Technical Disclosure Commons

Defensive Publications Series

April 2022

SYSTEM AND METHOD OF LOCKING AND RANKING THE RESOURCE FOR UTILIZATION IN A DISTRIBUTED SCHEDULING SYSTEM

Srijan Mukherjee
VISA

Nishant Kumar
VISA

Dhanuesh R C
VISA

Arunkumar P
VISA

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Mukherjee, Srijan; Kumar, Nishant; R C, Dhanuesh; and P, Arunkumar, "SYSTEM AND METHOD OF LOCKING AND RANKING THE RESOURCE FOR UTILIZATION IN A DISTRIBUTED SCHEDULING SYSTEM", Technical Disclosure Commons, (April 06, 2022)
https://www.tdcommons.org/dpubs_series/5048

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

SYSTEM AND METHOD OF LOCKING AND RANKING THE RESOURCE FOR UTILIZATION IN A DISTRIBUTED SCHEDULING SYSTEM

VISA

INVENTORS:

SRIJAN MUKHERJEE

NISHANT KUMAR

DHANUESH R C

ARUNKUMAR P

TECHNICAL FIELD

[0001] The present subject matter is related to payment platforms and payment infrastructure, and more particularly related to a system and method of locking and ranking the resource for utilization in a distributed scheduling system.

BACKGROUND

[0002] In a distributed environment, there are multiple schedulers which facilitate parallel job scheduling. A distributed system is a computing environment in which various components are spread across multiple computers (or other computing devices) on a network. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task. Computers in distributed systems "share resources" among themselves, like hardware (disks and printers), software (files, windows, and data objects), and data.

[0003] A job scheduler is responsible for scheduling ready jobs present in a pool amongst a collection of resources. Managing business processes with a job scheduler streamlines operations and saves time and money. Having an enterprise scheduling solution is a serious resource advantage for any organization. According to a research of Ecological Momentary Assessment (EMA) survey, the number one area where automation delivers the most business value is in reducing operational costs. To increase the profit margin, implementing enterprise job scheduling software is essential, as it is an effective way to reap better results. In the early days of batch job scheduling, automation was something that only Information Technology (IT) professionals dealt with. But as technology advances and new software is implemented, it becomes increasingly important for employees across all departments to be able to manage complex, automated workflows. For example, in an organization, the jobs which are to be processed will always grow and it would be an unwise strategy to always increase the computational power to deal with the increasing jobs. However, the major issues in a distributed environment are ‘race-condition’ and ‘starvation’ when numerous schedulers attempt to access the jobs.

[0004] The first possibility is starvation, which occurs when a Scheduler A (SA) picks up a job to execute, and the other schedulers keep waiting for SA to wait for the job to be executed. This is not a feasible solution as there are multiple schedulers idle which may delegate jobs to

the resources. And in the meantime, if more jobs keep getting added to the job pool, the waiting time for the newly added jobs also increases. In the second possibility, if multiple schedulers are allowed to pick up the jobs simultaneously, then it is possible that the multiple schedulers submit the same job to the resources for the execution. This leads to race conditions and inconsistency and is hence not a viable situation. Along with this, there are jobs which may have higher priority during the job throttling by the schedulers. Also, there are jobs which may take lesser time compared to other jobs. In the case where prioritization of the jobs is absent, it can lead to starvation of higher priority jobs by a lower priority and longer running job. Thus, a robust and synchronized mechanism is required to avoid race-condition among the schedulers, and starvation among both the jobs and the schedulers.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate exemplary embodiments and, together with the description, serve to explain the disclosed principles. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same numbers are used throughout the figures to reference like features and components. Some embodiments of device or system and/or methods in accordance with embodiments of the present subject matter are now described, by way of example only, and with reference to the accompanying figures, in which:

[0006] Fig. 1 illustrates an exemplary environment of a locking and ranking of resources in a distributed scheduling system consistent with the present disclosure.

[0007] Fig. 2 shows a block diagram illustrating a process for ranking the consumed job based on a job property.

[0008] Fig. 3 shows a flowchart illustrating a process of locking and ranking of resources for utilization in a distributed scheduling system.

[0009] Fig. 4 illustrates a block diagram explaining the current implementation approach of prioritizing tasks in a job pool.

[0010] Fig. 5 shows a flowchart illustrating a ranking process of consumed jobs using a rank function.

[0011] Fig. 6 shows a flowchart illustrating a dynamic throttler process flow.

[0012] Fig. 7 illustrates a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0013] The figures depict embodiments of the disclosure for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the disclosure described herein.

DESCRIPTION OF THE DISCLOSURE

[0014] In the present document, the word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or implementation of the present subject matter described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments.

[0015] While the disclosure is susceptible to various modifications and alternative forms, specific embodiment thereof has been shown by way of example in the drawings and will be described in detail below. It should be understood, however that it is not intended to limit the disclosure to the particular forms disclosed, but on the contrary, the disclosure is to cover all modifications, equivalents, and alternative falling within the spirit and the scope of the disclosure.

[0016] The terms “comprises”, “comprising”, or any other variations thereof, are intended to cover a non-exclusive inclusion, such that a setup, device or method that comprises a list of components or steps does not include only those components or steps but may include other components or steps not expressly listed or inherent to such setup or device or method. In other words, one or more elements in a device or system or apparatus proceeded by “comprises… a” does not, without more constraints, preclude the existence of other elements or additional elements in the device or system or apparatus.

[0017] The terms "an embodiment", "embodiment", "embodiments", "the embodiment", "the embodiments", "one or more embodiments", "some embodiments", and "one embodiment" mean "one or more (but not all) embodiments of the invention(s)" unless expressly specified otherwise.

[0018] The terms "including", "comprising", “having” and variations thereof mean "including but not limited to", unless expressly specified otherwise.

[0019] The present disclosure proposed a system and a method of locking and ranking the resource for utilization in a distributed scheduling system. Initially, all the jobs have a property based on timestamp which is initialized as “null”, “nil” or “zero-valued”. A job scheduler is responsible for scheduling ready jobs present in a pool amongst a collection of resources. Scheduling is a decision-making process which enables resource sharing among several activities by determining their execution order on the set of available resources. Prior to scheduling the jobs, all the jobs are ranked among each other based on the properties of the present job. In case of an addition of one or more jobs within time, the ranking may be re-evaluated and reflected in the pool of jobs (job ranked pool). Further, the present disclosure discloses ranking of jobs to ensure that there is unbiased throttling of the jobs. Further, to enable concurrent job scheduling, a lock-based state management may be used before allocating the jobs to the job scheduler. The lock-based state is basically binary in nature, where a lock on a job is in one of two states, either acquired state or released state. The jobs may be assigned a lock state based on the time of their access by the schedulers. The job scheduler then picks up the jobs which are not in lock state and then allocates the job on the resource based on the resource availability using a rank function. In the case where the job is not picked up by the resource at that instant, and to avoid scheduler starvation, the lock on the job may be removed. In the subsequent iterations, the jobs with no locks will be picked up by another scheduler. The advantage of using a lock based mechanism is that it not only prioritizes jobs based on their properties, but also , avoids job starvation and race-condition.

[0020] **Fig. 1** illustrates an exemplary illustration 100 of a system for implementing embodiments consistent with the present disclosure. A job is an entity which is formulated using a set of instructions, which requires resources for its execution. Consume jobs and add lock 101 is initiated once the jobs are selected for execution by a job scheduler, where jobs are protected by providing a timestamp to each job as well as locking each job. The lock on the job may be removed in two ways. In the first way, the job scheduler itself assigns the null value to

the timestamp of the job, making the job free from the lock. The other way to unlock the job is by comparing the non-null timestamp value of the job with the ‘Time-Out time’. The time-out time is fixed and constant for all the jobs. In the case when the modulus difference of the timestamp and the current time exceeds the timeout duration, the lock on the job is removed automatically. The time-out duration is introduced to increase the fault tolerance. In the case of a job scheduler failure, the job may be delayed in the scheduler since the scheduler is unable to progress and may hold the job waiting indefinitely. Thus, by enforcing a time-out, the jobs may not have to wait for an infinite amount of time, and the lock on the job may be removed as soon as the condition for timeout is matched.

[0021] In an embodiment, a set of jobs from the job pool is selected by a job scheduler and the consumed jobs 103 are ranked using the ranking function. The rank function considers the various factors linked to the job. The factors may include, without limiting to, any number of resources on which the job has to be executed, the computation power required by the job, and/or if any parallelism is required by the job as well as the level of parallelism required by the job. The jobs execution environment is a production environment or a non-production environment. Here, the term environment refers to an environment in which the resources are deployed and/or stored. A resource is considered to be in a production environment if the resource, in some embodiments is exposed to the customers. A resource is considered to be in the non-production environment if the resource is only used internally, for example, for performing certain quality control operations.

[0022] In an embodiment, the rank function may also consider numerical factors which may be reconfigured and normalized based on the job link and all the numerical values in the same range may be fetched. Based on the numerical factors, a linear function is used to assign a final normalized value to a job. The higher the magnitude of the normalized value, the higher will be the priority of the job. Further, using the prioritization technique, the jobs are ranked and added to a new ranked pool, which acts as a ‘First-in-First-out’ data structure whenever required by the job scheduler. For example, consider a job pool, say List 1, which contains numerous jobs ranging from Job-1 to Job-n, where each job has several properties as shown in Fig. 2. Thereafter, the rank function extracts the job properties and sorts the list of the jobs based on job priority. For example, a job (say Job 4) which is to be run in a production environment has a higher priority than other jobs (say Job 1 or Job 2) which is to be executed

in a non-production environment. Further, the newly ranked pool of jobs is persisted in a new list say List 2 and is retrieved from the list whenever required by the job scheduler.

[0023] In an embodiment, after prioritizing the job, the job scheduler selects the job from the ranked pool and allocates them to the resource in throttle the jobs 105. Before allocating the jobs to the resource, the job scheduler verifies the capacity of the resource. The job scheduler continues the process for other jobs in the queue if the resource is not in a state to select a certain type of job. For the jobs which were not allocated to the resource, the lock is removed by setting the timestamp property to null. Further, the released job may now be selected by other job schedulers. When one of the job schedulers fails or is down due to unforeseen reasons, then the job may be fixed for an infinite amount of time without being executed. Further, using the Time-Out time, the job may be set free and made available to be picked up again by an active job scheduler and allocates them to the resource.

[0024] **Fig. 3** shows a flowchart illustrating a process of locking and ranking of resources for utilization in a distributed scheduling system. **At block 301**, the method comprises using a job scheduler, scheduling the ready jobs in a pool across a collection of resources. **At block 303**, the method comprises selecting a job for execution, applying a lock on the job by assigning the current timestamp, which is initialized as empty. Here, the term “empty” may mean “null”, “nil”, or “empty”. The job scheduler assigns the empty value to the timestamp based on a property of the job, after the job execution. The job locking ensures that when a task has been selected from the list for execution by the throttler, it may not be selected again. **At block 305**, the method comprises using a rank function, prioritizing, and ranking the jobs before adding them to a new pool (job ranked pool). Based on the job properties, the jobs are prioritized and then rank function sorts the list of jobs based on their priority. A new job pool of sorted jobs is created. Further, **at block 307**, the method comprises selecting a job from the job ranked pool and allocating them to the resource. The throttler picks up the job from the ranked list and allocates them to the resource and verifies if the resource is available to execute the next job.

[0025] In an embodiment, Ansible job throttle using a static model is a system which is able to throttle the jobs submitted to a job execution orchestrator “Ansible Tower” based on any set parameters which involve queuing the jobs. The major approach is when API calls are made to the Ansible Tower for executing jobs at a regular interval of Time T (where T will be different for different instance groups) as long as the queue is not empty. Thereafter, based on the response, a batch of N jobs shall be throttled to the queue and as and when the jobs are throttled,

the corresponding information will be logged into the Lookup table in a distributed, in-memory caching system such as “Hazelcast”. Further, no changes are made to polling. In an embodiment, Hazelcast may also be used to implement a plurality of distributed in-memory queues, which may be used to partition the submitted jobs based on various instance groups. Look-up table in Hazelcast may also be required for backend implementation. Thereafter, the API for executing jobs is called at a regular interval of time after throttling the jobs for all the queues, as shown in Fig. 4. Further, the relative chronology of the tasks may be retained when ordering the jobs using a rank function. The various rank parameters are tabulated in Table 1 below.

Parameters	Default value	Where to find
Forks	5 – unless overwritten in ansible.config file which may not be able to locate	api/v2/job_templates/ — direct value
Timeout	No timeout (0 – waiting)	api/v2/job_templates/ — direct value
Labels	NA and Optional field	api/v2/job_templates/ — direct value
Modified + Last run	NA (can be used as a tie breaker)	api/v2/job_templates/ – direct values
User type(roles)	NA ()	
Job Type	only 2 values, run and check, allowed	api/v2/job_templates/ – direct values
Fork available	No default value	api/v2/instance_groups/ — multiply capacity * percentage_remaning_capacity

Table 1

Rank Function/Evaluator:

[0026] Fig. 5 shows a flowchart illustrating a ranking process of consumed jobs using a rank function. In step-1, a rank of the jobs will be obtained using various parameters including ‘Job Environment’, ‘Server Count’, ‘Slice Count’, ‘Forks Count’ and ‘Waiting Time’. Job environment is based on Production (Prod)/Non-Production (NonProd), where the jobs with "Prod" have higher priority than "NonProd" (Directly Proportional). Server Count is the job with a higher number of servers which may provide higher priority (Directly Proportional).

Slice Count is the job with a higher slice count which may be given higher priority (Directly Proportional). Forks Count is the job with higher forks which may be given higher priority (Directly Proportional). Waiting Time is the job with a higher waiting time which may be given higher priority (Directly Proportional).

[0027] In step-2, the parameters are normalized to perform mathematical operation.

- *Server Count* → *serverCount*
- *Slice Count* → *sliceCount*
- *Forks Count* → *forksCount*

[0028] In step-3, equation 1 is applied to scale the values with range [min, max] to a new range [a, b].

$$Z_i = \frac{(b-a)(x_i - \min(x))}{\max(x) - \min(x)} + a \qquad \ldots 1$$

[0029] In step-4, after the normalization, the three parameter become:

- $serverCount \rightarrow normalizedServerCount$
- $sliceCount \rightarrow normalizedSliceCount$
- $forksCount \rightarrow normalizedForksCount$

[0030] In step-5, the logical operation of summation is applied to these parameters based on their proportionality with the priority.

$$\text{normalizedValue} = \text{normalizedServerCount} + \text{normalizedSliceCount} + \text{normalizedForksCount}$$

[0031] In step-6, the obtained "**normalizedValue**" may be assigned to all the jobs in queue for the execution. In step-7, the job with the higher value of "**normalizedValue**" will be given a better rank (or will have higher priority). In step-8, when the "**normalizedValue**" is same for two jobs, the Waiting Time of the job will be used as the tie-breaker. The job with higher waiting time will get higher priority.

[0032] **Fig. 6** shows a flowchart illustrating a dynamic throttler process flow. After initializing the process, consume a list of jobs, where the pseudo-code for all available jobs is given by:

```
consumeJob {
    FOR all available jobs:
        IF job is consumed;
```

```
                continue;
            ELSE
                mark job as consumed;
                add to list based on key;
            ENDIF
        ENDFOR
    }
```

[0033] Thereafter, the job list is verified to find if it is empty or not. If the job list is empty, then the process ends. If the job is list is not empty, then the flow proceeds to verify if all the jobs processed or not. Timestamp is applied when the job is not processed or timestamp not present or timestamp aged condition and add to the job list based on key. Upon verification of the job process, a rank evaluator/function is initiated and the pseudo-code for the rank evaluator is given below:

```
    rankEvaluator {
        FOR all lists based on keys:
            Order the jobs in list;
        ENDFOR
    }
```

[0034] In an embodiment, after applying rank evaluator, API call is made to get resources available to allocate the jobs. Further, the throttle batch is initiated to complete the allocation of the jobs. Pseudo-code for the throttle batch is given by:

```
    throttleBatch {
        FOR all lists based on keys:
            make API call to get resources;
            IF resources available:
                make a batch of jobs;
                throttle the batch;
            ENDIF
        ENDFOR
    }
```

[0035] After ranking the Ansible jobs, the data needs to be stored in the Database (db) for later use and the weights required for ranking are modified. Thus, every time the ranking of the Jobs

is done, the complete ranked queue is stored in the db. Further, while utilizing the ranked data for modelling techniques, the entire history of how the job was ranked, as well as storing all the parametric values along with the rank of the job may be calculated during the rankFunction(). For example, a parameter like "Time spent by the ansible job in the queue" can be used as an optimizer.

General computer system:

[0036] **Fig. 7** illustrates a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0037] In an embodiment, the computer system 700 may be used to implement the system. The computer system 700 may include a central processing unit (“CPU” or “processor”) 702. The processor 702 may include at least one data processor of locking and ranking resource for utilization in a distributed system. The processor 702 may include specialized processing units such as, integrated system (bus) controllers, memory management control units, floating point units, graphics processing units, digital signal processing units, etc.

[0038] The processor 702 may be disposed in communication with one or more Input/Output (I/O) devices (712 and 713) via I/O interface 701. The I/O interface 701 employ communication protocols/methods such as, without limitation, audio, analog, digital, monoaural, radio corporation of America (RCA) connector, stereo, IEEE-1394 high speed serial bus, serial bus, universal serial bus (USB), infrared, personal system/2 (PS/2) port, Bayonet Neill-Concelman (BNC) connector, coaxial, component, composite, digital visual interface (DVI), high-definition multimedia interface (HDMI), radio frequency (RF) antennas, S-Video, video graphics array (VGA), IEEE 802.11b/g/n/x, Bluetooth, cellular e.g., code-division multiple access (CDMA), high-speed packet access (HSPA+), global system for mobile communications (GSM), long-term evolution (LTE), worldwide interoperability for microwave access (WiMax), or the like, etc.

[0039] Using the I/O interface 701, the computer system 700 may communicate with one or more I/O devices such as input devices 712 and output devices 713. For example, the input devices 712 may be an antenna, keyboard, mouse, joystick, (infrared) remote control, camera, card reader, fax machine, dongle, biometric reader, microphone, touch screen, touchpad, trackball, stylus, scanner, storage device, transceiver, video device/source, etc. The output

devices 713 may be a printer, fax machine, video display (e.g., cathode ray tube (CRT), liquid crystal display (LCD), light-emitting diode (LED), plasma, plasma display panel (PDP), organic light-emitting diode display (OLED) or the like), audio speaker, etc.

[0040] In some embodiments, the processor 702 may be disposed in communication with a communication network 709 via a network interface 703. The network interface 703 may communicate with the communication network 709. The network interface 1503 may employ connection protocols including, without limitation, direct connect, ethernet (e.g., twisted pair 10/100/1000 Base T), transmission control protocol/internet protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc. The communication network 709 may include, without limitation, a direct interconnection, local area network (LAN), wide area network (WAN), wireless network (e.g., using Wireless Application Protocol), the Internet, etc. Using the network interface 703 and the communication network 709, the computer system 700 may communicate with a database 714, which may be the enrolled templates database 713. The network interface 703 may employ connection protocols include, but not limited to, direct connect, ethernet (e.g., twisted pair 10/100/1000 Base T), transmission control protocol/internet protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc.

[0041] The communication network 709 includes, but is not limited to, a direct interconnection, a peer to peer (P2P) network, local area network (LAN), wide area network (WAN), wireless network (e.g., using Wireless Application Protocol), the Internet, Wi-Fi and such. The communication network 709 may either be a dedicated network or a shared network, which represents an association of the different types of networks that use a variety of protocols, for example, hypertext transfer protocol (HTTP), transmission control protocol/internet protocol (TCP/IP), wireless application protocol (WAP), etc., to communicate with each other. Further, the communication network 709 may include a variety of network devices, including routers, bridges, servers, computing devices, storage devices, etc.

[0042] In some embodiments, the processor 702 may be disposed in communication with a memory 705 (e.g., RAM, ROM, etc. not shown in Fig. 7) via a storage interface 704. The storage interface 704 may connect to memory 705 including, without limitation, memory drives, removable disc drives, etc., employing connection protocols such as, serial advanced technology attachment (SATA), integrated drive electronics (IDE), IEEE-1394, universal serial bus (USB), fiber channel, small computer systems interface (SCSI), etc. The memory drives may further include a drum, magnetic disc drive, magneto-optical drive, optical drive,

redundant array of independent discs (RAID), solid-state memory devices, solid-state drives, etc.

[0043] The memory 705 may store a collection of program or database components, including, without limitation, user interface 706, an operating system 707, etc. In some embodiments, computer system 700 may store user/application data, such as, the data, variables, records, etc., as described in this disclosure. Such databases may be implemented as fault-tolerant, relational, scalable, secure databases such as Oracle or Sybase.

[0044] The operating system 707 may facilitate resource management and operation of the computer system 1500. Examples of operating systems include, without limitation, Apple™ Macintosh ™ OS X™, UNIX™, Unix-like system distributions (e.g., Berkeley Software Distribution (BSD), FreeBSD™, Net BSD™, Open BSD™, etc.), Linux distributions (e.g., Red Hat™, Ubuntu™, K-Ubuntu™, etc.), International Business Machines (IBM™) OS/2™, Microsoft Windows™ (XP™, Vista/7/8, etc.), Apple iOS™, Google Android™, Blackberry™ operating system (OS), or the like.

[0045] In some embodiments, the computer system 700 may implement web browser 708 stored program components. Web browser 708 may be a hypertext viewing application, such as Microsoft™ Internet Explorer™, Google Chrome™, Mozilla Firefox™, Apple™ Safari™, etc. Secure web browsing may be provided using secure hypertext transport protocol (HTTPS), secure sockets layer (SSL), transport layer security (TLS), etc. Web browsers 708 may utilize facilities such as AJAX, DHTML, Adobe™ Flash, Javascript, Application Programming Interfaces (APIs), etc. In some embodiments, the computer system 700 may implement a mail server stored program component. The mail server may be an Internet mail server such as Microsoft Exchange, or the like. The mail server may utilize facilities such as ASP, ActiveX, ANSI C++/C#, Microsoft .NET, Common Gateway Interface (CGI) scripts, Java, JavaScript, PERL, PHP, Python, WebObjects, etc. The mail server may utilize communication protocols such as Internet Message Access Protocol (IMAP), Messaging Application Programming Interface (MAPI), Microsoft Exchange, Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), or the like. In some embodiments, the computer system 700 may implement a mail client stored program component. The mail client may be a mail viewing application, such as Apple Mail, Microsoft Entourage, Microsoft Outlook, Mozilla Thunderbird, etc.

[0046] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term "computer-readable medium" should be understood to include tangible items and exclude carrier waves and transient signals, i.e., be non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, Compact Disc (CD) ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0047] The described operations may be implemented as a method, system or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The described operations may be implemented as code maintained in a "non-transitory computer readable medium", where a processor may read and execute the code from the computer readable medium. The processor is at least one of a microprocessor and a processor capable of processing and executing the queries. A non-transitory computer readable medium may include media such as magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, DVDs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, Flash Memory, firmware, programmable logic, etc.), etc. Further, non-transitory computer-readable media may include all computer-readable media except for a transitory. The code implementing the described operations may further be implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.).

[0048] The illustrated steps are set out to explain the exemplary embodiments shown, and it should be anticipated that ongoing technological development will change the manner in which particular functions are performed. These examples are presented herein for purposes of illustration, and not limitation. Further, the boundaries of the functional building blocks have been arbitrarily defined herein for the convenience of the description. Alternative boundaries can be defined so long as the specified functions and relationships thereof are appropriately performed. Alternatives (including equivalents, extensions, variations, deviations, etc., of those

described herein) will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. Such alternatives fall within the scope and spirit of the disclosed embodiments. Also, the words "comprising," "having," "containing," and "including," and other similar forms are intended to be equivalent in meaning and be open ended in that an item or items following any one of these words is not meant to be an exhaustive listing of such item or items, or meant to be limited to only the listed item or items. It must also be noted that as used herein, the singular forms “a,” “an,” and “the” include plural references unless the context clearly dictates otherwise.

[0049] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term “computer readable medium” should be understood to include tangible items and exclude carrier waves and transient signals, i.e., are non-transitory. Examples include random access memory (RAM), read-only memory (ROM), volatile memory, non-volatile memory, hard drives, CD ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0050] Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the embodiments of the disclosure is intended to be illustrative, but not limiting, of the scope of the disclosure.

[0051] With respect to the use of substantially any plural and/or singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

SYSTEM AND METHOD OF LOCKING AND RANKING THE RESOURCE FOR UTILIZATION IN A DISTRIBUTED SCHEDULING SYSTEM

ABSTRACT

The present disclosure relates to a system and a method of locking and ranking the resource for utilization in a distributed scheduling system. The method comprises prioritizing the jobs from the job pool based on the job properties and allocating the jobs to the scheduler. Before allocating the jobs to the scheduler, if the jobs are consumed, then a lock is added to the jobs based on state management. The jobs may be assigned a lock state based on the time of their access by the schedulers. The scheduler is configured to pick up the jobs which are not in lock state. Thereafter, the set of jobs from the list selected by the job scheduler is ranked using a rank function. The ranking may be re-evaluated, if one or more jobs are added within the time and reflected in the new job pool. Ranking of jobs ensures that there is unbiased throttling of the jobs. Further, the job scheduler allocates the job on the resource based on the resource availability. The advantages of the present disclosure are that is helps in prioritizing the jobs based on their properties, and scheduling the jobs on a job scheduler by avoiding scheduler starvation and race-condition among the schedulers.

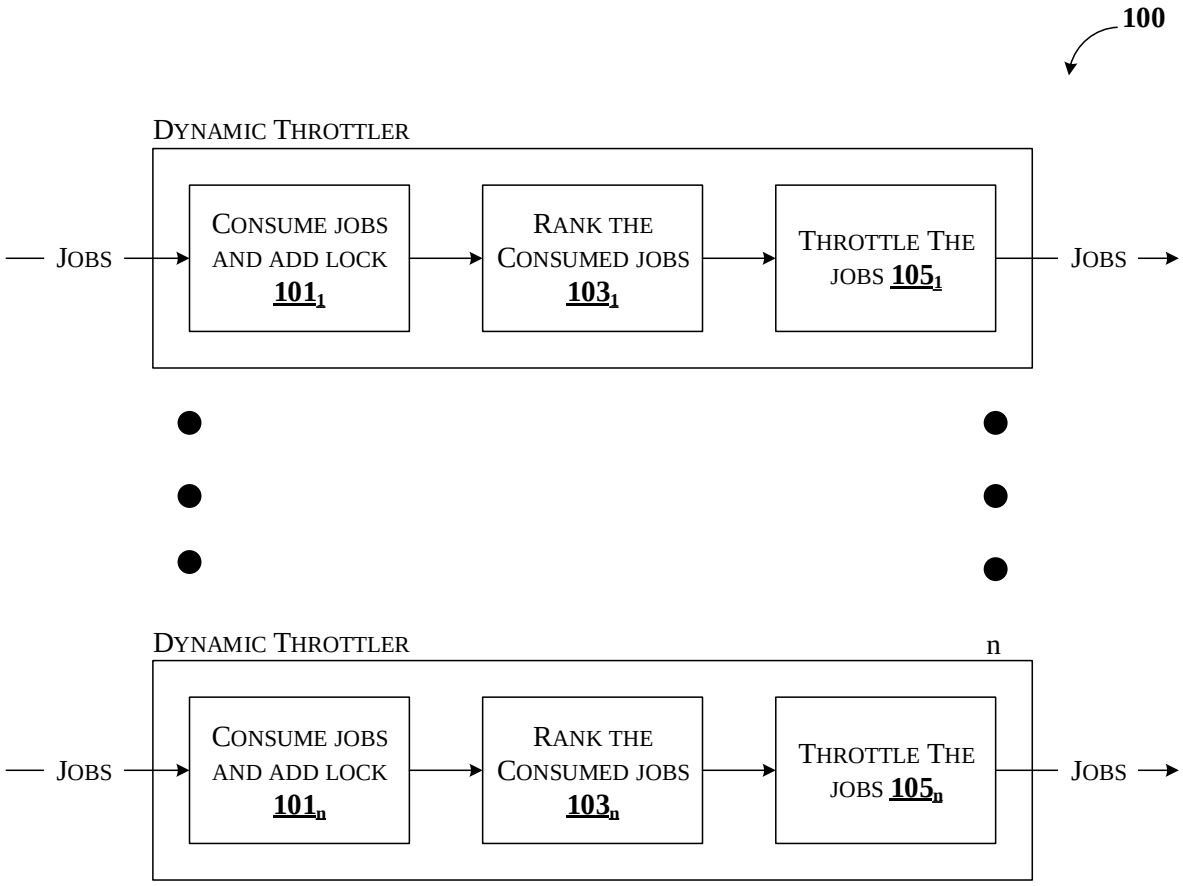
100
DYNAMIC THROTTLER
JOBS
CONSUME JOBS AND ADD LOCK 101₁
RANK THE CONSUMED JOBS 103₁
THROTTLE THE JOBS 105₁
JOBS
DYNAMIC THROTTLER
n
JOBS
CONSUME JOBS AND ADD LOCK 101ₙ
RANK THE CONSUMED JOBS 103ₙ
THROTTLE THE JOBS 105ₙ
JOBS

Fig. 1

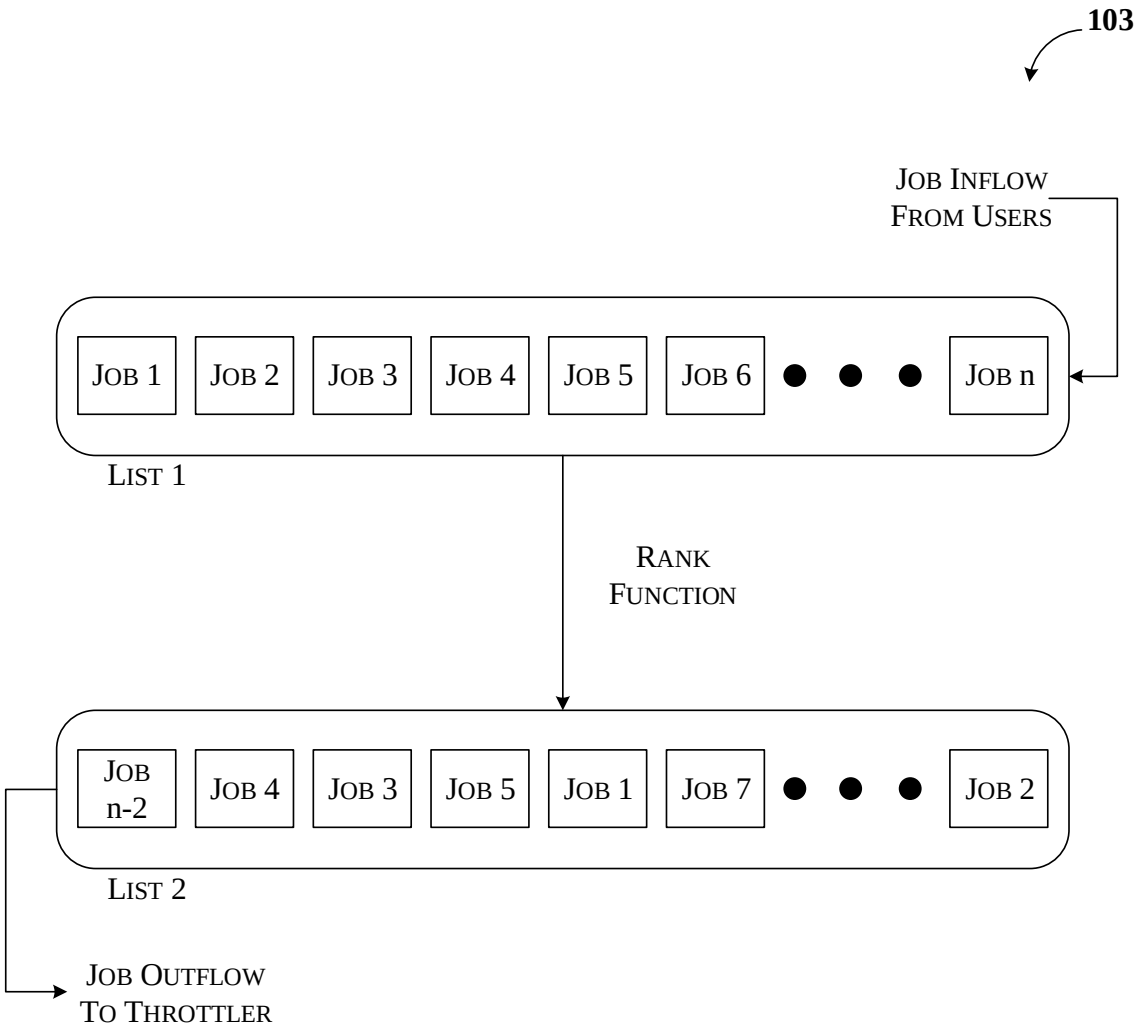
103
JOB INFLOW
FROM USERS
JOB 1
JOB 2
JOB 3
JOB 4
JOB 5
JOB 6
JOB n
LIST 1
RANK
FUNCTION
JOB
n-2
JOB 4
JOB 3
JOB 5
JOB 1
JOB 7
JOB 2
LIST 2
JOB OUTFLOW
TO THROTTLER

Fig. 2

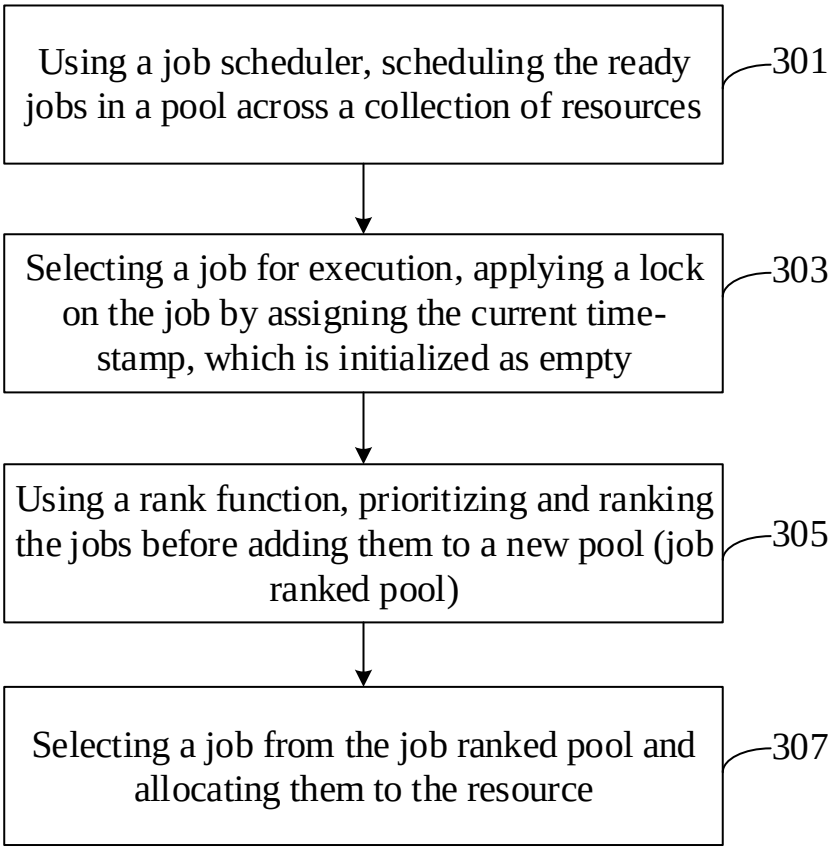
Using a job scheduler, scheduling the ready jobs in a pool across a collection of resources
301
Selecting a job for execution, applying a lock on the job by assigning the current time-stamp, which is initialized as empty
303
Using a rank function, prioritizing and ranking the jobs before adding them to a new pool (job ranked pool)
305
Selecting a job from the job ranked pool and allocating them to the resource
307

Fig. 3

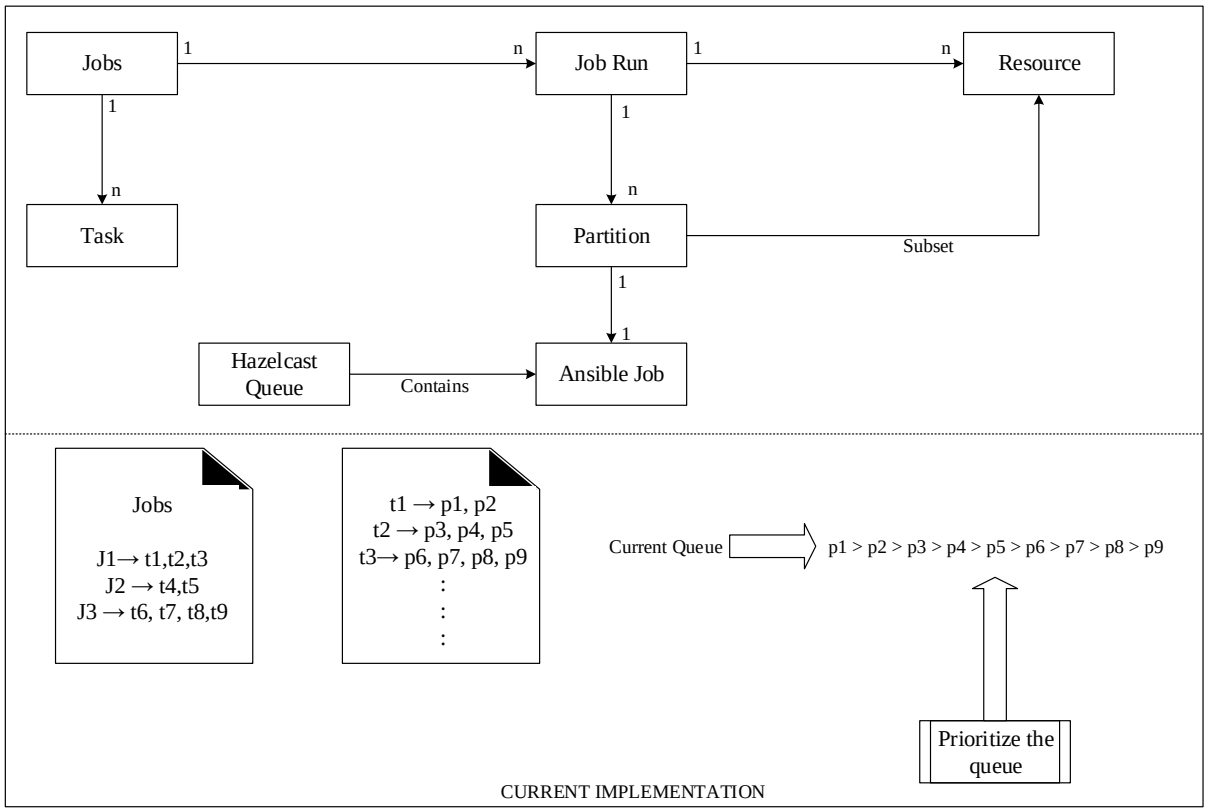
Jobs
Job Run
Resource
Task
Partition
Subset
Hazelcast Queue
Contains
Ansible Job
1
n
Jobs
J1→ t1,t2,t3
J2 → t4,t5
J3 → t6, t7, t8,t9
t1 → p1, p2
t2 → p3, p4, p5
t3→ p6, p7, p8, p9
Current Queue
p1 > p2 > p3 > p4 > p5 > p6 > p7 > p8 > p9
Prioritize the queue
CURRENT IMPLEMENTATION

Fig. 4

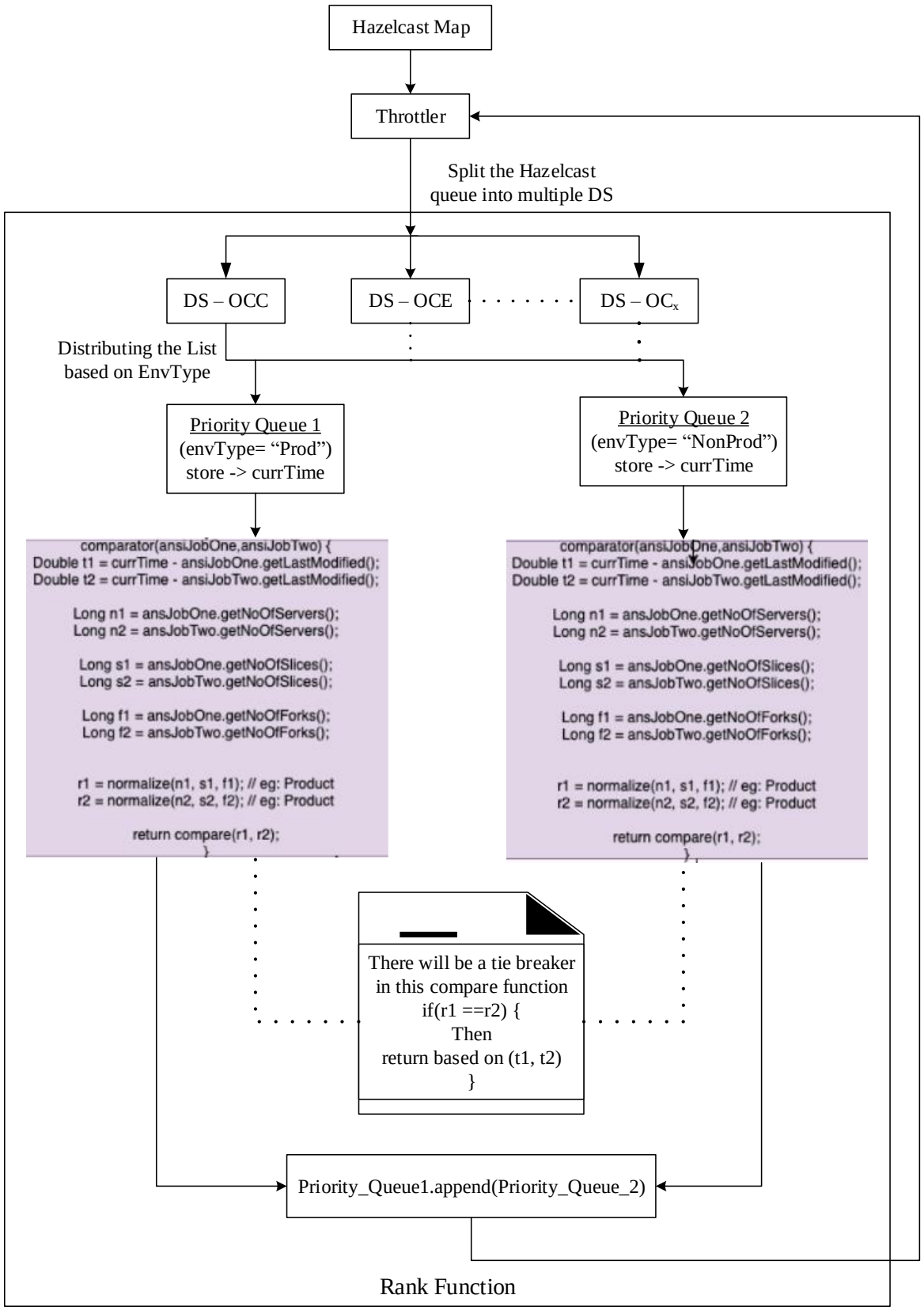
Hazelcast Map
Throttler
Split the Hazelcast queue into multiple DS
DS – OCC
DS – OCE
DS – OCx
Distributing the List based on EnvType
Priority Queue 1 (envType= "Prod") store -> currTime
Priority Queue 2 (envType= "NonProd") store -> currTime
comparator(ansiJobOne,ansiJobTwo) {
Double t1 = currTime - ansiJobOne.getLastModified();
Double t2 = currTime - ansiJobTwo.getLastModified();
Long n1 = ansJobOne.getNoOfServers();
Long n2 = ansJobTwo.getNoOfServers();
Long s1 = ansJobOne.getNoOfSlices();
Long s2 = ansJobTwo.getNoOfSlices();
Long f1 = ansJobOne.getNoOfForks();
Long f2 = ansJobTwo.getNoOfForks();
r1 = normalize(n1, s1, f1); // eg: Product
r2 = normalize(n2, s2, f2); // eg: Product
return compare(r1, r2);
}
There will be a tie breaker in this compare function
if(r1 ==r2) {
Then
return based on (t1, t2)
}
Priority_Queue1.append(Priority_Queue_2)
Rank Function

Fig. 5

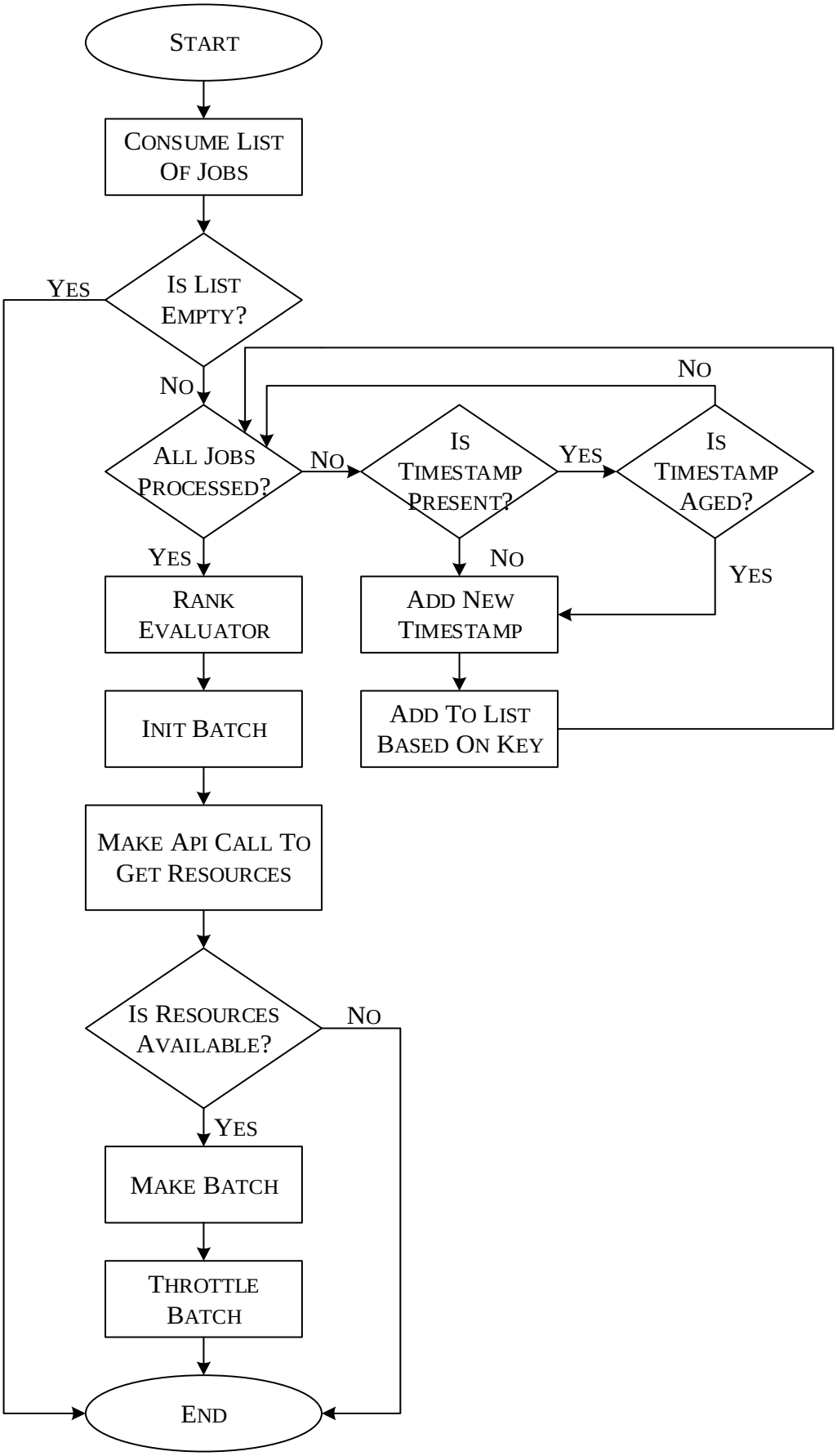
START
CONSUME LIST OF JOBS
IS LIST EMPTY?
YES
NO
ALL JOBS PROCESSED?
NO
IS TIMESTAMP PRESENT?
YES
IS TIMESTAMP AGED?
NO
YES
YES
NO
RANK EVALUATOR
ADD NEW TIMESTAMP
INIT BATCH
ADD TO LIST BASED ON KEY
MAKE API CALL TO GET RESOURCES
IS RESOURCES AVAILABLE?
NO
YES
MAKE BATCH
THROTTLE BATCH
END

Fig. 6

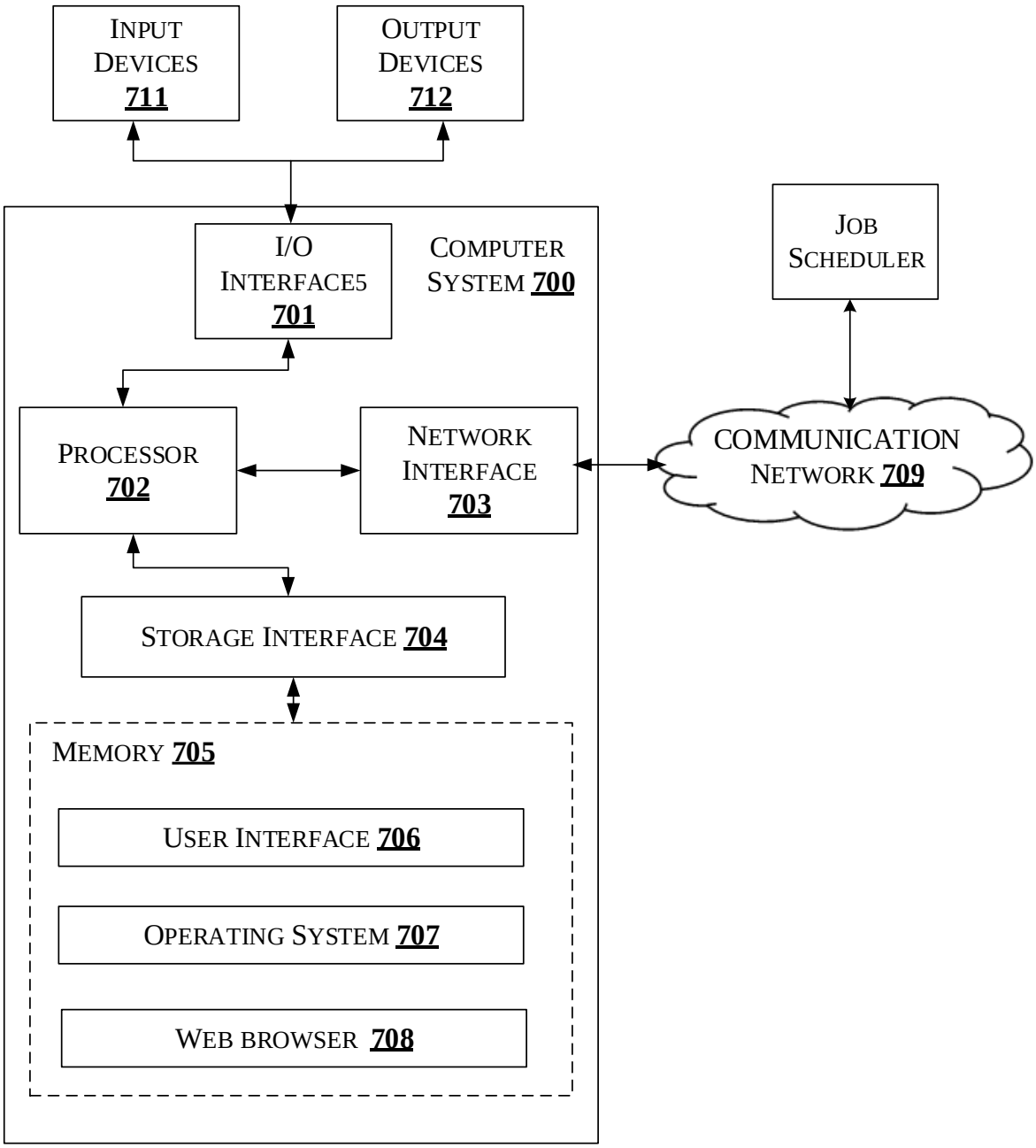
INPUT DEVICES 711
OUTPUT DEVICES 712
I/O INTERFACE5 701
COMPUTER SYSTEM 700
JOB SCHEDULER
PROCESSOR 702
NETWORK INTERFACE 703
COMMUNICATION NETWORK 709
STORAGE INTERFACE 704
MEMORY 705
USER INTERFACE 706
OPERATING SYSTEM 707
WEB BROWSER 708

Fig. 7