ABSTRACT

METHOD AND SYSTEM FOR SPLITTING AND CO-LOCATING MULTIPLE APPLICATIONS IN A MICROSERVICE

The present disclosure relates to a method and system for splitting and co-locating multiple applications in a microservice. The method comprises: comprises receiving, by a processing unit [102], a user request for execution of multiple applications; sending, by the processing unit [102] to a workload analyser [104], an indication of receipt of the user request for execution of multiple applications; fetching, by the workload analyser [104], multiple sets of a plurality of metrics related to multiple applications; checking, by the workload analyser [104], a set of predefined rules; generating, by the workload analyser [104], a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules; displaying, by a user interface unit [106], the decision generated by the workload analyser [104].

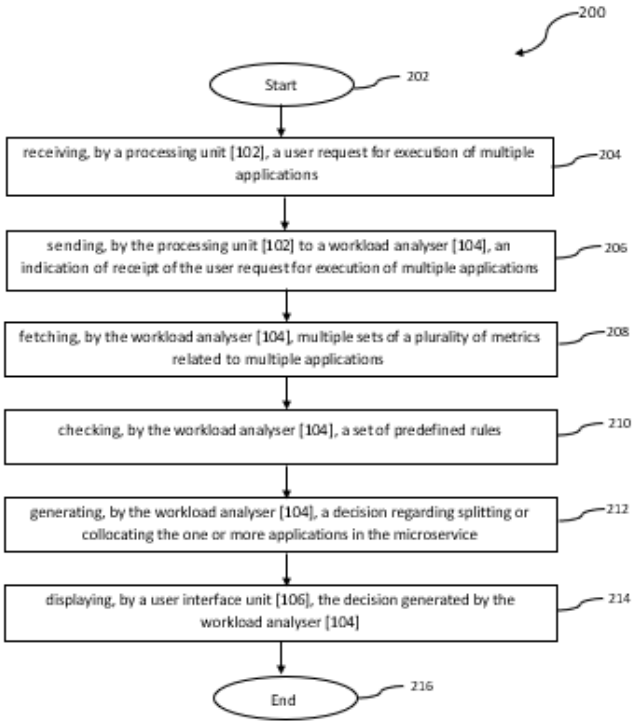

Figure 2

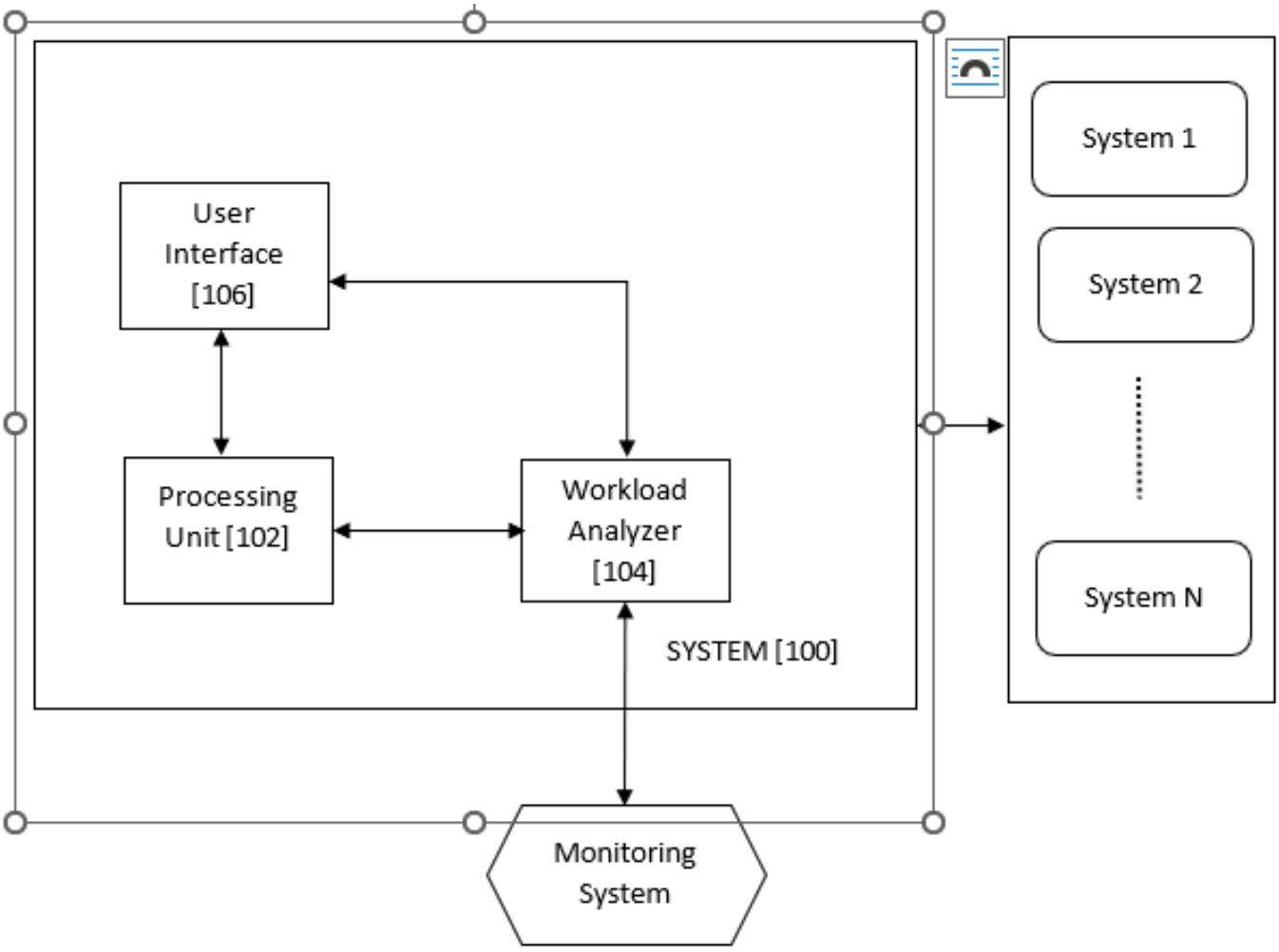

Figure 1

(GARIMA SAHNEY)
IN/PA-1826
AGENT FOR THE APPLICANT(S)
OF SAIKRISHNA & ASSOCIATES

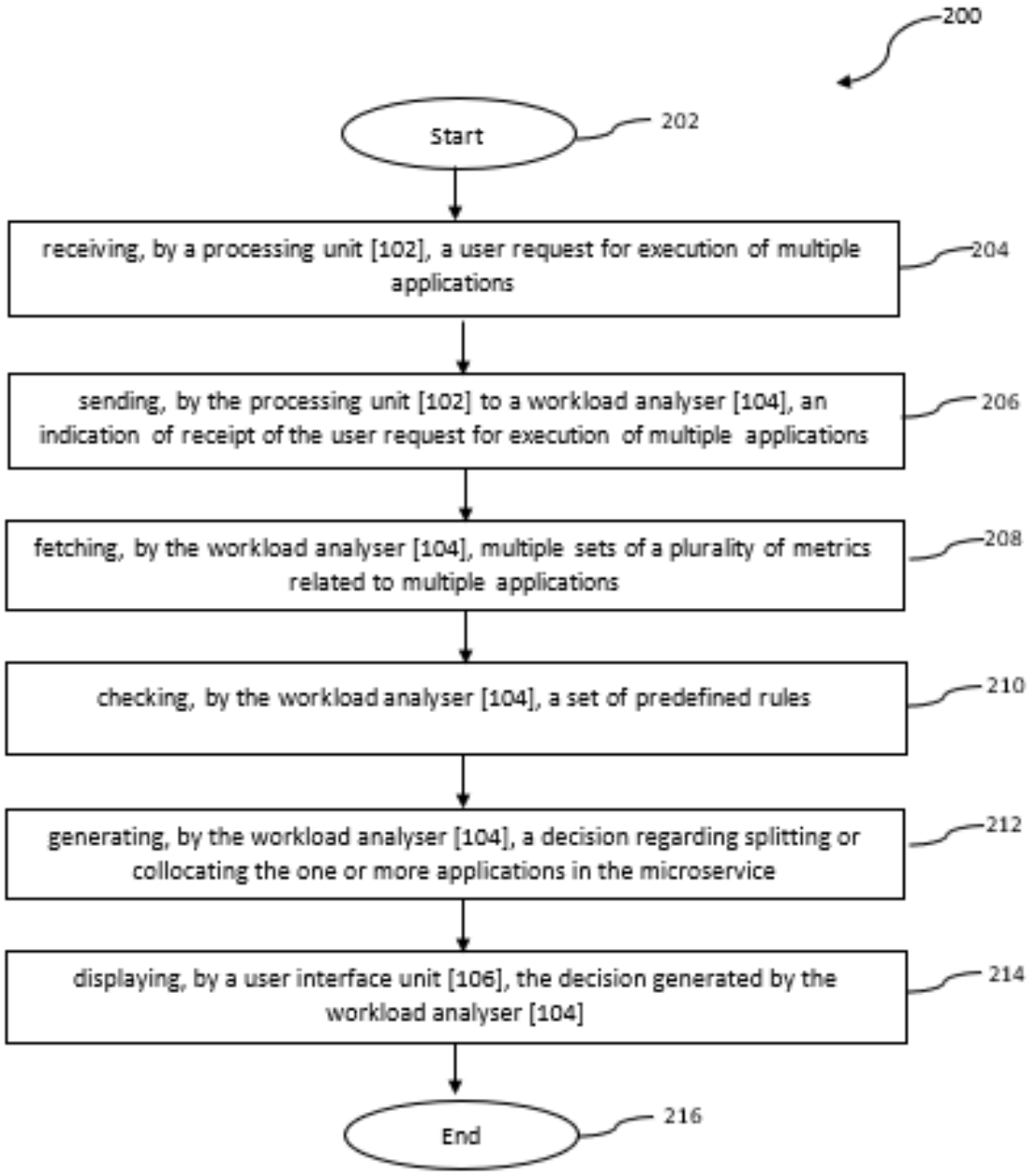

Figure 2

(GARIMA SAHNEY)
IN/PA-1826
AGENT FOR THE APPLICANT(S)
OF SAIKRISHNA & ASSOCIATES

FORM 2

THE PATENTS ACT, 1970

(39 OF 1970)

AND

THE PATENT RULES, 2003

COMPLETE SPECIFICATION

(See section 10 and rule 13)

"METHOD AND SYSTEM FOR SPLITTING AND CO-LOCATING MULTIPLE APPLICATIONS IN A MICROSERVICE"

We**, FLIPKART INTERNET PRIVATE LIMITED,** an Indian national, of Buildings Alyssa, Begonia & Clover, Embassy Tech Village, Outer Ring Road, Deverabeesanahalli Village, Bengaluru - 560103, Karnataka, India.

The following specification particularly describes the invention and the manner in which it is to be performed.

TECHNICAL FIELD:

The present invention generally relates to resource allocation in computer applications and more particularly to methods and systems for splitting and co-locating multiple applications in a microservice.

BACKGROUND OF THE DISCLOSURE:

The following description of the related art is intended to provide background information pertaining to the field of the disclosure. This section may include certain aspects of the art that may be related to various features of the present disclosure. However, it should be appreciated that this section is used only to enhance the understanding of the reader with respect to the present disclosure, and not as admissions of the prior art.

A monolithic service architecture or a complex microservice architecture can lead to cumbersome working, large system requirements, wastage of potentially needful resources, leading to expensive projects and also that are difficult to handle by the project teams. It is pertinent to understand that if all the functionalities of a project exist in a single codebase, then that application is known as a monolithic application. With the currently evolved system and numerous features provided to the users in every computer application, a monolithic application at times becomes too large and hence, difficult to manage. Further, one needs to redeploy the whole application, even for a small change. Also, as the size of the application increases, its start-up and deployment time also increases. Apart from this, it is very difficult to adopt any new technology which is well suited for a particular functionality as it affects the entire application, both in terms of time and cost. For such reasons, it is generally recommended to split a monolithic service/application architecture into multiple microservice architecture. Also, it is further recommended to split a microservice into even smaller microservices as and when needed for the systems. Thus, in a microservice architecture, the application is made up of

smaller services that handle a small portion of the functionality and data by communicating with each other directly using lightweight protocols.

However, splitting up a Monolith into Microservices or a microservice into multiple separate microservices is not an easy task. It requires human intervention and skills to decide where to spit a monolithic service or a complex microservice into small microservices and also to combine separate microservices under a single deployment/microservice. Further, even the human intervention is able to solve the purpose to a limited extent only. It may be effective in case of a small number of application programming interfaces (APIs) to be handled. However, when the number of APIs becomes large, say tens or hundreds, the human intervention is not effective in terms of providing optimal usage of resources by splitting or combining various applications (or APIs) to be handled by various deployments (or microservices). Deciding what application (or API) to split when and how to migrate pose some challenges.

When multiple APIs with different workloads are hosted in a single micro-service, a single low-throughput API can bring down the entire service as the resources (such as memory, thread, CPU, etc.) serving the APIs are all shared. This can cause unwanted and unexpected disruptions like unavailability of systems in production scenarios, which are avoidable by analyzing the workloads and optimally deciding when to split or co-locate them. While some people try to separate the resources such as threads to these APIs, it does not guarantee that the above issues will not occur as there will be other resources such as memory, CPU, network which are still shared.

There is no standard solution of the above-mentioned issues existing in the current art. People do it reactively when an issue occurs. However, a pro-active approach may be helpful in this to address the current issues. Thus, there exists an imperative need in the art to provide a method and a system for splitting and co-locating multiple applications in a microservice or a monolithic service, overcoming the above-mentioned drawbacks. This will help the users in

several ways such as discussed above and will contribute to better customer experience through better availability of resources and better output of the systems.

SUMMARY OF THE DISCLOSURE

This section is provided to introduce certain objects and aspects of the present invention in a simplified form that are further described below in the detailed description. This summary is not intended to identify the key features or the scope of the claimed subject matter.

Thus, a first object of the present disclosure is to obtain a method and system for splitting and co-locating multiple applications in a microservice or a monolithic service that overcomes the limitations of the existing approaches. Another object of the present disclosure is to obtain a method and system increases availability of resources of the systems by optimally assigning APIs to the various deployments. Yet another object of the present disclosure is to obtain a method and system that is able to obtain better output of the systems by optimally splitting or combining (or co-locating) various APIs.

In order to achieve at least one of the objectives as mentioned above, one aspect of the present invention relates to a method for splitting and co-locating multiple applications in a microservice. The method comprises receiving, by a processing unit, a user request for execution of multiple applications. Further, the method comprises sending, by the processing unit to a workload analyser, an indication of receipt of the user request for execution of multiple applications. Further, the method encompasses fetching, by the workload analyser, multiple sets of a plurality of metrics related to multiple applications, wherein each set of the plurality of metrics corresponds to one application of the multiple applications. The method further comprises checking, by the workload analyser, a set of predefined rules. Further, the method comprises generating, by the workload analyser, a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-

defined rules. This decision generated indicates splitting in an event one or more predefined rules of the set of pre-defined rules is satisfied, and indicates co-locating in an event none of the predefined rules of the set of pre-defined rules is satisfied. Finally, the method comprises displaying, by a user interface unit, the decision generated by the workload analyser.

Another aspect of the present invention relates to a system for splitting and co-locating multiple applications in a microservice. The system comprises a processing unit configured to receive a user request for execution of multiple applications. Further, the system comprises a workload analyser configured to receive an indication of receipt of the user request for execution of multiple applications by the processing unit. The workload analyser is also configured to fetch multiple sets of a plurality of metrics related to multiple applications, wherein each set of the plurality of metrics corresponds to one application of the multiple applications. further, the workload analyser is configured to check a set of predefined rules, and then generate a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules. This decision generated indicates splitting in an event one or more predefined rules of the set of pre-defined rules is satisfied, and indicates co-locating in an event none of the predefined rules of the set of pre-defined rules is satisfied. Further, the system comprises a user interface unit configured to display the decision generated by the workload analyser.

BRIEF DESCRIPTION OF DRAWINGS

The accompanying drawings, which are incorporated herein, and constitute a part of this disclosure, illustrate exemplary embodiments of the disclosed methods and systems in which like reference numerals refer to the same parts throughout the different drawings. Components in the drawings are not necessarily to scale, emphasis instead being placed upon clearly illustrating the principles of the present disclosure. Some drawings may indicate the components using block diagrams and may not represent the internal circuitry of

each component. It will be appreciated by those skilled in the art that disclosure of such drawings includes disclosure of electrical components, electronic components or circuitry commonly used to implement such components.

Figure 1 illustrates an exemplary overview of components of a system for splitting and co-locating multiple applications in a microservice, in accordance with exemplary embodiments of the present invention.

Figure 2 illustrates exemplary flow chart of a method for splitting and co-locating multiple applications in a microservice, in accordance with exemplary embodiments of the present invention.

The foregoing shall be more apparent from the following more detailed description of the disclosure.

DESCRIPTION OF THE INVENTION

In the following description, for the purposes of explanation, various specific details are set forth in order to provide a thorough understanding of embodiments of the present disclosure. It will be apparent, however, that embodiments of the present disclosure may be practiced without these specific details. Several features described hereafter can each be used independently of one another or with any combination of other features. An individual feature may not address any of the problems discussed above or might address only some of the problems discussed above.

The ensuing description provides exemplary embodiments only, and is not intended to limit the scope, applicability, or configuration of the disclosure. Rather, the ensuing description of the exemplary embodiments will provide those skilled in the art with an enabling description for implementing an exemplary embodiment. It should be understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope of the disclosure as set forth.

Specific details are given in the following description to provide a thorough understanding of the embodiments. However, it will be understood by

one of ordinary skill in the art that the embodiments may be practiced without these specific details. For example, circuits, systems, processes, and other components may be shown as components in block diagram form in order not to obscure the embodiments in unnecessary detail.

Also, it is noted that individual embodiments may be described as a process which is depicted as a flowchart, a flow diagram, a data flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed but could have additional steps not included in a figure.

The word “exemplary” and/or “demonstrative” is used herein to mean serving as an example, instance, or illustration. For the avoidance of doubt, the subject matter disclosed herein is not limited by such examples. In addition, any aspect or design described herein as “exemplary” and/or “demonstrative” is not necessarily to be construed as preferred or advantageous over other aspects or designs, nor is it meant to preclude equivalent exemplary structures and techniques known to those of ordinary skill in the art. Furthermore, to the extent that the terms “includes,” “has,” “contains,” and other similar words are used in either the detailed description or the claims, such terms are intended to be inclusive—in a manner similar to the term “comprising” as an open transition word—without precluding any additional or other elements.

As used herein, a “processor” or a “processing unit” may be a general-purpose or a special-purpose processing unit. Also, as used herein, a “processing unit” or “general-purpose processing unit” or “special-purpose processing unit” or “processor” or “operating processor” includes one or more processors, wherein processor refers to any logic circuitry for processing instructions. A processor may be a general-purpose processor, a special purpose processor, a conventional processor, a digital signal processor, a plurality of microprocessors,

one or more microprocessors in association with a DSP core, a controller, a microcontroller, Application Specific Integrated Circuits, Field Programmable Gate Array circuits, any other type of integrated circuits, etc. The processor may perform signal coding data processing, input/output processing, and/or any other functionality that enables the working of the system according to the present disclosure. More specifically, the processor or processing unit is a hardware processor.

As used herein, “a user equipment”, “a user device”, “a smart-user-device”, “a smart-device”, “an electronic device”, “a mobile device”, “a handheld device”, “a wireless communication device”, “a mobile communication device”, “a communication device” may be any electrical, electronic and/or computing device or equipment, capable of implementing the features of the present disclosure. The user equipment/device may include, but is not limited to, a mobile phone, smart phone, laptop, a general-purpose computer, desktop, personal digital assistant, tablet computer, wearable device or any other computing device which is capable of implementing the features of the present disclosure. Also, the user device may contain at least one input means configured to receive an input from a processing unit, a transceiver unit, a storage unit and any other such unit(s) which are required to implement the features of the present disclosure.

As used herein, “storage unit” or “memory unit” refers to a machine or computer-readable medium including any mechanism for storing information in a form readable by a computer or similar machine. For example, a computer-readable medium includes read-only memory (“ROM”), random access memory (“RAM”), magnetic disk storage media, optical storage media, flash memory devices or other types of machine-accessible storage media. The storage unit stores at least the data that may be required by one or more units of the system to perform their respective functions.

As used herein, a “user interface” typically includes an output device in

the form of a display, such as a liquid crystal display (LCD), cathode ray tube (CRT) monitors, light emitting diode (LED) screens, etc. and/or one or more input devices such as touchpads or touchscreens. The display may be a part of a portable electronic device such as smartphones, tablets, mobile phones, wearable devices, etc. They also include monitors or LED/LCD screens, television screens, etc. that may not be portable. The display is typically configured to provide visual information such as text and graphics. An input device is typically configured to perform operations such as issuing commands, selecting, and moving a cursor or selector in an electronic device.

As disclosed in the background section, existing technologies have many limitations and in order to overcome at least some of the limitations of the prior known solutions, the present disclosure provides a solution for splitting and co-locating multiple applications in a microservice.

Hereinafter, exemplary embodiments of the present disclosure will be described in detail with reference to the accompanying drawings so that those skilled in the art can easily carry out the present disclosure.

Referring to Figure 1, an exemplary block diagram of a system [100] for splitting and co-locating multiple applications in a microservice is shown. The system [100] comprises a processing unit [102] configured to receive one or more user requests for execution of multiple applications. This one or more user requests for execution of multiple execution may relate to same or different applications. For example, a first user may raise a login request at a digital platform. And at the same time a second user may raise a read request related to a product or service on the same digital platform. Also, at the same time, a third user may raise a write request on the same digital platform. Further, such requests of say, login, read, write, etc. may be raised by multiple users at the same time, meaning that, multiple users, for example, may raise a read request on the digital platform at the same time. Now, say, the system X is assigned to perform three tasks comprising login, read, and write. A user may raise such

request by accessing the digital platform using his/her user device. The processing unit [102] receives the user request(s) and gives an indication of receipt of such requests to the workload analyser [104]. The workload analyser [104] by receiving that indication knows the tasks to be performed, the application programming interfaces (APIs) involved in the same.

The workload analyser [104] fetches multiple sets of a plurality of metrics related to multiple applications. Each set of the plurality of metrics corresponds to one application (i.e., the API) of the multiple applications. These plurality of metrics for each API is fetched by the workload analyser [104] from a third-party monitoring system. This third-party system can be any system that is configured to obtain application metrics. For example, these systems may include Prometheus, SignalFx, Datadog, Wavefront and NewRelic. The metrics relate to a system (say for example, system 1 as shown in Figure 1) on which the multiple APIs are expected to run. Thus, the workload analyser [104], in this example, fetches the metrics relates to system 1. The metrics can be performance metrics of system 1 such as throughput (which indicates the amount of work performed by Application Server), latency (refers to the delay between a user's action and a web application's response to that action), etc. Also, say in another example, where there are more than one systems to run the applications, the plurality of metrics of each of the system is fetched by the workload analyser [104].

Further, the workload analyser [104] checks a set of predefined rules. These predefined rules can be dependencies-based rule, a throughput-based rule, a latency-based rule, and a cache-based rule. A dependencies-based rule is based on datastores, caches, upstream services, etc. Here, a “datastore” refers to a storage system used to store transactional or analytical information; a “cache” refers to a temporary storage of data to read and/or write data quickly; and an “upstream service” refers to a service that that the current service depends on to perform its operation. In an example, two APIs that have different upstream services such as a “read” service and a “delete” service on the digital

platform, they should be separated and included under different deployments. In another example, say, the sets of data to be fetched in two different operations are stored in two different datastores or the memory units, then also the APIs should be separated. In a throughput-based rule, if the throughputs of two different APIs are more than or equal to 10 times (that is, the throughput of one application or the API is greater than or equal to 10 times the throughput of another API, then also the two APIs should be separated and included under different deployments. Similarly in a latency-based rule, if the expected latencies of two different APIs are more than or equal to 10 times (that is, the expected latency of one application or the API is greater than or equal to 10 times the expected latency of another API, then the two APIs should be separated and included under different deployments. Furthermore, according to the cache-based rule, say, two APIs both have high-throughput and low latency, but do not have a adequate common-caches, then the two APIs should be separated and included under different deployments. A person skilled in the art would appreciate that although the above rules are explained using the example of 2 APIs only, but the same would be applicable, as also encompassed by the disclosure, for more than 2 APIs as well. Also, it is pertinent to mention here that the task of splitting and co-locating the APIs by the system may be performed severally or in combination, or may not be even performed if in case the APIs are already distributed optimally, or after checking the parameters, it requires no action of splitting and co-locating the APIs to be performed.

Further, the workload analyser [104] generates a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules. If any one of the above rules is satisfied, irrespective of other rules being satisfied of not, the workload analyser [104] generates a decision to split the applications and put them under different deployments. Thus, this decision generated indicates splitting of APIs into two different deployments or microservices in an event one or more predefined rules

of the set of pre-defined rules is satisfied, and indicates co-locating of APIs under same deployment in case none of the predefined rules of the set of pre-defined rules is satisfied.

In an implementation, one or more buckets are created prior to the workload analyser [104] generating a decision regarding splitting or co-locating the one or more applications in the microservice. For example, say, the throughput of three APIs under one microservice are 20 units (API1), 80 units (API2) and 250 units (API3). Now, the API1 and API3 with 20 units and 250 units respectively should be separated as their throughput is more than 10 times different. However, this is not so if one compares between (API1 and API2) and also between (API2 and API3). This issue may be solved if one or more buckets are created in the processing unit [102]. This may also be created by human intervention and be stored in a memory unit [not shown]. The workload analyser [104] in such case is configured to fetch the data from the memory unit [not shown] directly or via the processing unit [102] and decide based on the data as to how to separate the APIs. For example, in the above case of 3 APIs of throughputs 20 units (API1), 80 units (API2) and 250 units (API3), if the buckets of 200 units throughput range are created, then the APIs under 0-200 units range may be kept together. This means that API1 and API2 would be co-located under same deployment and API3 would be separated. Say for example, System 1 (as shown in Figure 1) was to execute the AP1, API2, and API3, then the workload analyser [104] may generate a decision to split the applications and to assign API1 and API2 to System 1 and to assign API3 to, say, System 2. Similarly, other rules would also apply in the same manner in this implementation. Also in this way, two or more APIs may be split into different microservices, or co-located under same microservice, or may be co-located under a separate microservice by the system.

Further, the user interface unit [106] displays the decision generated by the workload analyser [104]. This decision is shown to the users for performing

various other tasks related to, say, a bigger project. For example, the user might want to tweak the placement of some APIs under one or separate deployment of any reason on real-time basis as per the dynamic requirements, etc. This display for the decision would help the user know the current status of the system APIs.

Referring to Figure 2 which illustrates an exemplary flow chart of a method for splitting and co-locating multiple applications in a microservice, in accordance with exemplary embodiments of the present invention. The method of Figure 2 starts at step 202 and goes to step 204. At step 204, the method includes receiving, by a processing unit [102], a user request for execution of multiple applications. This one or more user requests for execution of multiple execution may relate to same or different applications. For example, a first user may raise a login request at a digital platform. And at the same time a second user may raise a read request related to a product or service on the same digital platform. Also, at the same time, a third user may raise a write request on the same digital platform. Further, such requests of say, login, read, write, etc. may be raised by multiple users at the same time, meaning that, multiple users, for example, may raise a read request on the digital platform at the same time. Now, say, the system X is assigned to perform three tasks comprising login, read, and write. A user may raise such request by accessing the digital platform using his/her user device.

Further at step 206, the processing unit [102] after receiving user request(s) gives an indication of receipt of such requests to the workload analyser [104]. The workload analyser [104] by receiving that indication knows the tasks to be performed, the application programming interfaces (APIs) involved in the same.

Further at step 208, The workload analyser [104] fetches multiple sets of a plurality of metrics related to multiple applications. Each set of the plurality of metrics corresponds to one application (i.e., the API) of the multiple applications. These plurality of metrics for each API is fetched by the workload analyser [104]

from a third-party monitoring system. This third-party system can be any system that is configured to obtain application metrics, such as Newrelics, Prometheus, SignalFx, Datadog, Wavefront, etc. The metrics relate to a system (say for example, system 1 as shown in Figure 1) on which the multiple APIs are expected to run. Thus, the workload analyser [104], in this example, fetches the metrics relates to system 1. The metrics can be performance metrics of system 1 such as throughput (which indicates the amount of work performed by Application Server), latency (refers to the delay between a user's action and a web application's response to that action), etc. Also, say in another example, where there are more than one systems to run the applications, the plurality of metrics of each of the system is fetched by the workload analyser [104].

Further, at step 210, the workload analyser [104] checks a set of predefined rules. These predefined rules can be dependencies-based rule, a throughput-based rule, a latency-based rule, and a cache-based rule. A dependencies-based rule is based on datastores, caches, upstream services, etc. Here, a “datastore” refers to a storage system used to store transactional or analytical information; a “cache” refers to a temporary storage of data to read and/or write data quickly; and an “upstream service” refers to a service that that the current service depends on to perform its operation. In an example, two APIs that have different upstream services such as a “read” service and a “delete” service on the digital platform, they should be separated and included under different deployments. In another example, say, the sets of data to be fetched in two different operations are stored in two different datastores or the memory units, then also the APIs should be separated. In a throughput-based rule, if the throughputs of two different APIs are more than or equal to 10 times (that is, the throughput of one application or the API is greater than or equal to 10 times the throughput of another API, then also the two APIs should be separated and included under different deployments. Similarly in a latency-based rule, if the expected latencies of two different APIs are more than or equal to 10 times (that

is, the expected latency of one application or the API is greater than or equal to 10 times the expected latency of another API, then the two APIs should be separated and included under different deployments. Furthermore, according to the cache-based rule, say, two APIs both have high-throughput and low latency, but do not have a adequate common-caches, then the two APIs should be separated and included under different deployments. A person skilled in the art would appreciate that although the above rules are explained using the example of 2 APIs only, but the same would be applicable, as also encompassed by the disclosure, for more than 2 APIs as well.

Further, at step 212, the workload analyser [104] generates a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules. If any one of the above rules is satisfied, irrespective of other rules being satisfied of not, the workload analyser [104] generates a decision to split the applications and put them under different deployments. Thus, this decision generated indicates splitting of APIs into two different deployments or microservices in an event one or more predefined rules of the set of pre-defined rules is satisfied, and indicates co-locating of APIs under same deployment in case none of the predefined rules of the set of pre-defined rules is satisfied.

In an implementation, one or more buckets are created prior to the workload analyser [104] generating a decision regarding splitting or co-locating the one or more applications in the microservice. For example, say, the throughput of three APIs under one microservice are 20 units (API1), 80 units (API2) and 250 units (API3). Now, the API1 and API3 with 20 units and 250 units respectively should be separated as their throughput is more than 10 times different. However, this is not so if one compares between (API1 and API2) and also between (API2 and API3). This issue may be solved if one or more buckets are created in the processing unit [102]. This may also be created by human intervention and be stored in a memory unit [not shown]. The workload analyser

[104] in such case is configured to fetch the data from the memory unit [not shown] directly or via the processing unit [102] and decide based on the data as to how to separate the APIs. For example, in the above case of 3 APIs of throughputs 20 units (API1), 80 units (API2) and 250 units (API3), if the buckets of 200 units throughput range are created, then the APIs under 0-200 units range may be kept together. This means that API1 and API2 would be co-located under same deployment and API3 would be separated. Say for example, System 1 (as shown in Figure 1) was to execute the AP1, API2, and API3, then the workload analyser [104] may generate a decision to split the applications and to assign API1 and API2 to System 1 and to assign API3 to, say, System 2. Similarly, other rules would also apply in the same manner in this implementation.

Thus, in an example indicating the overall working, say, there is a Catalog Service comprising catalog details of products on an e-commerce digital platform, and an Order Service comprising order details of the orders placed on the e-commerce digital platform. Say, the Catalog Service has a GET and POST and DELETE interface to get, upsert or delete a catalog item. The Catalog Service might have a very high throughput and low latency requirements for the read API (GET). Further, it might even be served from a different datastore such as a cache. The write API (POST) might be rare compared to the read API and high latencies on this API might be acceptable. The delete API might be even rarer and might be asynchronous in nature. When all of these functionalities of the same service are deployed as a single deployment, a failure of one or a higher latency of one API might negatively impact another API. This might be completely unnecessary and avoidable. For example, say, the GET API is working at a throughput of 10k queries per second (qps) with a average latency of 20 ms. The POST API is working at a throughput of 15qps and having an average latency of 500ms. To serve 10k qps with an average latency of 20ms, one will need 200 threads at least. To serve 15qps with an average latency of 500ms, one will need 7 threads at least. Now, say, 5 machines each provisioned with 50 threads each

are available to service the request. If the write API becomes latent, and its latency spikes to say 2 seconds, it will suddenly need 30 threads to process the request. This will give fewer threads for the read APIs to process and it will be in the web-service queue. This will in-turn increase the latency of the read API. If the latency of the read API increases to 30ms, we will need 300 threads to process the request at a throughput of 10k. This will in turn increase the wait time further and create a spiraling effect to in-turn make all the requests latent. So when there are two APIs with drastically different throughput and latency requirements that are co-located, the impact of one affecting the other is extremely high. As seen in the example mentioned above, a mere 10ms increase in the latency increased the number of threads from 200 to 300. In that case, splitting the applications into different deployments or co-locating the applications under same deployment is much required for optimum performance of the systems.

Finally, at step 214, the user interface unit [106] displays the decision generated by the workload analyser [104]. This decision is shown to the users for performing various other tasks related to, say, a bigger project. For example, the user might want to tweak the placement of some APIs under one or separate deployment of any reason on real-time basis as per the dynamic requirements, etc. This display for the decision would help the user know the current status of the system APIs. Thus, the process ends at step 214.

Thus, the present invention provides a novel solution for splitting and co-locating multiple applications in a microservice or a monolithic service. The present invention provides a solution that is technically advanced over the currently known solutions as it optimally determines which APIs to split and which APIs to co-locate in a microservice. Based on the implementation of features of the present invention, the method and system increases availability of resources of the systems by optimally assigning APIs to the various deployments. Further, by implementing the features of the present solution, a

person would be able to obtain a method and system that is able to obtain better output of the systems by optimally splitting or co-locating various APIs.

While considerable emphasis has been placed herein on the preferred embodiments, it will be appreciated that many embodiments can be made and that many changes can be made in the preferred embodiments without departing from the principles of the invention. These and other changes in the preferred embodiments of the invention will be apparent to those skilled in the art from the disclosure herein, whereby it is to be distinctly understood that the foregoing descriptive matter to be implemented merely as illustrative of the invention and not as limitation.

We claim:

1. A method for splitting and co-locating multiple applications in a microservice, the method comprising:
   - receiving, by a processing unit [102], one or more user requests for execution of multiple applications;
   - sending, by the processing unit [102] to a workload analyser [104], an indication of receipt of the one or more user requests for execution of multiple applications;
   - fetching, by the workload analyser [104], multiple sets of a plurality of metrics related to multiple applications, wherein each set of the plurality of metrics corresponds to one application of the multiple applications;
   - checking, by the workload analyser [104], a set of predefined rules;
   - generating, by the workload analyser [104], a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules,

     wherein the decision generated indicates splitting in an event one or more predefined rules of the set of pre-defined rules is satisfied, and

     wherein the decision generated indicates co-locating in an event none of the predefined rules of the set of pre-defined rules is satisfied;
   - displaying, by a user interface unit [106], the decision generated by the workload analyser [104].
2. The method as claimed in claim 1, wherein the predefined rules comprise at least a dependencies-based rule, a throughput-based rule, a latency-based rule, and a cache-based rule.
3. The method as claimed in claim 1, wherein one or more buckets are created prior to generating, by the workload analyser [104], a decision

regarding splitting or co-locating the one or more applications in the microservice.

4. The method as claimed in claim 3, wherein the one or more buckets are created based on the values of plurality of metrics fetched by the workload analyser [104].

5. A system for splitting and co-locating multiple applications in a microservice, the system comprising:

- a processing unit [102] configured to:
    - receive one or more user requests for execution of multiple applications;
- a workload analyser [104] configured to:
    - receive an indication of receipt of the one or more user requests for execution of multiple applications by the processing unit [102];
    - fetch multiple sets of a plurality of metrics related to multiple applications, wherein each set of the plurality of metrics corresponds to one application of the multiple applications;
    - check a set of predefined rules;
    - generate a decision regarding splitting or co-locating the one or more applications in the microservice based on the checking of the set of pre-defined rules,

      wherein the decision generated indicates splitting in an event one or more predefined rules of the set of pre-defined rules is satisfied, and

      wherein the decision generated indicates co-locating in an event none of the predefined rules of the set of pre-defined rules is satisfied; and
- a user interface unit [106] configured to:
    - display the decision generated by the workload analyser [104].

6. The system as claimed in claim 5, wherein the predefined rules comprise at least a dependencies-based rule, a throughput-based rule, a latency-based rule, and a cache-based rule.
7. The system as claimed in claim 5, wherein the workload analyser [104] is configured to create one or more buckets prior to generating the decision regarding splitting or co-locating the one or more applications in the microservice.
8. The system as claimed in claim 7, wherein the workload analyser [104] is configured to create one or more buckets based on the values of plurality of metrics.

Dated this **30th** day of **November, 2022**



(GARIMA SAHNEY)
IN/PA-1826
AGENT FOR THE APPLICANT(S)
OF SAIKRISHNA & ASSOCIATES