltk.downloader vader_lexicon
```

4. To download the Vosk model, run:

```
mkdir -p ~/benchmark/speech-to-text-sentiment-analysis

cd ~/benchmark/speech-to-text-sentiment-analysis

wget https://alphacephei.com/vosk/models/vosk-model-en-us-0.22.zip

unzip vosk-model-en-us-0.22.zip

mv vosk-model-en-us-0.22 vosk-model
```

5. To download the LibriSpeech test data, run:

```
wget https://www.openslr.org/resources/12/dev-clean.tar.gz

tar -xzf dev-clean.tar.gz
```

6. To create the **pre_test_setup.sh, stt_sentiment_batch_worker.py**, and **stt_runner.sh** scripts, take the following actions, referencing the **Appendix** for the file content:
   a. To create the scripts directory, run:

```
mkdir -p ~/benchmark/speech-to-text-sentiment-analysis/scripts
```

   b. To open the file for editing, run:

```
vim ~/benchmark/speech-to-text-sentiment-analysis/scripts/$FileName
```

   c. To use paste mode, press **:set paste <Enter>**.
   d. Press a to enter insert paste mode.
   e. Paste the content from the relevant appendix for the file name.
   f. To save and exit, press **:wq <Enter>**.
   g. To make the executable, run:

```
chmod +x ~/benchmark/speech-to-text-sentiment-analysis/scripts/$FileName
```

7. To create the power monitor script, copy the content of the **Power Monitoring** appendix to the local workstation.

Running Tests

This section outlines the steps taken when running the test:

1. To apply system optimizations, run:

```
source ~/benchmark/speech-to-text-sentiment-analysis/scripts/ pre_test_setup.sh
```

2. From the local workstation, start the power monitoring script, and then run the following, updating to the actual iDRAC IP address and user credentials:

```
~/tools/idrac_power_simple.py -i ${IdracIP} -u ${IdracUser} -p ${IdracPassword} --interval 5
```

3. To start the system monitoring and test, from the SUT, run:

```
~/benchmark/speech-to-text-sentiment-analysis/scripts/stt_runner.sh -o $TestName -v   ${ActualDiskID} -t
~/benchmark/speech-to-text-sentiment-analysis/scripts/
```

4. When the script finishes running, from the local workstation, press **Ctrl+C** to exit the power monitor script.

Apache® Bench and Apache JMeter™ Testing

This section outlines the steps taken to perform the Apache® Bench and Apache JMeter™ tests.

System Setup

This section outlines the steps taken to prepare the system for Apache web server-based tests.

1. To update all packages to the latest within Red Hat Enterprise Linux 9.5 version, run:

```
sudo subscription-manager release --set=9.5

sudo dnf update -y
```

2. Open an SSH connection from the local workstation to the SUT by running:

```
ssh user@$SystemIP
```

3. To install required packages, run:

```
sudo dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm

sudo dnf install -y httpd httpd-tools mod_ssl java-17-openjdk java-17-openjdk-devel wget curl tar gzip bc
numactl numactl-libs htop sysstat kernel-tools dstat nmon
```

4. To install Apache JMeter 5.6.3, run:

```
cd ~/benchmark

wget https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-5.6.3.tgz

tar -xzf apache-jmeter-5.6.3.tgz

sudo ln -sf ~/benchmark/apache-jmeter-5.6.3 /opt/jmeter
```

5. To create the systemd override directory, run:

```
sudo mkdir -p /etc/systemd/system/httpd.service.d
```

6. To add service limits for Apache, run:

```
sudo tee /etc/systemd/system/httpd.service.d/limits.conf << 'EOF'
[Service]
LimitNOFILE=1000000
LimitNPROC=65535
EOF
sudo systemctl daemon-reload
```

7. To create the kernel-tuning configuration, run:

```
sudo tee /etc/sysctl.d/99-benchmark.conf << 'EOF'
# Network performance tuning for benchmarking
net.core.somaxconn = 65535
net.core.netdev_max_backlog = 65535
net.ipv4.tcp_max_syn_backlog = 65535
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 15
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 87380 16777216
fs.file-max = 2097152
EOF
sudo sysctl -p /etc/sysctl.d/99-benchmark.conf
```

8. To create the test content directory structure, run:

```
mkdir -p ~/benchmark/www/html/api
tee ~/benchmark/www/html/index.html << 'EOF'
<!DOCTYPE html>
<html>
<head><title>Apache Benchmark Test</title></head>
<body>
<h1>Apache HTTP Server Benchmark Test Page</h1>
<p>This is a test page for performance benchmarking.</p>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua.</p>
</body>
```

```
</html>

EOF

tee ~/benchmark/www/html/about.html << 'EOF'

<!DOCTYPE html>

<html>

<head><title>About - Apache Benchmark</title></head>

<body>

<h1>About Page</h1>

<p>Secondary test page for multi-endpoint benchmarking.</p>

<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>

</body>

</html>

EOF

tee ~/benchmark/www/html/api/data << 'EOF'

{"status":"ok","message":"API endpoint for benchmarking","timestamp":1234567890}

EOF

sudo rm -rf /var/www/html

sudo ln -sf ~/benchmark/www/html /var/www/html

sudo chown -R apache:apache ~/benchmark/www

sudo chmod -R 755 ~/benchmark/www
```

9. To disable Apache logging, run:

```
sudo cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.backup

sudo sed -i 's/^\s*CustomLog/#CustomLog/' /etc/httpd/conf/httpd.conf
```

10. To create the configuration generator, configuration switcher, abj_runner, and run_benchmark scripts, the **Apache Bench and Apache JMeter appendix** contents and run:

```
vim ~/benchmark/${filename}
```

   a. To set paste mode, press **:set paste <Enter>**.
   b. To enter insert mode, press **a**.
   c. Paste the content from the relevant appendix.
   d. To save and exit, press **<Esc> :wq <Enter>**.
   e. To set the execute permission, run:

```
chmod +x ~/benchmark/${filename}
```

11. To generate Apache Multi-Processing Modules (MPM) configurations, run:

```
cd ~/benchmark

sudo ./generate_apache_configs.sh —configs $SELECTED_CONFIGS
```

   a. For the PowerEdge R740xd server, use **1, 2, 3, 4, 5,** and **6** for the selected configurations.
   b. For the PowerEdge R570 server, use **1, 2, 3, 4, 5, 6, 7, 8,** and **9** for the selected configurations.

12. To enable Apache to start on boot, run:

```
sudo systemctl enable httpd
```

13. To set the proper SELinux context, run:

```
sudo setsebool -P httpd_read_user_content 1

sudo setsebool -P httpd_enable_homedirs 1

sudo semanage fcontext -a -t httpd_sys_content_t "/home/user/benchmark/www(/.*)?"

sudo restorecon -Rv /home/user/benchmark/www

sudo chmod +x /home/user /home/user/benchmark /home/user/benchmark/www

sudo systemctl restart httpd
```

14. To set the server IP address, run:

```
IP=$(hostname -I | awk '{pri.')
```

Running Tests

This section contains the steps taken to run the test workloads.

1. From the local workstation, start the power monitoring script and run the following, updating to the actual iDRAC IP address and user credentials:

```
~/tools/idrac_power_simple.py -i ${IdracIP} -u ${IdracUser} -p ${IdracPassword} --interval 5
```

2. To start the system monitoring and test run, from the SUT, run:

```
~/benchmark/scripts/abj_runner.sh -o $TestName -v   ${ActualDiskID} -t ~/benchmark/speech-to-text-sentiment-analysis/scripts/$
```

3. When the script finishes running, from the local workstation, press **Ctrl+C** to exit the power monitor script.

OpenResty® and Redis® Testing

This section outlines the steps taken to perform the Redis and OpenResty® tests powered by WRK®.

System Setup

This section outlines the steps taken to prepare the system for OpenResty web server-based tests.

1. To update all packages to the latest within Red Hat Enterprise Linux 9.5, run:

```
sudo subscription-manager release --set=9.5

sudo dnf update -y
```

2. To install monitoring tools, run:

```
sudo dnf install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm

sudo dnf install -y nmon dstat sysstat
```

3. To prepare the data disk, run:

```
# To identify the disk

Sudo fdisk -l

# To create the file system on the disk

Sudo mfs.xfs /dev/$DiskID

# To get the uuid of the disk

Sudo blkid /dev/$DiskID
```

4. To mount the data disk, run:

```
Sudo vim /etc/fst
```

```
# Add the below line, updating the UUID from above

UUID=<your-uuid-here> /home/user/benchmark xfs defaults 0 2

# To create the mount point

Mkdir /home/user/benchmark

# To mount run

Systemctl daemon-reload

Sudo mount -a

# To verify run

Df -h /home/user/benchmark
```

5. To update permissions for the directory, run:

```
sudo chown user:user ~/benchmark

chmod o+x /home/user
```

6. To create the scripts directory, run:

```
mkdir -p /home/user/benchmark/scripts
```

7. To create the **openresty-redis.sh** and **or_runner.sh** files, reference the relevant appendix for the contents and run:

```
vim /home/user/benchmark/scripts/${filename}
```

   a. To set paste mode, press **:set paste <Enter>**.
   b. To enter insert mode, press **a**.
   c. Paste the content from the relevant appendix.
   d. To save and exit, press **<Esc> :wq <Enter>**.
   e. To set the execute permission, run:

```
chmod +x ~/benchmark/${filename}
```

8. To set the system type, run either **export SYSTEM_TYPE=r740** or **export SYSTEM_TYPE=r570**, as appropriate for the SUT.
9. To install system dependencies, run:

```
./openresty-redis.sh install
```

10. To generate 8,000 images to use in the test, run:

```
./openresty-redis.sh generate-images
```

Running Tests

This section contains the steps taken to run the test workloads.

1. From the local workstation, start the power monitoring script and run the following, updating to the actual iDRAC IP address and user credentials:

```
~/tools/idrac_power_simple.py -i ${IdracIP} -u ${IdracUser} -p ${IdracPassword} --interval 5
```

2. To start the system monitoring and test run, from the SUT, run:

```
/home/user/benchmark/scripts/or_runner.sh \ -t /home/user/benchmark/scripts \ -o /home/user/
results/$SystemType-openResty \ -v /dev/$DiskID
```

3. When the script finishes running, from the local workstation, press **Ctrl+C** to exit the power monitor script.

Appendix

The appendix sections contain the content for the files referenced in this document.

Power Monitoring

This Python script is used from a local workstation to monitor the power usage during the various test runs.

```
#!/usr/bin/env python3

"""

Simple Dell iDRAC Power Monitor

Grabs power data via racadm getsensorinfo and logs to CSV

"""

import subprocess

import csv

import time

import argparse

import sys

from datetime import datetime

from pathlib import Path

def run_racadm(idrac_ip, username, password):

    """Execute racadm getsensorinfo command via SSH"""

    cmd = [

        'sshpass', '-p', password,

        'ssh',

        '-o', 'StrictHostKeyChecking=no',

        '-o', 'UserKnownHostsFile=/dev/null',

        '-o', 'LogLevel=ERROR',

        f'{username}@{idrac_ip}',

        'racadm getsensorinfo'

    ]

    result = subprocess.run(cmd, capture_output=True, text=True, timeout=30)

    return result.stdout

def parse_power_data(output):

    """Extract power metrics from racadm output"""

    data = {

        'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

        'system_watts': None,

        'ps1_watts': None,
```

```
            'ps1_amps': None,
            'ps1_volts': None,
            'ps2_watts': None,
            'ps2_amps': None,
            'ps2_volts': None,
        }
        lines = output.split('\n')
        in_power_section = False
        in_current_section = False
        in_voltage_section = False
        for line in lines:
            # Track which section we're in
            if 'Sensor Type : POWER' in line:
                in_power_section = True
                in_current_section = False
                in_voltage_section = False
                continue
            elif 'Sensor Type : CURRENT' in line:
                in_current_section = True
                in_power_section = False
                in_voltage_section = False
                continue
            elif 'Sensor Type : VOLTAGE' in line:
                in_voltage_section = True
                in_power_section = False
                in_current_section = False
                continue
            elif 'Sensor Type :' in line:
                in_power_section = False
                in_current_section = False
                in_voltage_section = False
                continue
            # Parse POWER section: PS1 Status  Present  AC  230.250Watts
            if in_power_section:
                if 'PS1 Status' in line:
                    parts = line.split()
```

```
                # Last part should be XXX.XXXWatts
                if parts[-1].endswith('Watts'):
                    data['ps1_watts'] = parts[-1].replace('Watts', '')
            elif 'PS2 Status' in line:
                parts = line.split()
                if parts[-1].endswith('Watts'):
                    data['ps2_watts'] = parts[-1].replace('Watts', '')
        # Parse CURRENT section: PS1 Current 1  Ok  1.0Amps  NA   NA  ...
        if in_current_section:
            if 'PS1 Current' in line:
                parts = line.split()
                # Find the reading - it's after "Ok" and before "NA"
                for i, part in enumerate(parts):
                    if part == 'Ok' and i+1 < len(parts):
                        reading = parts[i+1]
                        if 'Amps' in reading:
                            data['ps1_amps'] = reading.replace('Amps', '')
                        break
            elif 'PS2 Current' in line:
                parts = line.split()
                for i, part in enumerate(parts):
                    if part == 'Ok' and i+1 < len(parts):
                        reading = parts[i+1]
                        if 'Amps' in reading:
                            data['ps2_amps'] = reading.replace('Amps', '')
                        break
            elif 'System Board Pwr Consumption' in line:
                parts = line.split()
                for i, part in enumerate(parts):
                    if part == 'Ok' and i+1 < len(parts):
                        reading = parts[i+1]
                        if 'Watts' in reading:
                            data['system_watts'] = reading.replace('Watts', '')
                        break
        # Parse VOLTAGE section: PS1 Voltage 1  Ok  212.00V  NA  NA
        if in_voltage_section:
```

```
            if 'PS1 Voltage' in line:
                parts = line.split()
                for i, part in enumerate(parts):
                    if part == 'Ok' and i+1 < len(parts):
                        reading = parts[i+1]
                        if 'V' in reading and reading.replace('.', '').replace('V', '').isdigit():
                            data['ps1_volts'] = reading.replace('V', '')
                        break
            elif 'PS2 Voltage' in line:
                parts = line.split()
                for i, part in enumerate(parts):
                    if part == 'Ok' and i+1 < len(parts):
                        reading = parts[i+1]
                        if 'V' in reading and reading.replace('.', '').replace('V', '').isdigit():
                            data['ps2_volts'] = reading.replace('V', '')
                        break
    return data
def main():
    parser = argparse.ArgumentParser(description='Monitor Dell iDRAC power metrics')
    parser.add_argument('-i', '--idrac-ip', required=True, help='iDRAC IP address')
    parser.add_argument('-u', '--username', required=True, help='iDRAC username')
    parser.add_argument('-p', '--password', required=True, help='iDRAC password')
    parser.add_argument('--interval', type=int, default=5, help='Sampling interval in seconds (default:
5)')
    parser.add_argument('--duration', type=int, default=None, help='Monitoring duration in seconds
(default: infinite)')
    args = parser.parse_args()
    # Create timestamped directory
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    log_dir = Path(f"idrac_power_{args.idrac_ip}_{timestamp}")
    log_dir.mkdir(exist_ok=True)
    csv_file = log_dir / "power_metrics.csv"
    print(f"Logging to: {log_dir}")
    print(f"Sampling interval: {args.interval} seconds")
    print(f"Duration: {'Infinite (Ctrl+C to stop)' if args.duration is None else f'{args.duration}
seconds'}")
    print("-" * 60)
```

```
    start_time = time.time()
    sample_count = 0
    try:
        while True:
            # Get data from iDRAC
            output = run_racadm(args.idrac_ip, args.username, args.password)
            data = parse_power_data(output)
            # Write to CSV
            file_exists = csv_file.exists()
            with open(csv_file, 'a', newline='') as f:
                writer = csv.DictWriter(f, fieldnames=data.keys())
                if not file_exists:
                    writer.writeheader()
                writer.writerow(data)
            sample_count += 1
            print(f"Sample {sample_count}: System={data['system_watts']}W, "
                  f"PS1={data['ps1_watts']}W/{data['ps1_amps']}A/{data['ps1_volts']}V, "
                  f"PS2={data['ps2_watts']}W/{data['ps2_amps']}A/{data['ps2_volts']}V")
            # Check duration
            if args.duration and (time.time() - start_time) >= args.duration:
                print(f"\nMonitoring complete. Collected {sample_count} samples.")
                break
            time.sleep(args.interval)
    except KeyboardInterrupt:
        print(f"\n\nMonitoring stopped. Collected {sample_count} samples.")
    except Exception as e:
        print(f"ERROR: {e}", file=sys.stderr)
        sys.exit(1)
if __name__ == '__main__':
    main()
```

STT files

pre_test_setup.sh

```
#!/bin/bash

ulimit -n 65535

sudo swapoff -a

if [ -d "/sys/devices/system/cpu/cpu0/cpufreq" ]; then

    for cpu in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do

        echo performance | sudo tee "$cpu" > /dev/null 2>&1

    done

fi

echo always | sudo tee /sys/kernel/mm/transparent_hugepage/enabled > /dev/null

sudo sysctl -w vm.swappiness=1 > /dev/null

export OMP_NUM_THREADS=1

export MKL_NUM_THREADS=1
```

Stt_sentiment_batch_worker.py

```
#!/usr/bin/env python3


import os

import json

from multiprocessing import Pool

from vosk import Model, KaldiRecognizer

from nltk.sentiment import SentimentIntensityAnalyzer

import soundfile as sf

import time

import sys


# Global model - loaded once per worker

_model = None


def init_worker(model_path):

    """Initialize worker with model"""

    global _model

    print(f"Worker {os.getpid()} loading model...")

    _model = Model(model_path)

    print(f"Worker {os.getpid()} ready")
```

```
def process_audio_file(audio_path):
    """Process single audio file"""
    global _model

    try:
        # Read audio
        audio_data, sample_rate = sf.read(audio_path, dtype='int16')

        # Convert to mono if needed
        if len(audio_data.shape) > 1:
            audio_data = audio_data.mean(axis=1).astype('int16')

        # Create recognizer
        rec = KaldiRecognizer(_model, sample_rate)
        rec.SetWords(True)

        # Process in chunks
        chunk_size = 4000
        transcription = ""
        for i in range(0, len(audio_data), chunk_size):
            chunk = audio_data[i:i+chunk_size].tobytes()
            if rec.AcceptWaveform(chunk):
                result = json.loads(rec.Result())
                transcription += result.get('text', '') + " "

        final_result = json.loads(rec.FinalResult())
        transcription += final_result.get('text', '')

        # Sentiment
        sia = SentimentIntensityAnalyzer()
        sentiment = sia.polarity_scores(transcription)

        print(f"Processed: {os.path.basename(audio_path)}")

        return {
            'file': os.path.basename(audio_path),
```

```
            'transcription': transcription.strip(),
            'sentiment': sentiment
        }

    except Exception as e:
        print(f"ERROR processing {audio_path}: {e}")
        return None

def main():
    # Check arguments
    if len(sys.argv) != 3:
        print("Usage: python3 stt_sentiment_batch_worker.py <audio_directory> <num_workers>")
        print("Example: python3 stt_sentiment_batch_worker.py ./LibriSpeech/dev-clean/1272 2")
        sys.exit(1)

    audio_dir = sys.argv[1]
    num_workers = int(sys.argv[2])
    model_path = "/home/user/projects/speech-to-text-sentiment-analysis/vosk-model"

    # Validate inputs
    if not os.path.exists(model_path):
        print(f"ERROR: Model not found at {model_path}")
        sys.exit(1)

    if not os.path.exists(audio_dir):
        print(f"ERROR: Directory not found: {audio_dir}")
        sys.exit(1)

    if num_workers < 1:
        print("ERROR: num_workers must be at least 1")
        sys.exit(1)

    # Find audio files
    print(f"Scanning {audio_dir}...")
    audio_files = []
    for root, dirs, files in os.walk(audio_dir):
```

```
        for file in files:
            if file.endswith((‘.wav’, ‘.flac’, ‘.ogg’, ‘.mp3’)):
                audio_files.append(os.path.join(root, file))

    if not audio_files:
        print(“ERROR: No audio files found”)
        sys.exit(1)

    print(f”\nFound {len(audio_files)} audio files”)
    print(f”Using {num_workers} workers”)
    print(f”Starting processing...\n”)

    start_time = time.time()

    # Process with worker pool
    with Pool(num_workers, initializer=init_worker, initargs=(model_path,)) as pool:
        results = pool.map(process_audio_file, audio_files)

    # Filter failures
    results = [r for r in results if r is not None]

    elapsed = time.time() - start_time

    # Print results
    print(“\n” + “=”*50)
    print(f”Total files: {len(audio_files)}”)
    print(f”Processed: {len(results)}”)
    print(f”Failed: {len(audio_files) - len(results)}”)
    print(f”Time: {elapsed:.2f} seconds”)
    print(f”Throughput: {len(results)/elapsed:.2f} files/sec”)
    print(“=”*50)

    # Save results
    with open(‘stt_results.json’, ‘w’) as f:
        json.dump(results, f, indent=2)
```

```
    print("\nResults saved to stt_results.json")


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

STT_runner.sh

```
#!/bin/bash
# Define the optstring
optstring="rspt:o:v:w:n:"
# Define the variables to store the option values
# Parse the command-line arguments
while getopts "$optstring" opt; do
  case "$opt" in
    t) test_path="$OPTARG" ;;
    o) requested_output_path="$OPTARG" ;;
    v) volume="$OPTARG" ;;

  esac
done

# Sanity check arguments
if [[ -z "$test_path" || -z "$requested_output_path" || -z "$volume"  ]]; then
  echo "Error: missing flags!  All below are required
        -t path to test files
        -o directory to create and save output into
        -v volume to be tested"
  exit 1
fi

# Extract the device name from the volume path
short_volume=$(basename $volume)

ts=$(date +%s)
output_path=${requested_output_path}.${ts}

# Prepare files and output path
```

```
mkdir -p $output_path

cp $test_path/$testType* $output_path/


# start the monitors

nmon -F $output_path/nmon.out -s2 -c100000 -t &


# Initialize d_pid

d_pid=""


# Check if mapped volume or device and start dstat accordingly

if [[ $short_volume == md* ]]; then

        dstat -tam -C total --md -M total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

elif [[ $short_volume == sd* || $short_volume == nv* ]]; then

        dstat -tam -C total -dD total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

fi


iostat -k -t -o JSON -cdx $volume 2 > $output_path/iostat.out &

sleep 2


# Run the tests

concurrency=(1 8 16 32 64 86 129 172 )


for c in ${concurrency[@]}; do

    {

        echo "Start: $(date -Iseconds)"

        python /home/user/benchmark/speech-to-text-sentiment-analysis/scripts/stt_sentiment_batch_worker.
py  /home/user/benchmark/speech-to-text-sentiment-analysis/LibriSpeech/dev-clean/ $c

        echo "End: $(date -Iseconds)"

    } | tee $output_path/sst_${c}_concurrent.raw

    sleep 15

done


# Cleanup monitors

killall nmon
```

```
if [[ -n "$d_pid" ]]; then
    kill $d_pid
fi
killall iostat
echo "]}]}}" >> $output_path/iostat.out
```

Apache Bench and Apache JMeter Files

This section contains the content for the files referenced for the Apache Bench and Apache JMeter testing.

Apache Configuration Generator

```
#!/bin/bash
#
# Apache MPM Configuration Generator
# Creates numbered config files based on worker/thread specifications
#
# Usage: ./generate_apache_configs.sh
#        ./generate_apache_configs.sh --list
#        ./generate_apache_configs.sh --custom 24 64  (24 workers, 64 threads)
#

CONFIG_DIR="/etc/httpd/conf.modules.d"

# Default configurations to generate
# Format: "config_num:workers:threads:description"
DEFAULT_CONFIGS=(
    "1:1:64:Single core baseline"
    "2:4:64:4 workers (small)"
    "3:8:64:8 workers (medium)"
    "4:16:64:16 workers"
    "5:32:64:32 workers"
    "6:43:64:43 workers (50pct of 86-core)"
    "7:64:64:64 workers"
    "8:86:128:86 workers (100pct physical 86-core)"
    "9:172:128:172 workers (100pct logical 172-cpu)"
)

generate_config() {
    local config_num=$1
```

```
    local workers=$2
    local threads=$3
    local description=$4

    local max_workers=$((workers * threads))
    local min_spare=$((workers * threads / 2))
    local max_spare=$((workers * threads * 3 / 4))

    # Ensure minimums
    [ $min_spare -lt 32 ] && min_spare=32
    [ $max_spare -lt 64 ] && max_spare=64

    local config_file="${CONFIG_DIR}/00-mpm-config${config_num}.conf"

    sudo tee "$config_file" > /dev/null << EOF
# Config ${config_num}: ${description}
# Workers: ${workers}, Threads/Worker: ${threads}, Max: ${max_workers}
LoadModule mpm_event_module modules/mod_mpm_event.so

<IfModule mpm_event_module>
    ServerLimit              ${workers}
    StartServers             ${workers}
    ThreadsPerChild          ${threads}
    MaxRequestWorkers        ${max_workers}
    MinSpareThreads          ${min_spare}
    MaxSpareThreads          ${max_spare}
    MaxConnectionsPerChild   0
</IfModule>

KeepAlive On
KeepAliveTimeout 2
MaxKeepAliveRequests 10000
EOF

    echo "Created Config ${config_num}: ${workers}w x ${threads}t = ${max_workers} max (${description})"
```

```
}

list_configs() {
    echo "=== Existing Apache MPM Configurations ==="
    local f
    for f in ${CONFIG_DIR}/00-mpm-config*.conf; do
        if [ -f "$f" ]; then
            local config_num
            config_num=$(basename "$f" | sed 's/00-mpm-config\([0-9]*\)\.conf/\1/')
            local header
            header=$(head -2 "$f" | tail -1)
            echo "  Config ${config_num}: ${header#\# }"
        fi
    done

    echo ""
    echo "=== Active Configuration ==="
    if [ -f "${CONFIG_DIR}/00-mpm.conf" ]; then
        head -3 "${CONFIG_DIR}/00-mpm.conf" | grep -E "^#" | head -2
    else
        echo "  No active configuration"
    fi
}

show_usage() {
    cat << 'EOF'
Apache MPM Configuration Generator

Usage:
  ./generate_apache_configs.sh              Generate all default configs (1-9)
  ./generate_apache_configs.sh --list       List existing configurations
  ./generate_apache_configs.sh --custom N W T "desc"
                                            Create custom config N with W workers, T threads
  ./generate_apache_configs.sh --configs 1,3,5
                                            Generate only specific configs
```

```
Default Configurations:
  1:  1w x 64t  =    64 max  (Single core baseline)
  2:  4w x 64t  =   256 max  (4 workers - small)
  3:  8w x 64t  =   512 max  (8 workers - medium)
  4: 16w x 64t  = 1,024 max  (16 workers)
  5: 32w x 64t  = 2,048 max  (32 workers)
  6: 43w x 64t  = 2,752 max  (43 workers - 50% of 86-core)
  7: 64w x 64t  = 4,096 max  (64 workers)
  8: 86w x 128t = 11,008 max (86 workers - 100% physical)
  9: 172w x 128t = 22,016 max (172 workers - 100% logical)

Examples:
  # Generate configs 1-5 for smaller system (r740)
  ./generate_apache_configs.sh --configs 1,2,3,4,5

  # Generate all configs for larger system (r570)
  ./generate_apache_configs.sh

  # Create custom 24-worker config
  ./generate_apache_configs.sh --custom 10 24 64 "24 workers custom"
EOF
}

# Parse arguments
case "$1" in
    --list)
        list_configs
        exit 0
        ;;
    --help|-h)
        show_usage
        exit 0
        ;;
    --custom)
        if [ $# -lt 5 ]; then
            echo "Error: --custom requires: config_num workers threads description"
```

```
            echo "Example: ./generate_apache_configs.sh --custom 10 24 64 \"24 workers custom\""
            exit 1
        fi
        generate_config "$2" "$3" "$4" "$5"
        exit 0
        ;;
    --configs)
        if [ -z "$2" ]; then
            echo "Error: --configs requires comma-separated list"
            echo "Example: ./generate_apache_configs.sh --configs 1,3,5"
            exit 1
        fi
        IFS=',' read -ra SELECTED <<< "$2"
        for config_num in "${SELECTED[@]}"; do
            for cfg in "${DEFAULT_CONFIGS[@]}"; do
                IFS=':' read -r num workers threads desc <<< "$cfg"
                if [ "$num" = "$config_num" ]; then
                    generate_config "$num" "$workers" "$threads" "$desc"
                    break
                fi
            done
        done
        exit 0
        ;;
    "")
        # Generate all default configs
        echo "Generating all default Apache MPM configurations..."
        echo ""
        for cfg in "${DEFAULT_CONFIGS[@]}"; do
            IFS=':' read -r num workers threads desc <<< "$cfg"
            generate_config "$num" "$workers" "$threads" "$desc"
        done
        echo ""
        echo "Done. Use './switch_config.sh N' to activate a configuration."
;;
"
```

```
    *)
        echo “Unknown option: $1”
        show_usage
        exit 1
        ;;
esac
```

Configuration Switcher

```
#!/bin/bash
CONFIG_DIR=”/etc/httpd/conf.modules.d”
if [ -z “$1” ]; then
    echo “Usage: $0 <config_number>”
    echo “”
    echo “Available configurations:”
    for f in ${CONFIG_DIR}/00-mpm-config*.conf 2>/dev/null; do
        if [ -f “$f” ]; then
            num=$(basename “$f” | sed ‘s/00-mpm-config\([0-9]*\)\.conf/\1/’)
            desc=$(head -2 “$f” | tail -1 | sed ‘s/^# //’)
            echo “  $num: $desc”
        fi
    done
    exit 1
fi

CONFIG_NUM=$1
CONFIG_FILE=”${CONFIG_DIR}/00-mpm-config${CONFIG_NUM}.conf”

if [ ! -f “$CONFIG_FILE” ]; then
    echo “ERROR: Config file not found: $CONFIG_FILE”
    exit 1
fi

echo “Switching to Config ${CONFIG_NUM}...”

sudo cp “$CONFIG_FILE” “${CONFIG_DIR}/00-mpm.conf”
sudo systemctl restart httpd
```

```
sleep 2


echo ""
echo "Active configuration:"
head -3 "${CONFIG_DIR}/00-mpm.conf" | grep -E "^#"
echo ""
echo "Apache workers: $(pgrep -c httpd 2>/dev/null || echo 0)"
echo "Apache status: $(systemctl is-active httpd)"
```

run_benchmark.sh

```
#!/bin/bash
#
# Apache Benchmark Runner
# Runs AB and JMeter tests for R740 or R570 profiles
#
# Usage: ./run_benchmark.sh --profile r740
#        ./run_benchmark.sh --profile r570
#


set -e


CONFIG_DIR="/etc/httpd/conf.modules.d"
JMETER_HOME="/opt/jmeter"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
IP=$(hostname -I | awk '{print $1}')


# Test parameters
AB_REQUESTS=200000
AB_CONCURRENCY=100


# JMeter parameters
JMETER_DURATION=300
JMETER_RAMPUP=60


# Profile definitions
# R740: Configs 1-5, Threads 500-5000
```

```
# R570: Configs 1-8, Threads 500-20000


declare -A PROFILE_CONFIGS

declare -A PROFILE_THREADS

declare -A PROFILE_AB_PARALLEL


PROFILE_CONFIGS[“r740”]=”1 2 3 4 5 6”

PROFILE_CONFIGS[“r570”]=”1 2 3 4 5 6 7 8”


PROFILE_THREADS[“r740”]=”500 1000 2000 3000 5000 8000 10000”

PROFILE_THREADS[“r570”]=”500 1000 2000 3000 5000 8000 10000 16000 20000”


PROFILE_AB_PARALLEL[“r740”]=”1 2 4 6 8 10 12 16 20”

PROFILE_AB_PARALLEL[“r570”]=”1 2 4 6 8 10 12 16 20”


# Parse arguments

PROFILE=””

while [[ $# -gt 0 ]]; do

    case $1 in

        --profile)

            PROFILE=”$2”

            shift 2

            ;;

        --help|-h)

            echo “Usage: $0 --profile <r740|r570>”

            echo “”

            echo “Profiles:”

            echo “  r740: Configs 1-5, threads 500-5000 (~2.5 hours)”

            echo “  r570: Configs 1-8, threads 500-20000 (~6.5 hours)”

            exit 0

            ;;

        *)

            echo “Unknown option: $1”

            exit 1

            ;;

    esac
```

```
done

if [ -z "$PROFILE" ]; then
    echo "Error: --profile is required"
    echo "Usage: $0 --profile <r740|r570>"
    exit 1
fi

if [ -z "${PROFILE_CONFIGS[$PROFILE]}" ]; then
    echo "Error: Unknown profile '$PROFILE'. Use r740 or r570."
    exit 1
fi

# Set profile-specific values
read -ra CONFIGS <<< "${PROFILE_CONFIGS[$PROFILE]}"
read -ra JMETER_THREADS <<< "${PROFILE_THREADS[$PROFILE]}"
read -ra AB_PARALLEL <<< "${PROFILE_AB_PARALLEL[$PROFILE]}"

RESULTS_DIR="./benchmark_${PROFILE}_${TIMESTAMP}"

log() {
    echo "[$(date '+%H:%M:%S')] $1"
}

get_timestamp() {
    date '+%Y-%m-%d %H:%M:%S'
}

get_jvm_opts() {
    local threads=$1
    if [ $threads -le 5000 ]; then
        echo "-Xms8g -Xmx8g"
    elif [ $threads -le 10000 ]; then
        echo "-Xms16g -Xmx16g -Xss512k"
    elif [ $threads -le 20000 ]; then
        echo "-Xms32g -Xmx32g -Xss256k"
```

```
    else
        echo "-Xms64g -Xmx64g -Xss256k -XX:MaxMetaspaceSize=512m"
    fi
}

get_config_info() {
    local config_num=$1
    local config_file="${CONFIG_DIR}/00-mpm-config${config_num}.conf"
    if [ -f "$config_file" ]; then
        head -2 "$config_file" | tail -1 | sed 's/^# //'
    else
        echo "Config ${config_num} (not found)"
    fi
}

switch_config() {
    local config_num=$1
    local config_file="${CONFIG_DIR}/00-mpm-config${config_num}.conf"

    if [ ! -f "$config_file" ]; then
        log "ERROR: Config file not found: $config_file"
        return 1
    fi

    sudo cp "$config_file" "${CONFIG_DIR}/00-mpm.conf"
    sudo systemctl restart httpd
    sleep 3

    if ! systemctl is-active --quiet httpd; then
        log "ERROR: Apache failed to start"
        return 1
    fi

    log "Config ${config_num} active ($(pgrep -c httpd) processes)"
}
```

```
run_ab_test() {
    local config=$1
    local instances=$2
    local requests_per=$((AB_REQUESTS / instances))
    local start_time=$(get_timestamp)

    for i in $(seq 1 $instances); do
        ab -n $requests_per -c $AB_CONCURRENCY -k "http://${IP}/" > "/tmp/ab_${i}.txt" 2>&1 &
    done
    wait

    local end_time=$(get_timestamp)
    local total_rps=0
    local total_failed=0
    local total_latency=0
    local count=0

    for i in $(seq 1 $instances); do
        if [ -f "/tmp/ab_${i}.txt" ]; then
            local rps=$(grep "Requests per second" "/tmp/ab_${i}.txt" | awk '{print $4}')
            local failed=$(grep "Failed requests" "/tmp/ab_${i}.txt" | awk '{print $3}')
            local latency=$(grep "Time per request" "/tmp/ab_${i}.txt" | head -1 | awk '{print $4}')

            if [ -n "$rps" ]; then
                total_rps=$(echo "$total_rps + $rps" | bc)
                total_failed=$((total_failed + ${failed:-0}))
                total_latency=$(echo "$total_latency + ${latency:-0}" | bc)
                count=$((count + 1))
            fi
        fi
    done

    local avg_latency=0
    [ $count -gt 0 ] && avg_latency=$(echo "scale=3; $total_latency / $count" | bc)

    rm -f /tmp/ab_*.txt
```

```
    echo "$config,$instances,$total_rps,$avg_latency,$total_failed,$start_time,$end_time"
}

create_jmeter_plan() {
    local threads=$1
    local plan_file="$RESULTS_DIR/jmeter_plan_${threads}t.jmx"

    cat > "$plan_file" << EOF
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="5.0" jmeter="5.6.3">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Load Test" enabled="true">
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments"/>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Users" enabled="true">
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
        <elementProp name="ThreadGroup.main_controller" elementType="LoopController">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <intProp name="LoopController.loops">-1</intProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">${threads}</stringProp>
        <stringProp name="ThreadGroup.ramp_time">${JMETER_RAMPUP}</stringProp>
        <stringProp name="ThreadGroup.duration">${JMETER_DURATION}</stringProp>
        <boolProp name="ThreadGroup.scheduler">true</boolProp>
      </ThreadGroup>
      <hashTree>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="GET /"
enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments"/>
          <stringProp name="HTTPSampler.domain">${IP}</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.path">/</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        </HTTPSamplerProxy>
```

```
        <hashTree/>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="GET /about.
html" enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments"/>
          <stringProp name="HTTPSampler.domain">${IP}</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.path">/about.html</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        </HTTPSamplerProxy>
        <hashTree/>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="GET /api/
data" enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments"/>
          <stringProp name="HTTPSampler.domain">${IP}</stringProp>
          <stringProp name="HTTPSampler.port">80</stringProp>
          <stringProp name="HTTPSampler.path">/api/data</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
        </HTTPSamplerProxy>
        <hashTree/>
      </hashTree>
    </hashTree>
  </hashTree>
</jmeterTestPlan>
EOF
    echo "$plan_file"
}

run_jmeter_test() {
    local config=$1
    local threads=$2
    local jvm_opts=$(get_jvm_opts $threads)
    local plan_file=$(create_jmeter_plan $threads)
    local results_file="$RESULTS_DIR/jmeter_c${config}_t${threads}.jtl"
    local log_file="$RESULTS_DIR/jmeter_c${config}_t${threads}.log"
```

```
    # Precise JMeter timing (excludes parsing)
    local jmeter_start=$(date -Iseconds)

    JVM_ARGS="$jvm_opts" $JMETER_HOME/bin/jmeter -n \
        -t "$plan_file" \
        -l "$results_file" \
        -j "$log_file" > /dev/null 2>&1

    local jmeter_end=$(date -Iseconds)

    # Parse results (timing not included in output)
    local rps=0
    local avg_latency=0
    local errors=0
    local total=0

    if [ -f "$results_file" ]; then
        total=$(tail -n +2 "$results_file" | wc -l)
        errors=$(tail -n +2 "$results_file" | awk -F',' '$8!="true"' | wc -l)
        local start_ts=$(tail -n +2 "$results_file" | head -1 | cut -d',' -f1)
        local end_ts=$(tail -n +2 "$results_file" | tail -1 | cut -d',' -f1)
        local duration_ms=$((end_ts - start_ts))
        local duration_sec=$(echo "scale=2; $duration_ms / 1000" | bc)
        rps=$(echo "scale=2; $total / $duration_sec" | bc 2>/dev/null || echo "0")
        avg_latency=$(tail -n +2 "$results_file" | awk -F',' '{sum+=$2; count++} END {if(count>0) printf
"%.2f", sum/count; else print "0"}')
    fi

    echo "$config,$threads,$rps,$avg_latency,$errors,$total,$jmeter_start,$jmeter_end"
}

generate_summary() {
    local report="$RESULTS_DIR/SUMMARY.md"

    cat > "$report" << EOF
```

```
# Apache Benchmark Results

**Profile:** ${PROFILE}
**Host:** $(hostname)
**Date:** $(date)
**Target:** http://${IP}/

## Test Configuration

- **Configs tested:** ${CONFIGS[*]}
- **JMeter threads:** ${JMETER_THREADS[*]}
- **JMeter duration:** ${JMETER_DURATION}s per test
- **AB parallel instances:** ${AB_PARALLEL[*]}

## Apache Configurations

EOF

    for config in "${CONFIGS[@]}"; do
        echo "- **Config ${config}:** $(get_config_info $config)" >> "$report"
    done

    cat >> "$report" << EOF

## ApacheBench Results

| Config | Parallel | Req/Sec | Avg Latency (ms) | Failed |
|--------|----------|---------|------------------|--------|
EOF

    while IFS=',' read -r cfg inst rps lat fail start end; do
        [ "$cfg" = "config" ] && continue
        printf "| %s | %s | %s | %s | %s |\n" "$cfg" "$inst" "$rps" "$lat" "$fail" >> "$report"
    done < "$RESULTS_DIR/ab_results.csv"

    cat >> "$report" << EOF
```

```
## JMeter Results

| Config | Threads | Req/Sec | Avg Latency (ms) | Errors | Total Requests |
|--------|---------|---------|------------------|--------|----------------|
EOF

    while IFS=',' read -r cfg thr rps lat err total start end; do
        [ "$cfg" = "config" ] && continue
        printf "| %s | %s | %s | %s | %s | %s |\n" "$cfg" "$thr" "$rps" "$lat" "$err" "$total" >>
"$report"
    done < "$RESULTS_DIR/jmeter_results.csv"

    echo "" >> "$report"
    echo "---" >> "$report"
    echo "Generated by run_benchmark.sh" >> "$report"

    log "Summary saved to $report"
}

# Main execution
main() {
    log "============================================="
    log "Apache Benchmark Runner"
    log "============================================="
    log "Profile: ${PROFILE}"
    log "Host: $(hostname)"
    log "Target: http://${IP}/"
    log "Configs: ${CONFIGS[*]}"
    log "JMeter threads: ${JMETER_THREADS[*]}"
    log "JMeter duration: ${JMETER_DURATION}s"
    log ""

    mkdir -p "$RESULTS_DIR"

    # Initialize CSVs
```

```
    echo "config,parallel,req_per_sec,avg_latency_ms,failed,start_time,end_time" > "$RESULTS_DIR/ab_
results.csv"
    echo "config,threads,req_per_sec,avg_latency_ms,errors,total_requests,jmeter_start,jmeter_end" >
"$RESULTS_DIR/jmeter_results.csv"

    # Estimate time
    local ab_tests=$((${#CONFIGS[@]} * ${#AB_PARALLEL[@]}))
    local jmeter_tests=$((${#CONFIGS[@]} * ${#JMETER_THREADS[@]}))
    local jmeter_minutes=$((jmeter_tests * (JMETER_DURATION + 60) / 60))
    log "Estimated time: ~${jmeter_minutes} minutes for JMeter + AB tests"
    log ""

    # Phase 1: ApacheBench
    log "=============================================="
    log "PHASE 1: ApacheBench Tests"
    log "=============================================="

    for config in "${CONFIGS[@]}"; do
        log ""
        log "--- Config ${config}: $(get_config_info $config) ---"
        switch_config $config

        for instances in "${AB_PARALLEL[@]}"; do
            log "  AB ${instances}x parallel..."
            result=$(run_ab_test $config $instances)
            echo "$result" >> "$RESULTS_DIR/ab_results.csv"

            rps=$(echo "$result" | cut -d',' -f3)
            failed=$(echo "$result" | cut -d',' -f5)
            log "    -> $rps req/sec (failed: $failed)"

            sleep 2
        done
    done

    # Phase 2: JMeter
```

```
log ""
log "=============================================="
log "PHASE 2: JMeter Tests (${JMETER_DURATION}s per test)"
log "=============================================="

for config in "${CONFIGS[@]}"; do
    log ""
    log "--- Config ${config}: $(get_config_info $config) ---"
    switch_config $config

    for threads in "${JMETER_THREADS[@]}"; do
        jvm_opts=$(get_jvm_opts $threads)
        log "  JMeter ${threads}t (${JMETER_DURATION}s)..."
        log "    JVM: ${jvm_opts}"
        result=$(run_jmeter_test $config $threads)
        echo "$result" >> "$RESULTS_DIR/jmeter_results.csv"

        rps=$(echo "$result" | cut -d',' -f3)
        errors=$(echo "$result" | cut -d',' -f5)
        log "    -> $rps req/sec ($errors errors)"

        sleep 3
    done
done

# Generate summary
log ""
log "=============================================="
log "Generating Summary"
log "=============================================="
generate_summary

log ""
log "=============================================="
log "COMPLETE"
log "=============================================="
```

```
    log “Results: $RESULTS_DIR”
}
main
```

abj_runner.sh

```
#!/bin/bash
# Define the optstring
optstring=”rspt:o:v:w:n:”
# Define the variables to store the option values
# Parse the command-line arguments
while getopts “$optstring” opt; do
  case “$opt” in
    t) test_path=”$OPTARG” ;;
    o) requested_output_path=”$OPTARG” ;;
    v) volume=”$OPTARG” ;;
  esac
done

# Sanity check arguments
if [[ -z “$test_path” || -z “$requested_output_path” || -z “$volume”  ]]; then
  echo “Error: missing flags!  All below are required
        -t path to test files
        -o directory to create and save output into
        -v volume to be tested”
  exit 1
fi

# Extract the device name from the volume path
short_volume=$(basename $volume)

ts=$(date +%s)
output_path=${requested_output_path}.${ts}

# Prepare files and output path
mkdir -p $output_path
cp $test_path/$testType* $output_path/
```

```
# start the monitors
nmon -F $output_path/nmon.out -s2 -c100000 -t &

# Initialize d_pid
d_pid=""

# Check if mapped volume or device and start dstat accordingly
if [[ $short_volume == md* ]]; then
        dstat -tam -C total --md -M total,$short_volume --output $output_path/dstat.out 2 &
        d_pid=$!
elif [[ $short_volume == sd* || $short_volume == nv* ]]; then
        dstat -tam -C total -dD total,$short_volume --output $output_path/dstat.out 2 &
        d_pid=$!
fi

iostat -k -t -o JSON -cdx $volume 2 > $output_path/iostat.out &
sleep 2

# Run the tests
configs=(1,2,3,4,5,6)
{
    echo "Start: $(date -Iseconds)"
    /home/user/benchmark/scripts/run_benchmark.sh --configs $configs
    echo "End: $(date -Iseconds)"
} | tee $output_path/ab-jm_config_${configs}.ra

# Cleanup monitors
killall nmon
if [[ -n "$d_pid" ]]; then
    kill $d_pid
fi
killall iostat
echo "]}]}}" >> $output_path/iostat.out
```

OpenResty Files

This section contains the content for the files referenced for the OpenResty testing.

Or_runner.sh

This script content is used to handle the monitoring and triggering of the WRK tests against the OpenResty server.

```
#!/bin/bash

# OpenResty + Redis Benchmark Runner Script

# Starts monitoring, runs the benchmark, stops monitoring


# Define the optstring

optstring="o:v:s:t:"


# Parse the command-line arguments

while getopts "$optstring" opt; do

  case "$opt" in

    o) requested_output_path="$OPTARG" ;;

    v) volume="$OPTARG" ;;

    s) system_type="$OPTARG" ;;

    t) test_path="$OPTARG" ;;

  esac

done


# Sanity check arguments

if [[ -z "$requested_output_path" || -z "$volume" || -z "$system_type" || -z "$test_path" ]]; then

  echo "Error: missing flags!  All below are required

        -o directory to create and save output into

        -v volume to be tested (e.g., /dev/sda)

        -s system type (r740 or r570)

        -t path to scripts (copied to output for reproducibility)"

  exit 1

fi


# Validate system type

if [[ "$system_type" != "r740" && "$system_type" != "r570" ]]; then

  echo "Error: system type must be 'r740' or 'r570'"

  exit 1
```

```
fi

# Path to the OpenResty benchmark script
BENCHMARK_SCRIPT="/home/user/benchmark/scripts/openresty-redis.sh"

if [[ ! -x "$BENCHMARK_SCRIPT" ]]; then
  echo "Error: benchmark script not found or not executable: $BENCHMARK_SCRIPT"
  exit 1
fi

# Extract the device name from the volume path
short_volume=$(basename $volume)

ts=$(date +%s)
output_path=${requested_output_path}.${ts}

# Prepare output path
mkdir -p $output_path

# Copy scripts to output for reproducibility
cp -r $test_path $output_path/scripts_used

echo "============================================="
echo "OpenResty + Redis Benchmark Runner"
echo "============================================="
echo "System Type:  $system_type"
echo "Output Path:  $output_path"
echo "Volume:       $volume"
echo "Scripts From: $test_path"
echo "Started:      $(date)"
echo "============================================="

# Start the monitors
nmon -F $output_path/nmon.out -s2 -c100000 -t &

# Initialize d_pid
```

```
d_pid=""

# Check if mapped volume or device and start dstat accordingly
if [[ $short_volume == md* ]]; then
        dstat -tam -C total --md -M total,$short_volume --output $output_path/dstat.out 2 &
        d_pid=$!
elif [[ $short_volume == sd* || $short_volume == nv* ]]; then
        dstat -tam -C total -dD total,$short_volume --output $output_path/dstat.out 2 &
        d_pid=$!
fi

iostat -k -t -o JSON -cdx $volume 2 > $output_path/iostat.out &
sleep 2

# Run the OpenResty benchmark
{
    echo "Start: $(date -Iseconds)"
    URL_LIST_SIZE=8000 SYSTEM_TYPE=$system_type $BENCHMARK_SCRIPT --output-dir $output_path run-optimized
    echo "End: $(date -Iseconds)"
} 2>&1 | tee $output_path/benchmark.log

# Cleanup monitors
killall nmon
if [[ -n "$d_pid" ]]; then
    kill $d_pid
fi
killall iostat
echo "]}]}}" >> $output_path/iostat.out

echo "============================================="
echo "Benchmark Complete"
echo "Output Path:  $output_path"
echo "Finished:     $(date)"
echo "============================================="
```

Openresty-redis.sh

This script contains the setup and configuration instructions for the OpenResty Redis testing.

```
#!/bin/bash

#================================================================================

# OpenResty + Redis Performance Testing Setup Script

# For RHEL 9.5 - SMB Socket Comparison (R740 Dual-Socket vs R570 Single-Socket)


#================================================================================

# COMMAND LINE ARGUMENT PARSING

#================================================================================


OUTPUT_DIR=""

SYSTEM_TYPE="${SYSTEM_TYPE:-r740}"

COMMAND=""

REMAINING_ARGS=()


parse_args() {

    while [[ $# -gt 0 ]]; do

        case "$1" in

            -o|--output-dir)

                if [[ -n "$2" && ! "$2" =~ ^- ]]; then

                    OUTPUT_DIR="$2"

                    shift 2

                else

                    echo "Error: --output-dir requires a directory path"

                    exit 1

                fi

                ;;

            -s|--system)

                if [[ -n "$2" && ! "$2" =~ ^- ]]; then

                    SYSTEM_TYPE="$2"

                    shift 2

                else

                    echo "Error: --system requires a value (r740 or r570)"

                    exit 1
```

```
                fi
                ;;
            -h|--help)
                show_help
                exit 0
                ;;
            -*)
                echo "Unknown option: $1"
                show_help
                exit 1
                ;;
            *)
                COMMAND="$1"
                shift
                REMAINING_ARGS=("$@")
                break
                ;;
        esac
    done
}

show_help() {
    cat << 'EOF'
OpenResty + Redis Performance Testing v7

Usage: ./openresty-redis-v7.sh [OPTIONS] <command> [args...]

Options:
  -o, --output-dir <path>   Set custom output directory for results
                            (default: /home/user/benchmark/results)
  -s, --system <type>       Set system type: r740 or r570
                            (default: r740, or from SYSTEM_TYPE env var)
  -h, --help                Show this help message

Commands:
  install                   Install dependencies
```

```
  generate-images           Generate test image files
  configure <config> <numa|non-numa>   Generate configuration
  start <config> <numa|non-numa>       Start services
  stop                      Stop all services
  test <config> <strategy> <threads> <connections> [duration]
  warmup [redis_count]      Run cache warmup
  run-config <config> <numa|non-numa>  Run tests for single config
  run-optimized             Run streamlined test suite (~5.5-6.5 hours)
  run-phase1                Phase 1: NUMA validation
  run-phase2                Phase 2: Apples-to-apples fixed configs
  run-phase3                Phase 3: Redis scaling (KEY)
  run-phase4                Phase 4: Client bottleneck validation (NEW)
  run-phase5                Phase 5: Ratio validation
  status                    Show service status
  info                      Show system information

Test Matrix (v7):
  Phase 1: NUMA validation (3 configs × numa + non-numa)
  Phase 2: Apples-to-apples (2 fixed configs, non-numa)
  Phase 3: Redis scaling at optimal workers
  Phase 4: Client bottleneck test with extended wrk resources
  Phase 5: Ratio validation around optimal configs

Examples:
  # Run optimized tests with custom output directory
  ./openresty-redis-v7.sh --output-dir ./results-r740 run-optimized

  # Run on R570 with custom output
  ./openresty-redis-v7.sh -s r570 -o ./results-r570 run-optimized

  # Run just Phase 4 (client bottleneck test)
  ./openresty-redis-v7.sh -o ./phase4-results run-phase4
EOF
}

#================================================================================
```

```
# CONFIGURATION
#================================================================================

PROJECT_ROOT="/home/user/benchmark"
USERNAME="user"

# Detected at runtime
TOTAL_CPUS=$(nproc)
NUMA_NODES=$(lscpu | grep "NUMA node(s):" | awk '{print $NF}' 2>/dev/null || echo "1")

# Image generation settings
NUM_IMAGES=8000
MIN_IMAGE_SIZE_KB=100
MAX_IMAGE_SIZE_KB=3000

# URL list size
URL_LIST_SIZE="${URL_LIST_SIZE:-6000}"

# Test settings
TEST_DURATION="${TEST_DURATION:-300}"
WARMUP_DURATION=30
REDIS_MAXMEMORY_GB="${REDIS_MAXMEMORY_GB:-12}"
REDIS_IO_THREADS=4

# Service startup settings
REDIS_STARTUP_WAIT=3
PORT_RELEASE_WAIT=5
PORT_CHECK_RETRIES=10
REDIS_PING_RETRIES=5

# v7: HIGHER CONCURRENCY to push past client bottleneck
# Old v6: ("2:50" "4:100")
# New v7: ("4:200" "8:400")
CONCURRENCY_LEVELS=("4:200" "8:400")

# Extended concurrency for Phase 4 (client bottleneck testing)
```

```
CONCURRENCY_EXTENDED=("16:1600")


#================================================================================
# CONFIGURATION DEFINITIONS - Format: "workers:redis_instances"
#================================================================================


# Phase 1: NUMA Validation (prove non-NUMA wins)
# Only these configs get both NUMA and non-NUMA testing
declare -A PHASE1_CONFIGS
PHASE1_CONFIGS["fixed-1"]="1:1"
PHASE1_CONFIGS["fixed-16"]="16:2"
PHASE1_CONFIGS["fixed-32"]="32:2"


# Phase 2: Apples-to-Apples Fixed Configs (non-NUMA only)
declare -A PHASE2_CONFIGS
PHASE2_CONFIGS["fixed-4"]="4:1"
PHASE2_CONFIGS["fixed-8"]="8:2"


# Phase 3: Redis Scaling at Optimal Workers
# R740: 16 workers (proven optimal), scale Redis 1-8
declare -A R740_PHASE3_CONFIGS
R740_PHASE3_CONFIGS["r740-16w-r1"]="16:1"
R740_PHASE3_CONFIGS["r740-16w-r2"]="16:2"
R740_PHASE3_CONFIGS["r740-16w-r3"]="16:3"
R740_PHASE3_CONFIGS["r740-16w-r4"]="16:4"
R740_PHASE3_CONFIGS["r740-16w-r6"]="16:6"
R740_PHASE3_CONFIGS["r740-16w-r8"]="16:8"


# R570: 32 workers (proven optimal), scale Redis 1-10
declare -A R570_PHASE3_CONFIGS
R570_PHASE3_CONFIGS["r570-32w-r1"]="32:1"
R570_PHASE3_CONFIGS["r570-32w-r2"]="32:2"
R570_PHASE3_CONFIGS["r570-32w-r4"]="32:4"
R570_PHASE3_CONFIGS["r570-32w-r6"]="32:6"
R570_PHASE3_CONFIGS["r570-32w-r8"]="32:8"
R570_PHASE3_CONFIGS["r570-32w-r10"]="32:10"
```

```
# Phase 4: Client Bottleneck Validation (extended wrk resources)
# Test multiple wrk thread/connection combos to find true server limit
declare -A R740_PHASE4_CONFIGS
R740_PHASE4_CONFIGS[“r740-16w-r4-t4c400”]=”16:4”
R740_PHASE4_CONFIGS[“r740-16w-r4-t8c800”]=”16:4”
R740_PHASE4_CONFIGS[“r740-16w-r4-t8c1600”]=”16:4”
R740_PHASE4_CONFIGS[“r740-16w-r4-t16c1600”]=”16:4”

declare -A R570_PHASE4_CONFIGS
R570_PHASE4_CONFIGS[“r570-32w-r8-t8c800”]=”32:8”
R570_PHASE4_CONFIGS[“r570-32w-r8-t16c1600”]=”32:8”
R570_PHASE4_CONFIGS[“r570-32w-r8-t32c3200”]=”32:8”
R570_PHASE4_CONFIGS[“r570-32w-r8-t32c6400”]=”32:8”

# Phase 5: Worker Scaling Validation
# Test range of worker counts to find true optimal
declare -A R740_PHASE5_CONFIGS
R740_PHASE5_CONFIGS[“r740-8w-r4”]=”8:4”
R740_PHASE5_CONFIGS[“r740-12w-r4”]=”12:4”
R740_PHASE5_CONFIGS[“r740-14w-r4”]=”14:4”
R740_PHASE5_CONFIGS[“r740-18w-r4”]=”18:4”
R740_PHASE5_CONFIGS[“r740-20w-r4”]=”20:4”
R740_PHASE5_CONFIGS[“r740-24w-r4”]=”24:4”

declare -A R570_PHASE5_CONFIGS
R570_PHASE5_CONFIGS[“r570-16w-r8”]=”16:8”
R570_PHASE5_CONFIGS[“r570-24w-r8”]=”24:8”
R570_PHASE5_CONFIGS[“r570-32w-r8”]=”32:8”
R570_PHASE5_CONFIGS[“r570-48w-r8”]=”48:8”
R570_PHASE5_CONFIGS[“r570-64w-r8”]=”64:8”
R570_PHASE5_CONFIGS[“r570-86w-r8”]=”86:8”

#==============================================================================
# NUMA TOPOLOGY AND WRK CPU CONFIGURATION
#==============================================================================
```

```
configure_system_settings() {
    case “$SYSTEM_TYPE” in
        r740)
            # R740: 32 CPUs, 2 NUMA nodes with INTERLEAVED numbering
            # NUMA node 0: even CPUs, NUMA node 1: odd CPUs
            WRK_CPUS_STANDARD=”30,31”
            WRK_CPUS_EXTENDED=”24-31”       # 8 CPUs for extended testing
            WRK_THREADS_STANDARD=8
            WRK_THREADS_EXTENDED=8          # Limited by available CPUs
            NUMA_NODE_CPUS=(“0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30” “1,3,5,7,9,11,13,15,17,19,21,23,
25,27,29,31”)
            REDIS_CPUS_NODE0=”0,2”
            REDIS_CPUS_NODE1=”1,3”
            ;;
        r570)
            # R570: 172 CPUs, 3 SNC nodes
            WRK_CPUS_STANDARD=”168-171”
            WRK_CPUS_EXTENDED=”156-171”     # 16 CPUs for extended testing
            WRK_CPUS_MAX=”140-171”          # 32 CPUs for max testing
            WRK_THREADS_STANDARD=8
            WRK_THREADS_EXTENDED=16
            WRK_THREADS_MAX=32
            NUMA_NODE_CPUS=(“0-56” “57-113” “114-171”)
            REDIS_CPUS_NODE0=”0-3”
            REDIS_CPUS_NODE1=”57-60”
            REDIS_CPUS_NODE2=”114-117”
            ;;
        *)
            echo “ERROR: Unknown SYSTEM_TYPE: $SYSTEM_TYPE”
            exit 1
            ;;
    esac
}

#================================================================================
```

```
# DIRECTORY STRUCTURE
#==============================================================================

setup_directories() {
    IMAGES_DIR="${PROJECT_ROOT}/images"
    CONFIGS_DIR="${PROJECT_ROOT}/configs"
    SCRIPTS_DIR="${PROJECT_ROOT}/scripts"
    LOGS_DIR="${PROJECT_ROOT}/logs"
    RUN_DIR="${PROJECT_ROOT}/run"
    REDIS_DATA_DIR="${PROJECT_ROOT}/redis-data"

    if [[ -n "$OUTPUT_DIR" ]]; then
        RESULTS_DIR="$OUTPUT_DIR"
    else
        RESULTS_DIR="${PROJECT_ROOT}/results"
    fi
}

create_directories() {
    mkdir -p "${IMAGES_DIR}"
    mkdir -p "${CONFIGS_DIR}/redis"
    mkdir -p "${CONFIGS_DIR}/nginx"
    mkdir -p "${SCRIPTS_DIR}"
    mkdir -p "${RESULTS_DIR}"
    mkdir -p "${LOGS_DIR}"
    mkdir -p "${RUN_DIR}"
    mkdir -p "${REDIS_DATA_DIR}"

    print_info "Results will be saved to: ${RESULTS_DIR}"
}

#==============================================================================
# UTILITY FUNCTIONS
#==============================================================================

print_header() {
```

```
    echo ""
    echo "================================================================================"
    echo "$1"
    echo "================================================================================"
}

print_info() {
    echo "[INFO] $(date '+%Y-%m-%d %H:%M:%S') $1"
}

print_error() {
    echo "[ERROR] $(date '+%Y-%m-%d %H:%M:%S') $1" >&2
}

print_warning() {
    echo "[WARN] $(date '+%Y-%m-%d %H:%M:%S') $1" >&2
}

show_system_info() {
    print_header "System Information"
    echo "System Type:       ${SYSTEM_TYPE}"
    echo "Total CPUs:        ${TOTAL_CPUS}"
    echo "NUMA Nodes:        ${NUMA_NODES}"
    echo "WRK CPUs Standard: ${WRK_CPUS_STANDARD}"
    echo "WRK CPUs Extended: ${WRK_CPUS_EXTENDED}"
    echo ""
    echo "Concurrency Levels: ${CONCURRENCY_LEVELS[*]}"
    echo "Extended Concurrency: ${CONCURRENCY_EXTENDED[*]}"
    echo ""
    echo "Directory Structure:"
    echo "  Project Root:  ${PROJECT_ROOT}"
    echo "  Results:       ${RESULTS_DIR}"
    echo ""
    echo "NUMA Topology:"
    lscpu | grep -E "NUMA node[0-9]" 2>/dev/null || echo "  Unable to detect"
}
```

```
get_config_spec() {
    local config_name="$1"

    # Phase 1
    [[ -n "${PHASE1_CONFIGS[$config_name]}" ]] && echo "${PHASE1_CONFIGS[$config_name]}" && return

    # Phase 2
    [[ -n "${PHASE2_CONFIGS[$config_name]}" ]] && echo "${PHASE2_CONFIGS[$config_name]}" && return

    # System-specific phases
    if [[ "$SYSTEM_TYPE" == "r740" ]]; then
        [[ -n "${R740_PHASE3_CONFIGS[$config_name]}" ]] && echo "${R740_PHASE3_CONFIGS[$config_name]}" &&
return
        [[ -n "${R740_PHASE4_CONFIGS[$config_name]}" ]] && echo "${R740_PHASE4_CONFIGS[$config_name]}" &&
return
        [[ -n "${R740_PHASE5_CONFIGS[$config_name]}" ]] && echo "${R740_PHASE5_CONFIGS[$config_name]}" &&
return
    fi

    if [[ "$SYSTEM_TYPE" == "r570" ]]; then
        [[ -n "${R570_PHASE3_CONFIGS[$config_name]}" ]] && echo "${R570_PHASE3_CONFIGS[$config_name]}" &&
return
        [[ -n "${R570_PHASE4_CONFIGS[$config_name]}" ]] && echo "${R570_PHASE4_CONFIGS[$config_name]}" &&
return
        [[ -n "${R570_PHASE5_CONFIGS[$config_name]}" ]] && echo "${R570_PHASE5_CONFIGS[$config_name]}" &&
return
    fi

    print_error "Unknown config: $config_name"
    return 1
}

#=============================================================================
# PORT MANAGEMENT
#=============================================================================

check_port_in_use() {
```

```
    local port="$1"
    ss -tuln 2>/dev/null | grep -q ":${port} " && return 0
    return 1
}

wait_for_port_free() {
    local port="$1"
    local max_wait="${2:-$PORT_CHECK_RETRIES}"
    local waited=0

    while check_port_in_use $port && [[ $waited -lt $max_wait ]]; do
        sleep 1
        ((waited++))
    done

    if check_port_in_use $port; then
        print_warning "Port $port still in use, forcing..."
        sudo fuser -k $port/tcp 2>/dev/null || true
        sleep 2
    fi

    ! check_port_in_use $port
}

#===============================================================================
# INSTALLATION
#===============================================================================

install_dependencies() {
    print_header "Installing Dependencies"

    print_info "Adding OpenResty repository..."
    sudo dnf install -y yum-utils
    sudo yum-config-manager --add-repo https://openresty.org/package/rhel/openresty.repo
    sudo dnf makecache
```

```
    print_info "Installing packages..."
    sudo dnf install -y --nogpgcheck \
        openresty \
        openresty-resty \
        redis \
        numactl \
        git \
        gcc \
        make \
        openssl-devel \
        lsof

    if ! command -v wrk &> /dev/null; then
        print_info "Installing wrk from source..."
        cd /tmp
        rm -rf wrk
        git clone https://github.com/wg/wrk.git
        cd wrk
        make -j$(nproc)
        sudo cp wrk /usr/local/bin/
        cd /tmp
        rm -rf wrk
    fi

    print_info "Installing lua-resty-redis..."
    sudo mkdir -p /usr/local/openresty/lualib/resty
    curl -sL https://raw.githubusercontent.com/openresty/lua-resty-redis/master/lib/resty/redis.lua \
        | sudo tee /usr/local/openresty/lualib/resty/redis.lua > /dev/null

    print_info "Configuring SELinux..."
    sudo setsebool -P httpd_can_network_connect 1 2>/dev/null || true

    print_info "Configuring system limits..."
    sudo tee /etc/security/limits.d/openresty.conf > /dev/null << 'EOF'
* soft nofile 200000
* hard nofile 200000
```

```
* soft nproc 65535

* hard nproc 65535

EOF


    sudo tee /etc/sysctl.d/99-openresty.conf > /dev/null << 'EOF'

net.core.somaxconn = 65535

net.ipv4.tcp_max_syn_backlog = 65535

net.core.netdev_max_backlog = 65535

net.ipv4.tcp_tw_reuse = 1

net.ipv4.ip_local_port_range = 1024 65535

EOF

    sudo sysctl -p /etc/sysctl.d/99-openresty.conf 2>/dev/null || true


    print_info "Installation complete!"

}


#================================================================================

# IMAGE GENERATION

#================================================================================


generate_images() {

    print_header "Generating Test Images"

    print_info "Generating ${NUM_IMAGES} images (${MIN_IMAGE_SIZE_KB}KB - ${MAX_IMAGE_SIZE_KB}KB)"


    create_directories

    cd "${IMAGES_DIR}"


    local count=0

    local range=$((MAX_IMAGE_SIZE_KB - MIN_IMAGE_SIZE_KB))


    for i in $(seq 1 ${NUM_IMAGES}); do

        local size=$((MIN_IMAGE_SIZE_KB + RANDOM % range))

        dd if=/dev/urandom of=product_${i}.jpg bs=1024 count=$size 2>/dev/null


        ((count++))

        if [[ $((count % 500)) -eq 0 ]]; then
```

```
            echo "Generated $count files... ($(du -sh . | awk '{print $1}'))"
        fi
    done

    chmod 644 "${IMAGES_DIR}"/*.jpg

    print_info "Generation complete!"
    echo "Total files: $(ls -1 *.jpg 2>/dev/null | wc -l)"
    echo "Total size:  $(du -sh "${IMAGES_DIR}" | awk '{print $1}')"
}

#===============================================================================
# URL LIST AND WRK SCRIPT
#===============================================================================

generate_url_list() {
    local url_file="${SCRIPTS_DIR}/test_urls.txt"
    mkdir -p "${SCRIPTS_DIR}"
    rm -f "${url_file}"

    local num_images=$(ls -1 "${IMAGES_DIR}"/*.jpg 2>/dev/null | wc -l)
    local list_size=$URL_LIST_SIZE
    [[ $list_size -gt $num_images ]] && list_size=$num_images

    for i in $(shuf -i 1-${num_images} -n ${list_size}); do
        echo "/images/product_${i}.jpg" >> "${url_file}"
    done

    print_info "Generated ${list_size} URLs in ${url_file}"
}

generate_wrk_script() {
    cat > "${SCRIPTS_DIR}/wrk_bench.lua" << EOF
local urls = {}
local url_file = "${SCRIPTS_DIR}/test_urls.txt"
```

```
for line in io.lines(url_file) do
    table.insert(urls, line)
end

local url_count = #urls

request = function()
    return wrk.format("GET", urls[math.random(url_count)])
end
EOF
    print_info "Generated wrk script"
}

#==============================================================================
# REDIS CONFIGURATION
#==============================================================================

generate_redis_config() {
    local config_name="$1"
    local numa_strategy="$2"
    local num_instances="$3"

    local config_dir="${CONFIGS_DIR}/redis/${config_name}-${numa_strategy}"
    mkdir -p "${config_dir}"
    mkdir -p "${REDIS_DATA_DIR}"

    for ((i=0; i<num_instances; i++)); do
        local port=$((6379 + i))

        cat > "${config_dir}/redis-${port}.conf" << EOF
bind 127.0.0.1
port ${port}
daemonize yes
pidfile ${RUN_DIR}/redis_${port}.pid
logfile ${LOGS_DIR}/redis-${port}.log
dir ${REDIS_DATA_DIR}
```

```
maxmemory ${REDIS_MAXMEMORY_GB}gb
maxmemory-policy allkeys-lru
save “”
appendonly no

io-threads ${REDIS_IO_THREADS}
io-threads-do-reads yes

tcp-backlog 4096
tcp-keepalive 60
timeout 0

lazyfree-lazy-eviction yes
lazyfree-lazy-expire yes
hz 100
dynamic-hz yes
EOF
    done
}

#==============================================================================
# NGINX CONFIGURATION
#==============================================================================

generate_nginx_config() {
    local config_name=”$1”
    local numa_strategy=”$2”
    local num_workers=”$3”
    local num_redis=”$4”

    local config_file=”${CONFIGS_DIR}/nginx/nginx-${config_name}-${numa_strategy}.conf”

    local cpu_affinity_directive=””
    if [[ “$numa_strategy” == “numa” ]]; then
        cpu_affinity_directive=$(generate_cpu_affinity “$num_workers” “$num_redis”)
```

```
    fi

    local redis_selection_lua
    if [[ "$numa_strategy" == "numa" ]]; then
        redis_selection_lua=$(generate_numa_aware_lua "$num_workers" "$num_redis")
    else
        redis_selection_lua=$(generate_round_robin_lua "$num_redis")
    fi

    cat > "${config_file}" << NGINXEOF
# OpenResty Configuration - v7
# Config: ${config_name} | Strategy: ${numa_strategy}
# Workers: ${num_workers} | Redis: ${num_redis}
# System: ${SYSTEM_TYPE}

worker_processes ${num_workers};
worker_rlimit_nofile 200000;
pid ${RUN_DIR}/nginx.pid;
error_log ${LOGS_DIR}/nginx-error.log crit;
${cpu_affinity_directive}

events {
    worker_connections 10000;
    use epoll;
    multi_accept on;
}

http {
    include /usr/local/openresty/nginx/conf/mime.types;
    default_type application/octet-stream;

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    keepalive_requests 10000;
```

```
    access_log off;

    lua_shared_dict counters 1m;

    init_worker_by_lua_block {
        ngx.shared.counters:set("requests", 0)
    }

    server {
        listen 8080 reuseport backlog=32768;
        server_name localhost;
        root ${IMAGES_DIR};

        location ~* \.(jpg|jpeg|png|gif)$ {
            content_by_lua_block {
                local redis = require "resty.redis"
                local counters = ngx.shared.counters

${redis_selection_lua}

                local key = ngx.var.uri
                local red = redis:new()
                red:set_timeouts(500, 500, 500)

                local ok = red:connect("127.0.0.1", redis_port)
                if not ok then
                    ngx.exit(ngx.HTTP_SERVICE_UNAVAILABLE)
                    return
                end

                local res = red:get(key)

                if res and res ~= ngx.null then
                    ngx.header["Content-Type"] = "image/jpeg"
                    ngx.header["X-Cache-Status"] = "HIT"
```

```
                ngx.header[“X-Redis-Port”] = tostring(redis_port)
                ngx.print(res)
                red:set_keepalive(60000, 200)
                return
            end

            local file_path = ngx.var.uri
            if file_path:sub(1, 8) == “/images/” then
                file_path = file_path:sub(9)
            end
            local full_path = “${IMAGES_DIR}/” .. file_path

            local file = io.open(full_path, “rb”)
            if not file then
                red:set_keepalive(60000, 200)
                ngx.exit(ngx.HTTP_NOT_FOUND)
                return
            end

            local content = file:read(“*all”)
            file:close()

            red:setex(key, 7200, content)

            ngx.header[“Content-Type”] = “image/jpeg”
            ngx.header[“X-Cache-Status”] = “MISS”
            ngx.header[“X-Redis-Port”] = tostring(redis_port)
            ngx.print(content)

            red:set_keepalive(60000, 200)
        }
    }

    location / {
        return 404;
    }
```

```
    }
}
NGINXEOF
}

generate_cpu_affinity() {
    local num_workers="$1"

    if [[ "$SYSTEM_TYPE" == "r570" ]]; then
        echo "worker_cpu_affinity auto;"
        return
    fi

    # R740: Generate explicit CPU affinity masks
    local node0_cpus=(4 6 8 10 12 14 16 18 20 22 24 26 28)
    local node1_cpus=(5 7 9 11 13 15 17 19 21 23 25 27 29)

    local workers_per_node=$(( (num_workers + 1) / 2 ))
    local affinity_masks=()
    local worker_idx=0

    for ((i=0; i<workers_per_node && worker_idx<num_workers; i++)); do
        local cpu=${node0_cpus[$((i % ${#node0_cpus[@]}))]}
        local mask=$(printf "%032d" $(echo "obase=2; $((1 << cpu))" | bc))
        affinity_masks+=("$mask")
        ((worker_idx++))
    done

    for ((i=0; worker_idx<num_workers; i++)); do
        local cpu=${node1_cpus[$((i % ${#node1_cpus[@]}))]}
        local mask=$(printf "%032d" $(echo "obase=2; $((1 << cpu))" | bc))
        affinity_masks+=("$mask")
        ((worker_idx++))
    done

    echo "worker_cpu_affinity ${affinity_masks[*]};"
```

```
}

generate_numa_aware_lua() {
    local num_workers="$1"
    local num_redis="$2"

    if [[ "$num_redis" -eq 1 ]]; then
        echo '                local redis_port = 6379'
        return
    fi

    local workers_per_node=$((num_workers / 2))

    cat << EOF
                local worker_id = ngx.worker.id() or 0
                local redis_port
                if worker_id < ${workers_per_node} then
                    redis_port = 6379
                else
                    redis_port = 6380
                end
EOF
}

generate_round_robin_lua() {
    local num_redis="$1"

    cat << EOF
                local req_count = counters:incr("requests", 1, 0)
                local redis_port = 6379 + (req_count % ${num_redis})
EOF
}

#================================================================================
# SERVICE MANAGEMENT
#================================================================================
```

```
stop_services() {
    print_info “Stopping services...”

    # Graceful nginx stop
    sudo /usr/local/openresty/nginx/sbin/nginx -s stop 2>/dev/null || true
    sleep 1

    # Graceful Redis shutdown
    for port in $(seq 6379 6394); do
        redis-cli -p $port SHUTDOWN NOSAVE 2>/dev/null || true
    done
    sleep 2

    # Force kill remaining
    sudo pkill -9 nginx 2>/dev/null || true
    sudo pkill -9 redis-server 2>/dev/null || true
    sleep 2

    # Clean PID files
    rm -f ${RUN_DIR}/nginx.pid ${RUN_DIR}/redis_*.pid 2>/dev/null || true

    # Verify ports free
    for port in 8080 6379 6380 6381 6382 6383 6384 6385 6386 6387 6388; do
        if check_port_in_use $port; then
            sudo fuser -k ${port}/tcp 2>/dev/null || true
        fi
    done
    sleep 1
}

start_redis() {
    local config_name=”$1”
    local numa_strategy=”$2”
    local num_instances=”$3”
```

```
    local config_dir="${CONFIGS_DIR}/redis/${config_name}-${numa_strategy}"

    print_info "Starting ${num_instances} Redis instances..."

    local failed=0

    for ((i=0; i<num_instances; i++)); do
        local port=$((6379 + i))
        local conf="${config_dir}/redis-${port}.conf"

        [[ ! -f "$conf" ]] && { print_error "Config not found: $conf"; ((failed++)); continue; }

        wait_for_port_free $port 5 || { print_error "Port $port not free"; ((failed++)); continue; }

        rm -f "${LOGS_DIR}/redis-${port}.log"

        if [[ "$numa_strategy" == "numa" ]]; then
            local numa_node=$((i % 2))
            local cpu_range
            [[ $numa_node -eq 0 ]] && cpu_range="0,2" || cpu_range="1,3"

            sudo numactl --cpunodebind=${numa_node} --membind=${numa_node} \
                taskset -c ${cpu_range} redis-server "${conf}" &
        else
            sudo redis-server "${conf}" &
        fi

        sleep $REDIS_STARTUP_WAIT
    done

    sleep 2

    # Verify
    local ok=0
    for ((i=0; i<num_instances; i++)); do
        local port=$((6379 + i))
```

```
        local retries=$REDIS_PING_RETRIES

        while [[ $retries -gt 0 ]]; do
            if redis-cli -p $port PING 2>/dev/null | grep -q PONG; then
                ((ok++))
                break
            fi
            ((retries--))
            sleep 1
        done
    done

    print_info "Redis: $ok/$num_instances running"
    [[ $ok -eq $num_instances ]]
}

start_nginx() {
    local config_name="$1"
    local numa_strategy="$2"

    local config_file="${CONFIGS_DIR}/nginx/nginx-${config_name}-${numa_strategy}.conf"

    print_info "Starting nginx..."

    [[ ! -f "$config_file" ]] && { print_error "Config not found: $config_file"; return 1; }

    # Kill any existing nginx
    if pgrep -x nginx >/dev/null 2>&1; then
        sudo /usr/local/openresty/nginx/sbin/nginx -s stop 2>/dev/null || true
        sleep 1
        sudo pkill -9 nginx 2>/dev/null || true
        sleep 1
    fi

    wait_for_port_free 8080 5 || { print_error "Port 8080 not free"; return 1; }
```

```
    sudo cp "${config_file}" /usr/local/openresty/nginx/conf/nginx.conf

    if ! sudo /usr/local/openresty/nginx/sbin/nginx -t 2>&1; then
        print_error "Nginx config test failed"
        return 1
    fi

    sudo /usr/local/openresty/nginx/sbin/nginx
    sleep 2

    local worker_count=$(ps aux | grep -c "[n]ginx: worker")
    print_info "Nginx: $worker_count workers running"
    [[ $worker_count -gt 0 ]]
}

start_services() {
    local config_name="$1"
    local numa_strategy="$2"

    local config_spec=$(get_config_spec "$config_name") || return 1
    local num_workers="${config_spec%%:*}"
    local num_redis="${config_spec##*:}"

    stop_services

    start_redis "$config_name" "$numa_strategy" "$num_redis" || return 1
    start_nginx "$config_name" "$numa_strategy" || return 1
}

#=============================================================================
# CSV HANDLING
#=============================================================================

init_csv() {
    local csv_file="${RESULTS_DIR}/${SYSTEM_TYPE}_results_v7.csv"
```

```
    if [[ ! -f "$csv_file" ]]; then
        echo "system,config,numa_strategy,workers,redis_instances,wrk_threads,wrk_connections,wrk_
cpus,duration,requests_total,requests_per_sec,transfer_per_sec,latency_avg,latency_stdev,latency_
max,latency_p50,latency_p75,latency_p90,latency_p99,socket_errors,redis_hits,redis_misses,redis_
keys,validation,phase,timestamp" > "$csv_file"
        print_info "Created CSV: $csv_file"
    fi
}

#==============================================================================
# TESTING
#==============================================================================

flush_redis() {
    local num=${1:-16}
    for ((i=0; i<num; i++)); do
        redis-cli -p $((6379 + i)) FLUSHALL 2>/dev/null || true
    done
}

run_single_test() {
    local config_name="$1"
    local numa_strategy="$2"
    local wrk_threads="$3"
    local wrk_connections="$4"
    local wrk_cpus="$5"
    local duration="${6:-$TEST_DURATION}"
    local phase="${7:-standard}"

    local config_spec=$(get_config_spec "$config_name") || return 1
    local num_workers="${config_spec%%:*}"
    local num_redis="${config_spec##*:}"

    flush_redis $num_redis

    print_info "Test: $config_name ($numa_strategy) t=$wrk_threads c=$wrk_connections cpus=$wrk_cpus"
```

```
    local output=$(taskset -c $wrk_cpus wrk \
        -t${wrk_threads} \
        -c${wrk_connections} \
        -d${duration}s \
        --latency \
        -s "${SCRIPTS_DIR}/wrk_bench.lua" \
        http://localhost:8080/ 2>&1)

    # Parse results
    local requests_total=$(echo "$output" | grep "requests in" | awk '{print $1}')
    local rps=$(echo "$output" | grep "Requests/sec:" | awk '{print $2}')
    local transfer=$(echo "$output" | grep "Transfer/sec:" | awk '{print $2}')

    local latency_line=$(echo "$output" | grep "Latency" | head -1)
    local lat_avg=$(echo "$latency_line" | awk '{print $2}')
    local lat_stdev=$(echo "$latency_line" | awk '{print $3}')
    local lat_max=$(echo "$latency_line" | awk '{print $4}')

    local lat_p50=$(echo "$output" | grep "50%" | awk '{print $2}')
    local lat_p75=$(echo "$output" | grep "75%" | awk '{print $2}')
    local lat_p90=$(echo "$output" | grep "90%" | awk '{print $2}')
    local lat_p99=$(echo "$output" | grep "99%" | awk '{print $2}')

    local errors=$(echo "$output" | grep -c "Socket errors" || echo "0")

    # Redis stats
    local total_hits=0 total_misses=0 total_keys=0
    for ((i=0; i<num_redis; i++)); do
        local port=$((6379 + i))
        local hits=$(redis-cli -p $port INFO stats 2>/dev/null | grep "keyspace_hits:" | cut -d: -f2 | tr
-d '\r')
        local misses=$(redis-cli -p $port INFO stats 2>/dev/null | grep "keyspace_misses:" | cut -d: -f2
| tr -d '\r')
        local keys=$(redis-cli -p $port DBSIZE 2>/dev/null | grep -oP '\d+' || echo 0)
        total_hits=$((total_hits + ${hits:-0}))
        total_misses=$((total_misses + ${misses:-0}))
```

```
        total_keys=$((total_keys + ${keys:-0}))
    done

    # Validation
    local validation="VALID"
    local rps_int=${rps%.*}
    rps_int=${rps_int:-0}
    [[ $total_hits -eq 0 && $rps_int -gt 50000 ]] && validation="INVALID:no_redis"
    [[ $total_keys -eq 0 && $total_hits -eq 0 && $rps_int -gt 5000 ]] && validation="INVALID:no_cache"

    # Write CSV
    local csv_file="${RESULTS_DIR}/${SYSTEM_TYPE}_results_v7.csv"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')
    echo "${SYSTEM_TYPE},${config_name},${numa_strategy},${num_workers},${num_redis},${wrk_threads},${wrk_
connections},${wrk_cpus},${duration},${requests_total:-0},${rps:-0},${transfer:-0},${lat_avg:-0},${lat_
stdev:-0},${lat_max:-0},${lat_p50:-0},${lat_p75:-0},${lat_p90:-0},${lat_p99:-0},${errors},${total_
hits},${total_misses},${total_keys},${validation},${phase},${timestamp}" >> "$csv_file"

    print_info "Result: ${rps:-0} req/s | p99: ${lat_p99:-N/A} | ${validation}"

    echo "$rps"
}

run_warmup() {
    local num_redis="${1:-4}"
    print_info "Warmup (${WARMUP_DURATION}s)..."
    flush_redis $num_redis
    taskset -c $WRK_CPUS_STANDARD wrk -t4 -c100 -d${WARMUP_DURATION}s \
        -s "${SCRIPTS_DIR}/wrk_bench.lua" http://localhost:8080/ > /dev/null 2>&1
}

#===============================================================================
# CONFIGURATION GENERATION
#===============================================================================

configure_single() {
    local config_name="$1"
```

```
    local numa_strategy="$2"

    local config_spec=$(get_config_spec "$config_name") || return 1
    local num_workers="${config_spec%%:*}"
    local num_redis="${config_spec##*:}"

    print_info "Configuring: $config_name ($numa_strategy) - ${num_workers}w/${num_redis}r"

    generate_redis_config "$config_name" "$numa_strategy" "$num_redis"
    generate_nginx_config "$config_name" "$numa_strategy" "$num_workers" "$num_redis"
}

#=============================================================================
# PHASE RUNNERS
#=============================================================================

run_config_standard() {
    local config_name="$1"
    local numa_strategy="$2"
    local phase="${3:-standard}"

    local config_spec=$(get_config_spec "$config_name") || return 1
    local num_redis="${config_spec##*:}"

    print_header "Testing: $config_name ($numa_strategy)"

    configure_single "$config_name" "$numa_strategy"
    start_services "$config_name" "$numa_strategy" || return 1
    run_warmup $num_redis

    for level in "${CONCURRENCY_LEVELS[@]}"; do
        local threads="${level%%:*}"
        local conn="${level##*:}"
        run_single_test "$config_name" "$numa_strategy" "$threads" "$conn" "$WRK_CPUS_STANDARD" "$TEST_
DURATION" "$phase"
        sleep 3
```

```
    done
}

run_config_extended() {
    local config_name="$1"
    local numa_strategy="$2"
    local wrk_threads="$3"
    local wrk_cpus="$4"
    local phase="${5:-extended}"

    local config_spec=$(get_config_spec "$config_name") || return 1
    local num_redis="${config_spec##*:}"

    print_header "Extended Test: $config_name (t=$wrk_threads, cpus=$wrk_cpus)"

    configure_single "$config_name" "$numa_strategy"
    start_services "$config_name" "$numa_strategy" || return 1
    run_warmup $num_redis

    for level in "${CONCURRENCY_EXTENDED[@]}"; do
        local conn="${level##*:}"
        run_single_test "$config_name" "$numa_strategy" "$wrk_threads" "$conn" "$wrk_cpus" "$TEST_
DURATION" "$phase"
        sleep 3
    done
}

run_phase1() {
    print_header "Phase 1: NUMA Validation"
    print_info "Testing NUMA vs Non-NUMA on key configs"

    local failed=()

    for config_name in "fixed-1" "fixed-16" "fixed-32"; do
        run_config_standard "$config_name" "non-numa" "phase1" || failed+=("${config_name}-non-numa")
        run_config_standard "$config_name" "numa" "phase1" || failed+=("${config_name}-numa")
```

```
    done

    [[ ${#failed[@]} -gt 0 ]] && print_warning "Failed: ${failed[*]}"
    print_info "Phase 1 complete"
}

run_phase2() {
    print_header "Phase 2: Apples-to-Apples Fixed Configs"
    print_info "Non-NUMA only (proven superior)"

    local failed=()

    for config_name in "fixed-4" "fixed-8"; do
        run_config_standard "$config_name" "non-numa" "phase2" || failed+=("$config_name")
    done

    [[ ${#failed[@]} -gt 0 ]] && print_warning "Failed: ${failed[*]}"
    print_info "Phase 2 complete"
}

run_phase3() {
    print_header "Phase 3: Redis Scaling at Optimal Workers"
    print_info "KEY DISCOVERY: Finding true throughput ceiling"

    local failed=()

    if [[ "$SYSTEM_TYPE" == "r740" ]]; then
        for config_name in "r740-16w-r1" "r740-16w-r2" "r740-16w-r3" "r740-16w-r4" "r740-16w-r6" "r740-
16w-r8"; do
            run_config_standard "$config_name" "non-numa" "phase3" || failed+=("$config_name")
        done
    else
        for config_name in "r570-32w-r1" "r570-32w-r2" "r570-32w-r4" "r570-32w-r6" "r570-32w-r8" "r570-
32w-r10"; do
            run_config_standard "$config_name" "non-numa" "phase3" || failed+=("$config_name")
        done
```

```
    fi

    [[ ${#failed[@]} -gt 0 ]] && print_warning "Failed: ${failed[*]}"
    print_info "Phase 3 complete"
}

run_phase4() {
    print_header "Phase 4: Client Bottleneck Validation"
    print_info "Testing multiple wrk thread/connection combos to find true server limit"

    local failed=()

    if [[ "$SYSTEM_TYPE" == "r740" ]]; then
        # R740: Test with 16w-r4, varying wrk threads and connections
        # All use the same server config, just different wrk parameters
        configure_single "r740-16w-r4-t4c400" "non-numa"
        start_services "r740-16w-r4-t4c400" "non-numa" || { failed+=("r740-phase4"); return 1; }

        local num_redis=4
        run_warmup $num_redis

        # t4c400 - baseline
        run_single_test "r740-16w-r4-t4c400" "non-numa" 4 400 "$WRK_CPUS_STANDARD" "$TEST_DURATION"
"phase4"
        sleep 3

        # t8c800 - double threads and connections
        run_single_test "r740-16w-r4-t8c800" "non-numa" 8 800 "$WRK_CPUS_STANDARD" "$TEST_DURATION"
"phase4"
        sleep 3

        # t8c1600 - same threads, more connections
        run_single_test "r740-16w-r4-t8c1600" "non-numa" 8 1600 "$WRK_CPUS_EXTENDED" "$TEST_DURATION"
"phase4"
        sleep 3

        # t16c1600 - more threads with extended CPUs (R740 limited to 8 CPUs for wrk)
```

```
        run_single_test “r740-16w-r4-t16c1600” “non-numa” 8 1600 “$WRK_CPUS_EXTENDED” “$TEST_DURATION”
“phase4”
        sleep 3

    else
        # R570: Test with 32w-r8, varying wrk threads and connections
        configure_single “r570-32w-r8-t8c800” “non-numa”
        start_services “r570-32w-r8-t8c800” “non-numa” || { failed+=(“r570-phase4”); return 1; }

        local num_redis=8
        run_warmup $num_redis

        # t8c800 - baseline
        run_single_test “r570-32w-r8-t8c800” “non-numa” 8 800 “$WRK_CPUS_STANDARD” “$TEST_DURATION”
“phase4”
        sleep 3

        # t16c1600 - extended
        run_single_test “r570-32w-r8-t16c1600” “non-numa” 16 1600 “$WRK_CPUS_EXTENDED” “$TEST_DURATION”
“phase4”
        sleep 3

        # t32c3200 - max threads
        run_single_test “r570-32w-r8-t32c3200” “non-numa” 32 3200 “$WRK_CPUS_MAX” “$TEST_DURATION”
“phase4”
        sleep 3

        # t32c6400 - max threads, double connections
        run_single_test “r570-32w-r8-t32c6400” “non-numa” 32 6400 “$WRK_CPUS_MAX” “$TEST_DURATION”
“phase4”
        sleep 3
    fi

    [[ ${#failed[@]} -gt 0 ]] && print_warning “Failed: ${failed[*]}”
    print_info “Phase 4 complete”
    print_info “Compare RPS across configs - if RPS increases with more wrk resources, client was
bottleneck”
}
```

```
run_phase5() {
    print_header "Phase 5: Worker Scaling Validation"
    print_info "Testing range of worker counts to find true optimal"

    local failed=()

    if [[ "$SYSTEM_TYPE" == "r740" ]]; then
        # R740: Test 8, 12, 14, 18, 20, 24 workers with r4 Redis
        # (16w-r4 already tested in Phase 3)
        for config_name in "r740-8w-r4" "r740-12w-r4" "r740-14w-r4" "r740-18w-r4" "r740-20w-r4" "r740-
24w-r4"; do
            run_config_standard "$config_name" "non-numa" "phase5" || failed+=("$config_name")
        done
    else
        # R570: Test 16, 24, 32, 48, 64, 86 workers with r8 Redis
        for config_name in "r570-16w-r8" "r570-24w-r8" "r570-32w-r8" "r570-48w-r8" "r570-64w-r8" "r570-
86w-r8"; do
            run_config_standard "$config_name" "non-numa" "phase5" || failed+=("$config_name")
        done
    fi

    [[ ${#failed[@]} -gt 0 ]] && print_warning "Failed: ${failed[*]}"
    print_info "Phase 5 complete"
}

#================================================================================
# MAIN RUNNERS
#================================================================================

run_optimized() {
    print_header "OpenResty + Redis Performance Test v7"
    print_info "System: $SYSTEM_TYPE | Results: $RESULTS_DIR"
    print_info "Estimated time: ~7.5-8.5 hours"

    local start_time=$(date +%s)
```

```
create_directories
generate_url_list
generate_wrk_script
init_csv

# Count tests
local phase1_tests=12  # 3 configs × 2 strategies × 2 concurrency
local phase2_tests=4   # 2 configs × 2 concurrency
local phase3_tests=12  # 6 configs × 2 concurrency (same for both systems)
local phase4_tests=4   # 4 wrk configurations (both systems)
local phase5_tests=12  # R740: 6×2, R570: 6×2

local total=$((phase1_tests + phase2_tests + phase3_tests + phase4_tests + phase5_tests))

print_info "Test plan: $total tests"
print_info "  Phase 1 (NUMA validation): $phase1_tests"
print_info "  Phase 2 (Apples-to-apples): $phase2_tests"
print_info "  Phase 3 (Redis scaling): $phase3_tests"
print_info "  Phase 4 (Bottleneck): $phase4_tests"
print_info "  Phase 5 (Ratio): $phase5_tests"

run_phase1
run_phase2
run_phase3
run_phase4
run_phase5

stop_services

local end_time=$(date +%s)
local duration=$((end_time - start_time))

print_header "Test Complete"
echo "Duration: $((duration / 3600))h $(((duration % 3600) / 60))m $((duration % 60))s"
echo "Results: ${RESULTS_DIR}/${SYSTEM_TYPE}_results_v7.csv"
```

```
}


#================================================================================
# MAIN
#================================================================================


main() {
    parse_args "$@"
    configure_system_settings
    setup_directories


    case "$COMMAND" in
        install)
            create_directories
            install_dependencies
            show_system_info
            ;;
        generate-images)
            create_directories
            generate_images
            ;;
        configure)
            [[ ${#REMAINING_ARGS[@]} -lt 2 ]] && { print_error "Usage: configure <config> <numa|non-numa>";
exit 1; }
            create_directories
            configure_single "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            ;;
        start)
            [[ ${#REMAINING_ARGS[@]} -lt 2 ]] && { print_error "Usage: start <config> <numa|non-numa>";
exit 1; }
            create_directories
            generate_url_list
            generate_wrk_script
            configure_single "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            start_services "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            ;;
```

```
        stop)
            stop_services
            ;;
        test)
            [[ ${#REMAINING_ARGS[@]} -lt 4 ]] && { print_error "Usage: test <config> <numa|non-numa>
<threads> <connections> [duration]"; exit 1; }
            create_directories
            generate_url_list
            generate_wrk_script
            init_csv
            configure_single "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            start_services "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            run_single_test "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}" "${REMAINING_ARGS[2]}"
"${REMAINING_ARGS[3]}" "$WRK_CPUS_STANDARD" "${REMAINING_ARGS[4]:-$TEST_DURATION}" "manual"
            ;;
        warmup)
            run_warmup "${REMAINING_ARGS[0]:-4}"
            ;;
        run-config)
            [[ ${#REMAINING_ARGS[@]} -lt 2 ]] && { print_error "Usage: run-config <config> <numa|non-
numa>"; exit 1; }
            create_directories
            generate_url_list
            generate_wrk_script
            init_csv
            run_config_standard "${REMAINING_ARGS[0]}" "${REMAINING_ARGS[1]}"
            stop_services
            ;;
        run-optimized)
            run_optimized
            ;;
        run-phase1)
            create_directories
            generate_url_list
            generate_wrk_script
            init_csv
            run_phase1
```

```
        stop_services
        ;;
    run-phase2)
        create_directories
        generate_url_list
        generate_wrk_script
        init_csv
        run_phase2
        stop_services
        ;;
    run-phase3)
        create_directories
        generate_url_list
        generate_wrk_script
        init_csv
        run_phase3
        stop_services
        ;;
    run-phase4)
        create_directories
        generate_url_list
        generate_wrk_script
        init_csv
        run_phase4
        stop_services
        ;;
    run-phase5)
        create_directories
        generate_url_list
        generate_wrk_script
        init_csv
        run_phase5
        stop_services
        ;;
    status)
        echo “Nginx workers: $(ps aux | grep -c ‘[n]ginx: worker’ || echo 0)”
```

```
            echo "Redis instances:"
            for port in $(seq 6379 6394); do
                if redis-cli -p $port PING 2>/dev/null | grep -q PONG; then
                    local keys=$(redis-cli -p $port DBSIZE 2>/dev/null | grep -oP '\d+' || echo 0)
                    local mem=$(redis-cli -p $port INFO memory 2>/dev/null | grep "used_memory_human:" |
cut -d: -f2 | tr -d '\r')
                    echo "  Port $port: OK ($keys keys, $mem)"
                fi
            done
            ;;
        info)
            create_directories
            show_system_info
            ;;
        ""|help)
            show_help
            ;;
        *)
            print_error "Unknown command: $COMMAND"
            show_help
            exit 1
            ;;
    esac
}

main "$@"
```