International Journal of Innovation and Applied Studies
ISSN 2028-9324 Vol. 27 No. 3 Oct. 2019, pp. 762-772

http://www.ijias.issr-journals.org/

Use of messaging patterns in applications that receive massive transactions seen from the teaching process

Jimmy Sornoza Moreira, Christopher Crespo León, Gary Reyes Zambrano, and Roberto José Zurita-Del Pozo

Facultad de Ciencias Matemáticas y Físicas, Universidad de Guayaquil, Guayaquil, Ecuador

ABSTRACT: Transaction processing systems (SPT) are the union of software, network equipment, servers, among others, that are used to work with large volumes of data. The inadecuated design of an SPT in one of its processes or the malfunction in one of its components or elements, it can directly impact the performance of an application and the company operation environment or product depending on the system objective, it could cause waste of time in process responses, it could have an impact on the total failure of the service. Failure to provide this service properly could cause economic losses in a company or organization.

KEYWORDS: Hardware, software, generators, volumes, transactions, consumer, patterns, pattern, messaging, processing.

1 INTRODUCTION

The workflows in operations that a company manages can be long or short in its execution, it depends on the company activity and the correlation that is activated by means of messages between one element and another in a process, or when resuming a stage in which an entry data is expected. (Microsoft, 2017)

It is not the same workflow of recharges of massive balances in a telecommunications company than a workflow of balances recharge in electronic games, the workflow varies according to: The activity of the company, the objective of the product or service and the number of transactions that will be handled in it.

Software Developers usually evolve with the application of good practices, the acquisition of new knowledge based on research or many times with implementation "in house" to solve a specific problem; decades ago they have been based on the reuse of codes, orienting their programming to: Objects, creation of classes, functions, procedures, etc.; but there is something in particular, the different developers worldwide go through the same problems, even seen from different perspectives are often similar, that is, they go through recurring problems and that's how the "Design Patterns" are born. (It-business, 2017)

A design pattern is the idea of how to solve a problem within a project; that is, it is a design solution to a recurring problem in a particular context, so every pattern has a name. (Leiva)

A pattern must have a good name, an intention -goals and objectives to be achieved- to solve a motivation -problem within a project-; Whenever this is applicable, three sections turn out to be the problem to be addressed.

Corresponding Author: Jimmy Sornoza Moreira

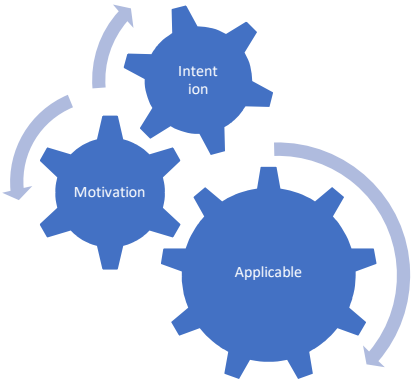


Fig. 1. Sections that make up a pattern

The proposed solution is given by a structure that includes participants, collaborations, an implementation that considers possible variations, code that demonstrates how it is used, related patterns, consequences.

A development team can make use of "n" design patterns to deal with different specific problems found in any part of a system; that is, through the use of patterns we can save time to design or improve a project done "in house", this fact allows to use that time in other tasks or actions that contribute to development, such as increasing standards of quality in the task performed.

This use of patterns should be applied according to the size of the problem, making an analogy with real life, it is not the same to walk to the store that is 5 homes away than go by motorcycle to the same store, the amount of time that it takes to turn on this vehicle can be the same that consumes going on foot and without consuming resources such as fuel, tires, etc.

MESSAGING PATTERNS

The fact of applying messaging patterns requires an infrastructure that supports the flexibility of its components, which allows to increase the scalability of a process that is part of the software and that makes possible the connection of these components.

Asynchronous messaging is effective at the time of application, likewise, this type of messaging demands that messages be interpreted, classified and that their priorities be verified.

Message patterns resemble service-oriented patterns, there is a difference in the way we think about them. Service-oriented models can be consumed by many customers. A message-oriented model focuses on the data that flows through the system and the set of actions applied to these data under a producer-to-consumer stance. (Microsoft, 2017)

The following table details different messaging patterns that can be applied to a specific problem.

Table 1. Messaging Patterns

Pattern	Summary
Competing Consumers	The pattern can be applied, allowing that n consumers can simultaneously process the messages received within the same messaging channel.
Pipes and Filters	A complete processing is broken down or distributed into n independent processes or elements that can be reused in another process.
Priority Queue	The requests sent to the different services are ordered by priority, the objective is that the higher priority has a greater preference for processing than the one with low priority.
Queue-Based Load Leveling	Implements a queue that receives transactions or tasks, which can be dispatched to a service allowing thus balancing heavy and intermittent loads.
Scheduler Agent Supervisor	Coordinates a set of "n" actions in a set of services that are distributed.

COMPETING CONSUMERS PATTERN

Also known as competition consumers, this pattern allows several transactions or messages to be processed simultaneously, without affecting the performance of a system or process, gaining scalability according to the dynamism of the business. (Redhat, 2018)

An application can trigger several transactions or messages, these can be treated synchronously and asynchronously, the described pattern works effectively asynchronously and thus prevents that transactions be blocked while they are processed. (Microsoft, 2017)

The number of requests or transactions triggered by an application or process, may vary depending on the dynamics of the business or schedule in which they are executed; likewise, these requests can come from different triggers and not necessarily from the same system.

As an example, you can describe a problem regarding the different types of users that generate transactions in a telecommunications company, there a customer can report a theft of their cell phone by calling the call center, this will generate a transaction that will block this equipment and simcard on different platforms.

That same transaction could be triggered by another client who reports the theft of their equipment from a customer service center, this could also be through the different channels provided by the company, that is, the number of messages generated by a transaction type is inversely proportional to the number of reports for theft generated and according to the number of subscribers that the company owns.

The proposed solution allows to distribute or balance the load according to the number of messages produced by an instance of the application, these messages go to a queue where they are provisioned or processed.

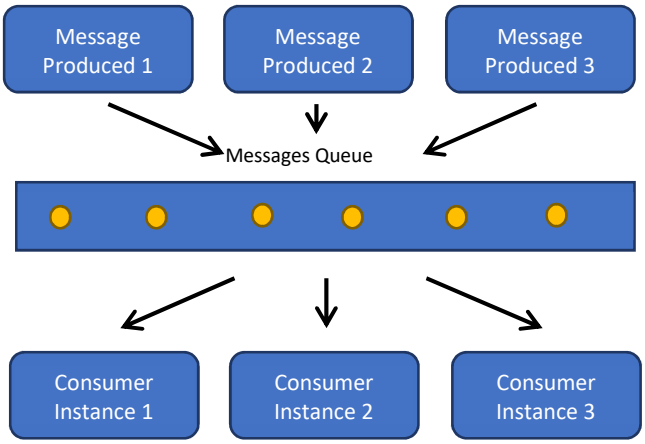


Fig. 2. Competing Consumers Pattern

In this way the reliability is improved, since the message produced by a service or application is not sent directly to a consumer, if this were the case, it would be necessary to monitor the consumer to confirm the operation and to prevent messages from being lost.

With the use of this pattern, if there is a problem in a consumer instance, the requests or messages will continue queuing without affecting the application or process that generates the message.

PIPES & FILTERS PATTERN

In this pattern, a complete process is broken down or distributed into n independent processes or elements that can be reused in another process.

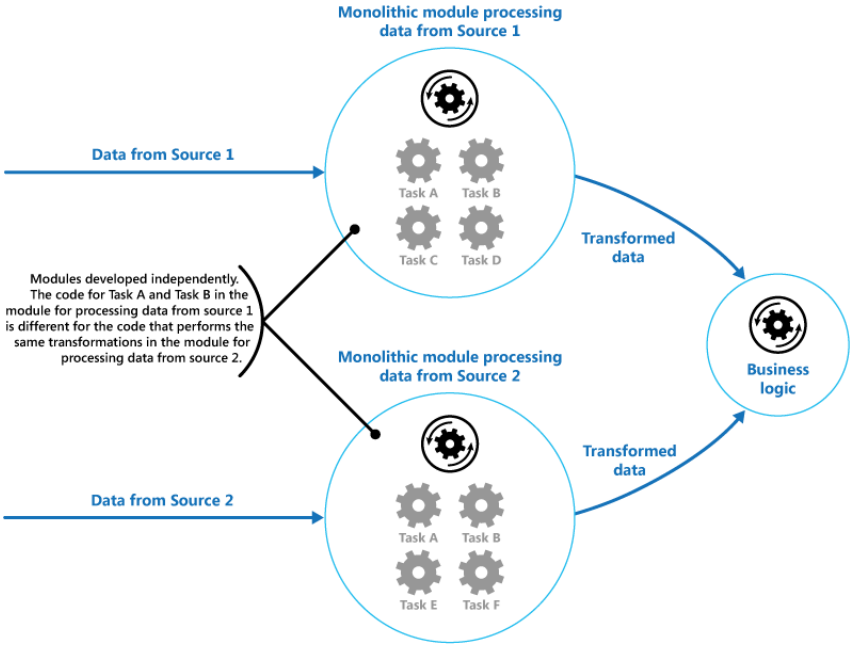


Fig. 3. Pipes & filters Pattern

Figure 3 shows by means of a monolithic approach, how problems in data processing are managed. In this case the data is received and processed from two sources, an independent module processes the data of each source and performs a transformation of them, before the result reaches the business logic of application.

Monolithic modules perform functionally similar tasks but the design of modules is done separately, the code is coupled in a very small way in a module and is developed considering its reuse or scalability slightly.

Tasks performed in each module or requirements to implement each task, can have a change from the update of business requirements.

Probably there are intensive process tasks that benefit if they are executed with high power hardware, while others do not need such high resources but in the future they may require additional processing or order changes in the task. (Iteratrlearning, 2016) (Microsoft, 2017)

PRIORITY QUEUE PATTERN

Requests sent to different services are ordered by priority, the objective is higher priority has a higher processing preference than lower priority. (Oscarblancarteblog, 2014)

This pattern can be explained by making an analogy with a telecommunications company in which a customer owns a television service that must pay monthly and that customer falls into default for non-payment, then the company executes a process to suspend the service, to once the customer makes the payment online and two transactions fall in line to be processed, should the priority between both transactions be the same?

The priorities are defined by the business according to their specific needs, the pattern in question orders these priorities and allows them to be processed accordingly.

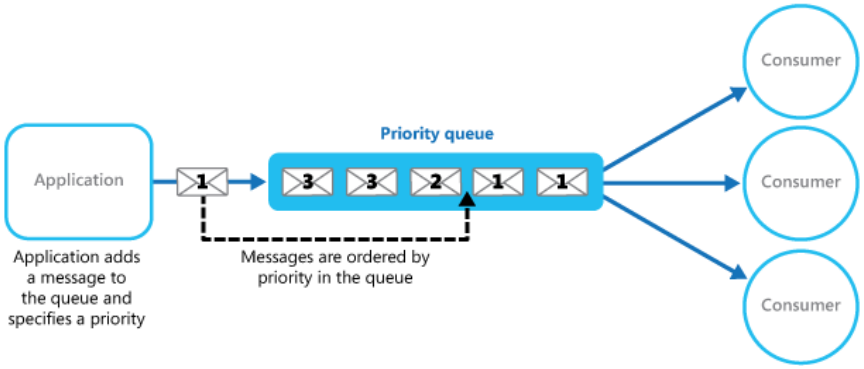


Fig. 4. Priority Queue

In case that transactions do not handle priorities, alternatives such as creating multiple queues can be used so that each one processes a type of transaction; taking as an example a company that makes massive promotions and that action should be taken on each promotion according to the action that a client made, that is, the messages would be queued according to the action that client made.

Use of the mechanism described above provides the following advantages:

- Transactions are prioritized according to needs of the business.
- Operating costs of a process can be reduced with use of a queue, if different queues are used the number of consumers can be reduced and more control in processing of each one can be had.
- The handling of different queues can help improve processing time, maximizing performance and scalability of application according to the dynamism that a company presents in relation to its products. (Microsoft, 2017)

QUEUE-BASED LOAD LEVELING PATTERN

This pattern implements a queue that receives transactions or tasks, which can be dispatched to a service allowing balancing heavy and intermittent loads. (Microsoft, 2017)

Queue to be created must work asynchronously, that is, messages or transactions that pass must be previously deposited to be eliminated after transaction be processed.

The queue may receive several messages or transactions from different media, which produce "n" transactions, which means that many producers and consumers will be able to work with it, but each message in the queue is attended only by one process. (Amazon, 2018).

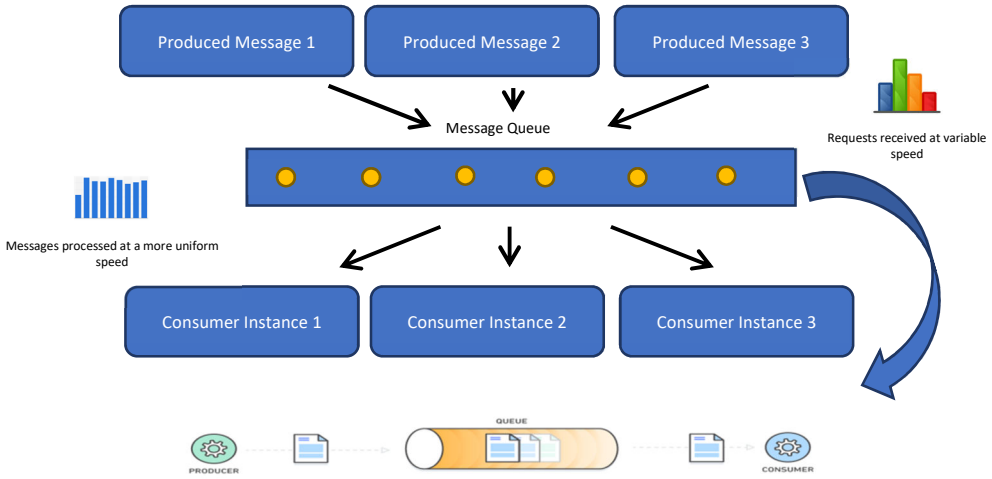


Fig. 5. Message queue with load leveling

This model is useful for any type of service that handles large volumes of information, if application or process has a minimum response latency, it would not be possible to use this pattern

This model provides the following advantages:

- It increases availability of system, since processing time of different services does not directly affect application; Likewise, if there is a problem with consumer, messages triggered from an application or process can continue to be stored in queue until affected service be restored.
- According to demand, structure of the model can be easily adapted to process this transactions, therefore its scalability is greater according to the growth of the company.
- You can manage thresholds to define the level of load for each service instance.

According to the example previously stated on the generation of transactions for cellular theft in a telecommunications company, these messages would be triggered by different channels and then wait for the storage service, it could happen that the system is used up and end Wait time for a message therefore the message is lost.

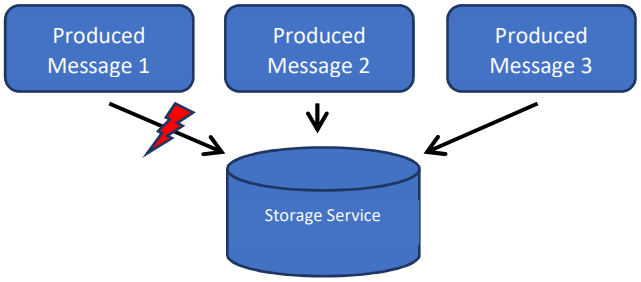


Fig. 6. Messages lost due to waiting time

This pattern is used to solve this problem, since it allows to queue transactions, then process them and dispatch in a balanced way to storage service.

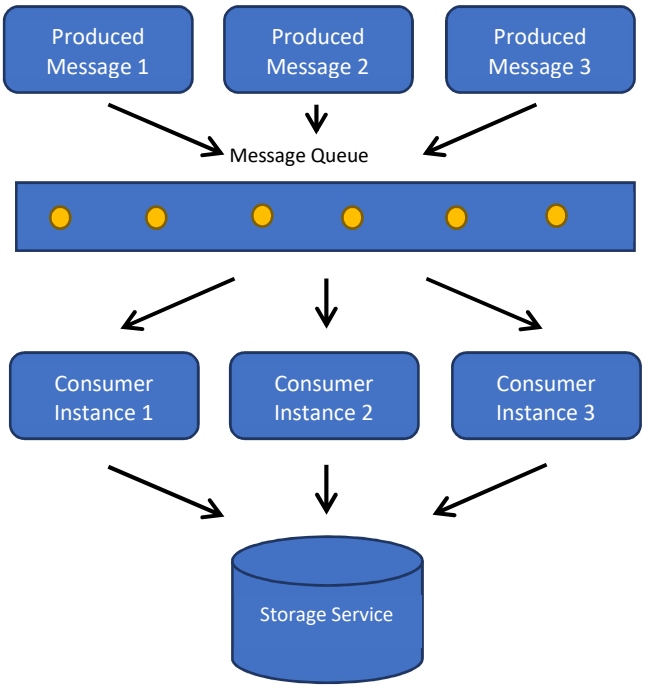


Fig. 7. Asynchronous queue with messages from different producers

SCHEDULER AGENT SUPERVISOR PATTERN

It coordinates different tasks in a distributed manner as a single operation, if an error occurs in one of the tasks, it manages the error in a transparent manner as if it were a single process, that is, it manages the error in an integral manner, it tries to restore the operation autonomously, if not achieved, repeat the whole operation. (Vasters, 2010)

If its application generates a workflow involving remote connections, service consumption, etc., there may be problems caused in several ways (network failure, unstable remote service, or inactive service) if the application detects that it is a permanent error, which is constant or can not be easily recovered, must be able to restore the system in a consistent state and ensure the integrity of the entire operation.

The Scheduler Agent Supervisor pattern defines actors that intervene in the solution of the problem:

- The Scheduler component organizes a task in steps and organizes its operation, the component is responsible for ensuring that each of these steps are executed in an orderly manner ensuring that it does so correctly. As each step of the task is executed, the scheduler component marks each execution with a status, that state generates three values ("step not yet started", "step in execution" or "step completed") likewise, there is an execution time for all the task that will be called "time of validity". If a step requires access to a remote resource or service, Scheduler component invokes the appropriate Agent component and transmits work details it must perform.
- Agent component contains a logic that encapsulates a call to a remote service or access to a remote resource to which a specific step of a task refers.
- Supervisor, is in charge of verifying each state of the component scheduler, if one of them has delays in the execution or in the response, calls the Agent component so that it directs the delayed step, this may involve the modification of a state of the affected step (Microsoft, 2017)

Logical components (Scheduler, Agent, supervisor) can be implemented physically but depend on the technology that is being used.

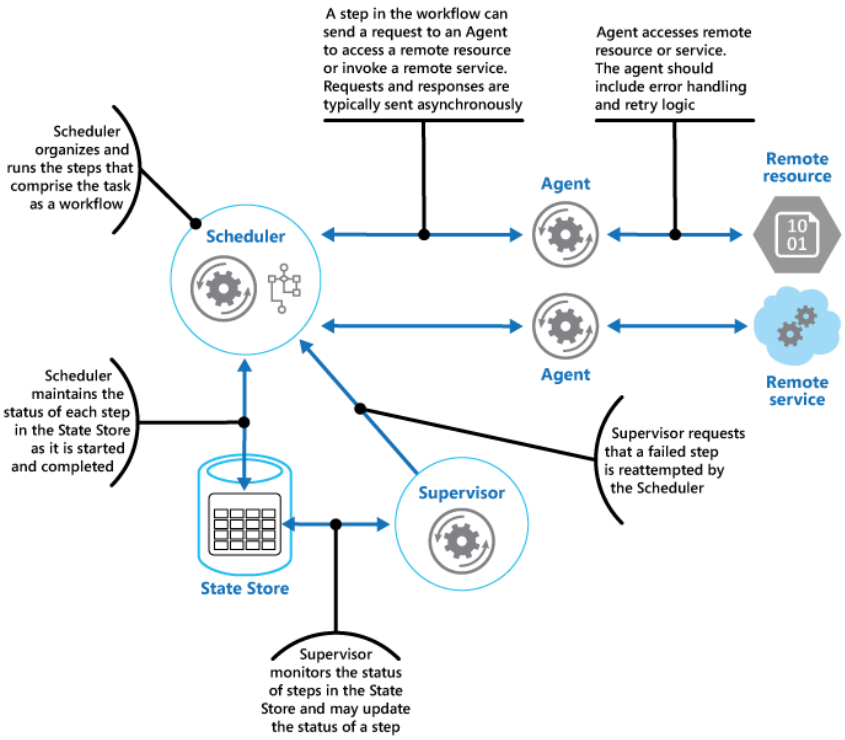


Fig. 8. Logic components of Scheduler Agent Supervisor Pattern

This pattern is used when a process runs distributedly, since it handles operational or communication problems, this pattern may not be suitable for tasks that do not invoke or access remote services.

2 DISCUSSION

A solution was developed with JAVA programming language and Oracle as a data repository. The messaging patterns described below were applied: Competing Consumers, Pipes & filters, Priority Queue, Queue-Based Load Leveling.

The solution was applied to a service suspension process due to non-payment, transactions or messages were triggered by Collection Management module (MGC) as mentioned above.

Messages are triggered by the MGC make blockages inside the application to avoid that some other type of transaction be made and at the same time they are shot to other platforms that block the service that is provided to the client.

Next, an analysis will be made regarding time and development of an application to visualize the differences when applying the aforementioned patterns.

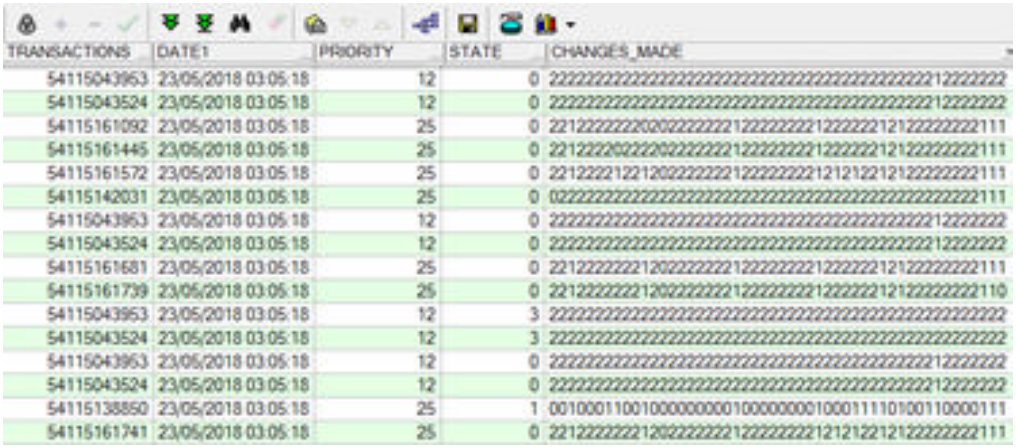
Before applying the patterns it was established to shoot an Universe of 106483 messages / transactions that would block services and states in different platforms.

Table 2. Average time in dispatching transactions without patterns.

DISPATCHERS/DEMONS	AMOUNT OF TRANSACTIONS	SECONDS	MINUTES	HOURS
1	106483	85186,40	1419,77	23,66

According to Table 2, the processing of a universe of 106,483 service blocking transactions on different platforms took 24 hours.

To improve processing time in the same type of transaction, messaging patterns were implemented, previously creating a queue in which different messages or transactions that block the different platforms will be deposited; The structure of queue contains a transaction code or service number, date of the transaction, priority, status and changes made.



TRANSACTIONS	DATE1	PRIORITY	STATE	CHANGES_MADE
54115043953	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115043524	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115161092	23/05/2018 03:05:18	25	0	22122222202022222221222222212222221212222222111
54115161445	23/05/2018 03:05:18	25	0	22122220222022222221222222212222221212222222111
54115161572	23/05/2018 03:05:18	25	0	22122221221202222222122222221212122121222222111
54115142031	23/05/2018 03:05:18	25	0	02222222222222222222222222222222222222222222111
54115043953	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115043524	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115161681	23/05/2018 03:05:18	25	0	22122222221202222222122222221222222121222222222111
54115161739	23/05/2018 03:05:18	25	0	22122222221202222222122222221222222121222222222110
54115043953	23/05/2018 03:05:18	12	3	22222222222222222222222222222222222222222222222222
54115043524	23/05/2018 03:05:18	12	3	22222222222222222222222222222222222222222222222222
54115043953	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115043524	23/05/2018 03:05:18	12	0	22222222222222222222222222222222222222222212222222
54115138850	23/05/2018 03:05:18	25	1	00100011001000000000100000000100011110100110000111
54115161741	23/05/2018 03:05:18	25	0	22122222221202222222122222221212122121222222222111

Fig. 9. Queue Structure

Transactions that arrive at message queue are prioritized according to the policies established by the business, priorities are dealt with according to transaction code, the higher the priority the less the time to be dispatched.

$$tx\_taken=max(priority)$$

Once queue was implemented and according to the priority assigned to each transaction, it was verified that the same number of transactions that arrived at said queue with the use of the established pattern be the same as that arrived without the application of messaging pattern, it was verified that a universe of 106,483 transactions has arrived in total.

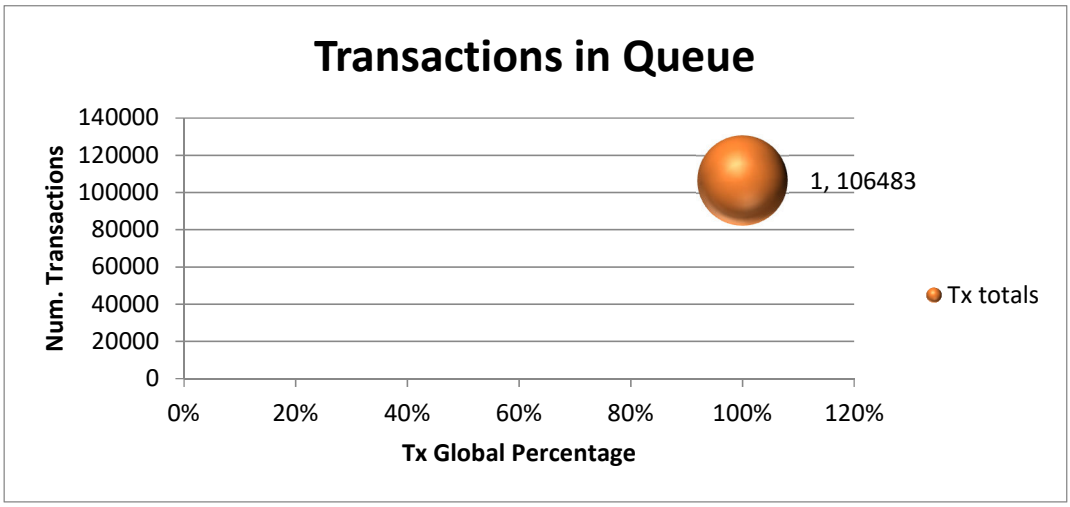
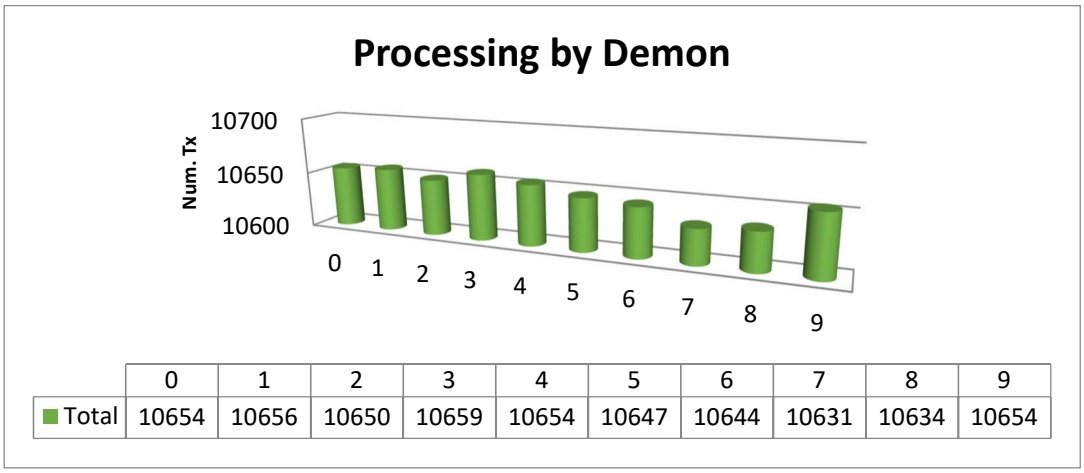


Fig. 10. Universe of transactions for service suspension

Once the transactions were deposited in the queue, the attention process managed transactions with an average of 10,500 messages per daemon, that is, 10 threads were created according to the threshold that was set at most to meet requirements, these transactions were deposited by different producers.

For this practical case of service blocking due to lack of payment, each thread verified the chain and which flag should be lit on different platforms.



	0	1	2	3	4	5	6	7	8	9
Total	10654	10656	10650	10659	10654	10647	10644	10631	10634	10654

Fig. 11. Processing by thread

Different producers triggered different messages or transactions to the queue in the course of 00:00 am to 06:00 am, having a duration of 6 hours, the messages were deposited to the queue asynchronously.

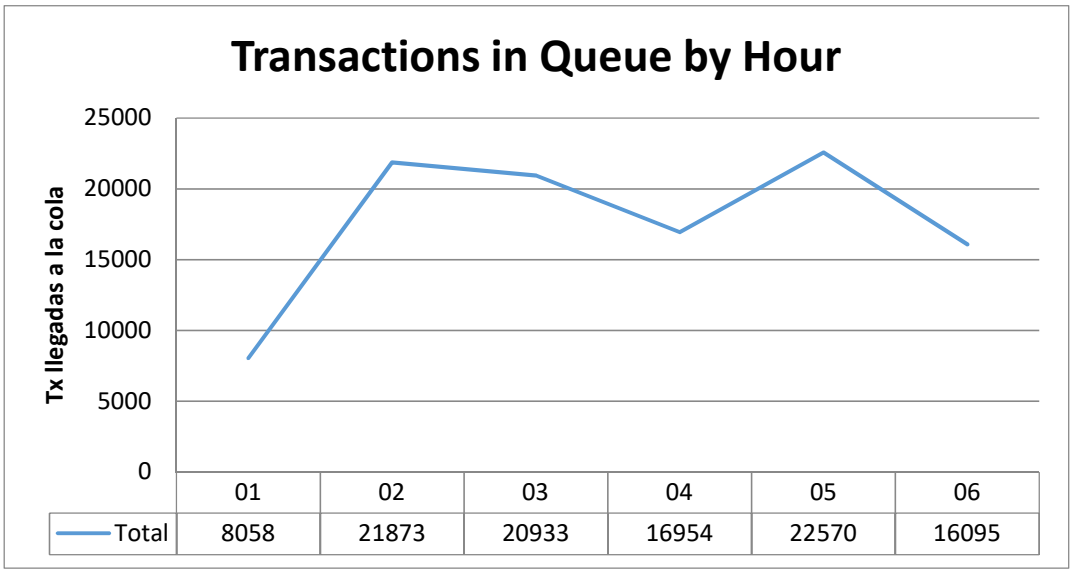


Fig. 12. Arrival of message / transactions per hour to queue

Processing of the queue, had an average duration of 2.37 hours as shown in table 3 according to the number of transactions that were deposited in queue for different consumers can take the transaction and proceed to block the service on different platforms.

Table 3. Average time in dispatching transactions using queue

DISPATCHERS / DEMONS	AMOUNT OF TRANSACTIONS	SECONDS	MINUTES	HOURS
0	10654	8523,20	142,05	2,37
1	10656	8524,80	142,08	2,37
2	10650	8520,00	142,00	2,37
3	10659	8527,20	142,12	2,37
4	10654	8523,20	142,05	2,37
5	10647	8517,60	141,96	2,37
6	10644	8515,20	141,92	2,37
7	10631	8504,80	141,75	2,36
8	10634	8507,20	141,79	2,36
9	10654	8523,20	142,05	2,37
AVERAGE TIME		**8518,64**	**141,98**	**2,37**

3 CONCLUSION

In the case of Study presented previously with reference to the blocking of services where the producer (collection module), triggers transactions to different platforms to cut the service due to non-payment, it was possible to identify that 100% of transactions took almost 23.66 hours to complete the process before implementation of the messaging pattern.

By implementing the messaging patterns, it was possible to identify that the processing of transactions mentioned above was done in 2.37 hours; that is, the processing times fell by 89.98% with reference to the blocking of services.

This means that the availability of processing services is increased by 89.98% and due to this there are greater opportunities to monitor the different services involved in the pattern.

Because it can measure the time which takes a producer to shot, processing time and consumption time, it can measure the efficiency of each member in the pattern and take some action in case of having some inconvenience in one of the actors.

In view of the fact that request times and response times can be measured, a more direct approach to a network element that can fail to handle large amounts of transactions can be made.

The case study discussed above focused on a type of transaction that has a great impact on a company -even economic-, as well as on the management of the different messages that are processed in it; this case could also be extended, for example there could be transactions that are directed to the same network elements or platforms with which we worked previously and a study could be carried out to analyze if those transactions can reach the queue according to their structure and determine whether or not it is necessary to design a new development that modifies the service provided by the different elements of the network.

REFERENCES

[1] Amazon. (2018). *Amazon*. Recuperado el 20 de 05 de 2018, de Amazon: https://aws.amazon.com/es/message-queue/

[2] It-empresarial. (05 de 12 de 2017). *It-empresarial*. Recuperado el 30 de 05 de 2018, de It-empresarial: http://www.it-empresarial.com/index.php/noticias/119-que-son-los-patrones-de-diseno-y-para-que-son-utiles

[3] Iteratrlearning. (26 de 12 de 2016). *Iteratrlearning*. Recuperado el 30 de 04 de 2018, de Iteratrlearning: http://iteratrlearning.com/java/2016/12/26/pipes-and-filters-actors-akka-java.html

[4] Leiva, A. (s.f.). *devexperto*. Recuperado el 30 de 04 de 2018, de devexperto: https://devexperto.com/patrones-de-diseno-software/

[5] Microsoft. (23 de 06 de 2017). *Microsoft - Priority Queue*. Recuperado el 05 de 06 de 2018, de Microsoft - Priority Queue: https://docs.microsoft.com/es-es/azure/architecture/patterns/priority-queue

[6] Microsoft. (23 de 06 de 2017). *Microsoft*. Recuperado el 25 de 05 de 2018, de Microsoft: https://docs.microsoft.com/es-es/azure/architecture/patterns/category/messaging

[7] Microsoft. (23 de 06 de 2017). *Microsoft*. Recuperado el 05 de 06 de 2018, de Microsoft: https://docs.microsoft.com/es-es/azure/architecture/patterns/scheduler-agent-supervisor

[8] Microsoft. (23 de 06 de 2017). *Microsoft*. Recuperado el 30 de 04 de 2018, de Microsoft: https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters

[9] Microsoft. (23 de 06 de 2017). *Msdn.microsoft.com*. Recuperado el 31 de 05 de 2018, de Msdn.microsoft.com: https://docs.microsoft.com/es-es/azure/architecture/patterns/competing-consumers

[10] Microsoft. (23 de 06 de 2017). *Patrón Queue-Based Load Leveling*. Recuperado el 25 de 05 de 2018, de Patrón Queue-Based Load Leveling: https://docs.microsoft.com/es-es/azure/architecture/patterns/priority-queue

[11] Microsoft. (23 de 06 de 2017). *Patrones de mensajería*. Recuperado el 22 de 05 de 2018, de https://docs.microsoft.com/es-es/azure/architecture/patterns/category/messaging

[12] Oscarblancarteblog. (01 de 08 de 2014). *Oscarblancarteblog*. Recuperado el 01 de 06 de 2018, de Oscarblancarteblog: https://www.oscarblancarteblog.com/2014/08/01/estructura-de-datos-queue-cola/

[13] Redhat. (05 de 06 de 2018). *Redhat*. Obtenido de Redhat: https://access.redhat.com/documentation/en-US/Fuse_ESB_Enterprise/7.1/html/Implementing_Enterprise_Integration_Patterns/files/MsgEnd-Competing.html

[14] Vasters, C. (27 de 09 de 2010). *AIRPLANES. CLOUD COMPUTING. AND ALIEN ABDUCTIONS*. Recuperado el 05 de 06 de 2018, de AIRPLANES. CLOUD COMPUTING. AND ALIEN ABDUCTIONS: http://vasters.com/archive/Cloud-Architecture-The-Scheduler-Agent-Supervisor-Pattern.html