An approach to identify performance influence on web services, using different parallel execution techniques

S.C Rangoda

2019

An approach to identify performance influence on web services, using different parallel execution techniques

A dissertation submitted for the Degree of Master of Computer Science

S.C. Rangoda

University of Colombo School of Computing

2019

Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name	: S.C. Rangoda
Registration Number	:2016/MCS/091
Index Number	:16440912

____

Signature: Date:

This is to certify that this thesis is based on the work of

Mr. S.C Rangoda

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name : Dr. Malik Silva

____

Signature: Date:

Contents

Table of Figures

Abbreviations

IoT	Internet of Things
CPU	Central Processing Unit
API	Application Programming Interface
http	Hyper Text Transfer Protocol
RPC	Remote Procedure Call
JVM	Java Virtual Machine
GIL	Global Interpreter Lock
OS	Operating System
REST	Representational State Transfer

Abstract

Communication between software services become an essential requirement in modern software industry. With the improvement of internet services, connecting existing software services with each other and provide unlimited services via the internet to the end users become a crucial requirement for modern businesses. Web services have been introduced as the standard communication medium between software services and micro services-based architecture which enrich the usage of web service. IoT services and increasement of smart devices create high demand for web services. Due to those reasons, performance of web services become very critical and crucial to the existing and new software services.

Web service performance can improve by distributing multiple instances among multiple nodes where the approach is not always cost effective. Other possible approach would be parallelly executes the web service requests. Multi-core architecture has been introduced to execute computer programs parallelly in processor level. To get optimum performance from this multi-core processors, computer programs should be able to execute parallelly at processor cores level.

Multi-threaded and Multi-core based parallel execution techniques were mainly evaluated by implementing prototype REST APIs for matrix multiplication. REST API performance impact was evaluated against the processor affinity.

When processing complexity increases in serial processing REST API, their performance (measured by throughput) decreases rapidly. When complexity increases for REST end points where they have parallelly execution methods, their performance also decreases but decrease amount is smaller by comparing to the serial processing. There is notable performance gain for multi-process parallelly executions over the multi-threaded parallel execution. Best throughput was achieved, when multi-core process binds to specific core of the processor (applying processor affinity).

1 Introduction

Communication efficiency of two parties has been improved in 21 st century than any other industry or technology. Today people are considering device to device direct communication without human involving. Those devices might be vehicles, home equipment, sensors etc. This much of communication efficiency has achieved with the improvement of internet and hardware devices, especially mobile phones. And people are identified that, to get maximum use of software product, it should be available as a service, where any other system can interact over the internet. That enables less processing on end-point devices which helps to build light weight user friendly mobile end devices. To interact between services, each software service should have common medium to communicate irrespective of the technology it was build. Web service has introduced as a standard way to communicate between software services over the internet.

With the increasement of mobile devices and IoT devices, the demand for web services increased exponentially. So that performance of critical web services will directly influence to expand the other services capabilities. For instance, modern society prefers mobile banking and online purchasing methods than any other payment medium, which makes payment gateway APIs are more critical. So that, performance of payment gateway API services makes significant influence on the growth of online payment transactions.

One of the most common approach to improve web service performance is, distribute web service into multiple instances. But this approach is not economical for all the conditions. Performance of the web service can obtain, improving the execution efficiency of background algorithm as well. Impact of this execution efficiency in web service will be evaluated in this study.

Generally, processor efficiency which is measured by clock speed, can improve the program execution time. In single core processor, execution of programs happens sequentially. When there are multiple programs to execute at same time, there is a latency added to the execution due to context switching. Since clock speed of processors achieved to their theoretical and physical limits, vendors invented new processor which is having multiple processing cores. When there are multiple cores available in the processors, best performance can be achieved by executing programs parallelly in each core of the processor.

In hardware level, there are two types of processors, multi-processors and multi-core processors. In Multiprocessor systems contain multiple CPUs that are not on the same chip whereas Multicore processors contain any number of multiple CPUs on a single chip. Multiprocessor system has a divided cache with long-interconnects and the multicore processors share the cache with short interconnects (refer Figure 1).

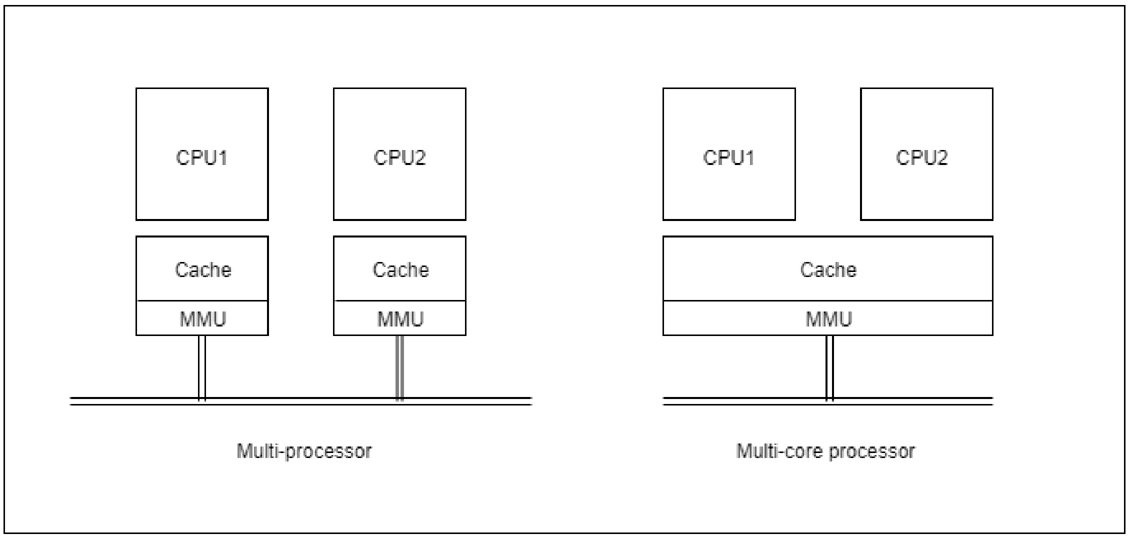


Figure 1: Parallelism Techniques

Multicore processor can speed up processor intensive operations on large data sets by segmenting those data sets. Each data segment processed by a core in the processor, which enables parallelly process multiple segments of the data set.

1.1 Motivation

There are many web services developed and maintained by different organizations in various domains. Recently, demand for those services has increased very rapidly. Some of those services are not designed and developed to serve this much of user demand. Distributing the service across many instances is the common approach to solve this problem. But that might not be fully utilized resources like CPU, unless those programs are implemented using parallel computing techniques. Increasing hardware capacity without utilizing their features is a wastage of investment on those resources. Most of the latest CPUs are having multiple cores, which supports parallel execution in processor level. Identifying the impact on web service performance by applying parallel execution techniques will be helpful for decision makers to enrich their solutions by utilizing optimum resources. And the motivation behind this study also, evaluate possible approaches and motivate the community to follow these approaches whenever it is applicable.

1.2 Objective

The objective of this project is to find a novel approach to analyze impact of applying parallel computing techniques on web service performance optimization. The output of this research is to produce an effective analysis of performance impact of different approaches of parallel computing in web service implementation. Furthermore, it identifies the relationships between web service efficiency and parallel computing techniques.

1.3 Scope

Parallelism can be implemented many ways. This study focuses on applying multi-core parallel programming methods in web service domain. CPU intensive program (like matrix multiplication) will be implemented using those identified methods. Study will focus on identifying efficient way of integrating parallel programs into client requests (http request) in application server. Other required implementation of web services like authentication, security etc. will not take into the evaluation.

1.4 Contribution

This research produces efficient approaches of parallel processing which can be applied for CPU intensive executions in web services. This would be the main contribution to computer science domain which is expected to observe, by conducting this research study.

Existing libraries and frameworks are used to implement prototype designs. For instance, framework like spring boot can be used to implement web services in java. Following industry standard frameworks and libraries, make research outcome more valuable for the community in enterprise application development. Hence, this research contributes not only improve computer science methodologies but also community in software application development as well.

1.5 Thesis Outline

Chapter 2 describes background and literature review about current parallel computing techniques which are used in the field of web service. And current studies in multi-core programming and their implementations in programming languages like java and python. Literature review further carried out on, standard methods to evaluate web service performance.

Chapter 3 describes prototype solutions for analyze research problem. In addition to that, proposed approach for the implementing parallelism for web services is discussed further throughout the chapter.

Next, chapter 4 implementation discusses about how the implementation is done for the design described in chapter 3 and further describes the tools, techniques and algorithms used for implementing the proposed approach.

Chapter 5 describes about the testing and evaluation phase how the testing and evaluation is done for the proposed system in above chapters. And it is the process by which a system or components are compared against requirements and specifications through testing. The results are evaluated to assess progress of design, performance, supportability, etc.

Finally, in chapter 6, conclusion and future work section conclude the dissertation discussing about how future work should continue.

2 Literature Review

Literature review was conducted to study and observe the existing research in problem domain, parallelism techniques applied in web services, multi-threaded programming concepts and their implementations, web service deployment impact on performance and standard methods to benchmark performance of web services.

2.1 Problem Domain

It is critical to have efficient web services to improve overall performance of the applications. As describe by the G M Tere1 et al [1], there are six important techniques which can be applied to improve performance of Restful web services. Fast manipulation of strings, streaming large representations, compressing SOAP response, partial representation, using caching techniques and using conditional methods are the possible techniques which are applied to improve performance of Restful web services.

REST and SOAP based web services are platform and programming language independent, and both architectures have loosely coupled client and server [3]. RESTful web services are web applications build upon the representational state transfer architecture. They expose resources through web URIs, and use the four main HTTP methods to create, retrieve, update and delete resources [10]. REST technology is generally known as more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth and has very less complexity when it compares with SOAP.

Service oriented architecture became most widely used standard paradigm to develop business applications over the internet, like Business to Business (B2B) and Business to Consumer (B2C) applications. These applications mainly based on web service interactions which represent objects, whose methods can be called through internet. This required to serialize and deserialize objects or data and it is a costly process in term of performance of the web service. By transparently parallelizing web service calls on multicore systems using OpenMP, web service applications can be efficiently parallelized on multicore systems [4].

2.2 Parallelism Techniques in Web Services

According to the S´ebastien Salva et al [4] there are two main approaches to parallelize web service request in multicore systems, using task pool paradigm and pipeline paradigm. Having a task pool to parallelize the web service requests is a basic method where one task represent one web service call. In this solution, if n threads are available, n calls can be easily done in parallel, assuming n cores are available.

In pipeline paradigm, web service call steps are executed successively and completely independent. Research has identified four stages of web service request, which can execute parallelly in pipelined method.

- Serialization stage (S)
- Web service call stage (C)
- Deserialization stage (D)
- Stage for the persistency (P)

This approach requires at least four cores in the processer to avoid thread interleaving and achieves a better use of resources. After analyzing web service response time over the threads, they have shown, pipeline approach gives much better results over thread pool, specially when higher number of threads are used (refer Figure 2).

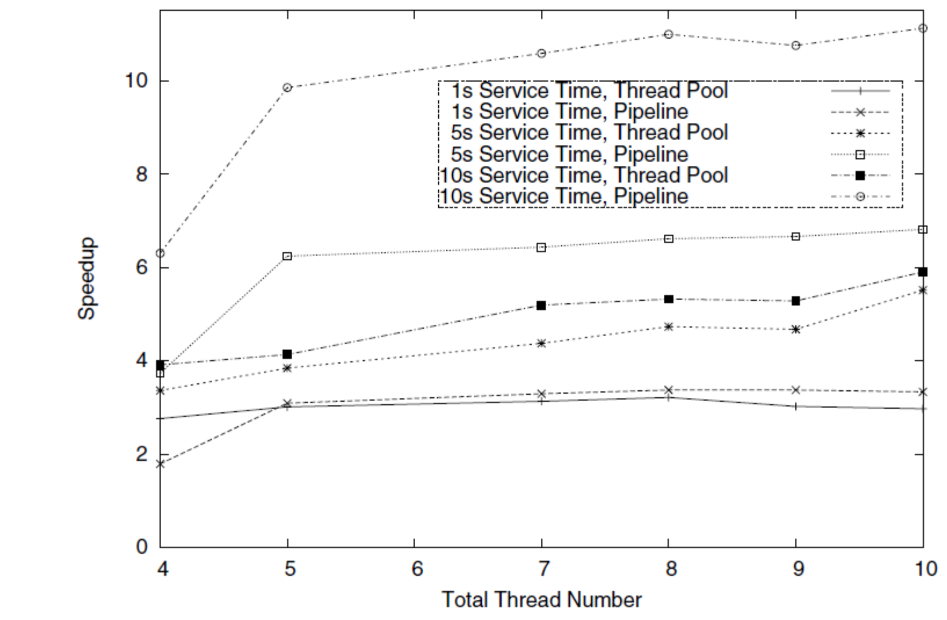


Fig. 7. Speedup for Various Waiting Times and Solutions

Figure 2: Latency Evaluation [4]

Pipeline parallelism organizes a parallel program as a linear sequence of stages and each stage processed elements of a data stream passed to the next stage. Pipeline parallelism is a well-known and best suited parallel programming technique for streaming applications even though it can apply many use cases [15]. In “on-the-fly” pipeline parallelism, structure of the pipeline emerges as the program executes rather than being specified before. Which allows programmer to specify a pipeline, where the structure is determined during the pipeline’s execution. This model can be applied in web service implementation as well.

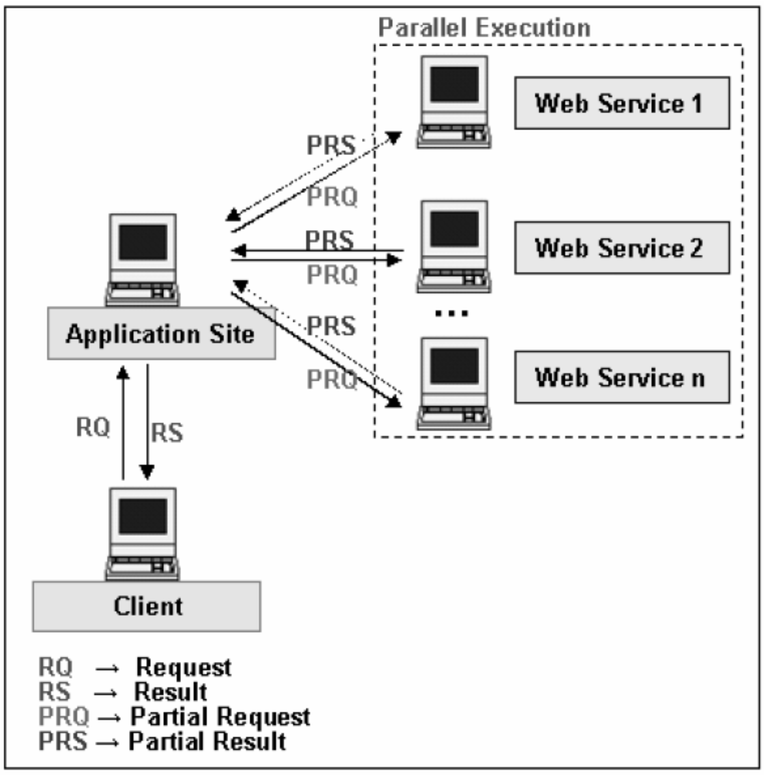


Figure 3: The Proposed Model for Parallel Execution [11]

Parallel processing can obtain by dividing program into multiple web services and client request will be served by referring many back-end webservices (refer Figure 3) by the application server [11]. This approach more towards to the distributed program. Dividing program into multiple web services added more communication latency and when one service depends on another, it cannot process parallelly. When individual web service does not able to process independently without depend on any other service, that will not be able to give efficient execution to the web application which invoke by the client. Building many web services requires many computational resources and applying parallel programming concepts to enhance execution efficiency of individual programs do not evaluated in the study.

2.3 Multi-threaded programming concepts & frameworks

Performance of web services has improved by using programming languages which has fully supported event-driven architecture [5]. Node.JS is one such language which has hidden event-loop support behind the convenient programming interface [6] which allows the developer to treat event-driven programming as a set of callback function invocations, taking advantage of the functional nature of the language. Main drawback of this model is, whenever it had to depend on some legacy back-end web service which has high latency, whole process needs to be wait due to the single-threaded, sequential event loop architecture. There is another major limitation in this event driven framework, which is impossibility of sharing a common memory space among processes. Programming model of Node. Scala has introduced to address above limitations which has features of a parallel programming model based on asynchronous call-back invocations. This framework enables blocking methods to invocate without blocking the service and concurrent requests running on different threads while safely sharing the state.

Java is a widely used and more popular programming language which has numerous high-performance implementations [8]. Java has two ways of parallel execution mechanisms, Remote Method Invocation and java threading API. From java 1.5 onwards the concurrent package provides more support for multithreaded programming.

Spring batch is a lightweight comprehensive batch processing framework for java-based enterprise application development [12]. When batch processing applies to online user interactions such as web service, data should be accessible simultaneously and it should be able to process within few seconds. To achieve this, spring batch uses parallel processing which is implemented using additional threads to process in parallel. Spring batch uses two modes of parallel processing, single process, multi-threaded and multi-process. There are four main categories of parallelism spring batch supported [12].

- Multi-threaded Step (single process)
- Parallel Steps (single process)
- Remote Chunking of Step (multi process)
- Partitioning a Step (single or multi process)

There are suitable problems which can be applied in each method. And integrating this parallel batch processing methods might not be suitable for all web service requirements unless web service requires to query large back-end database or processing through many files. In such scenarios, this will give performance improvement to the web service.

OpenMP is an industry standard for shared memory parallel programming [9]. Java has its own native threads model for shared memory programming, but it has some drawbacks which impact to the performance when compare with the OpenMP standard. To get maximum efficiency from shared memory multi-core architecture, it is required to execute exactly one thread per processor CPU and to keep these threads running during the whole lifetime of the parallel program. To achieve this, it is required to have runtime library to dispatch tasks to threads and provide efficient synchronization between the threads. This has not supported or implemented by java native thread model [9].

Multithreading and multiprocessing are two main parallel models used by programming languages to give ability of parallelly execute programs [18]. The key difference between multiprocessing and multithreading is that multiprocessing allows a system to have more than two CPUs added to the system whereas multithreading lets a process generate multiple threads to increase the computing speed of a system. Multiprocessing system executes multiple processes simultaneously whereas, the multithreading system let execute multiple threads of a process simultaneously [19]. Based on these two approaches programming languages built their parallel execution methods. For instance, Java supports multithreading approach, whereas python supports multiprocessing approach.

Multithreading programs in python is little bit tricky since python interpreter has a thread-safe mechanism of global interpreter lock. An interpreter that uses GIL always allows exactly one thread to execute at a time, even it runs on a multi-core processor. Due to this behavior python is restricted to use single OS thread, which cannot make use of the multiple cores and processors available on modern hardware. Having multiple threads to execute a program does not give a performance gain, since eventually all threads are executed as a serial process in OS level. As an alternative solution for parallel processing, python has *multiprocessing* library which provides simple API for the use of parallelism based on processes.

2.4 Impact of application deployment in web service deployment

Webservice response time varies according to their deployment. It can gain some potential latency benefits across three popular cloud infrastructure services, Amazon EC2, Google Compute Engine (GCE), and Microsoft Azure [2]. Study has discovered, the RTT (Round Trip Time) of webservice request can reduce up to 20% when a webservice is deployed across the three cloud services compared to deployments on one of these cloud services. They have observed three significant observations for this latency benefits. When webservices shift from single-cloud to multi-cloud deployments, their routing inefficiencies which exist user and their nearby cloud data center is vary. When web service is hosted multiple cloud services, whatever datacenter which is closed to the user can serve to the request quickly. It also discovered that, when users in several locations will continue to incur RTTs greater than 100ms even when webservices span three large-scale cloud services.

2.5 Benchmark performance of a web service

Over the past decades many performance modeling formalisms and prediction techniques for software architectures have been developed but there is lacking a performance model to predict the performance of a software system. Measuring performance of a software, normally requires extensive experience and complex time-consuming manual steps [3]. Research has proposed a query language, Descartes Query Language (DQL) which is a language to express the demanded performance metrics for prediction as well as the goals and constraints in the specific prediction scenario. It reduces the manual effort which heavily required in software performance testing.

Apache Jmeter is an open source java-based web application load testing tool [13]. It can use to test performance on static and dynamic web resources. As shown in Figure 4, Jmeter can simulate number of users for web application. It has ability to load and performance test many different applications including SOAP or REST webservices

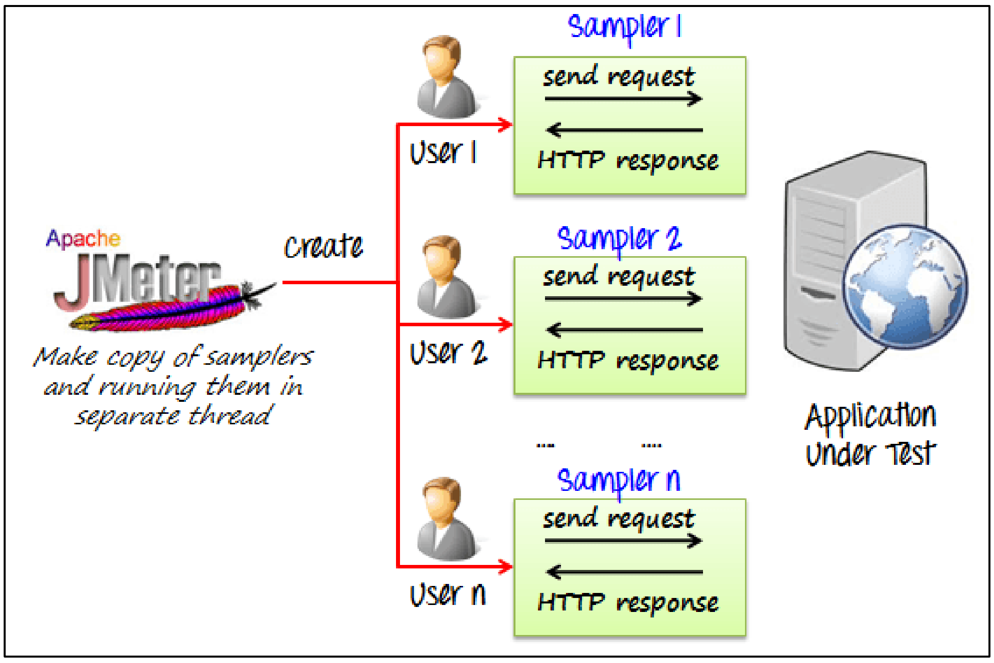


Figure 4: JMeter Performance Test Design [14]

3 Analysis

Importance of high performed web services and revolutionized web service architecture from RPC to micro services has discussed in initial part of the literature review. Many approaches are taken into improve web service performance. Most of them are focused on distributed technologies. Applying parallel processing techniques are limited in web service domain even though there are many techniques and implementations are available. The use of pipeline parallelism in core web service components like serialization, deserialization, http request call etc. are evaluated in previous studies. But there are many frameworks and tools are available for web service implementation, which has those capabilities build-in. The impact on applying parallelism in business logic of web service has not thoroughly evaluated.

With the technology improvement of computer hardware, venders are invented multicore processors which enhance parallel executions in hardware level as well. It is a kind of extended architecture of hyperthreaded into hardware level. The OpenMP API uses this shared memory multicore processors for executing programs parallelly. All programming languages does not fully support to this OpenMP technique, and java has its own implementation for OpenMP like parallelism.

Parallelism can be done in two ways, dividing whole set of large data set into pieces and process each piece parallelly. In this scenario all instances are doing same task. Other way of doing is, divide whole process into multiple tasks and each task perform parallelly, called as pipeline parallelism. Both approaches are possible to use with web service and their suitability will decide depends on the processing task.

Since this study it is mainly focused on how parallelism can improve performance of web service. One of those parallel approach should evaluate with distributing multiple instances of web service.

4 Design

To evaluate better parallelism approach for web service, it is required to build prototype implementations and measure their performance. Since each implementation is going to compare each other, it is required to implement similar kind of restful API in each approach.

4.1 Introduction

To analyze the impact of pipeline parallelism and data parallelism of web service in multi-core processor, both approaches are implemented as prototypes and their response times, was evaluated. As a use case for this comparison, the matrix multiplication was used. The prototype web service gives result of multiplying two matrixes with any number of rows. Limiting fix number of matrixes will reduce the complexity of the web service but allowing any number of rows provide enough work load to compare the two approaches.

4.2 Prototype Design

Following steps are executed whenever client request comes to the designed API endpoint. Generally, request goes through validation process, multiplication process and response generation. Logical flow of web service, which is in Figure 5, elaborates logical view of the API.

Prototype has implemented using java and python languages. Both languages have built-in libraries for parallel processing, but they have used different techniques for their parallel processing. Implementing same Rest API end-point in both languages, enables to evaluate their parallel processing efficiency.

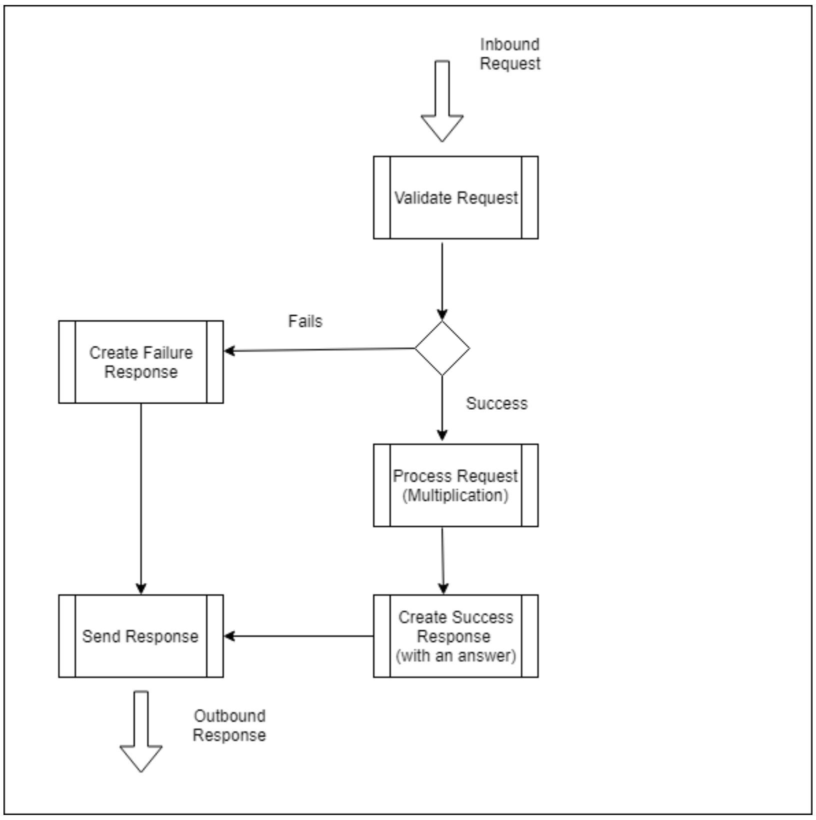


Figure 5: Web Service Request Flow Chart

Main sub tasks of each process can be listed as follows.

- Request Validation
  There are two main validations should be taken in place prior to begin any processing.
  - Validate json request body
  - Validate two matrixes

  Incoming request should have valid json request body as described by the RestFul web service API. To multiply two matrixes, the number of columns of the 1st matrix must be equal to number of rows of the 2nd matrix. Requests which satisfy both conditions are taken into processing otherwise those requests are marked as failures.

- Request Processing
  Multiplying two matrixes will be performed by this task. Implementation of this task depends on the parallelism method. To apply data parallelism, this task should be implemented as a sequential program. To apply pipeline parallelism, this should be divided into multiple sub processes.
  Applying generic algorithm in combinational problem like tuple multiplication can increase the efficiency of execution [16]. The generic algorithm can derive pipelines for matrix multiplication and shortest paths computation. The algorithm which describes in [16] for *n x n* matrix multiplication has complexity of O (n^3).

- Success / Failure Response
  Body of Response message will create based on the either result of request validation or request processing steps.

- Send Response

  Adds other required parameters like, http code, headers etc. to the message body and generated valid json response message to the client request.

Data parallelism can be applied for multiplication process of the matrix. Value of each element in result matrix, is independent from the other elements. So that production of row and column can calculate parallelly. To observe the difference, sequential processing algorithm also implemented as shown in Figure 6.

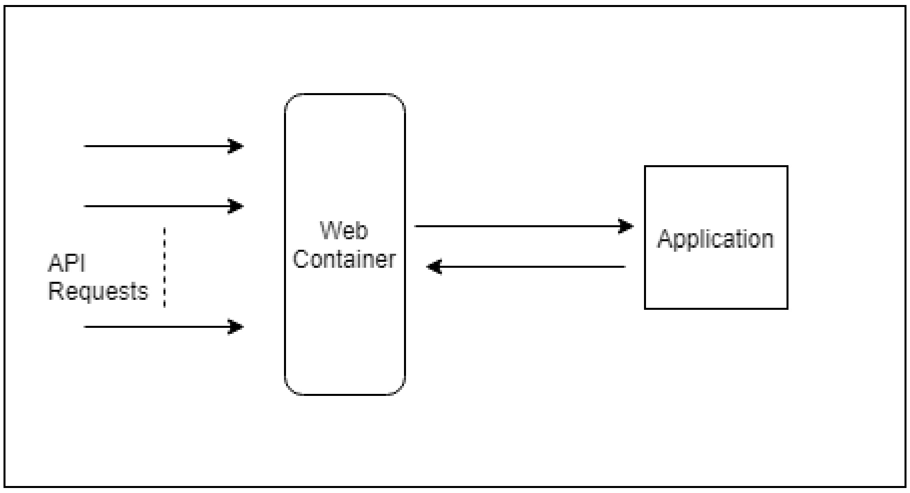


Figure 6: Design of Single Threaded Implementation

For concurrent processing, java provides multi-threaded execution. When number of tasks are not static and very large, using java threads for concurrent execution might be very risky approach. If there are defined number of threads, whenever the number of concurrent execution tasks go beyond that, it will fail. If it allows to create any number of threads, then there is a possibility for JVM to go out of memory. Java provides thread pool, which is much better approach for this scenario.

As shown in Figure 7, java thread pool has proposed to implement parallel execution of two matrixes. The number of threads in the pool is equal to number of rows in matrix one, but it will be fixed when row number goes beyond to 100. This will eliminate, JVM is going out of memory when ever the request comes with large matrix.

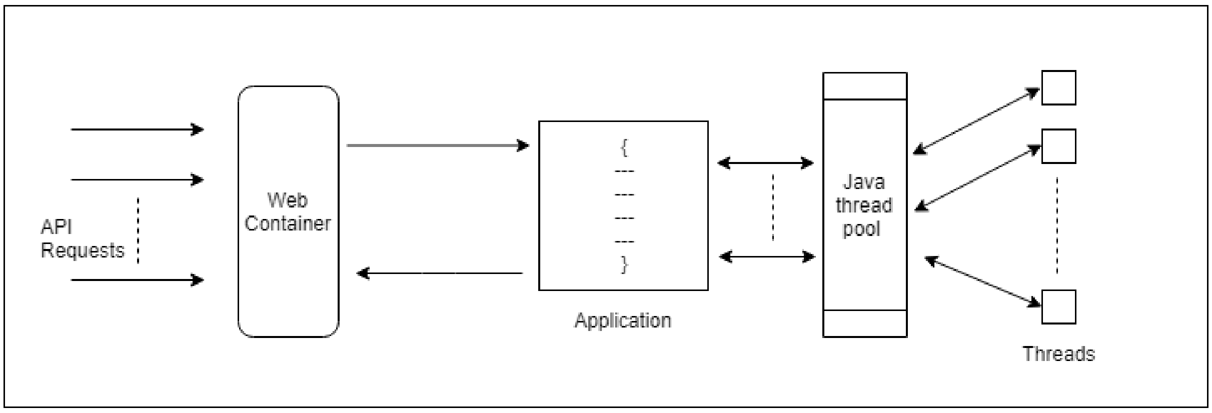


Figure 7: Thread Pool based Design

Even though programmatically parallelize the program into multiple threads, those threads are waited for processor scheduler to pick them for execution in a core of the processor. Schedular is an operating system owns program, who decides which process is going to execute and how long will it allow for execution at the processor level. This is totally beyond the control of JVM or any other program level.

The impact of this scheduler process can be evaluated when there is a dedicated core available for the program execution. Figure 8 shows the binding process of the program to processor core by using java OpenHFT [17] library.

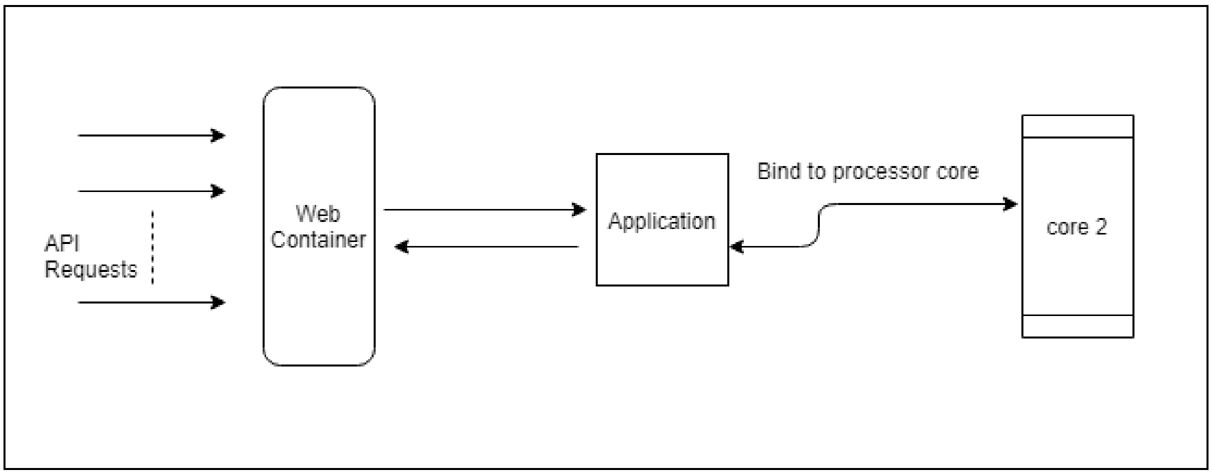


Figure 8: Design of Process Binding Implementation

4.3 Python Based Multiprocessing Implementation

Python has multiprocessing module which includes a very simple and intuitive API for dividing work between multiple processes [20]. Flask is a lightweight but very powerful python web framework which can use to build RESTful web service in minimum steps. So that python multiprocessing module and flask web framework has selected to implement the prototype implementation.

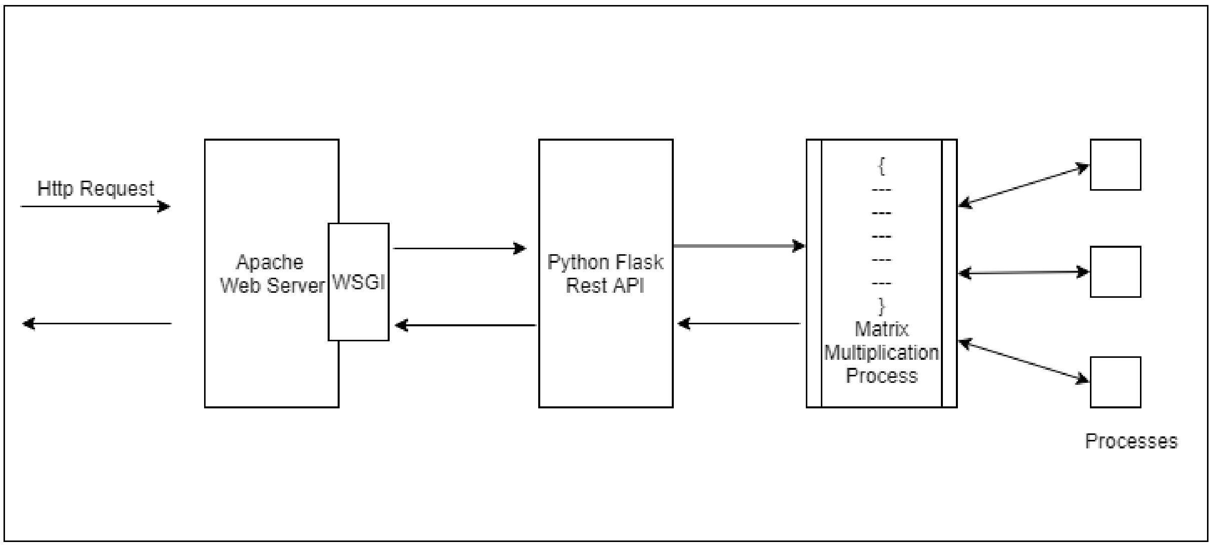


Figure 9: Python Rest API design with multiprocessing based parallel implementation

As shown in Figure 9, apache web server is deployed in front of flask rest API service since it is recommended deployment to get maximum performance from flask web framework.

Accepted API definitions are listed below.

Accepted json request body is shown in Figure 10.

```
{
   "requestId":2238,
   "matrixOne":{
      "columnCount":3,
      "rows":[
         {
            "value":"1,2,3"
         },
         {
            "value":"4,5,6"
         }
      ]
   },
   "matrixTwo":{
      "columnCount":2,
      "rows":[
         {
            "value":"7,8"
         },
         {
            "value":"9,10"
         },
         {
            "value":"11,12"
         }
      ]
   }
}
```

Figure 10: Inbound Json Request

RequestId should be unique for each request and response message will map with this id. Two matrixes attach as separate objects and each row of the matrix should be attach in the order of matrix.

Success response (in Figure 11) and failure response (in Figure 12) are listed below.

```
{
   "requestId":"2238",
   "rows":[
      [
         58,
         64
      ],
      [
         139,
         154
      ]
   ]
}
```

Figure 11 Outbound Json Response – success

Failure Response

```
{
  "requestId": 2243,
  "errorCode": "001",
  "errorMessage": "Invalid number of columns and rows for
multiplication"
}
```

Figure 12 Outbound Json Response – failure

4.4 Prototype Source Code

Prototype implementation is available in github, which is an open source repository. It delivers as open source project, where community can contribute near future.

- Url: https://github.com/sashikaR/mcs

Project build steps and other required third-party software installations are described in README guide of the project.

5 Evaluation

Performance of web service measures using the TPS (transactions per second) or throughput. Throughput gives number of client requests which can process and success within given time period. To get higher value for this threshold, it is essential to have less response time for client requests. So that response time (or latency) takes as the main measurement to compare the performance of web services.

Using load testing tool like Apache Jmeter, which is well recognized, industry standard performance testing tool will standardize client traffic generation process. Jmeter gives throughput and average latency with reliable and accurate figures. Analyzing statistics which gives by Jmeter report, the impact of parallelizing matrix multiplication over sequential execution process can be evaluated.

In this research also, response time and throughput of each web service implementation, takes as the measurements to compare their performance. Utilization of resources like CPU, Memory disk I/O etc. also collect as the test artifacts. These statistics are important to evaluate the cost of each web service to serve higher number of client requests.

As a first phase of the evaluation process, system accuracy will be tested. When system provides accurate results consistently across all implementations, their performance will be tested, as the second phase of the evaluation process.

5.1 Functional Test

Since prototype implementations are not having complex algorithms, unit tests will cover the functional test of the system. Junit has used to automate unit tests and those tests are executed when build the system using apache maven build tool. So that accuracy of each implementation will be tested prior to their performance evaluation.

5.2 Performance Test

In order to evaluate the impact on restful web services by applying multi-core parallel execution techniques, response time of the web service request will be analyzed. Response time depends on execution efficiency of the program. It can evaluate either hitting large number of requests or increasing the processing complexity. Processing complexity will achieve by increasing the number of rows and columns of the two matrixes which are going to multiply. Following performance test scenarios are identified, which can use to evaluate efficiency of each parallel processing approach of restful web services.

5.2.1 Performance Test Scenarios

Objective of the test T-A01 to T-A03 is, evaluate web service performance of each approach when number of client requests (concurrent users) are increased. Same behavior evaluates when processing complexity of the matrix getting increase as well. Analyzing results of following test scenarios (in Table 1), it can be decided how each approach behaves when they are getting higher number of requests with different processing complexities.

Table 1: Performance Test Scenarios

Test ID	Rest API End Point	Concurrent Users	Matrix Complexity
T-A01	S / M / AF	10	Low
T-A02	S / M / AF	30	Low
T-A03	S / M / AF	60	Low
T-B01	S / M / AF	10	Medium
T-B02	S / M / AF	30	Medium
T-B03	S / M / AF	60	Medium
T-C01	S / M / AF	10	High
T-C02	S / M / AF	30	High
T-C03	S / M / AF	60	High
T-D01	S	10	Low
T-D02	S	10	Medium
T-D03	S	10	High

Index key table (refer Table 2) is shown below.

Table 2: Index

Key	Value
S	Rest API Implementation 1, without parallel execution model
M	Rest API Implementation 2, with java thread pool-based execution model
AF	Rest API Implementation 3, with binding execution process to processor core
Low Complexity	Matrixes with 30 X 20 and 30 X 20

Medium Complexity	Matrixes with 1000 X 50 and 1000 X 20
High Complexity	Matrixes with 2000 X 50 and 2000 X 20

5.2.2 Data Collection

To provide detail reliable analysis, following measurements are collected by the time tests are executed.

- Average response time
- Throughput
- CPU utilization
- Memory usage

Jvm monitoring tools like jconsole, jmc, hawtios etc can be used to verify thread blocks and Jvm memory usage in each implementation.

5.2.3 Tools

There are many tools available for software performance testing like Apache Jmeter, LoadRunner, The Grinder etc. Out of those tools, apache Jmeter is used as the performance test tool, because it is an open source tool which has built-in support for Rest API load testing. It gives statistics like average latency, throughput etc with offline html reports including graphical representations.

To measure CPU, Memory etc, linux SAR (System Activity Report), command line tool will be used. SAR is a system monitor command, used to report on various system loads, including CPU activity, memory/paging, interrupts, device load etc.

5.2.4 Design of Deployment

All web services are deployed in one web container, which is provided by spring boot framework and multiple rest web services are differentiated by their resource url. Figure 13 illustrates how each REST Web Service deployed and client requests are going to generate.

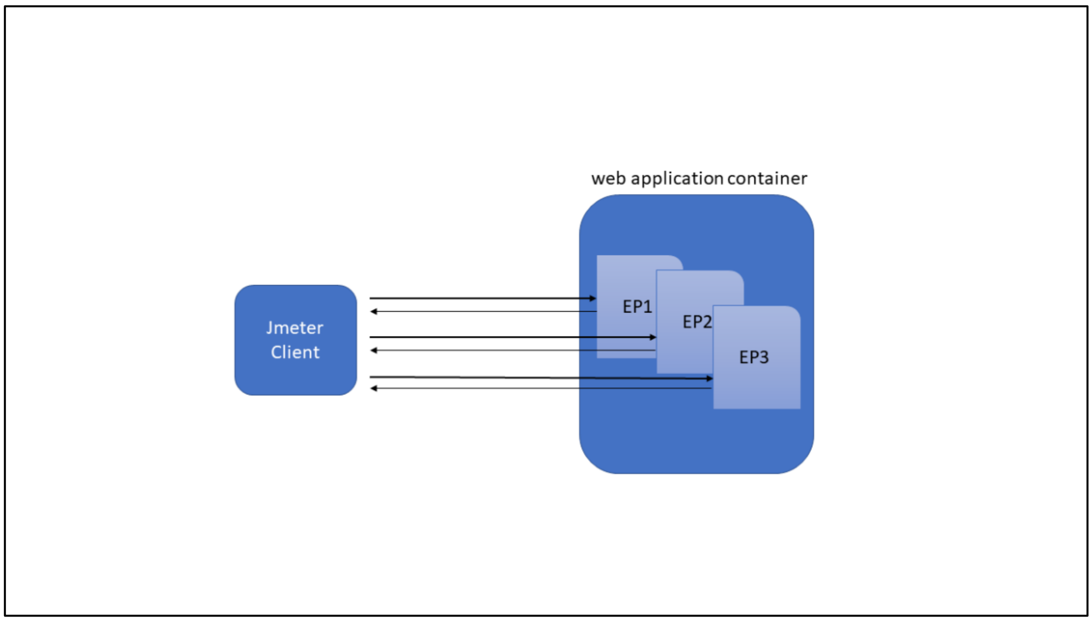


Figure 13: Performance of java thread-pool, when no. of users increased

6 Results and Observations

Results are observed separately for Java based implementations and Python based implementations. Java based matrix multiplication REST API services are tested and compared by increasing number of concurrent users and increasing the complexity of matrix multiplication. And those results are compared with the results which is taken from java affinity-based approach.

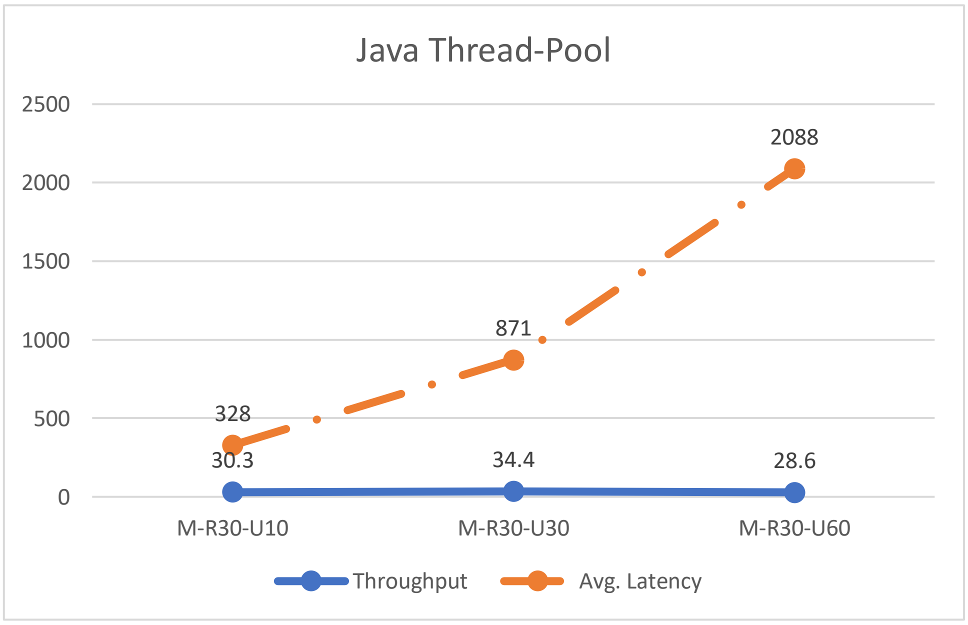


Figure 14: Performance of java thread-pool implementation, when no. of users increased.

As shown in Figure 14, when number of concurrent users are increased (from 10 – 30 to 60), there is no significant improvement in throughput. But average latency has increased significantly. Even though internal thread-pool has 30 threads to serve for each request to process, spring boot uses singleton pattern to serve http request. Which can be made many requests were queued inside java web container.

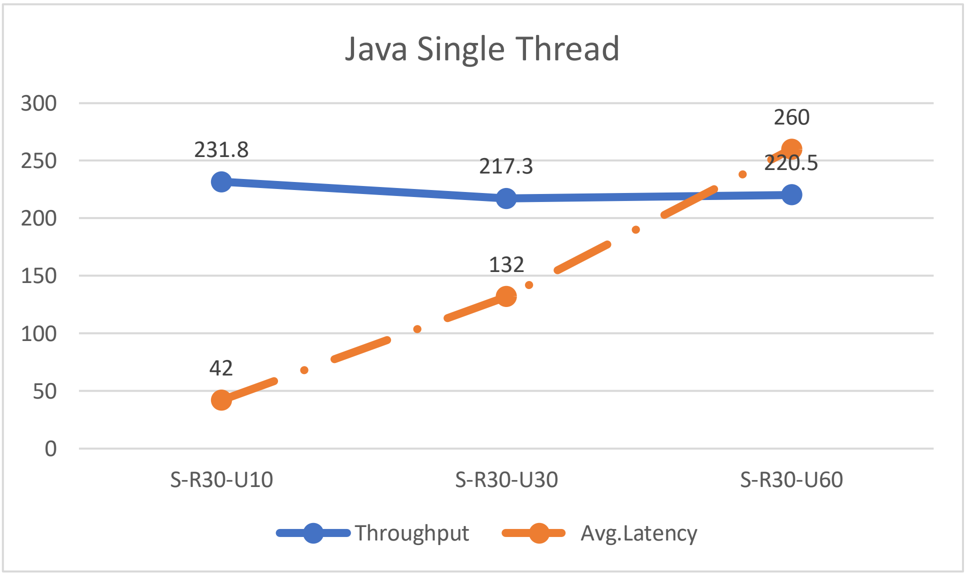


Figure 15: Performance of single threaded implementation, when no. of users increased

When compare with other approaches, single threaded sequential execution gives higher throughput in this complexity where each matrix of the request is having 30 rows (refer Figure 15). When number of concurrent users are increased, throughput is decreased by a small number. Which shows that, some http requests are delayed at web application server. That can be happened when each request processing time is higher than the incoming requests rate.

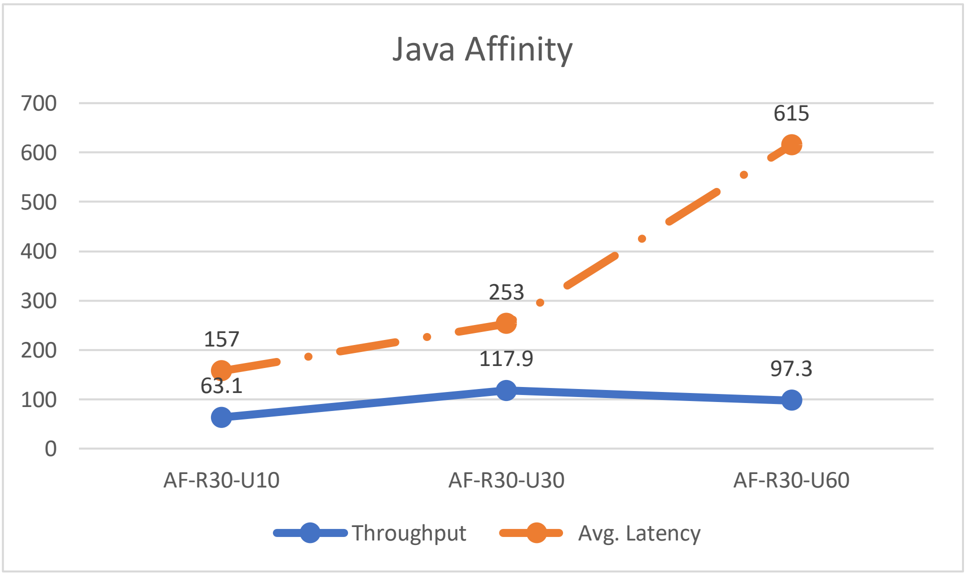


Figure 16: Performance of process binding to specific core, when no. of users increased

Throughput of java affinity is higher than the thread-pool implementation (refer Figure 16). Even though it is less than single threaded implementation, throughput has increased, when number of concurrent users are increased. There is some overhead added to the process when binding it to the core of the processor. This is not directly managed by the JVM. JVM access operating system libraries via java native threads. Execution time for these processes also added to the latency of client request.

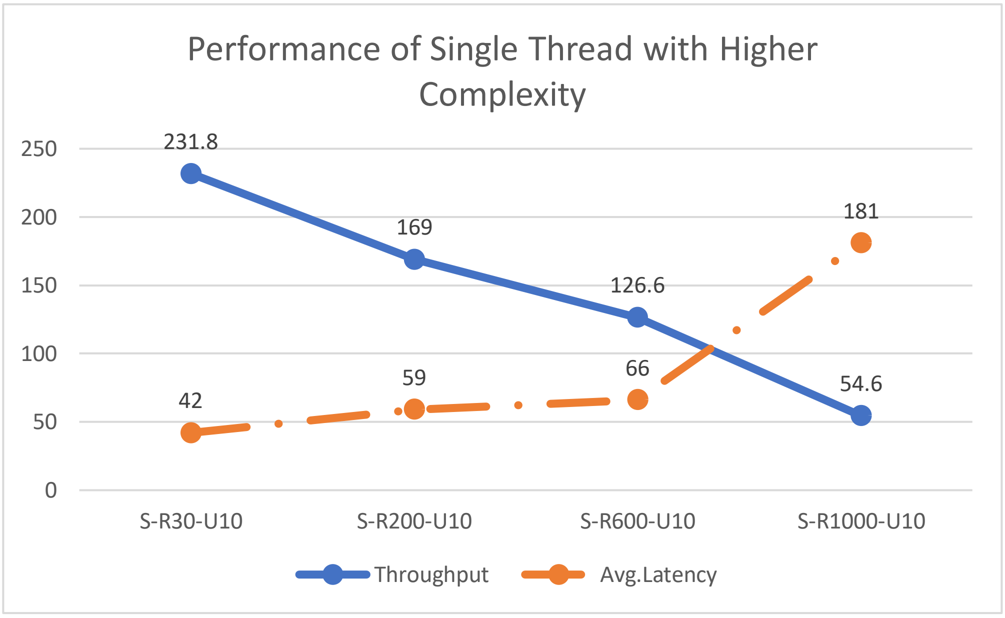


Figure 17: Performance of single threaded implementation, when processing complexity increased

As shown in Figure 17, throughput of single threaded implementation has decreased when the complexity of the matrix multiplication increased (number of rows from 30, 200,600,1000). This behavior is proved that, whenever the task requires higher CPU processing power, performance of sequential program decrease.

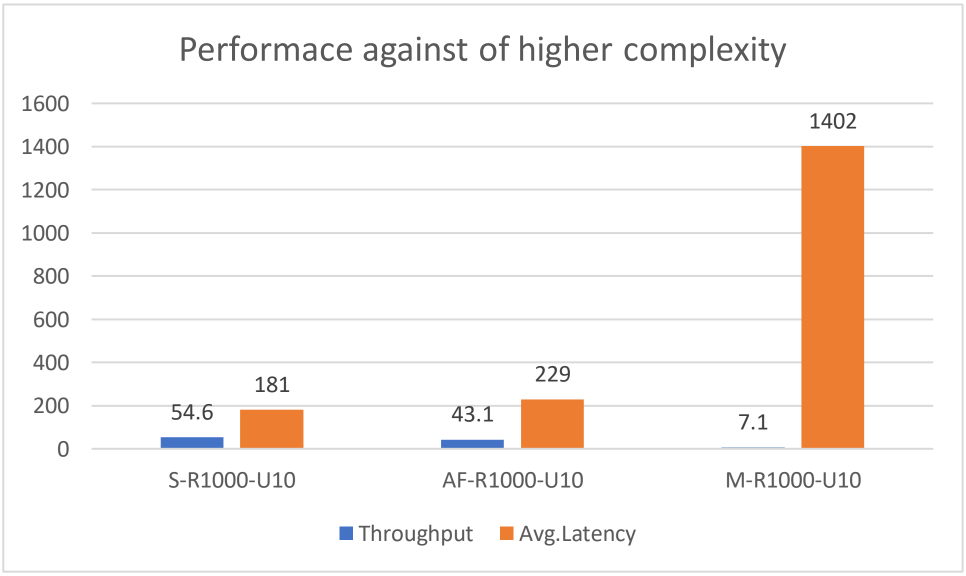


Figure 18: Performance comparison of each implementation against higher complexity.

Figure 18 shows how each implementation behave when matrix has 1000 rows. When complexity goes higher, throughput gap between single thread and java affinity goes lower. Since JVM cannot afford thread pool with equal number of rows in matrix, it has given 100 thread pool size as the maximum value.

6.1 Analysis of Python Based Implementation

Python based matrix multiplication REST API services are also tested and compared by increasing number of concurrent users and increasing the complexity of matrix multiplication. Python uses multi processes for their parallel executions instead of multi-threaded approach which is used by Java. Python multi-process has applied in two ways to evaluate web service performance.

1. Parallelly processed tuple multiplication (column and row of the matrix) in each web service request.
2. Parallelly processed incoming HTTP request, where matrix multiplication process sequentially in each request.

Apart from above approaches, linux based affinity (process binding to specific processor core) also tested with python-based implementation.

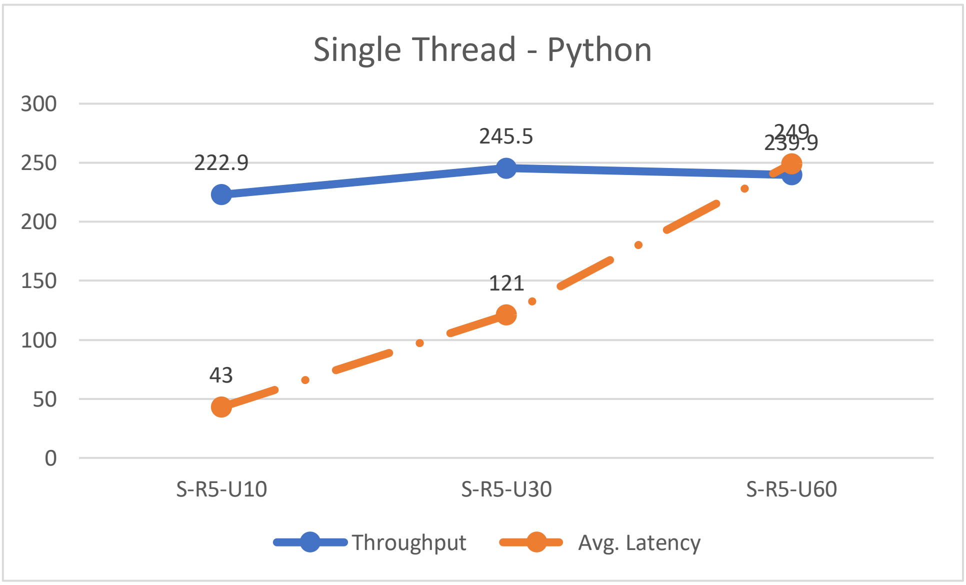


Figure 19: Single Threaded Implementation - Python Module

As shown in Figure 19, single threaded implementation's throughput has increased when number of user threads increased from 10 to 60. Since response time also increased, there is a maximum value for this API's throughput. After that it will decrease even though number of users increased. When compare with java-based implementation, there is no significant difference in term of throughput of the web service.

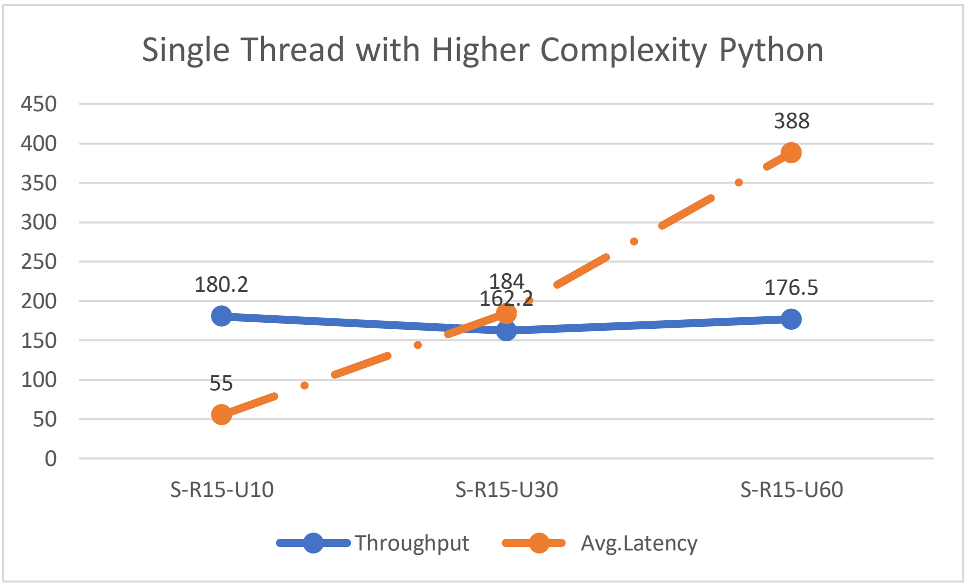


Figure 20: Performance of single thread implementation, when increasing the complexity - Python Module

When multiplication complexity increases, throughput of the API goes down even number of user threads are increased from 10,30 to 60. In low complexity (Figure 20), throughput has increased when number of users are increased. So that we can identify that how much of impact can be happened whenever the required processing power increased.

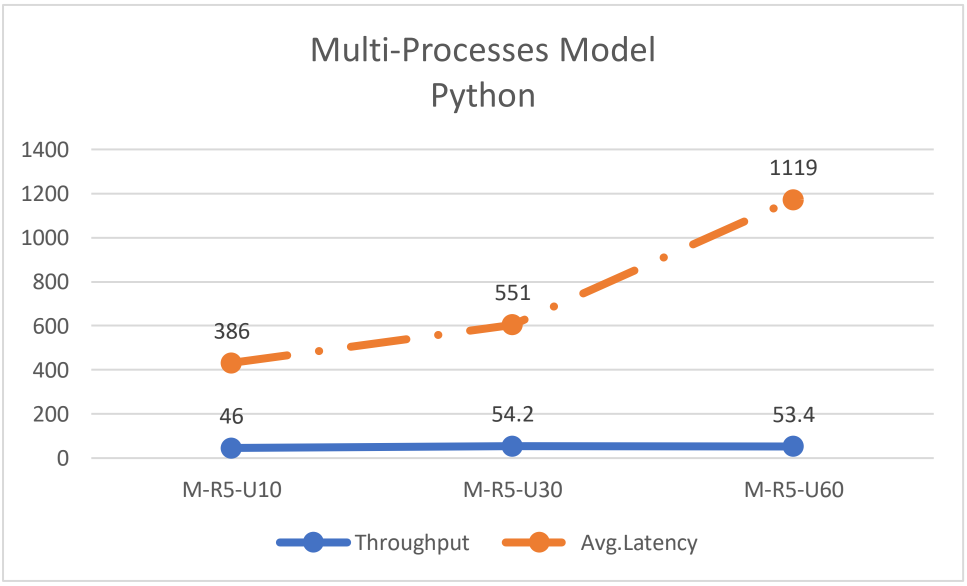


Figure 21: Parallel processing implementation with multi-process, Python module

When compare with same result of single thread implementation, python multi-process library gives less performance when implementation has less processing requirement. That might be mainly due to the parallel processing overhead (not being truly embarrassingly parallel), which is added additionally to the serial matrix multiplication. The throughput of API also does not increase significantly with the increasement of number of user threads (refer Figure 21).

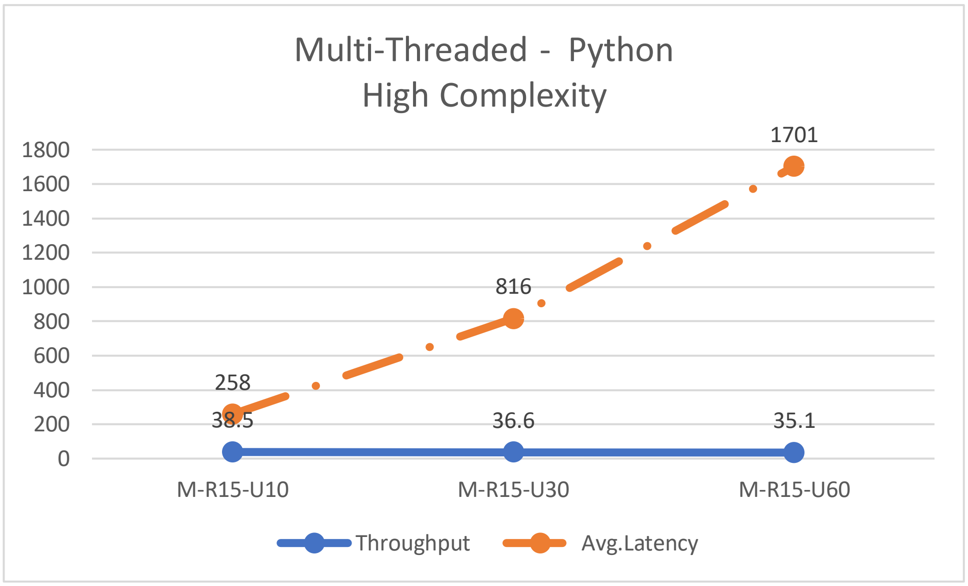


Figure 22: Performance of the API, when parallelly process cpu intensive tasks - Python

Figure 22 illustrates the performance of the web service, when number of user requests are increased with high CPU intensive tasks. Throughput is decreased, but not in a significant amount as in serial processing. Approximately there is a loss of 50 request per second in serial processing, when its complexity of the processing increased. But in parallel processing, there is around 15 request per second loss when its complexity of the processing increased in same amount. This reveals that, parallelize algorithm between multiple cores does not give significant performance improvement when it requires high CPU power. So that, instead of parallelize program, the incoming http requests to python flask web server, processed parallelly among the available CPU cores.

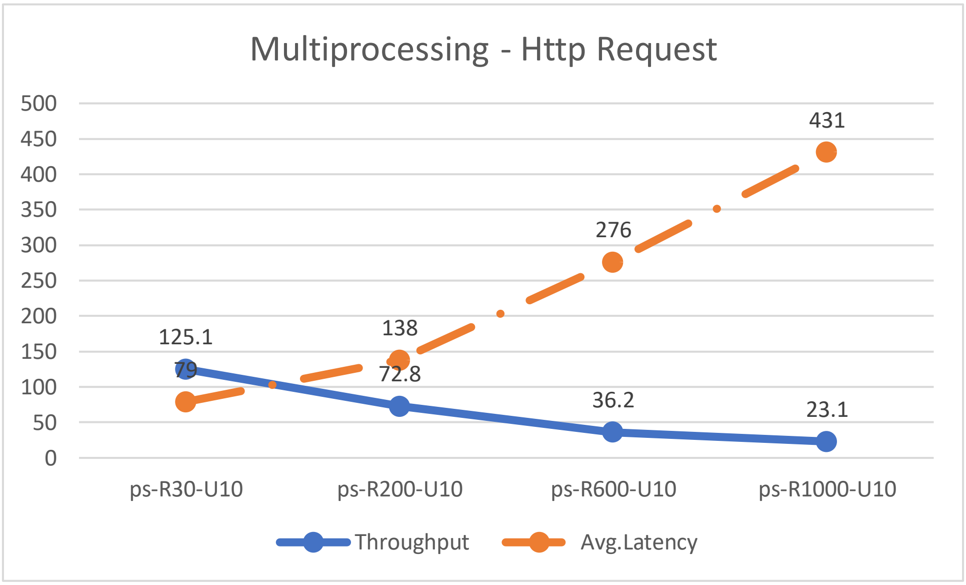
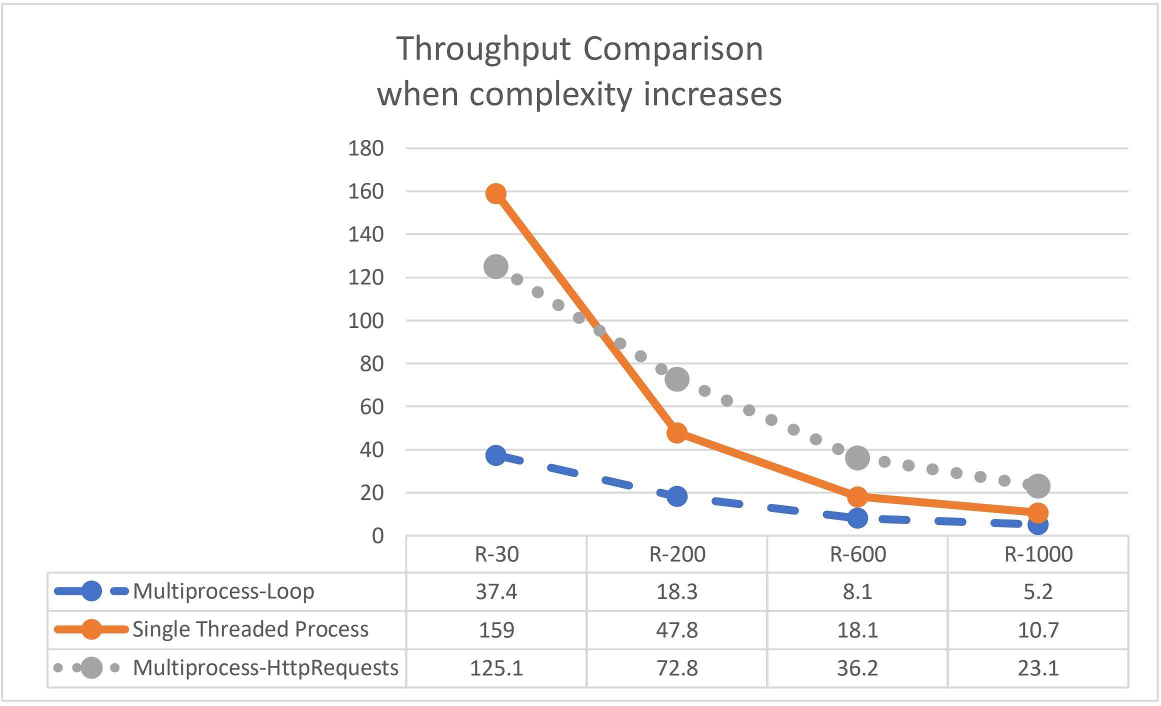


Figure 23:Test result of parallelize http requests using python multiprocessing

As shown in the Figure 23, multiprocessing http requests gives highest throughput when increasing the complexity of processing power. Following throughput comparison chart illustrates how each parallel processing approach behave, when it gets different levels of complexities for processing.

When comparing each approach, single threaded processing which does not have any parallelism, gives highest throughput, when the processing complexity is lower. (in our example, less than 200 rows in the matrix). But when processing power increases, (in this example, number of rows in the matrix from 200, 600 to 1000) multiprocessing incoming http request to web server gives highest throughput over the single processing. And it is important to note that, applying multiprocessing approach to algorithm (in this example, parallelize multiplying each matrix row and column) does not added significant advantage as shown in figure 23. The main reason might be, when multiprocessing approach applies in matrix multiplication, it requires shared memory variable to hold the value of multiplied tuple. And same time, there are more processes queued in this approach because of the number of available cores are less than the required number of processes for each request. So that latency which adds to the data communication between cores and queueing many processes, makes overall processing much slower, which impacts to the API throughput.



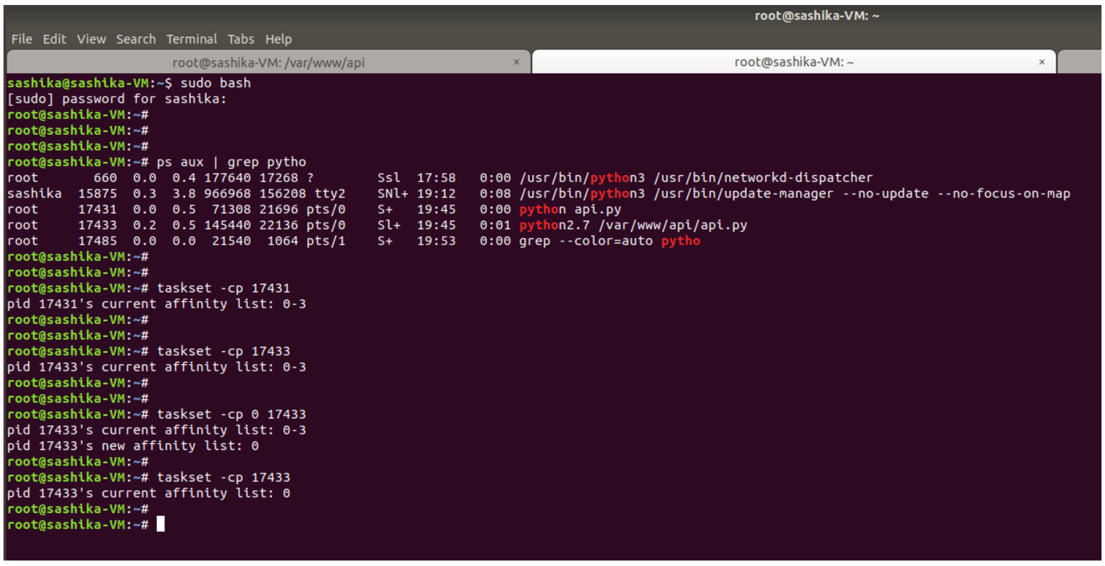
	R-30	R-200	R-600	R-1000
Multiprocess-Loop	37.4	18.3	8.1	5.2
Single Threaded Process	159	47.8	18.1	10.7
Multiprocess-HttpRequests	125.1	72.8	36.2	23.1

Figure 24: Throughput comparison of each python based parallel processing approach.

This result yields that, applying parallel processing among the processor cores always does not give the efficient result when processing web service request. It depends on the processing complexity of the algorithm, shared memory usage and number of queued processes.

Above observation makes a strong argument to evaluate web service behavior when its execution binds to a single core of the processor. When execution binds to a specific core of the processor, it does not need to depend on any other processor core or shared memory, when executing parallelly. But it creates heavy workload to the single core and remaining cores will be idle from the processing.

Linux operating system has command line tool (taskset) which can attach process to specific core of the processor. As shown in Figure 25, taskset command used to bind python web service process to processor core-0.


root@sashika-VM: ~
File Edit View Search Terminal Tabs Help
root@sashika-VM: /var/www/api
root@sashika-VM: ~

```
sashika@sashika-VM:~$ sudo bash
[sudo] password for sashika:
root@sashika-VM:~#
root@sashika-VM:~#
root@sashika-VM:~#
root@sashika-VM:~# ps aux | grep pytho
root       660  0.0  0.4 177640 17268 ?        Ssl  17:58   0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher
sashika  15875  0.3  3.8 966968 156208 tty2    SNl+ 19:12   0:08 /usr/bin/python3 /usr/bin/update-manager --no-update --no-focus-on-map
root     17431  0.0  0.5  71308 21696 pts/0    S+   19:45   0:00 python api.py
root     17433  0.2  0.5 145440 22136 pts/0    Sl+  19:45   0:01 python2.7 /var/www/api/api.py
root     17485  0.0  0.0  21540  1064 pts/1    S+   19:53   0:00 grep --color=auto pytho
root@sashika-VM:~#
root@sashika-VM:~#
root@sashika-VM:~# taskset -cp 17431
pid 17431's current affinity list: 0-3
root@sashika-VM:~#
root@sashika-VM:~#
root@sashika-VM:~# taskset -cp 17433
pid 17433's current affinity list: 0-3
root@sashika-VM:~#
root@sashika-VM:~#
root@sashika-VM:~# taskset -cp 0 17433
pid 17433's current affinity list: 0-3
pid 17433's new affinity list: 0
root@sashika-VM:~#
root@sashika-VM:~# taskset -cp 17433
pid 17433's current affinity list: 0
root@sashika-VM:~#
root@sashika-VM:~#
```

Figure 25: Process affinity using taskset command

Same performance test which is executed to compare web service throughputs of python multiprocessing, has been executed after python process bind to single processor core. Results are graphed as shown in Figure 26.

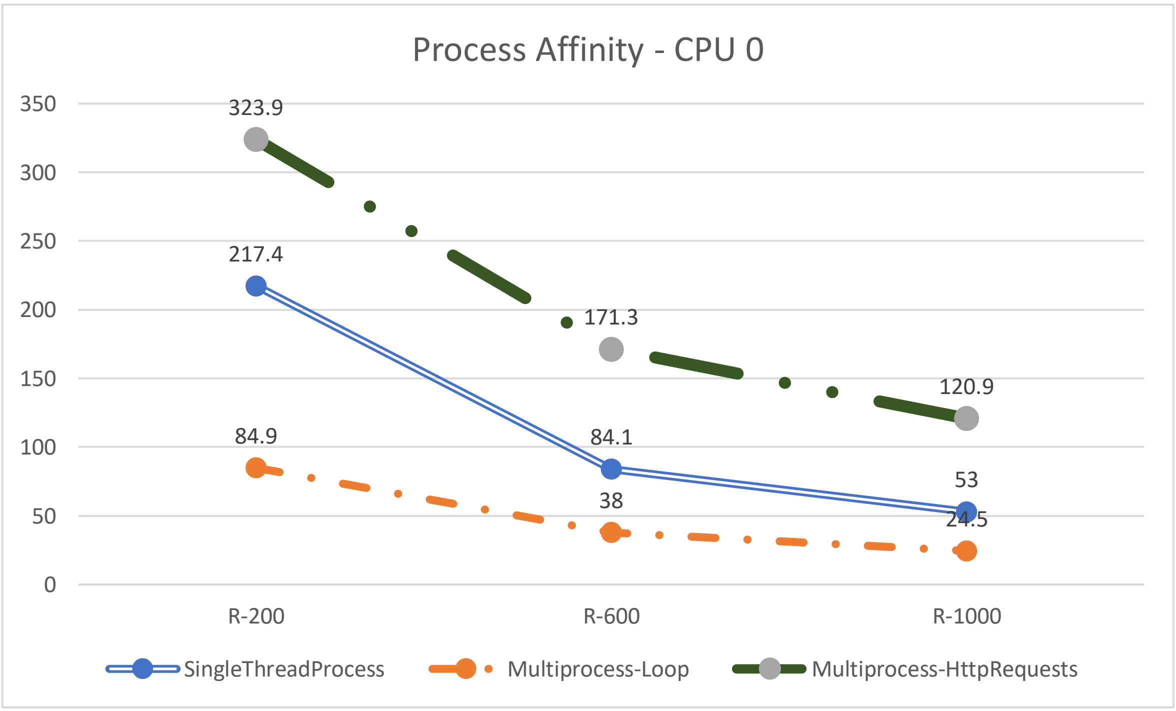


Figure 26: Throughput comparison of python based parallel execution methods, when process bind to a single processor core

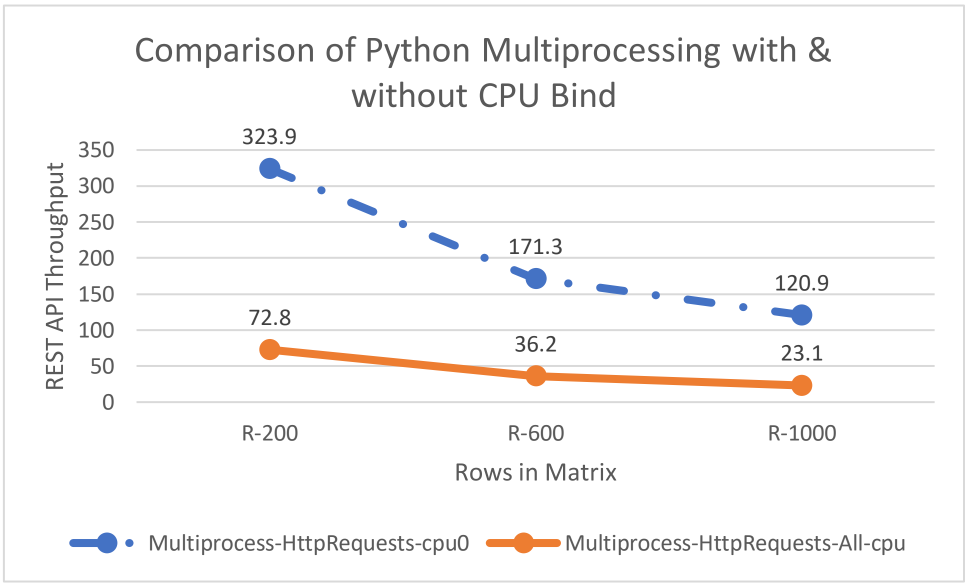


Figure 27: Comparison result of python multiprocessing parallelism with and without CPU bind

Python multiprocessing-based matrix multiplication implementations are performed much better when its bind to single core rather than it allows to use any of available cores.

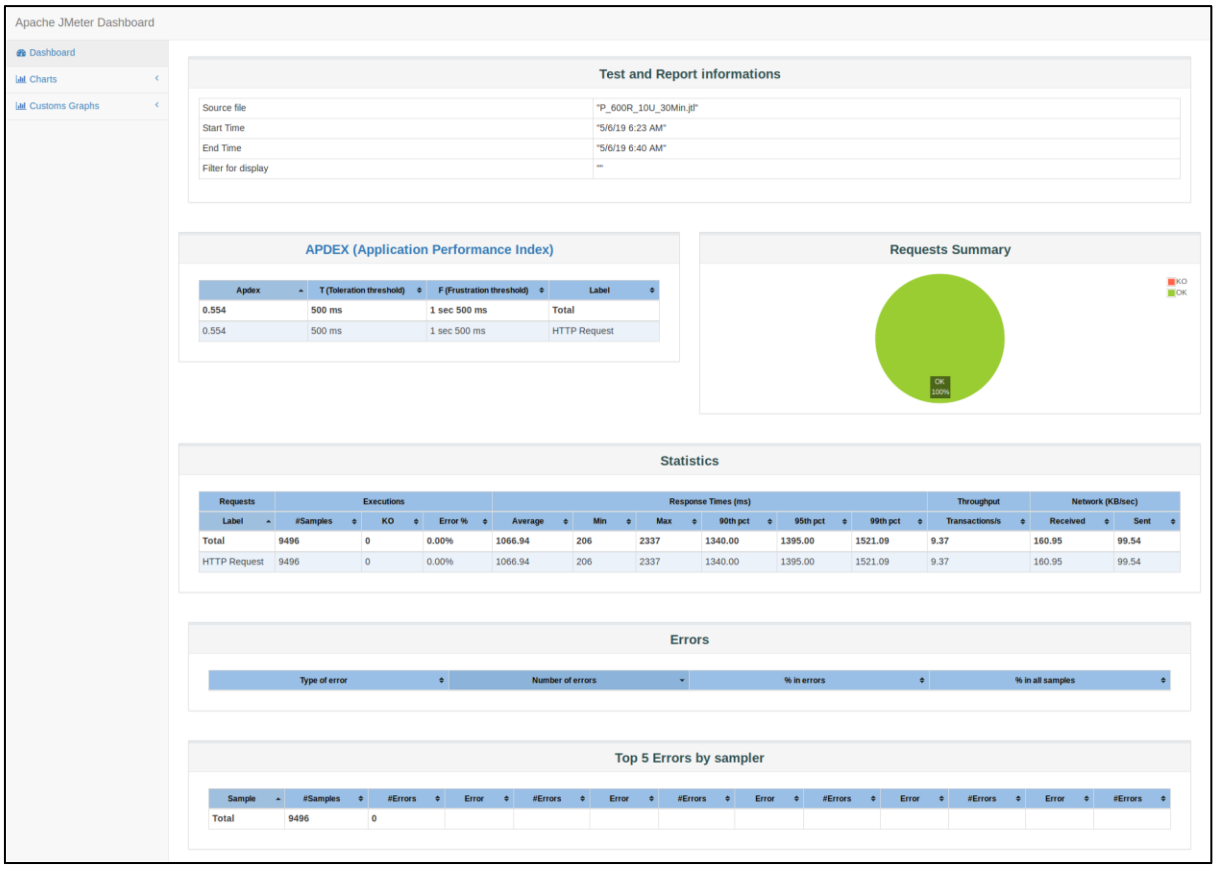
The Figure 27 shows, how best parallelism approach of python multiprocessing model behave when its process bind to single CPU core. Even though throughput gradually decrease when processing complexity increases, CPU bind approach gives much better result in each test cycle. Which indicates that, shared memory access time and process wait time for a CPU is a costly approach whenever web service has an algorithm which requires shared memory for its execution.

6.2 Jmeter Analysis Report

Apache jmeter 5.x.x version gives functionality to generate offline report, based on the test result. Since, generating html report is CPU and memory intensive task, jmeter generates reports after the test execution. Sample report's dashboard (refer Figure 28), Transaction Per Second analysis graph (refer Figure 29) and Response Time analysis graphs (refer Figure 30) are listed below. The attached details of the report are taken from the test of parallel processing with 10 jmeter threads with 30 rows in the matrix.

Jmeter Dashboard

As shown Figure 28, Jmeter dashboard shows test duration, pass and fail number of requests, detail summary of throughput etc. This overview makes the decision-making process simpler, by verifying threshold values are within the accepted range. Transaction Per Second graph (in Figure 29) gives the visibility of consistency of throughput, throughout the test execution duration. Generally, response time analysis graphs (in Figure 30) are helpful to troubleshoot issues and compare performance of web requests, and in this study, it can be used for evaluating REST API performance. When majority of the requests are receiving response time in same range, that API performs very consistently. Even though final throughput of the API is higher than the other APIs, when its response time fluctuates unevenly (without having bell shape curve) that API implementation can not be recommended over the other APIs.



Test and Report informations

Source file	"P_600R_10U_30Min.jtl"
Start Time	"5/6/19 6:23 AM"
End Time	"5/6/19 6:40 AM"
Filter for display	""

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.554	**500 ms**	**1 sec 500 ms**	**Total**
0.554	500 ms	1 sec 500 ms	HTTP Request

Requests Summary

Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**9496**	**0**	**0.00%**	**1066.94**	**206**	**2337**	**1340.00**	**1395.00**	**1521.09**	**9.37**	**160.95**	**99.54**
HTTP Request	9496	0	0.00%	1066.94	206	2337	1340.00	1395.00	1521.09	9.37	160.95	99.54

Errors

Type of error	Number of errors	% in errors	% in all samples

Top 5 Errors by sampler

Sample	#Samples	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors	Error	#Errors
Total	**9496**	**0**										

Figure 28: Jmeter Report Dashboard

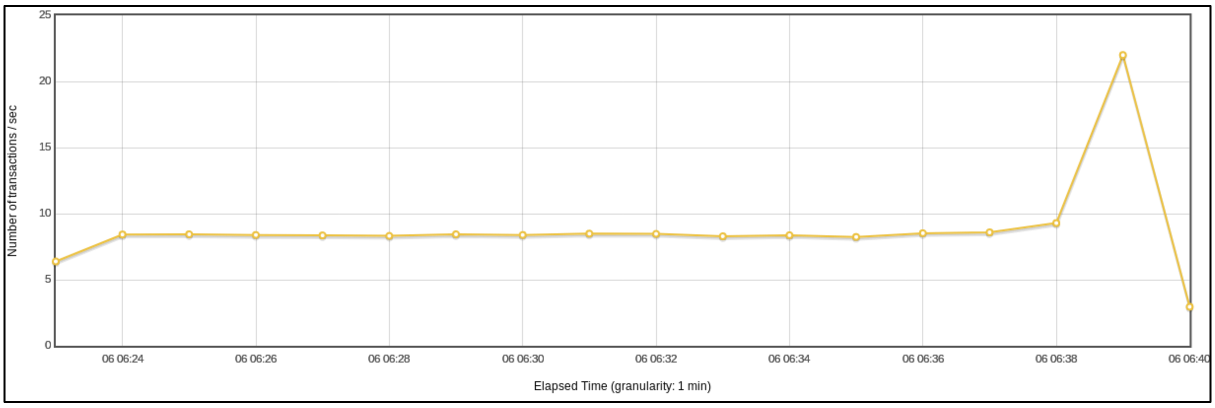
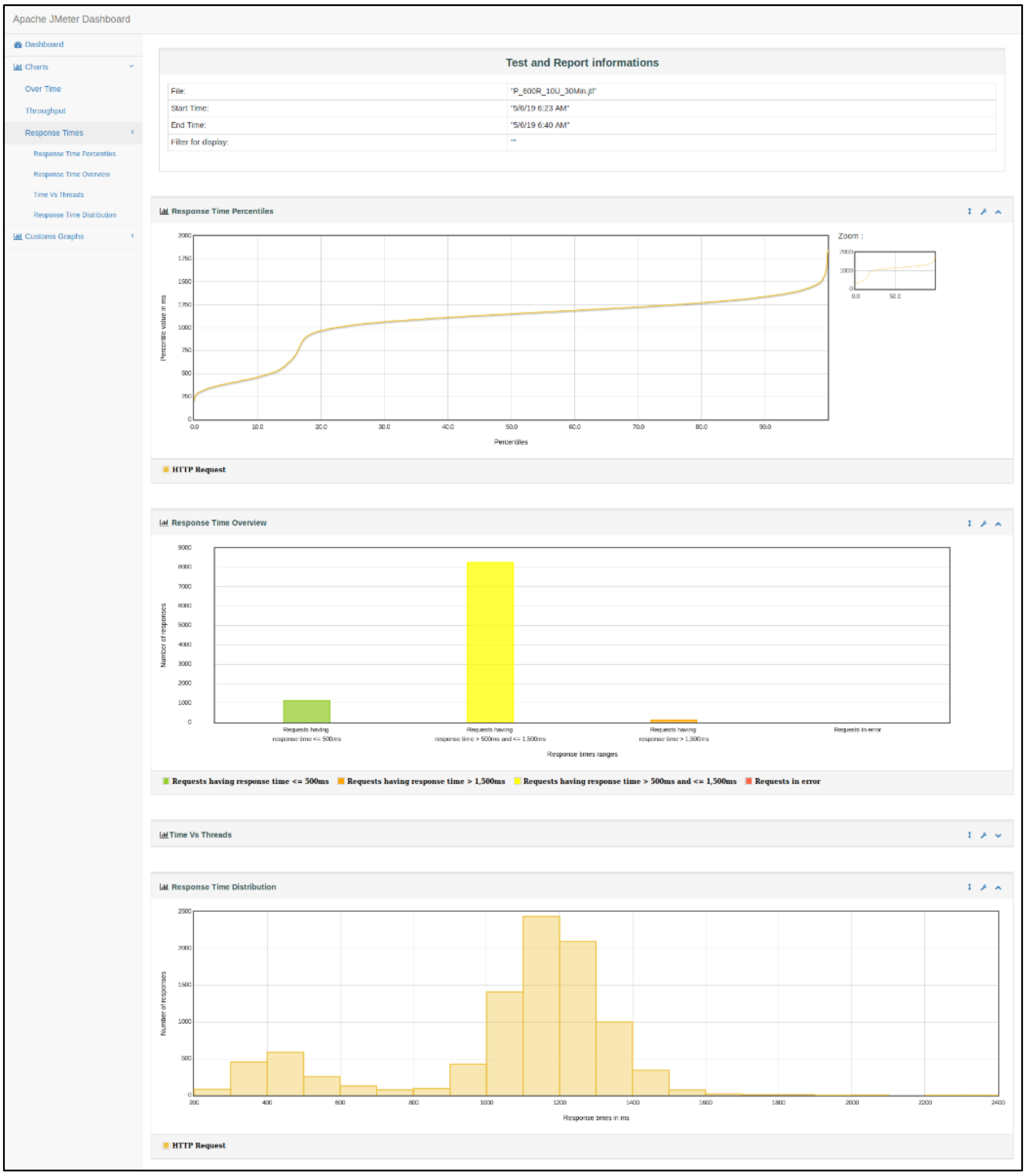


Figure 29: Jmeter Report, Transaction Per Second



Test and Report informations

File:	"P_600R_10U_30Min.jtl"
Start Time:	"5/6/19 6:23 AM"
End Time:	"5/6/19 6:40 AM"
Filter for display:	""

🟩 Requests having response time <= 500ms 🟧 Requests having response time > 1,500ms 🟨 Requests having response time > 500ms and <= 1,500ms 🟥 Requests in error

Figure 30: Jmeter Report, Response Time

7 Conclusion and Future Work

This study has been opened new aspect to examine when there is a requirement to improve performance of a web service. The efficiency of each parallel execution method against the complexity of execution algorithm, has discovered. The possible approaches to maximum utilization of CPU cores for the program execution and the amount of efficiency which can be obtained through that method also elaborated and discussed.

Following approaches of web service implementations are evaluated against the increasing number of users and increasing the processing complexity of algorithm.

- Java based single threaded sequential processing
- Java based parallel processing using multi-thread pool-based implementation
- Single core process binding (java affinity)
- Python based single process
- Python based multiprocessing, algorithm level
- Python based multiprocessing, http request
- Linux process binding

It is observed that, in each case whenever processing complexity increases, the web service throughput is decreased. But amount of decrease is vary according to the approach. Best result has observed when python web service process binds to a specific core of the processor. When there are more CPU cores available, binding to a specific core would be a resource wastage. In such cases, there will be two possible approaches to consider. Deploying multiple instances of same REST API service, where each instance binds to a specific core of the processor. Otherwise program should be able to process parallelly such a way that, each parallel process does not depend on other process, specially sharing the data.

Web service can give optimum result when it is able to process incoming http requests parallelly among the cores of the processor instead of creating multiple threads to process. Unless there is a data sharing between parallel executions, creating multiple threads for redundant work much slower than creating multiple processes, where each process can execute in different core of the processor. Because of that python multiprocessing library gives advantage over the java multi-threaded pool-based implementation.

It is also discovered that, efficiency of parallel execution depends on the complexity algorithm and CPU processing power. So that good assessment of required CPU processing power per request and expected number of requests per second, are the main pillars of selecting suitable parallel execution model for a web service.

When ever there is a requirement to improve web service performance by optimizing its execution on multi-core environment, it is recommended to test and evaluate results by binding web service instance into specific core of the processor together with other approaches.

In future, it is expected to have web service frameworks which can dynamically select suitable API endpoint to process by analyzing the web service API request complexity. For instance, whenever there is an API request with large number of rows in matrix, it will select parallel execution method otherwise it will select sequential method. Combination of both approaches will give most efficient response for many user requests.

As a continuation of this research or get maximum benefits out of this study, it is suggested to study and build a general framework, which can support parallelism and processor affinity, irrespective of business logic implementation.

8 References

[1]. Tere G.M.; Mudholkar R.R.; Jadhav B.T. (2014), Improving Performance of RESTful Web Services, IOSR Journal of Computer Science (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727 PP 12-16, (ICAET-2014)

[2]. Zhe W.U.; Harsha V. Madhyastha, Understanding the Latency Benefits of Multi-Cloud Webservice Deployments, April 2013 ACM SIGCOMM Computer Communication Review: Volume 43 Issue 2, April 2013

[3]. Fabian Gorsler; Fabian Brosig; Samuel Kounev, Performance Queries for Architecture-Level Performance Models,Proceedings of the 5th ACM/SPEC international conference on Performance engineering, March 2014 ICPE '14

[4]. S´ebastien Salva; Cl´ement Delamare; C´edric Bastoul3, Web Service Call Parallelization Using OpenMP

[5]. Daniele Bonetta; Danilo Ansaloni; Achille Peternier; Cesare Pautasso; Walter Binder, Node.Scala: Implicit Parallel Programming for High-Performance Web Services, University of Lugano (USI), Switzerland

[6]. Tilkov, S.; Vinoski, S., Node.js: Using JavaScript to Build High-Performance Net-work Programs. Internet Computing, IEEE 14(6), 80{83 (2010)

[7]. Michael Klemm; Ronald Veldema; Matthias Bezold; and Michael Philippsen, A Proposal for OpenMP for Java

[8]. R. Veldema; R. F. H. Hofman; R. A. F. Bhoedjang; H. E. Bal., Runtime optimizations for a Java DSM implementation. In 2001 joint ACM-ISCOPE Conf. on Java Grande, pages 153–162, Palo Alto, CA, USA, June 2001.

[9]. J. M. Bull; M. E. Kambites, JOMP - an OpenMP like Interface for Java

[10]. Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000

[11]. Alp Kut; Derya Birant, An Approach for Parallel Execution of Web Services, Department of Computer Engineering, Dokuz Eylul University

[12]. Docs.spring.io. (2018). 1. Spring Batch Introduction. [online] Available at: https://docs.spring.io/spring-batch/trunk/reference/html/spring-batch-intro.html [Accessed 1 Sep. 2018].

[13]. Jmeter.apache.org. (2018). Apache JMeter - Apache JMeter™. [online] Available at: https://jmeter.apache.org/ [Accessed 8 Sep. 2018].

[14]. Guru99.com. (2018). [online] Available at: https://www.guru99.com/jmeter-performance-testing.html [Accessed 4 Sep. 2018].

[15]. I-Ting Angelina Lee; Charles E. Leiserson; Tao B. Schardl; Jim Sukha; Zhunping Zhang, On-the-Fly Pipeline Parallelism

[16]. Per Brinch Hansen, A Generic Multiplication Pipeline, July 1991, School of Computer and Information Science Syracuse University Syracuse, New York.

[17]."OpenHFT/Java-Thread-Affinity", GitHub, 2019. [Online]. Available: https://github.com/OpenHFT/Java-Thread-Affinity. [Accessed: 29- Apr- 2019].

[18]. Wltrimbl.github.io, 2019. [Online]. Available: https://wltrimbl.github.io/2014-06-10-spelman/intermediate/python/04-multiprocessing.html. [Accessed: 01- May- 2019].

[19]. "Parallel Programming in Python with ease", The Nadig Blog, 2019. [Online]. Available: http://madhugnadig.com/articles/parallel-processing/2017/01/25/parallel-programming-in-python-with-ease.html. [Accessed: 01- May- 2019].

[20]. "16.6. multiprocessing — Process-based “threading” interface — Python 2.7.16 documentation", Docs.python.org, 2019. [Online]. Available: https://docs.python.org/2/library/multiprocessing.html. [Accessed: 01- May- 2019].